Акционерное общество
«Экспериментальный завод научного приборостроения
со Специальным конструкторским бюро
Российской академии наук»

УТВЕРЖДЕН
КУНИ.505200.023-01.01 33-ЛУ

SCADA-СИСТЕМА "СОНАТА"

Руководство программиста
КУНИ.505200.023-01.01 33

Листов 344

Инв. № подл.	Подпись и дата	Взам. инв. №	Инв. № дубл.	Подпись и дата

2025

Литера О1

АННОТАЦИЯ

Данный документ является руководством по языкам программирования, которые применяются в SCADA-системе "СОНАТА" для решения различных задач.

СОДЕРЖАНИЕ

1. РАЗРАБОТКА ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА "LUA"

Lua является дополняющим языком программирования, разработанным для поддержки обычного процедурного программирования со средствами описания данных. Lua также оказывает хорошую поддержку объектно-ориентированному программированию, функциональному программированию и программированию, управляемому данными. Lua предназначен для использования в качестве мощного, легковесного, встраиваемого скриптового языка для любых программ.

Lua является свободным программным обеспечением и предоставляется как обычно без каких-либо гарантий, как указано в его лицензии. Реализация,описанная в данном руководстве, доступна на официальном веб-сайте Lua, www.lua.org [http://www.lua.org/].

Как и у любого другого справочника, стиль изложения данного документа местами суховат. Для обсуждения решений разработки Lua смотрите технические документы, доступные на веб-сайте Lua [http://www.lua.org/docs.html]. Для более детального знакомства с программированием на Lua смотрите книгу Роберто Иерусалимского (Roberto Ierusalimschy) Programming in Lua.

1.1. Описание языка

В этой главе описывается лексика, синтаксис и семантика языка Lua. Другими словами, здесь представлены элементы языка, способы их комбинирования и значение языковых конструкций.

Конструкции языка будут вводиться с использованием расширенной BNF, где запись {a} означает 0 или более элементов a, а запись [a] означает его необязательное вхождение. Нетерминальные символы отображаются обычным шрифтом, ключевые слова выделяются жирным шрифтом **kword**. Полное описание синтаксиса языка Lua дано в конце Руководства.

1.1.1. Лексические соглашения

Именами (*идентификаторами*) в Lua могут быть любые строки из букв, цифр и символа подчеркивания, не начинающиеся с цифры. Это правило типично для большинства языков программирования. (Понятие буквы зависит от текущей локали: любой символ из алфавита текущей локали может быть использован в составе идентификатора). Идентификаторы используются для именования переменных и таблиц значений (table fields).

Следующие *ключевые слова* зарезервированы и не могут быть использованы в именах:

```
and     break    do     else       elseif
end     false    for    function   if
in      local    nill   not        or
repeat  return   then   true       until   while
```

Lua является языком, чувствительным к регистру символов: **and** – ключевое слово, тогда как And и AND – два разных допустимых идентификатора. По соглашению, имена, начинающиеся с символа подчеркивания и записанные в верхнем регистре (например _VERSION), зарезервированы для использования в качестве внутренних глобальных переменных, используемых Lua.

В следующих строках показаны другие допустимые символы:

```
+       *       /       %       ^       <       #
```

```
==      ~=      <=       >=      <        >       =
(       )       {       }       [       ]>
;       :       ,       .       ..      ...
```

Литеральные строки должны быть заключены в одинарные или двойные кавычки и могут содержать следующие C-подобные escape-последовательности: '\a' («звонок»), '\b' («забой»), '\f' («перевод страницы»), '\n' («перевод на новую строку»), '\r' («возврат каретки»), '\t' («горизонтальная табуляция»), '\v' («вертикальная табуляция»), '\"' («двойная кавычка»), and '\'' (апостроф [«одинарная кавычка»]). Кроме того, обратный слеш ставится перед концом строки в редакторе, когда для удобства набора длинные непрерывные строки записываются в несколько строк. Символ в строке также может быть представлен своим кодом с помощью escape-последовательности \ddd, где ddd- последовательность из не более чем трех цифр. (Заметим, что если после символа, записанного с помощью своего кода, должна идти цифра, то код символа в escape-последовательности должен содержать ровно три цифры). Строки в Lua могут содержать любые 8-битные значения, включая ноль, который записывается как '\0'.

1.1.2. Значения и типы

Lua представляет собой язык с *динамическим определением типов данных*. Переменная языка может содержать значения любого типа. Возможности определения пользовательских типов данных отсутствуют. Все значения в Lua могут храниться в переменных, использоваться в качестве аргументов при вызове функций и возвращаться в виде результата их выполнения.

В Lua восемь основных типов: nil (неопределенный), boolean (логический), number (числовой), string (строковый), function (функция), userdata (пользовательские данные), thread (поток) и table (таблица). Nil -это тип значения **nil** [пустое значение], главное свойство которого – отличаться от всех остальных значений и обозначать отсутствие пригодного значения. К типу Boolean относятся значения **false** (ложь) и **true** (истина). Значения **nil** и **false** считаются ложными, любое другое значение считается истинным. К типу Number относятся вещественные числа (двойной точности с плавающей запятой). (Легко можно сделать сборку интерпретатора Lua с другим внутренним представлением чисел, достаточно изменить определение в файле luaconf.h.). Тип String обозначает массивы символов. Строки Lua могут содержать любые 8 битные символы, включая ноль ('\0') (смотреть §2.1 [http://www.lua.ru/doc/2.1.html]).

В Lua можно использовать функции, написанные на Lua и на C (смотреть §2.5.8 [http://www.lua.ru/doc/2.5.8.html]).

Тип userdata (пользовательские данные) позволяет хранить любые данные из C в переменных Lua. Значение этого типа является ссылкой на блок физической памяти и не имеет предопределенных операций в Lua, за исключением присваивания и проверки на равенство. Однако, используя метатаблицы, програмист может определить операции над значениями этого типа (смотрите §2.8 [http://www.lua.ru/doc/2.8.html]). Значения типа userdata не могут быть созданы или изменены непосредственно в Lua, это возможно только с помощью C API. Такой подход гарантирует целостность данных, принадлежащих ведущей программе.

Тип thread (поток) обозначает независимый поток исполнения и используется при реализации механизма сопрограмм (смотрите §2.11 [http://www.lua.ru/doc/2.11.html]). Нельзя отождествлять потоки Lua с потоками операционной системы. Lua поддерживает подпрограммы даже в тех системах, где потоки на уровне операционной системы не поддерживаются.

Тип table (таблица) определяет ассоциативные массивы. Такие массивы могут индексироваться не только числами, но и любыми значениями (за исключением **nil**). Таблица может содержать значения сразу нескольких типов (кроме **nil**). Таблицы представляют собой единственный

механизм структурирования данных в Lua; они могут использоваться как простые массивы, таблицы сиволов, множества, поля записей, деревья и так далее. Для представления словарей Lua использует имя поля в качестве индекса таблицы. Представление в виде a.name считается тождественным представлению a["name"]. В Lua есть несколько способов создания таблиц (смотреть §2.5.7 [http://www.lua.ru/doc/2.5.7.html]).

Индексы и значения полей таблицы могут быть любого типа (кроме **nil**). В частности, так как функции являются значениями встроенного типа, поля таблицы могут содержать и функции. Таким образом, таблицы могут хранить методы methods (смотреть §2.5.9 [http://www.lua.ru/doc/2.5.9.html]).

Переменные типа *table*, *function*, *thread* и *userdata* не содержат самих данных, в них хранятся только ссылки на соответствующий объект. Присваивание, передача параметров и возврат результата из функции оперируют только ссылками на значения, эти операции никогда не ведут к созданию копий.

Библиотечная функция type возвращает строку, описывающую тип данного значения.

1.1.2.1. Приведение типов

Lua обеспечивает автоматическое преобразование между строковыми и числовыми значениями в процессе выполнения. Любая арифметическая операция, применяемая к строке, пытается преобразовать эту строку в соответствующее число по обычным правилам приведения. Когда же число используется там, где ожидается строка, это число преобразуется в строку в произвольном подходящем формате. Так что для получения какого-то конкретного представления числа в строке необходимо использовать функцию format из библиотеки работы со строками (смотрите функцию string.format).

1.1.3. Переменные

Переменные используются для хранения значений в процессе выполнения программы. В Lua есть три вида переменных: глобальные, локальные и поля таблиц.

Отдельный идентификатор может обозначать глобальную или локальную переменную (либо формальный параметр функции, что является частным случаем локальной переменной) :

```
var ::= Name
```

Где Name – идентификатор, определяемый в соответствии с §2.1. [http://www.lua.ru/doc/2.1.html]

Любая переменная считается глобальной, если она явно не объявлена как локальная (смотрите §2.4.7 [http://www.lua.ru/doc/2.5.8.html]). Локальные переменные существуют в лексическом контексте: локальные переменные доступны функциям, определенным внутри этого контекста (смотрите §2.6 [http://www.lua.ru/doc/2.6.html]).

До первого явного присвоения значением переменной является **nil**.

Квадратные скобки используются для доступа к элементу таблицы по индексу:

```
var ::= prefixexp '[' exp ']'
```

Способ доступа к глобальным переменным и полям таблицы может быть изменен с помощью метатаблиц. Доступ к переменной t[i] эквивалентен вызову gettable_event(t,i). (Полное описание

функции gettable_event смотрите в §2.8 [http://www.lua.ru/doc/2.8.html]. Эта функция недоступна в коде Lua, мы упомянули ее здесь в качестве примера).

Запись var.Name аналогична записи var["Name"]:

```
var ::= prefixexp '.' Name
```

Все глобальные переменные являются полями в обычных таблицах Lua, называемых таблицами окружения или кратко окружениями (смотреть §2.9 [http://www.lua.ru/doc/2.9.html]). Каждая функция имеет ссылку на свое собственное окружение, и все глобальные переменные внутри этой функции ссылаются на данную таблицу. В момент создания функция наследует окружение вызывающей функции.

1.1.4. Операторы

В Lua поддерживается в общем стандартный набор операторов, почти как в Pascal или C. Он состоит из операторов присваивания, операторов управления потоком исполнения, вызова функций и описания переменных.

1.1.4.1. Порции

Единица исполнения Lua называется chunk (порция). Порция – это любая последовательность операторов Lua. Операторы в порции могут разделяться запятыми:

```
chunk ::= {stat [';']}
```

Пустого оператора в языке нет, поэтому выражение ';;' недопустимо.

Lua воспринимает порцию как неименованную функцию с произвольным набором параметров (смотрите §2.5.9 [http://www.lua.ru/doc/2.5.9.html]). Порция может определять локальные переменные и возвращать значения.

Порция может храниться в файле или в строке базовой программы. В момент запуска порции на выполнение осуществляется компиляция ее в промежуточный байт-код (инструкции для виртуальной машины). Затем полученный код исполняется виртуальной машиной.

1.1.4.2. Блоки

Блок - это список операторов; синтаксически блок тождественно равен порции (chunk) :

```
block ::= chunk
```

Блок операторов может быть явно ограничен, таким образом представляется составной оператор:

```
stat ::= do block end
```

С помощью составных операторов можно ограничивать области видимости локальных переменных. Также составные операторы используются в циклах и условном операторе (смотрите §2.4.4 [http://www.lua.ru/doc/2.4.4.html]).

1.1.4.3. Присваивание

Lua поддерживает параллельное присваивание. В общем случае, оператор присваивания выглядит как список переменных, символ ‘=’ и список выражений. Элементы списков указываются через запятую:

```
stat ::= varlist1 '=' explist1
varlist1 ::= var {',' var}
explist1 ::= exp {',' exp}
```

Выражения (exp) рассмотрены в §2.5 [http://www.lua.ru/doc/2.5.html].

Перед выполнением присваивания список переменных согласовывается по длине со списком выражений. Если список справа длиннее, то его лишние элементы просто отбрасываются. Если короче, то недостающие позиции дополняются значениями **nil**. Если список операторов оканчивается вызовом функции, то перед согласованием все возвращаемые оттуда значения вставляются в правый список (за исключением случаев, когда вызов взят в скобки; смотрите §2.5 [http://www.lua.ru/doc/2.5.html]).

Перед выполнением присваивания вычисляется значение всех выражений. Код

```
i = 3
i, a[i] = i+1, 20
```

означает, что переменной a[3] присваивается значение 20, потому что i в выражении a[i] имеет то же самое значение, что и в момент вычисления выражения i+1. Аналогично, строка

```
x, y = y, x
```

является простым способом обмена значениями двух переменных.(при «традиционном» способе требуется дополнительная переменная).

Действие операции присваивания для глобальных переменных и полей таблиц может быть переопределено посредством метатаблиц. Присваивание индексной переменной t**[**i**]=** val эквивалентно вызову settable_event**(**t**,**i**,**val**)**. (Полное описание функции settable_event смотрите в §2.8 [http://www.lua.ru/doc/2.8.html]. Эта функция недоступна в коде Lua, мы упомянули ее здесь в качестве примера.)

Присваивание глобальной переменной x=val эквивалентно присваиванию _env.x=val, то же самое произойдет при вызове

```
settable_event(_env, "x", val)
```

где _env - окружение запущенной функции. (Переменная _env недоступна в Lua, мы приводим ее здесь в качестве примера.)

1.1.4.4. Управляющие конструкции

Операторы **if**, **while**, и **repeat** имеют обычное значение и знакомый синтаксис:

```
stat ::= while exp do block end
stat ::= repeat block until exp
```

```
stat ::= if exp then block
           {elseif exp then block}
           [else block]
         end
```

В Lua также имеется выражение **for** в двух вариантах (смотрите §2.4.5 [http://www.lua.ru/doc/2.4.5.html]).

Логическое выражение в управляющих конструкциях может возвращать любое значение. Значения **false** и **nil** считаются ложными. Все остальные значения считаются истинными (в том числе значение 0 и пустая строка!).

Цикл **repeat**–**until** заканчивается условием, идущим следом за **until**, поэтому в условии можно ссылаться на локальные переменные, описанные внутри цикла.

Выражение **return** используется для того, чтобы возвратить значения из функции или порции. Синтаксис оператора **return** позволяет функции или порции вернуть несколько значений:

```
stat ::= return [explist1]
```

Оператор **break** используется для досрочного выхода из циклов **while**, **repeat** и **for**:

```
stat ::= break
```

break прерывает цикл, в теле которого встречается, внешние циклы продолжают выполнение.

return (или **break)** должен быть *последним* оператором в блоке (иначе следующие за ним операторы никогда не выполнятся). Если действительно необходимо вставить **return** или **break** в середину блока, то следует применить составной оператор, например **do return end** и **do break end**.

1.1.4.5. Оператор for

Оператор **for** допускает простую и расширенную формы записи.

В простой форме **for** выполняет блок кода до тех пор, пока переменная цикла, изменяеющаяся в арифметической прогрессии, не достигнет установленного порога.

```
stat ::= for Name ' = ' exp ',' exp [', ' exp] do block end
```

block повторяется для переменной цикла name, начиная со значения первого выражения exp, до тех пор пока выполняется второе выражение exp с шагом третьего выражения exp.

Таким образом, запись

```
for v = e1, e2, e3 do block end
```

эквивалентна коду

```
do
    local var, limit, step = tonumber(e1), tonumber(e2), tonumber(e3)
    if not(var and limit and step) then error() end
    while (step>0 and var<=limit) or (step<=0 and var>=limit) do
        local v = var
        var = var + step
     end
```

```
end
```

Обратите внимание, что:

- все три логических выражения вычисляются только один раз перед началом цикла, причем полученные значения должны быть числами;

- var, limit, и step - неявные переменные, мы условно именовали их здесь для объяснения логики работы;

- если выражение step (шаг) отсутствует, то по умолчанию используется 1;

- для выхода из цикла **for** используйте **break**;

- переменная **v** является локальной для цикла; вы не сможете использовать ее значение после выхода из цикла **for**. Если вам необходимо значение этой переменной, присвойте его другой переменной перед выходом из цикла.

Расширенная форма оператора **for** реализована с использованием функций *итераторов*. На каждом обороте для получения нового значения переменной цикла вызывается *итератор*. Цикл заканчивается, когда *итератор* вернет **nil**. Синтаксис расширенного оператора **for**:

```
stat ::= for namelist in explist1 do block end
namelist ::= Name {', '~ Name}
```

Запись

```
for var_1, ···, var_n in explist do block end
```

можно представить как

```
do
    local f, s, var = explist
    while true do
        local var_1, ···, var_n = f(s, var)
        var = var_1
        if var == nil then
            break
        end
    end
end
```

Заметим, что

- explist вычисляется только однажды. Его результатом является функция - *итератор,* таблица состояний и начальное значение индекса;

- f, s, и var - неявные переменные, именованные здесь для примера;

- выйти из цикла можно с помощью оператора **break**;

- переменная var_i является локальной; вы не сможете использовать ее значение после выхода из **for**. Если вам необходимо ее значение, заранее сохраните его в другой переменной.

1.1.4.6. Вызов функции

Для создания побочных эффектов может быть полезен вызов функций, используемый в качестве оператора:

```
stat ::= functioncall
```

В этом случае все возвращаемые значения отбрасываются. Вызовы функций рассматриваются в §2.5.8 [http://www.lua.ru/doc/2.5.8.html].

1.1.4.7. Локальные объявления

Локальные переменные могут быть объявлены где угодно внутри блока. Объявление может включать инициализацию:

```
stat ::= local namelist ['=' explist1]
```

Инициализация обладает всеми свойствами операции присваивания (в том числе параллельностью) (смотрите §2.4.3 [http://www.lua.ru/doc/2.4.3.html]). По умолчанию все переменные инициализируются значением **nil**.

Порция является блоком (смотрите §2.4.1 [http://www.lua.ru/doc/2.4.1.html]), поэтому локальные переменные могут быть объявлены вне любого явно заданного блока. Областью действия таких локальных переменных являются границы порции.

Правила видимости для локальных переменных рассмотрены в §2.6 [http://www.lua.ru/doc/2.6.html].

1.1.5. Выражения

Выражениями в Lua являются следующие конструкции:

```
exp ::= prefixexp
exp ::= nil | false | true
exp ::= Number
exp ::= String
exp ::= function
exp ::= tableconstructor
exp ::= '...'
exp ::= exp binop exp
exp ::= unop exp
prefixexp ::= var | functioncall | '(' exp ')'
```

Числа и символьные строки рассмотрены в §2.1 [http://www.lua.ru/doc/2.5.8.html]; переменные - в §2.3 [http://www.lua.ru/doc/2.3.html]; описания функций - в §2.5.9 [http://www.lua.ru/doc/2.5.9.html]; вызовы функций - в §2.5.8 [http://www.lua.ru/doc/2.5.8.html]; конструкторы таблиц - в §2.5.7 [http://www.lua.ru/doc/2.5.7.html]. Неявные аргументы, обозначаемые '**...**', могут использоваться только внутри соответственно заданной функции; смотрите §2.5.9 [http://www.lua.ru/doc/2.5.9.html].

К бинарным операциям (binop в формальном определении выражения) относятся арифметические (смотрите §2.5.1 [http://www.lua.ru/doc/2.5.1.html]), операции сравнения (§2.5.2 [http://www.lua.ru/doc/2.5.2.html]), булевские (§2.5.3 [http://www.lua.ru/doc/2.5.3.html]) и операции конкатенации (смотреть §2.5.4 [http://www.lua.ru/doc/2.5.4.html]). Унарными являются унарный

минус (§2.5.1 [http://www.lua.ru/doc/2.5.1.html]), отрицание **not** (§2.5.3 [http://www.lua.ru/doc/2.5.3.html]) и операция получения длины # (§2.5.5 [http://www.lua.ru/doc/2.5.5.html]).

Результат вызова функций и неявные параметры могут содержать несколько значений. Если при этом они используются в качестве оператора (§2.4.6 [http://www.lua.ru/doc/2.4.6.html]) (только для функций), то все возвращенные значения отбрасываются. Если это последний (или единственный) элемент в списке выражений, то никакая корректировка не проводится (если вызов не взят в скобки). В остальных случаях Lua приводит возвращаемый список к одному элементу, отбрасывая все значения, кроме первого.

Далее несколько примеров:

```
f() -- результат функции отбрасывается
a = f() -- берется 1-е значение из списка - результата вызова f()
a, b = f()
         -- берется 1-е и 2-е значение из списка - результата вызова f()
         -- (причем a и b могут получить nil, если в качестве результата
         -- ничего не передано)
return f() -- возвращает все значения из f()
return ... -- возвращает все полученные неявные аргументы
return x, y, f() -- вернет x, y и все, что вернет f()
{f()} -- создаст список со всем результатами вызова f()
{...} -- создаст список со всеми неявными параметрами
{f(), nil} -- 1 результат из f()
```

Выражение, заключенное в скобки, всегда возвращает только одно значение. Таким образом, (f(x,y,z)) всегда даст единственное значение, даже если f возвращает несколько. Значение f(x,y,z) - это первое значение, полученное из f, или nil, если f не возвращает значений.

1.1.5.1. Арифметические операции

Lua поддерживает обычные арифметические операции: двоичные + (сложение), **-** (вычитание), ***** (умножение), / (деление), **%** (остаток от деления), и ^ (возведение в степень); а также унарный минус **-** (изменение знака числа). Если операнды являются числами или строками (которые могут быть преобразованы в числа§2.2.1 [http://www.lua.ru/doc/2.2.1.html]), то операции выполняются обычным образом . Возведение в степень работает для любого показателя степени. Например, x^(-0.5) подсчитывает величину, обратную квадратному корню из x. Остаток от деления определен как

```
a % b == a - math.floor(a/b)*b
```

1.1.5.2. Побитовые операторы

Lua поддерживает следующие побитовые операторы:

- **&**: побитовое И (bitwise and);
- |: побитовое ИЛИ (bitwise or);
- ~|: побитовое исключающее ИЛИ (bitwise exclusive or);

- >>: сдвиг вправо (right shift);
- <<: сдвиг влево (left shift) ;
- ~: унарный побитовый НЕ (unary bitwise not).

Все побитовые операции конвертируют свои операнды до целых чисел, работают на всех битах этих целых чисел и в результате получают целое число.

Оба оператора сдвига, правый и левый, заполняют незанятые биты нулями. При отрицательных значениях смещения сдвиг выполняется в противоположном направлении; перемещения абсолютных значений, равных или больших числа бит в целом числе, приводят в результате к нулю (поскольку все биты оказываются сдвинутыми).

1.1.5.2.1. Операции сравнения

Операции сравнения в Lua:

```
== ~= < > <= >=
```

Эти операции всегда возвращают **false** или **true**.

Сравнение на равенство (==) сначала сравнивает типы операндов. Если типы различны, то результатом будет **false**. Иначе сравниваются значения операндов. Числа и строки сравниваются обычным способом. Объекты (таблицы, пользовательские данные, потоки и функции) сравниваются по ссылке: два объекта считаются равными, только если они являются одним и тем же объектом. Создаваемый объект (таблица, пользовательские данные, поток или функция) не может быть равен ни одному из уже существующих.

Вы можете изменить способ, которым Lua сравнивает таблицы и пользовательские данные, используя метаметод "eq" (§2.8 [http://www.lua.ru/doc/2.8.html]).

Правила преобразования из §2.2.1 [http://www.lua.ru/doc/2.2.1.html] НЕ работают в сравнениях на равенство. Например, "0"**==0** вернет **false**, а t[0] и t["0"] обозначают различные записи в таблице.

Оператор ~= прямо противоположен оператору равенства (==).

Операторы сравнения на больше-меньше работают следующим образом. Если оба параметра - числа, то они сравниваются как обычно. Если оба параметра строки, то их значения сравниваются в соответствии с лексикографическим порядком. Во всех остальных ситуациях будет вызван метаметод "lt" или "le" (§2.8 [http://www.lua.ru/doc/2.8.html]).

1.1.5.2.2. Логические операции

В Lua это операции **and** (и), **or** (или), и **not** (не). Так же, как и в управляющих конструкциях (§2.4.4 [http://www.lua.ru/doc/2.4.4.html]), все логические операции рассматривают **false** и **nil** как ложь, а все остальное как истину.

Операция отрицания **not** всегда возвращает **false** или **true**. Операция конъюнкции **and** возвращает свой первый параметр, если его значение **false** или **nil**; в противном случае **and** возвращает второй параметр. Оператор дизъюнкции **or** возвращает первый параметр, если его значение отлично от **nil** и **false**; в противном случае **or** возвращает второй параметр. Оба оператора вычисляют второй операнд только в случае необходимости.

Примеры:

```
10 or 20 --> 10
```

```
10 or error() --> 10
nil or "a" --> "a"
nil and 10 --> nil
false and error() --> false
false and nil --> false
false or nil --> nil
10 and 20 --> 20
```

1.1.5.2.3. Конкатенация

Оператор конкатенации (соединения) строк в Lua обозначается двумя точками ('**..**'). Если оба операнда являются строками или числами, то они будут преобразованы в строки согласно правилам §2.2.1 [http://www.lua.ru/doc/2.2.1.html]. Иначе будет вызван метаметод "concat" (§2.8 [http://www.lua.ru/doc/2.8.html]).

1.1.5.2.4. Получение длины

Операция получения длины обозначается унарным #. В результате применения операции к строке возвращается количество байт (в обычном понимании это длина строки, в которой каждый символ занимает 1 байт).

Длиной таблицы t считается любой целый индекс n такой, что t**[**n**]** не равен **nil**, а t**[**n+1**]** равно **nil**. Кроме того, если t[**1**] равен **nil**, то #t **=0**. Для регулярных массивов от 1 до n, не содержащих значений **nil**, длиной является n, то есть индекс последнего значения. Если в массиве присутствуют "дыры" (т.е., значения **nil** между ненулевыми значениями), то значением #t является индекс элемента, непосредственно предшествующего элементу **nil** (поэтому любое значение **nil** по сути означает конец массива).

1.1.5.2.5. Приоритет операций

Приоритет операций Lua показан на таблице ниже. Самым высоким приоритетом обладает операция возведения в степень, далее по убыванию:

```
or
and
< > <= >= ~= ==
..
+ -
* / %
not # - (unary)
^
```

Как обычно, для изменения порядка вычисления выражений вы можете использовать скобки. Конкатенация ('**..**') и возведение в степень ('^') - правоассоциативные операторы. Все остальные бинарные операторы левоассоциативные.

1.1.5.2.6. Конструкторы таблиц

Конструкторы таблиц тоже относятся к выражениям. Обработка любого встречающегося в коде конструктора ведет к созданию новой таблицы. С помощью конструкторов можно создать как пустые, так и частично либо полностью заполненные таблицы. Полное описание синтаксиса конструкторов:

```
tableconstructor::= '{' [fieldlist] '}'

fieldlist::= field {fieldsep field} [fieldsep]

field::= '[' exp ']' '=' exp | Name '=' exp | exp

fieldsep::= ',' | ';'
```

Каждое поле вида **[**exp1**]=** exp2 добавляет в новую таблицу значение exp2 с ключом exp1. Поле вида name **=exp** эквивалентно **[**"name"**]=** exp. Поле вида **exp** эквивалентно **[**i**]=exp**, где i – целочисленный автоинкрементный счетчик, начинающийся с 1. Поля в других форматах не оказывают влияния на этот счетчик. Например,

```
a = { [f(1)] = g; "x", "y"; x = 1, f(x), [30] = 23; 45 }
```

эквивалентно

```
do
    local t = {}
    t[f(1)] = g
    t[1] = "x"
    t[2] = "y"
    t.x = 1
    t[3] = f(x)
    t[30] = 23
    t[4] = 45
    a = t
end
```

Если последнее поле в списке задано в форме exp, и exp – это вызов функции или неопределенный список параметров, то все значения, возвращаемые этим выражением, последовательно включаются в этот список (§2.5.8 [http://www.lua.ru/doc/2.5.8.html]). Чтобы этого избежать, необходимо заключить вызов функции (или список неопределенных параметров) в скобки (§2.5 [http://www.lua.ru/doc/2.5.html]).

Список полей может оканчиваться разделителем, что улучшает читабельность машинно-генерируемого кода.

1.1.5.2.7. Вызовы функций

Вызовы функций в Lua имеют следующий синтаксис:

```
functioncall ::= prefixexp args
```

В вызове функции сначала вычисляются префиксное выражение и аргументы. Если значение префиксного выражения имеет тип function, то эта функция будет вызвана с указанными аргументами. В противном случае вызывается метаметод “call”, параметрами которого будет значение префиксного выражения, за которым следуют первоначальные аргументы (§2.8 [http://www.lua.ru/doc/2.8.html]).

Форма записи

```
functioncall ::= prefixexp ':' Name args
```

может использоваться для вызова "методов". Запись v:name**(**args**)** синтаксически аналогична записи v.name**(**v**,**args**)**, только v вычисляется один раз.

Аргументы описываются следующим образом:

```
args ::= '(' [explist1] ')'
args ::= tableconstructor
args ::= String
```

Все выражения вычисляются перед вызовом. Вызов в форме f{fields} синтаксически аналогичен f({fields}); то есть список аргументов является по сути новой таблицей. Вызов в форме f'string' (или f"string" или f[[string]]) синтаксически равен f('string'); в данном случае список аргументов - единственная символьная строка.

Исключением в довольно свободном синтаксисе Lua является правило, по которому нельзя переходить на новую строку непосредственно перед символом '(' в вызове функции. Это ограничение позволяет избежать некоторой двусмысленности в языке. Если вы напишете

```
a = f
(g).x(a)
```

Lua трактует эту запись как выражение a = f**(**g**).**x**(**a**)**. Поэтому, если вам нужно 2 выражения, вы должны добавить точку с запятой между ними. Если вы действительно хотите вызвать f, вам необходимо убрать переход на новую строку перед (g).

Вызов в форме returnfunctioncall называется концевым вызовом. Lua также поддерживает концевой вызов «себя»(или *рекурсивный* концевой вызов): в этом случае вызванная функция использует стек вызывающей функции. Поэтому количество вложенных концевых вызовов может быть любым. Заметим только, что концевой вызов стирает отладочную информацию о вызывающей функции. Синтаксис концевого вызова допускает только единичный вызов функции после оператора **return**. Таким образом, **return** вернет в точности тот результат, что вернет вызов функции. Ни один из представленных ниже примеров не является допустимым концевым вызовом:

```
return (f(x)) -- список-результат обрезается
return 2 * f(x) -- удвоение результата функции
return x, f(x) -- возвращается несколько значений
f(x); return -- результат вызова отбрасывается
return x or f(x) -- список-результат обрезается
```

1.1.5.2.8. Объявление функций

Синтаксис объявления функций:

```
function ::= function funcbody
funcbody ::= '(' [parlist1] ')' block end
```

Или в упрощенном виде

```
stat ::= function funcname funcbody
stat ::= local function Namefuncbody
funcname ::= Name {'.' Name} [':' Name]
```

Выражение

```
function f () body end
```

транслируется в

```
f = function () body end
```

Выражение

```
function t.a.b.c.f () body end
```

транслируется в

```
t.a.b.c.f = function () body end
```

Выражение

```
local function f () body end
```

транслируется в

```
local f; f = function () body end
```

а не в

```
local f = function () body end
```

Разница проявится в том случае, если в теле функции используется имя этой функции, например, при рекурсивном вызове.

Объявление функции является выполняемым выражением, его результатом будет значение типа function. Когда Lua прекомпилирует порцию, тела всех упоминающихся в ней функций также прекомпилируются. Таким образом, всякий раз, когда Lua обрабатывает объявление функции, функция уже *конкретизирована* (или *замкнута*). Этот конкретный экземпляр функции (или замыкание) и является конечным значением выражения «объявление функции». Различные экземпляры одной и той же функции могу ссылаться на различные внешние локальные переменные и иметь различные таблицы окружения.

Параметры функции фактически являются локальными переменными, которые инициализированы входными значениями:

```
parlist1 ::= namelist [',' '...'] | '...'
```

В момент вызова функции длина списка передаваемых параметров приводится в соответствие спецификации, если это не функция с неопределенным количеством параметров. Для функций с

неопределенным количеством параметров такая коррекция не проводится; все входные параметры попадают в функцию в виде *неопределенного выражения*, которое также обозначается с тремя точками. Значением этого выражения является список всех полученных входных параметров, как в случае множественного результата функции. Если *неопределенноевыражение* используется внутри другого выражения или в середине списка выражений, то его значение-список урезается до одного элемента. Если это выражение стоит в конце списка выражений, урезания не происходит (если, конечно, вызов не заключен в круглые скобки).

Рассмотрим следующие объявления:

```
function f(a, b) end
function g(a, b, ...) end
function r() return 1,2,3 end
```

Пример отображения входных значений на параметры функции:

```
ВЫЗОВ                    ПАРАМЕТРЫ
f(3)                     a=3, b=nil
f(3, 4)                  a=3, b=4
f(3, 4, 5)               a=3, b=4
f(r(), 10)               a=1, b=10
f(r())                   a=1, b=2
g(3)                     a=3, b=nil, ... --> (ничто)
g(3, 4)                  a=3, b=4, ... --> (ничто)
g(3, 4, 5, 8)            a=3, b=4, ... --> 5 8
g(5, r())                a=5, b=1, ... --> 2 3
```

Результаты возвращаются из функции оператором **return** (см. §2.4.4 [http://www.lua.ru/doc/2.4.4.html]). . Если управление достигает конца функции, а оператор **return** не встретился, то функция завершается и ничего не возвращает.

Синтаксис с двоеточием**‘:’**используется для определения методов.Эти функции неявно получают параметр self в качестве первого аргумента. Таким образом, выражение

```
function t.a.b.c:f (params) body end
```

аналогично

```
t.a.b.c.f = function (self, params) body end
```

1.1.6. Области видимости

Lua - язык с лексическим разграничением областей видимости. Область видимости переменной начинается первым выражением *после* ее объявления и действует до конца блока, в котором это объявление встречается. Рассмотрим следующий пример:

```
x = 10 -- глобальная переменная variable
do -- начало блока
    local x = x -- объявление локальной переменной
    print(x) --> 10
    x = x+1
    do -- начало вложенного блока
```

```
        local x = x+1 -- другая локальная 'x'
        print(x) --> 12
    end
    print(x) --> 11
end
print(x) --> 10 (глобальная переменная)
```

Отметим, что в объявлении local x = x локальная переменная объявляется еще не в области своей видимости, поэтому присваивается именно внешняя переменная.

В соответствии с правилами лексического разграничения областей видимости, локальные переменные доступны в функциях, определенных внутри их области видимости. Локальная переменная, используемая в таких функциях, называется *внешней* локальной переменной(по отношению к определенной внутри ее области видимости функции).

Обработка каждого объявления **local** ведет к созданию новой локальной переменной. Рассмотрим следующий пример:

```
a = {}
local x = 20
for i=1,10 do
    local y = 0
    a[i] = function () y=y+1; return x+y end
end
```

Цикл создает 10 экземпляров функции, в которых используются различные переменные y и один и тот же x.

1.1.7. Обработка ошибок

Lua-код может явно генерировать ошибку, вызывая функцию error. Если вам нужно перехватывать ошибки в самом Lua, вы можете использовать функцию pcall.

1.1.8. Метатаблицы

Любое значение в Lua может иметь *метатаблицу*. *Метатаблица* –это обычная таблица Lua, в которой определены допустимые операции над значением. Вы можете изменить действие некоторых операций над значением путем задания соответствующих полей в его метатаблице. Для экземпляров классов, например, когда к нечисловым значениям применяется операция сложения, Lua ищет реализацию этой операции в поле "__add" его метатаблицы. Если реализация найдена, Lua запускает эту операцию для выполнения сложения.

Ключевые поля в метатаблице мы называем *событиями,*а значения - *метаметодами*. В рассмотренном примере событием является "add", а метаметодом является функция сложения.

Получить метатаблицу любого значения можно с помощью функции getmetatable.

Вы можете заменить метатаблицу с помощью функции setmetatable. Вы не можете изменить метатаблицу другим способом.

Таблицы и пользовательские данные имеют индивидуальные метатаблицы (в то же время множество таблиц и пользовательских данных может совместно использовать соответствующие

метатаблицы). Переменные всех других типов совместно используют одну метатаблицу на тип. То есть есть одна метатаблица на все числовые значения, одна на все строки и т.д.

В метатаблице могут быть заданы правила выполнения арифметических операций над объектом, порядок сравнения, конкатенации, способ вычисления длины и индексирования. Также в метатаблице может быть определена функция «сборки мусора». Для каждой из этих операций в Lua определен специальный ключ, называемый *событие*. В момент, когда Lua выполняет одну из этих операций над значением, проверяется, есть ли в метатаблице значение с соответствующим событием. Если оно найдено, значение по этому ключу (метаметод) и определяет способ выполнения операции.

Далее рассмотрим управление операциями с помощью метатаблиц. Каждая операция идентифицируется по своему имени. Ключ каждой операции представляет из себя строку из имени и двух нижних подчеркиваний перед ним. Для экземпляра – ключом операции “add” будет строка "__add". Для лучшего понимания мы в терминах Lua покажем запуск операции интерпретатором.

Представленный здесь код на Lua приведен в качестве иллюстрации, реальный код интерпретатора гораздо сложнее и эффективнее этого схематичного примера. Все функции, использованные в этом примере (rawget, tonumber и т.п.), описаны в §3.1 [http://www.lua.ru/doc/3.1.html]. Вообще говоря, для получения метаметода объекта мы используем конструкцию

```
metatable(obj)[event]
```

Это следует читать как

```
rawget(getmetatable(obj) or {}, event)
```

Таким образом, при получении доступа к метаметоду другие метаметоды не используются, поэтому доступ к объекту без метатаблицы не приводит к ошибке (мы просто получим **nil**).

- **"add"**: операция сложения.

Функция getbinhandler ниже показывает, каким образом Lua получает указатель на операцию. Во-первых, Lua пытается получить первый операнд. Если его тип не является типом указателя на операцию, Lua пытается получить второй операнд.

```
function getbinhandler(op1, op2, event)
    return metatable(op1)[event] or metatable(op2)[event]
end
```

С помощью этой функции поведение конструкции op1 + op2 выглядит как

```
function add_event (op1, op2)
    local o1, o2 = tonumber(op1), tonumber(op2)
    if o1 and o2 then -- операнды являются числами?
       return o1 + o2 -- '+' здесь – стандартная операция сложения
   else -- когда хотя бы один из операндов нечисловой
       local h = getbinhandler(op1, op2, "__add")
        if h then
           -- вызов функции с операндами по указателю
           return h(op1, op2)
        else -- указатель не найден: обработка ошибки
           error(···)
        end
    end
end
```

- **"sub"**: операция ‘ - ‘. Обработка аналогична "add";
- **"mul"**: операция ‘ * ’. Обработка аналогична "add";
- **"div"**: операция ‘ / ’ . Обработка аналогична “add”;
- **"mod"**: операция ‘ % ’. Обработка похожа на "add", только o1 - floor(o1/o2)*o2 подставляется вместо стандартной операции;
- **"pow"**: операция ‘ ^ ’. Обработка похожа на "add", вместо стандартной операции подставляется функция pow (из математической библиотеки C).

1.1.9. Окружение

Кроме метатаблиц, объекты типа thread, function и userdata обладают дополнительными таблицами, которые называют *окружением*. Подобно метатаблицам, окружения являются регулярными таблицами и множество объектов могут совместно использовать одно окружение.

Окружение, соответствующее типу userdata, не имеет смысла в Lua. Оно введено для удобства ассоциирования таблиц и значений типа userdata.

Окружения, относящиеся к типу threads, называют *глобальными окружениями*. Они используются как окружения по умолчанию для потоков и невложенных функций, созданных в потоке (с помощью loadfile, loadstring или load) и могут использоваться непосредственно в C-коде (смотрите §3.3 [http://www.lua.ru/doc/3.3.html]).

Окружения, ассоциированные со стандартными функциями Lua, используются для доступа к глобальным переменным из любой функции (смотрите §2.3 [http://www.lua.ru/doc/2.3.html]). Они используются по умолчанию для других функций Lua, построенных на базе этих..

1.1.10. Сборщик мусора

Lua осуществляет автоматическое управление памятью. Это означает, что вам не нужно думать о выделении памяти при создании новых объектов и ее освобождении, когда объект становится ненужным. Lua время от времени автоматически запускает процедуру *сборки мусора* для удаления *устаревших объектов* (то есть объектов, которые более недоступны из Lua). Сборщик мусора обрабатывает все объекты Lua: таблицы, данные типа userdata, функции, потоки и строки.

В Lua реализован инкрементный сборщик по принципу пометить-очистить. Цикл работы сборщика мусора зависит от двух параметров: *пауза сборки мусора* и *коэффициент шага сборки*.

Паузой определяется время между запусками циклов сборки. Большие значения этого параметра делают сборку мусора менее активной. Значения меньше 1 означают, что между запусками циклов сборки паузы нет. При значении 2 сборщик перед следующим запуском ждет удвоения объема использованной памяти.

Коэффициент шага сборки управляет скоростью сборки в зависимости от интенсивности выделения памяти. Большие значения параметра ускоряют работу сборщика, но при этом увеличивается размер каждого шага. Значения меньше 1 делают сборщик медленным и могут привести к тому, что цикл сборки никогда не закончится. По умолчанию используется значение 2, в этом случае сборщик работает вдвое быстрее процесса выделения памяти.

Вы можете менять эти параметры посредством вызова collectgarbage в Lua. В обоих случаях в качестве аргументов берутся проценты (т.е. аргумент 100 означает значение параметра 1). Этими функциями вы можете непосредственно управлять сборкой (например, останавливать ее или рестартовать).

1.1.10.1. Метаметоды сборщика мусора

Данные сборщика мусора с полем __gc в собственной метатаблице не хранятся непосредственно в нем. Вместо этого Lua сохраняет их в списке. После сбора мусора Lua выполняет над этими данными функцию, эквивалентную:

```
function gc_event (udata)
    local h = metatable(udata).__gc
    if h then
        h(udata)
    end
end
```

После каждого цикла сбора финализаторы каждого элемента данных вызываются в порядке, обратном порядку их создания в цикле. Таким образом, первый финализатор будет вызван для объекта, созданного последним.

1.1.10.2. Таблицы «слабых» ссылок

Таблицы этого типа используются для хранения «слабых» ссылок (weakreferences). Сборщик мусора игнорирует «слабые» ссылки. Другими словами, если единственной ссылкой на объект является «слабая» ссылка, то сборщик мусора сохраняет этот объект в такой таблице.

Таблица «слабых» ссылок может содержать «слабые» ключи, «слабые» значения, а также и то, и другое. Таблица со «слабыми» ключами позволяет хранить эти ключи, но не допускает хранение их значений. Таблица со «слабыми» ключами и «слабыми» значениями соответственно позволяет хранить и ключи, и значения. Таким образом, если ключ или значение сохранены, то соответствующая пара удаляется из таблицы. «Слабость» в таблице контролируется полем __mode ее метатаблицы. Если поле __mode - это строка, содержащая символ k', то ключи в таблице «слабые». Если __mode содержит 'v', то в таблице хранятся «слабые» значения.

После использования таблицы как метатаблицы вы не можете изменить поле __mode. Таким образом, «слабое» поведение таблицы, за которое отвечает метатаблица, переопределить нельзя.

1.1.11. Сопрограммы

Lua поддерживает сопрограммы, эту технологию часто называют общей многопоточностью. Сопрограмма Lua представляет собой независимый поток выполнения. Несмотря на это, в отличие от потоков в традиционных многопоточных системах, сопрограмма может приостановить свое выполнение только в результате явного вызова функции corutine.yield.

Сопрограммы создаются вызовом coroutine.create. Единственным аргументом является имя главной функции сопрограммы. Функция create только создает новую сопрограмму и возвращает указатель на нее (объект типа thread), запуск сопрограммы не выполняется.

При вызове функции coroutine.resume и передаче ей в качестве первого аргумента результата вызова coroutine.create, процедура запускается на выполнение с первого оператора ее главной функции. Остальные параметры из вызова coroutine.resume передаются в основную функцию сопрограммы. После запуска сопрограмма выполняется до завершения либо до вызова yields.

Сопрограмма останавливается только в двух случаях: нормально, когда осуществляется возврат (явно или неявно) из главной функции; или аварийно в случае необработанной ошибки. При нормальном завершении coroutine.resume возвращает **true** плюс любые значения, возвращаемые из основной функции сопрограммы. В случае ошибок, coroutine.resume вернет значение **false** плюс сообщение об ошибке.

Для приостановки выполнения подпрограммы используется функция corutine.yield. При вызове yields соответствующий coroutine.resume возвращает управление немедленно, точно так же, как если бы вызов corutine.yield произошел во вложенном вызове функции (т.е. не в главной функции, а в функции, вызванной непосредственно или опосредованно из нее). При вызове corutine.yield функция coroutine.resume также возвращает **true** плюс все входные параметры, переданные в corutine.yield. В следующий раз, когда сопрограмма продолжит работу, ее выполнение начнется с оператора, следующего за corutine.yield, соответственно из corutine.yield вернутся параметры, переданные в coroutine.resume.

Функция coroutine.wrap создает сопрограмму и осуществляет ее запуск. Параметры, переданные в нее в качестве дополнительных аргументов, попадают в неявный вызов coroutine.resume. Вызов coroutine.wrap возвращает те же значения, что и coroutine.resume, за исключением первого (булевского кода ошибки). В отличие от coroutine.resume, coroutine.wrap не перехватывает ошибки – все ошибки попадают на уровень вызывающей функции.

Рассмотрим в качестве примера следующий код:

```
function foo (a)
    print("foo", a)
    return coroutine.yield(2*a)
end

co = coroutine.create(function (a,b)
    print("co-body", a, b)
    local r =foo(a+1)
    print("co-body", r)
    local r, s =coroutine.yield(a+b, a-b)
    print("co-body", r, s)
    return b, "end"
end)

print("main", corutine.resume(co,1, 10))
print("main", corutine.resume(co, "r"))
print("main", corutine.resume(co, "x", "y"))
print("main", corutine.resume(co, "x", "y"))
```

При запуске на экран выведется:

```
co-body    1        10
foo        2
main       true     4
co-body    r
main       true     11       -9
co-body    x        y
main       true     10       end
main       false    cannot   resume   dead   coroutine
```

1.2. Стандартные библиотеки

1.2.1. Базовые функции

Базовая библиотека содержит некоторые основные функции Lua. Если Вы не включили эту библиотеку в Ваше приложение, Вам необходимо тщательно проверить, действительно ли не требуются эти возможности.

Таблица 1.1 - Описание базовых функций

Название функции	Описание функции
assert(v [, message])	Выдает сообщение об ошибке, если значение параметра v равно **false** (т.е., **nil** или **false**); в противном случае, возвращаются все параметры. message – это сообщение об ошибке; если параметр отсутствует, по умолчанию выводится "assertion failed!"
collectgarbage(opt [, arg])	Эта функция является интерфейсом к сборщику мусора (garbage collector)
В зависимости от значения параметра opt, выполняются различные функции: - "**stop**": остановка сборщика мусора; - "**restart**": рестарт сборщика мусора; - "**collect**": выполнение полного цикла сборки мусора; - "**count**": возврат общего количества используемой Lua памяти (в K байтах); - "**step**": выполнение шага по сборке мусора. "Размер" шага регулируется параметром arg (большая величина соответствует большему шагу. Если Вы хотите регулировать размер шага, Вы должны сначала определить экспериментальным путем оптимальное значение arg. Возвращает **true**, если шаг завершает полный цикл сборки мусора; - "**setpause**": присвоение значения arg/100 параметру *пауза сборки мусора* сборщика (см.§2.10 [http://www.lua.ru/doc/2.10.html]); - "**setstepmul**": присвоение значения arg/100 параметру *коэффициент шага сборки* сборщика (см.§2.10 [http://www.lua.ru/doc/2.10.html])	
dofile(filename)	Открывает указанный файл и выполняет его, как Lua задачу (chunk). При вызове без параметров, dofile выполняет содержимое стандартного потока ввода (stdin). Возвращает все данные, возвращаемые выполняемой задачей. В случае ошибки, dofile передает ошибку вызывающей задаче (т.е.dofile запускается в незащищенном режиме)
error(message [, level])	Завершает работу программы и передает движку Lua сообщение об ошибке. Обычно error добавляет информацию о местоположении возникновения ошибки в начало сообщения. Параметр level указывает способ определения места возникновения ошибки. Уровень 1 (по умолчанию) определяет положение ошибки в месте вызова функции error. Уровень 2 определяет место, где была вызвана функция, которая вызвала error; и т.д. Если параметр равен 0, то положение ошибки не добавляется в сообщение
_G	Глобальная переменная (не функция), которая содержит глобальное окружение (environment) (_G._G = _G). Lua не использует

Название функции	Описание функции
	эту переменную; изменение значения этой переменной не отражается на окружении и наоборот
getmetatable(object)	Если object не имеет metatable, возвращается **nil**. В противном случае, если метатаблица объекта имеет поле "__metatable", возвращается значение этого поля. Иначе возвращается метатаблица данного объекта
ipairs(t)	Возвращает три значения: итератор, таблицу t, и 0, поэтому конструкция `for i,v in ipairs(t) do body end` будет выполнять цикл парами (1,t[1]), (2,t[2]), ⋯, до первого целого ключа, отсутствующего в таблице
load(func [, chunkname])	Загружает составной оператор (chunk) с помощью функции func для получения составного оператора по частям. Каждый вызов func должен возвращать строку, которая соединяется с предыдущими результатами. Возврат значения **nil** (или отсутствие значения) говорит о конце составного оператора. В случае отсутствия ошибок возвращается откомпилированный составной оператор как функция; иначе, возвращается **nil** и сообщение об ошибке. Окружением возвращаемой функции является глобальное окружение. **chunkname** используется как имя задачи для сообщения об ошибке, т.е. как отладочная информация
loadfile([filename])	Аналогично load, но задача считывается из файла filename или из стандартного потока ввода, если имя файла не указано
loadstring(string [, chunkname])	Аналогично load, но задача считывается из данной строки. Для загрузки и выполнения данной строки, используйте следующую конструкцию `assert(loadstring(s))()`
next(table [, index])	Позволяет программе получить все поля таблицы. Первый параметр – это таблица, второй – индекс в этой таблице. next возвращает следующий индекс в таблице и соответствующее ему значение. Если второй параметр **nil**, next возвращает начальный индекс и связанное с ним значение. При вызове последнего индекса, или с **nil** в пустой таблице, next возвращает **nil**. Если второй параметр отсутствует, он интерпретируется как **nil**. В частности, Вы можете использовать next(t) для проверки, пустая таблица или нет. Порядок следования индексов не определен, даже для числовых индексов. Для прохождения по таблице в числовом порядке используйте числовой for или функцию ipairs [http://www.lua.ru/doc/5.1.html#pdf-ipairs].

Название функции	Описание функции
	Результат работы next *не определен* (undefined) если в процессе сканирования таблицы Вы присваиваете какое- либо значение в несуществующее поле таблицы. Однако Вы можете модифицировать существующие поля. В частности, Вы можете очистить существующие поля
pairs(t)	Возвращает три значения: функцию next (результат вызова, *примечание переводчика*), таблицу t и **nil**, поэтому конструкция `for k,v in pairs(t) do body end` будет выполнять итерацию по всем ключевым парам таблицы t. См. предупреждения для функции next если Вы хотите в процессе итерации модифицировать таблицу
pcall(f, arg1, ⋯)	Вызывает функцию f с данными параметрами в защищенном режиме (*protected mode*). Это значит, что при возникновении любой ошибки внутри функции f она дальше не передается; напротив, pcall перехватывает ошибку и возвращает статус. pcall возвращает сначала статус (тип boolean), который равен true, если вызов завершился без ошибок. В этом случае, pcall также возвращает все данные, которые возвращает функция f, сразу после статуса. В случае возникновения ошибки, pcall возвращает **false** и сообщение об ошибке
print(⋯)	Принимает любое количество параметров и отображает их значения в stdout, используется функция tostring для преобразования параметров в строки. print неудобен для форматированного вывода, он удобен для быстрого отображения значения, обычно используется для отладки. Чтобы сформатировать вывод, используйте string.format
rawequal(v1, v2)	Проверка равенства v1 и v2, без вызова каких-либо метаметодов. Возвращает статус типа boolean
rawget(table, index)	Возвращает реальное значение из table[index], без вызова метаметодов. table должен быть таблицей; index может быть любым значением
rawset(table, index, value)	Присваивает value реальное значение из table[index] без выполнения метаметодов. table должен быть таблицей, index - любое значение, отличное от **nil**, value – любое значение Lua. Данная функция возвращает table
select(index, ⋯)	Если index - число, возвращает все параметры после параметра с номером index. В противном случае, index должен быть строкой "#", в этом случае select возвращает общее количество параметров (исключая index)
setfenv(f, table)	Устанавливает окружение, которое будет использовано данной функцией. f может быть как Lua функцией, так и числом, которое указывает уровень стека функции: уровень 1 – это функция, вызывающая setfenv. setfenv возвращает данную функцию. В особом случае, когда f равно 0, setfenv изменяет окружение запущенной нити. В этом случае setfenv ничего не возвращает

Название функции	Описание функции
setmetatable(table, metatable)	Устанавливает метатаблицу для данной таблицы. (Вы не можете изменить метатаблицу из Lua, это можно сделать только из C. Если параметр metatable равен **nil**, то производится удаление метатаблицы для данной таблицы. Если метатаблица содержит поле "__metatable", то возникает ошибка. Функция возвращает таблицу table
tonumber(e [, base])	Пытается преобразовать параметр e в число. Если параметр уже является числом или строкой, конвертируемой в число, то tonumber возвращает это число; иначе возвращает **nil**. Необязательный параметр base указывает основание системы счисления для конвертации числа. Основание может быть любым целым числом в диапазоне от 2 до 36, включительно. Если основание больше 10, то символ 'A' (как в верхнем, так и в нижнем регистре) представляет 10, 'B' представляет 11, и так далее, символ 'Z' представляет 35. При основании 10 (по умолчанию), число может быть представлено в экспоненциальной форме (см. §2.1 [http://www.lua.ru/doc/2.1.html]). Для других оснований можно указывать только беззнаковые целые числа
tostring(e)	Принимает параметр любого типа и конвертирует его в строку в подходящем формате. Если требуется специальное форматирование, используйте функцию string.format. Если метатаблица для e имеет поле "__tostring", tostring вызывает его значение с параметром e и возвращает результат этого вызова
type(v)	Возвращает тип параметра в виде строки. Возможные результаты этой функции - это "nil" (как строка, а не значение **nil**), ">number", "string", "boolean", "table", "function", "thread" и "userdata"
unpack(list [, i [, j]])	Возвращает элементы из данной таблицы. Функция эквивалентна следующей конструкции `return list[i], list[i+1], ···, list[j]` с тем исключением, что указанная выше конструкция может быть написана только для фиксированного количества элементов. По умолчанию, i равно 1, а j – длине списка, как это определено в операторе length (см. §2.5.5 [http://www.lua.ru/doc/2.5.5.html])
_VERSION	Глобальная переменная (не функция) содержит строку, описывающую номер версии интерпретатора. В настоящий момент значение этой переменной равно "Lua 5.3"
xpcall(f, err)	Эта функция аналогична pcall, с тем отличием, что здесь Вы можете установить обработчик ошибок. xpcall вызывает функцию f в защищенном режиме, используя обработчик ошибок err. Любые ошибки внутри f не передаются в вызывающую программу; напротив, xpcall перехватывает ошибку, вызывает функцию err с объектом, вызвавшим ошибку, и возвращает

Название функции	Описание функции
	статус. Возвращает сначала статус (тип boolean), который равен **true**, если вызов завершился без ошибок. В этом случае, xpcall также возвращает всё, что возвращает вызов функции f, сразу после статуса. В случае возникновения ошибок xpcall возвращает **false** и результат из err

1.2.2. Работа с сопрограммами

Операции для работы с сопрограммами собраны в виде подбиблиотеки базовой библиотеки и содержатся внутри таблицы coroutine. См.§2.11 [http://www.lua.ru/doc/2.11.html] описание сопрограмм.

Таблица 1.2 - Описание функций работы с сопрограммами

Название функции	Описание функции
coroutine.create(f)	Создает новую сопрограмму, с телом f. Параметр f должен быть Lua функцией. Возвращает новую сопрограмму как объект типа нить ("thread")
coroutine.resume(co [, val1, ⋯])	Запускает или продолжает выполнение сопрограммы co. При первом использовании этой функции начинает выполняться тело функции. Значения val1, ⋯ передаются как параметры в эту функцию. Если сопрограмма была в состоянии yield, resume рестартует сопрограмму, передавая ей параметры val1, ⋯ а также результаты, полученные при вызове yield. Если сопрограмма запускается без ошибок, resume возвращает **true** плюс любые значения, переданные в yield (если сопрограмма была приостановлена ранее вызовом yield) или любые значения, возвращаемые функцией (если сопрограмма завершилась). В случае возникновения ошибок resume возвращает **false** плюс сообщение об ошибке
coroutine.running()	Возвращает выполняемую сопрограмму, или **nil**, если вызвана из главной нити
coroutine.status(co)	Возвращает статус сопрограммы co, в виде строки: "running" (выполняется), если сопрограмма выполняется (собственно, это и есть статус); "suspended" (приостановлена), если сопрограмма приостановлена с помощью вызова yield, либо она еще не запущена на выполнение; "normal" (норма), если сопрограмма активна, но еще не запущена на выполнение (т.е. она активизирована (resume) другой сопрограммой); либо "dead" (остановлена), если сопрограмма завершилась или если остановлена по ошибке
coroutine.wrap(f)	Создает новую сопрограмму с телом f. Параметр f должен быть Lua функцией. Возвращает функцию, с помощью которой можно активировать сопрограмму. Любые параметры, передаваемые в функцию, передаются как дополнительные параметры для resume. Возвращает те же самые значения, возвращаемые resume, кроме

Название функции	Описание функции
	первого boolean. В случае ошибки она передается в вызывающую программу
coroutine.yield(⋯)	Приостанавливает выполнение вызываемой сопрограммы. coroutine не может быть выполняющейся С функцией, метаметодом или итератором. Любые параметры, передаваемые в yield, в дальнейшем передаются как дополнительные параметры при вызове resume

1.2.3. Модули

Библиотека пакетов предоставляет основные возможности для загрузки и построения модулей в Lua. Также она (библиотека, примечание переводчика) экспортирует две функции в глобальное окружение: require и module. Остальные экспортируются в таблицу package.

Таблица 1.3 - Описание функций построения и загрузки модулей

Название функции	Описание функции
module(name [, ⋯])	Создает модуль. Если package.loaded[name] содержит таблицу, то эта таблица является модулем. В противном случае, если существует глобальная таблица t с данным именем name, то эта таблица – модуль. Иначе, если таблица не найдена, то создается новая таблица t и эта таблица помещается в поле - package.loaded[name]. Эта функция также инициализирует t._NAME заданным именем name, t._M собственно модулем (самой t), а t._PACKAGE именем пакета (полное имя модуля без последнего компонента; см. ниже). В конце module устанавливает t как новое окружение текущей функции и новое значение из package.loaded[name], т.о. require возвращает t. Если name – составное (compound) имя (т.е., компоненты имени разделены точками), module создает (или повторно использует, если уже существует) таблицы для каждого компонента. Например, если name равно a.b.c, то module сохраняет модуль в поле c поля b глобального объекта a. Данная функция может получать дополнительные опции options после имени модуля, где каждая опция является функцией, которая будет применена к модулю
require (modname)	Загружает данный модуль. Функция начинается с проверки таблицы package.loaded для определения, загружен модуль modname или нет. Если загружен, то require возвращает значение, хранящееся в package.loaded[modname]. В противном случае, она пытается найти загрузчик (loader) для данного модуля. При поиске загрузчика require сначала запрашивает package.preload[modname]. Если там есть значение, это значение (оно должно быть функцией) является загрузчиком. В противном случае require ищет Lua загрузчик, используя путь (path), хранящийся в package.path. Как только загрузчик найден, require вызывает его с одним параметром: modname. Если загрузчик возвращает какое-

Название функции	Описание функции
	либо значение, require помещает его в package.loaded[modname]. Если загрузчик ничего не возвращает, то package.loaded[modname] устанавливается функцией require в **true**. В любом случае, require возвращает значение из package.loaded[modname]. Если возникают какие - либо ошибки при загрузке или выполнении модуля, или, если не удалось обнаружить загрузчик для модуля, то require вызывает ошибку
package.loaded	Данная таблица используется функцией require для проверки, загружен модуль или нет. Если Вам нужен модуль modname и package.loaded[modname] не равно **false**, require просто возвращает значение, хранящееся там
package.loadlib(libname, funcname)	Динамическое связывание хостовой программы с C библиотекой libname. Внутри этой библиотеки производится поиск функции с именем funcname и она возвращается как C функция. Это функция низкого уровня (low-level function). Она игнорируется пакетной и модульной системой.В отличии от require, здесь не производится поиск путей и не производится автоматическое добавление расширений (extensions). libname должно быть полным именем файла C библиотеки, включая, если необходимо, путь и расширение. funcname должно быть именем, которое экспортировано из C библиотеки (это имя зависит от C компилятора и компоновщика (linker))
package.path	Путь, используемый функцией require для поиска Lua загрузчика. При старте Lua инициализирует эту переменную значением переменной окружения LUA_PATH или значением по умолчанию из luaconf.h, если переменная окружения не определена. Любое вхождение строки ";;" в переменную среды окружения замещается значением по умолчанию. Путь – это последовательность шаблонов (templates), разделенных точкой с запятой. Для каждого шаблона, require будет заменять каждый вопросительный знак на имя файла filename, which is modname with each dot replaced by a "directory separator" (such as "/" in Unix); then it will try to load the resulting file name. Поэтому, например, если Lua путь содержит `" ./ '.lua;./ '.lc;/usr/local/ '/init.lua "` то при поиске загрузки модуля foo будет выполняться поиск файлов ./foo.lua, ./foo.lc и /usr/local/foo/init.lua, именно в этом порядке
package.preload	Таблица, хранящая загрузчики для модулей (см. require)
package.seeall(module)	Устанавливает метатаблицу для модуля module с полем __index , ссылающимся на глобальное окружение, т.е. этот модуль наследует глобальное окружение. Это будет использоваться, как опция для функции module

1.2.4. Работа со строками

Библиотека предоставляет основные функции для работы со строками, такие, как поиск и выделение подстрок, а также поиск по шаблону. Строки в Lua индексируются с 1 (а не 0, как в C). Индексы могут быть отрицательными и интерпретируются как индекс с конца строки. Т.е. последний символ имеет позицию -1 и т.д.

Библиотека работы со строками предоставляет все функции в таблице string. Она также устанавливает поле __index метатаблицы строк на таблицу string. Также Вы можете использовать строковые функции в объектно-ориентированном стиле. Например, string.byte(s, i) может быть записано как s:byte(i).

Таблица 1.4 - Описание функций работы со строками

Название функции	Описание функции
string.byte(s [, i [, j]])	Возвращает числовые коды символов s[i], s[i+1], ⋯, s[j]. По умолчанию значение для i равно 1; по умолчанию значение для j равно i. Напоминаем, что числовые коды символов не одинаковы на разных платформах
string.char(⋯)	Принимает 0 или более целых чисел. Возвращает строку, длина которой равна количеству аргументов. Каждый аргумент преобразуется из кода в символ. Напоминаем, что числовые коды символов не одинаковы на разных платформах
string.dump(function)	Возвращает строку, содержащую двоичное представление данной функции function, так, что если после этого вызвать функцию loadstring на эту строку, мы получим копию функции. function должна быть Lua функцией без внешних локальных переменных (upvalues)
string.find(s, pattern [, init [, plain]])	Ищет первое вхождение шаблона pattern в строку s. Если поиск успешен, то find возвращает индексы s, где найдено совпадение с шаблоном, т.е. где начинается и кончается; иначе возвращает nil. Третий, необязательный числовой параметр init указывает, откуда начинать поиск; по умолчанию он равен 1, также он может быть отрицательным. Значение true в четвертом, необязательном параметре plain выключает возможность поиска по шаблону, в этом случае производится поиск подстроки как есть, т.е. считается, что она не содержит «шаблонных» ("magic") символов. Помните, что если указан параметр plain, то параметр init должен быть указан тоже. Если шаблон содержит захваты (captures) и поиск успешен, то захваченные значения также возвращаются, сразу после двух индексов
string.format(formatstring, ⋯)	Возвращает строку, сформированную на основании строки форматирования formatstring и опциональных аргументов. Строка форматирования должна строиться по тем же правилам, что и у функции printf языка C. Отличие только в том, что опции/ модификаторы *, l, L, n, p и h не поддерживаются. Все символы строки форматирования, кроме управляющих последовательностей, копируются в итоговую строку без изменений. Стандартным признаком начала управляющей последовательности

<table>
<tr><th>Название функции</th><th>Описание функции</th></tr>
<tr><td></td><td>является символ %, для вывода самого знака % используется его удвоение %%.
Управляющая последовательность имеет вид:
%[флаги][ширина][.точность][размер]тип
Обязательными составными частями являются символ начала управляющей последовательности (%) и тип</td></tr>
<tr><td colspan="2">Описание флагов</td></tr>
<tr><td colspan="2">
<table>
<tr><th>Флаг</th><th>Значение</th><th>В отсутствие этого знака</th><th>Примечание</th></tr>
<tr><td>-</td><td>выводимое значение выравнивается по левому краю в пределах минимальной ширины поля</td><td>по правому</td><td></td></tr>
<tr><td>+</td><td>всегда указывать знак (плюс или минус) для выводимого десятичного числового значения</td><td>только для отрицательных чисел</td><td></td></tr>
<tr><td></td><td>помещать перед результатом пробел, если первый символ значения не знак</td><td>Вывод может начинаться с цифры.</td><td>Символ + имеет больший приоритет, чем пробел. Используется только для десятичных числовых значений</td></tr>
<tr><td>#</td><td>«альтернативная форма» вывода значения</td><td></td><td>см. ниже</td></tr>
<tr><td>0</td><td>дополнять поле до ширины, указанной в поле ширина управляющей последовательности, символом 0</td><td>дополнять пробелами</td><td>Используется для типов d, i, o, u, x, X, a, A, e, E, f, F, g, G. Для типов d, i, o, u, x, X, если точность указана, этот флаг игнорируется. Для остальных типов поведение не определено</td></tr>
</table>
</td></tr>
<tr><td colspan="2">Спецификатор ширины</td></tr>
<tr><td colspan="2">Ширина (десятичное число) указывает минимальную ширину поля (включая знак для чисел). Если представление величины больше, чем ширина поля, то запись выходит за пределы поля (например, %2i для величины 100 даст значение поля в три символа), если представление величины менее указанного числа, то оно будет дополнено (по умолчанию) пробелами слева, поведение может меняться предшествующими флагами</td></tr>
<tr><td colspan="2">Спецификатор точности</td></tr>
</table>

Название функции	Описание функции
- указывает на минимальное количество символов, которое должно появиться при обработке типов d, i, o, u, x, X; - указывает на минимальное количество символов, которое должно появиться после десятичной запятой (точки) при обработке типов a, A, e, E, f, F; - максимальное количество значащих символов для типов g и G; - максимальное число символов, которые будут выведены для типа s. Точность задаётся в виде точки с последующим десятичным числом, если число отсутствует (присутствует только точка), то предполагается, что число равно нулю. Точка для указания точности используется даже в том случае, если при выводе чисел с плавающей запятой выводится запятая	
Спецификатор размера	
Поле размер позволяет указать размер данных, переданных функции. Необходимость в этом поле объясняется особенностями передачи произвольного количества параметров в функцию в языке Си: функция не может «самостоятельно» определить тип и размер переданных данных, так что информация о типе параметров и точном их размере должна передаваться явно	
Спецификатор типа	
Тип указывает не только на тип величины (с точки зрения языка программирования Си), но и на конкретное представление выводимой величины (например, числа могут выводить в десятичном или шестнадцатеричном виде). Записывается в виде одного символа. В отличие от остальных полей является обязательным. d, i — десятичное знаковое число. По умолчанию записывается с правым выравниванием, знак пишется только для отрицательных чисел. '%d' и '%i' ведут себя одинаково: - o — восьмеричное беззнаковое число; - u — десятичное беззнаковое число; - x и X — шестнадцатеричное число, x использует маленькие буквы (abcdef), X большие (ABCDEF); - f и F — числа с плавающей запятой. По умолчанию выводятся с точностью 6, если число по модулю меньше единицы, перед десятичной точкой пишется 0. Величины ±∞ представляются в форме [-]inf или [-]infinity. Величина Nan представляется как [-]nan или [-]nan(любой текст далее). Использование F выводит указанные величины заглавными буквами (-INF, NAN); - e и E — числа с плавающей запятой в экспоненциальной форме записи (вида 1.1e+44); e выводит символ «e» в нижнем регистре, E — в верхнем (3.14E+0); - g и G — число с плавающей запятой; форма представления зависит от значения величины (f или e); - a и A — число с плавающей запятой в шестнадцатеричном виде; - c — вывод символа с кодом, соответствующим переданному аргументу; - S и s — вывод строки с нулевым завершающим байтом; - q - позволяет вернуть строку в формате, безопасно воспринимаемом Lua интерпретатором: строка выводится в двойных кавычках, а все двойные кавычки, перевод строки, символы с кодом 0 и обратный слеш внутри строки перекодируются (escaped); - % — символ для вывода знака процента (%), используется для возможности вывода символов процента в строке printf, всегда используется в виде %%. Опции c, d, E, e, f, g, G, i, o, u, X и x должны использоваться только для числовых параметров, а q и s - строковых. Эта функция не принимает строковые параметры, содержащие символы с кодом 0, кроме параметров для формата, имеющего опцию q.	

Название функции	Описание функции

Пример применения:

```
string1 = "Lua"
string2 = "Tutorial"

number1 = 10
number2 = 20

-- Basic string formatting
print(string.format("Basic formatting %s %s",string1,string2))

-- Date formatting
date = 2; month = 1; year = 2014
print(string.format("Date formatting %02d/%02d/%03d", date, month,
      year))

-- Decimal formatting
print(string.format("%.4f",1/3))
```

В результате получим:

```
Basic formatting Lua Tutorial
Date formatting 02/01/2014
0.3333
```

string.gmatch(s, pattern)	Возвращает итератор, который, при каждом вызове, возвращает следующее захваченное значение. Если шаблон pattern не содержит захватов (captures), то простое сравнение будет выполнено при каждом вызове

Например,следующий цикл

```
s = "hello world from Lua"
for w in string.gmatch(s, "%a+") do
    print(w)
end
```

будет проходить по всем словам в строке s, печатая их по одному в строке. Следующий пример собирает все парные ключи (pairs key)=value из указанной строки в таблицу:

```
t = {}
s = "from=world, to=Lua"
for k, v in string.gmatch(s, "(%w+)=(%w+)") do
    t[k] = v
end
```

Название функции	Описание функции
Для данной функции символ '^' в начале шаблона не работает как признак поиска с начала строки, поскольку это помешает итерации	
Формат шаблона	
Шаблон представляет собой совокупность шаблонов элементов. Шаблон элемента может быть представлен: - x — просто символом, совпадающим по шаблону символа (см. ниже); - x* — символом со звёздочкой, что соответствует 0 или более повторений символа. Отыскивается последовательность максимальной длины; - x+ — символом с плюсом, что соответствует 1 или более повторений символа. Отыскивается последовательность максимальной длины; - x- — символом с минусом, что соответствует 0 или более повторений символа. Отыскивается последовательность минимальной длины; - x? — символом со знаком вопроса, что соответствует 0 или 1 повторению символа; - x%n, где n от 1 до 9 указывает строгое соответствие количества повторов указанного символа x	
Захваты	
Захватами называются элементы шаблона, которые указывают, какие части анализируемой строки должны быть извлечены в виде подстрок. Шаблон для захвата указывается в круглых скобках. Особым случаем является пустой захват, представляющий собой пустые круглые скобки (). Данный захват возвращает число - текущую позицию в исходной строке	
Символьный шаблон	
Символьный шаблон используется для представления набора символов: - x — любой символ, представляющий самого себя, не совпадающий с одним из управляющих символов: ^$( )%.[ ]*+-?. ; - . — точка представляет любой символ; - %a — представляет любую букву; - %c — представляет любой управляющий символ; - %d — представляет любую цифру; - %g — представляет любой печатаемый символ, за исключением пробела; - %l — представляет любую букву в нижнем регистре; - %p — представляет любой символ пунктуации; - %s — представляет любой разделительный символ; - %u — представляет любую букву в верхнем регистре; - %w — представляет любую букву или цифру; - %x — представляет символы шестнадцатиричной системы счисления; - %x — (где x - любой не буквенно-цифровой символ) представляет символ x, то есть символ % используется для экранировки управляющих и служебных символов; - [set] — квадратные скобки служат для указания набора символов. Непрерывный диапазон символов может задаваться при помощи первого и последнего символа диапазона с разделителем дефис. Пример [0-7] - все цифры от 0 до 7	
string.gsub(s, pattern, repl [, n])	Возвращает копию s, в которой все вхождения pattern заменяются на repl, который может быть строкой, таблицей или функцией. gsub также возвращает как второе значение – общее количество проведенных подстановок. Если repl строка, то используется ее значение для замены. Символ % работает, как символ со специальным назначением: любая последовательность в repl в виде %n, где n от 1 до 9, заменяется

Название функции	Описание функции
	на n-ную захваченную подстроку (см. ниже). Последовательность %0 заменяется на найденную подстроку. Последовательность %% считается как одиночный символ %. Если repl является таблицей, то она запрашивается для каждого сравнения, с использованием первого захваченного значения как ключ; если шаблон не содержит захватов, то используется результат простого сравнения как ключ. Если repl является функцией, то эта функция вызывается каждый раз, когда обнаруживается совпадение. В качестве параметров ей передаются все захваченные подстроки; если шаблон не содержит захватов (captures), то передается результат сравнения как один параметр. Если значение, возвращаемое таблицей или функцией, является строкой или числом, то это значение используется для замены; в противном случае, если значение равно **false** или **nil**, то замена не производится (т.е. найденное значение остается без замены). Необязательный последний параметр n ограничивает максимальное количество замен. Например, если n равно 1, то будет выполнено не более одной замены

Несколько примеров:

```
x = string.gsub("hello world", "(%w+)", "%1 %1")
    --> x="hello hello world world"

x = string.gsub("hello world", "%w+", "%0 %0", 1)
    --> x="hello hello world"

x = string.gsub("hello world from Lua", "(%w+)%s*(%w+)", "%2 %1")
    --> x="world hello Lua from"

x = string.gsub("home = $HOME, user = $USER", "%$(%w+)", os.getenv)
    --> x="home = /home/roberto, user = roberto"

x = string.gsub("4+5 = $return 4+5$", "%$(.-)%$", function (s)
    return loadstring(s)()
    end)
    --> x="4+5 = 9"

    local t = {name="lua", version="5.1"}
    x = string.gsub("$name-$version.tar.gz", "%$(%w+)", t)
    --> x="lua-5.1.tar.gz"
```

string.len(s)	Возвращает длину строки, переданной в качестве параметра. Пустая строка "" имеет длину 0. Вложенные символы с кодом 0 также считаются как символ, т.е. "a\000bc\000" имеет длину 5

Название функции	Описание функции
string.lower(s)	Возвращает копию строки s, где все большие буквы заменены на маленькие. Все остальные символы остаются неизменными. Понятие «большие буквы» зависит от текущей кодовой страницы
string.match(s, pattern [, init])	Поиск первого вхождения шаблона pattern в строку s. В случае обнаружения, match возвращает захваченные значения (captures); в противном случае возвращает **nil**. Если pattern не содержит захватов, то производится простое сравнение. Третий, необязательный числовой параметр init указывает, с какого символа строки необходимо начинать поиск; по умолчанию этот параметр равен 1. Также он может быть отрицательным
string.rep(s, n)	Возвращает строку, в которой содержится n копий строки s
string.reverse(s)	Возвращает строку, в которой символы строки s расположены в обратном порядке
string.sub(s, i [, j])	Возвращает подстроку строки s, которая начинается с символа с индексом i и продолжается до символа с индексом j; i и j могут быть отрицательными. Если j не указан, то считается, что он равен -1 (то же самое, что длина строки). В частности, вызов string.sub(s,1,j) возвращает начальную часть строки с длиной j, а string.sub(s, -i) возвращает конец строки s длиной i
string.upper(s)	Принимает строку и возвращает копию этой строки, где все маленькие буквы меняются на большие. Все остальные остаются неизменными. Понятие «маленькие буквы» зависит от текущей кодовой страницы

1.2.5. Поддержка UTF-8

Эта библиотека предоставляет базовую поддержку для кодирования UTF-8. Она размещает все свои функции в таблице utf8. Данная библиотека не предоставляет какой-либо поддержки Unicode, кроме обработки кодирования. Любая операция, требующая обозначения символа, вроде классификации символов, находится вне сферы применения данной библиотеки.

Если не указано иное, все функции, ожидающие местоположения байта в качестве параметра, предполагают, что заданная позиция является либо началом последовательности байтов, либо оно же плюс длина представленной строки. Как и в строковой библиотеке, отрицательные индексы отсчитываются от конца строки.

Таблица 1.5 - Описание функций UTF-8

Название функции	Описание функции
utf8.char(...)	Функция получает ноль или более целых чисел, преобразовывает каждый из них в соответствующую UTF-8 последовательность байтов, склеивает их в UTF-8 строку, которую возвращает
utf8.charpattern	Данный образец (т.е. это строка, а не функция) равен "[\0-\x7F\xC2-\xF4][\x80-\xBF]*" (смотрите §6.4.1 [http://www.lua.ru/doc/6.4.1.html]), что соответствует ровно одной байтовой

Название функции	Описание функции
	последовательности UTF-8 при условии, что субъект является допустимой строкой UTF-8.
utf8.codes(s)	Функция возвращает значения, так что конструкция for p, c in utf8.codes(s) do body end будет перебирать все символы в строке s, с p, что является позицией (в байтах) и с - кодовая точка каждого символа. Функция выдает ошибку, если встретит любую недопустимую последовательность байтов
utf8.codepoint(s [, i [, j]])	Возвращает кодовые точки (коды символов) (в виде целых чисел) от всех символов в строке s, которая начинается между байтовой позицией i и j (включая и ту и другую). Значением по умолчанию для i является 1, а для j - i. Функция выдает ошибку, если встретит любую недопустимую последовательность байтов
utf8.len(s [, i [, j]])	Функция возвращает число символов UTF-8 в строке s, которая начинается между позициями i и j (включая и ту и другую). Значением по умолчанию для i является 1, и -1 для j. Если она находит любую недопустимую последовательность байтов, то возвращает значение *false* плюс положение первого неправильного байта
utf8.offset(s, n [, i])	Возвращает позицию (в байтах), с которой начинается кодировка символа строки s под номером n (считая от позиции i). Отрицательный n получает символы до позиции i. По умолчанию i равна 1, когда n не является отрицательным, и #s + 1 в остальных случаях, поэтому utf8.offset(s, -n) получит смещение n-ного символа от конца строки. Если заданный символ не находится внутри строки или сразу после ее конца, функция возвращает **nil**. Как особый случай, когда n равен 0, функцию возвращает начало кодировки символа, который содержит i-ый по счету байт строки s. Данная функция полагается на то, что s является допустимой строкой UTF-8.

1.2.6. Обработка таблиц

Эта библиотека предоставляет основные функции для обработки таблиц. Все функции включены в таблицу table.

Большинство функций в библиотеке таблиц предполагают, что таблица является массивом или списком. Для этих функций, когда мы говорим о параметре "длина" таблицы, мы имеем в виду результат оператора length.

Таблица 1.6 - Описание функций обработки таблиц

Название функции	Описание функции
table.concat(table [, sep [, i [, j]]])	Задан массив, в котором все элементы – строки или числа, возвращает table[i]..sep..table[i+1] ⋯ sep..table[j]. Значение по умолчанию для sep – пустая строка, значение по умолчанию для i – 1, а для j – длина таблицы. Если i больше j, функция возвращает пустую строку

Название функции	Описание функции
table.insert(table, [pos,] value)	Вставляет элемент value в позицию pos в table, сдвигая вверх остальные элементы. Значение по умолчанию для pos равно n+1, где n - это длина таблицы (см.§2.5.5 [http://www.lua.ru/doc/2.5.5.html]), т.о. вызов table.insert(t,x) добавляет x в конец таблицы t
table.maxn(table)	Возвращает наибольший положительный числовой индекс заданной таблицы, или zero, если таблица не имеет положительных числовых индексов. Для выполнения запроса эта функция перебирает все индексы таблицы
table.remove(table [, pos])	Удаляет из table элемент в позиции pos, сдвигая вниз остальные элементы, если это необходимо. Возвращает значение удаленного элемента. Значение по умолчанию для – n, где n – длина таблицы, т.о. вызов table.remove(t) удаляет последний элементы таблицы t. (Примечание: использование insert-remove со значениями по умолчанию позволяет работать с таблицей, как со стандартным LIFO – стеком)
table.sort(table [, comp])	Сортирует элементы таблицы в заданном порядке, внутри таблицы, начиная с table[1] и заканчивая table[n], где n – длина таблицы. Если параметр comp задан, то он должен быть функцией, которая для двух получаемых параметров возвращает true, если первый из них меньше второго (т.о. not comp(a[i+1],a[i]) будет верно для любого I, будет давать true после окончания сортировки). Если comp не задан, то вместо него будет использован стандартный оператор Lua "<". Алгоритм сортировки не стабилен; в том смысле, что равные элементы могут быть переставлены в процессе сортировки

1.2.7. Математические функции

Эта библиотека – интерфейс к стандартной библиотеке math языка C. Она предоставляет все функции внутри таблицы math.

Таблица 1.7 - Описание математических функций

Название функции	Описание функции
math.abs (x)	Возвращает модуль x
math.acos (x)	Возвращает арккосинус x (в радианах)
math.asin (x)	Возвращает арксинус x (в радианах)
math.atan (x)	Возвращает арктангенс x (в радианах)
math.atan2 (x, y)	Возвращает арктангенс x/y (в радианах), но использует знаки обоих параметров для вычисления «четверти» на плоскости (также корректно обрабатывает случай, когда y равен нулю)
math.ceil (x)	Возвращает наименьшее целое число, большее или равное x (округление «вверх»)
math.cos (x)	Возвращает косинус x (угол – в радианах)

Название функции	Описание функции
math.cosh (x)	Возвращает косинус (гиперболический косинус) x
math.deg (x)	Переводит угол, заданный в радианах (x), в градусы
math.exp (x)	Возвращает exp
math.floor (x)	Возвращает наибольшее целое число, меньшее или равное x (округление «вниз»)
math.fmod (x, y)	Возвращает остаток от деления x на y
math.frexp (x)	Возвращает m и e, такие, что x = m2e, e – целое, а модуль m находится в интервале [0.5, 1) (либо ноль, если x равен нулю).Разложение числа с фиксированной запятой
math.huge	Значение HUGE_VAL, значение большее, либо равное любому числовому значению
math.ldexp (m, e)	Возвращает m2e (e должно быть целым). Восстановление значения по мантиссе и показателю
math.log (x)	Возвращает натуральный логарифм x
math.log10 (x)	Возвращает логарифм x по основанию 10
math.max (x,⋯)	Возвращает максимальный из аргументов
math.min (x, ⋯)	Возвращает минимальный из аргументов
math.modf (x)	Возвращает два числа, целую часть x и дробную часть x
math.pi	Значение pi
math.pow (x, y)	Возвращает xy. Вы также можете использовать запись x^y для вычисления значения этой функции
math.rad (x)	Конвертирует угол x, заданный в градусах, в радианы
math.random ([m [, n]])	Эта функция является интерфейсом к простейшему генератору псевдослучайных чисел rand, предоставляемому ANSI C. Нет никаких гарантий по поводу его статистических свойств. При вызове без аргументов возвращает псевдослучайное действительное число в интервале [0,1). При вызове с аргументом m, math.random возвращает псевдослучайное целое число из отрезка [1, m]. При вызове с двумя аргументами – m и n math.random возвращает псевдослучайное целое число из отрезка [m, n]
math.randomseed (x)	Инициализирует генератор псевдослучайных чисел параметром x ("seed"): каждый параметр порождает соответствующую (но одну и ту же) последовательность псевдослучайных чисел
math.sin (x)	Возвращает синус x (аргумент – в радианах)
math.sinh (x)	Возвращает синус (гиперболический синус) x
m7ath.sqrt (x)	Возвращает квадратный корень x. Вы также можете использовать выражение x^0.5 для вычисления этого значения
math.tan (x)	Возвращает тангенс угла x (аргумент – в радианах)
math.tanh (x)	Возвращает гиперболический тангенс x

1.2.8. Битовые операнды

Lua поддерживает следующие побитовые операторы:
- & - побитовое и;
- / - побитовое или;
- ~ - побитовое исключение или;
- > > - сдвиг вправо;
- < < - сдвиг влево;
- ~ - унарный побитовый оператор not.

Все побитовые операции преобразуют его операнды в целые числа (см. далее), работают со всеми битами этих целых чисел и приводят к целому числу. Как правые, так и левые сдвиги заполняют пустые биты нулями. Отрицательные смещения смещаются в другую сторону; смещения с абсолютными значениями, равными или превышающими число битов в целом, приводят к нулю (так как все биты смещены).

Побитовые операторы всегда преобразуют плавающие операнды в целые числа. Преобразование из float в integer проверяет, имеет ли float точное представление в виде целого числа (то есть float имеет целочисленное значение и находится в диапазоне целочисленного представления). Если это так, то это представление является результатом. В противном случае преобразование завершится неудачей.

1.2.9. Средства ввода-вывода

1.2.9.1. Работа с файлами

Библиотека ввода-вывода предоставляет два различных возможных «стиля» для работы с файлами. Первый использует неявные дескрипторы файлов; т.е., существуют операции по установке файла «по умолчанию» ввода или вывода, и все операции ввода-вывода работают с этими файлами. Второй «стиль» использует явные дескрипторы файлов.

При использовании неявных файловых дескрипторов все операции предоставляются таблицей io. При использовании явных дескрипторов операция io.open возвращает дескриптор, а после этого все операции являются методами данного дескриптора.

Таблица io также предоставляет три предопределенных (predefined) файловых дескриптора со стандартными (в C) значениями: io.stdin, io.stdout, и io.stderr.

Если не указано иное, все функции ввода-вывода возвращают **nil** при ошибочном завершении (а также сообщение об ошибке как второй результат и системно-независимый код ошибки в третьем результате) и какое-либо значение, отличное от **nil** при успешном завершении.

Таблица 1.8 - Описание функций работы с файлом

Название функции	Описание функции
io.close([file])	Эквивалентна file:close [#_file:close_()]. Без параметра file закрывает стандартный поток вывода
io.flush()	Эквивалентна file:flush [#_file:flush_()] для стандартного потока вывода
io.input([file])	При вызове с указанием имени файла открывает данный файл (в текстовом режиме) и направляет его поток на стандартный

Название функции	Описание функции
	поток ввода. При вызове с хендлером файла делает хендлер файла стандартным хендлером ввода (перенаправляет поток, соответствующий хендлеру файла на стандартный поток ввода). При вызове функции без параметров возвращает текущий файл ввода по умолчанию. В случае ошибок данная функция возбуждает ошибку вместо того, чтобы возвратить код ошибки
io.lines([filename])	Открывает файл с данным именем в режиме чтения и возвращает функцию-итератор (iterator function), которая при каждом последующем вызове возвращает новую строчку из файла. Т.о., конструкция `for line in io.lines(filename) do body end` обработает все строки файла. При обнаружении функцией-итератором конца файла она возвращает **nil** (для окончания цикла) и автоматически закрывает файл. Вызов io.lines() (без имени файла) эквивалентен io.input():lines(); т.о., он обрабатывает строки стандартного файла ввода. В этом случае, файл по окончанию итераций не закрывается автоматически
io.open(filename [, mode])	Эта функция открывает файл в режиме, указанном в строке mode. Возвращает объект файла, или, в случае ошибок, **nil** и сообщение об ошибке
Строка mode может содержать следующие значения:	
- **"r":** режим чтения (используется по умолчанию); - **"w":** режим записи; - **"a":** режим дозаписи в конец файла; - **"r+":** режим изменения, все ранее хранившиеся данные сохраняются; - **"w+":** режим изменения, все ранее хранившиеся данные стираются; - **"a+":** режим изменения с дозаписью в конец, все ранее хранившиеся данные защищены, запись разрешена только в конец файла. Строка mode может также содержать 'b' в конце; этот символ нужен для некоторых систем для открытия файла в двоичном режиме. Эта строка полностью повторяет синтаксис функции C fopen	
io.output([file])	Аналогична io.input, но работает со стандартным файлом вывода
io.popen(prog [, mode])	Запускает программу prog в отдельном процессе и возвращает хендлер файла, который вы можете использовать для чтения данных из этой программы (если mode равен "r", значение по умолчанию) или для записи данных в эту программу (если mode равен "w"). Эта функция системно зависима и доступна не на всех платформах
io.read(⋯)	Аналогична io.input():read

Название функции	Описание функции
io.tmpfile ()	Возвращаер хендлер для временного файла. Этот файл открывается в режиме изменения и автоматически удаляется при завершении программы
io.type(obj)	Проверяет, является ли obj объектом файла. Возвращает строку "file", если obj – открытый объект файла, "closed file", если obj - закрытый объект файла, или nil, если obj не является объектом файла
io.write(⋯)	Эквивалентна io.output():write
file:close()	Закрывает file. Обратите внимание, что файлы автоматически закрываются, когда их объекты уничтожаются сборщиком мусора, но это может случиться в любой момент и не предсказуемо
file:flush()	Сохраняет все данные, записанные в файл file
file:lines()	Возвращает функцию-итератор, которая при каждом вызове возвращает новую строку из файла. Т.о., код `for line in file:lines() do body end` обработает все строки файла. В отличие от io.lines [#_file:lines_()], эта функция не закрывает файл по окончании цикла (т.е. достижении конца файла)
file:read (⋯)	Читает данные из файла file, в соответствии с заданными форматами, которые определяют, что читать. Для каждого формата функция возвращает строку (или число) с прочитанными символами, или **nil**, если не может прочитать данные в указанном формате. При вызове без указания формата использует стандартный формат, соответствующий чтению всей следующей строки (см. ниже)
Возможные форматы:	
- **"*n":** читает число; это – единственный формат, возвращающий число вместо строки; - **"*a":** читает весь файл, начиная с текущей позиции. Если позиция совпадает с концом файла, возвращает пустую строку; - **"*l":** читает следующую строку (пропуская конец строки), возвращает **nil** в конце файла. Это – формат по умолчанию; - **число:** читает строку, но не более заданного количества символов, возвращает **nil** по достижении конца файла. Если число равно нулю, функция ничего не читает, возвращая пустую строку, или **nil** по достижении конца файла	
file:seek([whence] [, offset])	Получает и ыыставляет текущую позицию в файле, отсчитываемую от начала файла, в позицию, заданную параметром offset плюс значение (исходная позиция), заданное строкой whence, следующим образом: - **"set":** исходная позиция равна 0 (начало файла); - **"cur":** исходная позиция – текущая; - **"end":** исходная позиция – конец файла. В случае успешного выполнения функция seek возвращает выставленную позицию в файле, отсчитываемую от начала файла.

Название функции	Описание функции
	Если функция завершается неудачно, она возвращает **nil** и строку – описание ошибки. Значение по умолчанию для параметра whence равно "cur", а offset – 0. Т.о., вызов file:seek() возвращает текущую позицию в файле, не изменяя ее; вызов file:seek("set") перемещает указатель текущей позиции в начало файла (и возвращает 0); а вызов file:seek("end") перемещает указатель текущей позиции в конец файла и возвращает его длину
file:setvbuf(mode [, size])	Изменяет режим файла записи (output file) на буферизованный режим. Существует 3 возможных режима: - **"no":** отключить буферизацию; результат записи в файл немедленно «сбрасывается» на диск; - **"full":** полная буферизация; операции записи на диск выполняются только при переполнении буфера (или при вызове функции очистки буфера (см. io.flush); - **"line":** построчная буферизация; запись буферизуется до достижения конца строки или происходит ввод из особенных (специальных) файлов (таких как, например, терминал). В двух последних случаях size указывает размер буфера в байтах. По умолчанию используется «необходимый и достаточный размер»
file:write (⋯)	Записывает значение каждого из аргументов в файл file. Аргументами могут быть строки или числа. Для записи других значений используйте функции tostring или string.format перед вызовом функции write

```
-- Открываем файл на перезапись.
local f = io.open("c:/test.txt", "w+")
-- Проверяем, что открылся.
if f~=nil then
    for i=1,5 do
        -- Формируем строку с данными.
        local line = os.date() .. "\t"..tostring(i).."\n"
        -- Пишем строку в файл.
        f:write(line)
    end
    -- Принудительно сохраняем данные из файлового кэша на диск.
    f:flush()
    -- Закрываем файл.
    f=nil
end
```

1.2.10. Функции операционной системы

Эта библиотека включена в таблицу os.

ВНИМАНИЕ! Для корректной работы данных функций в операционных системах семейства Линукс приложению, которое их выполняет (приложение LuaEngine), необходимы различные суперпользовательские права. Для добавления данных прав в дистрибутиве СКАДА "Соната" для Линукс есть специальный скрипт. Информацию о данном скрипте и его применении смотрите в Руководстве системного программиста КУНИ.505200.023-01.01 32 п. 3.7.9.2 Необходимые настройки для работы дистрибутива SCADA "СОНАТА" под пользователем с ограниченными правами, если нельзя применять предыдущий вариант (нельзя дилегировать суперпользовательские права на файлы дистрибутива).

Таблица 1.9 - Описание функций операционной системы

Название функции	**Описание функции**
os.clock()	Возвращает строго монотонное время в секундах с момента старта программы. Секунды могут иметь дробную часть
os.date ([format [, time]])	Возвращает строку или таблицу, содержащую дату и время, отформатированные в соответствии с заданным параметром format
Описание параметров	
- Аргумент time является опциональным. Если он не задан, то будет использовано текущая дата и время; - Если параметр format начинается с '!', то время форматируется в соответствии с универсальным глобальным временем (по Гринвичу). После этого опционального символа, если format равен "*t", то date возвращает таблицу со следующими полями: - year (год, четыре цифры); - month (месяц, 1 – 12); - day (день, 1 – 31); - hour (час, 0 – 23); - min (минуты, 0 – 59); - sec (секунды, 0 – 61); - usec (микросекунды, 0 – 999999); - wday (день недели, воскресенью соответствует 0); - yday (день года). - При вызове без аргументов, date возвращает данные «в обычном формате», который зависит от системы и текущих настроек операционной системы (т.е., os.date() эквивалентна os.date("%c")). Служебные токены формата: - %a - аббревиатура названия дня недели в зависимости от текущей локали; - %A - полное название дня недели в зависимости от текущей локали; - %b - аббревиатура названия месяца в зависимости от текущей локали; - %B - полное название месяца в зависимости от текущей локали; - %c - предпочтительный формат даты и времени для текущей локали; - %d - день месяца в десятичной форме (от 01 до 31); - %G - указывает год как четырехзначное число (по стандарту ISO 8601). Имеет тот же формат и значение, что и %y, но если неделя входит также в прошедший (или последующий) год (в соответствии со стандартом ISO о номерах недель), то в этом случае отображается прошедший год (TZ); - %g - то же, что и %G, но без первых двух чисел, то есть двухразрядное число, 00-99 (TZ); - %H - показывает час как десятичное число от 00 до 23;	

Название функции	Описание функции
- %I - показывает час как десятичное число от 01 до 12; - %j - показывает день года как десятичное число от 001 до 366; - %k - показывает час как десятичное число от 0 до 23; начальные нули заменяются пробелами (см. также %H,(TZ); - %l- показывает час, как десятичное число (от 1 до 12); начальные нули заменяются пробелами (см. также %I (TZ); - %m - показывает месяц как десятичное число от 01 до 12; - %M - показывает минуты как десятичное число от 00 до 59; - %p - показывает `AM' или `PM', в зависимости от времени суток или от соответствующей локали. Полдень обозначен как `pm', а полночь - как 'am'.; - %s - показывает количество секунд с начала 1970-01-01 00:00:00 UTC (TZ); - %S - отображает секунды в десятичной форме от 00 до 61; - %w - показывает день недели как десятичное число от 0 до 6, и воскресенье считается равным нулю. См. также %u; - %x - показывает дату в формате, указанном в текущей локали, без времени - %X - показывает время в формате, указанном в текущей локали, без даты; - %y -показывает год как двухразрядное число от 00 до 99, без указания века; - %Y показывает год как четырехразрядное десятичное число (с указанием века); - %z показывает часовой пояс как смещение от GMT (Гринвича). Требует совместимости с RFC822 (форматы: "%a, %d %b %Y %H:%M:%S %z"; - %Z - показывает часовой пояс, или его название, или аббревиатуру; - %+ - показывает дату и время в формате date [http://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=date&category=1] (1)(TZ); - %% - символ '%'; - %N или %9N - вывод наносекуд дробной части секунд с ведущими нулями; - %1N - вывод 1/10 долей секунды; - %3N - вывод милисекунд дробной части секунд с ведущими нулями; - %6N - вывод микросекунд дробной части секунд с ведущими нулями	
os.difftime(t2, t1)	Возвращает число секунд, прошедшее от времени t1 до времени t2. В POSIX, Windows и некоторых других системах это значение равно в точности t2-t1
os.ping(IP [, timeout_s])	Осуществляет проверку связи (ping) с указанным IP адресом. Возвращает два результата: boolean, string. Первый результат показывает есть ли связь или нет. Второй строковый результат описывает, в случае отсутствия связи, возможную проблему.
os.execute([command])	Эта функция эквивалентна функции C system. Она передает параметр command на выполнение шеллу операционной системы. Она возвращает системно-зависимый статус. Если параметр command не передается, то возвращает не ноль, если шелл доступен и ноль в противном случае
os.exit([code])	Вызывает функцию C exit, с опциональным параметром code, для останова выполнения программы-хозяина. Значение по умолчанию для параметра code – код успешного завершения. Примечание: в ISO C описаны коды 0, EXIT_SUCCESS (обычно равен 0, но не стандартизирован, может обозначать «успешное завершение,

Название функции	Описание функции
	отличное от состояния 0»), EXIT_FAILURE (обычно какое-либо ненулевое значение, но не стандартизирован)
os.getenv(varname)	Возвращает значение переменной окружения varname, или **nil**, если переменная не определена
os.now()	Возвращает текущее время UTC
os.remove(filename)	Удаляет файл или директорию с заданным именем. Директории должны быть пусты. Если функция не может провести удаления, она возвращает **nil** плюс строку, содержащую описание ошибки
os.rename(oldname, newname)	Переименовывает файл или директорию oldname в newname. Если функция не может провести переименования, она возвращает **nil** плюс строку, содержащую описание ошибки
os.setlocale(locale [, category])	Изменяет текущие региональные настройки (locale) программы. Параметр locale является строкой, содержащей региональные настройки (locale); category – опциональный строчный параметр, содержащий категорию, для которой производится изменение: "all", "collate", "ctype", "monetary", "numeric", или "time"; значением по умолчанию является категория "all". Функция возвращает значение новых настроек (locale), или **nil**, если вызов не может быть обработан. Если locale – пустая строка, региональные настройки изменяются на региональные настройки, определяемые реализацией (implementation-defined native locale). Если locale – строка "C", текущие региональные настройки изменяются на стандартные C-настройки. При вызове с **nil** вместо первого аргумента эта функция возвращает текущие региональные настройки для заданной категории
os.settime(time)	Задаёт системе новое время. В качестве отсчёта времени используется UTC
os.sleep(delay)	Приостанавливает исполнение программы на указанный в секундах промежуток времени. Секунды могут иметь дробную часть.Если нужно несколько флагов, то просто складываются
os.spawn(fileName, [args], [sync=false])	Функция производит запуск приложения fileName. В качестве аргументов запуска приложения ему могут быть переданы опциональные аргументы строкой args. Третьим аргументом функции является способ запуска приложения: синхронный или асинхронный. В случае успеха функция возвращает 0
os.time([table])	Возвращает текущее время и дату при вызове без аргументов, или время и дату, указанные в передаваемой таблице. Эта таблица должна иметь поля year, month и day, и может иметь поля hour, min, sec и isdst (описание этих полей см. в описании функции os.date). Возвращаемое значение – это число, значение которого зависит от системы. В POSIX, Windows и некоторых других системах это число соответствует количеству секунд, отсчитываемому от некоторого заданного момента времени ("эпоха"). В других системах значение не специфицировано, и число, возвращаемое функцией time, может быть использовано только как аргумент функций date и difftime

Название функции	Описание функции

	`-- Создаём пустую таблицу.`
	`local t = { }`
	`-- Заполняем поля таблицы.`
	`t.year = 2015`
	`t.month = 7`
	`t.day = 22`
	`t.hour = 16`
	`t.min = 35`
	`t.sec = 23`
	`t.usec = 123456`
	`-- Преобразуем таблицу в posix время.`
	`local start = os.time(t)`
	`-- Засекаем текущее время.`
	`local finish = os.time)`
	`-- Находим разницу в секундах.`
	`local delta = finish – t`

os.today()	Возвращает текущую дату UTC
os.tmpname ()	Возвращает строку с именем файла, который может быть использован в качестве временного файла. Файл должен быть явно открыт до использования и явно удален, если больше не будет нужен
os.tz()	Возвращает текущее смещение локального часового пояса в секундах от UTC
os.getComputerName()	Возвращает строку с именем компьютера в сети. local str = os.getComputerName()
os.getOsInfo()	Возвращает строку с названием и версией операционной системы. local str = os.getOsInfo()
os.getBoardInfo()	Возвращает строку с описанием материнской платы компьютера. local str = os.getBoardInfo()
os.getCpuInfo()	Возвращает строку с описанием типа процессора компьютера. local str = os.getCpuInfo()
os.getNetInfo()	Возвращает строку с описанием настроек сетевых интерфейсов и MAC адресов. local str = os.getNetInfo()
os.getDiskInfo()	Возвращает строку с описанием типов установленных дисков компьютера. local str = os.getDiskInfo()
os.getCpuLoad()	Возвращает число - суммарную загрузку всех ядер процессора в процентах за последние 3 секунды. local n = os.getCpuLoad()

Название функции	Описание функции
os.getMemSpace()	Возвращает структуру(таблицу) с полями occupied и total типа, в которых содержится занятое и общее количество оперативной памяти компьютера, выраженное в мегабайтах. local t = os.getMemSpace()
os.getDiskSpace()	Возвращает массив со структурами (таблицами), у которых по три поля: name, occupied и total. Поле name содержит имя диска, поле occupied - количество занятого на диске места в Мб, total - объем диска в Мб. local t = os.getDiskSpace()

1.3. Расширения Lua при работе в составе SCADA "Соната"

Стандартный интерпретатор языка Lua 5.3 при работе в составе SCADA "Соната" получает новые функции и расширяет некоторые существующие.

1.3.1. Стандартные функциональные блоки IEC-61131

Работа функциональных блоков реализована в виде замыканий - функций с внутренним состоянием. В Lua интерпретаторе задекларированы классы (таблицы) с именами типов ФБ IEC-61131. У каждого такого класса есть два эквивалентных метода create() и new(), которые возвращают экземпляр класса данного ФБ. Для исполнения (пересчёта состояния) экземпляра функционального блока его нужно вызывать как функцию. В качестве аргументов должны выступать все входные сигналы ФБ, результатом вызова будут являться все выходные сигналы ФБ.

Рассмотрим пример создания и использования функционального блока TON.

```
       +-------+
       |  TON  |
BOOL---|IN    Q|---BOOL
TIME---|PT   ET|---TIME
       +-------+
```

```
local ton = TON.create() -- Создаём экземпляр ФБ типа TON.
local in = true
local pt = 10
local q, et = ton(in, pt) -- Вызов пересчёта блока.
```

Блок SR — триггер доминирующего включения

```
              +------+
              |  SR  |
        BOOL---|S1  Q1|---BOOL
        BOOL---|R     |
              +------+
```

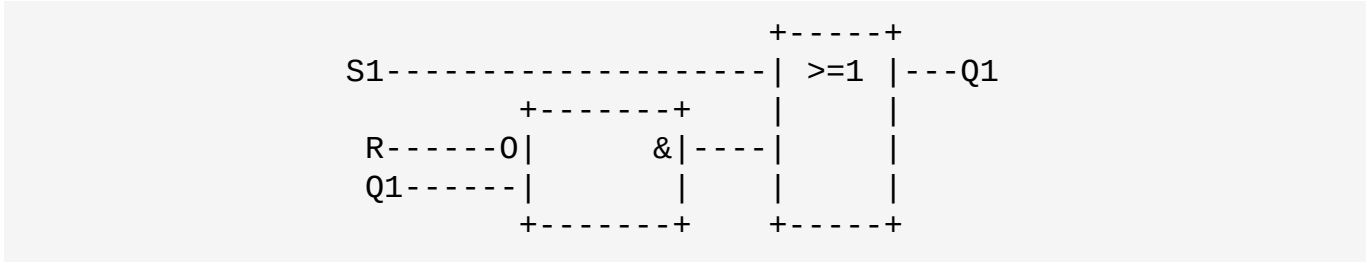
Описание работы.

Выход Q1 = TRUE, когда вход S1 = TRUE или когда предыдущее значение Q1 = TRUE и вход R = FALSE. Во всех остальных случаях выход Q1 = FALSE.

Реализация алгоритма блока на языке ST.

```
FUNCTION_BLOCK SR
   VAR_INPUT
      S1: BOOL;
      R: BOOL;
   END_VAR
   VAR_OUTPUT
      Q1: BOOL;
   END_VAR
   Q1 = (NOT R AND Q1) OR S1;
END_FUNCTION_BLOCK
```

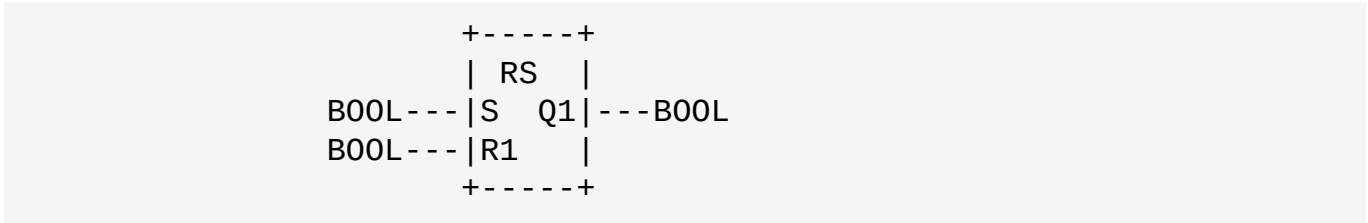
Реализация на языке FBD.



Пример использования.

```
local sr = SR.create() -- Создаём экземпляр ФБ типа SR.
local s1 = true
local r = false
local q1 = sr(s1, r) -- Вызов пересчёта блока.
```

Блок RS - триггер доминирующее выключение



Описание работы.

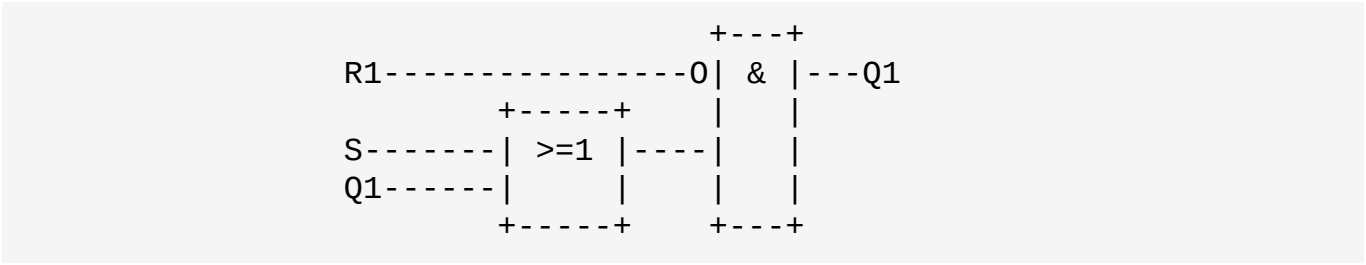
Выход Q1 = FALSE блока SR, пока вход R1 = TRUE. При R1 = FALSE выход Q1 = TRUE, если S1 = TRUE или предыдущее значение Q1 = TRUE.

Реализация алгоритма блока на языке ST.

```
FUNCTION_BLOCK RS
   VAR_INPUT
      S: BOOL;
      R1: BOOL;
   END_VAR
   VAR_OUTPUT
      Q1: BOOL;
   END_VAR
   Q1 = NOT R AND (Q1 OR S1)
```

```
END_FUNCTION_BLOCK
```

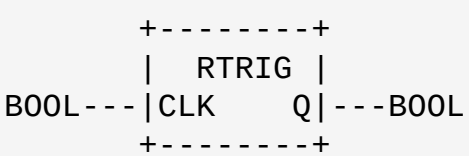
Реализация на языке FBD.



Пример использования.

```
local rs = RS.create() -- Создаём экземпляр ФБ типа RS.
local s = true
local r1 = false
local q1 = rs(s, r1) -- Вызов пересчёта блока.
```

Блок RTRIG - определителя переднего фронта.



Описание работы.

Вырабатывает импульс (поддержание состояния TRUE только в пределах одного вызова ФБ) на выходе Q всякий раз, когда происходит смена значения входа CLK с FALSE на TRUE.

Реализация алгоритма блока на языке ST.

```
FUNCTION_BLOCK RTRIG
    VAR_INPUT CLK: BOOL; END_VAR
    VAR_OUTPUT Q: BOOL; END_VAR
    VAR M: BOOL; END_VAR
        Q := CLK AND NOT M;
        M := CLK;
END_FUNCTION_BLOCK
```

Пример использования.

```
local rtrig = RTRIG.create() -- Создаём экземпляр ФБ типа RTRIG.
local clk = true
local q = rtrig(clk) -- Вызов пересчёта блока.
```

Блок FTRIG - определителя заднего фронта

Описание работы.

Вырабатывает импульс (поддержание состояния TRUE только в пределах одного вызова ФБ) на выходе Q всякий раз, когда происходит смена значения входа CLK с TRUE на FALSE.

Реализация алгоритма блока на языке ST.

```
FUNCTION_BLOCK R_TRIG
    VAR_INPUT CLK: BOOL; END_VAR
    VAR_OUTPUT Q: BOOL; END_VAR
    VAR M: BOOL; END_VAR
        Q := NOT CLK AND NOT M;
        M := NOT CLK;
END_FUNCTION_BLOCK
```

Пример использования.

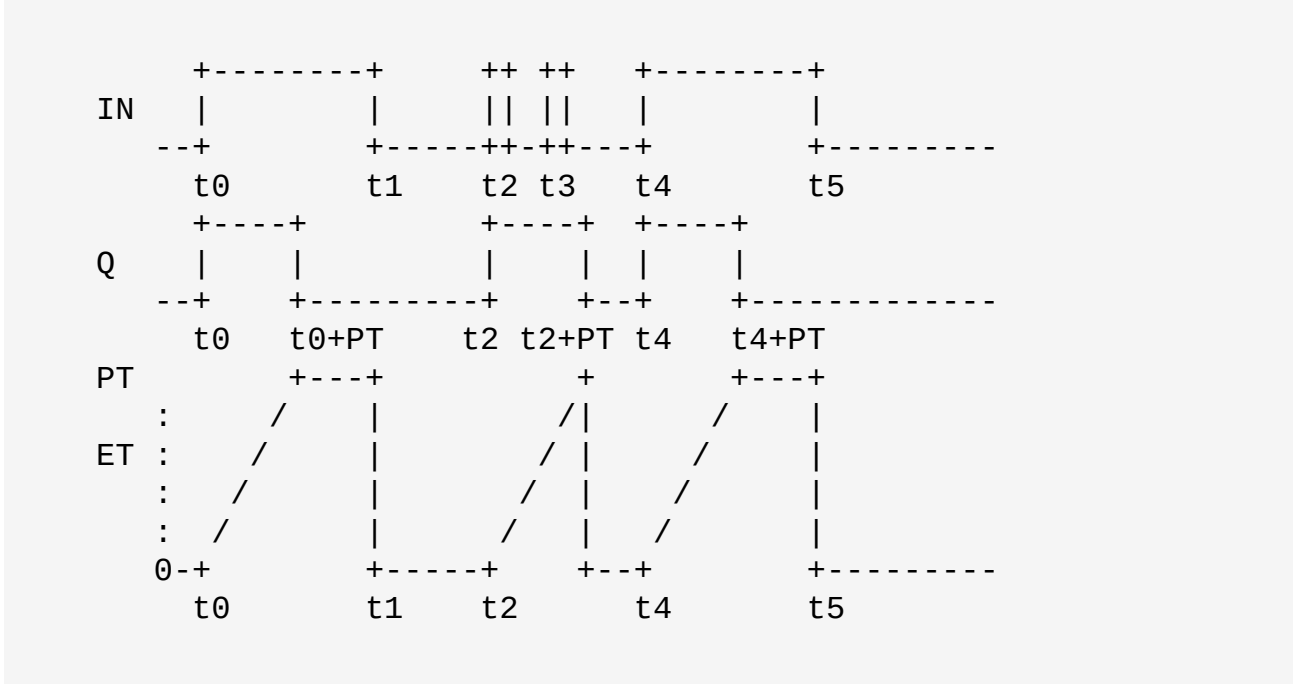
```
local ftrig = FTRIG.create() -- Создаём экземпляр ФБ типа FTRIG.
local clk = true
local q = ftrig(clk) -- Вызов пересчёта блока.
```

Блок TP - таймера

Описание работы.

Пока вход IN = FALSE, выход Q = FALSE, выход ET = 0. При переходе IN в TRUE, выход Q = TRUE, и таймер начинает отсчет времени на выходе ET до достижения длительности заданной PT. Далее счетчик не увеличивается. Таким образом, выход Q генерирует импульс длительностью PT по фронту входа IN.

Диаграмма работы.



Пример использования.

```
local tp = TP.create() -- Создаём экземпляр ФБ типа TP.
```

```
local in = true
local pt = 10
local q, et = tp(in, pt) -- Вызов пересчёта блока.
```

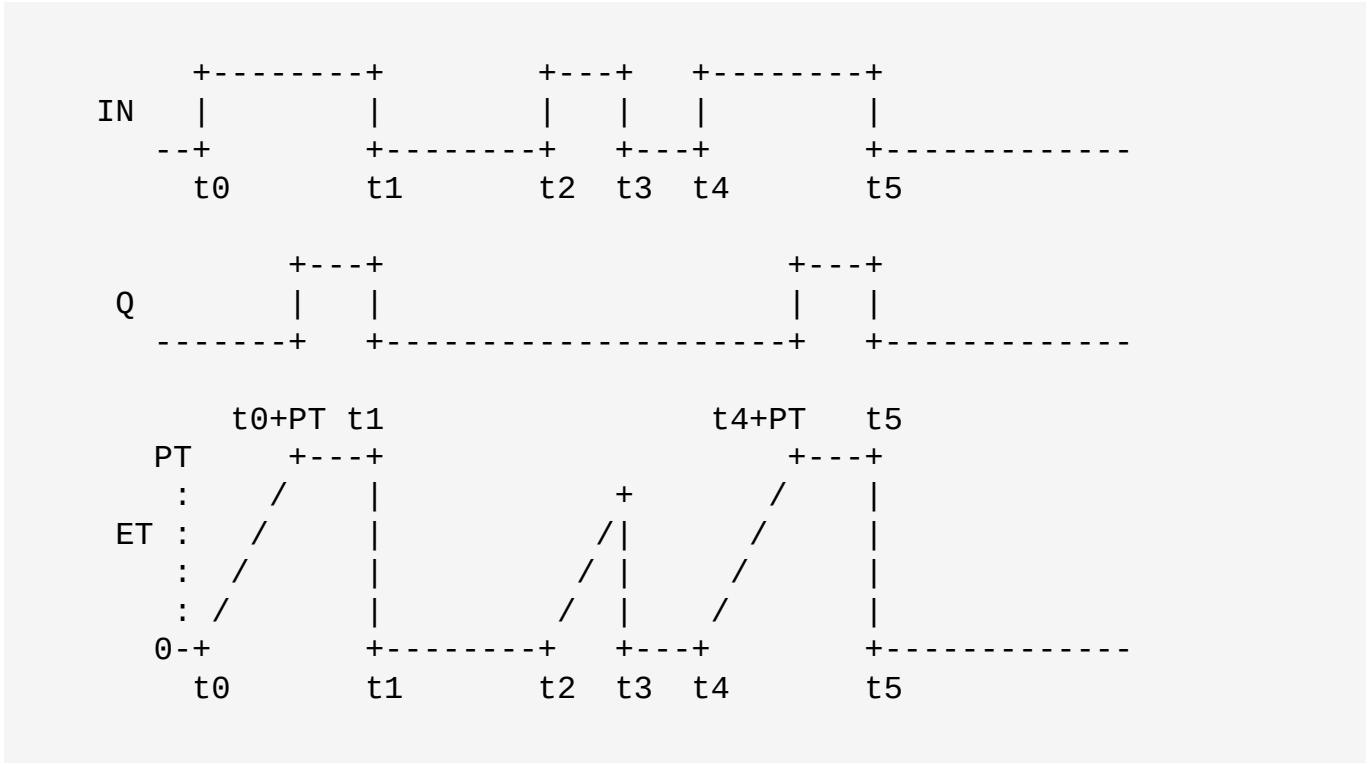
Блок TON - таймера с задержкой включения

```
       +-------+
       |  TON  |
   BOOL---|IN    Q|---BOOL
   TIME---|PT   ET|---TIME
       +-------+
```

Описание работы.

Пока вход IN = FALSE, выход Q = FALSE, выход ET = 0. Как только IN становится TRUE, начинается отсчет времени на выходе ET до значения, равного PT. Далее счетчик не увеличивается. Q = TRUE, когда IN = TRUE и ET = PT, иначе FALSE. Таким образом, выход Q устанавливается с задержкой PT от фронта входа IN.

Диаграмма работы.



Пример использования.

```
local ton = TON.create() -- Создаём экземпляр ФБтипа TON.
local in = true
local pt = 10
local q, et = ton(in, pt) -- Вызов пересчёта блока.
```

Блок TOF - таймера с задержкой выключения

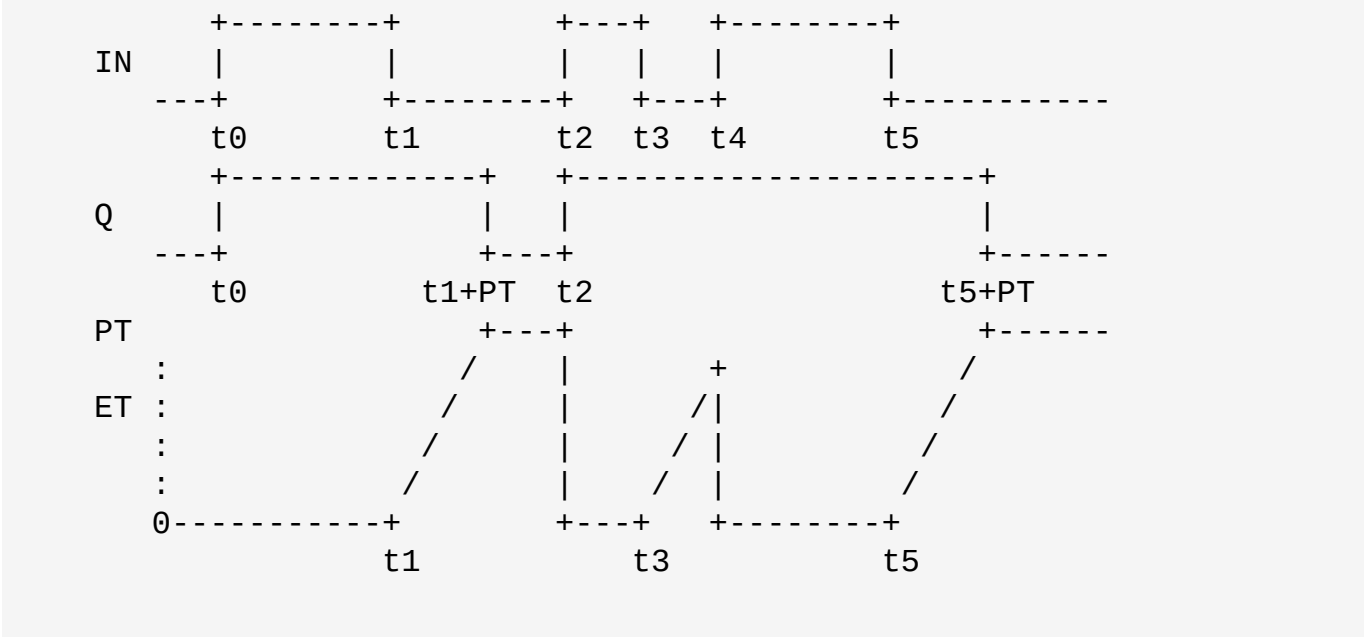
```
       +-------+
       |  TOF  |
   BOOL---|IN    Q|---BOOL
   TIME---|PT   ET|---TIME
```

```
          +-------+
```

Описание работы.

Если вход IN = TRUE, то выход Q = TRUE и выход ET = 0. Как только IN переходит в FALSE, начинается отсчет времени на выходе ET. При достижении заданной длительности отсчет останавливается. Выход Q = FALSE, если IN = FALSE и ET = PT, иначе - TRUE. Таким образом, выход Q сбрасывается с задержкой PT от спада входа IN.

Диаграмма работы.



Пример использования.

```
local tof = TOF.create() -- Создаём экземпляр ФБтипа
local in = true
local pt = 10>
local q, et = tof(in, pt) -- Вызов пересчёта блока.
```

Блок LAG1 - НЧ-фильтр первого порядка.

```
          +-----------+
          |    LAG1   |
    BOOL---|RUN        |
    REAL---|XIN    XOUT|---REAL
    TIME---|TAU        |
    TIME---|CYCLE      |
          +-----------+
```

Описание работы.

Осуществляет низкочастотную фильтрацию входного сигнала XIN при помощи НЧ-фильтра первого порядка.

Реализация алгоритма блока на языке ST.

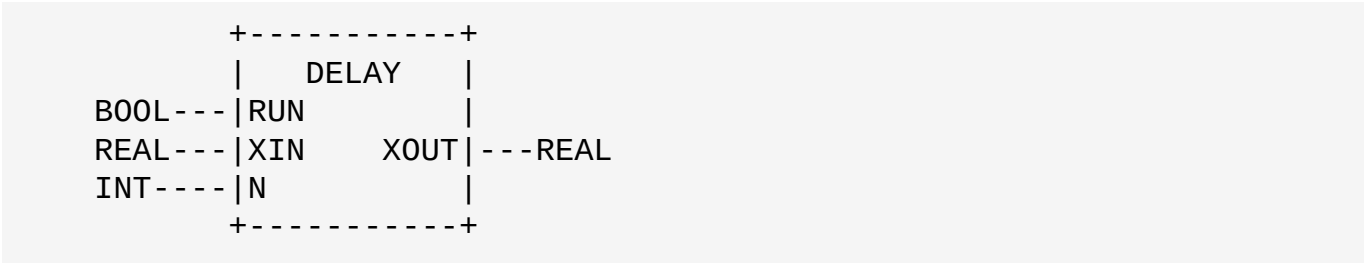
```
FUNCTION_BLOCK LAG1
   VAR_INPUT
      RUN : BOOL ; (* 1 = run, 0 = reset *)
```

```
      XIN : REAL ; (* Input variable *)
      TAU : TIME ; (* Filter time constant *)
      CYCLE : TIME ; (* Sampling time interval *)
   END_VAR
   VAR_OUTPUT XOUT : REAL ; END_VAR (* Filtered output *)
      VAR K : REAL ; (* Smoothing constant, 0.0<=K<1.0 *)
   END_VAR
   IF RUN THEN XOUT := XOUT + K * (XIN - XOUT) ;
   ELSE XOUT := XIN ;
      K := TIME_TO_REAL(CYCLE) / TIME_TO_REAL(CYCLE + TAU) ;
   END_IF ;
END_FUNCTION_BLOCK
```

Пример использования.

```
local lag1 = LAG1.create() -- Создаём экземпляр ФБтипа LAG1.
local run = true
local xin = 10
local tau = 0.1
local cycle = 1
local xout = lag1(run, xin, tau, cycle) -- Вызов пересчёта блока.
```

Блок DELAY – задержка входного сигнала на N отсчётов



Описание работы.

При истинном входном сигнале RUN осуществляет передачу входного сигнала XIN на выход XOUT с заданной задержкой N, заданной в количестве выборок. В противном случае, осуществляет передачу сигнала без задержки.

Реализация алгоритма блока на языке ST.

```
FUNCTION_BLOCK DELAY (* N-sample delay *)
    VAR_INPUT
        RUN : BOOL ; (* 1 = run, 0 = reset *)
        XIN : REAL ;
        N : INT (* 0 <= N < 128or manufacturer- *)
    END_VAR (* specified maximum value *)
    VAR_OUTPUT XOUT : REAL; END_VAR (* Delayed output *)
        VAR X : ARRAY [0..1023] (* N-Element queue *)
        OF REAL; (* with FIFO discipline *)
        I, IXIN, IXOUT : INT := 0;
    END_VAR
    IF RUN THEN IXIN := MOD(IXIN + 1, 1024) ; X[IXIN] := XIN ;
        IXOUT := MOD(IXOUT + 1, 1024) ; XOUT := X[IXOUT];
    ELSE XOUT := XIN ; IXIN := N ; IXOUT := 0;
        FOR I := 0 TO N DO X[I] := XIN; END_FOR;
```

```
    END_IF ;
END_FUNCTION_BLOCK
```

Пример использования.

```
local delay = DELAY.create() -- Создаём экземпляр ФБтипа DELAY.
local run = true
local xin = 10
local n = 20
local xout = delay(run, xin, n) -- Вызов пересчёта блока.
```

Блок HYSTERESIS – гистерезис

```
       +------------+
       | HYSTERESIS |
   REAL---|XIN1       Q|---BOOL
   REAL---|XIN2        |
   REAL---|EPS         |
       +------------+
```

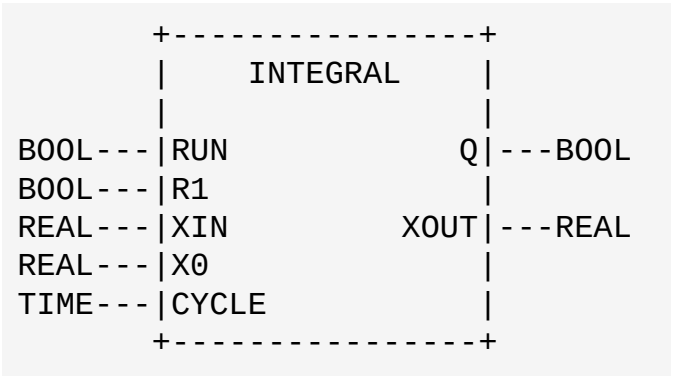
Описание работы.

Гистерезис входных сигналов XIN1, XIN2 с учётом погрешности EPS.

Реализация алгоритма блока на языке ST.

```
FUNCTION_BLOCK HYSTERESIS
(* Boolean hysteresis on difference *)
(* of REAL inputs, XIN1 - XIN2 *)
    VAR_INPUT XIN1, XIN2, EPS : REAL; END_VAR
    VAR_OUTPUT Q : BOOL := 0; END_VAR
    IF Q THEN IF XIN1 < (XIN2 - EPS) THEN Q := 0; END_IF ;
    ELSIF XIN1 > (XIN2 + EPS) THEN Q := 1 ;
    END_IF ;
END_FUNCTION_BLOCK
```

Блок INTEGRAL - интегрирование методом прямоугольников

Графическое представление	Описание
	**RUN** = TRUE - интегрировать; **RUN**= FALSE – остановить интегрирование; **R1** = TRUE - сброс; **XIN** – интегрируемая величина; **X0** -начальное значение интеграла; **CYCLE** – интервал (в секундах) между последовательными вызовами данного функционального блока; **Q** = NOT R1; **XOUT** - выходная переменная (интеграл от входной переменной); **Примечание**: при перезапуске начальное значение интеграла восстанавливается и равно X0

Реализация алгоритма блока на языке ST

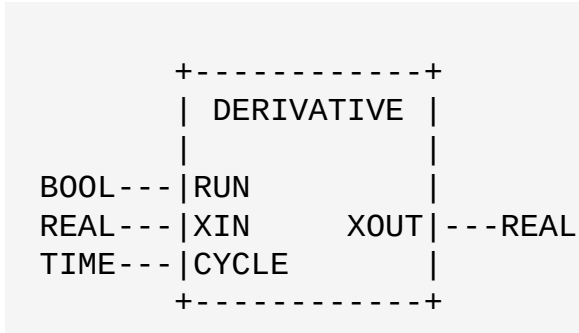
```
FUNCTIONAL_BLOCK INTEGRAL

    VAR INPUT
        RUN : BOOL;     (* 1 = inegrate, 0 = hold *)
        R1 : BOOL;      (* Overriding reset *)
        XIN : REAL;     (* Input variable *)
        X0 : REAL;      (* Initial value *)
        CYCLE : TIME;   (* Sampling period *)
    END_VAR
    VAR_OUTPUT
        Q : BOOL;       (* NOT R1 *)
        XOUT: REAL;     (* Integrated output *)
    END_VAR

    Q := NOT R1;
    IF R1 THEN XOUT := X0;
    ELSIF RUN THEN XOUT := XOUT + XIN * TIME_TO_REAL(CYCLE);
    END_IF;

END_FUNCTION_BLOCK
```

Блок DERIVATIVE - дифференцирование по времени

Графическое представление	Описание
	**RUN** = TRUE - фильтровать; **RUN** = FALSE - остановиться; **R1** = TRUE - сброс; **XIN** – дифференцируемая величина; **CYCLE** – интервал (в секундах) между последовательными вызовами данного функционального блока; **XOUT** - выходная переменная (производная по времени от входной переменной XIN)

Реализация алгоритма блока на языке ST

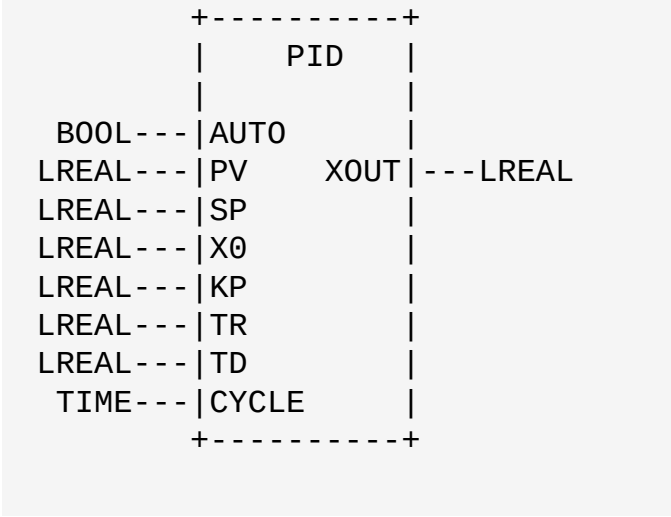
```
FUNCTIONAL_BLOCK DERIVATIVE

    VAR INPUT
        RUN : BOOL;     (* 0 = reset *)
        XIN : REAL;     (* Input to be differentiated *)
        CYCLE : TIME;   (* Sampling period *)
    END_VAR
    VAR_OUTPUT
        XOUT: REAL;     (* Differentiated output *)
    END_VAR
```

```
    VAR X1, X2, X3: REAL; END_VAR
    IF RUN THEN
        XOUT := (3.0 * (XIN - X3) + X1 - X2)
                / (10.0 * TIME_TO_REAL(CYCLE));
        X3 := X2; X2 := X1; X1 := XIN;
    ELSE XOUT := 0.0; X1 := XIN; X2 := XIN; X3 := XIN;
    END_IF;

END_FUNCTION_BLOCK
```

Функциональный блок PID-закона регулирования

Представление	**Описание и реализация**
	**Входные переменные:** **AUTO** – AUTO=TRUE – управление включено; **PV –** выходное значение сигнала объекта управления; **SP** – требуемое значение выхода объекта управления. ПИД регулятор вычисляет рассогласование (ошибку) между желаемым значением (SP) и действительным значением (PV) сигнала с объекта управления. По этой ошибке и происходит управление; **X0** - смещение выхода регулятора; **KP**- коэффициент пропорциональной составляющей; **TR** – постоянная интегрирования; **TD** – постоянная времени дифференцирования; **CYCLE** - интервал (в секундах) между последовательными вызовами данного функционального блока; **XOUT** - выходная переменная

Реализация алгоритма блока на языке ST

```
FUNCTIONAL_BLOCK PID

    VAR INPUT
        AUTO : BOOL;          (* 0 - manual, 1 - automatic *)
        PV : REAL;            (* Process variable *)
        SP : REAL;            (* Set point *)
        X0 : REAL;            (* Manual output adjustment - *)
                              (* Typically from transfer satation *)
        KP : REAL;            (* Proportionality constant *)
        TR : REAL;            (* Reset time *)
        TD : REAL;            (* Derivative time constatnt *)
        CYCLE : TIME;         (* Sampling period *)
```

```
    END_VAR
    VAR_OUTPUT
        XOUT: REAL;
    END_VAR
    VAR ERROR : REAL;         (* PV - SP *)
        ITERM : INTEGRAL;     (* FB for inegral term *)
        DTERM : DERIVATIVE;   (* FB for derivative term *)
    END_VAR

    ERROR := PV - SP;
    (*** Adjust ITERM so that XOUT := X0 when AUTO = 0 ***)
    ITERM (RUN := AUTO, R1 := NOT AUTO, XIN := ERROR,
           X0 := TR * (X0 - ERROR), CYCLE := CYCLE);
    DTERM (RUN := AUTO, XIN := ERROR, CYCLE := CYCLE);
    XOUT := KP * (ERROR + ITERM.XOUT/TR + DTERM.XOUT*TD);

END_FUNCTION_BLOCK
```

Блок RAMP – линейно изменяющегося сигнала

```
       +-----------+
       |    RAMP   |
BOOL---|RUN    BUSY|---BOOL
REAL---|X0     XOUT|---REAL
REAL---|X1         |
TIME---|TR         |
TIME---|CYCLE      |
       +-----------+
```

Описание работы.

Реализация алгоритма блока на языке ST.

```
FUNCTION_BLOCK RAMP
VAR_INPUT
    RUN : BOOL ; (* 0 - track X0, 1 - ramp to/track X1 *)
    X0,X1 : REAL ;
    TR : TIME ; (* Ramp duration *)
    CYCLE : TIME ; (* Sampling period *)
END_VAR
VAR_OUTPUT
    BUSY : BOOL ; (* BUSY = 1 during ramping period *)
    XOUT : REAL := 0.0 ;
END_VAR
VAR XI : REAL ; (* Initial value *)
    T : TIME := T#0s; (* Elapsed time of ramp *)
END_VAR
BUSY := RUN ;
IF RUN THEN
    IF T >= TR THEN BUSY := 0 ; XOUT := X1 ;
    ELSE XOUT := XI + (X1-XI) * TIME_TO_REAL(T)/ TIME_TO_REAL(TR) ;
        T := T + CYCLE ;
```

```
    END_IF ;
ELSE XOUT := X0 ; XI := X0 ; T := t#0s ;
END_IF ;
END_FUNCTION_BLOCK
```

1.3.2. Работа с сигналами управляемого приложения SCADA

При использовании Lua движка в режиме управляемого приложения SCADA программисту доступны так называемые сигналы ядра (Core). Значения этих сигналов синхронизируются с сигналами других приложений. Таким образом, производя чтение и запись общих сигналов, можно максимально просто осуществлять взаимодействие между приложениями SCADA.

Для обращения к сигналам используется следующий синтаксис: Core.signalName или эквивалентный Core["signalName"].

```
-- запись значения 1 в сигнал "signalName"
Core.signalName = 1
-- чтение значения сигнала "signalName" и присвоение его значения
-- переменной var
localvar = Core["signalName"]
```

Если сигнал представляет собой массив, то для доступа к ячейкам следует использовать квадратные скобки с указанием номера ячейки.

ВНИМАНИЕ! Следует помнить, что в SCADA системе нумерация ячеек массива производится с нуля. В Lua, по-умолчанию, с единицы.

```
-- запись значения в нулевую ячейку
Core.signalName[0] = 1
-- чтение значения и печать первой ячейки
print(Core.signalName[1])
```

ВНИМАНИЕ! При работе с сигналами существует ограничение по синтаксису. Для обычных переменных в Lua можно применять множественное присвоение, когда список значений или переменных присваивается списку переменных. Для сигналов ядра такое недопустимо. Только единичное присвоение.

```
-- множественное присвоение обычных переменных
local a, b, c, d = 1, 2, 3, x
-- множественное присвоение для сигналов ядра не допустимо!
Core.SIG1, Core.SIG2 = 1, 0 -- ОШИБКА!
```

Для обращения к полям структурного типа можно использовать как точечную нотацию, так и операцию через квадратные скобки [].

Пример. Core["signalName"].Field1["Field2"]. Все эти формы являются эквивалентными. Также можно использовать косвенное обращение к полям структуры по их имени, находящемся в вспомогательной переменной. Пример. Core[signalName][fieldName]

1.3.2.1. Работа с сигналами через функции

Для одновременного считывания или записи значения сигнала, его флагов и времени изменения можно воспользоваться доступом через специальные функции.

Таблица 1.10 - Описание функций для работы с сигналами

result = Core.setSignal(signalName, value[, utc|offset = 0, flags = nil, valid = true])
Функция производит запись нового значения сигнала, его флагов и времени изменения. Где signalName (тип STRING) - имя сигнала; value (тип зависит от сигнала, которому производится запись значения) - новое значение, utc|offset (тип DWORD) - абсолютное время в utc (>0) или смещение от текущего времени (≤0) в секундах изменения значения; flags (тип INT) - флаги сигнала (см. Приложение 2), valid (тип BOOL) - позволяет управлять НОПР флагом достоверности значения сигнала (по умолчанию функция Core.setSignal считает значение сигнала валидным). Результатом вызова функции является числовой результат (тип USINT). Если запись значения сигнала прошла успешно, то result=0 **Пример:** 1. Некоторому сигналу testSignal (тип BOOL) установим значение равное true. Остальные параметры функции оставим со значениями по умолчанию. **local result = Core.setSignal("testSignal", true)** 2. Некоторому сигналу testSignal (тип BOOL) установим значение равное true, время изменения оставим текущим (смещение сделаем равное 0), установим во флагах сигнала значение нулевого и первого пользовательского битов равными 1 и укажем сигналу, что его значение недостоверно (НОПР сделаем равным false). **local result = Core.setSignal("testSignal", true, 0, 3, false)**
result, value, utc, flags = Core.getSignal(signalName)
Функция производит считывание значения сигнала, его флагов и времени изменения. Где signalName (тип STRING) - имя сигнала; value (тип зависит от сигнала, значение которого считывается) - значение, utc (тип DWORD) - абсолютное время изменения значения в utc (секунды); flags (тип INT) - флаги сигнала (см. Приложение 2). Результатом вызова функции является числовой результат (тип USINT). Если запись значения сигнала прошла успешно, то result=0 **Пример:** 1. Считаем значение, флаги и время изменения некоторого сигнала testSignal. **local result, value, utc, flags = Core.getSignal("testSignal")**
valid = Core.isSignalValid(signalName)
Функция возвращает true, если значение сигнала достоверно, иначе возвращает false. Где signalName (тип STRING) - имя сигнала. **Пример:** 1. Проверим признак достоверности некоторого сигнала testSignal. **local isValid = Core.isSignalValid("testSignal")**

1.3.2.2. Получение архивного значения сигнала

В данном Lua движке реализована возможность получения архивного значения сигнала (при условии, что данный сигнал архивируется).

Формат запроса следующий.

```
local arch = Core(DT).signalName
-- или
local arch = Core(DT)["signalName"]
```

Где DT - это запрашиваемое время сигнала, signalName - имя сигнала. Результатом вызова такой синтаксической конструкции является таблица, которая присваивается переменной arch. У таблицы два поля: result - результат выполнения запроса и val - значение сигнала. Если запрос прошёл успешно, то result=0.

Если требуется запросить архивные значения множества сигналов от одной и той же отметки времени, то можно воспользоваться следующей синтаксической конструкцией.

```
-- Создаём экземпляр ядра для запроса архивных сигналов от DT.
local archCore = Core(DT)
local arch1 = archCore.Signal_1
local arch2 = archCore.Signal_2
...
```

1.3.2.3. Получение времени изменения сигнала

Для получения времени модификации сигнала в данном Lua движке используется следующая синтаксическая конструкция.

```
local ch = Core("change").signalName
-- или
local ch = Core("change")["signalName"]
```

Где signalName - имя сигнала. Результатом вызова такой синтаксической конструкции является время изменения указанного сигнала.

Если требуется запросить время изменения множества сигналов, то можно воспользоваться следующей синтаксической конструкцией.

```
-- Создаём экземпляр ядра для запроса времени изменения сигналов.
local changeCore = Core("change")
local ch1 = changeCore.Signal_1
local ch2 = changeCore.Signal_2
...
```

ВНИМАНИЕ! Время изменения сигнала возвращается в UTC.

1.3.2.4. Справочные функции по сигналам

Таблица 1.11 - Описание справочных функций по сигналам

value = Core.getSignalComment(signalName)

Данная функция по полному имени сигнала возвращает его описание (тип STRING), содержащееся в интерфейсе приложения. Если сигнал по имени не найден, то функция возвращает nil

Пример:

1. Получим значение из поля Комментарий для некоторого сигнала testSignal.

local value = Core.getSignalComment("testSignal")

value = Core.getSignalMeta(signalName)
Данная функция по полному имени сигнала возвращает его метаинформацию (тип STRING), содержащуюся в интерфейсе приложения. Если сигнал по имени не найден, то функция возвращает nil **Пример:** 1. Получим значение из поля Мета для некоторого сигнала testSignal. **local value = Core.getSignalMeta("testSignal")**

1.3.3. Функции ядра приложения SCADA

1.3.3.1. Прямая работа с сигналами ядер

Таблица 1.12 - Описание функций прямой работы с сигналами ядер

Core.directGet(ip:port | appName, tout, signalId|signalName, cellIndex [, archTime, archOffset])
Производит прямой запрос значения сигнала у указанного приложения (используется для получения значений системных сигналов других приложений). Аргументы: - **ip:port | appName** (тип STRING) - адрес приложения задаётся в виде строки "ip:port". Пример "127.0.0.1:10000" либо в виде полного имени приложения "Node_1.App_1"; - **tout** (тип USINT или другой числовой) - тайм-аут указывается в секундах; - **signalId|signalName** (тип DWORD/STRING) – числовой идентификатор сигнала либо имя системного сигнала. Для системных сигналов реализована подстановка в виде "@ИМЯ_СИСТЕМНОГО_СИГНАЛА"; - **cellIndex** (тип USINT или другой числовой) – номер ячейки сигнала; - **archTime** (тип DWORD) – запрашиваемое архивное время значения, заданное в секундах; - **archOffset** (целочисленный числовой тип) – смещение архивного значения от запрашиваемого архивного времени Возвращает: - **result** (тип USINT) – результат запроса. result=0, если нет ошибок (коды см. в Приложение А); - **value** (тип зависит от типа запрашиваемого сигнала) – значение ячейки сигнала; - **type** (тип STRING) – строка, описывающая тип значения; - **archTime** (тип DWORD) – реальное архивное время полученного значения; - **archPos** (тип UINT или другой целочисленный тип) – абсолютная позиция записи значения сигнала в архиве

```
local addr = "127.0.0.1:10000"
local tout = 1
local signalId = 1024
```

```
local cellIndex = 0
local result,val,type,archTime,archPos=Core.directGet(addr,
                                                      tout,
                                                      signalId,
                                                      cellIndex)
```

ВНИМАНИЕ! Чтобы эта функция работала, необходимо наличие хотя бы одного общего сигнала между исходным и конечным приложением. В Сонате используется оптимизация для приложений: из интерфейса приложения удаляются неиспользуемые сигналы. В связи с этим общий сигнал должен не только присутствовать в интерфейсе, но и использоваться самим приложением, иначе он будет удален при компиляции и функция Core.directGet не будет работать

Пример:

1. Получим значение системного сигнала @STATE от приложения ST, которое располагается на узле Controller с ip-адресом 192.168.1.248. Данное приложение работает на порту 10001.

Вариант 1:

local result, value, _type, archTime, archPos = Core.directGet("192.168.1.248:10001", 1, "@STATE", 0)

Вариант 2:

local result, value, _type, archTime, archPos = Core.directGet("Controller.ST", 1, "@STATE", 0)

Core.directSet(ip:port | appName, tout, signalId|signalName, cellIndex, value, type)

Производит прямую установку значения сигнала у указанного приложения (используется для системных сигналов других приложений). Аргументы:

- **ip:port | appName** (тип STRING) - адрес приложения задаётся в виде строки "ip:port". Пример "127.0.0.1:10000" либо в виде полного имени приложения "Node_1.App_1";
- **signalId|signalName** (тип DWORD/STRING) – числовой идентификатор сигнала либо имя системного сигнала. Для системных сигналов реализована подстановка в виде "@ИМЯ_СИСТЕМНОГО_СИГНАЛА";
- **cellIndex** (тип USINT или другой числовой) – номер ячейки сигнала;
- **value** (тип зависит от типа запрашиваемого сигнала) – значение ячейки сигнала;
- **type** (тип STRING) – строка, описывающая тип значения

Возвращает:

- **result** (тип USINT) – результат операции. result=0, если нет ошибок (коды см. в Приложение А);

```
result = Core.directGet("127.0.0.1:10000", 1, 1024, 0, value, "LREAL")
```

ВНИМАНИЕ! Чтобы эта функция работала, необходимо наличие хотя бы одного общего сигнала между исходным и конечным приложением. В Сонате используется оптимизация для архивов и драйверов: из интерфейса приложения удаляются неиспользуемые сигналы. В связи с этим общий сигнал, если это архив или драйвер, должен не только присутствовать в интерфейсе, но и использоваться самим приложением, иначе он будет удален при компиляции и функция Core.directSet не будет работать

Пример:

1. Установим значение системного сигнала @MESSAGE приложения ST, которое располагается на узле Controller с ip-адресом 192.168.1.248. Данное приложение работает на порту 10001.

Вариант 1:

local result = Core.directSet("192.168.1.248:10001", 1, "@MESSAGE", 0, "Тестовое значение", "STRING")

Вариант 2:

local result = Core.directSet("Controller.ST", 1, "@MESSAGE", 0, "Тестовое значение", "STRING")

Core.directInfo(ip:port | appName, timeout, signalId)

Производит прямой запрос информации о сигнале у указанного приложения. Аргументы:

- **ip:port | appName** (тип STRING) - адрес приложения задаётся в виде строки "ip:port". Пример "127.0.0.1:10000" либо в виде полного имени приложения "Node_1.App_1";
- **tout** (тип USINT или другой числовой) - тайм-аут указывается в секундах;
- **signalId** (тип DWORD) – числовой идентификатор предыдущего сигнала (=0 в начале)

Возвращает:

- **result** (тип USINT) – результат запроса. result=0, если нет ошибок (коды см. в Приложение А);
- **nextSignalId** (тип DWORD) – числовой идентификатор следующего в интерфейсе приложения сигнала;
- **signalId** (тип DWORD) – текущий числовой идентификатор сигнала;
- **signalName** (тип STRING) – имя сигнала;
- **type** (тип STRING) – тип сигнала;
- **size** (тип UINT или другой целочисленный числовой) - размер сигнала (если сигнал является массивом, то это его размерность, если нет, то значение будет равно 0)

ВНИМАНИЕ! Чтобы эта функция работала, необходимо наличие хотя бы одного общего сигнала между исходным и конечным приложением. В Сонате используется оптимизация для архивов и драйверов: из интерфейса приложения удаляются неиспользуемые сигналы. В связи с этим общий сигнал, если это архив или драйвер, должен не только присутствовать в интерфейсе, но и использоваться самим приложением, иначе он будет удален при компиляции и функция Core.directInfo не будет работать

Пример:

1. Получим информацию о сигнале @MESSAGE (по его signalId = 4294967291) приложения ST, которое располагается на узле Controller с ip-адресом 192.168.1.248. Данное приложение работает на порту 10001. SignalId системных сигналов являются уникальными и неизменными, а signalId других сигналов могут меняться.

Вариант 1:

local result, nextSignalId, SignalId, signalName, type, size = Core.directInfo("192.168.1.248:10001", 1, 4294967291)

Вариант 2:

local result, nextSignalId, SignalId, signalName, type, size = Core.directInfo("Controller.ST", 1, 4294967291)

1.3.3.2. Передача и приём данных между приложениями

ВНИМАНИЕ! Чтобы данные функции работали, необходимо наличие хотя бы одного общего сигнала между исходным и конечным приложением. В Сонате используется оптимизация для архивов и драйверов: из интерфейса приложения удаляются неиспользуемые сигналы. В связи с этим общий сигнал, если это архив или драйвер, должен не только присутствовать в интерфейсе, но и использоваться самим приложением, иначе он будет удален при компиляции и функции не будут работать.

Таблица 1.13 - Описание функций приёма и передачи между приложениями

Core.sendData(appName, data)
Производит отправку произвольных данных указанному приложению. Данные должны иметь тип строки. Аргументы: - **appName** (тип STRING) - полное имя приложения в виде строки. Пример "Node_1.Algorithm"; - **data** - строка с данными. Возвращает: - **result** (тип USINT) – результат постановки в очередь на отправку данных. result=0, если нет ошибок (коды см. в Приложение А)
Core.getData(appName)
Получает (при наличии) готовые данные от указанного приложения. Аргументы: - **appName** (тип STRING) - полное имя приложения в виде строки. Пример "Node_1.Algorithm". Возвращает: - **result** (тип USINT) – результат получения данных. result=0, если нет ошибок (коды см. в Приложение А); - **data | nil** – данные, если нет ошибок или nil в случае ошибки
Core.deleteData(appName)
Удаляет неиспользуемые данные для освобождения памяти. Данные можно не удалять, если идёт постоянный обмен между приложениями. Следующая посылка данных удалит старые данные. Аргументы: - **appName** (тип STRING) - полное имя приложения в виде строки, полученные данные которого нужно удалить. Пример "Node_1.Algorithm". Возвращает: - **result** (тип USINT) – результат удаления данных. result=0, если нет ошибок

1.3.3.3. Прочие функции ядра

Таблица 1.14 - Описание прочих функций ядра

Core.addLogMsg(msg[, filePath, maxFileSize])
Добавляет сообщение **msg** в лог-файл. - **msg** (тип STRING) – сообщение; - **filePath** (тип STRING) – имя файла (опциональный параметр). Если имя файла не указывается, то запись происходит в LOG файл приложения; - **maxFileSize** (целочисленный тип) – максимальный размер файла (опциональный параметр, указывается в байтах). **Пример:** 1. Будем писать тестовое сообщение в файл testLogFile.log, который будет располагаться там же, где и дистрибутив. Максимальный объем данного файла ограничим 100000 Байт. **Core.addLogMsg("Тестовое сообщение.", "testLogFile.log", 100000)** 2. Будем писать тестовое сообщение в лог файл приложения. **Core.addLogMsg("Тестовое сообщение.")**
Core.addEvent(msg [, category, state, source, user, groupUUID, userDT, meta])

Добавляет сообщение о произошедшем событии в хранилище.

- **msg** (тип STRING) – сообщение;
- **catergory** (тип UINT) – категория события:
  - 0 - системное информационное сообщение;
  - 1 - системное предупреждение;
  - 2 - системное сообщение об аварии;
  - 3 - системное событие безопасности;
  - 4-9999 - пользовательские события, не относящиеся к какой-либо тревоге;
  - 10000-19999 - пользовательские тревожные события, относящиеся к обязательно квитируемым тревогам;
  - 20000-29999 - пользовательские тревожные события, относящиеся к необязательно квитируемым тревогам;
  - 30000-39999 - пользовательские тревожные события, относящиеся к неквитируемым тревогам
- **state** (тип USINT) – состояние события (0 — исчезло, 1 — возникло);
- **source** (тип STRING) – источник события;
- **user** (тип STRING) – имя пользователя, вызвавшего данное событие;
- **groupUUID** (тип STRING) – уникальный идентификатор тревоги по данному событию;
- **userDT** – дополнительная метка времени, которая может отличаться от системного;
- **meta** (тип STRING) - дополнительная метаинформация, сохраняемая с событием

Пример:

1. Создадим событие с сообщением - Тестовое событие, категория события будет 10000, источник АРМ1, пользователь operator1 и уникальный идентификатор test_UUID. Параметр userDT и meta не будем использовать.

Core.addEvent("Тестовое событие", 10000, 1, "ARM1", "operator1", "test_UUID")

Core.getEvents(startDT, endDT)

Запрашивает у системы список событий за указанный диапазон дат. Возвращает таблицу, каждая ячейка которого описывает одно событие. Событие описывается в виде таблицы со следующими полями:

- **pos** – абсолютная позиция события в архиве;
- **dt** – системное время возникновения события;
- **userDT** – пользовательское время возникновения события;
- **UUID** – уникальный идентификатор тревоги по данному событию;
- **groupUUID** – уникальный идентификатор тревоги по данному событию;
- **category** – категория события;
- **state** – состояние события;
- **application** – имя приложения, породившее событие;
- **source** – источник события;
- **user** – имя пользователя, вызвавшего данное событие;
- **msg** – сообщение, которое выодится при данном событии;
- **meta –** дополнительная метаинформация, сохраняемая с событием.

Пример использования

```
-- Получаем текущее время.
local dt = os.time()
-- Запрашиваем список событий от текущего времени и 100 секунд назад.
local events = Core.getEvents(dt-100, dt)
-- Сохраняем события в лог-файл.
for _,event in ipairs(events) do
 local date = os.date("%c", event.dt)
```

```
 local msg = string.format("pos=%d,dt=%s,%s",event.pos,date,event.msg)
 Core.addLogMsg(msg)
end
```

Core.setMessage(msg)

Выставляет текущее информационное сообщение приложения

Core.getApps([local=false])

Возвращает кортеж приложений, связанных с данным приложением. Ключом кортежа является полное имя приложения. Значением ячейки является строка с адресом(ами) приложения. Если в качестве аргумента функции передать true, то функция возвращает перечень приложений, исполняемых на том же узле.

local value = Core.getLatency(fullAppName [,interface])

Возвращает время в [s], необходимое для передачи сигнала до данного приложения fullAppName (тип STRING). Если с указанным приложением нет связи, то возвращает -1. Если с приложением используется несколько линий связи, то возвращается минимальное время по любой линии. Если при вызове функции указать опциональный числовой аргумент interface=0..N-1, то функция вернёт время передачи через указанный интерфейс. ВНИМАНИЕ! Чтобы эта функция работала, необходимо наличие хотя бы одного общего сигнала между исходным и конечным приложением. В Сонате используется оптимизация для архивов и драйверов: из интерфейса приложения удаляются неиспользуемые сигналы. В связи с этим общий сигнал, если это архив или драйвер, должен не только присутствовать в интерфейсе, но и использоваться самим приложением, иначе он будет удален при компиляции и функция Core.getLatency не будет работать

Пример:

1. Проверим на доступность по линии связи приложения Loader на узле Controller.

local value = Core.getLatency("Controller.Loader")
if value ~= -1 then
--приложение доступно
else
--приложение недоступно
end

TResult result = Core.setReserve(nil | nodeName | fullAppName, true|false [, timeout=1])

Отправляет команду указанному узлу или приложению на переход или выход из состояния резерва

Пример:

1. Отправляет команду локальному узлу (узел, на котором выполняется наше приложение ЛУА) на переход в резерв

Core.setReserve(nil, true)

true|false|nil = Core.isReserved(nil | nodeName | fullAppName, timeout)

Опрашивает узел или приложение, находится ли оно в состоянии резерва

Пример:

1. Проверим находится ли в резерве локальный узел (узел, на котором выполняется наше приложение ЛУА)

local result = Core.isReserved(nil, 1)

2. Проверим находится ли в резерве узел ARM1

local result = Core.isReserved("ARM1", 1)

3. Проверим находится ли в резерве приложение HMI на узле ARM1

local result = Core.isReserved("ARM1.HMI", 1)

local name = Core.getPeerNodeName()
Возвращает имя узла-партнера по резерву
local name = Core.getNodeName()
Возвращает имя текущего узла

1.3.4. Использование событий в Lua

При использовании Lua движка в режиме управляемого приложения SCADA программисту доступны обработчики событий: событий об изменении сигналов извне и события таймеров. При возникновении того или иного события среда Lua прерывает обычное исполнение программы и переходит к выполнению функции-обработчика события. По окончании исполнения функции-обработчика события среда Lua продолжает исполнять прерванную программу.

1.3.4.1. Установка обработчика на изменение сигналов

Программист может создавать неограниченное количество обработчиков на изменение сигналов извне программы. Один и тот же сигнал может быть использован в разных обработчиках. При изменении такого сигнала извне программы все обработчики будут вызваны последовательно.

ВНИМАНИЕ! Есть возможность создавать один обработчик на группу сигналов. Данный функционал нужно считать приоритетным при работе с большим количеством сигналов, которые обрабатываются одной и той же функцией. Если для обработчика задано несколько сигналов и несколько сигналов изменились, то обработчик будет вызван только один раз для всей группы.

Для задания или удаления обработчика факта изменения сигналов извне служит следующая функция.

Core.onExtChange (signals, function | nil, [userArg])

- **signals** - массив строк с именами сигналов (или их ячеек) ядра, для которых создаётся или удаляется обработчик. Имена сигналов (ячеек) указываются без ключевого слова "Core". Пример {"SIG1", "SIG2", "SIG3[4]"}. ВНИМАНИЕ! В текущей версии Lua движка не поддерживается задание обработчиков изменения для структурных сигналов!"

- **function | nil** - имя Lua функция (пользовательская функция, которая обрабатывает изменившийся сигнал) или nil для задания обработчика или его удаления соответственно;

- **userArg** - опциональный аргумент, который будет передаваться в пользовательскую функцию-обработчик. Благодаря этому аргументу можно определять, какое поведение должна иметь функция-обработчик при использовании её для нескольких обработчиков. ВНИМАНИЕ! Значение аргумента userArg запоминается на момент вызова функции Core.onExtChange и при вызовах обработчика событий подставляется в неизменном виде.

ВНИМАНИЕ! Событие об изменении сигнала возникает только при внешнем изменении значения сигнала другими программами проекта. При изменении сигнала в той же программе, где установлен обработчик, он (обработчик) не будет вызван.

Пример 1. В данном примере созданы два обработчика. В первом обработчике при изменении значения SIG1 или SIG2, вызывается функция callback, которой в качестве аргументов передаются имена сигналов SIG3 и SIG4, а далее их значения инвертируются. Во втором обработчике при изменении значения SIG3 или SIG4, вызывается та же функция callback, которой в качестве аргументов передаются имена сигналов SIG1 и SIG2, а далее их значения инвертируются. Учтите,

что изменение значений сигналов SIG1, SIG2, SIG3 и SIG4 внутри данной программы не приводит к срабатыванию обработчиков.

```
 -- описываем функцию-обработчик
 local function callback(userArg [, cellName, signalId, cellIndex])
     for _, v in ipairs(userArg) do
         Core[v] = not Core[v]
     end
 end
 -- задаем обработчики
Core.onExtChange({"SIG1", "SIG2"}, callback, {"SIG3", "SIG4"})
Core.onExtChange({"SIG3", "SIG4"}, callback, {"SIG1", "SIG2"})

 -- Переводим программу в режим ожидания
Core.waitEvents()
```

Пример 2. В данном примере есть две группы глобальных сигналов (пусть все сигналы будут типа INT), которые обрабатываются двумя разными функциями. Для каждой группы сигналов создается свой обработчик. Есть сигнал G_1, который входит в обе группы. Для него последовательно будут вызваны оба обработчика. Так же показано, что скалярный сигнал и массив по разному вызываются в функциях (см. функцию callback).

```
-- Функция, которая разбирает все параметры изменившегося сигнала
-- userArgs - пользовательские аргументы
-- cellName - имя изменившегося сигнала
-- signalId - Id изменившегося сигнала
-- cellIndex - номер ячейки, если сигнал - массив (у скалярного сигнала = 0)
local function callback(userArgs, cellName, signalId, cellIndex)
    -- Запишем в лог приложения, что была вызвана данная функция
    Core.addLogMsg(userArgs..": сигнал "..cellName)
    -- Далее выводим в лог приложения всю информацию об изменившемся сигнале
    Core.addLogMsg("Имя сигнала: "..cellName)
    Core.addLogMsg("Id сигнала: "..signalId)
    Core.addLogMsg("Индекс ячейки: "..cellIndex)
    if(cellName ~= "G_3") then
        -- Для скалярных сигналов
        Core.addLogMsg("Значение сигнала: "..Core[cellName])
    else
        -- Для массивов
        Core.addLogMsg("Значение сигнала: "..Core[cellName][cellIndex])
end

-- Функция, которая пишет в логи приложения значение квадратного корня изменившег
local function SQRT(userArgs, cellName)
 Core.addLogMsg(userArgs..": сигнал "..cellName)
 Core.addLogMsg("Квадратный корень из "..Core[cellName].." равен "..math.sqrt(
end

-- Сформируем таблицу из нескольких сигналов
-- Сигнал G_3 - массив
local signals_group1 = {"G_1", "G_2", "G_3"}
```

```
-- Задаем один обработчик на данную группу сигналов
Core.onExtChange(signals_group1, callback, "Вызван обработчик для группы 1")

-- Сформируем таблицу из нескольких сигналов
local signals_group2 = {"G_1", "G_4", "G_5"}
-- Задаем один обработчик на данную группу сигналов
Core.onExtChange(signals_group2, SQRT, "Вызван обработчик для группы 2")

-- Переводим программу в режим ожидания
Core.waitEvents()
```

1.3.4.2. Установка обработчика на периодические события

Программист может создавать неограниченное количество обработчиков периодических событий (таймеров).

Для задания или удаления обработчика таймера служит следующая функция.

Core.onTimer (timer, period, function | nil, [userArg], [now=true])>

- **timer** - целочисленный номер таймера;

- **period** - период вызова функции-обработчика в [s];

- **function | nil** - имя Lua функция или nil для задания обработчика или его удаления, соответственно;

- **userArg** - опциональный аргумент, который будет передаваться в функцию обработчик. ВНИМАНИЕ! Значение аргумента userArg запоминается на момент вызова функции Core.onTimer и при вызовах обработчика событий подставляется в неизменном виде;

- **now** - булевский опциональный аргумент, сообщающий, следует ли осуществить вызов функции-обработчика сразу после регистрации.

```
 -- описываем функцию-обработчик
 local function blink(userArg)
    Core[userArg] = not Core[userArg]
 end
 -- задаем обработчики
Core.onTimer(1, 0.5, blink, "SIG1")
Core.onTimer (2, 1, blink, "SIG2")
Core.onTimer (3, 2, blink, "SIG3")
```

1.3.4.3. Установка обработчика на отправку данных

Программист может создавать обработчик, который будет вызван по завершении передачи данных другому приложению.

Для задания или удаления обработчика служит следующая функция.

Core.onDataSent (function | nil [,userArg])>

- **function | nil** - имя Lua функция или nil для задания обработчика или его удаления, соответственно;

- **userArg** - опциональный аргумент, который будет передаваться в функцию обработчик. ВНИМАНИЕ! Значение аргумента userArg запоминается на момент вызова функции Core.onDataSent и при вызовах обработчика событий подставляется в неизменном виде.

```
 -- Описываем функцию-обработчик.
 local function sent(appName, dataSize, result, userArg)
    ...
 end
 -- Задаем обработчик.
Core.onDataSent(sent, "Node_1.Alg");
 -- Отправляем данные.
Core.sendData("Node_1.Alg", "test");
```

1.3.4.4. Установка обработчика на приём данных

Программист может создавать обработчик, который будет вызван по завершении приёма данных от стороннего приложения.

Для задания или удаления обработчика служит следующая функция.

Core.onDataRecv (function | nil [,userArg])>

- **function | nil** - имя Lua функция или nil для задания обработчика или его удаления, соответственно;

- **userArg** - опциональный аргумент, который будет передаваться в функцию обработчик. ВНИМАНИЕ! Значение аргумента userArg запоминается на момент вызова функции Core.onDataRecv и при вызовах обработчика событий подставляется в неизменном виде.

```
 -- Описываем функцию-обработчик.
 local function recv(appName, dataSize, userArg)
    ...
 end
 -- Задаем обработчик.
Core.onDataRecv(recv, "Node_1.Alg");
 -- Ожидаем прихода данных.
Core.waitEvents();
```

1.3.4.5. Перевод программы в ожидание событий

Для завершения последовательного исполнения программы и перевода её в режим ожидания событий служит следующая функция.

Core.waitEvents( )

Последовательное исполнение Lua программы останавливается на месте вызова этой функции, до прихода сигнала останова программы. Программа ожидает возникновение событий по изменению сигнала или срабатыванию таймера, при этом не тратит процессорное время.

1.3.5. Работа с директориями

Все функции для работы с директориями сосредоточены в таблице lfs.

Таблица 1.15 - Описание функций работы с директориями

lfs.attributes (filepath [, aname])
В случае успеха возвращает таблицу с атрибутами файла или только значение одного атрибута, если имя атрибута задано явно через аргумент aname. В случае ошибки возвращает nil и код ошибки Атрибуты у файла могут быть следующие: - **dev** — в Unix-подобных системах указывает, что данный файл является устройством, в Windiows системах представляет собой имя диска; - **ino** — в Unix-подобных системах указывает inode номер. В Windows системах не имеет значения; - **mode** — строка, представляющая описание вида защиты (file, directory, link, socket, named pipe, char device, block device); - **nlink** — количество жёстких ссылок на файл; - **uid** — идентификатор владельца файла в Unix. В Windows всегда 0; - **gid** — идентификатор группы владельцов файла в Unix. В Windows всегда 0; - **access** — время последнего доступа к файлу; - **modification** — время последней модификации; - **change** — время последней модификации статуса файла; - **size** — размер файла в байтах; - **blocks** — количество блоков, выделенных для файла (Unix); - **blksize** — оптимальный размер блока файловой системы (Unix)
lfs.chdir (path)
Изменяет текущую рабочую директорию на указанную. Возвращает true в случае успеха или nil и строку с ошибкой в случае неудачи
lfs.currentdir ()
Возвращает строку с текущим путём рабочей директории. В случае ошибки возвращает nil и код ошибки
iter, dir_obj = lfs.dir (path)

Эта функция возвращает итератор по объектам файловой системы для указанного пути
`local function printDir(path)`
`  for file in lfs.dir(path) do`
`  if file ~= "." and file ~= ".." then`
`  local f = path..file`
`  local attr = lfs.attributes (f)`
`  if attr.mode ~= "directory" then`
`  print (f)`
`  end`
`  end`
`  end`
`end`

lfs.lock (filehandle, mode[, start[, length]])

Блокирует доступ к файлу или его части. Данная функция работает с открытым файлом. Режим блокировки mode может принимать два значения: "r" - блокировка чтения и "w" - блокировка записи (или эксклюзивная блокировка). Для указания блокирования части служат опциональные аргументы start и length Возвращает true в случае успеха или nil и строку с ошибкой в случае неудачи
lfs.link (src, name[, symlink])
Создаёт ссылку на файл. Первый аргумент src – имя файла, на который создаётся ссылка. Второй аргумент name – имя ссылки. Третий аргумент опциональный. Если он не задан или задан как false, то создаётся жёсткая ссылка, иначе создаётся символьная ссылка
lfs.mkdir (dirname)
Создаёт новую директорию с указанным именем в текущей папке. Возвращает true в случае успеха или nil и строку с ошибкой в случае неудачи
lfs.mkpath(dirname)
Создаёт полный путь, который может состоять из нескольких вложенных директорий. Возвращает true в случае успеха или nil и строку с ошибкой в случае неудачи
lfs.rmdir (dirname)
Удаляет указанную директорию. Возвращает true в случае успеха или nil и строку с ошибкой в случае неудачи
lfs.setmode (file, mode)
Задаёт режим записи данных в файл ("binary" или "text"). Возвращает строку с предыдущим режимом записи в файл или nil в случае ошибки
lfs.touch (filepath [, atime [, mtime]])
Устанавливает время доступа **atime** и модификации **mtime** указанного файла **filepath**. Возвращает true в случае успеха или nil и строку с ошибкой в случае неудачи
lfs.unlock (filehandle[, start[, length]])
Снимает блокировку с файла или его части. См. Описание функции lfs.lock(). Возвращает true в случае успеха или nil и строку с ошибкой в случае неудачи

1.3.6. Работа с последовательным портом

В Lua интерпретаторе есть библиотека SerialPort для работы с последовательным портом. В данной библиотеке есть три эквивалентные функции для создания объекта для работы с последовательным портом.

Таблица 1.16 - Описание функций работы с последовательным портом

SerialPort.new(name [, br=115200, db=8, sb=2, p=”NONE”, to=0, invRTS=false])
SerialPort.create(name [, br=115200, db=8, sb=2, p=”NONE”, to=0, invRTS=false])
SerialPort.open(name [, br=115200, db=8, sb=2, p=”NONE”, to=0, invRTS=false])
Эти функции в случае успеха возвращают экземпляр объекта для работы с последовательным портом или nil, если порт не удалось открыть Аргументы функций: - **name** – имя последовательного порта (\\.\COM1 или /dev/ser1);

- **br** – скорость обмена (9600, 19200, 38400, 57600, 115200);
- **db** – количество бит данных (6,7,8);
- **sb** – количество стоповых бит (1,2);
- **p** – тип контроля чётности данных ("NONE", "EVEN", "ODD");
- **to** – тайм-аут передачи и приёма [us];
- **invRTS** – признак инверсии сигнала RTS при передаче.

Пример использования

```
local port = SerialPort.new("COM1", 115200) -- создание объекта
port:send("test") -- отправка данных
port = nil – уничтожение объекта
```

У созданного объекта port доступны следующие методы
port:close(a)
Закрывает COM порт
port:send(data)
Отправляет строку с данными data в последовательный порт. Возвращает количество реально отправленных байт. В случае фатальной ошибки возвращает nil. Для контроля исправности оборудования следует сравнивать количество отправляемых байт с количеством реально отправленных байт
port:recv(count)
Пытается принять из последовательного порта указанное количество count байт с ожиданием заданного при создании объекта port тайм-аута. Возвращает строку с данными в случае успеха или nil в случае ошибки
port:clearBuffers( )
Очищает приёмный и передающий буфер
port:recvBytesAvailable( )
Возвращает количество байт, находящихся в приёмном буфере и доступных для считывания или nil в случае ошибки
port:sendBytesAvailable( )
Возвращает количество байт, находящихся в буфере на отправку, или nil в случае ошибки

1.3.7. Описание функций работы с syslog

Для работы с syslog используется библиотека lsyslog. Данную библиотеку не нужно подключать функцией require.

Таблица 1.17 - Описание функций работы с syslog

lsyslog.open([appName], [facility])
Запускает в работу службу по записи сообщений в системный лог Описание параметров: - **appName** - опциональное имя приложения; - **facility** - опциональная категория, к которой относится данная программа.

Для параметра facility зарезервированы следующие константы: - **lsyslog.FACILITY.AUTH** - сообщения о безопасности/авторизации (РЕКОМЕНДУЕТСЯ использовать вместо него LOG_AUTHPRIV); - **lsyslog.FACILITY.AUTHPRIV** - сообщения о безопасности/авторизации (частные); - **lsyslog.FACILITY.CRON** - демон часов (cron и at); - **lsyslog.FACILITY.DAEMON** - другие системные демоны; - **lsyslog.FACILITY.FTP** - демон FTP; - **lsyslog.FACILITY.KERN** - сообщения ядра; - **lsyslog.FACILITY.LOCAL0** - зарезервированы для определения пользователем; - **lsyslog.FACILITY.LOCAL1** - зарезервированы для определения пользователем; - **lsyslog.FACILITY.LOCAL2** - зарезервированы для определения пользователем; - **lsyslog.FACILITY.LOCAL3** - зарезервированы для определения пользователем; - **lsyslog.FACILITY.LOCAL4** - зарезервированы для определения пользователем; - **lsyslog.FACILITY.LOCAL5** - зарезервированы для определения пользователем; - **lsyslog.FACILITY.LOCAL6** - зарезервированы для определения пользователем; - **lsyslog.FACILITY.LOCAL7** - зарезервированы для определения пользователем; - **lsyslog.FACILITY.LPR** - подсистема принтера; - **lsyslog.FACILITY.MAIL** - почтовая подсистема; - **lsyslog.FACILITY.NEWS** - подсистема новостей USENET; - **lsyslog.FACILITY.SYSLOG** - сообщения, генерируемые syslogd; - **lsyslog.FACILITY.USER** (по умолчанию) - общие сообщения на уровне пользователя; - **lsyslog.FACILITY.UUCP** - подсистема UUCP.
lsyslog.log(level, msg)
Данная функция записывает сообщение в системный лог Описание параметров: - **level** - степень важности сообщения; - **msg** - сообщение. Для параметра level зарезервированы следующие константы: - **lsyslog.LEVEL.EMERG** - система остановлена; - **lsyslog.LEVEL.ALERT** - требуется немедленное вмешательство; - **lsyslog.LEVEL.CRIT** - критические условия; - **lsyslog.LEVEL.ERR** - ошибки; - **lsyslog.LEVEL.WARNING** - предупреждения; - **lsyslog.LEVEL.NOTICE** - важные рабочие условия; - **lsyslog.LEVEL.DEBUG** - сообщения об отладке.
lsyslog.close()
Данная функция останавливает службу в программе для работы сообщениями в системном логе

1.3.8. LuaSocket

1.3.8.1. Введение

LuaSocket - это библиотека для работы с сетью. Она предоставляет низкоуровневые транспортные функции TCP и UDP протоколов и функции разрешения имён хостов DNS. Для

более детальной информации по работе тех или иных сетевых функций обратитесь к справочному руководству по вашей операционной системе.

1.3.8.2. TCP

TCP (Transfer Control Protocol) - это потоковый протокол передачи данных. Приложения, взаимодействующие друг с другом через TCP, могут отправлять и принимать данные, как поток байт. Протокол TCP самостоятельно обеспечивает разбивку потока на сетевые пакеты, передачу, контроль целостности и сборку в конечной точке исходного потока. Библиотека позволяет пользователям работать с потоком байт несколькими способами: чтение по строкам, чтение блоками указанного размера и чтение всего, что придёт до закрытия соединения.

Библиотека различает три вида TCP сокетов: *master*, *client* и *server* сокеты.

Мастер (master) сокет - это новый TCP сокет, возвращаемый функцией socket.tcp. Этот сокет может быть преобразован в серверный (server) сокет, для организации сервера, обслуживающего подключения, или в клиентский (client) сокет, для создания подключения к удалённому серверу. Для преобразования в серверный сокет нужно связать данный мастер сокет при помощи функции bind с адресом, по которому будут приниматься входящие клиентские подключения. Затем следует вызвать функцию listen, чтобы заставить сокет принимать соединения. Для преобразования мастер сокета в клиентский сокет следует вызвать метод connect с указанием адреса сервера, к которому следует подключиться.

На стороне сервера приложение должно использовать функцию accept для того, чтобы принять подключение клиентского сокета и установить новое соединение. Функция вернёт объект серверного сокета, у которого доступны следующие методы. getsockname, setoption, settimeout, close.

Клиентские сокеты имеют следующие методы. send, receive, getsockname, getpeername, setoption, settimeout, shutdown, close.

Пример:

Простой эхо-сервер, использующий LuaSocket. Эта программа связывает master сокет с адресом 0.0.0.0 и 2000 портом. Данный адрес позволяет принимать подключения из любой подсети. После установления соединения программа считывает строку данных, отправляет их обратно и сразу закрывает соединение. Вы можете использовать программу Telnet для проверки данного сервера.

```
-- load namespace
local socket = require("socket")
-- create a TCP socket and bind it to the local host, at any port
local server = assert(socket.bind("*", 2000))
-- find out which port the OS chose for us
local ip, port = server:getsockname()
-- print a message informing what's up
print("Please telnet to localhost on port " .. port)
print("After connecting, you have 10s to enter a line to be echoed")
-- loop forever waiting for clients
while 1 do
    -- wait for a connection from any client
    local client = server:accept()
    -- make sure we don't block waiting for this client's line
    client:settimeout(10)
    -- receive the line
    local line, err = client:receive()
```

```
    -- if there was no error, send it back to the client
    if not err then client:send(line .. "\n") end
    -- done with client, close the object
    client:close()
end
```

1.3.8.3. UDP

UDP (User Datagram Protocol) - протокол негарантированной передачи датаграмм (пакетов). Приложения, использующие для общения UDP, посылают и принимают данные, как независимые блоки для которых не гарантируется ни порядок доставки, ни сам факт их доставки. Пакеты принимаются атомарно. Пакет либо пришёл полностью, либо не пришёл совсем. Главным достоинством протокола является его простота и отсутствие необходимости устанавливать и обслуживать подключение. Пользователь просто пересылает датаграмму с одного адреса до другого, при этом должен сам контролировать тайм-ауты и факт доставки.

Для создания объекта UDP сокета используется функция socket.udp. После создания объекта UDP сокета можно сразу вызывать функцию оправки сетевого пакета sendto на указанный адрес и сетевой порт. В этой функции допустимо использовать только IP адрес. Использование имени сервера заблокировано, так как это существенно снижает быстродействие программы из-за необходимости разрешения имени сервера. Методы receive и receivefrom могут быть использованы для получения датаграмм. Метод receivefrom дополнительно возвращает IP адрес и порт отправителя.

Когда осуществляется постоянное взаимодействие с одним и тем же партнёром, то можно вызвать метод setpeername для указания постоянного партнёра сокета, это снизит накладные расходы на 30%. После этого нельзя будет использовать методы sendto и receivefrom, вместо них следует использовать методы send и receive, в которых не указывается адрес принимающей стороны.

До указания UDP сокету его локального адреса приложение должно вызвать метод setsockname *перед* отправкой любой датаграммы. В противном случае сокет будет автоматически привязан к произвольном локальному адресу, который невозможно будет изменить. Также для объекта UDP сокета доступны следующие методы: getpeername, getsockname, settimeout, setoption и close.

Пример:

Простой клиент для получения точного времени, использующий LuaSocket. Эта программа подключается к удалённому серверу и пытается получить текущее время, печатает его или ошибку на экране в случае неудачи.

```
-- change here to the host an port you want to contact
local host, port = "localhost", 13
-- load namespace
local socket = require("socket")
-- convert host name to ip address
local ip = assert(socket.dns.toip(host))
-- create a new UDP object
local udp = assert(socket.udp())
-- contact daytime host
assert(udp:sendto("anything", ip, port))
-- retrieve the answer and print results
io.write(assert(udp:receive()))
```

1.3.8.4. Пространство имён socket

Пространство имён socket содержит основную функциональность библиотеки LuaSocket.
Для подключения библиотеки сокетов следует использовать следующий код.

```
-- loads the socket module
local socket = require("socket")
```

Таблица 1.18 - Описание функций библиотеки LuaSocket

socket.bind(address, port [, backlog])
Данная функция библиотеки создаёт и возвращает объект серверного TCP сокета, привязанного к указанному локальному адресу и порту, готового к приёму клиентских подключений. Опциональный аргумент backlog (по умолчанию равный 32) передаётся автоматически в метод listen. Внимание! Создаваемый серверный сокет автоматически использует параметр "reuseaddr", установленный в **true**
socket.connect(address, port [, locaddr, locport])
Данная функция создаёт объект клиентского TCP сокета и устанавливает соединение с удалённым хостом по указанному адресу и порту. Опционально можно указать локальный адрес и порт
socket.newtry(finalizer)

Данная функция возвращает функцию обёртки try над функцией пользователя finalizer, которая позволяет произвести безопасную обработку ошибки после возникновения исключения.
Данная методика позволяет сократить большое количество работы по обработке исключений сокетов.
Пример
`foo = socket.protect(function()`
`-- connect somewhere`
`local c = socket.try(socket.connect("somewhere", 42))`
`-- create a try function that closes 'c' on error`
`local try = socket.newtry(function() c:close() end)`
`-- do everything reassured c will be closed`
`try(c:send("hello there?\r\n"))`
`local answer = try(c:receive())`
`...`
`try(c:send("good bye\r\n"))`
`c:close()`
`end)`

socket.protect(func)
Возвращает функцию-обёртку над функцией пользователя func, превращая её в безопасную. Данная функция-обёртка перехватывает только исключения, вызываемые функциями try и newtry и не перехватывает обычные ошибки Lua. При возникновении исключения внутри функция -обёртка вернёт **nil** и сообщение об ошибке. Внимание! Будьте осторожны. Если ваша функция производит другие нелегальные действия, вызывающие исключение, то защищённая функця-обёртка перехватит исключение и вернёт её описание как строку

socket.select(recvt, sendt [, timeout])

Ожидает изменения состояния указанного набора сокетов. recvt - массив сокетов, для которых требуется проверка наличия данных для чтения. sendt - массив сокетов, для которых требуется проверка возможности отправки данных. Timeout - максимальное время (в секундах) ожидания изменения состояния сокетов. Если тайм-аут не задан или его значение равно **nil** или меньше нуля, то функция ожидает неограниченное время. recvt и sendt могут быть пустыми таблицами или **nil**. Неверные значения сокетов или ячейки в массиве с нечисловыми индексами будут игнорироваться.

Данная функция возвращает: список сокетов, готовых для чтения данных, список сокетов, готовых для отправки данных и либо сообщение "timeout" (в случае тайм-аута) или **nil** во всех других случаях. Возвращаемые таблицы содержат два набора ключей: целые числа и сами сокеты.

Важное замечание! В операционной системе Windows присутствует ошибка. Функция select неправильно работает для неблокирующих TCP сокетов. Функция может сообщить, что сокет готов к передаче новой порции данных, хотя в действительности не готов.

Ещё одно важное замечание! Вызов функции select с серверным сокетом в параметре recvt перед вызовом метода accept не гарантирует отсутствие блокировки. Вместо этого используйте метод settimeout, в противном случае метод accept может заблокировать исполнение программы на неограниченное время

socket.skip(d [, ret1, ret2 ... retN])

Выбрасывает указанное количество d первых аргументов из последовательности ret_1 ... ret_N и возвращает остаток. ret_{d+1} to ret_N.

Эту функцию удобно использовать, чтобы предотвратить создание бесполезных переменных.

Пример

```
-- get the status code and separator from SMTP server reply
local code, sep = socket.skip(2, string.find(line, "^(%d%d%d)(.?)"))
```

socket.sleep(time)

Останавливает выполнение программы на указанное количество секунд

socket.gettime()

Данная функция возвращает отметку времени в секундах для вычисления интервалов времени. Данная функция похожа на функцию os.clock, однако использование функции os.clock предпочтительнее, так как она не подвержена переводу часов и обладает большей точностью.

```
t = socket.gettime()
-- do stuff
print(socket.gettime() - t .. " seconds elapsed")
```

socket.try(ret1 [, ret2 ... retN])

Вызывает исключение. Исключение может быть перехвачено только функцией protect. Аргументы ret1 по retN могут быть произвольными, однако, как правило, в их качестве используются значения, возвращаемые функцией try.

Функция возвращает аргументы с ret1 по retN, если ret1 не равен **nil**. В противном случае, функция вызывает ошибку при помощи функции error, передавая ей аргумент ret2.

```
-- connects or throws an exception with the appropriate error message
c = socket.try(socket.connect("localhost", 80))
```

socket._VERSION
Данная переменная содержит номер версии библиотеки LuaSocket

1.3.8.5. Функции разрешения имён хостов DNS

Таблица 1.19 - Описание функций разрешения имён хостов

socket.dns.gethostname()
Функция возвращает строку - имя текущего хоста, на котором исполняется программа
socket.dns.tohostname(address)
Функция преобразует строку address в имя хоста. Address может быть как строкой IP адреса, так и именем хоста. В случае успеха функция возвращает каноническое имя хоста и таблицу с дополнительной информацией. В случае ошибки функция возвращает **nil** и сообщение об ошибке. Структура таблицы с дополнительной информацией представлена ниже. resolved = { name = *canonic-name*, alias = *alias-list*, ip = *ip-address-list*} Внимание! Список псевдонимов может быть пустым
socket.dns.toip(address)
Преобразует имя хоста в IP адрес. Address может быть как IP адресом, так и именем. В случае успеха функция возвращает первый IP адрес и таблицу с дополнительной информацией. В случае ошибки функция возвращает **nil** и сообщение об ошибке. Структура таблицы с дополнительной информацией эквивалентна таблице, возвращаемой функцией socket.dns.tohostname( )

1.3.8.6. Транспортные функции UDP

Таблица 1.20 - Описание транспортных функций UDP

socket.udp()
В случае успеха создаёт и возвращает объект непривязанного (unconnected) UDP сокета, который используется для связи без установления соединения. В противном случае функция возвращает **nil** и сообщение об ошибке. Данный сокет поддерживает следующие методы: sendto, receive, receivefrom, getsockname, setoption, settimeout, setpeername, setsockname и close. UDP протокол не поддерживает установку соединения, однако, если требуется осуществлять связь только с одним сторонним партнёром, то можно связать локальный сокет с удалённым адресом при помощи метода setpeername. Это позволяет сократить накладные расходы при отправке и приёме сообщений на 30%. Данный метод возвращает объект привязанного (connected) сокета в случае успеха и **nil** и сообщение об ошибке, в противном случае
connected:close()
unconnected:close()
Закрывает UDP сокет. Используемый внутри объекта сокета локальный адрес освобождается и становится доступным другим приложениям. На закрытом сокете недопустимы никакие методы.

Внимание! Важно закрывать все сокеты, если они более не требуются, так как каждый сокет использует ограниченные ресурсы системы. Допустимо, однако не рекомендуется, присваивать незакрытому сокету значение **nil**. В этом случае произойдёт автоматическое закрытие сокета при очередной очистке мусора. Время исполнения очистки непредсказуемо, и, соответственно, непредсказуемо время закрытия сокета
connected:getpeername()
Возвращает информацию о привязанном UDP партнёре: IP адрес и порт. Для непривязанного сокета данный вызов метода не имеет смысла и не предусмотрен
connected:getsockname()
unconnected:getsockname()
Метод возвращает: строку с локальным IP адресом сокета и числовое значение порта. В случае ошибки данный метод возвращает **nil**. Внимание! UDP сокет изначально не имеет локального адреса до первого вызова метода setsockname или sendto. После вызова любого из этих методов UDP сокет автоматически получает произвольный локальный адрес
connected:receive([size])
unconnected:receive([size])
Получает датаграмму у указанного сокета. Если UDP сокет привязанный, то датаграмма получается только от привязанного удалённого сокета, в противном случае датаграмма получается от кого угодно. Опциональный аргумент size определяет максимальный размер датаграммы, который может быть получен. Если исходная датаграмма по размеру превышает указанный размер, то лишние байты отбрасываются. Если датаграмма меньше указанного размера, то возвращается как есть. Если аргумент size не указан, то принимается значение, равное 8192 байтам. В случае успеха данный метод возвращает принятую датаграмму. В противном случае возвращает **nil** и строку с ошибкой 'timeout'
unconnected:receivefrom([size])
Данный метод работает аналогично методу receive, но дополнительно возвращает IP адрес и порт, откуда была получена датаграмма
connected:send(datagram)
Отправляет датаграмму привязанному партнёру. Аргумент datagram представляет собой строку с содержимым датаграммы. Для UDP протокола максимальный размер датаграммы не может превышать размер в 64K минус размер IP заголовка. При размере датаграммы, превышающей MTU=1514 байт, происходит её фрагментация и передача отдельными сетевыми пакетами. Данный метод является неблокирующим. Возвращает 1 в случае успеха или **nil** и сообщение об ошибке в противном случае
unconnected:sendto(datagram, ip, port)
Отправляет датаграмму на указанный IP адрес и порт. Данный метод является неблокирующим. Возвращает 1 в случае успеха или **nil** и сообщение об ошибке в противном случае
connected:setpeername('*')
unconnected:setpeername(address, port)
Данный метод позволяет изменять привязку сокета. Первый вариант метода позволяет отвязать связанный сокет. Второй вариант метода позволяет связать несвязанный сокет с указанным IP адресом и портом.

Связанный UDP сокет должен использовать методы send и receive. Несвязанный UDP сокет должен использовать методы sendto и receivefrom. В качестве аргумента address можно указывать как IP адрес, так и имя хоста. Аргумент port задаёт адрес порта. Метод возвращает 1 в случае успеха или **nil** и сообщение об ошибке в противном случае.
connected:setsockname(address, port)
Задаёт UDP сокету локальный адрес. Адрес может быть именем хоста или IP адресом. Если в качестве адреса задано '*', то библиотека привязывает сокет к широковещательному адресу INADDR_ANY. Если в качестве значения порта указан 0, то библиотека автоматически выбирает свободный порт. Метод возвращает 1 в случае успеха или **nil** и сообщение об ошибке в противном случае. Внимание! Данный метод нужно вызывать однократно до отправки любой датаграммы, так как в противном случае система автоматически назначит сокету локальный адрес, который нельзя уже будет поменять
connected:setoption(option [, value])
unconnected:setoption(option [, value])
Данный метод позволяет задавать параметры работы UDP сокета. Параметры задаются при помощи строкового имени и значения. Допустимы следующие параметры. - 'dontroute'. Установка данного параметра в **true** сообщает сетевому протоколу, что для доставки исходящего сообщения не следует использовать стандартный механизм маршрутизации; - 'broadcast': Установка данного параметра в **true** сообщает сетевому протоколу, что данный сокет может принимать широковещательные сообщения. Метод возвращает 1 в случае успеха или **nil** и сообщение об ошибке в противном случае. Для дополнительного пояснения данных параметров обратитесь к справочному руководству по операционной системе
connected:settimeout(value)
unconnected:settimeout(value)
Данный метод позволяет задать время тайм-аута сокета в секундах. По умолчанию методы receive и receivefrom являются блокирующими, то есть выполнение программы останавливается на неограниченное время до момента прихода данных. Задание времени тайм-аута для сокета позволяет гарантировать, что время выполнения методов receive и receivefrom не превысит установленный лимит. Задание отрицательной величины тайм-аута или **nil** переводит сокет в блокирующий режим. Внимание! У UDP сокетов методы send и sendto являются строго неблокирующими. И тайм-аут для них не применим

1.3.8.7. Транспортные функции TCP

Таблица 1.21 - Описание транспортных функций TCP

socket.tcp()
Создаёт и возвращает объект TCP мастер-сокета. Этот сокет может быть преобразован в серверный (server) сокет, для организации сервера, обслуживающего подключения, или в клиентский (client) сокет для создания подключения к удалённому серверу. Для преобразования в серверный сокет нужно связать данный мастер-сокет при помощи функции bind с адресом, по которому будут приниматься входящие клиентские подключения. Затем следует вызвать

функцию listen, чтобы заставить сокет принимать соединения. Для преобразования мастер-сокета в клиентский сокет следует вызвать метод connect с указанием адреса сервера, к которому следует подключиться. Функция возвращает сокет в случае успеха или **nil** и сообщение об ошибке в противном случае
server:accept()
Подтверждает прием входящего соединения и возвращает новый объект подключённого клиентского сокета этого соединения. В случае ошибки метод возвращает **nil** и строку с сообщением. Внимание! Вызов функции socket.select с серверным сокетом в параметре recvt перед вызовом метода accept не гарантирует отсутствие блокировки. Функция select обрабатывает только приём и передачу данных, но не установление соединения. Используйте метод settimeout для настройки времени блокировки сокета в методе accept
master:bind(address, port)
Данный метод привязывает мастер-сокет к указанному локальному адресу и порту. Аргумент address может быть как IP адресом, так и и именем хоста. Аргумент port должен быть числовым значением в интервале [0..64K). Если в качестве адреса задана строка '*', то библиотека привязывает сокет к широковещательному адресу INADDR_ANY. Если в качестве значения порта указан 0, то библиотека автоматически выбирает свободный порт. Метод возвращает 1 в случае успеха или **nil** и сообщение об ошибке в противном случае. Внимание! Также доступна библиотечная **функция** socket.bind, которая сразу создаёт и возвращает привязанный серверный сокет
master:close()
client:close()
server:close()
Данный метод закрывает TCP сокет. Используемый внутри объекта сокета локальный адрес особождается и становится доступным другим приложениям. На закрытом сокете не допустимы никакие методы. Внимание! Важно закрывать все сокеты, если они более не требуются, так как каждый сокет использует ограниченные ресурсы системы. Допустимо, однако не рекомендуется, присваивать незакрытому сокету значение **nil**. В этом случае произойдёт автоматическое закрытие сокета при очередной очистке мусора. Время исполнения очистки непредсказуемо, и, соответственно, непредсказуемо время закрытия сокета
master:connect(address, port)
Данный метод выполняет попытку установить соединение с удалённым хостом и произвести преобразование мастер-сокета в клиентский сокет. Аргумент address может быть как IP адресом, так и именем хоста. Аргумент port должен быть числом в интервале [1..64K). Метод возвращает 1 в случае успеха или **nil** и сообщение об ошибке в противном случае. Клиентский сокет поддерживает следующие методы send, receive, getsockname, getpeername, settimeout и close. Внимание! Также доступна библиотечная **функция** socket.connect, которая сразу пытается выполнить подключение и в случае успеха возвращает клиентский сокет. Внимание! Метод settimeout влияет на поведение метода connect. Если за указанный интервал времени timeout не будет установлено соединение, то метод connect вернёт **nil** и сообщение об ошибке 'timeout'
client:getpeername()

Данный метод возвращает информацию о партнёре сокета. Возвращает IP адрес в виде строки и номер порта. В случае ошибки возвращает **nil** и сообщение об ошибке. Внимание! Данный метод не работает для серверного сокета
master:getsockname()
client:getsockname()
server:getsockname()
Возвращает информацию о локальном адресе сокета. В случае успеха метод возвращает IP адрес в виде строки и номер порта в виде числа, в противном случае метод возвращает **nil** и сообщение об ошибке
master:getstats()
client:getstats()
server:getstats()
Метод возвращает: количество принятых байт, количество отправленных байт и возраст сокета в секундах. Эта статистическая информация может быть использована для планирования загрузки пропускной способности сети
master:listen(backlog)
Указывает сокету, что он должен ожидать подключения клиентов. Иными словами, данный метод преобразует мастер-сокет в серверный сокет. У серверного сокета доступны следующие методы: accept, getsockname, setoption, settimeout и close. Аргумент backlog задаёт количество одновременно обслуживаемых подключений. Метод возвращает 1 в случае успеха или **nil** и сообщение об ошибке в противном случае
client:receive([pattern [, prefix]])
Данный метод считывает данные из клиентского сокета согласно заданному шаблону pattern и добавляет в начале данных опциональный аргумет prefix.Опциональный аргумент pattern может принимать следующие значения: - '*a': указывает сокету считывать данные до момента закрытия соединения с противоположной стороны; - '*l': указывает сокету считывать данные в виде строки до момента прихода символа LF (ASCII 10). Допустим приход опционального символа CR (ASCII 13). Символы CR и LF не включаются в возвращаемые методом данные; - число: указывает сокету, сколько считывать данных в байтах. В случае успеха метод возвращает считанные данные, в противном случае, возвращает **nil** и сообщение об ошибке. Сообщение об ошибке может содержать строку 'timeout' в случае тайм-аута и 'closed' в случае закрытия соединения с противоположной стороны
client:send(data [, i [, j]])
Метод отправляет данные через клиентский сокет. Агрумент data представляет собой строку с отправляемыми данными. Опциональные аргументы i, j работают аналогично аргументам функции string.sub(s, [i, j]). Аргумент i указывает, с какой позиции из исходной строки извлекать данные. Аргумент j указывает длину извлекаемого фрагмента. Внимание! Отправка пакетов не буфферезирована. Всегда лучше сначала объединить несколько мелких строк в одну
client:setoption(option [, value])
server:setoption(option [, value])
Данный метод позволяет задать параметры TCP сокета. Доступны следующие виды опций:

- 'keepalive': Если данный параметр установлен в **true**, то это заставляет библиотеку периодически отправлять сообщения подключённой стороне для проверки исправности связи;

- 'linger': Данный параметр управляет поведением библиотеки в случае наличия непереданных данных при закрытии соединения. Параметр представляет собой таблицу { on=true| false, timeout=NNN}. Если значение поля 'on' установлено в **true**, система заблокирует процесс закрытия соединения, либо до окончания передачи всех данных, либо до истечения тайм-аута, указанного в секундах. Если значение поля 'on' установлено в **false**, то соединение будет закрываться без оглядки на наличие непереданных данных;

- 'reuseaddr': установка данного параметра в **true** разрешает библиотеке повторно использовать один и тот же адрес при вызове метода bind;

- 'tcp-nodelay': установка данного параметра в **true** отключает механизм накопления пакетов (Nagle's algorithm).

Метод возвращает 1 в случае успеха или **nil** в противном случае.

Внимание! Для дополнительного разъяснения параметров обратитесь к справочному руководству по вашей операционной системе

master:setstats(received, sent, age)

client:setstats(received, sent, age)

server:setstats(received, sent, age)

Устанавливает указанное пользователем значение в качестве начального для последующего накопления. Аргумент received - количество полученных байт, аргумент sent - количество отправленных байт. Аргумент age - возраст сокета в секундах.

Данный метод возвращает 1 в случае успеха или **nil** в противном случае

master:settimeout(value [, mode])

client:settimeout(value [, mode])

server:settimeout(value [, mode])

Задаёт тайм-аут сокета в секундах. По умолчанию все операции ввода-вывода являются блокирующими. Такие методы как: send, receive и accept будут останавливать выполнение программы на неограниченное время до момента завершения операции. Метод settimeout позволяет задать предельное время, в течение которого данные методы будут ожидать завершения операций ввода-вывода. При достижении данного времени и невыполнении операции ввода-вывода все перечисленные выше методы вернут **nil** и сообщение об ошибке 'timeout'.

В данной библиотеке поддерживается два режима тайм-аутов, которые можно настраивать одновременно и раздельно:

- 'b': *block* тайм-аут. Задаёт тайм-аут для единичной операции ввода-вывода. Этот режим используется по умолчанию;

- 't': *total* тайм-аут. Задаёт совокупный тайм-аут на то, сколько библиотека LuaSocket сможет блокировать исполнение Lua скрипта.

Значение тайм-аута, равное **nil** или любому отрицательному числу, блокирует выполнение программы неограниченное время

client:shutdown(mode)

Останавливает часть двустороннего соединения сокета.

Аргумет mode сообщает, какая часть должна быть остановлена:

- "both": прекращает передачу и приём данных. Этот режим используется по умолчанию;
- "send": прекращает передачу данных;
- "receive": прекращает приём данных..

Эта функция возвращает 1

2. РАЗРАБОТКА СОБЫТИЙНЫХ ПРОГРАММ

2.1. Библиотека функциональных блоков событийных приложений

2.1.1. Обобщённые типы данных из стандарта IEC61131

В стандарте IEC61131 описаны обобщенные или универсальные типы данных, которые объявляют допустимый диапазон типов данных, используемый в рассматриваемом блоке или функции.

Обобщенные типы данных начинаются с префикса ANY.

Обобщённые типы данных	**Допустимые типы данных**
ANY	Все элементарные типы или структуры
ANY_ELEMENTARY	ANY_MAGNITUDE, ANY_BIT, ANY_CHARS OR ANY_DATE
ANY_MAGNITUDE	ANY_INT, ANY_REAL OR TIME
ANY_INTEGRAL	ANY_INT OR ANY_BIT
ANY_NUM	ANY_INT OR ANY_REAL
ANY_REAL	REAL, LREAL
ANY_INT	SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT
ANY_UNSIGNED	USINT, UINT, UDINT, ULINT
ANY_SIGNED	SINT, INT, DINT, LINT
ANY_BIT	BOOL, BYTE, WORD, DWORD, LWORD
ANY_CHARS	ANY_CHAR, ANY_STRING
ANY_CHAR	CHAR, WCHAR
ANY_STRING	STRING, WSTRING
ANY_DATE	DATE_AND_TIME, DATE

2.1.2. Общие типы функциональных блоков

2.1.2.1. Блоки преобразования типа

2.1.2.1.1. Стандартные блоки

_TO_*
Преобразование типа переменной.

_TO_*	
* - начальный тип, ** - конечный тип. Допустимы следующие преобразования: - ANY_BIT, ANY_INT в BOOL - ANY_BIT, ANY_INT в BYTE - ANY_BIT, ANY_INT в WORD - ANY_BIT, ANY_INT, REAL в DWORD - ANY_BIT, ANY_NUM в USINT - ANY_BIT, ANY_NUM в SINT - ANY_BIT, ANY_NUM в UINT - ANY_BIT, ANY_NUM в INT - ANY_BIT, ANY_NUM в UDINT - ANY_BIT, ANY_NUM в DINT - ANY_BIT, ANY_NUM в LINT - ANY_BIT, ANY_NUM в ULINT - ANY_NUM, DWORD в REAL - ANY_NUM в LREAL - ANY в STRING	
Входные переменные	
IN0	Преобразуемое значение
Выходные переменные	
OUT	Преобразованное значение

TRUNC	
Отсечение дробной части: 1.6 ⇒ 1 -1.6 ⇒ -1 1.4 ⇒ 1 -1.4 ⇒ -1	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Преобразованное значение, DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

2.1.2.1.2. Преобразование в формат Base64

BASE64_ENCODE
Преобразует входную строку в формат BASE64

BASE64_ENCODE	
Входные переменные	
IN0	Тип STRING. Входная строка
Выходные переменные	
OUT	Тип STRING. Выходные данные в формате BASE64

BASE64_DECODE	
Преобразует входные данные из формата BASE64 в строку	
Входные переменные	
IN0	Тип STRING. Входные данные в формате BASE64
Выходные переменные	
OUT	Тип STRING. Преобразованная строка
VALID	Тип BOOL. Значение TRUE - преобразование прошло успешно, FALSE - в ходе преобразования возникла ошибка, при этом выходная строка будет пустой

2.1.2.1.3. Форматирование

2.1.2.1.3.1. Вещественные числа

FORMAT_REAL	
Преобразование вещественного числа в строку с форматированием	
Входные переменные	
SPECIFIER	Спецификатор формата, INT: 0 – формат без экспоненты, фиксированное количество знаков; 1 – формат с экспонентой; 2 - автоматический выбор формата для минимизации количества печатаемых символов
FLAGS	Флаги, INT. Конечное значение представляет собой сумму следующих констант, определяющих формат: 1 – выравнивание влево (по умолчанию – вправо); 2 – всегда печатать знак, даже если число положительное; 4 – печатать пробел вместо знака ‘+’; 8 – всегда печатать десятичную точку;

FORMAT_REAL	
	16 – заполнять недостающие позиции символом ‘0’, а не пробелами
WIDTH	Ширина результата (в символах), INT. Если значение равно ‘-1’, то данный параметр не учитывается
PRECISION	Количество десятичных знаков после запятой, INT. Если значение равно ‘-1’, то данный параметр не учитывается
IN0	Преобразуемое значение, LREAL, REAL
Выходные переменные	
OUT	Отформатированное значение

2.1.2.1.3.2. Дата и время

FORMAT_DT	
Форматирование даты и времени	
Входные переменные	
FORMAT	Строка, описывающая формат вывода. Для вывода даты могут быть использованы следующие элементы: - d - день в виде одной или двух цифр (1-31); - dd - день в виде двух цифр (01-31); - M - месяц в виде одной или двух цифр (1-12); - MM - месяц в виде двух цифр (01-12); - YY - год в виде двух цифр (00-99); - YYYY - год в виде четырех цифр; - Для вывода времени могут быть использованы следующие элементы: - h - часы в виде одной или двух цифр (0-23); - hh - часы в виде двух цифр (00-23); - m - минуты в виде одной или двух цифр (0-59); - mm - минуты в виде двух цифр (00-59); - s - секунды в виде одной или двух цифр (0-59); - ss - секунды в виде двух цифр (00-59); - z - миллисекунды в виде одной, двух или трех цифр (0-999); - zzz - миллисекунды в виде трех цифр (000-999);

FORMAT_DT	
	- u - микросекунды в виде 0-999999; - uuuuuu - микросекунды в виде шести цифр (000000 to 999999)
IN0	Форматируемое значение
Выходные переменные	
OUT	Строка, содержащая отформатированное значение

FORMAT_DATE	
Форматирование даты	
Входные переменные	
FORMAT	Строка, описывающая формат вывода. Для вывода даты могут быть использованы следующие элементы: - d - день в виде одной или двух цифр (1-31); - dd - день в виде двух цифр (01-31); - M - месяц в виде одной или двух цифр (1-12); - MM - месяц в виде двух цифр (01-12); - YY - год в виде двух цифр (00-99); - YYYY - год в виде четырех цифр
IN0	Форматируемое значение
Выходные переменные	
OUT	Строка, содержащая отформатированное значение

FORMAT_TOD	
Форматирование времени.	
Входные переменные	
FORMAT	Строка, описывающая формат вывода: - h - часы в виде одной или двух цифр (0-23); - hh - часы в виде двух цифр (00-23); - m - минуты в виде одной или двух цифр (0-59); - mm - минуты в виде двух цифр (00-59); - s - секунды в виде одной или двух цифр (0-59); - ss - секунды в виде двух цифр (00-59); - z - миллисекунды в виде одной, двух или трех цифр (0-999);

FORMAT_TOD	
	- zzz - миллисекунды в виде трех цифр (000-999); - u - микросекунды в виде 0-999999; - uuuuuu - микросекунды в виде шести цифр (000000 to 999999)
IN0	Форматируемое значение
Выходные переменные	
OUT	Строка, содержащая отформатированное значение

2.1.2.2. Блоки операций и математических функций

2.1.2.2.1. Операции

ADD	
Сложение. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать.	
Входные переменные	
IN0	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, TIME
IN1	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, TIME
...	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, TIME
INN	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, TIME
Выходные переменные	
OUT	IN0 + IN1 + … + INN, тип совпадает с типом входных переменных

SUB	
Вычитание. Типы входных и выходной переменных должны совпадать	
Входные переменные	
IN0	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, TIME
IN1	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, TIME

SUB	
Выходные переменные	
OUT	IN0 - IN1, тип совпадает с типом входных переменных

MUL	
Умножение. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
IN1	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
...	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
INN	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	IN0 * IN1 * … * INN, тип совпадает с типом входных переменных

DIV	
Деление. Типы входных и выходной переменных должны совпадать. Если IN0, IN1 – целочисленные переменные, то деление целочисленное	
Входные переменные	
IN0	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
IN1	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	IN0 / IN1, тип совпадает с типом входных переменных

MOD	
Остаток от деления нацело. Типы входных и выходной переменных должны совпадать	
Входные переменные	
IN0	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

MOD	
IN1	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Остаток от деления IN0 на IN1

EXPT	
Возведение в степень (IN0^IN1). Типы входных и выходной переменных должны совпадать	
Входные переменные	
IN0	LREAL или REAL
IN1	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	IN0 в степени IN1

NEG	
Унарный минус. Типы входных и выходной переменных должны совпадать	
Входные переменные	
IN0	DINT, INT, LINT, LREAL, REAL, SINT
Выходные переменные	
OUT	-IN0

2.1.2.2.2. Математические функции

ABS	
Абсолютное значение	
Входные переменные	
IN0	DINT, INT, LINT, LREAL, REAL, SINT
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

SQRT
Квадратный корень
Входные переменные

SQRT	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

LN	
Натуральный логарифм	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

LOG	
Десятичный логарифм	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

EXP	
Экспонента (e^IN0). Входная и выходная переменные могут иметь тип LREAL, REAL	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

SIN	
Синус	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

COS	
Косинус	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

TAN	
Тангенс	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

ASIN	
Аркинус	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

ACOS	
Арккосинус	
Входные переменные	
IN0	LREAL, REAL
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

ATAN	
Арктангенс	
Входные переменные	
IN0	LREAL, REAL

ATAN	
Выходные переменные	
OUT	Вычисленное значение, тип совпадает с типом входной переменной

2.1.2.3. Блоки поразрядных действий

2.1.2.3.1. Битовые операции

AND	
Побитовое «И». Входных переменных может быть от 2 до 32 (настраивается в свойствах объекта). Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DWORD, WORD
IN1	BOOL, BYTE, DWORD, WORD
...	BOOL, BYTE, DWORD, WORD
INN	BOOL, BYTE, DWORD, WORD
Выходные переменные	
OUT	IN0 and IN1 and … and INN, тип совпадает с типом входных переменных

OR	
Побитовое «ИЛИ». Входных переменных может быть от 2 до 32 (настраивается в свойствах объекта). Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DWORD, WORD
IN1	BOOL, BYTE, DWORD, WORD
...	BOOL, BYTE, DWORD, WORD
INN	BOOL, BYTE, DWORD, WORD
Выходные переменные	
OUT	IN0 or IN1 or … or INN, тип совпадает с типом входных переменных

XOR
Побитовое « исключающее ИЛИ».

XOR	
Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DWORD, WORD
IN1	BOOL, BYTE, DWORD, WORD
...	BOOL, BYTE, DWORD, WORD
INN	BOOL, BYTE, DWORD, WORD
Выходные переменные	
OUT	IN0 xor IN1 xor … xor INN, тип совпадает с типом входных переменных

NOT	
Побитовое «НЕ»	
Входные переменные	
IN0	BOOL, BYTE, DWORD, WORD
Выходные переменные	
OUT	not IN0, тип совпадает с типом входной переменной

2.1.2.3.2. Операции сдвига

SHL	
Побитовый сдвиг влево	
Входные переменные	
IN0	Сдвигаемое значение, тип BOOL, BYTE, DWORD, WORD
N	Размер сдвига, тип DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Результат, тип BOOL, BYTE, DWORD, WORD

SHR	
Побитовый сдвиг вправо	
Входные переменные	
IN0	Сдвигаемое значение, тип BOOL, BYTE, DWORD, WORD

SHR	
N	Размер сдвига, тип DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Результат, тип BOOL, BYTE, DWORD, WORD

ROL	
Циклический побитовый сдвиг влево	
Входные переменные	
IN0	Сдвигаемое значение, тип BOOL, BYTE, DWORD, WORD
N	Размер сдвига, тип DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Результат, тип BOOL, BYTE, DWORD, WORD

ROR	
Циклический побитовый сдвиг вправо	
Входные переменные	
IN0	Сдвигаемое значение, тип BOOL, BYTE, DWORD, WORD
N	Размер сдвига, тип DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Результат, тип BOOL, BYTE, DWORD, WORD

2.1.2.3.3. Дополнительные блоки

PACK	
Собирает одно BYTE-значение из 8-ми BOOL-значений	
Входные переменные	
IN0	Переменная, значение которой определяет разряд 0 результата
IN1	Переменная, значение которой определяет разряд 1 результата
IN2	Переменная, значение которой определяет разряд 2 результата

PACK	
IN3	Переменная, значение которой определяет разряд 3 результата
IN4	Переменная, значение которой определяет разряд 4 результата
IN5	Переменная, значение которой определяет разряд 5 результата
IN6	Переменная, значение которой определяет разряд 6 результата
IN7	Переменная, значение которой определяет разряд 7 результата
Выходные переменные	
OUT	Результат

UNPACK	
Разбирает одно BYTE-значение на 8 BOOL-значений	
Входные переменные	
IN0	Упакованное значение
Выходные переменные	
OUT0	Переменная, значение которой определяется разрядом 0 входа
OUT1	Переменная, значение которой определяется разрядом 1 входа
OUT2	Переменная, значение которой определяется разрядом 2 входа
OUT3	Переменная, значение которой определяется разрядом 3 входа
OUT4	Переменная, значение которой определяется разрядом 4 входа
OUT5	Переменная, значение которой определяется разрядом 5 входа
OUT6	Переменная, значение которой определяется разрядом 6 входа
OUT7	Переменная, значение которой определяется разрядом 7 входа

EXTRACT	
Получение значения бита	
Входные переменные	
X	Тип DWORD. Число, значение бита которого требуется получить

EXTRACT	
N	Тип INT. Номер получаемого бита (0..31)
Выходные переменные	
OUT	Тип BOOL. Значение извлеченного бита

PUTBIT	
Установка бита	
Входные переменные	
X	Тип DWORD. Число, бит которого требуется установить
N	Тип INT. Номер устанавливаемого бита (0..31)
B	Тип BOOL. Значение устанавливаемого
Выходные переменные	
OUT	Тип DWORD. Число X, бит N которого установлен в значение B

2.1.2.4. Блоки выбора

SEL	
Выбор из двух значений. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
G	Переключатель выбора, BOOL
IN0	Первое возможное значение, тип BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	Второе возможное значение, тип BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	Если G=0, то OUT = IN0, иначе OUT=IN1, тип совпадает с типом входных переменных

MAX
Выбор максимального значения. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать

MAX	
Входные переменные	
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
...	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
INN	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	Максимальное значение среди всех входов; тип совпадает с типом входных переменных

MIN	
Выбор минимального значения. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
...	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
INN	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	Минимальное значение среди всех входов; тип совпадает с типом входных переменных

LIMIT
Выбор с ограничением интервала значений. Типы всех входных и выходной переменных должны совпадать

LIMIT	
Входные переменные	
MN	Нижний предел, тип BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN0	Значение, тип BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
MX	Верхний предел, BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	OUT := MIN(MAX(MN, IN0), MX); тип совпадает с типом входных переменных

MUX	
Множественный выбор. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
K	Переключатель выбора, тип ANY_INT
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
...	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
INN	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	K принимает значения от 0 до N. В зависимости от этого, OUT принимает значения одного из оставшихся входов

MOVE
Копирование входа в выход

MOVE	
Входные переменные	
IN0	Входное значение, тип ANY
Выходные переменные	
OUT	Тип ANY. Копия входного значение. Типы переменных IN0 и OUT должны совпадать

2.1.2.5. Блоки сравнения

GT	
Больше. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
...	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
INN	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	OUT := (IN0 > IN1) and (IN1 > IN2) and …, тип результата BOOL

GE	
Больше или равно. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE

GE	
...	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
INN	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	OUT := (IN0 >= IN1) and (IN1 >= IN2) and …, тип результата BOOL

EQ	
Равно. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
...	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
INN	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	OUT := (IN0 = IN1) and (IN1 = IN2) and …, тип результата BOOL

LE	
Меньше или равно. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE

LE	
...	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
INN	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	OUT := (IN0 <= IN1) and (IN1 <= IN2) and …, тип результата BOOL

LT	
Меньше. Входных переменных может быть от 2 до 16. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
...	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
INN	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	
OUT	OUT := (IN0 < IN1) and (IN1 < IN2) and …, тип результата BOOL

NE	
Не равно. Типы всех входных и выходной переменных должны совпадать	
Входные переменные	
IN0	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
IN1	BOOL, BYTE, DINT, DWORD, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, WORD, STRING, TIME, TOD, DT, DATE
Выходные переменные	

NE	
OUT	OUT := (IN0 <> IN1), тип результата BOOL

2.1.2.6. Блоки работы со строками

2.1.2.6.1. Стандартные блоки

LEN	
Длина строки	
Входные переменные	
IN0	STRING
Выходные переменные	
OUT	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

LEFT	
Подстрока слева	
Входные переменные	
IN0	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	L первых символов из строки IN0, тип STRING

RIGHT	
Подстрока справа	
Входные переменные	
IN0	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	L последних символов из строки IN0, тип STRING

MID
Подстрока из середины

MID	
Входные переменные	
IN0	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
P	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	L символов из строки IN0, начиная с P; тип STRING

CONCAT	
Сцепка строк. Входных переменных может быть от 2 до 16	
Входные переменные	
IN0	STRING
IN1	STRING
...	STRING
INN	STRING
Выходные переменные	
OUT	IN0 + IN1 + … +INN, тип STRING

INSERT	
Вставка одной строки в другую	
Входные переменные	
IN0	STRING
IN1	STRING
P	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Строка IN1 вставляется в строку IN0, начиная с позиции P; тип STRING

DELETE	
Удаление подстроки	
Входные переменные	
IN0	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

DELETE	
P	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Из строки IN0 удаляется фрагмент длины L, начиная с позиции P; тип STRING

REPLACE	
Замена подстроки в строке	
Входные переменные	
IN0	STRING
IN1	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
P	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Из строки IN0 удаляется фрагмент длины L, начиная с позиции P; тип STRING

FIND	
Поиск подстроки в строке	
Входные переменные	
IN0	STRING
IN1	STRING
Выходные переменные	
OUT	В строке IN0 ищется первое вхождение строки IN1. Первый символ имеет номер 1. Если строка не найдена, то OUT принимает значение 0. Тип результата: DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

2.1.2.6.2. Блоки работы со строками в формате UTF8

UTF8_LEN	
Длина строки, хранящейся в формате UTF8	
Входные переменные	
IN0	STRING
Выходные переменные	

UTF8_LEN	
OUT	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

UTF8_LEFT	
Подстрока слева. Исходная строка хранится в формате UTF8	
Входные переменные	
IN0	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	L первых символов из строки IN0, тип STRING

UTF8_RIGHT	
Подстрока справа. Исходная строка хранится в формате UTF8	
Входные переменные	
IN0	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	L последних символов из строки IN0, тип STRING

UTF8_MID	
Подстрока из середины. Исходная строка хранится в формате UTF8	
Входные переменные	
IN0	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
P	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	L символов из строки IN0, начиная с P; тип STRING

UTF8_CONCAT
Сцепка строк, хранящихся в формате UTF8. Входных переменных может быть от 2 до 16
Входные переменные

UTF8_CONCAT	
IN0	STRING
IN1	STRING
...	STRING
INN	STRING
Выходные переменные	
OUT	IN0 + IN1 + … +INN, тип STRING

UTF8_INSERT	
Вставка одной строки в другую. Обе строки хранятся в формате UTF8	
Входные переменные	
IN0	STRING
IN1	STRING
P	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Строка IN1 вставляется в строку IN0, начиная с позиции P; тип STRING

UTF8_DELETE	
Удаление подстроки.Исходная строка хранится в формате UTF8	
Входные переменные	
IN0	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
P	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
OUT	Из строки IN0 удаляется фрагмент длины L, начиная с позиции P; тип STRING

UTF8_REPLACE	
Замена подстроки в строке. Обе строки хранятся в формате UTF8	
Входные переменные	
IN0	STRING
IN1	STRING
L	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
P	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

UTF8_REPLACE	
Выходные переменные	
OUT	В строке IN0 фрагмент длины L, начиная с позиции P, заменяется на строку IN1; тип STRING

UTF8_FIND	
Поиск подстроки в строке. Обе строки хранятся в формате UTF8	
Входные переменные	
IN0	STRING
IN1	STRING
Выходные переменные	
OUT	В строке IN0 ищется первое вхождение строки IN1. Первый символ имеет номер 1. Если строка не найдена, то OUT принимает значение 0. Тип результата: DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

2.1.2.6.3. Прочие блоки

TRIM	
Удаляет пробельные символы в начале и конце строки	
Входные переменные	
IN0	Тип STRING. Исходная строка
Выходные переменные	
OUT	Тип STRING. Строка, в начале и конце которой удалены пробельные символы

2.1.2.7. Триггеры

2.1.2.7.1. Обычные триггеры

SR	
SR - триггер	
Входные переменные	
S	Тип BOOL. Установка триггера
R	Тип BOOL. Сброс триггера

SR	
Выходные переменные	
OUT	Если S=1, то Q=1 вне зависимости от состояния R. Если S=0 и R=1, то Q=0

RS	
RS - триггер	
Входные переменные	
S	Тип BOOL. Установка триггера
R	Тип BOOL. Сброс триггера
Выходные переменные	
OUT	Если R=1, то Q=0 вне зависимости от состояния S. Если S=1 и R=0, то Q=1

2.1.2.7.2. Событийные триггеры

E_SR	
Событийный SR - триггер.	
Входные события	
S	Установка триггера
R	Сброс триггера
Выходные события	
EO	Возникает, когда переменная OUT изменяет значение
Выходные переменные	
OUT	Если S=1, то Q=1 вне зависимости от состояния R. Если S=0 и R=1, то Q=0. Возникновение EO свидетельствует о возникновении входных событий

E_RS	
Событийный RS - триггер	
Входные события	
S	Установка триггера
R	Сброс триггера
Выходные события	
EO	Возникает, когда переменная OUT изменяет значение

E_RS	
Выходные переменные	
OUT	Если R=1, то Q=0 вне зависимости от состояния S. Если S=1 и R=0, то Q=1. Возникновение EO свидетельствует о возникновении входных событий

2.1.2.8. Регистрация фронтов

E_R_TRIG	
Регистратор переднего фронта	
Выходные события	
EO	Возникает, если Q меняет значение с 0 на 1
Входные переменные	
Q	BOOL

E_F_TRIG	
Регистратор заднего фронта	
Выходные события	
EO	Возникает, если Q меняет значение с 1 на 0
Входные переменные	
Q	BOOL

2.1.2.9. Счетчики

E_CTU	
Счетчик в сторону увеличения	
Входные события	
CU	Если возникло событие, то счетчик (CV) увеличивается на 1
R	Сброс счетчика (CV=0, Q=0)
Выходные события	
CUO	Возникает, если изменилось значение CV
RO	Возникает, если счетчик был сброшен
Входные переменные	
PV	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT

E_CTU	
Выходные переменные	
CV	Счетчик, тип DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Q	Q := (CV >= PV), тип BOOL
Тип данных у входной переменной PV должен совпадать с типом данных выходной переменной CV	

E_CTD	
Счетчик в сторону уменьшения	
Входные события	
CD	Если возникло событие, то счетчик (CV) уменьшается на 1
LD	Установка счетчика на верхний предел (CV=PV, Q=0)
Выходные события	
CDO	Возникает, если изменилось значение CV
LDO	Возникает, если счетчик был сброшен
Входные переменные	
PV	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Выходные переменные	
CV	Счетчик, тип DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Q	Q := (CV >= 0), тип BOOL
Тип данных у входной переменной PV должен совпадать с типом данных выходной переменной CV	

E_CTUD	
Двунаправленный счетчик	
Входные события	
CU	Если возникло событие, то счетчик (CV) увеличивается на 1
CD	Если возникло событие, то счетчик (CV) уменьшается на 1
LD	Установка счетчика на верхний предел (CV=PV, Q=0)
R	Сброс счетчика в 0 (CV=0, Q=0)
Выходные события	
CDO	Возникает, если уменьшилось значение CV

E_CTUD	
CUO	Возникает, если увеличилось значение CV
LDO	Возникает, если счетчик был установлен на верхний предел
RO	Возникает, если счетчик был сброшен
Входные переменные	
PV	ANY_INT
Выходные переменные	
CV	Счетчик, тип ANY_INT
Q	Если счетчик увеличивается (приходит событие CU), то Q := (CV >= PV). Если счетчик уменьшается (приходит событие CD), то Q := (CV >= 0). Тип BOOL
Тип данных у входной переменной PV должен совпадать с типом данных выходной переменной CV	

2.1.2.10. Блоки работы с событиями

E_MOVE	
Копирует вход на выход по событию.	
Входные события	
EI	Событие, по которому происходит копирование входа на выход
Входные переменные	
IN0	Тип ANY. Входное значение
Выходные переменные	
OUT	Тип ANY. Копия входного значения

E_MERGE	
Объединение событий. Количество входных событий может быть от 2 до 32 (настраивается в свойствах объекта).	
Входные события	
EI0	Первое из объединяемых событий
EIXX	Первое из объединяемых событий
EI31	Тридцать второе из объединяемых событий
Выходные события	
EO	Возникает, если возникло любое из событий EI0 ... EI31

E_PERMIT	
Ограничение прохождения события.	
Входные события	
EI	Событие, прохождение которого контролируется
Входные переменные	
PERMIT	BOOL
Выходные события	
EO	Если PERMIT=1, то при возникновении события EI возникает EO. Если PERMIT=0, то событие EO не возникает

E_REND	
Регистрация последовательного наступления событий	
Входные события	
EI0	Первое событие
EI1	Второе событие
R	Сброс ожидания второго события
Выходные события	
EO	Возникает, если возникло EI0, а затем EI1, либо сначала EI1, а затем EI0

E_SELECT	
Выбор возникновения одного из двух событий.	
Входные события	
EI0	Первое из событий для выбора
EI1	Второе из событий для выбора
Входные переменные	
G	Тип BOOL.
Выходные события	
EO	Если G=0, то EO возникает при возникновении EI0. EI1 игнорируется. Если G=1, то EO возникает при возникновении EI1. EI0 игнорируется

E_SELECTOR
Выбор возникновения одного из нескольких событий. Количество входных событий может быть от 2 до 32 (настраивается в свойствах объекта).
Входные события

E_SELECTOR	
EI00	Первое из событий для выбора
EI01	Второе из событий для выбора
...	... из событий для выбора
EI31	Тридцать второе из событий для выбора
Входные переменные	
INIT_INDEX	Тип INT. Индекс выходной переменной, которая становится равной TRUE при инициализации кадра
Выходные переменные	
OUT00	Тип BOOL. Становится равной TRUE при появлении события на входе EI00. Все остальные выходные переменные сбрасываются в FALSE
OUTXX	Тип BOOL. Становится равной TRUE при появлении события на входе EIXX. Все остальные выходные переменные сбрасываются в FALSE
OUT31	Тип BOOL. Становится равной TRUE при появлении события на входе EI31. Все остальные выходные переменные сбрасываются в FALSE

E_SWITCH	
Перенаправление события	
Входные события	
EI	Перенаправляемое событие
Входные переменные	
G	Тип BOOL. Переключатель
Выходные события	
EO0	Возникает, если возникло EI и G=0
EO1	Возникает, если возникло EI и G=1

2.1.2.11. Блоки таймеров

TTimer	
Периодическая генерация события	
Входные переменные	
active	Тип BOOL. Если значение равно 1, то таймер генерирует события
period	Тип UDINT. Временной интервал в мс между возникновением событий

TTimer	
Выходные события	
event	Событие таймера
Выходные переменные	
counter	Тип UDINT. Счетчик числа возникших событий

TP	
Импульс	
Входные переменные	
IN0	Тип BOOL. Включает импульс
PT	Тип TIME. Продолжительность импульса
Выходные переменные	
Q	Тип BOOL. Если Q=1, то изменение IN0 игнорируется. Если Q=0, то при установке IN0 в 1 Q принимает значение 1 и остается таковым в течение времени не менее PT. Если IN0 осталось в значении 1 после истечения времени PT, то Q также остается в значении 1
ET	Тип TIME. ET показывает, сколько времени прошло с момента включения IN0 в 1. ET линейно нарастает, пока не станет равным PT, после чего не изменяется. При установке Q в 0 значение ET также сбрасывается в 0

TON	
Задержка включения	
Входные переменные	
IN0	Тип BOOL. Переменная, включение которой задерживается
PT	Тип TIME. Продолжительность задержки
Выходные переменные	
Q	Тип BOOL. При включении IN0 начинается отсчет времени. Как только пройдет время PT, значение IN0 проверяется. Если IN0=0, то ничего не происходит. Если IN0=1, то Q становится равным 1. При сбросе IN0 в 0 происходит сброс значений Q и ET в 0
ET	Тип TIME.

TON	
	ET показывает, сколько времени прошло с момента включения IN0 в 1. ET линейно нарастает, пока не станет равным PT, после чего не изменяется. При установке Q в 0 значение ET также сбрасывается в 0

TOF	
Задержка выключения	
Входные переменные	
IN0	Тип BOOL. Переменная, выключение которой задерживается
PT	Тип TIME. Продолжительность задержки
Выходные переменные	
Q	Тип BOOL. При включении IN0 начинается отсчет времени. Как только пройдет время PT, значение IN0 проверяется. Если IN0=1, то ничего не происходит. Если IN0=0, то Q становится равным 0. При сбросе IN0 в 1 происходит сброс значений Q в 1
ET	Тип TIME. ET сбрасывается в 0, когда In0 переходит из 0 в 1. Отсчет ET начинается с момента сброса IN0 в 0

E_CYCLE	
Периодическая генерация события	
Входные события	
START	Запуск генерации событий
STOP	Остановка генерации событий
Входные переменные	
DT	Тип TIME. Временной интервал между событиями
Выходные события	
EO	Возникает с периодичностью, определяемой переменной DT

E_DELAY
Задержка прохождения события

E_DELAY	
Входные события	
START	Событие, прохождение которого задерживается
STOP	Остановка ожидания прохождения
Входные переменные	
DT	Тип TIME. Задержка
Выходные события	
EO	Возникает после истечения времени DT, следующего за наступлением события START

E_TP	
Импульс заданной длительности	
Входные события	
EI	Событие, формирующее импульс
Входные переменные	
DTV	Тип TIME. Длительность импульса
Выходные события	
EO	Возникает после истечения времени DT, следующего за наступлением события EI
Выходные переменные	
Q	Тип BOOL. Значение равно 1 от момента прихода события EI до истечения времени DTV

MOVE_P	
Передаёт вход на выход с заданной периодичностью	
Входные переменные	
PERIOD	Тип TIME. Периодичность срабатывания
IN0	Тип ANY. Входная переменная
Выходные переменные	
OUT	Тип ANY. Выходная переменная

2.1.2.12. Блоки работы с датой и временем

ADD_DT_TIME	
Увеличивает дату и время на указанное время	
Входные переменные	
IN0	Тип DT. Дата и время

ADD_DT_TIME	
IN1	Тип TIME. Прибавляемая продолжительность
Выходные переменные	
OUT	Тип DT. IN0 + IN1

CURRENT_DATE_TIME	
Возвращает текущие дату и время. Выходные переменные поддерживаются в актуальном состоянии	
Выходные переменные	
OUT_DT	Тип DT. Текущие дата и время
OUT_DATE	Тип DATE. Текущая дата
OUT_TOD	Тип TOD. Текущее время суток

DECODE_DT	
Возвращает компоненты даты и времени	
Входные переменные	
IN0	Тип DT. Дата и время
Выходные переменные	
YEAR	Тип INT. Год
MONTH	Тип INT. Месяц (0..11)
DAY	Тип INT. День (0..30)
WDAY	Тип INT. День недели (0..6), где 0 соответствует воскресенью
YDAY	Тип INT. День в году (0..365)
HOUR	Тип INT. Часы (0..23)
MIN	Тип INT. Минуты (0..59)
SEC	Тип INT. Секунды (0..59)
MSEC	Тип INT. Миллисекунды (0..999)

DECODE_TIME	
Возвращает компоненты интервала времени	
Входные переменные	
IN0	Тип TIME. Временной интервал
Выходные переменные	
DAYS	Тип UDINT. Дни
HOURS	Тип UINT. Часы (0..23)
MINS	Тип UINT. Минуты (0..59)

DECODE_TIME	
SECS	Тип UINT. Секунды (0..59)
USECS	Тип UDINT. Микросекунды (0..999999)

ENCODE_TIME	
Собирает значение интервала времени из компонентов.	
Входные переменные	
DAYS	Тип UDINT. Дни
HOURS	Тип UINT. Часы (0..23)
MINS	Тип UINT. Минуты (0..59)
SECS	Тип UINT. Секунды (0..59)
USECS	Тип UDINT. Микросекунды (0..999999)
Выходные переменные	
OUT	Тип TIME. Временной интервал

SUB_DT_TIME	
Уменьшает дату и время на указанное время.	
Входные переменные	
IN0	Тип DT. Дата и время
IN1	Тип TIME. Убавляемая продолжительность
Выходные переменные	
OUT	Тип DT. IN0 - IN1

TIME_AS_SECONDS	
Преобразует временной интервал в значение продолжительности в секундах	
Входные переменные	
IN0	Тип TIME. Время (продолжительность)
Выходные переменные	
OUT	Тип LREAL. Продолжительность временного интервала в секундах

UTC_TO_LOCAL	
Преобразование времени UTC (всемирное координированное время) в локальное время	
Входные переменные	
IN0	Тип DT. Дата и время в UTC
Выходные переменные	

UTC_TO_LOCAL	
OUT	Тип DT. Локальные дата и время

LOCAL_TO_UTC	
Преобразование локального времени в UTC (всемирное координированное время)	
Входные переменные	
IN0	Тип DT. Локальные дата и время
Выходные переменные	
OUT	Тип DT. Дата и время в UTC

2.1.2.13. Блоки работы с файловой системой

FILE	
Блок работы с файлом	
Входные события	
OPEN	Открыть файл. Путь к файлу задается значением переменной PATH. Режим открытия задается значением переменной MODE. Если файл удалось открыть, то возникает событие OPENED. В этом случае переменная HANDLE получает ненулевое значение. Если файл открыть не удалось, то возникает событие ERROR. Значение переменной HANDLE при этом становится равным 0
READ	Чтение данных из файла. Размер читаемых данных определяется типом переменной OUT_DATA. Если читаются данные типа STRING, то размер читаемых данных (в байтах) определяется значением переменной SIZE. После успешного завершения чтения возникнет событие READ_DONE. Если в ходе чтения произойдет ошибка, то возникнет событие ERROR. Если в ходе чтения будет достигнут конец файла, то переменная EOF примет значение TRUE
READ_LINE	Чтение строки из файла. Переменная OUT_DATA должна иметь тип STRING.

FILE	
	После успешного завершения чтения возникнет событие READ_DONE. При этом будет прочитана строка с завершающими строку символами, но в переменную OUT_DATA попадет строка без них. Если в ходе чтения произойдет ошибка, то возникнет событие ERROR. Если в ходе чтения будет достигнут конец файла, то переменная EOF примет значение TRUE
WRITE	Запись данных в файл. Размер записываемых данных определяется типом переменной DATA. Если записываются данные типа STRING, то будут записаны все символы строки, исключая символ конца строки (символ с кодом 0). После успешного завершения записи возникнет событие WRITTEN. Если в ходе записи произойдет ошибка, то возникнет событие ERROR
WRITE_LINE	Запись строки в файл. Переменная OUT_DATA должна иметь тип STRING. После успешного завершения записи возникнет событие WRITTEN. При этом строка будет записана с символами конца строки и завершающим строку символом (имеет код 0). Если в ходе записи произойдет ошибка, то возникнет событие ERROR
FLUSH	Сбросить данные внутреннего буфера записи на диск. При успешном выполнении операции возникает событие FLUSHED. Если в ходе сброса буфера происходит ошибка, то возникает событие ERROR
CLOSE	Закрыть файл. При успешном закрытии файла возникает событие CLOSED. Если в ходе закрытия файла происходит ошибка, то возникает событие ERROR
Входные переменные	
PATH	Тип STRING. Путь к файлу.
MODE	Тип STRING. Режим открытия файла. ‘r’ – для чтения (‘rb’ для двоичных файлов);

FILE	
	‘w’ – для записи (если существовал старый файл с тем же именем, то он удаляется; ‘wb’ для двоичных файлов); ‘a’ – для дозаписи в конец файла (если файла не было, то он создается; ’ab’ для двоичных файлов)
SIZE	Тип DINT. Количество читаемых байт при чтении строки путем вызова события READ
DATA	Тип ANY_ELEMENTARY. Данные для записи
Выходные события	
OPENED	Файл открыт
READ_DONE	Данные прочитаны
WRITTEN	Данные записаны
FLUSHED	Буфер записи сброшен на диск
CLOSED	Файл закрыт
ERROR	В ходе выполнения операции с файлом возникла ошибка
Выходные переменные	
HANDLE	Тип LINT. Дескриптор файла
EOF	Тип BOOL. Достигнут конец файла
OUT_DATA	Тип ANY_ELEMENTARY. Прочитанные данные

DIR_ITERATOR	
Последовательный обход файлов в каталоге	
Входные события	
FIRST	Выбор первого файла в каталоге
NEXT	Выбор следующего файла в каталоге
Входные переменные	
DIR	Тип STRING. Путь к каталогу
EXT	Тип STRING. Расширение выбираемых файлов
Выходные события	
RESET	Начат обход каталога
READY	Получен путь к файлу
FINISHED	Обход каталога закончен
ERROR	Ошибка обхода каталога
Выходные переменные	

DIR_ITERATOR	
FILE_NAME	Тип STRING. Имя файла из указанного каталога

CRC32	
Расчет CRC32	
Входные события	
CHECK	Расчет CRC32
Входные переменные	
PATH	Тип STRING. Путь к файлу, CRC32 для которого рассчитывается
Выходные переменные	
CRC	Тип UDINT. Значение CRC32
VALID	Тип BOOL. TRUE, если значение CRC32 рассчитано

2.1.2.14. Блоки среды исполнения

2.1.2.14.1. Блоки работы с архивами

ARCHIVE_DATA	
Получает данные из архива сигналов	
Входные события	
GET	Получить данные
Входные переменные	
SIGNAL	Тип STRING. Имя сигнала из архива
DATE_TIME	Тип DT. Дата и время, на которые необходимо извлечь данные
Выходные события	
READY	Данные получены
ERROR	При получении данных возникла ошибка
Выходные переменные	
RES	Тип INT. Результат выполнения запроса (0 – нет ошибок)
DATA	Тип ANY_ELEMENTARY. Возвращенный результат
OUT_DT	Тип DT. Дата и время возвращенного результата

2.1.2.14.2. Блоки работы с сигналами

CUR_DATA	
Получение текущего значения переменной	
Входные переменные	
VAR	Тип STRING. Имя переменной
Выходные переменные	
VALID	Тип BOOL. TRUE, если переменная с именем, заданным значением переменной VAR, найдена и ее значение корректно преобразовано к типу выходной переменной
DATA	Тип ANY. Переменная, в которую будет помещено прочитанное значение

SET_CUR_DATA	
Установка текущего значения переменной. Установка происходит при изменении значения переменной DATA либо при возникновении события SET	
Входные события	
SET	Установка значения
Входные переменные	
VAR	Тип STRING. Имя переменной
DATA	Тип ANY. Переменная, из которой будет прочитано значение
Выходные переменные	
VALID	Тип BOOL. TRUE, если переменная с именем, заданным значением переменной VAR, найдена и ее значение корректно преобразовано к типу задаваемой переменной

DIRECT_GET	
Запрашивает значение сигнала приложения-партнера (данный функциональный блок работает только для системных сигналов, т.е. имя которых начинается с @)	
Входные события	
GET	Осуществляет запрос значения сигнала
Входные переменные	
APP	Тип STRING. Имя ядра-партнера
SIGNAL	Тип STRING. Имя сигнала
CELL_INDEX	Тип UINT. Номер ячейки сигнала (если сигнал представляет собой массив)
TIMEOUT	Тип DINT. Предельное время ожидания ответа (в мс)

DIRECT_GET	
Выходные события	
READY	Возникает, когда обработка запроса окончена
Выходные переменные	
RES	Тип INT. Код ошибки ядра (0, если ошибки нет)
DATA	Тип ANY_ELEMENTARY. Полученное значение (если ошибок не было)

SIGNAL_DATA	
Получение текущего значения, флагов и даты/времени изменения сигнала	
Входные переменные	
NAME	Тип STRING. Имя сигнала
Выходные переменные	
RES	Тип INT. Код ошибки ядра (0, если запрос прошел успешно)
FLAGS	Тип WORD. Флаги сигнала
DATA	Тип ANY_ELEMENTARY. Переменная, в которую будет помещено текущее значение
DATE_TIME	Тип DT. Дата и время изменения сигнала (UTC)

SET_SIGNAL_DATA	
Установка текущего значения, флагов и даты/времени изменения сигнала	
Входные события	
SET	При возникновении данного события происходит установка данных сигнала
Входные переменные	
NAME	Тип STRING. Имя сигнала
VALUE	Тип ANY_ELEMENTARY. Переменная, из которой будет взято текущее значение
FLAGS	Тип WORD. Флаги сигнала
MASK	Тип WORD. Маска сигнала
DATE_TIME	Тип DT. Дата и время изменения сигнала (UTC)
Выходные переменные	
RES	Тип INT. Код ошибки ядра (0, если установка прошла успешно)

VAR_DATA	
На вход данного блока подаётся значение сигнала, а на выходе - копия значения, флаги сигнала и признак достоверности значения.	
Входные переменные	
IN0	Тип ANY. Подаётся сигнал.
Выходные переменные	
VALUE	Тип ANY. Копия значения сигнала, который подан на вход.
FLAGS	Тип UINT. Флаги сигнала.
VALUE_VALID	Тип BOOL. Признак достоверности сигнала: TRUE - достоверен, FALSE - недостоверен.

2.1.2.14.3. Блоки работы с приложениями

APP_INFO	
Возвращает информацию о текущем приложении	
Выходные переменные	
NAME	Тип STRING. Имя приложения

CORE_LATENCY	
Возвращает среднее время доступа к ядру-партнеру данного ядра приложения	
Входные события	
CHECK	При возникновении события производится вычисление среднего времени доступа
Входные переменные	
CORE	Тип STRING. Имя ядра-партнера
INDEX	Тип INT. Индекс сетевого интерфейса (-1, 0, ...). Если значение равно -1, то возвращается наименьшее время среди всех интерфейсов. Если значение 0 и больше, то возвращается время для указанного интерфейса (первый интерфейс имеет индекс 0)
Выходные переменные	
OUT	Тип TIME. Среднее время доступа к ядру-партнеру в сек; -1, если ядро-партнер недоступно

CORE_MONITOR
Следит за состоянием приложения-партнера

CORE_MONITOR	
Входные события	
CHECK	Принудительная проверка
Входные переменные	
CORE	Тип STRING. Имя ядра-партнера
Выходные события	
AVAILABLE	Ядро-партнер стало доступным
UNAVAILABLE	Ядро-партнер стало недоступным
Выходные переменные	
LATENCY	Тип TIME. Среднее время доступа к ядру-партнеру в сек; -1, если ядро-партнер недоступно

2.1.2.14.4. Блоки работы с событиями и тревогами

ALARM	
Управление тревогой	
Входные события	
ACKNOWLEDGE	Квитировать тревогу
Входные переменные	
FLAGS	Тип BYTE. Бит 0 (значение 1 означает отслеживание условия тревоги, данное значение установлено по умолчанию)
CATEGORY	Тип DINT. Категория тревоги (10000..39999)
ID	Тип STRING. Уникальный идентификатор тревоги. Не может быть пустой строкой
CONDITION	Тип BOOL. Условие тревоги (0 - нет тревоги, 1 - есть тревога)
SOURCE	Тип STRING. Источник тревоги
MESSAGE	Тип STRING. Текст сообщения тревоги
TAG	Тип STRING. Дополнительная информация тревоги
Выходные переменные	
STATE	Тип INT. Состояние тревоги (0 - нет тревоги, 1 - есть тревога, 2 - тревога квитирована, 3 - тревога пропала)
ALERT	Тип BOOL. Показывает необходимость индикации тревоги (1, если тревога находится в состояниях 1 или 3)

ALARM_COUNTER	
Счетчик тревог	
Входные переменные	
LOW_CAT	Тип DINT. Нижняя граница категорий
HIGH_CAT	Тип DINT. Верхняя граница категорий
STATE	Тип INT. Состояние тревоги
SOURCE_ID	Тип STRING. ID источника данных о тревоге (относится к работе со смежными проектами и настраивается в Панели инструментов Редактировать - Приложение - Настройка событий и тревог)
Выходные переменные	
OUT	Тип DINT. Количество тревог в интервале категорий [LOW_CAT..HIGH_CAT], находящихся в состоянии STATE

TEventRegistrator	
Регистрация события Sonata	
Входные события	
E_INFO	При возникновении события регистрируется информационное событие Sonata с текстом, взятым из переменной Text
E_WARNING	При возникновении события регистрируется событие – предупреждение Sonata с текстом, взятым из переменной Text
E_CRITICAL	При возникновении события регистрируется критическое событие Sonata с текстом, взятым из переменной Text
E_REGISTER	При возникновении события регистрируется событие Sonata с категорией, состоянием, источником, текстом и дополнительной информацией, взятыми из соответствующих переменных
Входные переменные	
CATEGORY	Тип DINT. Категория события (0..9999)
STATE	Тип INT. Состояние события (0 - исчезновение, 1 - появление)
SOURCE	Тип STRING. Источник события
TEXT	Тип STRING. Текст события
TAG	Тип STRING. Дополнительная информация события

2.1.2.15. Прочие блоки

SYS_EXEC	
Выполнение командной строки	
Входные события	
EXEC	При возникновении события производится выполнение командной строки, заданной путем к приложению и аргументами
Входные переменные	
PATH	Тип STRING. Путь к запускаемому приложению
ARGS	Тип STRING. Аргументы командной строки
SYNC	Тип BOOL. Если TRUE, то вычисление блока завершится только после завершения выполнения приложения, запущенного командной строкой
Выходные переменные	
RES	Тип DINT. Результат выполнения команды

2.1.3. Графические типы функциональных блоков

2.1.3.1. Графические примитивы

2.1.3.1.1. Элементарные объекты

TArc	
Дуга (фрагмент контура эллипса)	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
color	Тип TColor. Цвет объекта
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
frame_width	Тип UINT. Толщина контура объекта
hint	Тип STRING. Всплывающая подсказка объекта

TArc	
moveable	Тип BOOL. Объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
startAngle	Тип LREAL. Начальный угол дуги (в градусах)
style	Тип INT. Стиль контура дуги
sweepLength	Тип LREAL. Длина дуги (в градусах)
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TEllipse	
Эллипс, либо сегмент эллипса	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
bg_color	Тип color. Цвет фона объекта
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
frameStyle	Тип INT. Стиль контура объекта
frame_color	Тип TColor. Цвет контура объекта
frame_width	Тип UINT. Толщина контура объекта
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. Объект можно перемещать

TEllipse	
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
startAngle	Тип LREAL. Начальный угол дуги (в градусах)
spanAngle	Тип LREAL. Длина сегмента (в градусах)
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TLine	
Линия	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
arrowSize	Тип UINT. Размер стрелок
beginArrow	Тип UINT. Стрелка начала линии: 1 - 2 - 3 - 4 -  5 - 6 - 
color	Тип TColor. Цвет объекта
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события

TLine	
endArrow	Тип UINT. Стрелка конца линии. Значения и изображения аналогичны, описанным в параметре beginArrow.
hint	Тип STRING. Всплывающая подсказка объекта
length	Тип UINT. Длина линии
moveable	Тип BOOL. Объект можно перемещать
pos	Тип TPos. Положение объекта
style	Тип INT. Стиль линии
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
width	Тип UINT. Толщина линии
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TPolygon	
Объект - многоугольник	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
bg_color	Тип TColor. Цвет фона объекта
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
frameStyle	Тип INT. Стиль контура объекта
frame_color	Тип TColor. Цвет контура объекта
frame_width	Тип UINT. Толщина контура объекта

TPolygon	
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. Объект можно перемещать
pointCount	Тип UINT. Количество вершин многоугольника
points	Массив из 32 элементов типа TPos. Вершины многоугольника
pos	Тип TPos. Положение объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TRect	
Прямоугольник	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
bg_color	Тип TColor. Цвет фона объекта
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
frameStyle	Тип INT. Стиль контура объекта
frame_color	Тип TColor. Цвет контура объекта
frame_width	Тип UINT. Толщина контура объекта
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. Объект можно перемещать

TRect	
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TText	
Текстовый объект. В данном текстовом объекте есть возможность использовать тэги html.	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
font	Тип TFont. Шрифт текста
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. Объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
text	Тип STRING. Строка текста. В данной переменной допустимо использование тэгов html, таких как и другие
text_color	Тип TColor. Цвет текста
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	

TText	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TSimpleText	
Текстовый объект.	
Входные переменные	
alignment	Тип TAlignment. Выравнивание текста
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
font	Тип TFont. Шрифт текста
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. Объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
text	Тип STRING. Строка текста
text_color	Тип TColor. Цвет текста
visible	Тип BOOL. Признак видимости объекта
wordWrap	Тип BOOL. Необходимость переноса текста по словам
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте

TSimpleText	
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

THugePolyline	
Объект - ломаная линия	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
frameStyle	Тип INT. Стиль контура объекта
frame_color	Тип TColor. Цвет контура объекта
frame_width	Тип UINT. Толщина контура объекта
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. Объект можно перемещать
pointCount	Тип UINT. Количество вершин многоугольника
X	Тип REAL[512]. Значения по оси абсцисс
Y	Тип REAL[512]. Значения по оси ординат
pos	Тип TPos. Положение объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте

THugePolyline	
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

2.1.3.1.2. Изображения

TGraphics	
Объект для отображения картинки в формате SVG	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
hint	Тип STRING. Всплывающая подсказка объекта
image	Тип TImage. Изображение
itemId	Тип STRING. Идентификатор отображаемого объекта
moveable	Тип BOOL. Объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
animated	Тип BOOL. TRUE, если изображение анимированное
FPS	Тип INT. Количество кадров в секунду для анимированных изображений
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте

TGraphics	
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TMovie	
Объект для отображения анимированных gif	
Входные события	
START	Начать анимацию
STOP	Остановить анимацию
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
moveable	Тип BOOL. Объект можно перемещать
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
DATA	Тип TImage. Имя объекта анимации
CYCLE	Тип BOOL. Если TRUE, то зацикливание анимации по кругу, если FALSE, то нет зацикливания. Это работает в случае, если gif сам не циклический
SPEED	Тип INT. Скорость анимации в процентах (диапазон от 0 до 100 %)
Выходные события	
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта

TMovie	
mouseLeave	Выход указателя «мыши» за пределы объекта
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте

TPixmap	
Объект для отображения картинки в формате BMP, PNG, JPEG * нельзя изменять размеры данного объекта	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
hint	Тип STRING. Всплывающая подсказка объекта
image	Тип TImage. Изображение
moveable	Тип BOOL. Объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте.
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TPixmapGroup
Объект для отображения картинки в формате BMP, PNG, JPEG * можно изменять размеры данного объекта
Входные переменные

TPixmapGroup	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать выходные события
hint	Тип STRING. Всплывающая подсказка объекта
images	Тип TImage[16]. Массив изображений
index	Тип INT. Индекс показываемого изображения
moveable	Тип BOOL. Объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки«мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

2.1.3.2. Элементы управления

2.1.3.2.1. Редакторы

2.1.3.2.1.1. Целочисленные типы

TINTEditor, TDINTEditor	
Редакторы значений типов INT (DINT)	
Входные события	
reset	Устанавливает значение в редакторе равным значению входной переменной i_value
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
font	Тип TFont. Шрифт текста
hint	Тип STRING. Вспывающая подсказка объекта
i_value	Тип INT (DINT). Входное значение
moveable	Тип BOOL. TRUE, если объект перемещаемый
maximum	Тип INT (DINT). Максимальное значение
minimum	Тип INT (DINT). Минимальное значение
pos	Тип TPos. Положение объекта
prefix	Тип STRING. Префикс значения
size	Тип TSize. Размер объекта
suffix	Тип STRING. Суффикс значения
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте

TINTEditor, TDINTEditor	
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
edited	Возникает, если пользователь изменил значение в редакторе
Выходные переменные	
o_value	Тип INT (DINT). Выходное значение

2.1.3.2.1.2. Вещественные типы

TRealEditor, TDoubleEditor	
Редакторы значений типов REAL (LREAL).	
Входные события	
reset	Устанавливает значение в редакторе равным значению входной переменной i_value
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
decimals	Тип INT. Количество знаков после запятой
font	Тип TFont. Шрифт текста
hint	Тип STRING. Всплывающая подсказка объекта
i_value	Тип REAL (LREAL). Входное значение
moveable	Тип BOOL. TRUE, если объект перемещаемый
maximum	Тип REAL (LREAL). Максимальное значение
minimum	Тип REAL (LREAL). Минимальное значение
single_step	Тип LREAL. Шаг изменения значения в редакторе (данная входная переменная используется только в версии 1.4)
pos	Тип TPos. Положение объекта
prefix	Тип STRING. Префикс значения
size	Тип TSize. Размер объекта
style	Тип INT. Стиль редактора: 0 - обычный редактор, 1 - редактор, предназначенный

TRealEditor, TDoubleEditor	
	для панельных компьютеров, не имеющих клавиатуры
suffix	Тип STRING. Суффикс значения
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
edited	Возникает, если пользователь изменил значение в редакторе
Выходные переменные	
o_value	Тип REAL (LREAL). Выходное значение

2.1.3.2.1.3. Дата и время

TDateTimeEditor, TDateEditor	
Редакторы значений типов DT (DATE)	
Входные события	
reset	Устанавливает значение в редакторе равным значению входной переменной i_value
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
font	Тип TFont. Шрифт текста
hint	Тип STRING. Вспывающая подсказка объекта
i_value	Тип DT (DATE). Входное значение

TDateTimeEditor, TDateEditor	
moveable	Тип BOOL. TRUE, если объект перемещаемый
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
edited	Возникает, если пользователь изменил значение в редакторе
Выходные переменные	
o_value	Тип DT (DATE). Выходное значение

2.1.3.2.1.4. Продолжительнсть

TTIMEEditor	
Редактор значения типа TIME.	
Входные события	
reset	Устанавливает значение в редакторе равным значению входной переменной i_value
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах.
enabled	Тип BOOL. Задает разрешенность объекта
font	Тип TFont. Шрифт текста
hint	Тип STRING. Вспывающая подсказка объекта

TTIMEEditor	
i_value	Тип TIME. Входное значение
moveable	Тип BOOL. TRUE, если объект перемещаемый
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
edited	Возникает, если пользователь изменил значение в редакторе
Выходные переменные	
o_value	Тип TIME. Выходное значение

2.1.3.2.1.5. Строки

TSTRINGEditor	
Редактор значения типа STRING	
Входные события	
reset	Устанавливает значение в редакторе равным значению входной переменной i_value
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
font	Тип TFont. Шрифт текста

TSTRINGEditor	
hint	Тип STRING. Вспывающая подсказка объекта
i_value	Тип STRING. Входное значение
moveable	Тип BOOL. TRUE, если объект перемещаемый
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
edited	Возникает, если пользователь изменил значение в редакторе
Выходные переменные	
o_value	Тип STRING. Выходное значение

TTextEditor	
Многострочный редактор текста	
Входные события	
reset	Устанавливает значение в редакторе равным значению входной переменной i_value
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый

TTextEditor	
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Вспывающая подсказка объекта
size	Тип TSize. Размер объекта
font	Тип TFont. Шрифт текста
i_value	Тип STRING. Входное значение
Выходные события	
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
Выходные переменные	
o_value	Тип STRING. Выходное значение

2.1.3.2.2. Стандартные элементы

TButton	
Кнопка Данный элемент объявлен устаревшим и исключён из дерева библиотечных объектов! Вместо него необходимо пользоваться блоком TTLButton!	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
color	Тип TColor. Цвет кнопки
enabled	Тип BOOL. Задает разрешенность объекта
font	Тип TFont. Шрифт объекта
hint	Тип STRING. Всплывающая подсказка объекта

TButton	
moveable	Тип BOOL. TRUE, если объект перемещаемый
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
tag	Тип INT. Дополнительные данные кнопки
text	Тип STRING. Строка текста
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
o_clicked	Событие нажатия кнопки
Выходные переменные	
o_tag	Тип INT. Копия значения tag

TTree	
Древовидный объект	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
font	Тип TFont. Шрифт объекта
hint	Тип STRING. Всплывающая подсказка объекта
initial_name	Тип STRING. Имя изначально выделенного элемента
initial_tag	Тип INT. Тэг изначально выделенного элемента

TTree	
initially_expanded	Тип BOOL. Если равно 1, то все узлы дерева при инициализации раскрываются
moveable	Тип BOOL. TRUE, если объект перемещаемый
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
title	Тип STRING. Заголовок дерева
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши»за пределы объекта
o_changed	Возникает при изменении состояния выделения элемента дерева
Выходные переменные	
o_current_data	Тип STRING. Данные выделенного элемента
o_current_name	Тип STRING. Имя выделенного элемента
o_current_tag	Тип INT. Тэг выделенного элемента

TComboBox	
Выпадающий список строк	
Входные события	
RESET	Событие сброса. Выходные переменные сбрасываются на значение входных переменных
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах

TComboBox	
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. Задает возможность перемещать объект во время выполнения
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Определяет текст всплывающей подсказки объекта
size	Тип TSize. Размер объекта
font	Тип TFont. Шрифт объекта
I_INDEX	Тип INT. Индекс элемента, который должен быть выбран
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
CHANGED	Текущая строка в списке изменена пользователем или программным путем
edited	Возникает, если пользователь изменил значение в редакторе
Выходные переменные	
O_INDEX	Тип INT. Номер текущей выбранной строки или -1, если ничего не выбрано
O_TEXT	Тип STRING. Текст текущей выбранной строки

TList	
Список строк	
Входные события	
ADD	Добавление новой строки в нижнюю часть списка.

TList	
	В качестве текста вставляемой строки используется значение переменной TEXT. После завершения добавления возникнет выходное событие ADDED
INSERT	Вставка новой строки в список. Номер строки определяется значением переменной INDEX. Номер должен находиться в пределах [0..количество строк]. В качестве заголовка вставляемой строки используется значение переменной TEXT. После успешного завершения вставки возникнет выходное событие INSERTED
SET	Установка текста строки списка. Номер строки определяется значением переменной INDEX. Номер должен находиться в пределах [0..количество строк). Устанавливаемый текст определяется значением переменной TEXT. После успешного завершения выделения возникнет выходное событие SET_DONE
REMOVE	Удаление строки из списка. Номер строки определяется значением переменной INDEX. Номер должен находиться в пределах [0..количество строк). После успешного завершения удаления возникнет выходное событие REMOVED
CLEAR	Удаляет все строки списка. После завершения очистки возникнет выходное событие CLEARED
SELECT	Выделение строки списка. Номер строки определяется значением переменной INDEX. Номер должен находиться в пределах [0..количество строк). После успешного завершения выделения возникнет выходное событие SELETED. При этом будут установлены значения выходных переменных CUR_TEXT, CUR_INDEX
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. Задает возможность перемещать объект во время выполнения
visible	Тип BOOL. Признак видимости объекта

TList	
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Определяет текст всплывающей подсказки объекта
size	Тип TSize. Размер объекта
font	Тип TFont. Шрифт объекта
TEXT	Тип STRING. Текст строки
INDEX	Тип INT. Номер строки
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
ADDED	Строка добавлена
INSERTED	Строка вставлена
SET_DONE	Установлен текст строки
REMOVED	Строка удалена
CLEARED	Список очищен
SELECTED	Выбрана строка списка
DONE	Операция модификации списка завершена
CHANGED	Изменена текущая строка списка
Выходные переменные	
CUR_TEXT	Текст выделенной строки
CUR_INDEX	Номер выделенной строки
COUNT	Количество строк списка

TTable	
Таблица строк	
Входные события	
ADD_ROW	Добавление новой строки в нижнюю часть таблицы.

TTable	
	В качестве заголовка вставляемой строки используется значение переменной TEXT. После завершения добавления возникнет выходное событие ROW_ADDED
INSERT_ROW	Вставка новой строки в таблицу. Номер строки определяется значением переменной ROW. Номер должен находиться в пределах [0..количество строк]. В качестве заголовка вставляемой строки используется значение переменной TEXT. После успешного завершения вставки возникнет выходное событие ROW_INSERTED. Если индекс находится за пределами допустимого диапазона, то возникнет событие ERROR
REMOVE_ROW	Удаление строки из таблицы. Номер строки определяется значением переменной ROW. Номер должен находиться в пределах [0..количество строк). После успешного завершения удаления возникнет выходное событие ROW_REMOVED. Если индекс находится за пределами допустимого диапазона, то возникнет событие ERROR
REMOVE_ ROWS	Удаляет все строки таблицы. После завершения очистки возникнет выходное событие ROWS_REMOVED.
SET	Установка текста ячейки таблицы. Номер строки определяется значением переменной ROW. Номер должен находиться в пределах [-1..количество строк). Номер столбца определяется значением переменной COLUMN. Номер должен находиться в пределах [-1..количество столбцов). Устанавливаемый текст определяется значением переменной TEXT. Если номер строки равен -1, то текст будет присвоен заголовку колонки, определяемой значением переменной COLUMN. Если номер столбца равен -1, то текст будет присвоен заголовку строки, определяемой значением переменной ROW. После успешного завершения выделения возникнет выходное событие SET_DONE. Если адрес выделяемой ячейки находится за пределами допустимого диапазона, то возникнет событие ERROR

TTable	
SELECT	Выделение ячейки таблицы. Номер строки определяется значением переменной ROW. Номер должен находиться в пределах [0..количество строк). Номер столбца определяется значением переменной COLUMN. Номер должен находиться в пределах [0..количество столбцов). После успешного завершения выделения возникнет выходное событие SELETED. При этом будут установлены значения выходных переменных CURRENT_TEXT, CURRENT_ROW, CURRENT_COLUMN. Если адрес выделяемой ячейки находится за пределами допустимого диапазона, то возникнет событие ERROR. При этом значения переменных CURRENT_ROW, CURRENT_COLUMN примут значения -1, а значение переменной CURRENT_TEXT станет пустой строкой
GET_CONTENT	Получение содержимого таблицы в формате CSV. Данные будут выведены в выходную переменную CONTENT. После окончания получения данных возникнет выходное событие CONTENT_READY
PRINT	Печать таблицы
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. Задает возможность перемещать объект во время выполнения
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Определяет текст всплывающей подсказки объекта
size	Тип TSize. Размер объекта. Состоит из 2ух полей: - size.width - ширина таблицы (единицами измерения является средняя ширина символа шрифта);

TTable	
	- size.height - высота таблицы (измеряется в пикселях)
font	Тип TFont. Шрифт объекта
TEXT	Тип STRING. Текст элемента
TEXT_COLOR	Тип TColor. Цвет текста элемента
ROW	Тип INT. Номер строки
COLUMN	Тип INT. Номер колонки
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
CELL_CLICKED	Возникает при клике левой кнопкой «мыши» на ячейке таблицы
ROW_ADDED	Строка добавлена
ROW_INSERTED	Строка вставлена
ROW_REMOVED	Строка удалена
ROWS_REMOVED	Удалены все строки из таблицы
SELECTED	Выбрана ячейка таблицы
SET_DONE	Установлен текст ячейки
CONTENT_READY	Содержимое таблицы в формате CSV подготовлено
ERROR	При выполнении операции возникла ошибка
Выходные переменные	
ROW_COUNT	Количество строк таблицы
COLUMN_COUNT	Количество столбцов таблицы
CURRENT_TEXT	Текст выделенной ячейки
CURRENT_ROW	Номер строки выделенной ячейки
CURRENT_COLUMN	Номер столбца выделенной ячейки
CONTENT	Содержимое таблицы в формате CSV

TTable	
ERROR_INFO	Информация об ошибке, возникшей в ходе выполнения какой-либо операции

Изменение содержимого таблицы может производиться следующими способами:

- в ходе разработки графического приложения;
- в ходе работы при помощи модифицирующих событий (ADD_ROW, SET и т.п.);
- в ходе работы при помощи прямого доступа к отдельным ячейкам таблицы.

Диалог настройки, используемый для первоначального формирования содержимого таблицы, а также для открытия доступа к отдельным ячейкам таблицы, приведен на рис. 2.1. В ходе настройки таблицы разработчик формирует перечень столбцов,строк, а также производит настройку отдельных ячеек.

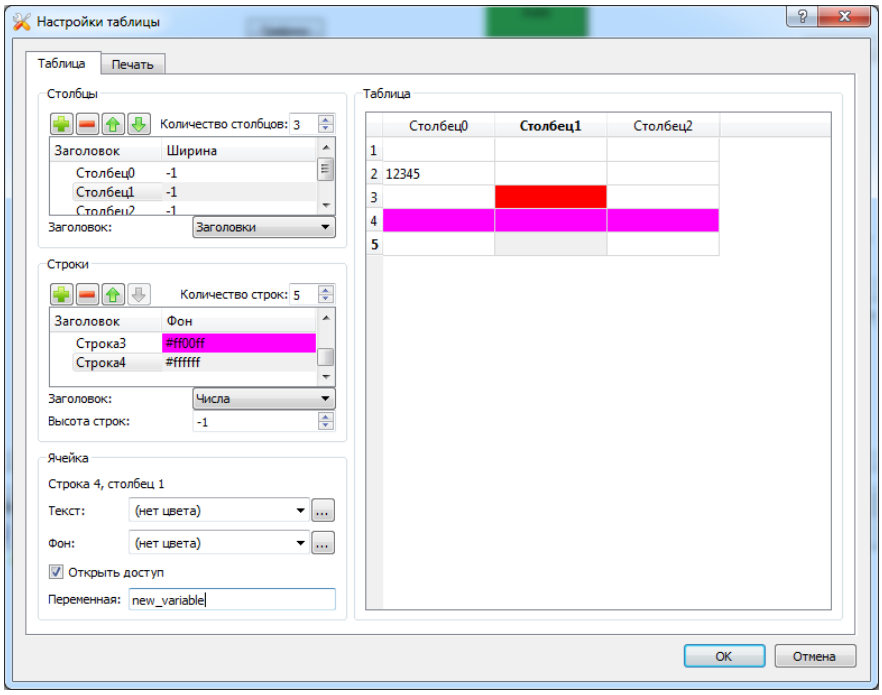


Рисунок 2.1 - Окно настройки свойств объекта TTable

Элементы управления, предназначенные для настройки столбцов, расположены в верхней левой части диалога. Кнопки и предназначены для изменения количества столбцов, а кнопки и - для изменения их порядка. Помимо перечисленных кнопок, количество столбцов может быть задано вручную при помощи соответствующего редактора, расположенного справа от кнопок.

Таблица столбцов, расположенная под кнопками, позволяет разработчику задать имена и ширины столбцов. Значение ширины, приведенное в таблице, задается в символах. Таким образом, в столбце, ширина которого равна 10, можно будет поместить примерно 10 символов. Также ширину столбца можно задать в поле просмотра таблицы, расположенного в правой части диалога.

Под таблицей столбцов расположен выпадающий список, в котором разработчик выбирает способ отображения заголовка столбца. Возможны следующие варианты отображения:

- **Нет заголовка** - заголовок не отображается;
- **Числа** - заголовки отображаются в виде порядкового номера столбца;
- **Заголовки** - отображаются заголовки, указанные при формировании списка столбцов.

Элементы управления, предназначенные для настройки строк, расположены в средней левой части диалога. Кнопки и предназначены для изменения количества строк, а кнопки и - для изменения их порядка. Помимо перечисленных кнопок, количество строк может быть задано вручную при помощи соответствующего редактора, расположенного справа от кнопок.

TTable

Таблица строк, расположенная под кнопками, позволяет разработчику задать имена и цвета строк. Цвет строки может быть введен вручную, выбран из выпадающего списка или из палитры.

Под таблицей строк расположен выпадающий список, в котором разработчик выбирает способ отображения заголовка строки. Возможны следующие варианты отображения:

- **Нет заголовка** - заголовок не отображается;
- **Числа** - заголовки отображаются в виде порядкового номера строки;
- **Заголовки** - отображаются заголовки, указанные при формировании списка строк.

Ниже располагается настройка высоты строк таблицы. При значении -1 выставляется высота по умолчанию.

В нижней левой части диалога расположены элементы управления, позволяющие разработчику задать цвет текста и цвет фона ячейки таблицы, а также открыть доступ к выбранной ячейке извне. Текст в выбранную ячейку вводится в поле просмотра таблицы, расположенное в правой части диалога. Цвет фона ячейки таблицы является приоритетным по отношению к цвету фона строки. Это означает, что если для строки в целом задан какой-то цвет, но для определенной ячейки задан другой цвет, то вся строка, а также ячейки, для которых цвет не задан, будут отрисованы цветом строки, но цвет отдельной ячейки будет отличаться.

Если разработчик укажет в настройках ячейки необходимость открытия доступа, то в интерфейсе объекта - таблицы появится дополнительная входная переменная типа TCell. Имя переменной задается в настройках ячейки и должно быть уникальным среди всех переменных и событий объекта-таблицы.

В качестве начального значения переменной, связанной с ячейкой, используются значения текста, цвета текста и цвета фона, заданные для данной ячейки при ее настройке.

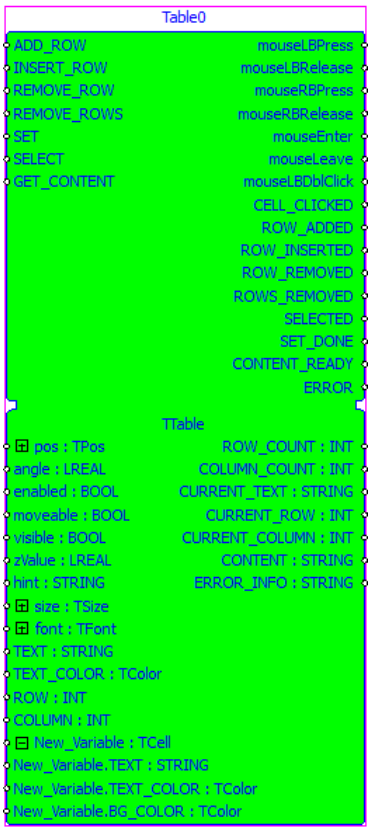


Рисунок 2.2 - Функциональный блок TTable с открытым доступом к одной из ячеек таблицы на диаграмме в програме IECEditor

TTable	
TCell	
Ячейка таблицы	
Входные переменные	
TEXT	Текст отображаемый в ячейке. Тип STRING
TEXT_COLOR	Цвет текста. Тип TColor
BG_COLOR	Цвет фона. Тип TColor

Окно настройки печати таблицы приведено на рис. 2.3.

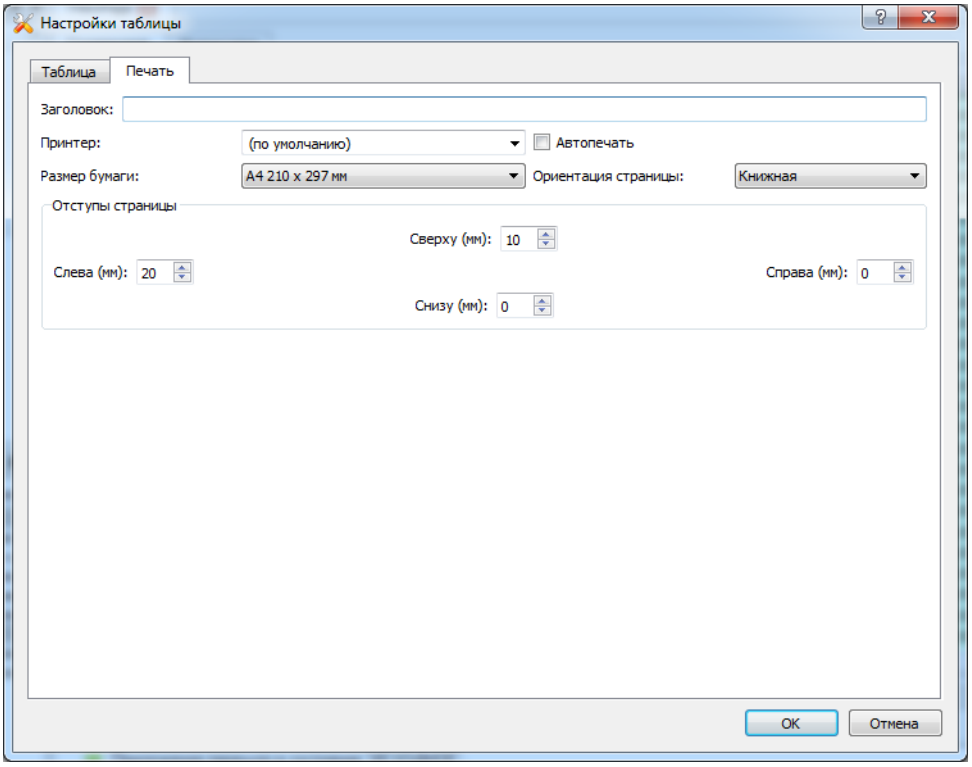


Рисунок 2.3 - Окно настроек печати таблицы

Поле ввода заголовка служит для ввода текста, выводимого над таблицей в ходе печати.

Выпадающий список "Принтер" предназначен для выбора устройства, которое будет выбрано в качестве предлагаемого принтера для печати. Первым элементом выпадающего списка является строка "(по умолчанию)", при выборе которой в качестве принтера для печати будет предложен принтер по умолчанию того компьютера, на котором будет выполняться печать. Имя принтера может быть выбрано как из выпадающего списка, так и введено вручную.

Если включить опцию "Автопечать", то блок не выведет диалог печати, а сразу же отправит данные на выбранный принтер. Если же данная опция отключена, то при вызове события печати блок выведет системный диалог, в котором можно будет выбрать принтер и указать требуемые параметры печати.

Ниже выпадающего списка выбора принтера расположены выпадающие списки для выбора размера и ориентации бумаги.

В нижней части диалога расположены элементы редактирования, предназначенные для задания отступов от края страницы. Значения задаются в миллиметрах.

2.1.3.2.3. Прочие элементы

2.1.3.2.3.1. Объекты меню

TMainMenu	
Главное меню видеоменеджера	
Входные переменные	
enabled	Массив из 32 элементов типа BOOL. Определяют доступность опций
font	Тип TFont. Шрифт меню
visible	Массив из 32 элементов типа BOOL. Определяют видимость опций
Выходные события	
select	Возникает, когда выбрана какая-либо опция меню
Выходные переменные	
itemId	Тип INT. Id выбранной опции меню

TPopupMenu	
Контекстое меню	
Входные события	
exec	При возникновении события показывается меню
Входные переменные	
enabled	Массив из 32 элементов типа BOOL. Определяют доступность опций
font	Тип TFont. Шрифт меню
visible	Массив из 32 элементов типа BOOL. Определяют видимость опций
Выходные события	
select	Возникает, когда выбрана какая-либо опция меню
Выходные переменные	
itemId	Тип INT. Id выбранной опции меню

2.1.3.2.3.2. Графики

TGraph	
Объект, содержащий множество графиков	
Входные события	
ADD_POINT	Добавление новой точки в график. В качестве номера графика берется значение переменной INDEX. В качестве координат точки берутся значения переменных X (X_DT), Y. После завершения добавления возникнет выходное событие POINT_ADDED
CLEAR	Очистка графика. В качестве номера графика берется значение переменной INDEX. После завершения добавления возникнет выходное событие CLEARED
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. Задает возможность перемещать объект во время выполнения
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Определяет текст всплывающей подсказки объекта
size	Тип TSize. Размер объекта
font	Тип TFont. Шрифт объекта
INDEX	Тип INT. Номер модифицируемого графика
X	Тип LREAL. X-координата добавляемой точки, если ось X-вещественная
X_DT	Тип DT. X-координата добавляемой точки, если ось X-дата и время
Y	Тип LREAL. Y-координата добавляемой точки
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте

TGraph	
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
POINT_ADDED	Точка добавлена
CLEARED	График очищен
ERROR	При попытке модификации графика возникла ошибка (значение переменной INDEX находится за пределами диапазона индексов графиков)

Диалог настройки, используемый для конфигурации свойств объекта TGraph, а также для открытия доступа к осям и графикам, представлен на рис. 2.4. В ходе настройки графиков разработчик настраивает отображение осей, доабавляет графики, настраивает внешний вид графиков, настраивает размер буфера для хранения данных графиков.

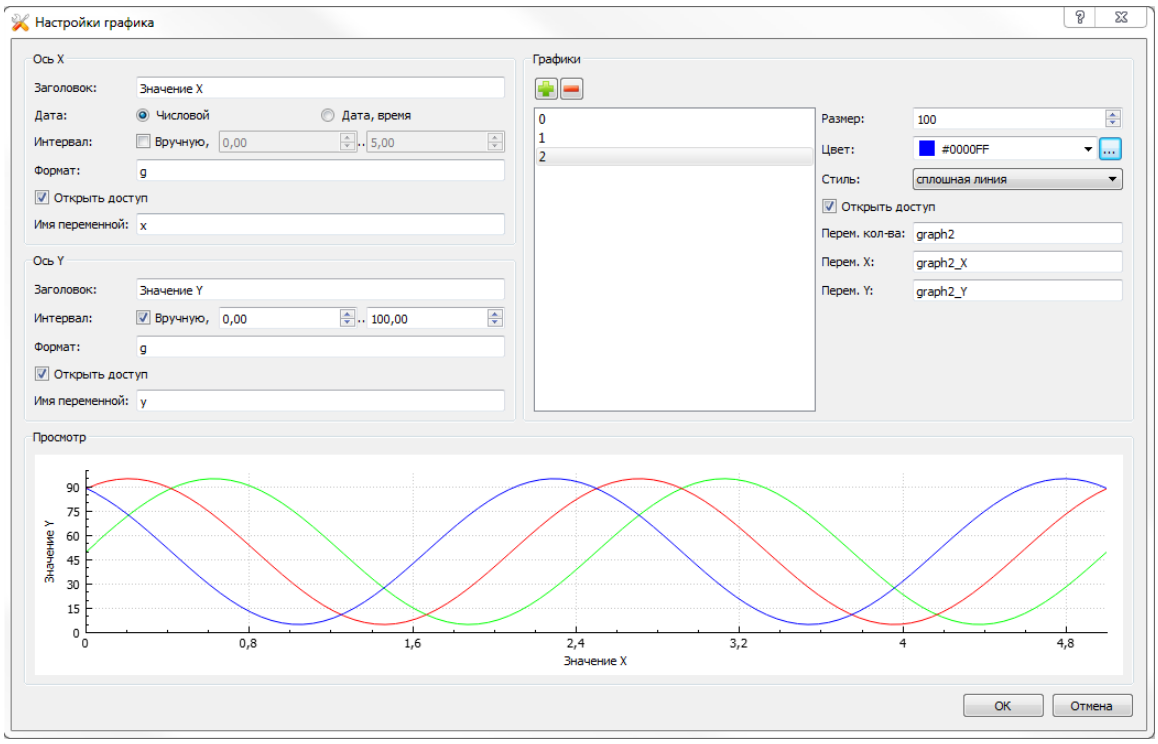


Рисунок 2.4 - Окно настройки свойств объекта TGraph

Элементы управления, предназначенные для настройки оси X, расположены в верхней левой части диалога. Для настроек имеются следующие параметры:

- заголовок - определяет подпись под осью X;

TGraph

- дата - определяет вариант отображение данных по оси X - числовые значения или дата, время;

- интервал - по умолчанию задается автоматически, но у разработчика есть возможность задать вручную, указав необходимый интервал в соответствующих полях;

- формат - формат отображения числовых данных по оси X. Всего возможно три варианта **g** (General), **e** (Exponential) и **f** (Fixed). Формат **e** отображает числа в виде десятичной дроби с шестью нулями после запятой, умноженной на десять в соответствующей степени. Например, 1.000000e2, **f** в виде десятичной дроби с шестью знаками после запятой. Например, 100.000000, формат **g** может отображать как **e** и как **f** в зависимости от ситуации, но в отличии от них выводит после запятой только значимые знаки;

- открыть доступ – если разработчик укажет в настройках оси необходимость открытия доступа к оси, то в интерфейсе объекта TGraph появится дополнительная входная переменная типа TAxisLREAL. Имя переменной задается в настройках оси и должно быть уникальным среди всех переменных и событий объекта.

Рисунок 2.5 - Объект TGraph на диаграмме в программе IECEditor

TGraph

Элементы управления, предназначенные для настройки оси Y, расположены в средней левой части диалога. Для настроек имеются следующие параметры:

- заголовок - это поле позволяет разработчику задать подпись под осью X;
- интервал - по умолчанию задается автоматически, но у разработчика есть возможность задать вручную, указав необходимый интервал в соответствующих полях;
- формат - формат отображения числовых данных по оси X. Всего возможно три варианта **g** (General), **e** (Exponential) и **f** (Fixed). Формат **e** отображает числа в виде десятичной дроби с шестью нулями после запятой, умноженной на десять в соответствующей степени. Например, 1.000000e2, **f** в виде десятичной дроби с шестью знаками после запятой. Например, 100.000000, формат **g** может отображать как **e** и как **f** в зависимости от ситуации, но в отличии от них выводит после запятой только значимые знаки;
- открыть доступ – если разработчик укажет в настройках оси необходимость открытия доступа к оси, то в интерфейсе объекта TGraph появится дополнительная входная переменная типа TAxisLREAL. имя переменной задается в настройках оси и должно быть уникальным среди всех переменных и событий объекта.

В правой верхней части находятся настройки графиков. Кнопки и предназначены для изменения количества графиков.

Справа от таблицы графиков находятся следующие настройки отображения графиков. У разработчика есть возможность изменить размер буфера для хранения данных графика, который по умолчанию равен 100 точкам. Цвет графика может быть введен вручную или выбран из выпадающего списка или палитры.

Ниже расположен выпадающий список, в котором разработчик выбирает стиль отображения графика. Возможны следующие варианты отображения:

- нет линии;
- сплошная линия;
- пунктирная линия;
- точечная линия;
- штрих-пунктирная линия;
- штрих-штрих-пунктирная линия .

Если разработчик укажет в настройках графика необходимость открыть доступ, то в интерфейсе объекта появятся три дополнительные входные переменные. Переменная количества хранит информацию о том, сколько точек в буфере на данный момент и имеет тип DINT. Например, при размере буфера 100 в каждый момент времени в буфере хранятся последние 100 точек. Переменная количества будет по мере заполнения буфера меняться от 0 до 100. После заполнения буфера переменная количества не меняется. Переменные X и Y представляют из себя массивы типа LREAL для хранения данных по оси X и Y соответственно с размером, указанным в поле "Размер".

TAxisLREAL	
Ось графика	
Входные переменные	
TITLE	Тип STRING. Название оси
FORMAT	Тип STRING. Формат отображения данных оси
RANGE_MANUAL	Тип BOOL. Индикатор, сообщающий, задавать ли диапазон данных вручную
RANGE_MIN	Тип LREAL. Минимум оси

TGraph	
TAxisLREAL	
RANGE_MAX	Тип LREAL. Максимум оси

2.1.3.2.3.3. Просмотр HTML-документа

THTMLViewer	
Объект просмотра HTML-документа	
Входные события	
PRINT	Печать документа
PREVIEW	Предварительный просмотр документа перед печатью
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. Задает возможность перемещать объект во время выполнения
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Определяет текст всплывающей подсказки объекта
size	Тип TSize. Размер объекта
font	Тип TFont. Шрифт объекта
DEFAULT_ENCODING	Тип STRING. Кодировка HTML-документа по умолчанию
PATH	Тип STRING. Путь к HTML-документу
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте

THTMLViewer	
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
Настройка	

Окно настройки объекта приведено на рис. 2.6

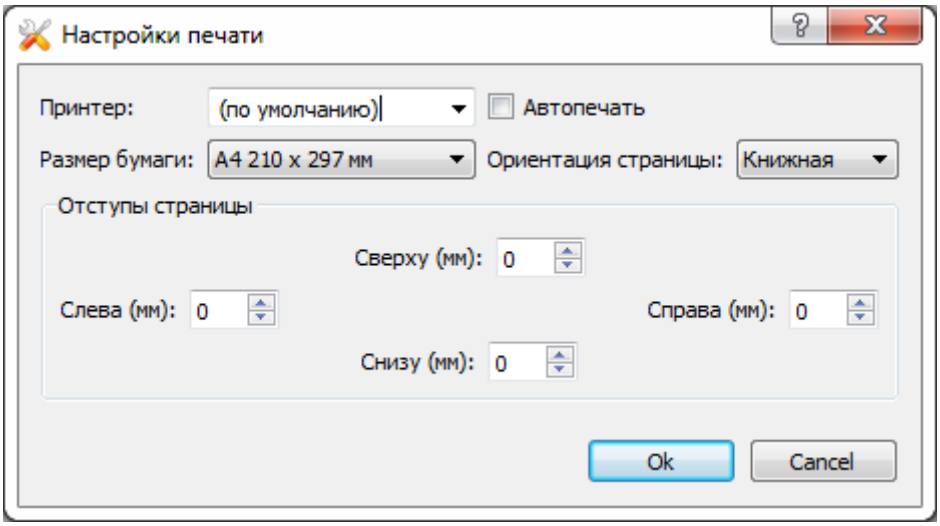


Рисунок 2.6 - Окно диалога настройки блока просмотра HTML"

Выпадающий список "Принтер" предназначен для выбора устройства, которое будет выбрано в качестве предлагаемого принтера для печати. Первым элементом выпадающего списка является строка "(по умолчанию)", при выборе которой в качестве принтера для печати будет предложен принтер по умолчанию того компьютера, на котором будет выполняться печать. Имя принтера может быть выбрано как из выпадающего списка, так и введено вручную.

Если включить опцию "Автопечать", то блок THTMLViewer не выведет диалог печати, а сразу же отправит данные на выбранный принтер. Если же данная опция отключена, то при вызове события печати блок выведет системный диалог, в котором можно будет выбрать принтер и указать требуемые параметры печати.

Ниже выпадающего списка выбора принтера расположены выпадающие списки для выбора размера и ориентации бумаги.

В нижней части диалога расположены элементы редактирования, предназначенные для задания отступов от края страницы. Значения задаются в миллиметрах.

2.1.3.3. Контейнеры

TGraphicsComposite	
Графический композитный блок	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах

TGraphicsComposite	
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать события работы от «мыши»
hint	Тип STRING. Всплывающая подсказка объкета
moveable	Тип BOOL. TRUE, если объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TPage	
Страница главного окна	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
bg_color	Тип TColor. Цвет фона окна
caption	Тип STRING. Заголовок главного окна
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать события работы от «мыши»
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. TRUE, если объект можно перемещать
pos	Тип TPos. Положение объекта

TPage	
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

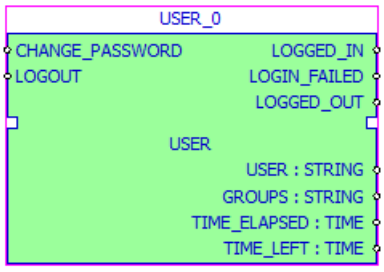
TWindow	
Контейнер – отдельное окно	
Входные события	
show	Показать окно
show_modal	Показать окно как модальное (диалоговое)
hide	Скрыть окно
Входные переменные	
pos	Тип TPos. Положение объекта
visible	Тип BOOL. Признак видимости объекта
size	Тип size. Размер объекта
bg_color	Тип TColor. Цвет фона окна
caption	Тип STRING. Заголовок окна
flags	Тип UDINT. Флаги окна
hScrollBarPolicy	Тип SINT. Политика показа горизонтальной полосы прокрутки: 0 - показывать полосы прокрутки при необходимости (по умолчанию); 1 - никогда не показывать полосы прокрутки; 2 - всегда показывать полосы прокрутки
vScrollBarPolicy	Тип SINT. Политика показа вертикальной полосы прокрутки:

TWindow	
	0 - показывать полосы прокрутки при необходимости (по умолчанию); 1 - никогда не показывать полосы прокрутки; 2 - всегда показывать полосы прокрутки
Выходные события	
showed	Окно появилось на экране
hid	Окно скрыто

2.1.3.4. Блоки среды исполнения

2.1.3.4.1. Безопасность и работа с пользователями

CHECK_RIGHTS	
Осуществляет проверку прав текущего пользователя. Срабатывает при входе/выходе пользователя, а также при возникновении события CHECK	
Входные события	
CHECK	Проверка
Входные переменные	
IN0	Проверяемая строка
Выходные переменные	
OUT	TRUE, если значение IN0 находится среди прав пользователя

USER

Управляет пользователем и предоставляет информацию о нем
Входные события
CHANGE_PASSWORD	Смена пароля пользователя. Блок выводит диалог смены пароля и проверяет валидность нового пароля
LOGOUT	Выход пользователя
Выходные события

USER	
LOGGED_IN	Возникает, когда пользователь входит в систему
LOGIN_FAILED	Возникает при неудачной попытке входа
LOGGED_OUT	Возникает, когда пользователь выходит из системы
Выходные переменные	
USER	Тип STRING. Имя пользователя, зарегистрированного в системе, или пустая строка, если никто не зарегистрирован
GROUPS	Тип STRING. Список групп пользователя, зарегистрированного в системе, или пустая строка, если никто не зарегистрирован
TIME_ELAPSED	Тип TIME. Сколько прошло времени с начала сессии пользователя
TIME_LEFT	Тип TIME. Сколько осталось времени до конца сессии пользователя, если она ограничена. Если нет ограничения, то выдаёт значение 0

USERS_EDITOR	
Диалоговое окно редактора пользователей	
Входные события	
SHOW_MODAL	Открывает диалог редактирования пользователей
HIDE	Закрывает диалог редактирования пользователей
Входные переменные	
POS	Тип TPos. Положение диалога
SIZE	Тип TSize. Размер диалога

USER_SESSION	
Проверяет права текущего пользователя и, при недостаточности прав, начинает новую сессию, выводя диалог ввода входного имени и пароля	
Входные события	
BEGIN	Запрос начала сессии с правами, заданными значением переменной RIGHTS
END	Оповещение о завершении сессии
Входные переменные	
RIGHTS	Тип STRING. Права, требуемые для начала сессии. Если прав недостаточно, то будет выведен диалог ввода входного имени и пароля
Выходные события	

USER_SESSION	
STARTED	Сессия начата
FINISHED	Сессия завершена
CANCELED	Сессия отменена

Описание работы

Для обеспечения контроля прав пользователя видеоменеджер содержит стек пользователей. При запуске видеоменеджера стек пользователей пуст. После входа в стеке появляется первый пользователь.

Однако, в ряде случаев, прав вошедшего пользователя недостаточно для осуществления какой-либо операции. Примером может служить ситуация, когда вошел пользователь, обладающий правами оператора, но необходимо произвести настройку уставок, для чего требуется участие технолога. Технолог должен внести изменения, после чего оператор должен продолжить свою работу. Для реализации подобной функциональности предназначен блок, организующий пользовательские сессии. Сессией будем называть временной промежуток, когда помимо основного пользователя в системе регистрируется кто-то еще без выхода из системы первого пользователя.

Для того, чтобы получить доступ к функциональности сессий, необходимо добавить один или несколько блоков USER_SESSION в мнемосхему пользовательского интерфейса. Блок не имеет графического окна и может быть размещен на любом уровне иерархии объектов.

Для начала новой сессии необходимо сформировать на входной переменной RIGHTS строку, содержащую требуемые права, после чего сформировать входное событие BEGIN. Подсистема безопасности проверит, есть ли у текущего пользователя права, значение которых содержится в переменной RIGHTS. Если прав достаточно, то начинается новая сессия, текущий пользователь копируется и помещается в стек, после чего возникает событие STARTED. Если прав недостаточно, то видеоменеджер выводит диалог входа пользователя. Пользователь вводит имя и пароль. Подсистема безопасности проверяет введенные данные и, если прав нового пользователя достаточно, то данные вошедшего пользователя помещаются в стек, после чего возникает событие STARTED. Если пользователь отказывается от входа, то сессия не начинается и возникает выходное событие CANCELED.

Для завершения сессии необходимо сформировать на входе блока событие END. Подсистема безопасности проверяет, есть ли начатая сессия и, если сессия есть, то завершает ее. В ходе завершения сессии из стека удаляется последний вошедший пользователь, после чего возникает выходное событие FINISHED.

Следует заметить, что объектов USER_SESSION может быть множество, а стек пользователей один на все приложение, поэтому необходимо следить за правильной последовательностью формирования событий BEGIN/END блоков USER_SESSION.

2.1.3.4.2. Архивы

ARCHIVE_TREND	
Блок показа архивных графиков	
Входные события	
PRINT	Напечатать данные. Данное событие появляется, если разработчик указал внешнее управление печатью в настройках объекта

ARCHIVE_TREND	
SAVE	Сохранить данные. Данное событие появляется, если разработчик указал внешнее управление сохранением в настройках объекта
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать события работы от «мыши»
moveable	Тип BOOL. TRUE, если объект можно перемещать
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объкета
size	Тип TSize. Размер объекта
font	Тип TFont. Шрифт объекта
DTV	Тип DT. Начало временной шкалы (оси абсцисс)
Y_AXIS_MIN	Тип LREAL. Нижнее значение (в %) оси ординат. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью ординат
Y_AXIS_MAX	Тип LREAL. Верхнее значение (в %) оси ординат. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью ординат
X_AXIS_LENGTH	Тип TIME. Длина оси абсцисс. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью абсцисс
FILE_PATH_TEMPLATE	Тип STRING. Шаблон имени файла, в котором будут сохраняться данные. По умолчанию директорией сохранения файла является рабочая папка с проектом. Обычные символы в названии файла не преобразуются. Символы, определяющие преобразования, предваряются символом %. В итоговом названии файла их заменят следующие символы: - %Y - показывает год как четырехразрядное десятичное число (с указанием века);

ARCHIVE_TREND	
	- %y - показывает год как двухразрядное число от 00 до 99 (без указания века); - %m - показывает месяц как десятичное число от 01 до 12; - %d - день месяца в десятичной форме (от 01 до 31); - %j - показывает день года как десятичное число от 001 до 366; - %H - показывает час как десятичное число от 00 до 23; - %M - показывает минуты как десятичное число от 00 до 59; - %S - отображает секунды в десятичной форме от 00 до 61; - Пример шаблона имени файла - "Data from %Y%m%d--%H-%M-%S.txt" (использовать символ **:** в имени файла, к примеру, для такой записи как %H:%M:%S, нельзя, его можно использовать только для указания пути к файлу в операционных системах семейства Windows, к примеру, "D:/Data from %Y%m%d.txt"). Когда данный входной параметр определен, диалог сохранения данных в файл выводиться не будет.
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
SAVED	Завершено сохранение данных графиков в файл
PRINTED	Завершена печать данных графиков
Выходные переменные	
MARKERA_POS	Тип DT. Положение маркера A
MARKERB_POS	Тип DT. Положение маркера B

ARCHIVE_TREND	
O_DTV	Тип DT. Начало временной шкалы (оси абсцисс)

Описание

Данный тип функционального блока предназначен для отображение графика архивных значений выбранных параметров. Внешний вид блока на мнемосхеме приведен на рис. 2.7

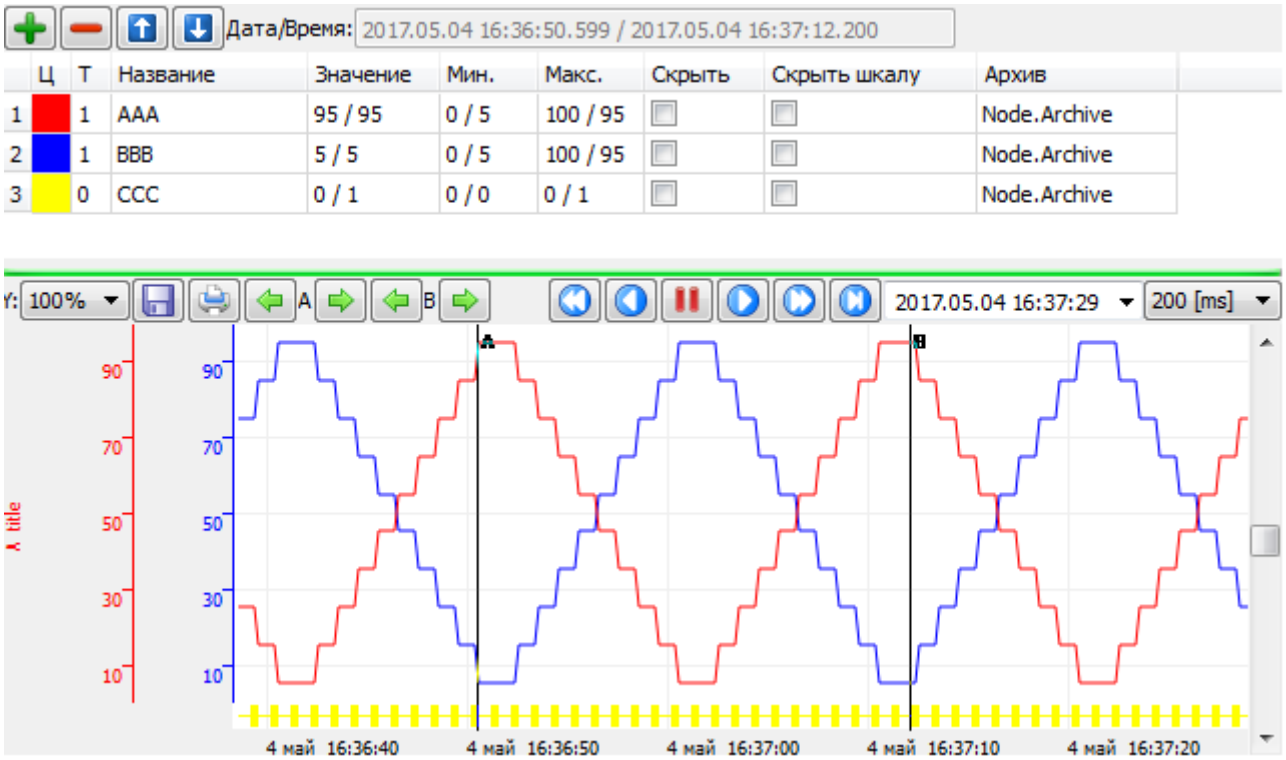


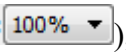
Рисунок 2.7 - Блок типа ARCHIVE_TREND, размещенный на мнемосхеме

Блок типа ARCHIVE_TREND состоит из двух частей - легенды, используя которую оператор может изменить набор отображаемых графиков, и основной области, состоящей из панели управления и собственно графиков.

Легенда состоит из списка графиков и панели управления набором графиков. В список выводятся порядковый номер, цвет, толщина линии, значения в точках пересечения маркеров и графиков, минимальные и максимальные значения в этих же точках, опции, позволяющие скрыть как весь график целиком, так и только вертикальную шкалу графика. В последнюю колонку списка выводится имя приложения-архива, из которого производится получение данных. Цвет графика может быть изменен путем нажатия левой кнопки «мыши» в колонке цвета строки, соответствующей выбранному графику.

В панели управления легендой размещены кнопки добавления (), удаления() графика, а также кнопки перемещения графика вверх () и вниз () в списке. Правее на панели инструментов легенды располагается поле, в которое выводятся временные метки маркеров графиков.

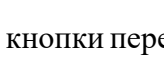
Рассмотрим элементы управления, выведенные в панель управления области графиков. Рассмотрение будем производить слева направо.

Панель инструментов начинается с выпадающего списка (), в котором производится выбор масштаба оси ординат. Данный элемент редактирования скрывается из панели инструментов, если в настройках указана необходимость внешнего управления осью ординат.

ARCHIVE_TREND

Далее расположена кнопка сохранения данных (), при нажатию на которую программа выводит диалог выбора файла для сохранения. Пользователь может сохранить данные в формате txt, либо в формате png. При выборе текстового формата данные сохраняются в табличном виде. При выборе формата png-изображения данные сохраняются в виде картинки. Кнопка сохранения присутствует в панели инструментов, если в настройках объекта не указано внешнее управление сохранением.

Следом расположена кнопка печати (), при нажатии на которую программа выводит диалог печати, если в настройках печати не выбрана опция автопечати. Если же указана автопечать, то печать осуществляется на выбранный принтер с указанными настройками. Кнопка печати присутствует в панели инструментов, если в настройках объекта не указано внешнее управление печатью.

Еще правее расположены кнопки перемещения маркеров A () и B (). Кнопка  в каждой паре служит для смещения маркера влево, а кнопка  - вправо.

Далее располагается группа навигационных кнопок, обеспечивающих навигацию вдоль оси абсцисс. Кнопки  и  служат для промотки одного экрана назад и вперед, кнопки  и для промотки одного блока размером в расстояние между вертикальными линиями сетки назад и вперед соответственно, кнопка  служит для возврата правого края графика к концу оси ординат. Кнопки  и  служат для приостановки и возобновления получения данных.

Панель инструментов завершается выпадающим списком частот опроса архива (). При изменении частоты опроса объект автоматически выбирает подходящую длину оси абсцисс. Данный элемент редактирования скрывается из панели инструментов, если в настройках указана необходимость внешнего управления длиной оси абсцисс.

Поле отрисовки графиков можно условно разделить на четыре области. Первая область - это область вертикальных осей, расположенная в левой части поля. Вторая область - это область оси абсцисс, в которой выводятся значения временных меток вертикальных осей сетки. Выше нее расположена область графиков сигналов логического типа (BOOL). Вся остальная часть отведена под отрисовку графиков числовых величин.

Нажав на левую кнопку «мыши» в поле графиков и проведя указателем в сторону, можно получить два маркера - A и B, пр этом первый маркер останется на месте нажатия левой кнопки «мыши», а второй - в месте ее отпускания. Маркеры могут быть перемещены независимо - для этого необходимо подвести указатель «мыши» к маркеру, нажать на левую кнопку «мыши» и переместить его, удерживая нажатой левую кнопку «мыши». Перемещая маркеры, можно их совместить - в этом случае маркер останется только один до следующего разделения.

Настройка

Окно настроек представляет собой многостраничный диалог, содержащий вкладки общих настроек объекта, настроек списка графиков, настроек интерфейса и настроек печати.

Страница общих настроек приведена на рис. 2.8.

ARCHIVE_TREND

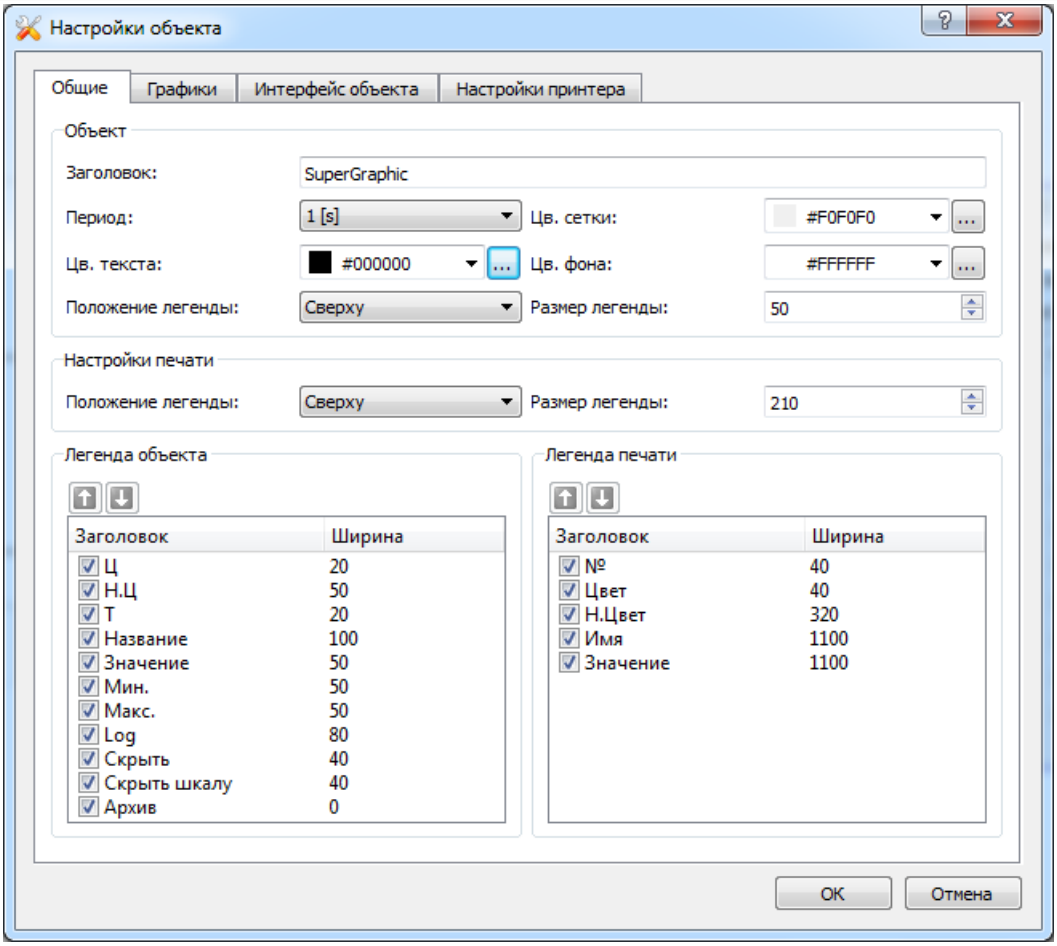


Рисунок 2.8 - Страница общих настроек

Группа элементов "Объект" предназначена для редактирования настроек области графика. В поле ввода "Заголовок" вносится наименование графика, выводимое при сохранении и печати. Используя выпадающий список "Период", разработчик выбирает частоту опроса архива. При помощи элементов "Цв. текста", "Цв. сетки" и "Цв. фона" данной группы разработчик выбирает цвета текста, сетки и фона графика соответственно. Выпадающий список положений легенды показывает положение списка сигналов относительно области отрисовки графиков. Поле ввода размера легенды используется разработчиком для того, чтобы задать изначальный размер легенды (в пикселях). Если значение, введенное в данное поле, равно 0, то легенда изначально скрыта.

Группа элементов "Настройки печати" служит для редактирования положения легенды, выводимой на печать, относительно области графиков и ее размера.

Группы элементов "Легенда объекта" и "Легенда печати" предназначены для определения набора колонок легенд, задания порядка их следования и их ширин (в пикселях). Кнопки и служат для перемещения колонок таблиц влево и вправо соответственно.

Элементы легенды объекта:

- **Ц** - цвет, которым отображается шкала и график значений данного объекта (сигнала);
- **Н.Ц** - цвет, которым отображается шкала и график значений данного объекта (сигнала), когда он невалидный;
- **Т** - толщина линии на графике;
- **Название** - название для данного объекта, которое можно задать во вкладке Графики - Заголовок (см. далее);
- **Значение** - значение сигнала в выбранный момент времени или в диапазоне времени;
- **Мин.** - минимальное значение шкалы объекта;

ARCHIVE_TREND

- **Макс.** - максимальное значение шкалы объекта;
- **Log** - логарифмическая шкала объекта;
- **Скрыть** - скрыть шкалу и график значений объекта;
- **Скрыть шкалу** - скрыть шкалу объекта;
- **Архив** - имя приложения - архива, из которого берутся значения для графика.

Элементы легенды печати позволяют настроить поля, которые выводятся на печать. Их смысл аналогичен элементам в легенде объекта.

Страница настройки графиков предназначена для формирования набора и указания характеристик графиков, изначально добавленных в список графиков объекта. Внешний вид страницы приведен на рис. 2.9.

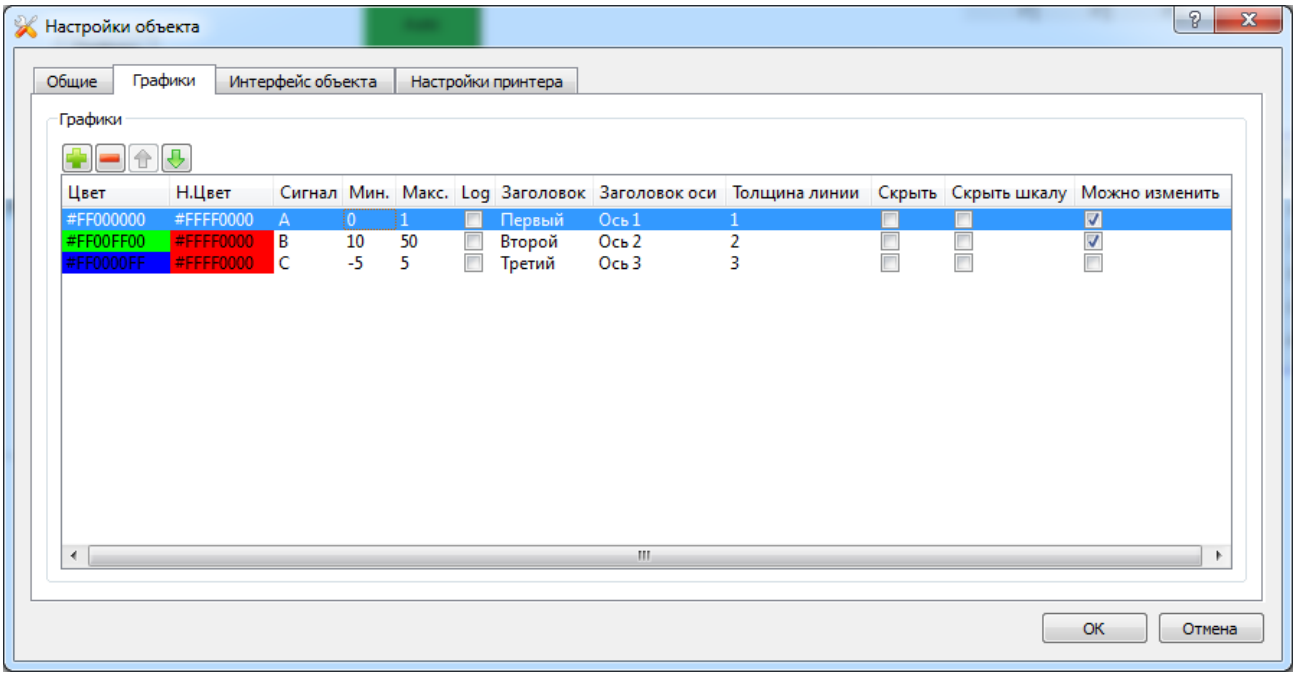


Рисунок 2.9 - Страница редактирования набора графиков

Элементы редактирования, размещенные на данной странице, включают в себя список графиков и размещенную над ним панель инструментов. Кнопки  и  служат для добавления и удаления графика, а кнопки  и  - для изменения порядка их следования.

Описание полей:

- **Цвет** - цвет, которым будут отображаться валидные значения сигнала на графике;
- **Н.Цвет** - цвет, которым будут отображаться невалидные значения сигнала на графике;
- **Сигнал** - сигнал, архивные значения которого отображаются на графике;
- **Мин.** - минимальное значение шкалы на графике;
- **Макс.** - максимальное значение шкалы на графике;
- **Log** - логарифмическое отображение шкалы сигнала;
- **Заголовок** - название графика, которое будет отображаться в поле Название легенды объекта;
- **Заголовок оси** - заголовок, который будет отображаться рядом с осью графика;
- **Толщина линии** - толщина линии графика;
- **Скрыть** - скрыть график и шкалу данного сигнала;
- **Скрыть шкалу** - скрыть шкалу сигнала;

- **Можно изменить** - данная галочка, разрешает изменять выше описанные поля сигнала в процессе работы проекта.

Двойной щелчок «мышью» в колонке сигнала приведет к открытию диалога выбора сигнала. Этот же диалог открывается при нажатии на кнопку добавления сигнала в панели инструментов.

Заголовок графика выводится в таблицу сигналов при печати или сохранении данных в виде изображения. Заголовок оси выводится в области отрисовки вертикальных осей.

Страница настройки интерфейса объекта предназначена для редактирования параметров, определяющих набор входных элементов интерфейса объекта. Внешний вид страницы приведен на рис. 2.10.

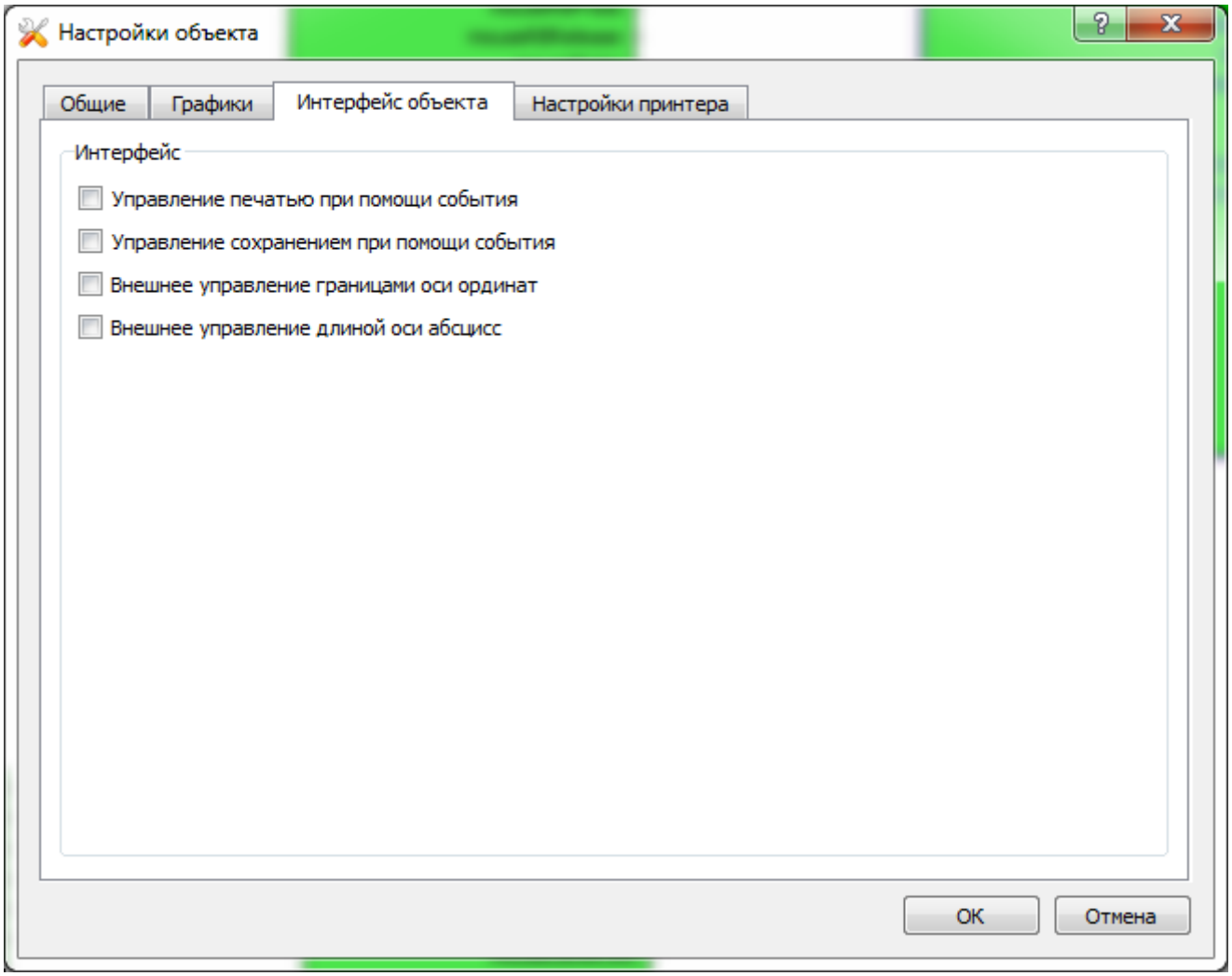


Рисунок 2.10 - Страница редактирования интерфейса

Выбор опции "Управление печатью при помощи события" приводит к появлению входного события PRINT и скрытию кнопки печати с панели инструментов.

Выбор опции "Управление сохранением при помощи события" приводит к появлению входного события SAVE и скрытию кнопки сохранения с панели инструментов.

Выбор опции "Внешнее управление границами оси ординат" приводит к появлению входных переменных Y_AXIS_MIN и Y_AXIS_MAX и скрытию выпадающего списка масштаба оси ординат с панели инструментов.

Выбор опции "Внешнее управление длиной оси абсцисс" приводит к появлению входной переменной X_AXIS_LENGTH и скрытию выпадающего списка длин оси абсцисс с панели инструментов.

ARCHIVE_TREND

Страница настроек печати приведена на рис. 2.11.

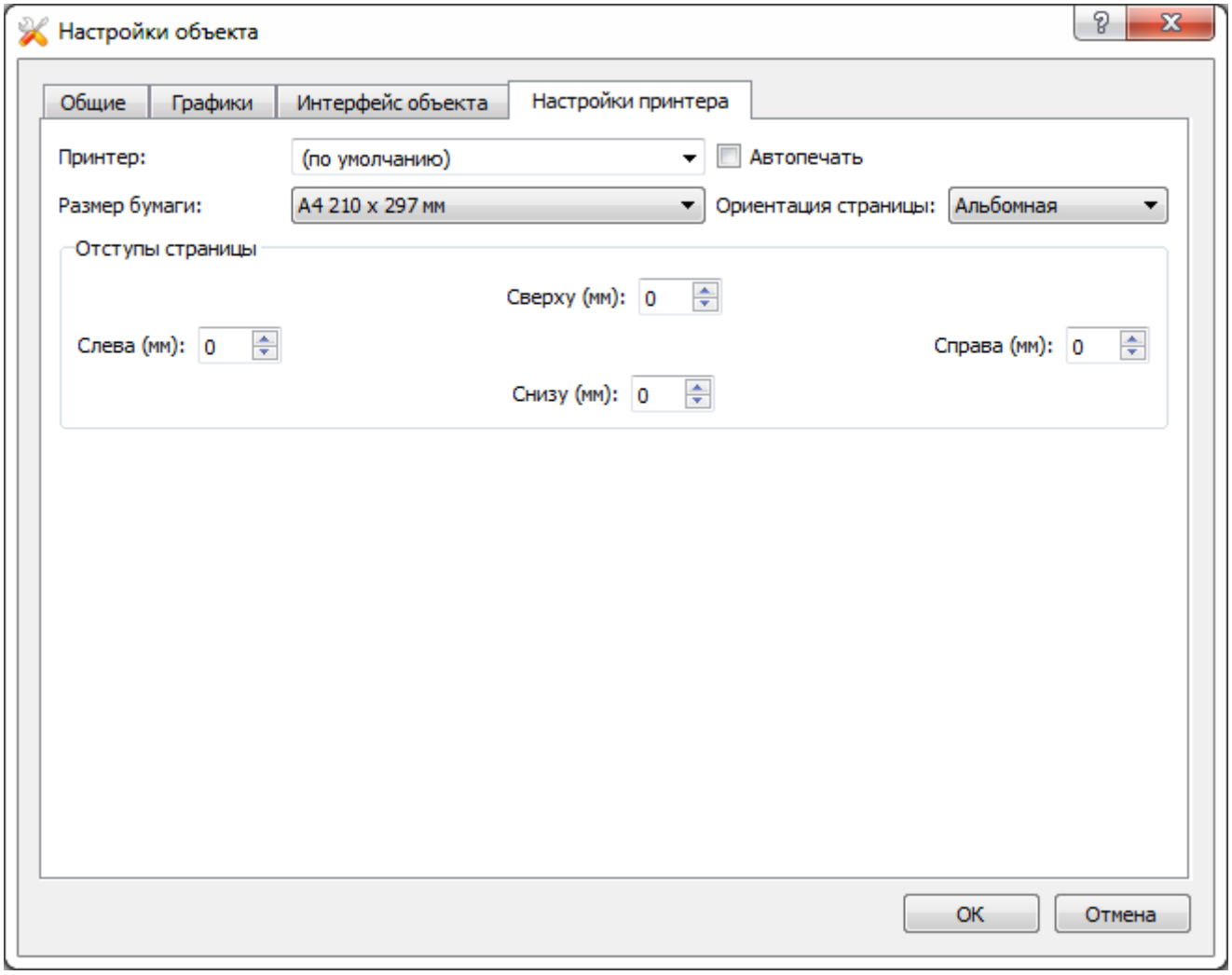


Рисунок 2.11 - Страница настроек печати

Выпадающий список "Принтер" предназначен для выбора устройства, которое будет выбрано в качестве предлагаемого принтера для печати. Первым элементом выпадающего списка является строка "(по умолчанию)", при выборе которой в качестве принтера для печати будет предложен принтер по умолчанию того компьютера, на котором будет выполняться печать. Имя принтера может быть выбрано как из выпадающего списка, так и введено вручную.

Если включить опцию "Автопечать", то блок не выведет диалог печати, а сразу же отправит данные на выбранный принтер. Если же данная опция отключена, то при вызове события печати блок выведет системный диалог, в котором можно будет выбрать принтер и указать требуемые параметры печати.

Ниже выпадающего списка выбора принтера расположены выпадающие списки для выбора размера и ориентации бумаги.

В нижней части диалога расположены элементы редактирования, предназначенные для задания отступов от края страницы. Значения задаются в миллиметрах.

ARCHIVE_TREND_WINDOW

Блок показа архивных графиков в отдельном окне.

Описание интерфейса данного блока аналогично блоку ARCHIVE_TREND.

ARCHIVE_TREND_WINDOW	
Входные события	
SHOW	Показать окно
HIDE	Спрятать окно
PRINT	Напечатать данные. Данное событие появляется, если разработчик указал внешнее управление печатью в настройках объекта
SAVE	Сохранить данные. Данное событие появляется, если разработчик указал внешнее управление сохранением в настройках объекта
Входные переменные	
POS	Тип TPos. Положение объекта
SIZE	Тип TSize. Размер объекта
font	Тип TFont. Шрифт объекта
DTV	Тип DT. Начало временной шкалы (оси абсцисс)
Y_AXIS_MIN	Тип LREAL. Нижнее значение (в %) оси ординат. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью ординат
Y_AXIS_MAX	Тип LREAL. Верхнее значение (в %) оси ординат. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью ординат
X_AXIS_LENGTH	Тип TIME. Длина оси абсцисс. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью абсцисс
FILE_PATH_TEMPLATE	Тип STRING. Шаблон имени файла, в котором будут сохраняться данные. По умолчанию директорией сохранения файла является рабочая папка с проектом. Обычные символы в названии файла не преобразуются. Символы, определяющие преобразования, предваряются символом %. В итоговом названии файла их заменят следующие символы: - %Y - показывает год как четырехразрядное десятичное число (с указанием века); - %y - показывает год как двухразрядное число от 00 до 99 (без указания века); - %m - показывает месяц как десятичное число от 01 до 12; - %d - день месяца в десятичной форме (от 01 до 31);

ARCHIVE_TREND_WINDOW	
	- %j - показывает день года как десятичное число от 001 до 366; - %H - показывает час как десятичное число от 00 до 23; - %M - показывает минуты как десятичное число от 00 до 59; - %S - отображает секунды в десятичной форме от 00 до 61; - Пример шаблона имени файла - "Data from %Y%m%d--%H-%M-%S.txt" (использовать символ **:** в имени файла, к примеру, для такой записи как %H:%M:%S, нельзя, его можно использовать только для указания пути к файлу в операционных системах семейства Windows, к примеру, "D:/Data from %Y%m%d.txt"). Когда данный входной параметр определен, диалог сохранения данных в файл выводиться не будет.
Выходные события	
SHOWED	Окно появилось на экране
HID	Окно скрыто
SAVED	Завершено сохранение данных графиков в файл
PRINTED	Завершена печать данных графиков
Выходные переменные	
MARKERA_POS	Тип DT. Положение маркера A
MARKERB_POS	Тип DT. Положение маркера B
O_DTV	Тип DT. Начало временной шкалы (оси абсцисс)

2.1.3.4.3. События и тревоги

TAlarmViewer	
Список тревог в системе	
Входные события	
SETUP_FILTER	Настроить фильтр списка
ACKNOWLEDGE	Квитировать выбранную тревогу
ACKN_FILTERED	Квитирование отфильтрованных тревог
ACKNOWLEDGE_ALL	Квитировать все тревоги
QUERY_ACTION	Выполнить дополнительное действие

TAlarmViewer	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать события работы от «мыши»
font	Тип TFont. Шрифт объекта
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. TRUE, если объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
ACTION_ID	Тип INT. Числовой код дополнительного действия
FILTER	Тип TAlarmFilter. Внешний фильтр тревог (опционально)
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
DO_ACTION	Произведен запрос дополнительного действия
ERROR	Произошла ошибка
Выходные переменные	
ALARM_DT	Тип DT. Дата и время выбранной тревоги
ALARM_STATE	Тип INT. Состояние выбранной тревоги
ALARM_SOURCE	Тип STRING. Источник выбранной тревоги

TAlarmViewer	
ALARM_MESSAGE	Тип STRING. Сообщение выбранной тревоги
ALARM_TAG	Тип STRING. Дополнительные данные выбранной тревоги
O_ACTION_ID	Тип INT. Код дополнительного действия (копия ACTION_ID)
COMMON_STATE	Тип INT. Если нет тревог, то значение на выходе будет равно 0. Если есть хотя бы одна тревога в активном состоянии, то значение на выходе будет равно 1. Если нет активных тревог, но есть хотя бы одна в состоянии тревога исчезла (но не подтверждена), то значение на выходе будет равно 3. Если нет активных тревог и тревог в состоянии исчезла (но не подтверждена), но есть хотя бы одна тревога в состоянии квитирована (подтверждена), то значение на выходе будет равно 2

Настройка

Окно настройки отображения списка тревог представляет собой многостраничный диалог. В ходе настройки разработчик формирует список категорий тревог, задает внешний вид их отображения, настраивает первичные параметры фильтра списка тревог, выбирает столбцы для отображения и задает порядок их следования.

Внешний вид отображения тревог каждой категории настраивается на вкладке "Палитра". Кнопки  и  предназначены для изменения количества категорий в списке. Таблица категорий, расположенная под кнопками, позволяет разработчику менять цвет фона и текста для каждого состояния тревоги каждой категории. Цвет шрифта и фона может быть введен вручную, выбран из выпадающего списка или из палитры. Вкладка "Палитра" диалога настройки отображения списка событий представлена на рис. 2.12

TAlarmViewer

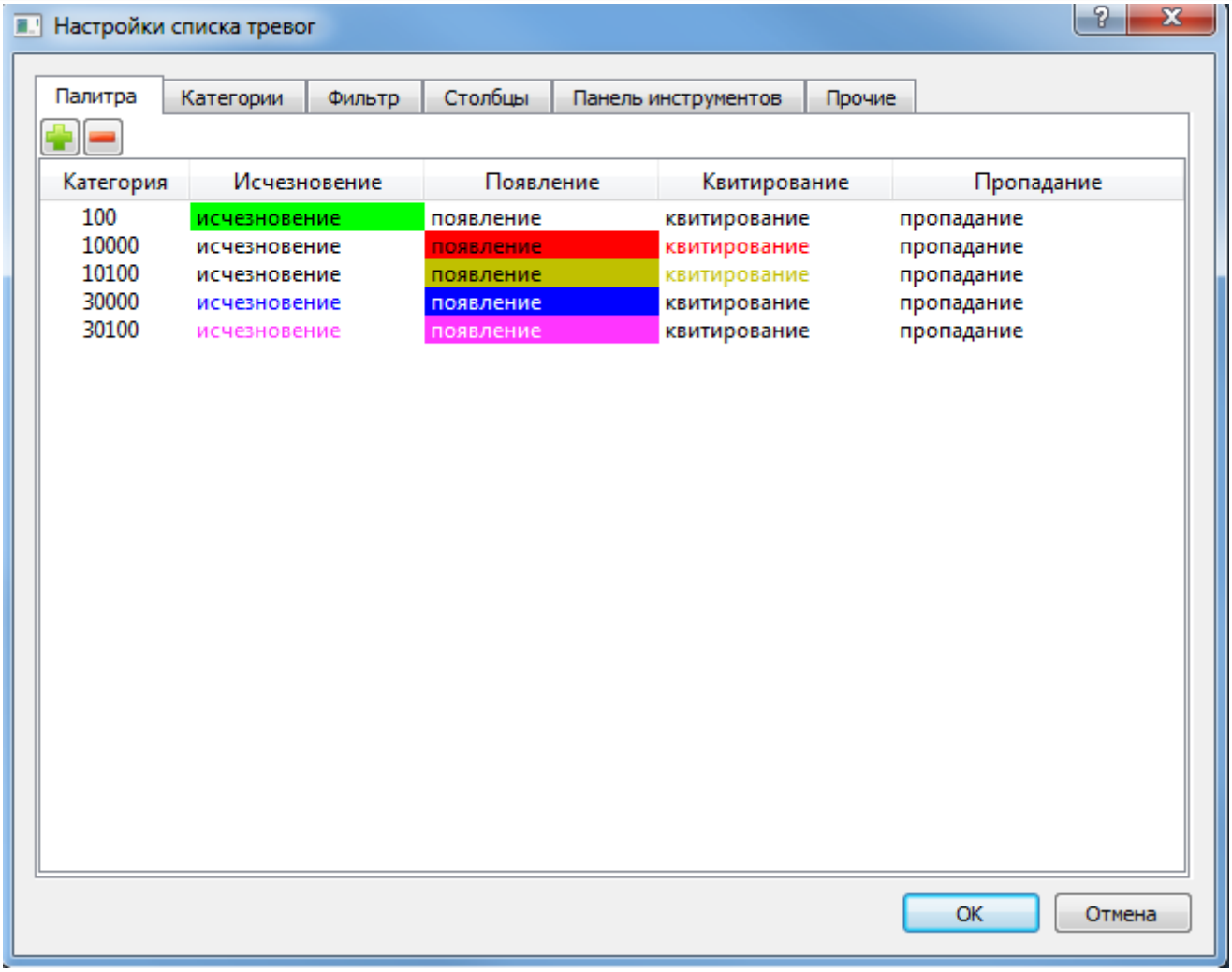
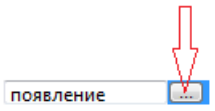


Рисунок 2.12 - Вкладка настройки палитры списка тревог

Если нажать двойным щелчком мыши в поле, где настраивается цвет, то в данном поле справа появится дополнительная кнопка настроек 

, нажав на которую откроется окно настройки цвета текста и фона (см. рис. 2.13).

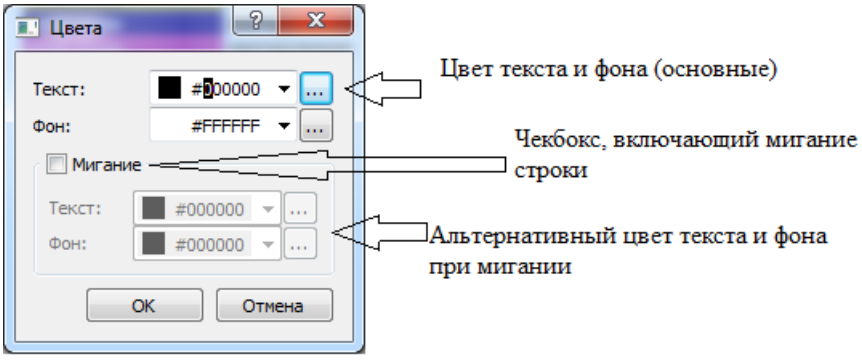


Рисунок 2.13 - Меню цветовых настроек палитры

На вкладке "Категории" разработчиком присваиваются категориям имена для удобства настройки фильтра. Кнопки  и  предназначены для изменения количества категорий в

списке. Таблица категорий, расположенная под кнопками, позволяет разработчику задать название для каждой категории. Вкладка "Категории" диалога настройки отображения списка тревог представлена на рис. 2.14

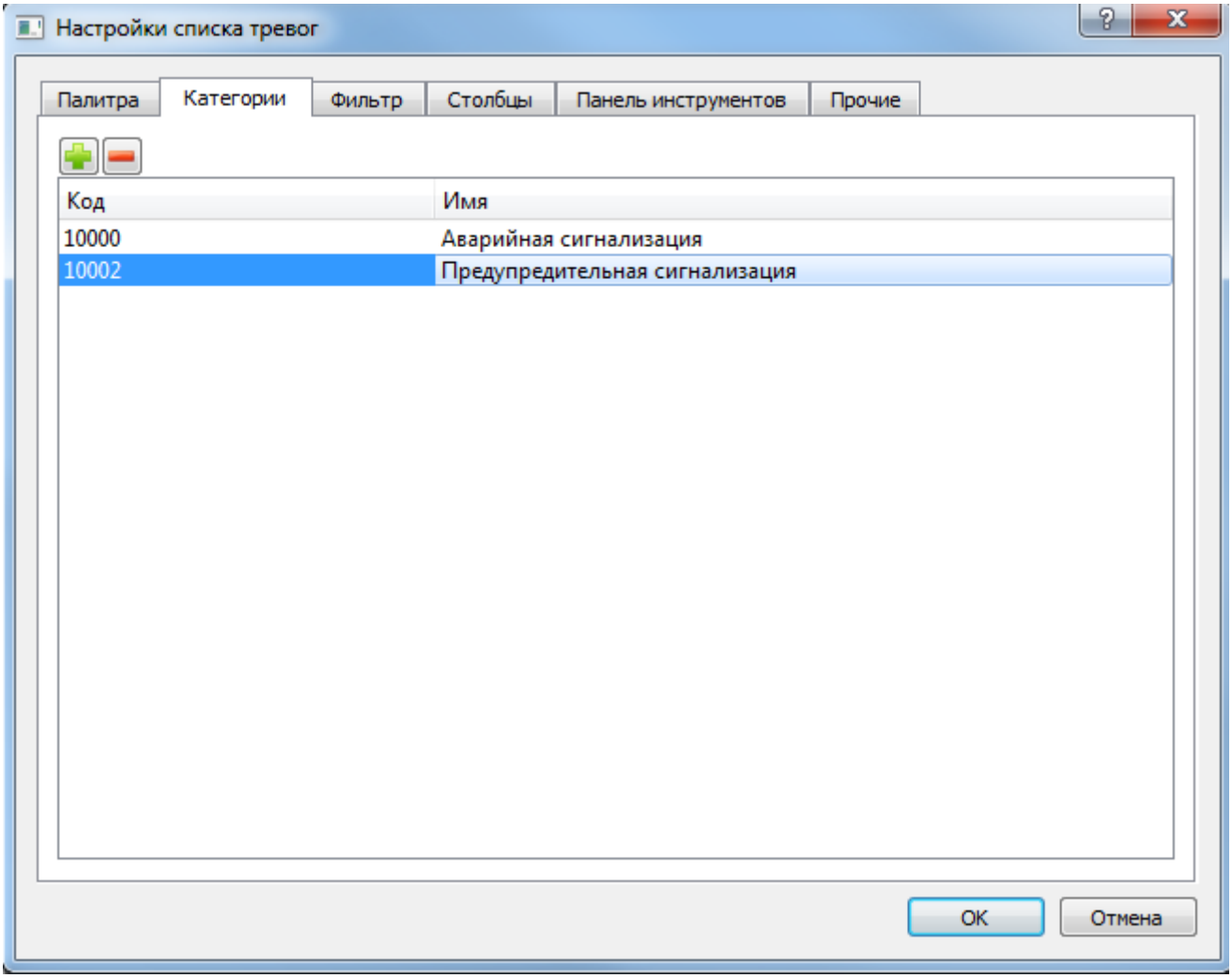


Рисунок 2.14 - Вкладка настройки категорий списка тревог

Первичные настройки фильтра задаются оператором на вкладке "Фильтр". Фильтровать тревоги можно по категориям, состояниям, источнику, сообщению и системным дате и времени. Вкладка "Фильтр" диалога настройки отображения списка тревог представлена на рис. 2.15

TAlarmViewer

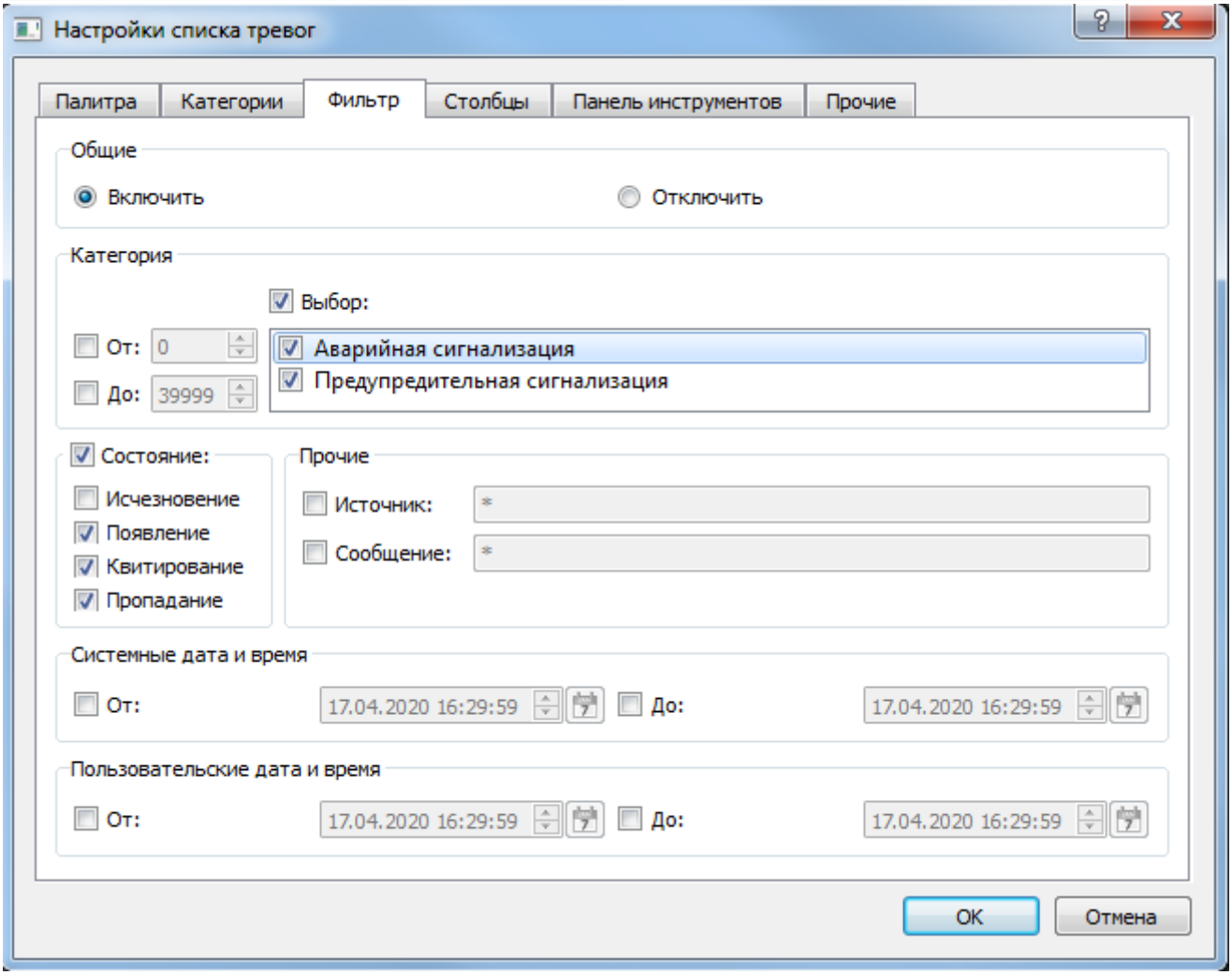


Рисунок 2.15 - Вкладка настройки фильтра списка тревог

В разделе Фильтр - Прочие, где можно настраивать фильтрацию по столбцу Источник и Сообщение, применяются следующие условные записи:

- ABC* - любая строка, начинающаяся с символов ABC;
- *ABC - любая строка, заканчивающаяся символами ABC;
- * - любая строка;
- ~ABC* - любая строка, которая не начинается с ABC;
- (~A*|*ABC) - любая строка, которая не начинается с символа A или заканчивается символами ABC (условий ИЛИ может быть несколько);
- \ - экранирующий символ. Если в вашей строке данных, к примеру, есть символ ~ и необходимо настроить по нему поиск, то нужно в строке условий написать \~, тогда данный символ не будет восприниматься как спец символ. Так же можно искать по символам \\, \(, \), \|.

Примечание. Для фильтрации по времени лучше использовать Системные дата и время, т.к. по данному критерию фильтр отрабатывает намного быстрее.

На вкладке "Столбцы" оператор выбирает, какие столбцы отображать при отображении списка событий, а также определяет порядок следования столбцов. Кнопки и предназначены для изменения порядка следования столбцов. Таблица элементов, расположенная ниже, позволяет включить или отключить отображение каждого столбца и задать его ширину. Ширина указывается в символах. Значение ширины столбца, равное -1, соответствует значению по умолчанию,

которое равно 10 символам. Вкладка "Столбцы" диалога настройки отображения списка тревог представлена на рис. 2.16

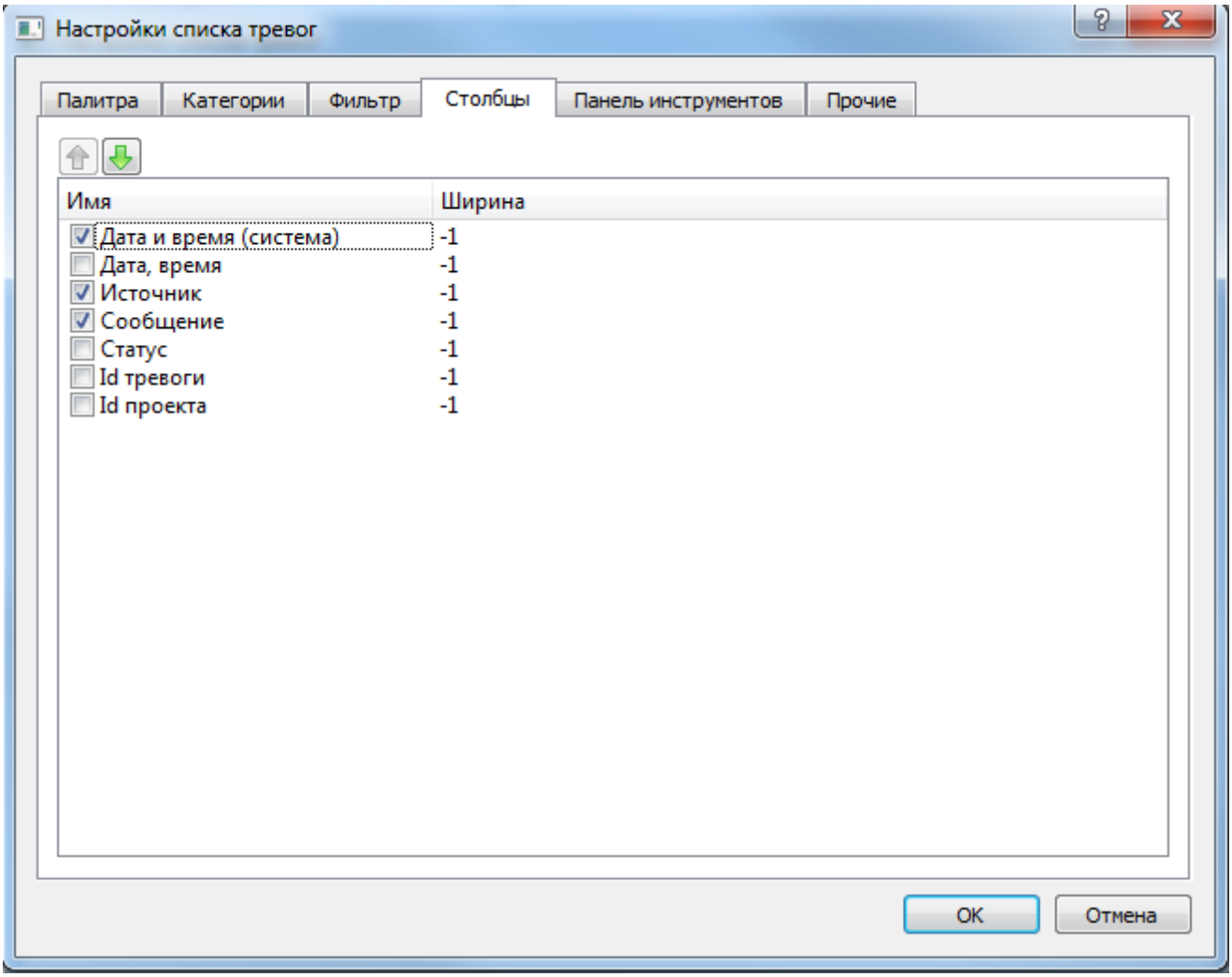


Рисунок 2.16 - Вкладка настройки колонок списка тревог

На вкладке "Панель инструментов" оператор выбирает какие кнопки будут располагаться на панели инструментов блока на мнемосхеме и их порядок (см. рис. 2.17).. Кнопки и предназначены для изменения порядка следования кнопок. Таблица элементов, расположенная ниже позволяет включить или отключить отображение каждой кнопки, а так же изменить ее название. Возможность изменения названия кнопки появляется после двойного нажатия левой кнопки мышки на соответствующем элементе. В столбце "Описание" содержится информация о назначении каждой отдельной кнопки.

Ниже расположены три настройки, позволяющие отображать панель управления на соответствующем блоке мнемосхемы, отображать источник данных, из которого в данный момент берутся события и возможность выбора источника данных.

Назначение кнопок:

- **Фильтр** - настройки фильтра отображения событий;
- **Квитировать** - квитировать выделенные события;
- **Квитировать отфильтр.** - квитировать события, прошедшие через фильтр;
- **Квитировать все** - квитировать все события;
- **Доп. действие** - Дополнительное действие, ID которого задается в переменной ACTION_ID блока. Подробное описание см. в документе SCADA-система "Соната" Руководство пользователя

TAlarmViewer

КУНИ.505200.023-01.01 95 в разделе Описание работы с приложением BRIDGE (межпроектный обмен).;

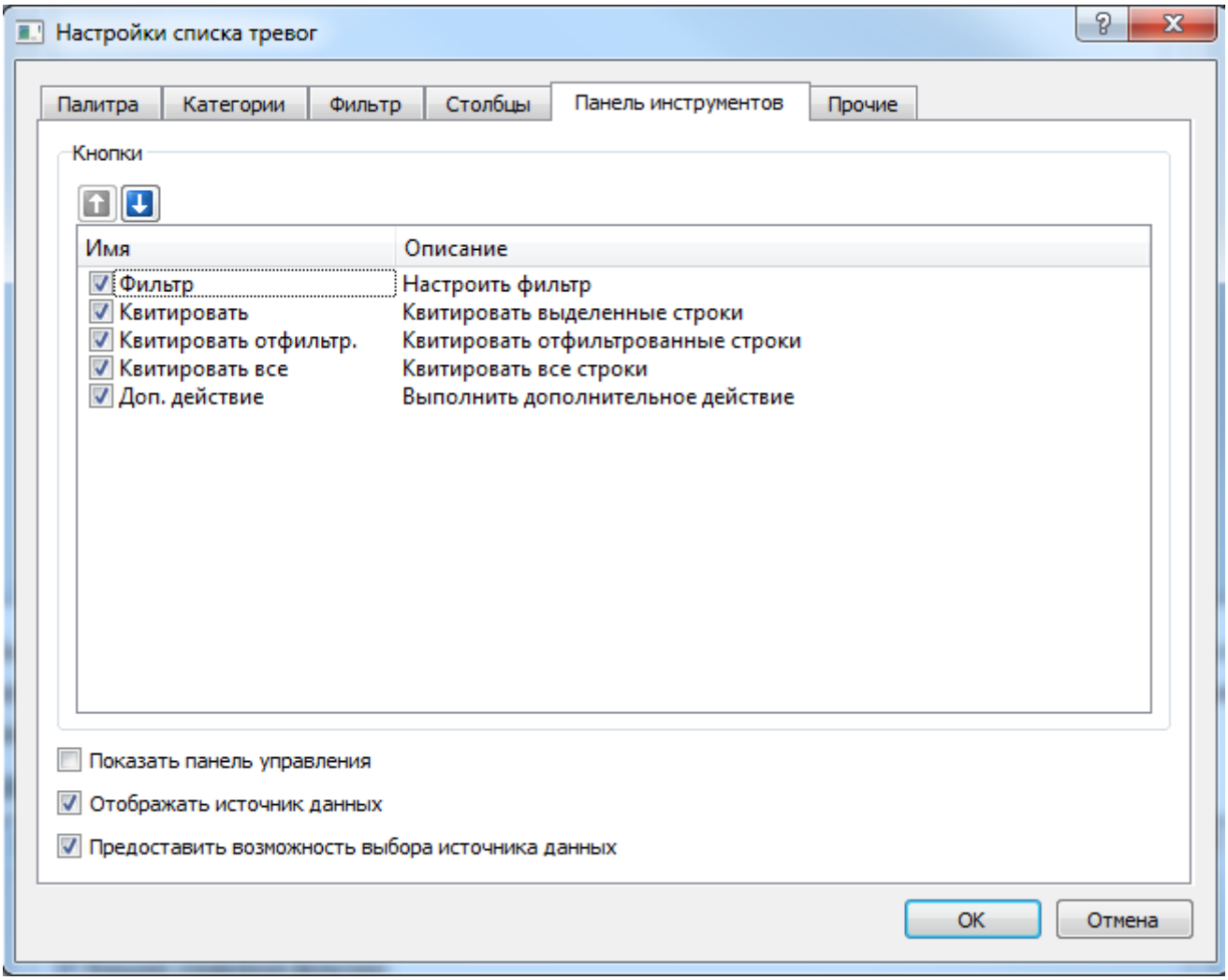


Рисунок 2.17 - Вкладка настроек Панель инструментов

На вкладке "Прочие" оператор может включить или отключить дополнительную переменную внешнего фильтра в интерфейсе блока на FBD диаграмме. В поле ввода Id необходимо указать из какого источника будут браться события (источник можно указать только один). Подробное описание см. в документе SCADA-система "Соната" Руководство пользователя КУНИ.505200.023-01.01 95 в разделе Описание работы с приложением BRIDGE (межпроектный обмен). Вкладка "Прочие" диалога настройки отображения списка тревог представлена на рис. 2.18

TAlarmViewer

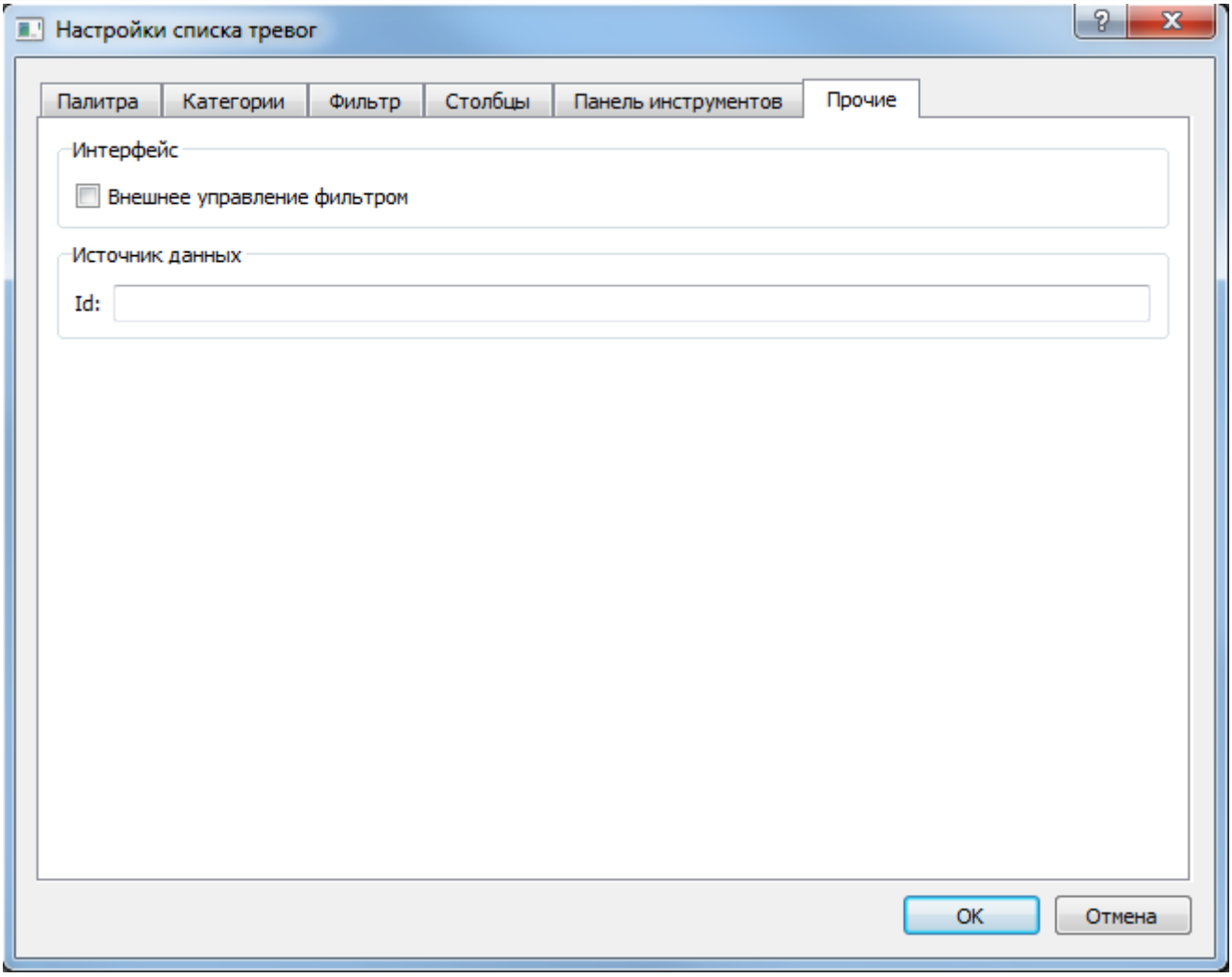



Рисунок 2.18 - Вкладка настройки интерфейса

TEventViewer	
Список событий	
Входные события	
SETUP_FILTER	Настроить фильтр списка
SAVE	Сохранить отфильтрованные события
SAVE_ALL	Сохранить все события
PRINT	Напечатать отфильтрованные события
PRINT_ALL	Напечатать все события
ACKNOWLEDGE	Квитировать тревогу, связанную с выбранным событием
ACKNOWLEDGE	Квитировать тревогу, связанную с выбранным событием
ACKN_FILTERED	Квитирование отфильтрованных тревог
QUERY_ACTION	Выполнить дополнительное действие
Входные переменные	

TEventViewer	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать события работы от «мыши»
font	Тип TFont. Шрифт объекта
hint	Тип STRING. Всплывающая подсказка объкета
moveable	Тип BOOL. TRUE, если объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
ACTION_ID	Тип INT. Числовой код дополнительного действия
FORCE_FILTER	Тип TForceEventFilter. Флаги использования полей фильтра, заданного переменной FILTER
FILTER	Тип TEventFilter. Внешний фильтр событий
FILE_PATH_TEMPLATE	Тип STRING. Шаблон имени файла, в котором будут сохраняться данные. По умолчанию директорией сохранения файла является рабочая папка с проектом. Обычные символы в названии файла не преобразуются. Символы, определяющие преобразования, предваряются символом %. В итоговом названии файла их заменят следующие символы: - %Y - показывает год как четырехразрядное десятичное число (с указанием века); - %y - показывает год как двухразрядное число от 00 до 99 (без указания века); - %m - показывает месяц как десятичное число от 01 до 12; - %d - день месяца в десятичной форме (от 01 до 31); - %j - показывает день года как десятичное число от 001 до 366; - %H - показывает час как десятичное число от 00 до 23; - %M - показывает минуты как десятичное число от 00 до 59;

TEventViewer	
	- %S - отображает секунды в десятичной форме от 00 до 61; - Пример шаблона имени файла - "Data from %Y%m%d--%H-%M-%S.txt" (использовать символ **:** в имени файла, к примеру, для такой записи как %H:%M:%S, нельзя, его можно использовать только для указания пути к файлу в операционных системах семейства Windows, к примеру, "D:/Data from %Y%m%d.txt"). Когда данный входной параметр определен, диалог сохранения данных в файл выводиться не будет.
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
DO_ACTION	Произведен запрос дополнительного действия
SAVED	Закончено сохранение событий
PRINTED	Закончена печать событий
ERROR	Произошла ошибка
Выходные переменные	
EVENT_DT	Тип DT. Дата и время выбранного события
STATE	Тип INT. Состояние выбранного события
SOURCE	Тип STRING. Источник выбранного события
MESSAGE	Тип STRING. Сообщение выбранного события
TAG	Тип STRING. Дополнительные данные выбранного события
O_ACTION_ID	Тип INT. Код дополнительного действия (копия ACTION_ID)

TEventViewer

Настройка

Настройка отображения списка событий в программе представляет собой многостраничный диалог. В ходе настройки разработчик формирует список категорий событий, задает внешний вид их отображения, настраивает первичные параметры фильтра списка событий, выбирает столбцы для отображения и задает порядок их следования, при необходимости указывает заголовок для печати.

Внешний вид отображения событий каждой категории настраивается на вкладке "Палитра". Кнопки  и  предназначены для изменения количества категорий в списке. Таблица категорий, расположенная под кнопками, позволяет разработчику менять цвет фона и текста для каждого состояния тревоги каждой категории. Цвет текста и фона может быть введен вручную, выбран из выпадающего списка или из палитры. Вкладка "Палитра" диалога настройки отображения списка событий представлена на рис. 2.19

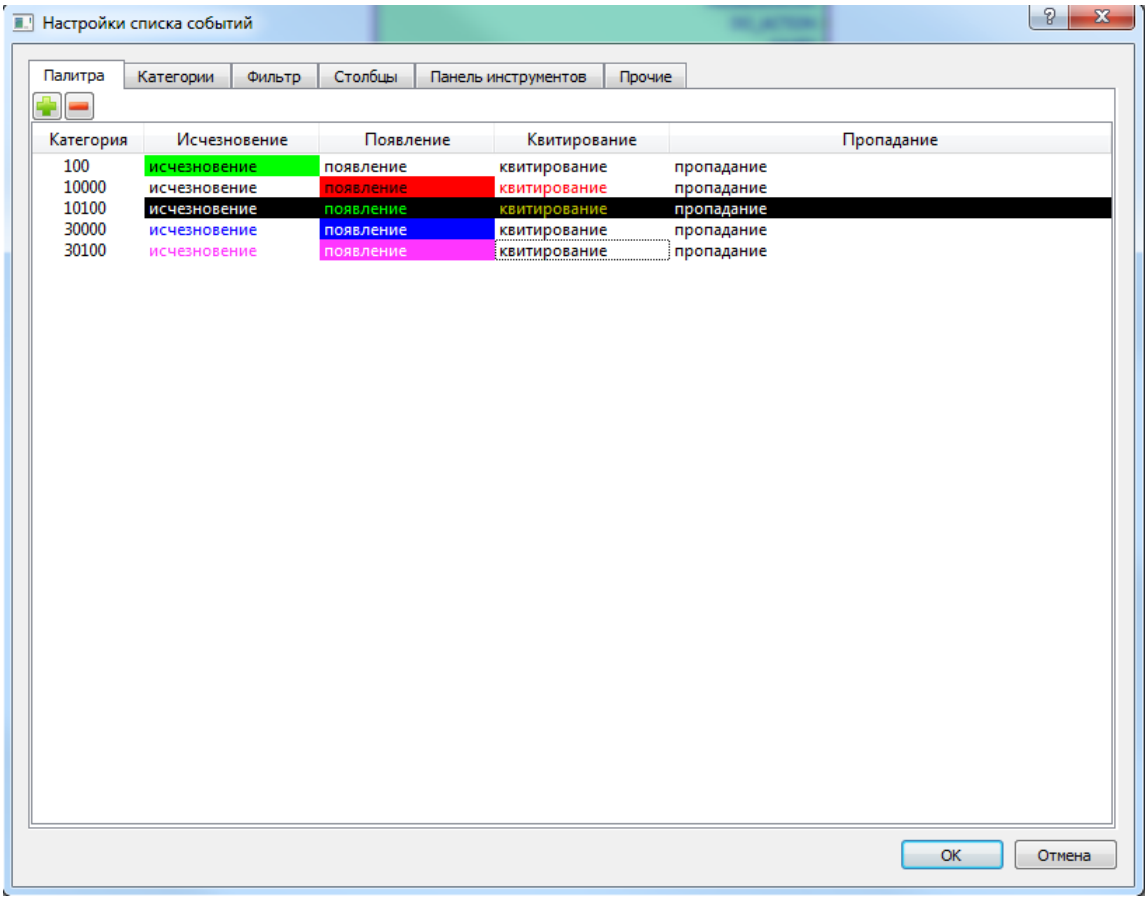
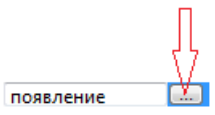


Рисунок 2.19 - Вкладка "Палитра" диалога настройки списка событий

Если нажать двойным щелчком мыши в поле, где настраивается цвет, то в данном поле справа появится дополнительная кнопка настроек 

, нажав на которую откроется окно настройки цвета текста и фона (см. рис. 2.20).

TEventViewer

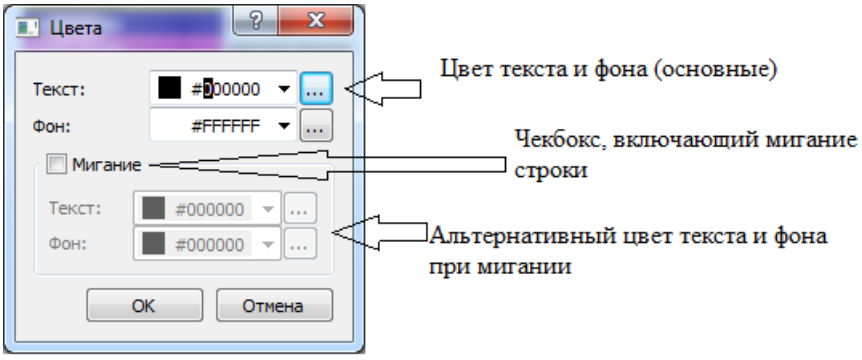


Рисунок 2.20 - Меню цветовых настроек палитры

На вкладке "Категории" разработчиком присваиваются имена категориям для удобства настройки фильтра. Кнопки и предназначены для изменения количества категорий в списке. Таблица категорий, расположенная под кнопками, позволяет разработчику задать название для каждой категории. Вкладка "Категории" диалога настройки отображения списка событий представлена на рис. 2.21

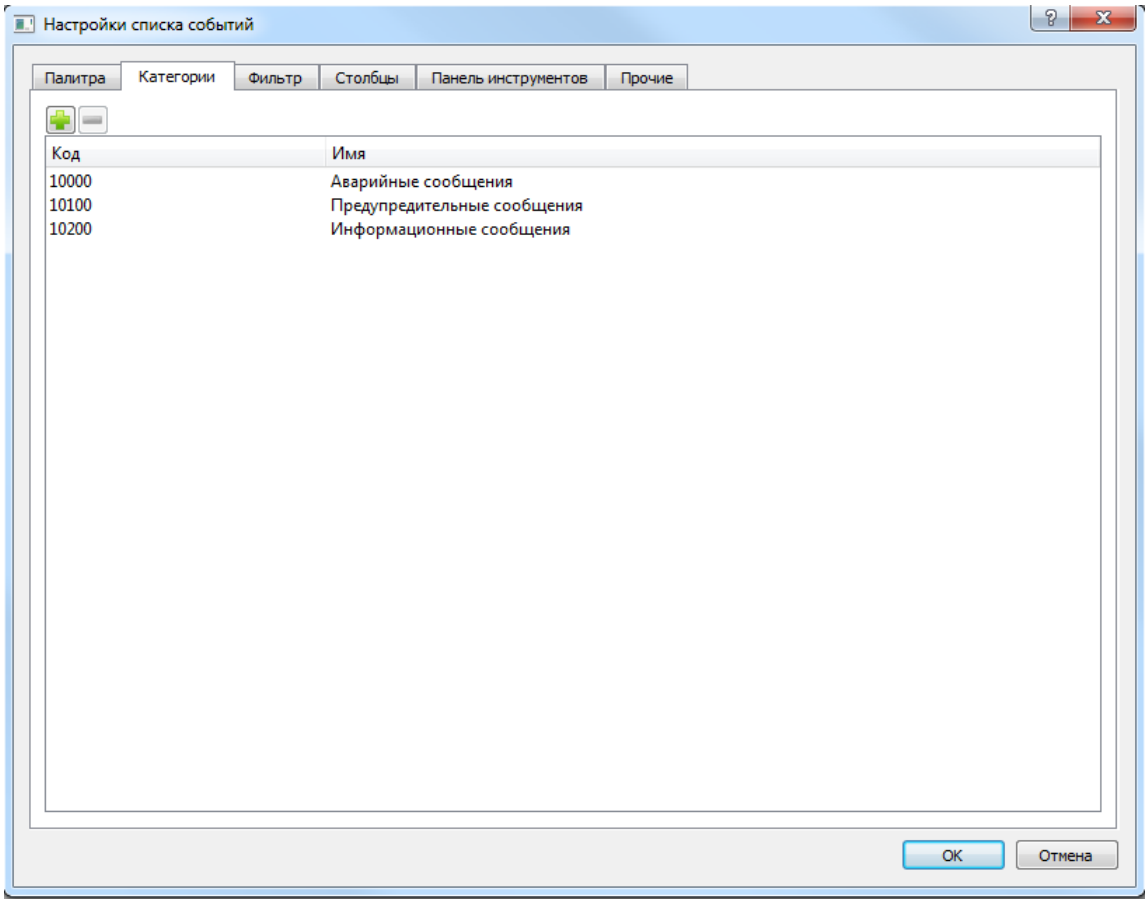


Рисунок 2.21 - Вкладка "Категории" диалога настройки списка событий

Первичные настройки фильтра задаются оператором на вкладке "Фильтр". Фильтровать события можно по дате и времени, по категориям, состояниям, источнику, сообщению, оператору, системным дате и времени. Вкладка "Фильтр" диалога настройки отображения списка событий представлена на рис. 2.22

TEventViewer

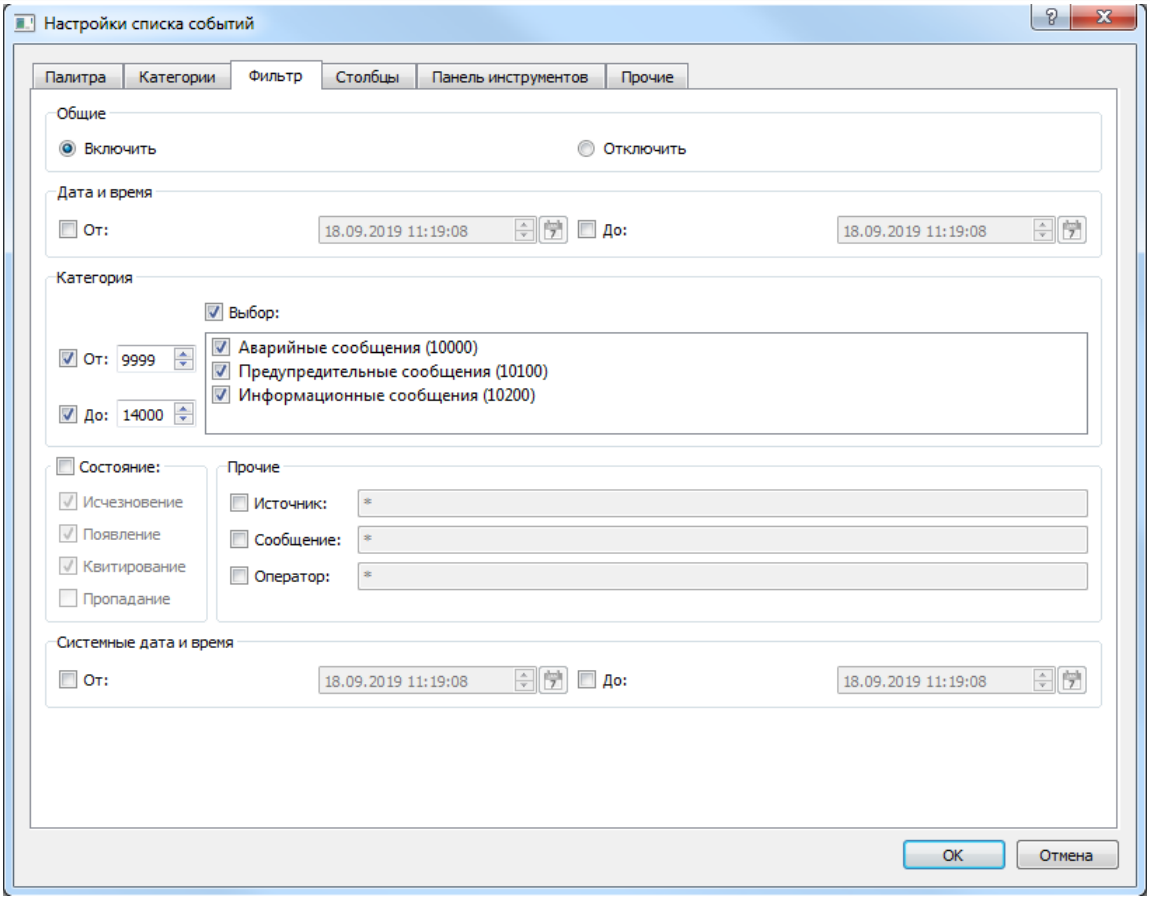


Рисунок 2.22 - Вкладка "Фильтр" диалога настройки списка событий

В разделе Фильтр - Прочие, где можно настраивать фильтрацию по столбцу Источник, Сообщение и Оператор, применяются следующие условные записи:

- ABC* - любая строка, начинающаяся с символов ABC;
- *ABC - любая строка, заканчивающаяся символами ABC;
- * - любая строка;
- ~ABC* - любая строка, которая не начинается с ABC;
- (~A*|*ABC) - любая строка, которая не начинается с символа A или заканчивается символами ABC (условий ИЛИ может быть несколько);
- \ - экранирующий символ. Если в вашей строке данных, к примеру, есть символ ~ и необходимо настроить по нему поиск, то нужно в строке условий написать \~, тогда данный символ не будет восприниматься как спец символ. Так же можно искать по символам \\, \(, \), \|.

Примечание. Для фильтрации по времени лучше использовать Системные дата и время, т.к. по данному критерию фильтр отрабатывает намного быстрее.

На вкладке "Столбцы" оператор выбирает, какие столбцы отображать при отображении списка событий, а также определяет порядок следования столбцов. Кнопки и предназначены для изменения порядка следования столбцов. Таблица элементов, расположенная ниже, позволяет включить или отключить отображение каждого столбца и задать его ширину. Ширина указывается в символах. Значение ширины столбца, равное -1, соответствует значению по умолчанию, которое равно 10 символам. Вкладка "Столбцы" диалога настройки отображения списка событий представлена на рис. 2.23. На рис. 2.24 отображается меню выбора используемого архива.

TEventViewer

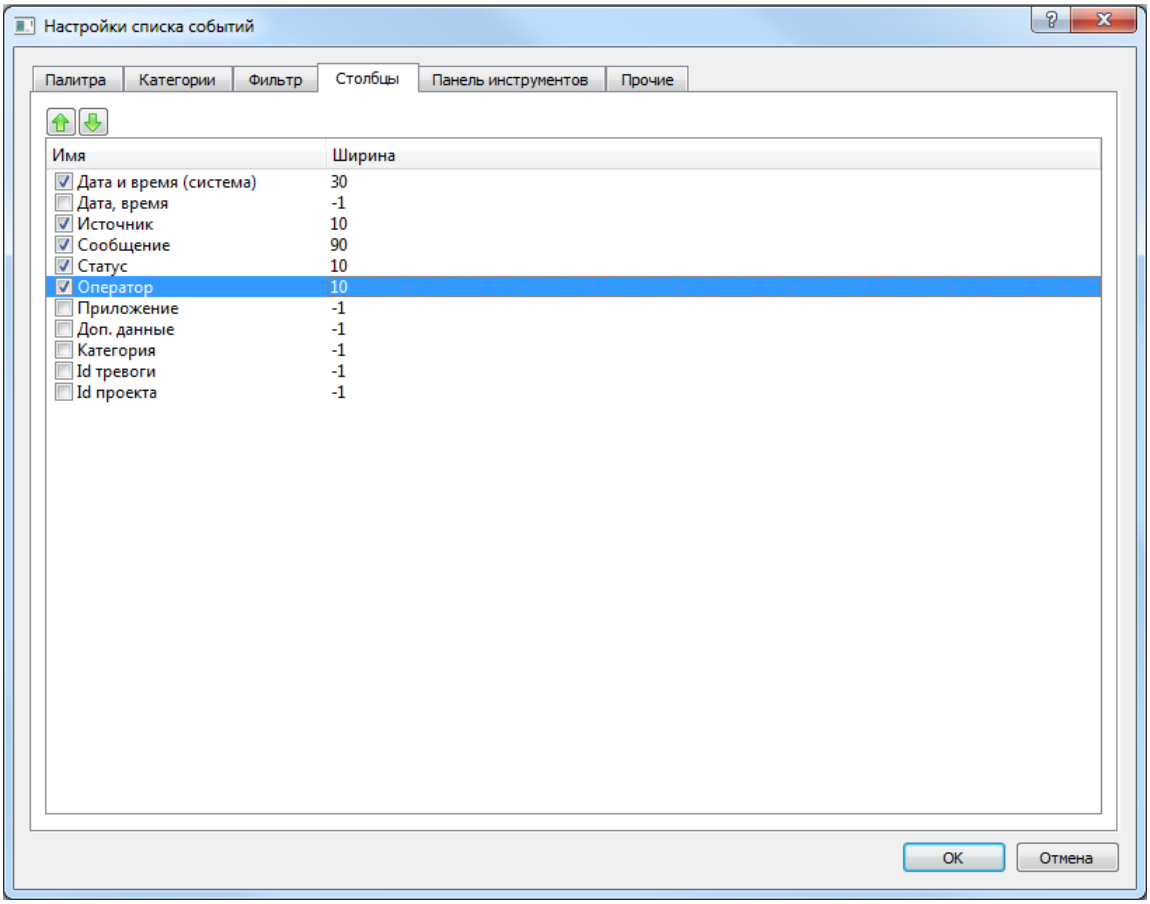


Рисунок 2.23 - Вкладка "Столбцы" диалога настройки списка событий

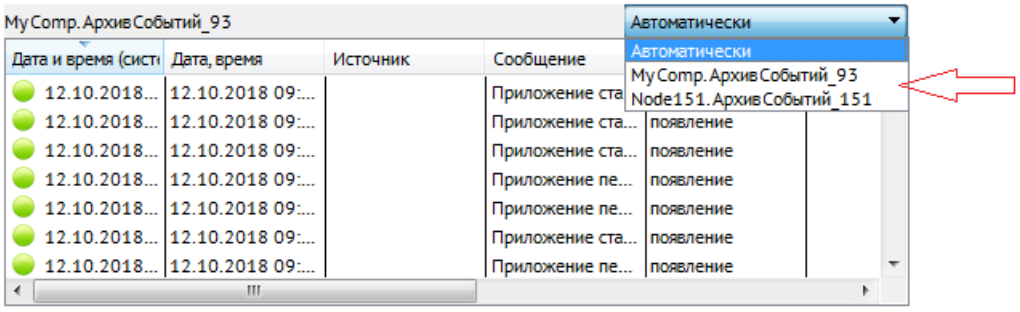


Рисунок 2.24 - Меню выбора используемого архива событий

На вкладке "Панель инструментов" оператор выбирает какие кнопки будут располагаться на панели инструментов блока на мнемосхеме и их порядок (см. рис. 2.25).. Кнопки и предназначены для изменения порядка следования кнопок. Таблица элементов, расположенная ниже позволяет включить или отключить отображение каждой кнопки, а так же изменить ее название. Возможность изменения названия кнопки появляется после двойного нажатия левой кнопки мышки на соответствующем элементе. В столбце "Описание" содержится информация о назначении каждой отдельной кнопки.

Ниже расположены три настройки, позволяющие отображать панель управления на соответствующем блоке мнемосхемы, отображать источник данных, из которого в данный момент берутся события и возможность выбора источника данных.

Назначение кнопок:

- **Фильтр** - настройки фильтра отображения событий;

TEventViewer

- **Сохранить** - сохранение всех событий, прошедших через фильтр в файл;
- **Сохранить все** - сохранение всех событий в файл;
- **Сохранить выдел.** - сохранить выделенные события в файл;
- **Напечатать** - печать всех событий, прошедших через фильтр;
- **Напечатать все** - печать всех событий;
- **Напечатать выдел.** - печать выделенных событий;
- **Квитировать** - квитировать выделенные события;
- **Квитировать отфильтр.** - квитировать события, прошедшие через фильтр;
- **Квитировать все** - квитировать все события;
- **Доп. действие** - Дополнительное действие, ID которого задается в переменной ACTION_ID блока. Подробное описание см. в документе SCADA-система "Соната" Руководство пользователя КУНИ.505200.023-01.01 95 в разделе Описание работы с приложением BRIDGE (межпроектный обмен).;

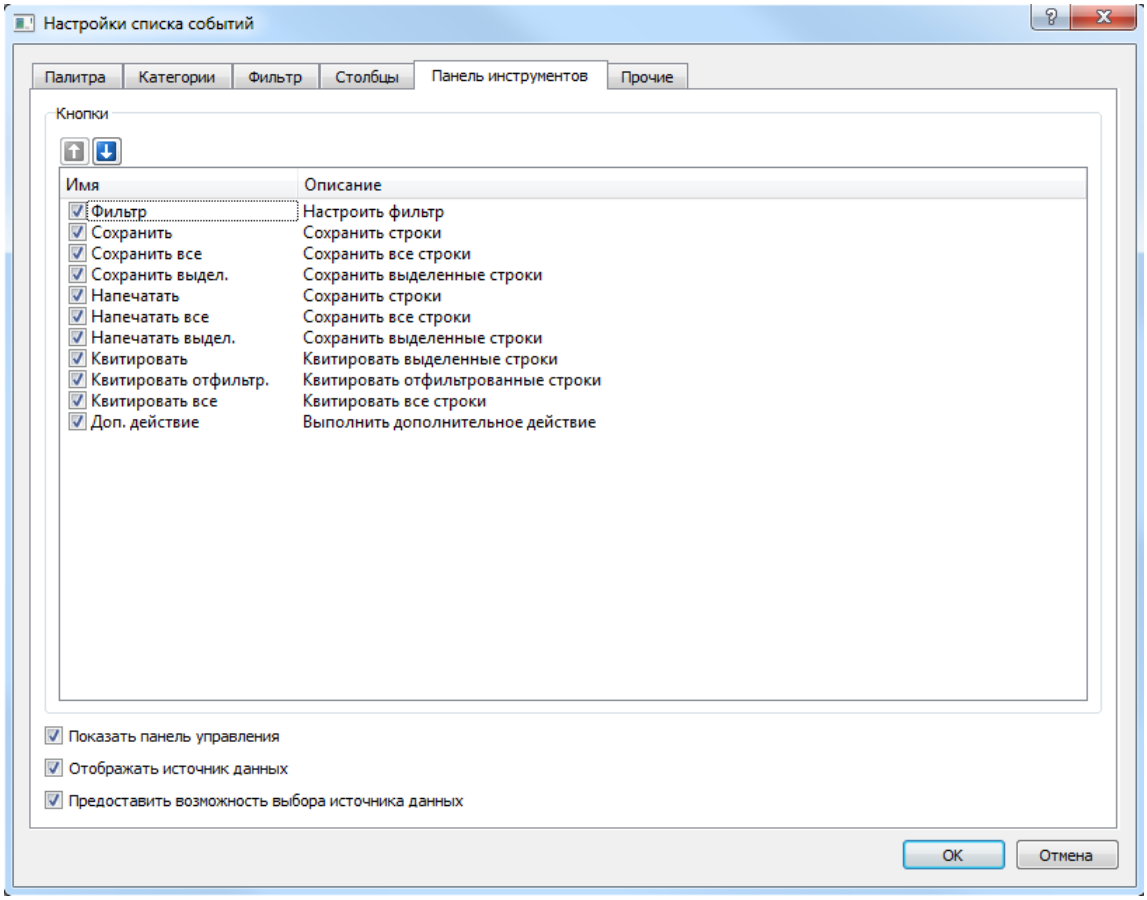


Рисунок 2.25 - Вкладка настроек Панель инструментов

На вкладке настроек Прочие располагаются настройки Печати, Сохранения, Источника данных и Загрузки событий (см. рис. 2.26).

TEventViewer

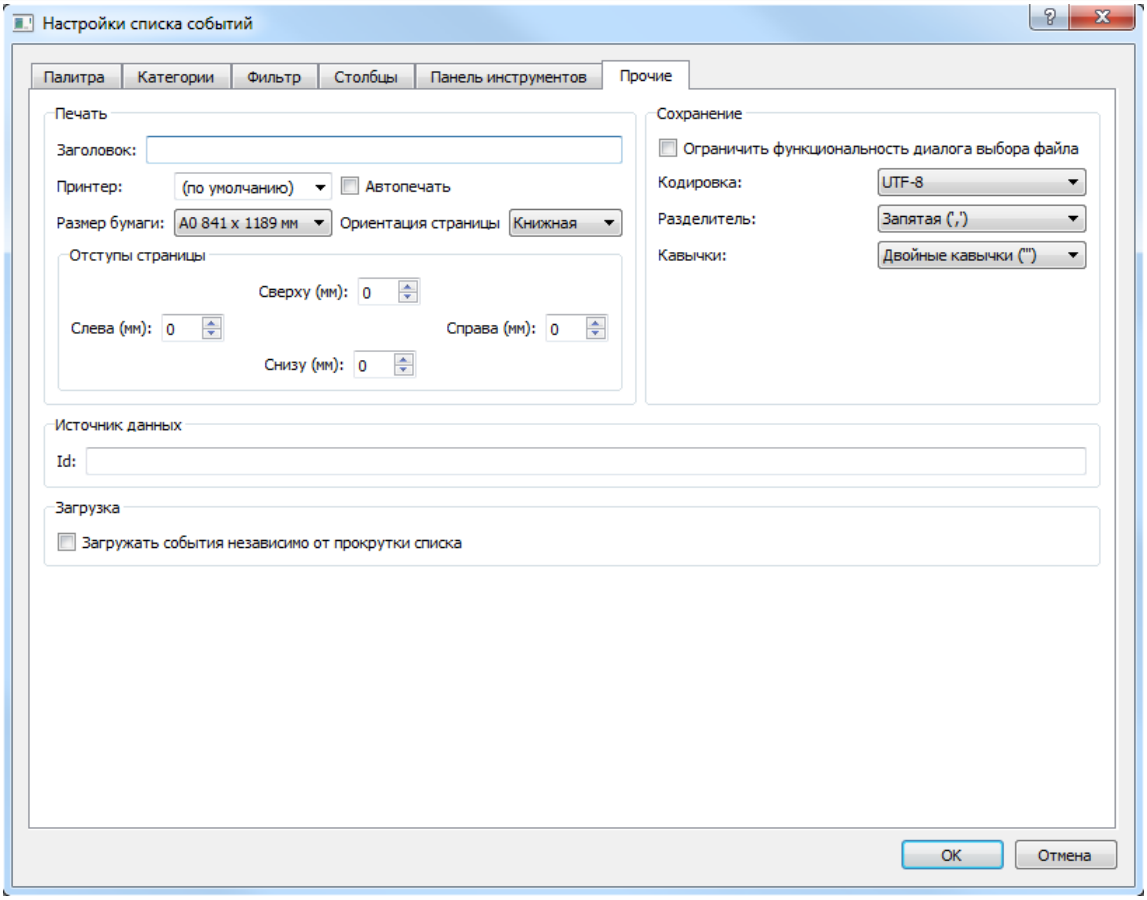


Рисунок 2.26 - Вкладка настроек Прочие

Настройки печати

Поле ввода заголовка служит для ввода текста, выводимого над таблицей событий в ходе печати.

Выпадающий список "Принтер" предназначен для выбора устройства, которое будет выбрано в качестве предлагаемого принтера для печати. Первым элементом выпадающего списка является строка "(по умолчанию)", при выборе которой в качестве принтера для печати будет предложен принтер по умолчанию того компьютера, на котором будет выполняться печать. Имя принтера может быть выбрано как из выпадающего списка, так и введено вручную.

Если включить опцию "Автопечать", то блок не выведет диалог печати, а сразу же отправит данные на выбранный принтер. Если же данная опция отключена, то при вызове события печати блок выведет системный диалог, в котором можно будет выбрать принтер и указать требуемые параметры печати.

Ниже выпадающего списка выбора принтера расположены выпадающие списки для выбора размера и ориентации бумаги.

В нижней части диалога расположены элементы редактирования, предназначенные для задания отступов от края страницы. Значения задаются в миллиметрах.

Настройки сохранения

Файл со списком событий сохраняется в формате CSV (текстовый формат, предназначенный для предоставления табличных данных).

На странице настроек сохранения списка событий находятся следующие поля:

- Ограничить функциональность диалога выбора файла - данный функционал позволяет ограничить доступ пользователя к другим папкам системы кроме папки сохранения;
- Кодировка - можно установить в какой кодировке будет сохраняться текст;

TEventViewer
- Разделитель - разделитель полей таблицы при сохранении в текстовый формат; - Кавычки - необходимо указать каким видом кавычек обрамляются зарезервированные символы. К примеру, чтобы открыть файл формата CSV в программе Microsoft Exel, необходимо выбрать кодировку windows-1251, разделитель - ; , а кавычки выбираются в зависимости от вашего содержимого в файле. ***Настройки источника данных*** В поле ввода **Id** необходимо указать из какого источника будут браться события (источник можно указать только один). Подробное описание см. в документе SCADA-система "Соната" Руководство пользователя КУНИ.505200.023-01.01 95 в разделе Описание работы с приложением BRIDGE (межпроектный обмен). ***Настройки загрузки*** Загружать события независимо от прокрутки списка - если активировать данную опцию, то список событий будет подгружаться в EventViewer автоматически. Если не включать данную опцию, то список событий будет подгружаться только при прокрутке списка вниз.

2.1.3.4.4. Управление системой

CONTROL_CENTER	
Центр управления	
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать события работы от «мыши»
font	Тип TFont. Шрифт объекта
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. TRUE, если объект можно перемещать
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
projectPath	Тип STRING. Подключается к проекту по указанному пути
FILE_PATH_TEMPLATE	Тип STRING. Шаблон имени файла, в котором будут сохраняться данные. По умолчанию директорией сохранения файла является рабочая папка с проектом.

CONTROL_CENTER	
	Обычные символы в названии файла не преобразуются. Символы, определяющие преобразования, предваряются символом %. В итоговом названии файла их заменят следующие символы: - %Y - показывает год как четырехразрядное десятичное число (с указанием века); - %y - показывает год как двухразрядное число от 00 до 99 (без указания века); - %m - показывает месяц как десятичное число от 01 до 12; - %d - день месяца в десятичной форме (от 01 до 31); - %j - показывает день года как десятичное число от 001 до 366; - %H - показывает час как десятичное число от 00 до 23; - %M - показывает минуты как десятичное число от 00 до 59; - %S - отображает секунды в десятичной форме от 00 до 61; - %ext - данный параметр используется для добавления раширения к имени файла. Данное расширение разное, в зависимости от того, какой файл с данными нужно сохранить: файл с данными о лицензии, файл с информацией о системе, файл с данными о целостности или файл с логами. Значения ключа %ext: li - для лицензии; si - для системной информации; cs - для контрольных сумм; log - для логов. - Пример шаблона имени файла - "Data from %Y%m%d--%H-%M-%S.%ext" (использовать символ **:** в имени файла, к примеру, для такой записи как %H:%M:%S, нельзя, его можно использовать только для указания пути к файлу в операционных системах семейства Windows, к примеру, "D:/Data from %Y%m%d.%ext"). Когда данный входной параметр определен, диалог сохранения данных в файл выводиться не будет
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте

CONTROL_CENTER	
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

2.1.3.4.4.1. Сигналы

TTrend	
Блок показа графика текущего значения сигнала	
Входные события	
START	Начать накопление данных и отрисовку графика
STOP	Закончить накопление данных
CLEAR	Удалить все накопленные данные (очистить область отрисовки)
SAVE	Сохранить накопленные данные. Данное событие появляется, если разработчик указал внешнее управление сохранением в настройках объекта
PRINT	Напечатать данные. Данное событие появляется, если разработчик указал внешнее управление печатью в настройках объекта
Входные переменные	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать события работы от «мыши»
font	Тип TFont. Шрифт объекта
hint	Тип STRING. Всплывающая подсказка объекта
moveable	Тип BOOL. TRUE, если объект можно перемещать

TTrend	
pos	Тип TPos. Положение объекта
size	Тип TSize. Размер объекта
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
TITLE	Тип STRING. Наименование графика
DATA	Тип LREAL. Данные
LINE_COLOR	Тип TColor. Цвет линии графика
LINE_WIDTH	Тип INT. Толщина линии графика
DATA_MIN	Тип LREAL. Минимальное значение отображаемых данных
DATA_MAX	Тип LREAL. Максимальное значение отображаемых данных
Y_TITLE	Тип STRING. Подпись оси ординат
HIDE_SCALE	Тип BOOL. Если значение данной переменной равно TRUE, то ось ординат скрыта
Y_AXIS_MIN	Тип LREAL. Нижнее значение (в %) оси ординат. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью ординат
Y_AXIS_MAX	Тип LREAL. Верхнее значение (в %) оси ординат. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью ординат
X_AXIS_LENGTH	Тип TIME. Длина оси абсцисс. Данная переменная появляется в интерфейсе объекта, если в настройках указано внешнее управление осью абсцисс
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта

TTrend	
SAVED	Завершено сохранение данных графика в файл
PRINTED	Завершена печать данных графика
Выходные переменные	
MARKERA_POS	Тип DT. Положение маркера A
MARKERB_POS	Тип DT. Положение маркера B
MARKERA_VALUE	Тип ANY_MAGNITUDE. Значение данных в точке пересечения графика с маркером A
MARKERB_VALUE	Тип ANY_MAGNITUDE. Значение данных в точке пересечения графика с маркером B
Описание	

Данный тип функционального блока предназначен для отображения графика текущего значения подключенного параметра за указанный период времени назад, начиная от текущего момента. Внешний вид блока на мнемосхеме приведен на рис. 2.27

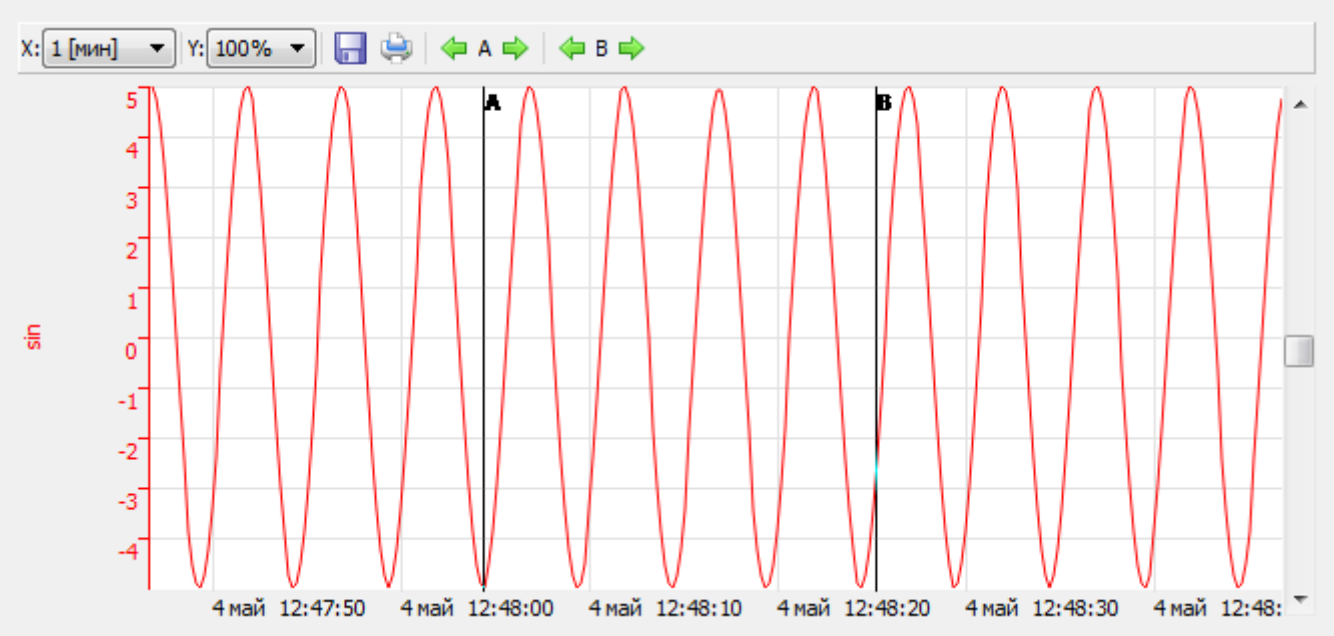


Рисунок 2.27 - Блок типа TTrend, размещенный на мнемосхеме

В верхней части объекта размещена панель управления. Остальная часть представляет собой поле, в котором отрисовывается график.

Панель инструментов содержит в себе элементы редактирования для выбора размера оси абсцисс ("X: 1 [мин] "), масштаба оси ординат ("Y: 100% "), кнопки сохранения(" ") и печати (" ") данных, а также кнопки смещения маркеров A (" A ") и B (" B ").

Выпадающий список длин оси абсцисс присутствует в панели инструментов, если в настройках объкта не указано внешнее управление осью абсцисс.

Выпадающий список масштаба оси ординат присутствует в панели инструментов, если в настройках объкта не указано внешнее управление осью ординат.

TTrend
Кнопка сохранения присутствует в панели инструментов, если в настройках объекта не указано внешнее управление сохранением. При нажатии на кнопку сохранения программа выводит диалог выбора файла для сохранения. Пользователь может сохранить данные в формате txt либо в формате png. При выборе текстового формата данные сохраняются в табличном виде. При выборе формата png-изображения данные сохраняются в виде картинки. Кнопка печати присутствует в панели инструментов, если в настройках объекта не указано внешнее управление печатью. При нажатии на кнопку печати программа выводит диалог печати, если в настройках печати не выбрана опция автопечати. Если же указана автопечать, то печать осуществляется на выбранный принтер с указанными настройками. Кнопки "  " и "  " сдвигают выбранный маркер на один отсчет влево или вправо соответственно. Поле отрисовки графиков можно условно разделить на четыре области. Первая область - это область вертикальных осей, расположенная в левой части поля. Вторая область - это область оси абсцисс, в которой выводятся значения временных меток вертикальных осей сетки. Выше нее расположена область графиков сигналов логического типа (BOOL). Вся остальная часть отведена под отрисовку графиков числовых величин. Нажав на левую кнопку «мыши» в поле графиков и проведя указателем в сторону, можно получить два маркера - A и B, пр этом первый маркер останется на месте нажатия левой кнопки «мыши», а второй - в месте ее отпускания. Маркеры могут быть перемещены независимо - для этого необходимо подвести указатель «мыши» к маркеру, нажать на левую кнопку «мыши» и переместить его, удерживая нажатой левую кнопку «мыши». Перемещая маркеры, можно их совместить - в этом случае маркер останется только один до следующего разделения.
Настройка
Окно настроек представляет собой многостраничный диалог, содержащий вкладки общих настроек объекта и настроек печати. Страница общих настроек приведена на рис. 2.28.

TTrend

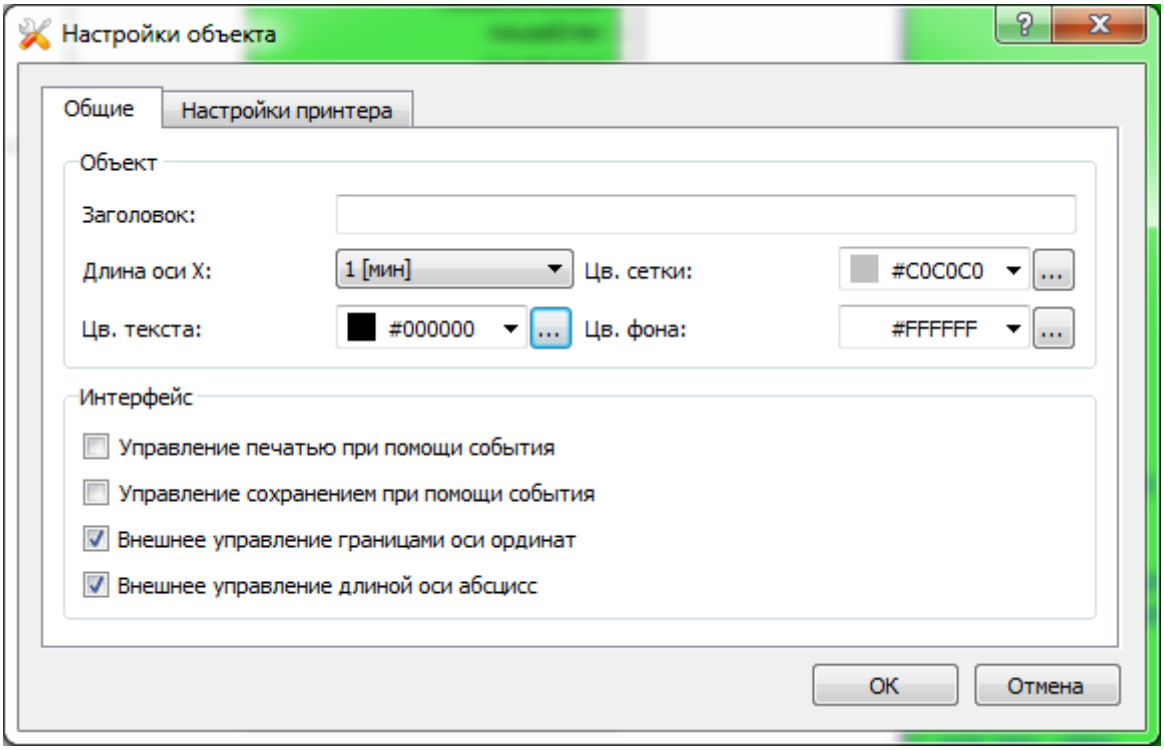


Рисунок 2.28 - Страница общих настроек

Группа элементов "Объект" предназначена для редактирования настроек области графика. В поле ввода "Заголовок" вносится наименование графика, выводимое при сохранении и печати. Используя выпадающий список "Длина оси X", разработчик выбирает расстояние между левым и правым краем оси абсцисс. При помощи остальных элементов данной группы разработчик выбирает цвета текста, сетки и фона графика.

Выдающий список для оси абсцисс скрыт, если в группе настройки интерфейса указано внешнее управление осью абсцисс.

Группа элементов "Интерфейс" предназначена для редактирования настроек интерфейса объекта.

Выбор опции "Управление печатью при помощи события" приводит к появлению входного события PRINT и скрытию кнопки печати с панели инструментов.

Выбор опции "Управление сохранением при помощи события" приводит к появлению входного события SAVE и скрытию кнопки сохранения с панели инструментов.

Выбор опции "Внешнее управление границами оси ординат" приводит к появлению входных переменных Y_AXIS_MIN и Y_AXIS_MAX и скрытию выпадающего списка масштаба оси ординат с панели инструментов.

Выбор опции "Внешнее управление длиной оси абсцисс" приводит к появлению входной переменной X_AXIS_LENGTH и скрытию выпадающего списка длин оси абсцисс с панели инструментов.

Страница настроек печати приведена на рис. 2.29.

TTrend

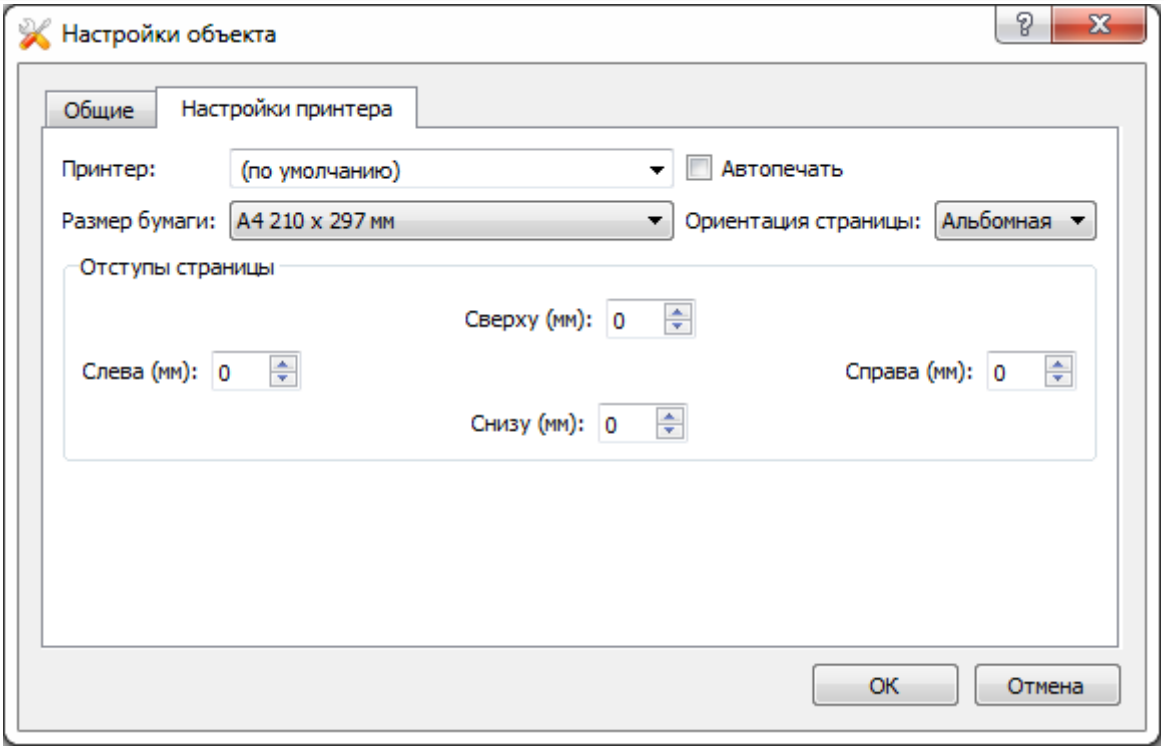


Рисунок 2.29 - Страница настроек печати

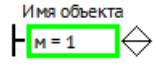
Выпадающий список "Принтер" предназначен для выбора устройства, которое будет выбрано в качестве предлагаемого принтера для печати. Первым элементом выпадающего списка является строка "(по умолчанию)", при выборе которой в качестве принтера для печати будет предложен принтер по умолчанию того компьютера, на котором будет выполняться печать. Имя принтера может быть выбрано как из выпадающего списка, так и введено вручную.

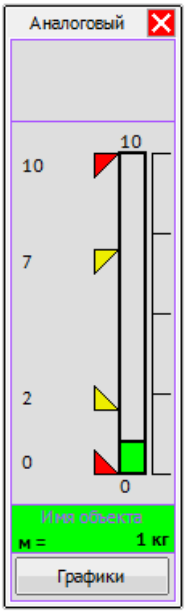
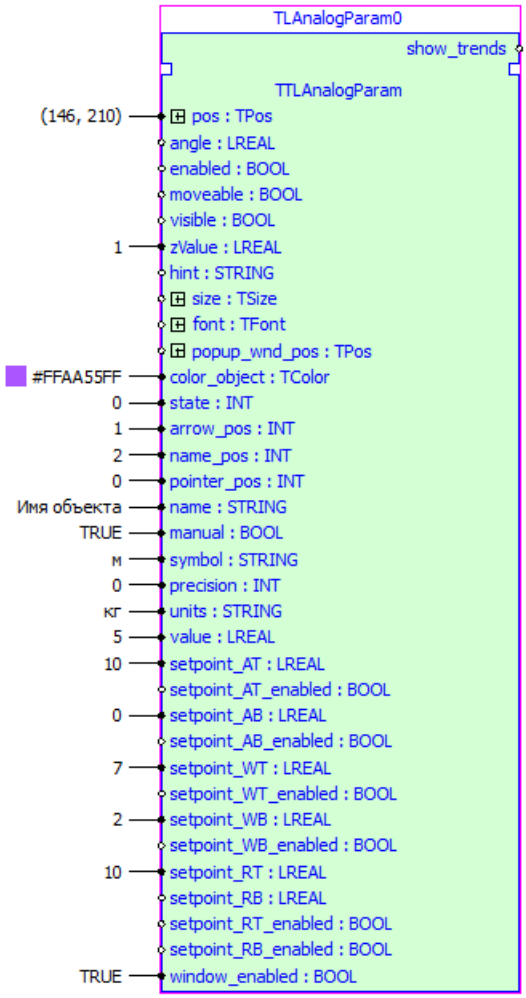
Если включить опцию "Автопечать", то блок не выведет диалог печати, а сразу же отправит данные на выбранный принтер. Если же данная опция отключена, то при вызове события печати блок выведет системный диалог, в котором можно будет выбрать принтер и указать требуемые параметры печати.

Ниже выпадающего списка выбора принтера расположены выпадающие списки для выбора размера и ориентации бумаги.

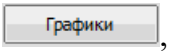
В нижней части диалога расположены элементы редактирования, предназначенные для задания отступов от края страницы. Значения задаются в миллиметрах.

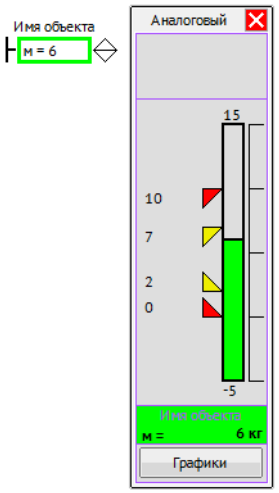
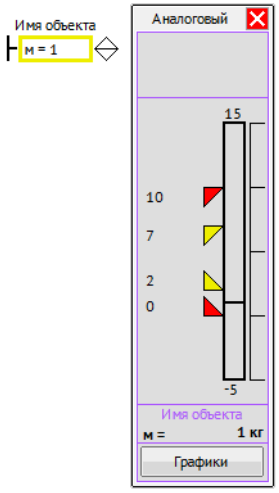
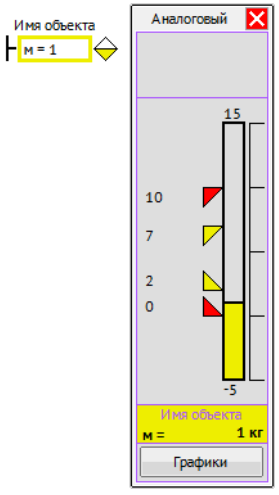
2.1.3.5. Техническая библиотека

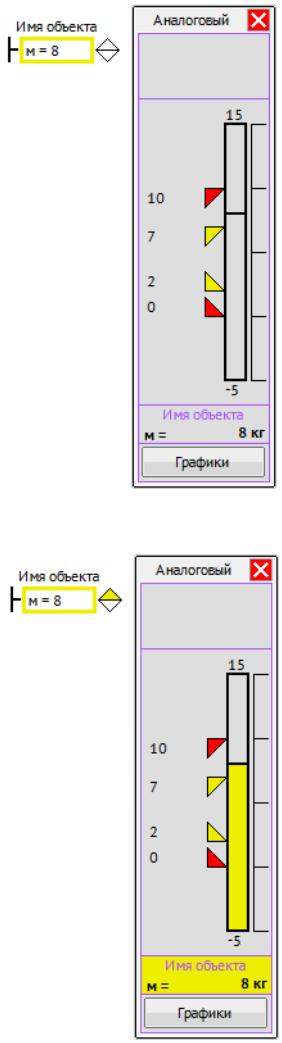
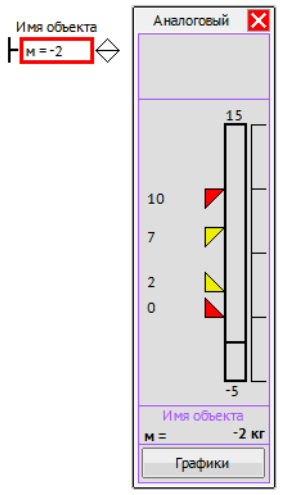
TTLAnalogParam	
Объект аналоговый датчик	
1. Отображение на мнемосхеме:  2. Вторичный видеокадр с управляющими кнопками	Отображение содержимого объекта

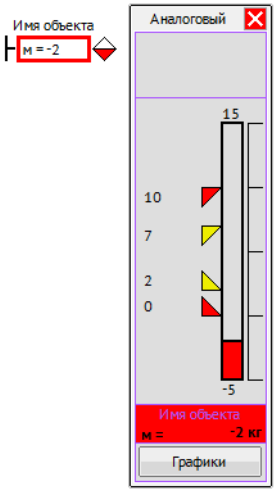
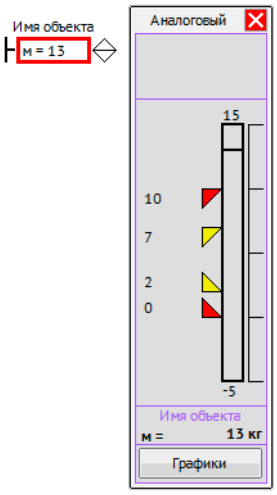
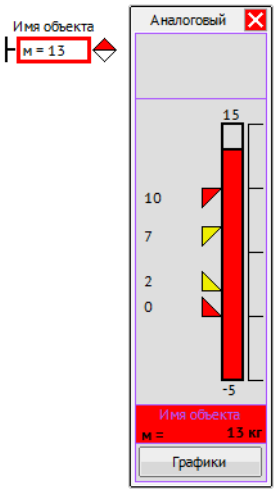
TTLAnalogParam	
	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. TRUE, если объект доступен для управления и может порождать события работы от «мыши»
moveable	Тип BOOL. TRUE, если объект можно перемещать
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_object	Тип TColor. Цвет контура объекта (на вторичном видеокадре)

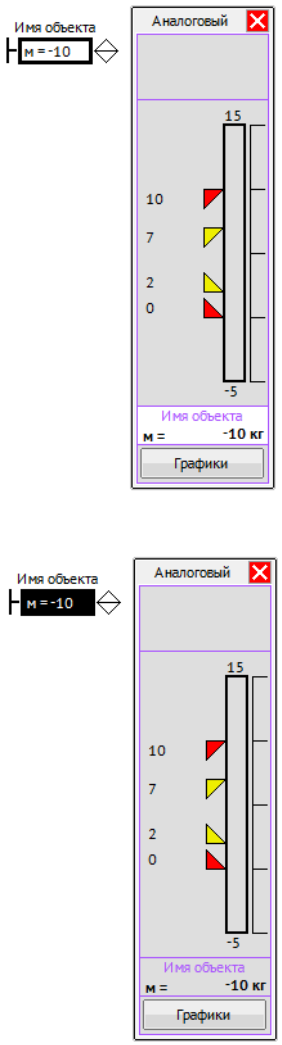
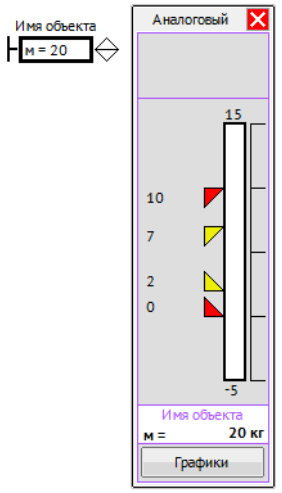
TTLAnalogParam	
state	Тип INT. Возможные состояния объекта (описание смотрите ниже)
arrow_pos	Тип INT. Позиция индикатора-стрелки, показывающих превышение или значение меньшее, чем уставки  (-1 - убрать отображение; 0 - слева; 1 - справа; 2 - сверху; 3 - снизу)
name_pos	Тип INT. Позиция заголовка объекта (0 - слева; 1 - справа; 2 - сверху; 3 - снизу)
pointer_pos	Тип INT. Расположение точки соединения (0 - слева, 1 - справа, 2 - сверху, 3 - снизу)
name	Тип STRING. Заголовок объекта
manual	Тип BOOL. Признак вывода параметра в ремонт
symbol	Тип STRING. Символ параметра
precision	Тип INT. Точность значения параметра
units	Тип STRING. Единицы измерения значения параметра
value	Тип LREAL. Значение параметра
value_specifier	Тип INT. Формат отображения значения параметра на мнемосхеме: 0 - отображение в десятичном формате, 1 - отображение в экспоненциальном формате
style	Тип INT. Стиль отображения значения параметра на мнемосхеме: 0 - без отображения единиц измерения, 1 - с отображением единиц измерения
setpoint_AT	Тип LREAL. Значение верхней аварийной уставки (отображается на вторичном видеокадре )
setpoint_AT_enabled	Тип BOOL. Разрешение использования верхней аварийной уставки
setpoint_AB	Тип LREAL. Значение нижней аварийной уставки (отображается на вторичном видеокадре )
setpoint_AB_enabled	Тип BOOL. Разрешение использования нижней аварийной уставки
setpoint_WT	Тип LREAL. Значение верхней предупредительной уставки (отображается на вторичном видеокадре )

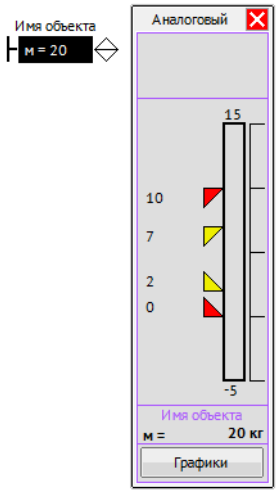
TTLAnalogParam	
setpoint_WT_enabled	Тип BOOL. Разрешение использования верхней предупредительной уставки
setpoint_WB	Тип LREAL. Значение нижней предупредительной уставки (отображается на вторичном видеокадре )
setpoint_WB_enabled	Тип BOOL. Разрешение использования нижней предупредительной уставки
setpoint_RT	Тип LREAL. Верхняя граница диапазона измерения
setpoint_RT_enabled	Тип BOOL. Разрешение использования сигнализации о превышении верхней границы диапазона измерения
setpoint_RB	Тип LREAL. Нижняя граница диапазона измерения
setpoint_RB_enabled	Тип BOOL. Разрешение использования сигнализации о значении меньшем, чем нижняя граница диапазона измерения
window_enabled	Тип BOOL. Разрешение использования вторичного видеокадра с управляющими кнопками
font	Тип TFont. Шрифт объекта
Выходные события	
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
show_trends	Выходное событие, которое вырабатывается по нажатию на кнопку  , находящуюся на вторичном видеокадре с управляющими кнопками (применяется для подачи на входное событие show функционального блока ARCHIVE_TREND_WINDOW)

TTLAnalogParam			
Описание			
Возможные состояния аналогового датчика			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
показания в норме		Отображаются показания датчика. Для отображения используется зелёный цвет	0
выход показания за нижнюю предупредительную уставку	 	Для отображения используется жёлтый цвет. Происходит мигание нижней стрелки на объекте и мигают показание датчика и область со значением на вторичном видеокадре	1

TTLAnalogParam			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
выход показания за верхнюю предупредительную уставку		Для отображения используется жёлтый цвет. Происходит мигание верхней стрелки на объекте и мигают показание датчика и область со значением на вторичном видеокадре	2
выход показания за нижнюю аварийную уставку		Для отображения используется красный цвет. Происходит мигание нижней стрелки на объекте и мигают показание датчика и область со значением на вторичном видеокадре	3

TTLAnalogParam			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
			
выход показания за верхнюю авраийную уставку	 	Для отображения используется красный цвет. Происходит мигание верхней стрелки на объекте и мигают показание датчика и область со значением на вторичном видеокадре	4

TTLAnalogParam			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
выход показания за нижнюю границу диапазона		Происходит мигание объекта (белый/черный) и мигание в области со значением на вторичном видеокадре (белый/серый)	5
выход показания за верхнюю границу диапазона		Происходит мигание объекта (белый/черный) и мигание в области со значением на вторичном видеокадре (белый/серый)	6

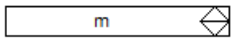
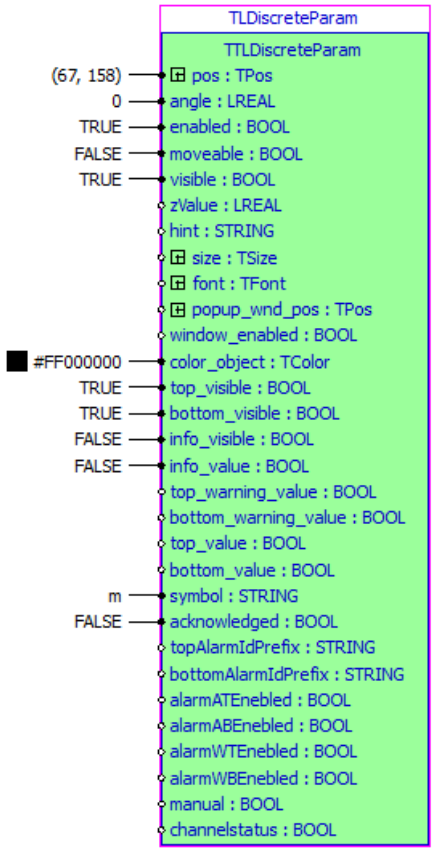
TTLAnalogParam			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
			

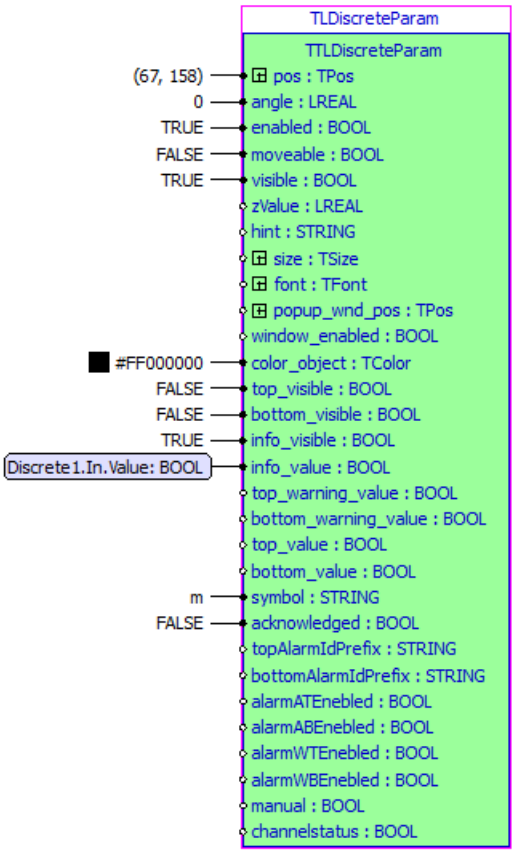
TTLButton	
Кнопка Данный элемент реализован без использования Qt и отображается одинаково в независимости от операционной системы.	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
checkable	Тип BOOL. Если значение равно TRUE, то кнопка работает с залипанием при нажатии, если значение равно FALSE, то кнопка работает без залипания при нажатии.
color	Тип TColor. Цвет кнопки
hightlightColor	Тип TColor. Цвет кнопки при наведении курсора
text	Тип STRING. Строка текста
font	Тип TFont. Шрифт объекта
text_color	Тип TColor. Цвет текста

TTLButton	
tag	Тип INT. Дополнительные данные кнопки
corner	Тип REAL. Скругление краёв (0 - 10)
checked	Тип BOOL. Состояние кнопка была нажата - TRUE, кнопка не была нажата - FALSE. Данный вход работает, если вход checkable равен TRUE
Выходные события	
clicked	Щелчок "мыши" на объекте
pressed	Нажатие кнопки «мыши» на объекте
released	Отпускание кнопки «мыши» на объекте
Выходные переменные	
o_tag	Тип INT. Копия значения tag
o_checked	Тип BOOL. Выходное значение состояния кнопки: TRUE - нажата, FALSE - не нажата. Данный выход работает, если вход checkable равен TRUE

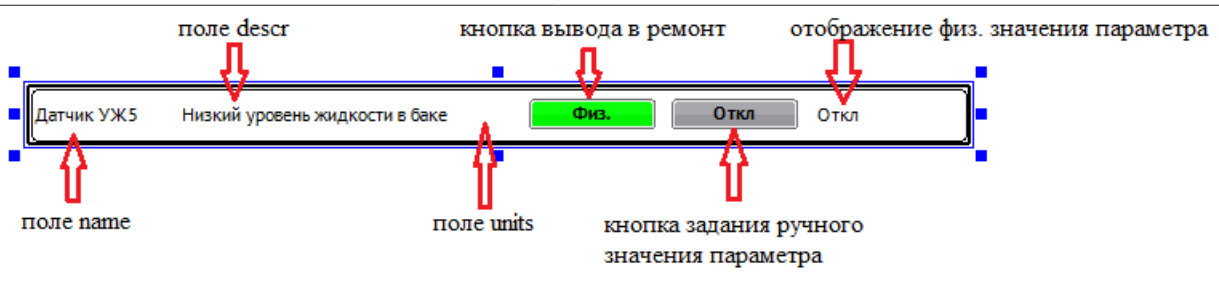
TTLCheckbox	
Чекбокс	
Входные события	
reset	Не используется (будет удалён)
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
frameColor	Тип TColor. Цвет контура
bgColor	Тип TColor. Цвет фона
checkColor	Тип TColor. Цвет символа нажатия
i_state	Тип BOOL. Входное значение состояния
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте

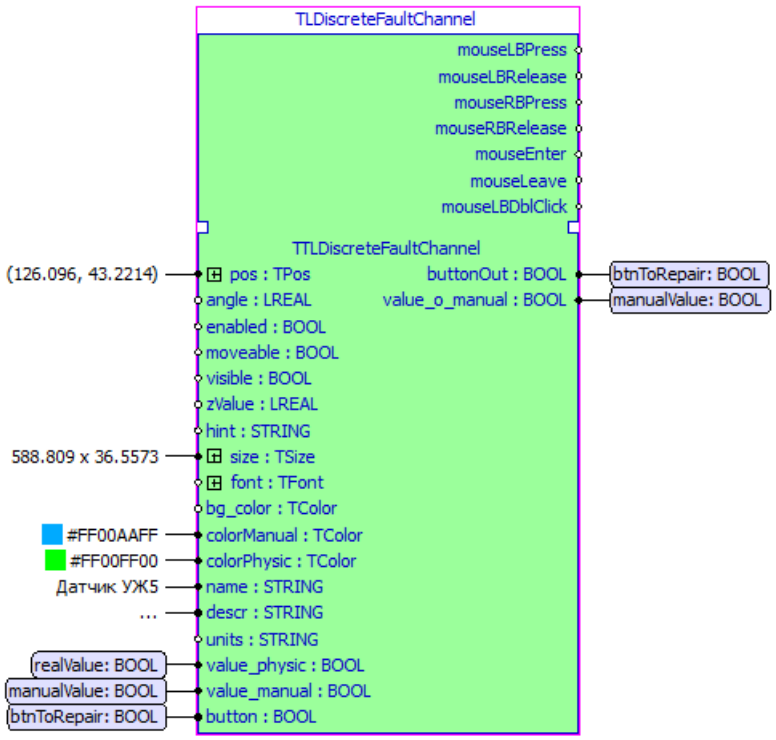
TTLCheckbox	
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
changed	Возникает при изменении состояния элемента checkbox
Выходные переменные	
o_state	Тип BOOL. Выходное значение состояния

TTLDiscreteParam	
Объект дискретный датчик	
Отображение на мнемосхеме: 1 вариант Данный вариант может использовать значения от 4 дискретных датчиков и показывать предупредительную и аварийную цветовую сигнализацию. В качестве предупредительных значений используются top_warning_value и bottom_warning_value, в качестве аварийных значений используются top_value и bottom_value. Общий вид  Предупредительная сигнализация о превышении верхнего значения  Аварийная сигнализация о превышении верхнего значения  Аналогично отображается предупредительная и аварийная сигнализация о достижении нижних пороговых значений.	Отображение содержимого объекта 
Отображение на мнемосхеме: 2 вариант Данный вариант может отображать один дискретный параметр.	Отображение содержимого объекта

TTLDiscreteParam	
Выключен  Включен  	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
font	Тип TFont. настройки шрифта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
window_enabled	Тип BOOL.
color_object	Тип TColor. Цвет объекта
top_visible	Тип BOOL. Видимость индикатора превышения верхнего уровня ⏢

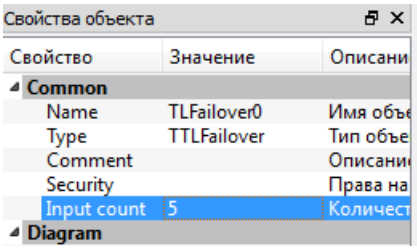
TTLDiscreteParam	
bottom_visible	Тип BOOL. Видимость индикатора занижения нижнего уровня ▽
info_visible	Тип BOOL. Отображение индикатора дискретного сигнала ○
info_value	Тип BOOL. Значение дискретного сигнала. Если оно равно FALSE, то индикатор ○, если равно TRUE, то индикатор ●
top_value	Тип BOOL. TRUE - превышение верхнего уровня (индикатор △ мигает красным цветом)
bottom_value	Тип BOOL. TRUE - занижение нижнего уровня (индикатор ▽ мигает красным цветом)
symbol	Тип STRING. Символ измеряемого параметра
acknowledged	Тип BOOL. Квитирование
topAlarmIdPrefix	Тип STRING. Префикс для ID тревоги по превышению верхнего значения
bottomAlarmIdPrefix	Тип STRING. Префикс для ID тревоги по занижению нижнего значения
alarmATEnabled	Тип BOOL. Создавать или нет тревогу по превышению верхнего значения
alarmABEnabled	Тип BOOL. Создавать или нет тревогу по занижению нижнего значения
alarmWTEnabled	Тип BOOL. Создавать или нет предупреждение по превышению предупредительного верхнего значения
alarmWBEnabled	Тип BOOL. Создавать или нет предупреждение по занижению предупредительного нижнего значения
manual	Тип BOOL. Вывод параметра в ручной режим
channel status	Тип BOOL. Статус канала: FALSE - исправен, TRUE - неисправен

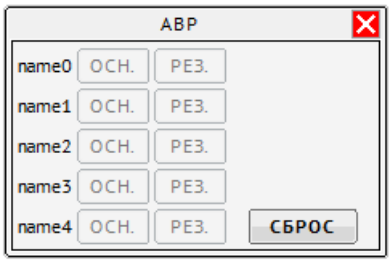
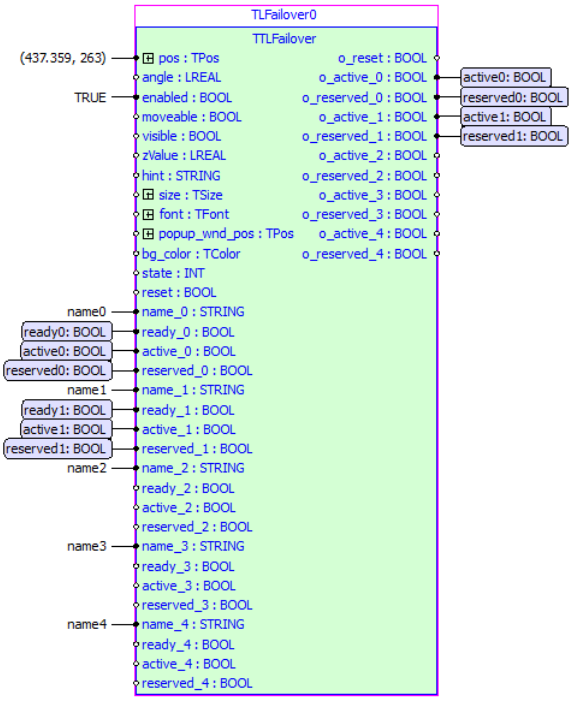
TTLDiscreteFaultChannel
Объект вывода в ремонт дискретного параметра

Отображение содержимого объекта

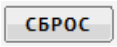
TTLDiscreteFaultChannel


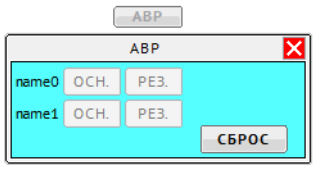
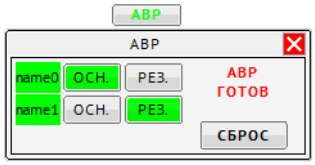
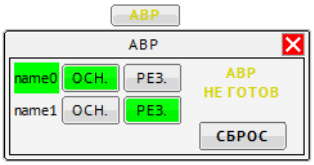
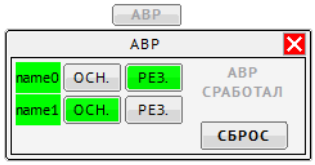
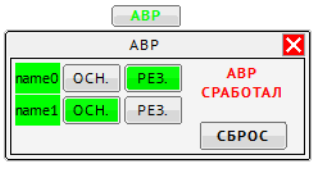
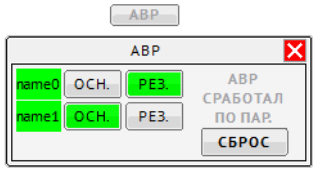
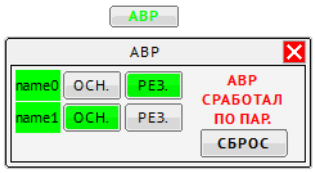
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
font	Тип TFont. настройки шрифта
bg_color	Тип TColor. Цвет фона
colorManual	Тип TColor. Цвет кнопки вывода в ремонт при состоянии в ремонте
colorPhysic	Тип TColor. Цвет кнопки вывода в ремонт при состоянии в работе
name	Тип STRING. Имя объекта
descr	Тип STRING. Описание объекта
units	Тип STRING. Единицы измерения
value_physic	Тип BOOL. Вход для реального физического значения

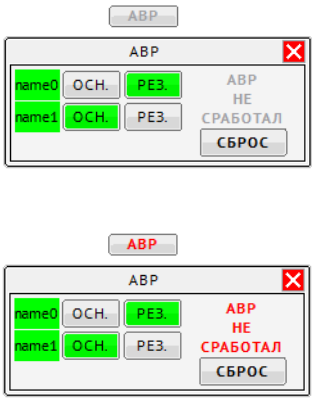
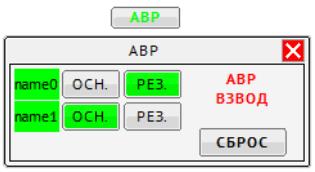
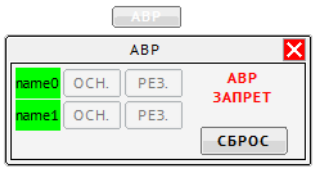
TTLDiscreteFaultChannel	
value_manual	Тип BOOL. Вход для отображения ручного значения параметра (используется для обратной связи на разных кадрах)
button	Тип BOOL. Состояние кнопки вывода в ремонт (используется для обратной связи на разных кадрах)
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
Выходные переменные	
buttonOut	Тип BOOL. Состояние кнопки вывода в ремонт
value_o_manual	Тип BOOL. Ручное значение параметра, полученное по кнопке ручного задания значения параметра

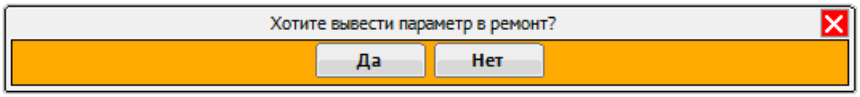
TTLFailover	
Объект АВР (автомат включения резерва). Данный объект может работать с 2 - 5 механизмами. Входные и выходные переменные будут описаны для 2ух механизмов (остальные аналогичны)	
1. Отображение на мнемосхеме  2. Вторичный видеокадр с управляющими кнопками (механизмов может быть от 2 до 5, количестово настраивается в свойстве объекта **Input count** )	Отображение содержимого объекта

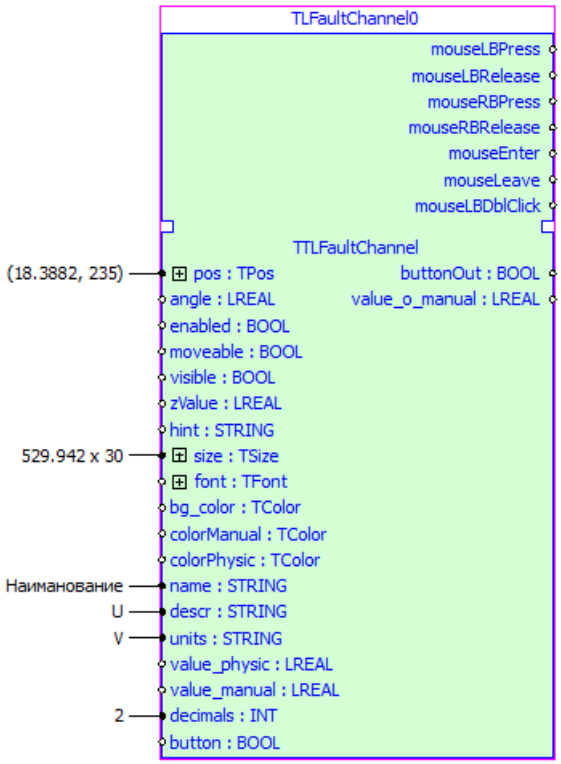
TTLFailover	
	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
bg_color	Тип TColor. Цвет заднего фона на вторичном видеокадре
state	Тип INT. Возможные состояния объекта (описание смотрите ниже)
reset	Тип BOOL. Входное значение (из алгоритма) на сброс
name_0	Тип STRING. Заголовок механизма 0

TTLFailover	
ready_0	Тип BOOL. Механизм 0 готов. TRUE - наименование механизма на вторичном видеокадре отображается зелёным цветом 
active_0	Тип BOOL. Механизм 0 в работе. TRUE - состояние ОСН. отображается зелёным цветом 
reserved_0	Тип BOOL. Механизм 0 в резерве. TRUE - состояние РЕЗ. отображается зелёным цветом 
name_1	Тип STRING. Заголовок механизма 1
ready_1	Тип BOOL. Механизм 1 готов. TRUE - наименование механизма на вторичном видеокадре отображается зелёным цветом 
active_1	Тип BOOL. Механизм 1 в работе. TRUE - состояние ОСН. отображается зелёным цветом 
reserved_1	Тип BOOL. Механизм 1 в резерве. TRUE - состояние РЕЗ. отображается зелёным цветом 
Выходные переменные	
o_reset	Тип BOOL. Выходное значение на сброс (по нажатию на кнопку  на вторичном видеокадре)
o_active_0	Тип BOOL. Выходной сигнал об активности механизма 0 (может подаваться с кнопки ОСН. на вторичном видеокадре)
o_reserved_0	Тип BOOL. Выходной сигнал, сообщающий, что механизм 0 в резерве (может подаваться с кнопки РЕЗ. на вторичном видеокадре)
o_active_1	Тип BOOL. Выходной сигнал об активности механизма 1 (может подаваться с кнопки ОСН. на вторичном видеокадре)
o_reserved_1	Тип BOOL. Выходной сигнал, сообщающий, что механизм 1 в резерве (может подаваться с кнопки РЕЗ. на вторичном видеокадре)
Возможные состояния	

TTLFailover			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
инициализация		Изображение (надпись АВР) на объекте отображается серым цветом. Состояния ОСН./РЕЗ. на вторичном видеокадре отображаются серым цветом	0
АВР готов		Изображение (надпись АВР) на объекте отображается зелёным цветом. На вторичном видеокадре красным цветом отображается надпись АВР ГОТОВ	1
АВР не готов		Изображение (надпись АВР) на объекте отображается жёлтым цветом. На вторичном видеокадре жёлтым цветом отображается надпись АВР НЕ ГОТОВ	2
АВР сработал	 	Изображение (надпись АВР) на объекте мигает серым/зелёным цветом. На вторичном видеокадре мигает серым/красным цветом надпись АВР СРАБОТАЛ	3
АВР сработал по параметру	 	Изображение (надпись АВР) на объекте мигает серым/зелёным цветом. На вторичном видеокадре мигает серым/красным цветом надпись АВР СРАБОТАЛ ПО ПАР.	4

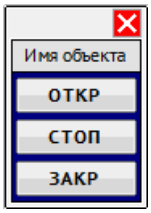
TTLFailover			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
АВР не сработал		Изображение (надпись АВР) на объекте мигает серым/красным цветом. На вторичном видеокадре мигает серым/красным цветом надпись АВР НЕ СРАБОТАЛ	5
АВР взведён		Изображение (надпись АВР) на объекте горит зелёным цветом. На вторичном видеокадре красным цветом горит надпись АВР ВЗВОД	6
АВР запрет		Изображение (надпись АВР) на объекте горит белым цветом. На вторичном видеокадре красным цветом горит надпись АВР ЗАПРЕТ. Кнопки ОСН. и РЕЗ. заблокированы	7

TTLFaultChannel
Вывод в ремонт аналогового канала
Отображение на мнемосхеме: 1. Объект  2. Вторичный видеокадр с управляющими кнопками 
Отображение содержимого объекта

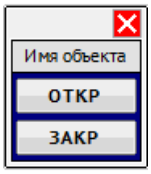
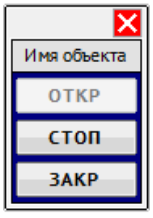
TTLFaultChannel


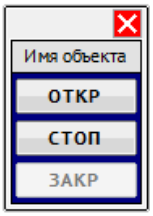
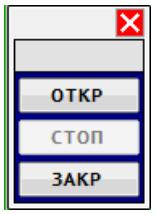
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
font	Тип TFont.
bg_color	Тип TColor. Цвет заднего фона объекта
colorManual	Тип TColor.
colorPhysic	Тип TColor.
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
descr	Тип STRING.
units	Тип STRING.
value_physic	Тип LREAL.

TTLFaultChannel	
value_manual	Тип LREAL.
decimals	Тип INT.
button	Тип BOOL.
Выходные события	
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
Выходные переменные	
buttonOut	Тип BOOL.
value_o_manual	Тип LREAL.
Описание	

TTLGateValve	
Объект задвижка без датчика положения	
Отображение на мнемосхеме: 1. Объект  2. Вторичный видеокадр с управляющими кнопками 	Отображение содержимого объекта

TTLGateValve	
	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
popup_wnd_title	Тип STRING. Заголовок вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_bg	Тип TColor. Цвет заднего фона объекта
color_object	Тип TColor. Цвет контура объекта
color_on	Тип TColor. Цвет объекта во включенном состоянии (соответствует state = 1)

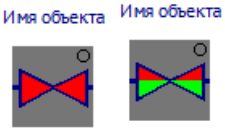
TTLGateValve	
color_off	Тип TColor. Цвет объекта в выключенном состоянии (соответствует state = 2)
color_fault	Тип TColor. Не используется для данного объекта
color_undefined	Тип TColor. Цвет объекта, отображающий неопределенное состояние (соответствует state = 0)
color_disassembled	Тип TColor. Цвет объекта, отображающий состояние - схема разобрана (соответствует state = 9)
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
name_angle	Тип LREAL. Угол поворота заголовка объекта (в градусах)
name_visible	Тип BOOL. Признак видимости заголовка объекта
state	Тип INT. Возможные состояния объекта (смотри описание возможных состояний ниже)
i_ackn	Тип BOOL. Снятие неисправности
flange_width	Тип INT. Ширина фланца
orientation	Тип INT. Поворот объекта (возможные значения 0, 1, 2 и 3 соответствуют повороту объекта на 0, 90, 180 и 270 градусов)
allow_stop	Тип BOOL. Разрешение останова. Если равно TRUE, то на видеокадре с управляющими кнопками есть кнопка СТОП, если равно FALSE, то данной кнопки нет. 
open_blocked	Тип BOOL. Запрет открытия. Если равен TRUE, то кнопка ОТКР на вторичном видеокадре станет неактивной 

TTLGateValve	
close_blocked	Тип BOOL. Запрет закрытия. Если равен TRUE, то кнопка ЗАКР на вторичном видеокадре станет неактивной 
stop_blocked	Тип BOOL. Запрет останова (кнопка СТОП на вторичном видеокадре с управляющими кнопками будет неактивной) 
Выходные переменные	
o_ackn	Тип BOOL. Выходная команда снятия неисправности
open	Тип BOOL. Выходная команда на открытие (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)
close	Тип BOOL. Выходная команда на закрытие (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)
stop	Тип BOOL. Выходная команда на останов (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)

Описание

Возможные состояния задвижки

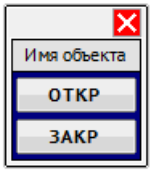
Наименование состояния	Отображение состояния на экране АРМа	Описание отображения	Значение сигнала состояния
инициализация		Изображение мигает (прозрачное / цвет, который настроен во входном параметре color_undefined)	0

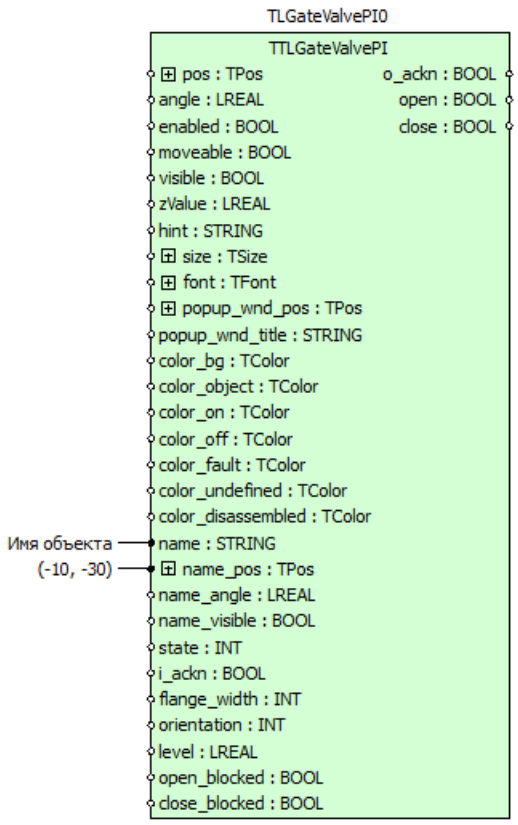
TTLGateValve			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
задвижка открыта		Изображение горит ровным цветом, которое настроено во входном параметре color_on	1
задвижка закрыта		Изображение горит ровным цветом, которое настроено во входном параметре color_off	2
задвижка открывается		Левая (верхняя) половинка мигает цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	3
задвижка закрывается		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) мигает цветом, который настроен во входном параметре color_on	4
задвижка не сошла с концевика по команде закрытие		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_on. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_off	5

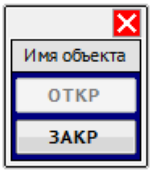
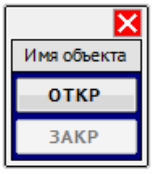
TTLGateValve			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
- задвижка не дошла до концевика по команде открытие - задвижка не дошла до концевика по команде закрытие	Имя объекта 	Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_on. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_off	6
- задвижка самоход - открыта - задвижка самоход - закрыта - задвижка самоход - середина	Имя объекта 	Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_on. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_off	7
задвижка в промежуточном состоянии	Имя объекта 	Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	8
схема разобрана	Имя объекта 	Изображение горит ровным цветом, которое настроено во входном параметре color_disassembled	9
включение двух противонаправленных автоматик	Имя объекта 	Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая	12

TTLGateValve			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
		(нижняя) горит ровным цветом, который настроен во входном параметре color_on	
включение двух противонаправленных блокировок		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	13
не включился пускатель		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	18
не отключился пускатель		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	19
самовключение пускателя		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным	20

TTLGateValve			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
		цветом, который настроен во входном параметре color_on	
самоотключение пускателя		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	21
неисправность концевых выключателей		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	22

TTLGateValvePI	
Объект - задвижка с указателем положения и управлением открытием/закрытием путём нажатия и удерживания соответствующих кнопок вторичного видеокадра	
Отображение на мнемосхеме 1. Объект  2. Вторичный видеокадр с управляющими кнопками 	Отображение содержимого объекта

TTLGateValvePI	
	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
popup_wnd_title	Тип STRING. Заголовок вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_bg	Тип TColor. Цвет заднего фона объекта
color_object	Тип TColor. Цвет контура объекта
color_on	Тип TColor. Цвет объекта во включенном состоянии (соответствует state = 1)

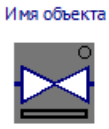
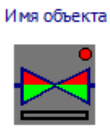
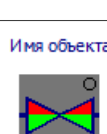
TTLGateValvePI	
color_off	Тип TColor. Цвет объекта в выключенном состоянии (соответствует state = 2)
color_fault	Тип TColor. Не используется для данного объекта
color_undefined	Тип TColor. Цвет объекта, отображающий неопределённое состояние (соответствует state = 0)
color_disassembled	Тип TColor. Цвет объекта, отображающий состояние - схема разобрана (соответствует state = 9)
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
name_angle	Тип LREAL. Угол поворота заголовка объекта (в градусах)
name_visible	Тип BOOL. Признак видимости заголовка объекта
state	Тип INT. Возможные состояния объекта
i_ackn	Тип BOOL. Снятие неисправности
flange_width	Тип INT. Ширина фланца
orientation	Тип INT. Поворот объекта (возможные значения 0, 1, 2 и 3 соответствуют повороту объекта на 0, 90, 180 и 270 градусов)
level	Тип LREAL. Значение от 0 до 100. Характеризует положение задвижки
open_blocked	Тип BOOL. Запрет открытия. Если равен TRUE, то кнопка ОТКР на вторичном видеокадре станет неактивной 
close_blocked	Тип BOOL. Запрет закрытия. Если равен TRUE, то кнопка ЗАКР на вторичном видеокадре станет неактивной 
Выходные переменные	
o_ackn	Тип BOOL. Выходная команда снятия неисправности

TTLGateValvePI	
open	Тип BOOL. Выходная команда на открытие (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)
close	Тип BOOL. Выходная команда на закрытие (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)

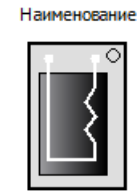
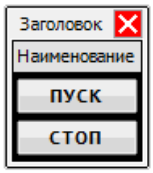
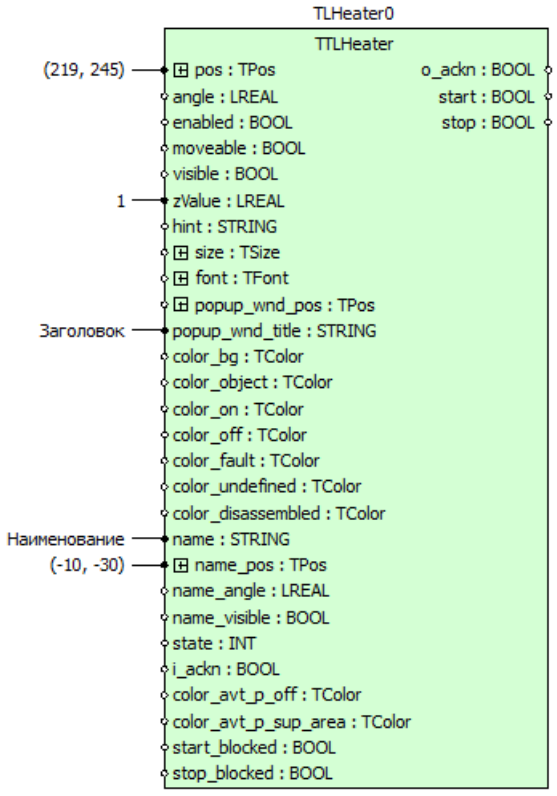
Возможные состояния задвижки с указателем положения

Наименование состояния	Отображение состояния на экране АРМа	Описание отображения	Значение сигнала состояния
инициализация		Изображение мигает (прозрачное / цвет, который настроен во входном параметре color_undefined)	0
задвижка открыта		Изображение горит ровным цветом, которое настроено во входном параметре color_on	1
задвижка закрыта		Изображение горит ровным цветом, которое настроено во входном параметре color_off	2
задвижка открывается		Левая (верхняя) половинка мигает цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	3
задвижка закрывается		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) мигает цветом, который	4

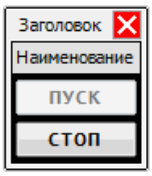
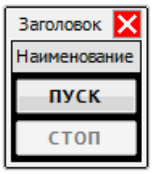
TTLGateValvePI			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
		настроен во входном параметре color_on	
- задвижка не сошла с концевика по команде на открытие - задвижка не сошла с концевика по команде на закрытие		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_on. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_off	5
- задвижка не дошла до концевика по команде на открытие - задвижка не дошла до концевика по команде на закрытие		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_on. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_off	6
- задвижка самоход - открыта - задвижка самоход - закрыта - задвижка самоход - середина		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_on. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_off	7
задвижка в промежуточном состоянии		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который	8

TTLGateValvePI			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
		настроен во входном параметре color_on	
схема разобрана		Изображение горит ровным цветом, которое настроено во входном параметре color_disassembled	9
включение двух противоположно направленных автоматик		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	12
включение двух противоположно направленных блокировок		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	13
неисправность концевых выключателей		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	14
неисправность аналоговых входов		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая	15

TTLGateValvePI			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
		(нижняя) горит ровным цветом, который настроен во входном параметре color_on	
включен дистанционный режим		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on	16

TTLHeater	
Объект нагреватель	
Отображение на мнемосхеме: 1. Объект  2. Вторичный видеокадр с управляющими кнопками 	Отображение содержимого объекта 
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах

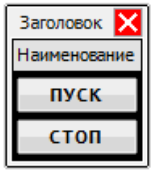
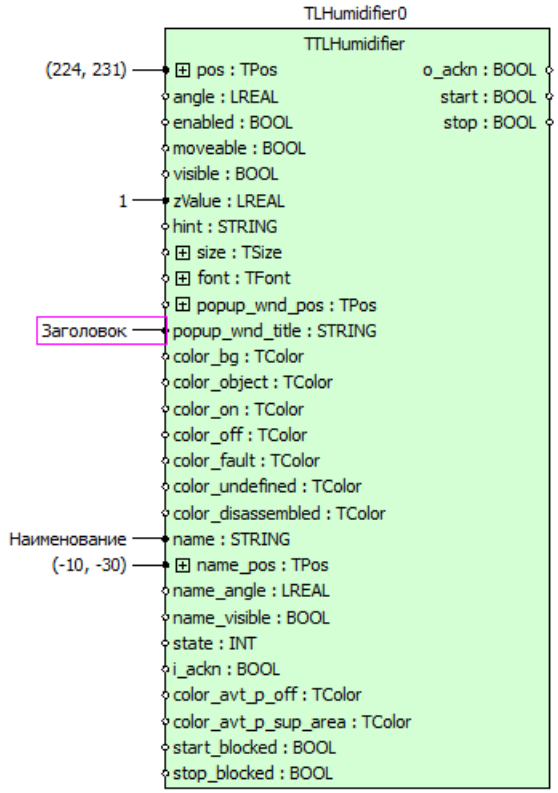
TTLHeater	
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
popup_wnd_title	Тип STRING. Заголовок вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_bg	Тип TColor. Цвет заднего фона на объекте
color_object	Тип TColor. Цвет фона объекта (нагревателя)
color_on	Тип TColor. Цвет объекта (линия накаливания) во включённом состоянии (state = 1)
color_off	Тип TColor. Цвет объекта (линия накаливания) в выключённом состоянии (state = 2)
color_fault	Тип TColor. Цвет объекта (линия накаливания), отображающий состояние неисправности
color_undefined	Тип TColor. Цвет объекта (линия накаливания), отображающий неопределённое состояние (state = 0)
color_disassembled	Тип TColor. Цвет объекта (линия накаливания), отображающий состояние - схема разобрана (state = 9)
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
name_angle	Тип LREAL. Угол поворота заголовка объекта (в градусах)
name_visible	Тип BOOL. Признак видимости заголовка объекта
state	Тип INT. Возможные состояния объекта (описание смотрите ниже)
i_ackn	Тип BOOL. Снятие неисправности

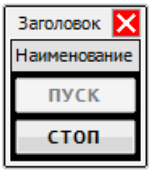
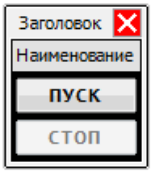
TTLHeater	
color_avt_p_off	Тип TColor.
color_avt_p_sup_area	Тип TColor.
start_blocked	Тип BOOL. Запрет запуска. Если равен TRUE, то кнопка ПУСК на вторичном видеокадре станет неактивной 
stop_blocked	Тип BOOL. Запрет останова. Если равен TRUE, то кнопка СТОП на вторичном видеокадре станет неактивной 
Выходные переменные	
o_ackn	Тип BOOL. Выходная команда снятия неисправности
start	Тип BOOL. Выходная команда на включение (с кнопки управления ПУСК, находящейся на вторичном видеокадре)
stop	Тип BOOL. Выходная команда на отключение (с кнопки управления СТОП, находящейся на вторичном видеокадре)

Возможные состояния нагревателя

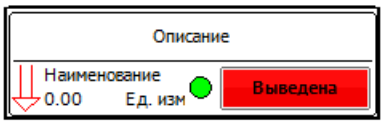
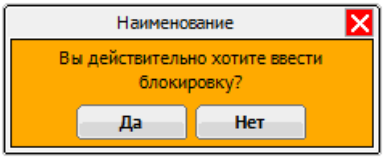
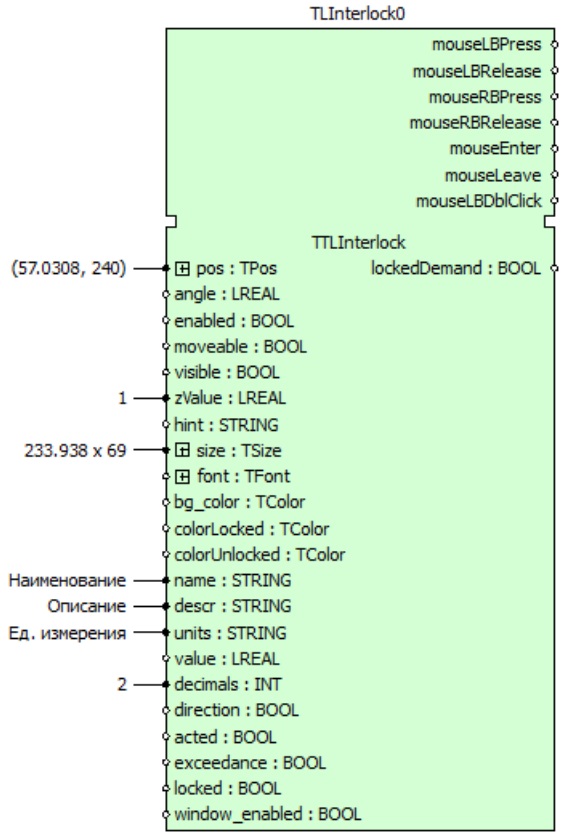
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**

TTLHumidifier	
Объект увлажнитель	
Отображение на мнемосхеме: 1. Объект Наименование  2. Вторичный видеокадр с управляющими кнопками	Отображение содержимого объекта

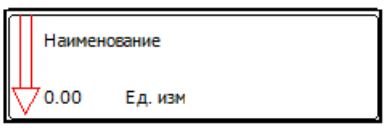
TTLHumidifier	
	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
popup_wnd_title	Тип STRING. Заголовок вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_bg	Тип TColor. Цвет заднего фона на объекте
color_object	Тип TColor. Цвет объекта и заднего фона на вторичном видеокадре
color_on	Тип TColor. Цвет объекта во включённом состоянии (state = 1)

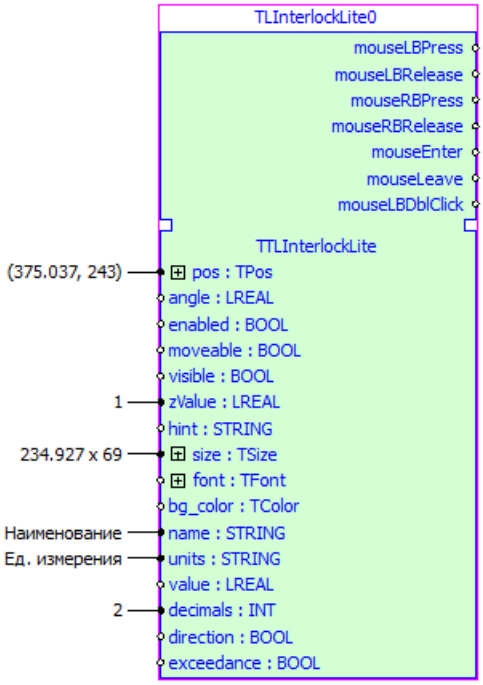
TTLHumidifier	
color_off	Тип TColor. Цвет объекта в выключенном состоянии (state = 2)
color_fault	Тип TColor. Цвет объекта, отображающий состояние неисправности
color_undefined	Тип TColor. Цвет объекта, отображающий неопределённое состояние (state = 0)
color_disassembled	Тип TColor. Цвет объекта, отображающий состояние - схема разобрана (state = 9)
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
name_angle	Тип LREAL. Угол поворота заголовка объекта (в градусах)
name_visible	Тип BOOL. Признак видимости заголовка объекта
state	Тип INT. Возможные состояния объекта (описание смотрите ниже)
i_ackn	Тип BOOL. Снятие неисправности
color_avt_p_off	Тип TColor.
color_avt_p_sup_area	Тип TColor.
start_blocked	Тип BOOL. Запрет запуска. Если равен TRUE, то кнопка ПУСК на вторичном видеокадре станет неактивной 
stop_blocked	Тип BOOL. Запрет останова. Если равен TRUE, то кнопка СТОП на вторичном видеокадре станет неактивной 
Выходные переменные	
o_ackn	Тип BOOL. Выходная команда снятия неисправности
start	Тип BOOL. Выходная команда на включение (с кнопки управления ПУСК, находящейся на вторичном видеокадре)
stop	Тип BOOL. Выходная команда на отключение (с кнопки управления СТОП, находящейся на вторичном видеокадре)

TTLHumidifier			
Возможные состояния увлажнителя			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**

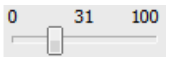
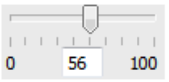
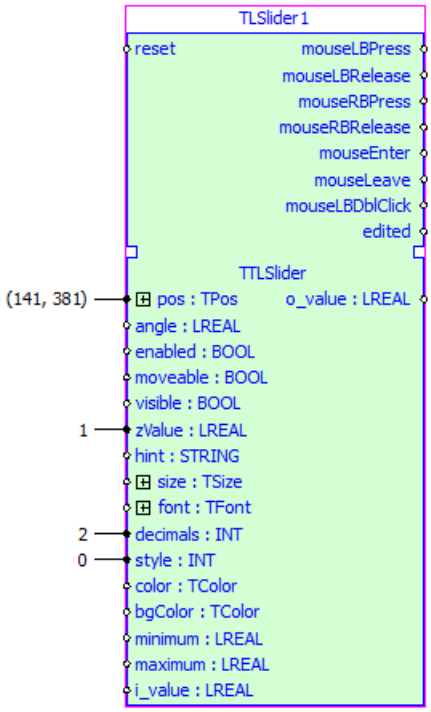
TTLInterlock	
Блокировка аналогового параметра	
Отображение на мнемосхеме: 1. Объект  2. Вторичный видеокадр  	Отображение содержимого объекта 
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта

TTLInterlock	
font	Тип TFont. Шрифт
bg_color	Тип TColor. Цвет заднего фона
colorLocked	Тип TColor. Цвет в состоянии блокировки
colorUnlocked	Тип TColor. Цвет в состоянии разблокирован
name	Тип STRING. Заголовок
descr	Тип STRING.
units	Тип STRING.
value	Тип LREAL. Значение
decimals	Тип INT. Количество знаков после запятой у значения
direction	Тип BOOL.
acted	Тип BOOL.
exceedance	Тип BOOL.
locked	Тип BOOL.
window_enabled	Тип BOOL.
Выходные события	
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
Выходные переменные	
lockedDemand	Тип BOOL.

TTLInterlockLite	
Блокировка аналогового параметра (упрощенная версия)	
Отображение на мнемосхеме: 1. Объект  	Отображение содержимого объекта

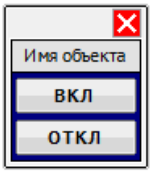
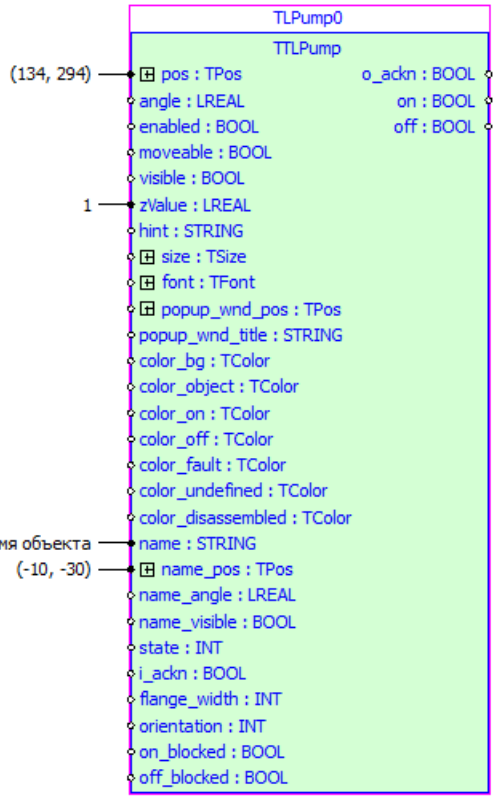
TTLInterlockLite	
	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
font	Тип TFont. Шрифт
bg_color	Тип TColor. Цвет заднего фона
name	Тип STRING. Заголовок
units	Тип STRING.
value	Тип LREAL. Значение
decimals	Тип INT. Количество знаков после запятой у значения
direction	Тип BOOL.
exceedance	Тип BOOL.
Выходные события	
mouseLBPress	Нажатие левой кнопки «мыши» на объекте

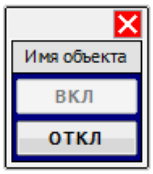
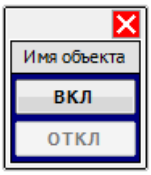
TTLInterlockLite	
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте

TTLSlider	
Объект ползунок	
Отображение на мнемосхеме: 1. Объект при style = 0  2. Объект при style = 1  3. Объект при style = 2  4. Объект при style = 3  5. Объект при style = 4  	Отображение содержимого объекта 
Входные события	
reset	Входное событие на сброс показаний
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый

TTLSlider	
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
font	Тип TFont. Шрифт
decimals	Тип INT. Количество знаков после запятой
style	Тип INT. Стиль отображения объекта. Возможные значения: 0 - ползунок без цифр; 1 - ползунок с цифровыми значениями; 2 - ползунок с метками снизу; 3 - то же, что и 2, но с отрисовкой цифр под метками; 4 - то же, что и 3, но в качестве отображения текущего значения используется редактор, в котором можно вводить свои нужные значения вручную
color	Тип TColor. Цвет контура объекта
bgColor	Тип TColor. Цвет заднего фона объекта
minimum	Тип LREAL. Минимальное значение
maximum	Тип LREAL. Максимальное значение
i_value	Тип LREAL. Значение на входе
Выходные события	
mouseLBPress	Нажатие левой кнопки «мыши» на объекте
mouseLBRelease	Отпускание левой кнопки «мыши» на объекте
mouseRBPress	Нажатие правой кнопки «мыши» на объекте
mouseRBRelease	Отпускание правой кнопки «мыши» на объекте
mouseEnter	Вход указателя «мыши» в пределы объекта
mouseLeave	Выход указателя «мыши» за пределы объекта
mouseLBDblClick	Двойной щелчок левой кнопки «мыши» на объекте
edited	
Выходные переменные	
o_value	Тип LREAL. Значение на выходе

TTLPump	
Объект помпа	
Отображение на мнемосхеме:	Отображение содержимого объекта

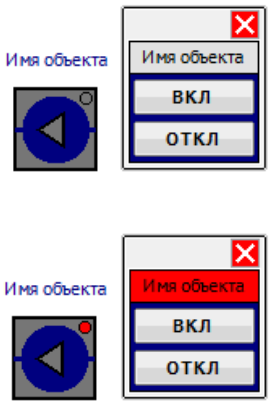
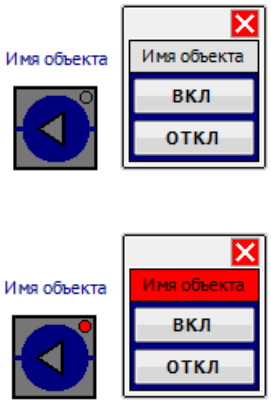
TTLPump	
1. Объект Имя объекта  2. Вторичный видеокадр с управляющими кнопками 	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
popup_wnd_title	Тип STRING. Заголовок вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_bg	Тип TColor. Цвет заднего фона объекта
color_object	Тип TColor. Цвет контура объекта
color_on	Тип TColor. Цвет объекта во включённом состоянии (соответствует state = 1)

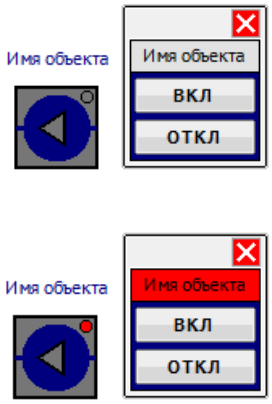
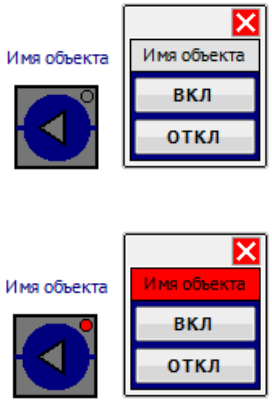
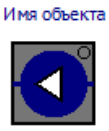
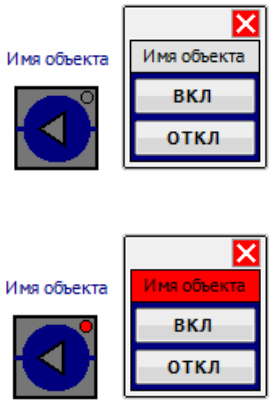
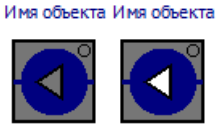
TTLPump	
color_off	Тип TColor. Цвет объекта во включённом состоянии (соответствует state = 2)
color_fault	Тип TColor. Не используется для данного объекта
color_undefined	Тип TColor. Цвет объекта, отображающий неопределённое состояние (соответствует state = 0)
color_disassembled	Тип TColor. Цвет объекта, отображающий состояние - схема разобрана (соответствует state = 9)
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
name_angle	Тип LREAL. Угол поворота заголовка объекта (в градусах)
name_visible	Тип BOOL. Признак видимости заголовка объекта
state	Тип INT. Возможные состояния объекта
i_ackn	Тип BOOL. Снятие неисправности
flange_width	Тип INT. Ширина фланца
orientation	Тип INT. Поворот объекта (возможные значения 0, 1, 2 и 3 соответствуют повороту объекта на 0, 90, 180 и 270 градусов)
on_blocked	Тип BOOL. Запрет включения. Если равен TRUE, то кнопка ВКЛ на вторичном видеокадре станет неактивной 
off_blocked	Тип BOOL. Запрет отключения. Если равен TRUE, то кнопка ОТКЛ на вторичном видеокадре станет неактивной 
Выходные переменные	
o_ackn	Тип BOOL. Выходная команда снятия неисправности
on	Тип BOOL. Выходная команда на включение (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)

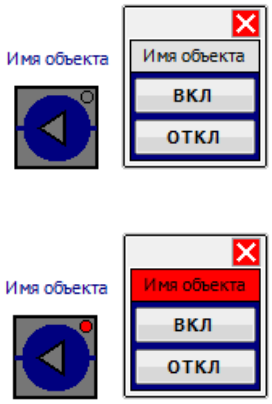
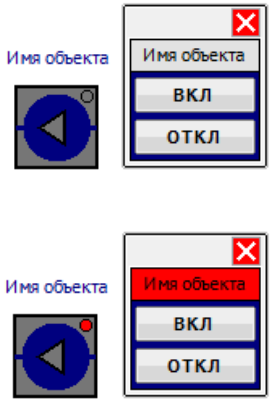
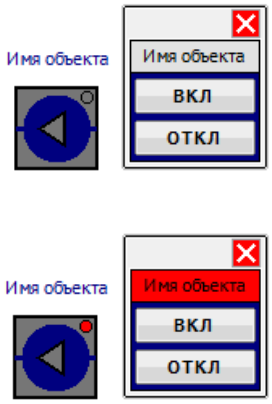
TTLPump	
off	Тип BOOL. Выходная команда на отключение (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)

Описание

Возможные состояния помпы

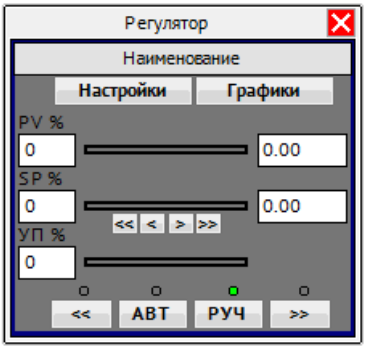
Наименование состояния	Отображение состояния на экране АРМа	Описание отображения	Значение сигнала состояния
инициализация		Изображение мигает (прозрачное/цвет, который настроен во входном параметре color_undefined)	0
двигатель включен		Изображение горит ровным цветом, которое настроено во входном параметре color_on	1
двигатель отключен		Изображение горит ровным цветом, которое настроено во входном параметре color_off	2
двигатель не включился		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	5
двигатель не отключился		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	6

TTLPump			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
самовключение		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	7
самоотключение		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	8
схема разобрана		Изображение горит ровным цветом, которое настроено во входном параметре color_disassembled	9
схема разобрана		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	12
два противоположных состояния		Изображение мигает (прозрачное/цвет, который настроен во входном параметре color_undefined)	14

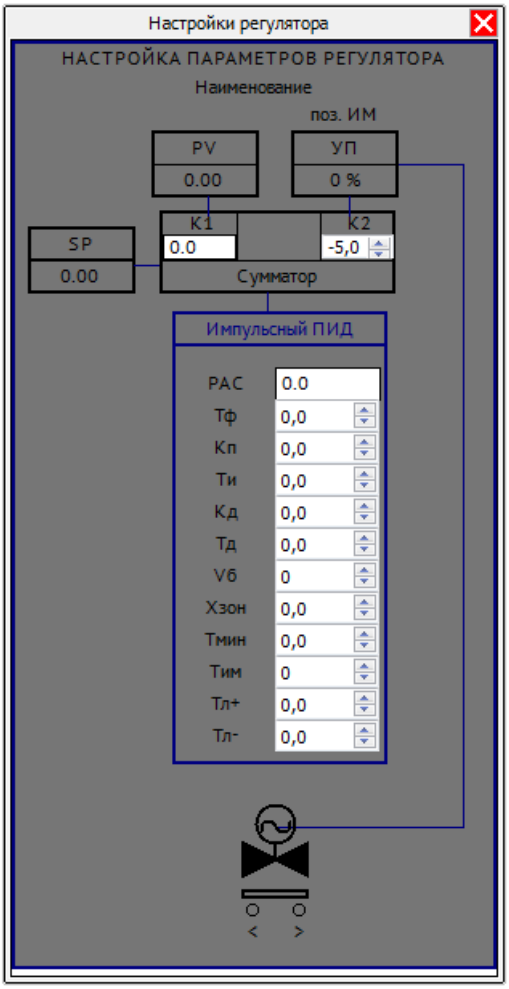
TTLPump			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
включение двух противонаправленных автоматик		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	15
включение двух противонаправленных блокировок		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	16
неисправность физических каналов		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	22

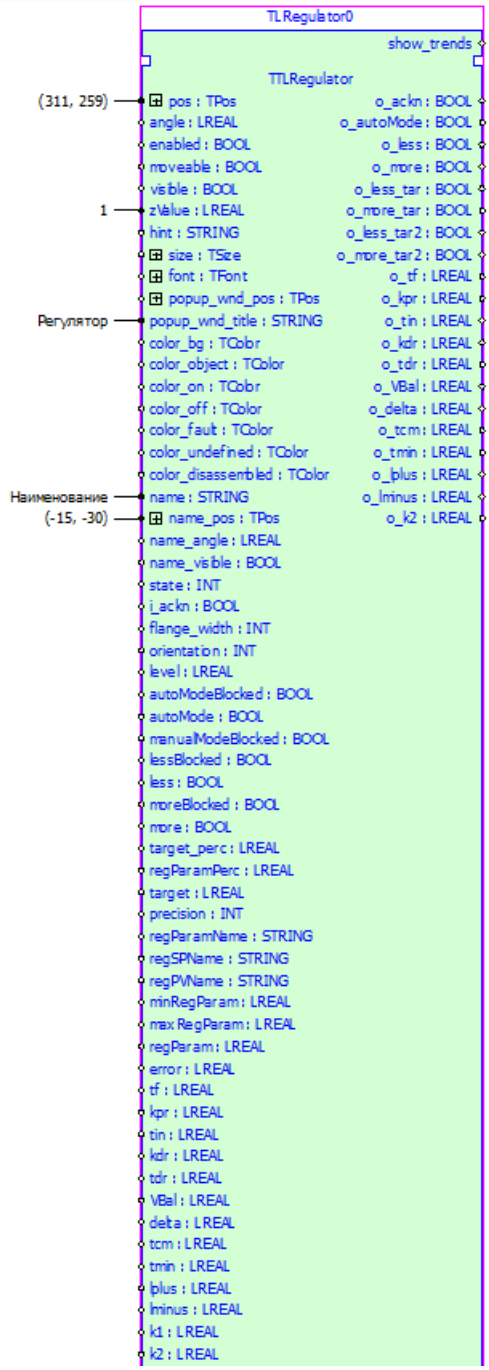
TTLRegulator	
Объект регулятор	
Отображение на мнемосхеме: 1. Объект  2. Вторичный видеокадр с управляющими кнопками	Отображение содержимого объекта

TTLRegulator



3. По нажатию на кнопку Настройки на вторичном видеокадре откроется окно Настройки регулятора





Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта

TTLRegulator	
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
popup_wnd_title	Тип STRING. Заголовок вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_bg	Тип TColor. Цвет заднего фона объекта
color_object	Тип TColor. Цвет контура объекта
color_on	Тип TColor. Цвет объекта во включённом состоянии
color_off	Тип TColor. Цвет объекта в выключенном состоянии
color_fault	Тип TColor. Не используется для данного объекта
color_undefined	Тип TColor. Цвет объекта, отображающий неопределённое состояние (соответствует state = 0)
color_disassembled	Тип TColor. Цвет объекта, отображающий состояние - схема разобрана (соответствует state = 9)
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
name_angle	Тип LREAL. Угол поворота заголовка объекта (в градусах)
name_visible	Тип BOOL. Признак видимости заголовка объекта
state	Тип INT. Возможные состояния объекта (описание смотрите ниже)
i_ackn	Тип BOOL. Снятие неисправности
flange_width	Тип INT. Ширина фланца
orientation	Тип INT. Поворот объекта (возможные значения 0, 1, 2 и 3 соответствуют повороту объекта на 0, 90, 180 и 270 градусов)
level	Тип LREAL.
autoModeBlocked	Тип BOOL. Запрет включения автоматического режима. Кнопка АВТ на вторичном видеокадре станет неактивной.

TTLRegulator	
autoMode	Тип BOOL. Включение автоматического режима.
manualModeBlocked	Тип BOOL. Запрет включения ручного режима. Кнопка РУЧ на вторичном видеокадре станет неактивной.
lessBlocked	Тип BOOL. Запрет
less	Тип BOOL.
moreBlocked	Тип BOOL.
more	Тип BOOL.
target_perc	Тип LREAL.
regParamPerc	Тип LREAL.
target	Тип LREAL.
precision	Тип INT.
regParamName	Тип STRING.
regSPName	Тип STRING.
regPVName	Тип STRING.
miniRegParam	Тип LREAL.
maxRegParam	Тип LREAL.
RegParam	Тип LREAL.
error	Тип LREAL.
tf	Тип LREAL. Постоянная времени фильтра
kpr	Тип LREAL. Коэффициент пропорциональности
tin	Тип LREAL. Время интегрирования
kdr	Тип LREAL.
tdr	Тип LREAL. Постоянная времени дифференцирования
VBal	Тип LREAL.
delta	Тип LREAL.
tcm	Тип LREAL.
tmin	Тип LREAL.
lplus	Тип LREAL.
lminus	Тип LREAL.
k1	Тип LREAL.
k2	Тип LREAL.
Выходные события	
show_trends	Выходное событие, которое вырабатывается по нажатию на кнопку Графики, находящуюся на вторичном

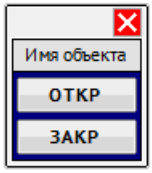
TTLRegulator	
	видеокадре (применяется для подачи на входное событие show функционального блока ARCHIVE_TREND_WINDOW)
Выходные переменные	
o_ackn	Тип BOOL.
o_autoMode	Тип BOOL.
o_less	Тип BOOL.
o_more	Тип BOOL.
o_less_tar	Тип BOOL.
o_more_tar	Тип BOOL.
o_less_tar2	Тип BOOL.
o_more_tar2	Тип BOOL.
o_tf	Тип LREAL.
o_kpr	Тип LREAL.
o_tin	Тип LREAL.
o_kdr	Тип LREAL.
o_tdr	Тип LREAL.
o_VBal	Тип LREAL.
o_delta	Тип LREAL.
o_tcm	Тип LREAL.
o_tmin	Тип LREAL.
o_lplus	Тип LREAL.
o_k2	Тип LREAL.

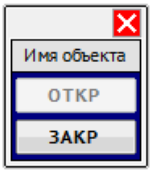
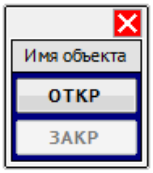
Возможные состояния увлажнителя

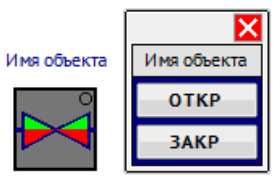
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
инициализация механизма			0
регулятор открыт			1
регулятор закрыт			2
регулятор открывается			3
регулятор закрывается			4
- регулятор не сошел с концевика по команде на открытие - регулятор не сошел с концевика по команде на закрытие			5

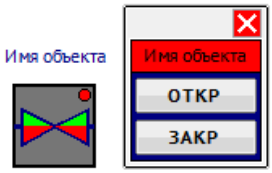
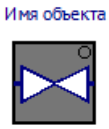
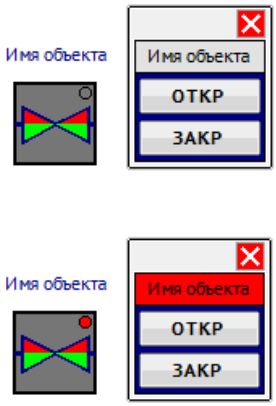
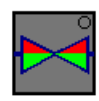
TTLRegulator			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
- задвижка не дошла до концевика по команде на открытие - задвижка не дошла до концевика по команде на закрытие			6
- задвижка самоход - открыта - задвижка самоход - закрыта - задвижка самоход - середина			7
задвижка в промежуточном состоянии			8
задвижка - схма разобрана			9
задвижка - включение двух противонаправленных автоматик			12
задвижка - включение двух противонаправленных блокировок			13
задвижка - неисправность концевых выключателей			14

TTLValve	
Объект клапан	
Отображение на мнемосхеме: 1. Объект  2. Вторичный видеокадр с управляющими кнопками	Отображение содержимого объекта

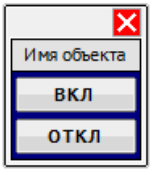
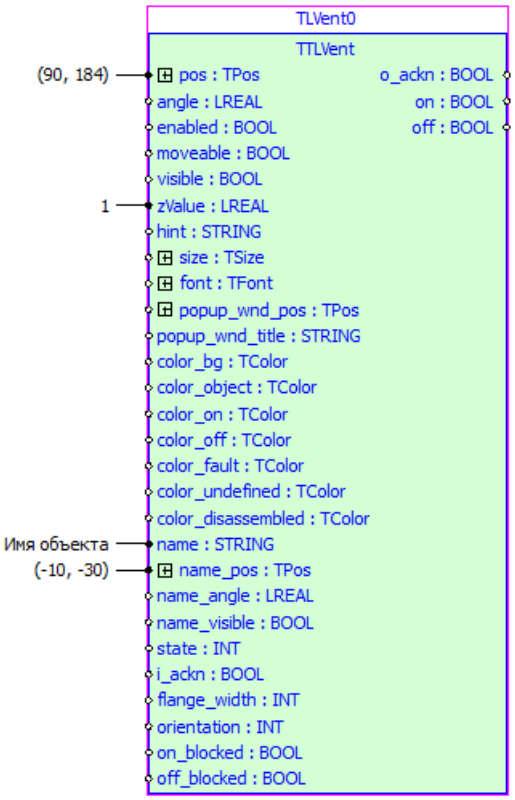
TTLValve	
	
Входные переменные	
pos	Тип TPos. Положение объекта
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
popup_wnd_title	Тип STRING. Заголовок вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_bg	Тип TColor. Цвет заднего фона объекта
color_object	Тип TColor. Цвет контура объекта
color_on	Тип TColor. Цвет объекта во включенном состоянии (соответствует state = 1)

TTLValve	
color_off	Тип TColor. Цвет объекта в выключенном состоянии (соответствует state = 2)
color_fault	Тип TColor. Не используется для данного объекта
color_undefined	Тип TColor. Цвет объекта, отображающий неопределённое состояние (соответствует state = 0)
color_disassembled	Тип TColor. Цвет объекта, отображающий состояние - схема разобрана (соответствует state = 9)
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
name_angle	Тип LREAL. Угол поворота заголовка объекта (в градусах)
name_visible	Тип BOOL. Признак видимости заголовка объекта
state	Тип INT. Возможные состояния объекта (описание возможных состояний смотри ниже)
i_ackn	Тип BOOL. Снятие неисправности
flange_width	Тип INT. Ширина фланца
orientation	Тип INT. Поворот объекта. Возможные значения: 0, 1, 2, 3. Они соответствуют повороту на 90, 180, 270 и 360 градусов соответственно
open_blocked	Тип BOOL. Запрет открытия. Если равен TRUE, то кнопка ОТКР на вторичном видеокадре станет неактивной 
close_blocked	Тип BOOL. Запрет закрытия. Если равен TRUE, то кнопка ЗАКР на вторичном видеокадре станет неактивной 
Выходные переменные	
o_ackn	Тип BOOL. Выходная команда снятия неисправности
open	Тип BOOL. Выходная команда на открытие (с кнопок управления, расположенных

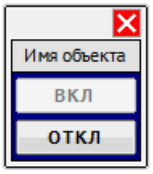
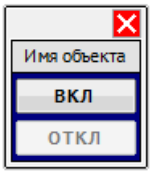
<table>
<tr><th colspan="2">TTLValve</th></tr>
<tr><td></td><td>на вторичном видеокадре с управляющими кнопками)</td></tr>
<tr><td>close</td><td>Тип BOOL. Выходная команда на закрытие (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)</td></tr>
<tr><th colspan="2">Описание</th></tr>
<tr><td colspan="2">Клапан может находиться в одном из следующих состояний:
<table>
<tr><th>Наименование состояния</th><th>Отображение состояния на экране АРМа</th><th>Описание отображения</th><th>Значение сигнала состояния</th></tr>
<tr><td>инициализация</td><td> 
</td><td>Изображение мигает (прозрачное/цвет, который настроен во входном параметре color_undefined)</td><td>0</td></tr>
<tr><td>клапан открыт</td><td> 
</td><td>Изображение горит ровным цветом, которое настроено во входном параметре color_on</td><td>1</td></tr>
<tr><td>клапан закрыт</td><td> 
</td><td>Изображение горит ровным цветом, которое настроено во входном параметре color_off</td><td>2</td></tr>
<tr><td>- клапан не открылся
- клапан не закрылся</td><td> 
</td><td>Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_on. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_off. На вторичном видеокадре красным цветом мигает заголовок объекта</td><td>6</td></tr>
<tr><td>- клапан самооткрытие
- клапан самозакрытие</td><td> 
</td><td>Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во</td><td>7</td></tr>
</table></td></tr>
</table>

TTLValve			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
		входном параметре color_on. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_off. На вторичном видеокадре красным цветом мигает заголовок объекта	
схема разобрана		Изображение горит ровным цветом, которое настроено во входном параметре color_disassembled	9
включение двух противоположных автоматик		Мигает красный кружок. Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on. На вторичном видеокадре красным цветом мигает заголовок объекта	12
включение двух противоположных блокировок		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on.	13
неисправность концевых выключателей		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая	14

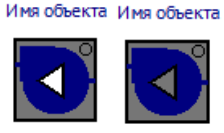
TTLValve			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
		(нижняя) горит ровным цветом, который настроен во входном параметре color_on.	
два концевых выключателя		Левая (верхняя) половинка горит ровным цветом, который настроен во входном параметре color_off. Правая (нижняя) горит ровным цветом, который настроен во входном параметре color_on.	15

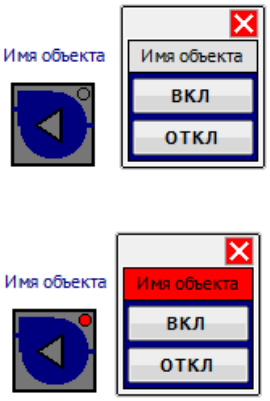
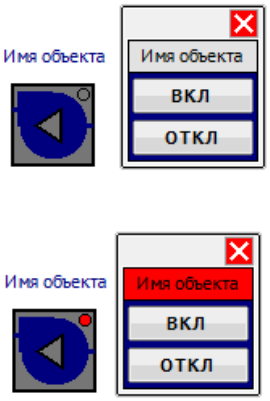
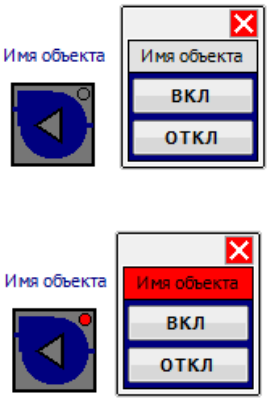
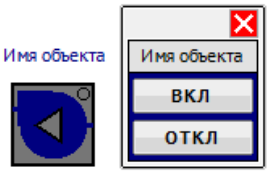
TTLVent	
Объект вентилятор	
Отображение на мнемосхеме: 1. Объект  2. Вторичный видеокадр с управляющими кнопками 	Отображение содержимого объекта 
Входные переменные	
pos	Тип TPos. Положение объекта

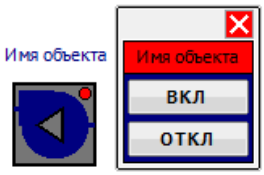
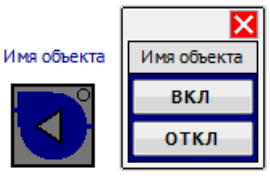
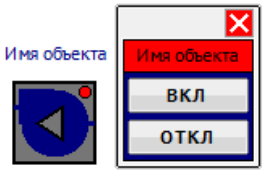
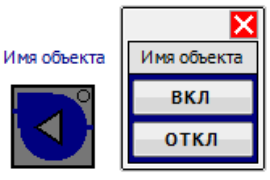
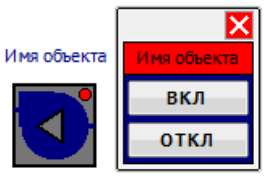
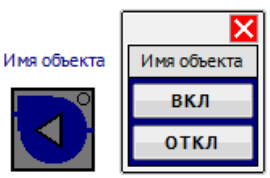
TTLVent	
angle	Тип LREAL. Угол поворота объекта в градусах
enabled	Тип BOOL. Задает разрешенность объекта
moveable	Тип BOOL. TRUE, если объект перемещаемый
visible	Тип BOOL. Признак видимости объекта
zValue	Тип LREAL. Z-координата объекта
hint	Тип STRING. Всплывающая подсказка объекта
size	Тип TSize. Размер объекта
popup_wnd_pos	Тип TPos. Начальная позиция вторичного видеокадра объекта (видеокадра с управляющими кнопками)
popup_wnd_title	Тип STRING. Заголовок вторичного видеокадра объекта (видеокадра с управляющими кнопками)
color_bg	Тип TColor. Цвет заднего фона объекта
color_object	Тип TColor. Цвет контура объекта
color_on	Тип TColor. Цвет объекта во включенном состоянии (соответствует state = 1)
color_off	Тип TColor. Цвет объекта в выключенном состоянии (соответствует state = 2)
color_fault	Тип TColor. Не используется для данного объекта
color_undefined	Тип TColor. Цвет объекта, отображающий неопределенное состояние (соответствует state = 0)
color_disassembled	Тип TColor. Цвет объекта, отображающий состояние - схема разобрана (соответствует state = 9)
name	Тип STRING. Заголовок объекта
name_pos	Тип TPos. Начальная позиция заголовка объекта
name_angle	Тип LREAL. Угол поворота заголовка объекта (в градусах)
name_visible	Тип BOOL. Признак видимости заголовка объекта
state	Тип INT. Возможные состояния объекта
i_ackn	Тип BOOL. Снятие неисправности
flange_width	Тип INT. Ширина фланца

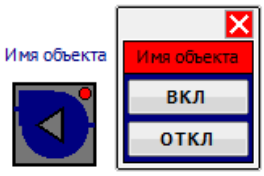
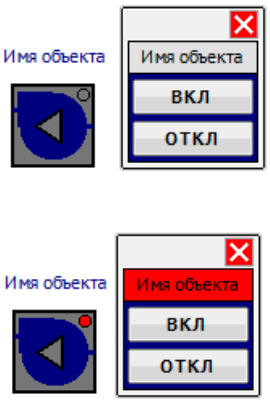
TTLVent	
orientation	Тип INT. Поворот объекта (возможные значения 0, 1, 2 и 3 соответствуют повороту объекта на 0, 90, 180 и 270 градусов)
on_blocked	Тип BOOL. Запрет включения. Если равен TRUE, то кнопка ВКЛ на вторичном видеокадре неактивна 
off_blocked	Тип BOOL. Запрет отключения. Если равен TRUE, то кнопка ОТКЛ на вторичном видеокадре неактивна 
Выходные переменные	
o_ackn	Тип BOOL. Выходная команда снятия неисправности
on	Тип BOOL. Выходная команда на включение (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)
off	Тип BOOL. Выходная команда на отключение (с кнопок управления, расположенных на вторичном видеокадре с управляющими кнопками)

Возможные состояния вентилятора

Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
инициализация		Изображение мигает (прозрачное/цвет, который настроен во входном параметре color_undefined)	0
двигатель включен		Изображение горит ровным цветом, которое настроено во входном параметре color_on	1

TTLVent			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
двигатель отключен		Изображение горит ровным цветом, которое настроено во входном параметре color_off	2
двигатель не включился		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	5
двигатель не отключился		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	6
самовключение		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	7
самоотключение		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	8

TTLVent			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
			
схема разобрана		Изображение горит ровным цветом, которое настроено во входном параметре color_disassembled	9
схема разобрана	 	Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	12
два противоположных состояния		Изображение мигает (прозрачное/цвет, который настроен во входном параметре color_undefined)	14
включение двух противонаправленных автоматик	 	Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	15
включение двух противонаправленных блокировок		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	16

TTLVent			
Наименование состояния	**Отображение состояния на экране АРМа**	**Описание отображения**	**Значение сигнала состояния**
			
неисправность физических каналов		Мигает красный кружок. На вторичном видеокадре красным цветом мигает заголовок объекта	22

2.1.3.6. Прочие типы функциональных блоков

2.1.3.6.1. Блоки диалоговых окон

MESSAGE_BOX	
Блок, выводящий небольшое диалоговое окно с текстом и кнопками «OK», «Отмена», «Да», «Нет»	
Входные события	
EXEC	При приходе сообщения показывается окно
Входные переменные	
ICON	Выводимая иконка (тип INT): 0 – нет иконки; 1 – иконка информации; 2 – иконка предупреждения; 3 – иконка критической ошибки; 4 – иконка запроса.
FONT	Тип font. Шрифт окна
TITLE	Тип STRING. Заголовок окна сообщения
MESSAGE	Тип STRING. Текст сообщения
BUTTONS	Тип UDINT. Кнопки окна сообщения: 1024 – кнопка «OK»;

MESSAGE_BOX	
	4194304 – кнопка «Отмена»; 16384 – кнопка «Да»; 65536 – кнопка «Нет»; результатирующий набор кнопок определяется путем сложения указанных кодов.
Выходные события	
E_OK	Возникает, когда нажата кнопка «ОК»
E_CANCEL	Возникает, когда нажата кнопка «Отмена» или клавиша «ESC»
E_YES	Возникает, когда нажата кнопка «Да»
E_NO	Возникает, когда нажата кнопка «Нет»
E_CLOSED	Возникает, когда окно запроса закрывается
Выходные переменные	
RESULT	Тип UDINT. Код нажатой кнопки

FILE_DIALOG	
Вызов диалога выбора файла для открытия либо сохранения	
Входные события	
EXEC	Вызов диалога
Выходные события	
OK	Пользователь нажал на кнопку "OK" в диалоге
CANCEL	Пользователь нажал на кнопку "Отмена" в диалоге
Входные переменные	
KIND	Вид диалога (тип INT). Значение 0 соответствует диалогу выбора файла для чтения, значение 1 - для сохранения
CAPTION	Заголовок диалога (тип STRING)
DIR	Каталог для выбора файла (тип STRING)
FILTER	Набор фильтров для диалога (тип STRING), например, "Images (*.png *.xpm *.jpg);;Text files (*.txt);;XML files (*.xml)"
SELECTED_FILTER	Выбранный фильтр (тип STRING)
Выходные переменные	
PATH	Путь к выбранному файлу, если нажата кнопка "OK", либо пустая строка, если нажата кнопка "Отмена"

2.1.3.6.2. Мультимедийные блоки

SOUND	
Проигрывает указанный Wav-файл	
Входные события	
PLAY	Запуск проигрывания файла
STOP	Остановка проигрывания файла
Входные переменные	
DATA	Имя элемента ресурса, содержащего аудиоданные
Выходные переменные	
PLAYING	TRUE, если в данный момент аудиофайл проигрывается, FALSE в противном случае

2.1.3.6.3. Печать

PRINTER	
Передает информацию на печать	
Входные события	
PRINT_SCREEN	Печать содержимого экрана
Настройка	

Окно настройки объекта приведено на рис. 2.30.

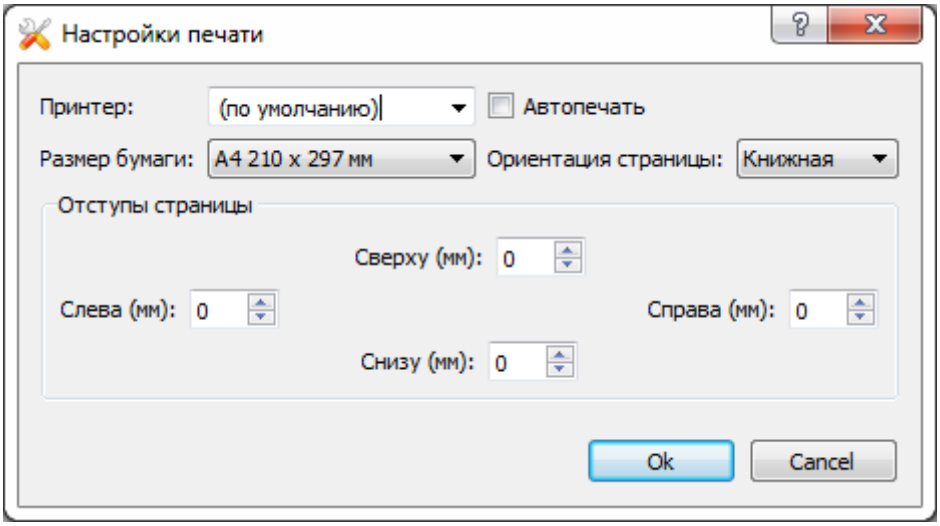


Рисунок 2.30 - Окно диалога настройки блока печати

Выпадающий список "Принтер" предназначен для выбора устройства, которое будет выбрано в качестве предлагаемого принтера для печати. Первым элементом выпадающего списка является строка "(по умолчанию)", при выборе которой в качестве принтера для печати будет

PRINTER
предложен принтер по умолчанию того компьютера, на котором будет выполняться печать. Имя принтера может быть выбрано как из выпадающего списка, так и введено вручную. Если включить опцию "Автопечать", то блок PRINTER не выведет диалог печати, а сразу же отправит данные на выбранный принтер. Если же данная опция отключена, то при вызове события печати блок выведет системный диалог, в котором можно будет выбрать принтер и указать требуемые параметры печати. Ниже выпадающего списка выбора принтера расположены выпадающие списки для выбора размера и ориентации бумаги. В нижней части диалога расположены элементы редактирования, предназначенные для задания отступов от края страницы. Значения задаются в миллиметрах.

2.1.3.6.4. Настройка главного окна

STATUS_BAR	
Строка статуса главного окна	
Входные переменные	
MESSAGE	Тип STRING. Сообщение, выводимое в строку статуса
VISIBLE	Тип BOOL. TRUE, если строка статуса присутствует в главном окне

2.1.3.6.5. Прочие типы функциональных блоков

BLINKER	
Переключатель цветов	
Входные переменные	
BASIC	Тип TColor. Основной цвет
ALTER	Тип TColor. Альтернативный цвет
FREQ	Тип INT. Определяет частоту переключения: 0 - нет переключения; 1 - очень часто (4 Гц); 2 - часто (2 Гц); 3 - редко (1 Гц); 4 - очень редко (0.5 Гц)
Выходные переменные	
OUT	Тип TColor. Текущий цвет
Описание	
Данный блок предназначен для организации переключения между двумя цветами, основным и альтернативным, с заданной частотой. Блок удобен для организации индикации аварийной ситуации, хода какого-то процесса (открытия клапана, например) и т.п. Все блоки, частота которых совпадает, переключаются на альтернативный цвет и возвращаются к базовому цвету одновременно.	

BLINKER
Если блок отключен (FREQ=0), то выходное значение цвета равно базовому цвету.

NAVIGATOR	
Блок навигации (переключения) между окнами приложения	
Входные события	
E_GO	Событие, при получении которого, на экране будет отображено окно со значением ID окна, как во входном параметре ID
Входные переменные	
ID	Тип STRING. ID окна, которое будет отображено на экране, при получении входного события E_GO
Описание	
Данный блок предназначен для организации переключения между окнами приложения	

2.2. Описание языка ST событийных приложений (применяется для написания алгоритмов в базовых блоках TBasic)

Structured Text (ST, язык структурированного текста) – текстовый язык программирования высокого уровня для создания гибких процедур обработки данных. По структуре этот язык похож на Паскаль. Основу языка ST составляют выражения, каждое из которых должно заканчиваться точкой с запятой. Для присвоения значений переменным используется оператор «:=». Для описания математических выражений используются общепринятые символы: +, *, /, (,), <, <=, >=, > и др. Порядок вычисления выражений такой же, как и в алгебре – действия выполняются слева направо. Сначала выполняются действия, заключенные в скобки, затем – операции умножения или деления, и только после этого – сложение или вычитание. Для увеличения функциональных возможностей языка структурированного текста в нём предусмотрены операторы выбора (IF ... THEN ... ELSE ... ENDIF), операторы множественного выбора (CASE ... OF ... ELSE ... ENDCASE), операторы цикла (WHILE ... DO ... ENDWHILE, REPEAT ... UNTIL ... ENDREPEAT или FOR ... TO ... BY ... DO ... ENDFOR).

2.2.1. Типы данных

Каждая переменная в языке ST обязана иметь тип. Язык ST строго типизированный, что означает необходимость приводить типы операндов к одному и тому же в каждой операции.

В реализации языка ST в SCADA-системе «Соната» предусмотрены 19 типов данных, совпадающих с видами сигналов (см. п.1.1):

Битовые типы данных (**BOOL**, **BYTE**, **WORD**, **DWORD**):

- **BOOL** – логический тип данных (0 или 1, FALSE или TRUE: за истину полагается единица, за ложь – ноль);

- **BYTE** – байт, размер – 8 бит, значения от 0 до 255;

- **WORD** – слово, размер – 16 бит, значения от 0 до 65 535;

- **DWORD** – двойное слово, размер – 32 бита, значения от 0 до 4 294 967 295.

Целочисленные типы данных (**USINT**, **SINT**, **UINT**, **INT**, **UDINT**, **DINT**, **ULINT**, **LINT**):

- **USINT** – беззнаковый, размер – 1 байт, значения от 0 до 255;
- **SINT** – знаковый, размер – 1 байт, значения от -128 до 127;
- **UINT** – беззнаковый, размер – 2 байта, значения от 0 до 65 535;
- **INT** – знаковый, размер – 2 байта, значения от -32 768 до 32 767;
- **UDINT** – беззнаковый, размер – 4 байта, значения от 0 до 4 294 967 295;
- **DINT** – знаковый, размер – 4 байта, значения от -2 147 483 648 до 2 147 483 647.
- **ULINT** – беззнаковый, размер – 8 байт, значения от 0 до 18 446 744 073 709 551 615.
- **LINT** – знаковый, размер – 8 байт, значения от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.

Вещественные типы данных (**REAL**, **LREAL**):

- **REAL** – размер – 4 байта, значения от 1,4×10−45 (1,2×10−38) до 3,4×10+38;
- **LREAL** – размер – 8 байт, значения от 5,0×10−324(2,3×10−308) до 1,7×10+308.

Типы данных, содержащие значения даты и времени (**DT**, **DATE**, **TIME**, **TOD**):

- **DT** – содержит значения даты и времени;
- **DATE** – содержит значение даты;
- **TOD** – содержит значение времени суток;
- **TIME** – содержит значение длительности – продолжительности промежутка времени.

Строковые типы данных (**STRING**):

- **STRING** – строка длиной до 511 байт.

Помимо перечисленных, в графических приложениях могут использоваться дополнительные типы данных:

- **color** – содержит значение цвета, размер – 4 байта, **0xAARRGGBB** (**A** – прозрачность, **RGB** – цвета);
- **pos** – содержит значения положения – структура с полями X, Y типа **LREAL**;
- **size** – содержит значения размера – структура с полями width, height типа **LREAL**;
- **font** – шрифт – структура с полями family (**STRING**), size(**INT**), bold (**BOOL**), italic(**BOOL**), underline (**BOOL**), strikeout (**BOOL**).

2.2.2. Родовые типы данных

Родовые типы данных – это типы данных, объединяющие группы простых типов. Например, тип **ANY_INT** включает простые типы **DINT**, **INT**, **SINT**, **UDINT**, **UINT** и **USINT**. Следовательно, если функция имеет на входе **ANY_INT**, то переменная-аргумент функции может иметь тип **DINT**, **INT**, **SINT**, **UDINT**, **UINT** или **USINT**.

Родовые типы данных выстраиваются по следующей иерархии:

ANY
ANY_NUM
ANY_REAL
REAL
LREAL
ANY_INT
LINT, DINT, INT, SINT

ULINT, UDINT, UINT, USINT
ANY_BIT
DWORD, WORD, BYTE, BOOL
STRING
TIME

2.2.3. Применение структур и массивов

Если необходимо использовать поле переменной структурного типа, то имя поля указывается после точки: ИМЯ_ПЕРЕМЕННОЙ.ИМЯ_ПОЛЯ.

Если необходимо использовать элемент массива, то индекс указывается в квадратных скобках: ИМЯ_ПЕРЕМЕННОЙ[ИНДЕКС]. Индекс задается выражением типа, имеющим тип UINT.

ВНИМАНИЕ! Индексация массивов начинается с 0 элемента, т.е. массив из 10 элементов нужно будет перебирать от 0 до 9 индекса.

2.2.4. Работа с битовыми полями у битовых типов данных (битовые строки)

Специальные типы BYTE, WORD, DWORD представляют собой битовые строки длиной 8, 16 и 32 бит соответственно. К битовым строкам можно обращаться побитно: ИМЯ_ПЕРЕМЕННОЙ.НОМЕР_БИТА. Например: a.3:=1 (установить бит 3 переменной a).

2.2.5. Числовые литералы

Литералы представляют собой константы, включаемые непосредственно в текст программы. Числовые литералы делятся на два класса: целые и вещественные. Количество символов в литерале зависит от типа литерала, допустимого диапазона значений и точности представления значения в выбранном типе.

Литерал может быть типизированным и нетипизированным. Типизированный литерал применяется тогда, когда необходимо точно указать тип задаваемого значения. Для того, чтобы задать типизированный литерал, необходимо перед строкой символов, задающей значение, указать тип, за именем которого должен следовать символ ‘#’. Если тип числового литерала не задан, то предполагается, что литерал с десятичной точкой имеет тип **LREAL**, а целый литерал имеет тип **INT**.

Целочисленный литерал может задавать не только десятичную константу, но и двоичную, восьмеричную или шестнадцатеричную. Для этого, непосредственно перед цифрами, необходимо поставить символы 2#, 8# или 16# соответственно. В последнем случае в качестве цифр также выступают символы A, B, C, D, E, F.

В качестве литерала логического типа могут также выступать предопределенные константы **TRUE** (истина) или **FALSE** (ложь).

Примеры числовых литералов приведены в таблице.

Целочисленные литералы	12 0 -127 61234

Вещественные литералы	-12.0 127.4
Вещественные литералы с экспоненциальной частью	-134.5e+5
Двоичные литералы	2#001110001
Восьмеричные литералы	8#12756
Шестнадцатеричные литералы	16#AB56FE
Логические 0 и 1	0 1
Логические TRUE и FALSE	TRUE FALSE
Типизированные литералы	UDINT#16#2578 –SINT#456

2.2.6. Строковые литералы

Строковый литерал представляет собой набор символов, заключенный в одинарные кавычки. Длина строки символов не может превышать 511 символов.

Помимо обычных символов, строка может содержать специальные символы, перечень которых приведен в таблице. Для вставки специального символа необходимо указать префикс, роль которого играет знак доллара, за которым следует собственно специальный символ.

$$	Знак доллара.
$’	Одиночная кавычка.
$L или $l	Перевод строки.
$N или $n	Новая строка.
$R или $r	Возврат каретки.
$T или $t	Табуляция.

2.2.7. Литералы продолжительности

Литерал продолжительности начинается с префикса T#, t#, TIME# или time#. После префикса следуют собственно данные, задаваемые днями, часами, секундами и миллисекундами или их комбинацией.

Примеры литералов продолжительности: T#14ms, T#-14ms, T#14.7s, T#14.7m, time#5d14h25m10s11ms.

2.2.8. Литералы даты и времени

Литерал даты начинается с префикса DATE# или D#, за которым следует строка даты в формате ‘YYYY-MM-DD’, например DATE#1984-06-25.

Литерал времени суток начинается с префикса TIME_OF_DAY# или TOD#, за которым следует строка в формате ‘HH:MM:SS.MS’, например TOD#16:25:47.56.

Литерал даты и времени начинается с префикса DATE_AND_TIME# или DT#, за которым следует строка даты, затем тире, затем строка времени, например: DT#1984-06-25-16:25:47.56.

2.2.9. Массивы

Если необходимо использовать элемент массива, то индекс указывается в квадратных скобках: ИМЯ_ПЕРЕМЕННОЙ[ИНДЕКС]. Индекс задается переменной или выражением, имеющим тип UINT.

ВНИМАНИЕ! Индексация массивов начинается с 0 элемента, т.е. массив из 10 элементов нужно будет перебирать от 0 до 9 индекса.

2.2.10. Выражения

Выражение состоит из операций и операндов. Перечень операций приведен в таблице. Операции приведены в порядке убывания приоритета.

Операция	Символ	Приоритет
Скобки (изменение приоритета)	()	Самый высокий
Вычисление функций	ИМЯ_ФУНКЦИИ(аргумент, аргумент)	
Унарный минус Логическое, либо побитовое «НЕ»	- NOT	
Возведение в степень	**	
Умножение Деление Остаток от деления	* / MOD	
Сложение Вычитание	+ -	
Сравнение	<, >, <=, >=	
Равенство Неравенство	= <>	
Логическое, либо побитовое «И»	AND	
Логическое, либо побитовое «исключающее ИЛИ»	XOR	
Логическое, либо побитовое «ИЛИ»	OR	Самый низкий

В качестве операнда может быть использована константа, переменная, результат вызова функции или другое выражение. В качестве переменной может использоваться поле структуры или элемент массива. Если необходимо использовать поле переменной структурного типа, то имя

поля указывается после точки: ИМЯ_ПЕРЕМЕННОЙ.ИМЯ_ПОЛЯ. Если необходимо использовать элемент массива, то индекс указывается в квадратных скобках: ИМЯ_ПЕРЕМЕННОЙ[ИНДЕКС]. Индекс задается выражением типа, имеющим тип UINT.

2.2.11. Инструкции

Алгоритм представляет собой набор инструкций, разделенных точкой с запятой (‘;’). Перечень инструкций языка ST, реализованных в системе «Sonata», приведен в таблице.

Инструкция	Синтаксис
Присваивание	ПЕРЕМЕННАЯ := ВЫРАЖЕНИЕ;

IF	IF УСЛОВИЕ THEN
	ИНСТРУКЦИИ
	ELSEIF УСЛОВИЕ THEN
	ИНСТРУКЦИИ
	ELSE
	ИНСТРУКЦИИ
	END_IF;

FOR	FOR ПЕРЕМЕННАЯ:=НАЧ_ЗНАЧ TO КОН_ЗНАЧ BY ШАГ DO ИНСТРУКЦИИ END_FOR
WHILE	WHILE УСЛОВИЕ DO ИНСТРУКЦИИ END_WHILE;
REPEAT	REPEAT ИНСТРУКЦИИ UNTIL УСЛОВИЕ END_REPEAT;

2.2.11.1. Особенности событийного ST (используется в Базовых блоках)

В событийном ST нужно учитывать следующие особенности:

- нельзя использовать имя var, array в качестве внутренних, входных или выходных переменных, т.к. это ключевые слова;

- нельзя работать с битовыми полями, т.е. с конструкиями типа a.3 := 1, где a имеет тип данных BYTE, WORD или DWORD и мы хотели бы установить 3 бит данной переменной;

- индекс элементов массива имеет тип данных UINT;

- при проектировании поведения Базового блока нужно учитывать, что факт появления входного события запоминается до момента реакции на его появление. Т.е., на примере рис. 2.31, если сначала появится событие event2, то факт его появления будет запомнен и при появлении события event1 произойдёт переход из состояния state1 в state2, а так же дальше из state2 в state3 и т.д.

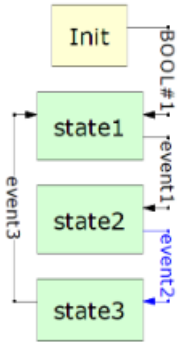


Рисунок 2.31 - Пример схемы переходов состояний в Базовом блоке

В событийном ST нельзя использовать следующие функции:
- EXIT.

3. РАЗРАБОТКА ЦИКЛИЧЕСКИХ ПРОГРАММ

3.1. Описание языка ST циклических приложений

3.1.1. Типы данных

3.1.1.1. Основные типы данных

Каждая переменная в языке ST обязана иметь тип. Язык ST строго типизированный, что означает необходимость приводить типы операндов к одному и тому же в каждой операции.

В реализации языка ST в системе «Sonata» предусмотрены следующие типы данных:

Битовые типы данных	
BOOL	0 или 1, FALSE или TRUE
BYTE	8 бит (0…255)
WORD	16 бит (0…65535)
DWORD	32 бита (0.. 4294967295)
Целочисленные типы данных	
SINT	1 байт, знаковый (-128…127)
INT	2 байта, знаковый (-32768…32767)
DINT	4 байта, знаковый (-2147483648… 2147483647)
LINT	8 байт, знаковый (-9223372036854775808… 9223372036854775807)
USING	1 байт, беззнаковый (0…255)
UINT	2 байта, беззнаковый (0…65535)
UDINT	4 байта, беззнаковый (0.. 4294967295)
ULINT	8 байт, беззнаковый
Вещественные типы данных	
REAL	4 байта, одинарной точности
LREAL	8 байт, двойной точности
Дата и время	
DT	Дата и время
DATE	Дата
TOD	Время
TIME	Продолжительность
Строковые типы данных	

STRING	Строка длиной до 511 символов

Помимо перечисленных, в графических приложениях могут использоваться дополнительные типы данных:

TColor	Цвет, 4 байта, 0xAARRGGBB (A – прозрачность, RGB – цвета)
TPos	Положение. Структура с полями X, Y типа LREAL
TSize	Размер. Структура с полями width, height типа LREAL
TFont	Шрифт. Структура с полями family (STRING), size(INT), bold (BOOL), italic(BOOL), underline (BOOL), strikeout (BOOL)

3.1.1.2. Родовые типы данных

Родовые типы данных – это типы данных, объединяющие группы простых типов. Например, тип ANY_INT включает простые типы DINT, INT, SINT, UDINT, UINT и USINT. Следовательно, если функция имеет на входе ANY_INT, то переменная-аргумент функции может иметь тип DINT, INT, SINT, UDINT, UINT или USINT.

Родовые типы данных:
ANY
ANY_ELEMENTRARY
ANY_MAGNITUDE
ANY_NUM
ANY_REAL
REAL
LREAL
ANY_INT
LINT, DINT, INT, SINT
ULINT, UDINT, UINT, USINT
TIME
ANY_BIT
DWORD, WORD, BYTE, BOOL
STRING
ANY_DATE
DATE, TOD, DT

3.1.1.3. Применение структур и массивов

Если необходимо использовать поле переменной структурного типа, то имя поля указывается после точки: ИМЯ_ПЕРЕМЕННОЙ.ИМЯ_ПОЛЯ.

Если необходимо использовать элемент массива, то индекс указывается в квадратных скобках: ИМЯ_ПЕРЕМЕННОЙ[ИНДЕКС]. Индекс задается выражением типа, имеющим тип INT.

3.1.1.4. Работа с битовыми полями у битовых типов данных (битовые строки)

Специальные типы BYTE, WORD, DWORD представляют собой битовые строки длиной 8, 16 и 32 бит соответственно. К битовым строкам можно обращаться побитно: ИМЯ_ПЕРЕМЕННОЙ.НОМЕР_БИТА. Например: a.3:=1 (установить бит 3 переменной a).

3.1.1.5. Числовые литералы

Числовые литералы делятся на два класса: целые и вещественные. Количество символов в литерале зависит от типа литерала, допустимого диапазона значений и точности представления значения в выбранном типе.

Литерал может быть типизированным и нетипизированным. Типизированный литерал применяется тогда, когда необходимо точно указать тип задаваемого значения. Для того, чтобы задать типизированный литерал, необходимо перед строкой символов, задающей значение, указать тип, за именем которого должен следовать символ ‘#’ . Если тип числового литерала не задан, то предполагается, что литерал с десятичной точкой имеет тип REAL, а целый литерал имеет тип INT.

Целочисленный литерал может задавать не только десятичную константу, но и двоичную, восьмеричную или шестнадцатеричную. Для этого, непосредственно перед цифрами, необходимо поставить символы 2#, 8# или 16# соответственно. В последнем случае в качестве цифр также выступают символы A, B, C, D, E, F.

В качестве литерала логического типа могут также выступать предопределенные константы TRUE (истина) или FALSE (ложь).

Примеры числовых литералов приведены в таблице.

Целочисленные литералы	12 0 -127 61234
Вещественные литералы	-12.0 127.4
Вещественные литералы с экспоненциальной частью	-134.5e+5
Двоичные литералы	2#001110001
Восьмеричные литералы	8#12756
Шестнадцатеричные литералы	16#AB56FE
Логические 0 и 1	0 1
Логические TRUE и FALSE	TRUE FALSE
Типизиров. литералы	UDINT#16#2578 –SINT#456

3.1.1.6. Строковые литералы

Строковый литерал представляет собой набор символов, заключенный в одинарные кавычки. Длина строки символов не может превышать 511 символов.

Помимо обычных символов, строка может содержать специальные символы, перечень которых приведен в таблице. Для вставки специального символа необходимо указать префикс, роль которого играет знак доллара, за которым следует собственно специальный символ.

$$	Знак доллара.
$’	Одиночная кавычка.
$L или $l	Перевод строки.
$N или $n	Новая строка.
$R или $r	Возврат каретки.
$T или $t	Табуляция.

3.1.1.7. Литералы продолжительности

Литерал продолжительности начинается с префикса T#, t#, TIME# или time#. После префикса следуют собственно данные, задаваемые днями, часами, секундами и миллисекундами или их комбинацией.

Примеры литералов продолжительности: T#14ms, T#-14ms, T#14.7s, T#14.7m, time#5d14h25m10s11ms.

3.1.1.8. Литералы даты и времени

Литерал даты начинается с префикса DATE# или D#, за которым следует строка даты в формате ‘YYYY-MM-DD’, например DATE#1984-06-25.

Литерал времени суток начинается с префикса TIME_OF_DAY# или TOD#, за которым следует строка в формате ‘HH:MM:SS.MS’, например TOD#16:25:47.56.

Литерал даты и времени начинается с префикса DATE_AND_TIME# или DT#, за которым следует строка даты, затем тире, затем строка времени, например: DT#1984-06-25-16:25:47.56.

3.1.2. Выражения

Выражение состоит из операций и операндов. Перечень операций приведен в таблице. Операции приведены в порядке убывания приоритета.

Операция	Символ	Приоритет
Скобки (изменение приоритета)	()	Самый высокий
Вычисление функций	ИМЯ_ФУНКЦИИ(аргумент, аргумент, …)	
Унарный минус Логическое либо побитовое «НЕ»	- NOT	
Возведение в степень	**	
Умножение Деление Остаток от деления	* / MOD	

Сложение Вычитание	+ -	
Сравнение	<, >, <=, >=	
Равенство Неравенство	= <>	
Логическое либо побитовое «И»	AND	
Логическое либо побитовое «исключающее ИЛИ»	XOR	
Логическое либо побитовое «ИЛИ»	OR	Самый низкий

В качестве операнда может быть использована константа, переменная, результат вызова функции или другое выражение. В качестве переменной может использоваться поле структуры или элемент массива. Если необходимо использовать поле переменной структурного типа, то имя поля указывается после точки: ИМЯ_ПЕРЕМЕННОЙ.ИМЯ_ПОЛЯ. Если необходимо использовать элемент массива, то индекс указывается в квадратных скобках: ИМЯ_ПЕРЕМЕННОЙ[ИНДЕКС]. Индекс задается выражением типа, имеющим тип INT.

3.1.3. Вызов функций

Выражение может содержать вызовы функций, как системных, так и пользовательских, включенных разработчиком в проект. Для вызова функции необходимо указать ее имя, после чего привести в скобках перечень аргументов, разделенный запятыми.

Язык ST позволяет использовать два синтаксиса передачи аргументов: неформальный и формальный.

Неформальный синтаксис передачи аргументов предполагает, что аргументы перечислены при вызове функции точно в том же порядке, в котором они перечислены в описании функции. Количество аргументов должно точно соответствовать количеству входных, выходных и входных/выходных переменных. Для передачи входного аргумента используется выражение, для передачи выходных и входных/выходных аргументов должна использоваться переменная. Например, если описание функции FUNC содержит переменные A:INT (вход), B:INT (выход), C:INT (внутренняя), D:INT (вход/выход), то ее вызов может быть таким:

E := FUNC(F + 2, G, H),

где F, G, H – некие переменные типа INT. При таком вызове значение выражения F+2 будет передано в параметр A, значение переменной H будет передано параметру D. После вычисления функции значение параметра B будет возвращено в переменную G, а значение параметра D – в переменную H.

Формальный синтаксис передачи параметров предполагает, что аргументы перечисляются в произвольном порядке. Количество их также произвольное – можно не указывать входные и выходные аргументы. Параметры, объявленные как вход/выход, обязаны присутствовать в списке аргументов.

Для передачи входного аргумента используется строка вида «ИМЯ_ПАРАМЕТРА:=ВЫРАЖЕНИЕ», где ИМЯ_ПАРАМЕТРА – это имя входной переменной функции. Для передачи выходного аргумента либо аргумента вход/выход используется строка вида «ИМЯ_ПАРАМЕТРА=>ПЕРЕМЕННАЯ», где ИМЯ_ПАРАМЕТРА – это имя выходной переменной функции либо переменной вход/выход. ПЕРЕМЕННАЯ – имя переменной, через которую будет передаваться значение и в которую будет возвращено значение после выполнения тела функции.

Примером использования формального синтаксиса передачи параметров может служить следующая строка кода:

E := FUNC(B=>G, A:=F + 2).

3.1.4. Функции пользователя

Функция представляет собой элемент организации программы (POU, в терминах стандарта IEC 61113), не сохраняющий свое внутреннее состояние.

Для описания функции необходимо указать список ее переменных, после чего написать тело функции на языке ST. Тело функции представляет собой набор инструкций языка ST, каждая из которых должна заканчиваться точкой с запятой. Помимо списка переменных, задаваемых разработчиком, в функции всегда присутствуют три системные переменные EN:BOOL (разрешение выполнения функции), ENO:BOOL(ошибка выполнения), а также переменная, имя которой совпадает с именем функции, через которую производится возврат результата выполнения функции.

Если значение переменной EN равно FALSE, то тело функции выполнено не будет и значение переменной ENO будет установлено в FALSE (ошибка). Если значение EN равно TRUE, то программа установит значение переменной ENO в TRUE и выполнит тело функции.

Если значение входной переменной функции передано через список аргументов, то данная переменная будет инициализирована переданным значением, в противном случае входная переменная инициализируется значением по умолчанию.

Выходная переменная перед выполнением функции инициализируется значением по умолчанию. Значение выходной переменной передается вызывающему блоку через переменную, указанную в списке аргументов.

Переменная вход/выход инициализируется значением переменной, переданной в списке аргументов. После выполнения тела функции в переданную переменную будет возвращено измененное значение.

Помимо переменных, перечисленных при описании функции, в функции можно использовать глобальные переменные. Глобальной переменной считается входная, выходная, вход/выход переменная программы ST.

Пример функции, осуществляющей деление двух чисел типа LREAL:

```
DIV_LREAL – функция, результат типа LREAL
A, B – входные переменные типа LREAL

Тело функции:

if B = LREAL#0 then
    ENO := FALSE;
else
    DIV_LREAL := A / B;
end_if;
```

3.1.5. Функциональные блоки пользователя

Функциональный блок представляет собой элемент организации программы (POU, в терминах стандарта IEC 61113), сохраняющий свое внутреннее состояние.

Для описания блока необходимо указать список его переменных, после чего написать тело блока на языке ST. Тело блока представляет собой набор инструкций языка ST, каждая из которых должна заканчиваться точкой с запятой. Помимо списка переменных, задаваемых разработчиком, в функции всегда присутствуют две системные переменные EN:BOOL (разрешение выполнения блока) и ENO:BOOL(ошибка выполнения).

Если значение переменной EN равно FALSE, то тело блока выполнено не будет и значение переменной ENO будет установлено в FALSE (ошибка). Если значение EN равно TRUE, то программа установит значение переменной ENO в TRUE и выполнит тело блока.

Если значение входной переменной блока передано через список аргументов, то данная переменная будет инициализирована переданным значением, в противном случае входная переменная инициализируется значением по умолчанию.

Выходная переменная перед выполнением блока инициализируется значением по умолчанию. Значение выходной переменной передается вызывающему блоку через переменную, указанную в списке аргументов.

Переменная вход/выход инициализируется значением переменной, переданной в списке аргументов. После выполнения тела блока в переданную переменную будет возвращено измененное значение.

Помимо переменных, перечисленных при описании блока, в теле блока можно использовать глобальные переменные. Глобальной переменной считается входная, выходная, вход/выход переменная программы ST.

Пример блока, осуществляющего вычисление корней квадратного уравнения:

```
A, B, C – входные переменные типа LREAL
D – внутренняя переменная типа LREAL
X1, X2 – выходные переменные типа LREAL

Тело блока:

D := B * B – LREAL#4 * A * C;
if D < LREAL#0 then
    ENO := FALSE;
else
    X1 := (-B + SQRT(D)) / (LREAL#2 * A);
    X2 := (-B - SQRT(D)) / (LREAL#2 * A);
end_if
```

3.1.6. Инструкции

Тело функции, функционального блока или программы представляет собой последовательность инструкций, разделенных точкой с запятой (‘;’).

3.1.6.1. Инструкция присваивания

Инструкция присваивания имеет следующий вид:

ПЕРЕМЕННАЯ := ВЫРАЖЕНИЕ;

Типы переменной и выражения должны совпадать. В качестве переменной может выступать поле структуры или элемент массива.

3.1.6.2. Условная инструкция

Инструкция имеет следующий вид:

IF*условие***THEN**
набор инструкций
ELSIF*условие***THEN**
набор инструкций
ELSE
набор инструкций
END_IF;

Условие должно быть выражением типа BOOL. Ветвей ELSIF может быть произвольное количество. Ветвь ELSE может отсутствовать, но, если она присутствует, то она должна быть последней.

3.1.6.3. Инструкция выбора

Инструкция имеет следующий вид:

CASE*выражение***OF**
*значения***:**
список инструкций
ELSE
набор инструкций
END_CASE;

Выражение должно иметь целый тип (группа ANY_INT). Типы возможных значений должны совпадать с типом выражения. Значения задаются путем перечисления (1,2,3,7) или интервалом (1..7). В одном списке можно использовать оба способа задания значений (1,2,7..9).

3.1.6.4. Инструкция цикла FOR

Инструкция имеет следующий вид:

FOR переменная:=*init* **TO** final **BY** incr **DO**
список инструкций
END_FOR;

Переменная-счетчик цикла должна иметь целый тип (группа ANY_INT). Типы выражений init (начальное значение счетчика цикла), final (конечное значение счетчика цикла), incr (приращение счетчика цикла) должны совпадать с типом переменной. Значения всех выражений вычисляются один раз перед первым выполнением тела цикла.

Фрагмент BY может отсутствовать. В этом случае значение приращения принимается равным 1.

3.1.6.5. Инструкция цикла WHILE

Инструкция имеет следующий вид:
WHILE условие **DO**
список инструкций
END_WHILE;
Условие должно быть выражением типа BOOL. Цикл будет выполняться до тех пор, пока условие остается истинным.

3.1.6.6. Инструкция цикла REPEAT

Инструкция имеет следующий вид:
REPEAT
список инструкций
UNTIL условие
END_REPEAT;
Условие должно быть выражением типа BOOL. Цикл будет выполняться до тех пор, пока условие не станет истинным.

3.1.6.7. Инструкция выхода из цикла EXIT

Инструкция применяется для выхода из цикла, минуя проверку условия завершения. Если несколько циклов вложено друг в друга, то выход будет осуществлен только из того цикла, в котором вызвана данная инструкция.

3.1.6.8. Инструкция выхода из функции или функционального блока RETURN

Данная инструкция применяется для выхода из функции или функционального блока ранее, чем закончится выполнение тела функции или блока.

3.1.6.9. Инструкция вызова функционального блока

Для вызова функционального блока необходимо создать переменную, тип которой совпадает с типом вызываемого блока. Если такая переменная создается на уровне программы (верхний уровень иерархии), то переменная типа функционального блока может быть только внутренней.
Синтаксис вызова функционального блока должен быть следующим:

ИМЯ_ПЕРЕМЕННОЙ_БЛОКА(перечень аргументов);

Как и в случае функций, синтаксис передачи аргументов может быть формальным и не формальным. Описание каждой разновидности приведено в описании вызова функций.

Значения выходных и вход/выход переменных блока могут быть получены после вызова блока. Обращение к ним осуществляется так же, как и к полям переменной структурного типа.

3.2. Библиотека функциональных блоков циклических приложений

Перечисленные в данном приложении функции реализованы для циклических приложений, написанных на языке ST.

Помимо указанных в таблице аргументов, каждый функциональный блок имеет две дополнительные переменные логического (BOOL) типа:

EN – разрешение выполнения блока;

ENO – ошибка выполнения.

Если значение входной переменной EN истинно, то функциональный блок будет выполнен. Если в ходе выполнения блока не возникнет ошибок, то значение переменной ENO будет истинным. При возникновении ошибки значение переменной ENO станет ложным. Если же значение входной переменной EN ложно, то значение переменной ENO станет ложным и блок выполнен не будет.

3.2.1. Блоки поразрядных действий

UNPACK	
Распаковка BYTE-значения в 8 BOOL-значений (поразрядно)	
Входные переменные	
X	Тип BYTE. Распаковываемое значение
Выходные переменные	
B0	Тип BOOL. Значение разряда 0
B1	Тип BOOL. Значение разряда 1
B2	Тип BOOL. Значение разряда 2
B3	Тип BOOL. Значение разряда 3
B4	Тип BOOL. Значение разряда 4
B5	Тип BOOL. Значение разряда 5
B6	Тип BOOL. Значение разряда 6
B7	Тип BOOL. Значение разряда 7

3.2.2. Триггеры

SR
SR - триггер.

SR	
Входные переменные	
S1	BOOL
R	BOOL
Выходные переменные	
Q1	BOOL Если S1=1, то Q1=1 вне зависимости от состояния R. Если S1=0 и R=1, то Q1=0. Если S1=0 и R=0, то состояние блока не изменяется

RS	
RS - триггер	
Входные переменные	
S	BOOL
R1	BOOL
Выходные переменные	
Q1	BOOL Если R1=1, то Q1=0 вне зависимости от состояния S. Если S=1 и R1=0, то Q1=1. Если S1=0 и R=0, то состояние блока не изменяется

3.2.3. Регистрация фронтов

R_TRIG	
Регистратор переднего фронта	
Входные переменные	
CLK	BOOL
Выходные переменные	
Q	BOOL Если значение CLK изменяется с 0 на 1, то значение Q становится равным 1 и сбрасывается в 0 на следующем цикле вычисления

F_TRIG
Регистратор заднего фронта
Входные переменные

F_TRIG	
CLK	BOOL
Входные переменные	
Q	BOOL Если значение CLK изменяется с 1 на 0, то значение Q становится равным 1 и сбрасывается в 0 на следующем цикле вычисления

3.2.4. Счетчики

CTU	
Возрастающий счетчик типа INT	
Входные переменные	
CU	BOOL; увеличение счетчика
R	BOOL; сброс счетчика
PV	INT; верхняя граница счетчика
Выходные переменные	
Q	BOOL Равна 1, если CV >= PV
CV	INT Текущее значение счетчика. Увеличивается на каждом цикле вычисления, при CU=1, пока не достигнет значения PV

CTU_*	
Возрастающий счетчик типа DINT, LINT, UDINT, ULINT (*)	
Входные переменные	
CU	BOOL; увеличение счетчика
R	BOOL; сброс счетчика
PV	тип *; верхняя граница счетчика
Выходные переменные	
Q	BOOL Равна 1, если CV >= PV
CV	тип * Текущее значение счетчика. Увеличивается на каждом цикле вычисления, при CU=1, пока не достигнет значения PV

CTD	
Убывающий счетчик типа INT	
Входные переменные	
CD	BOOL; уменьшение счетчика
LD	BOOL; сброс счетчика в значение PV
PV	INT; начальное значение счетчика
Выходные переменные	
Q	BOOL Равна 1, если CV <= 0
CV	INT Текущее значение счетчика. Уменьшается на каждом цикле вычисления, при CD=1, пока не достигнет значения 0

CTD_*	
Убывающий счетчик типа DINT, LINT, UDINT, ULINT (*)	
Входные переменные	
CD	BOOL; уменьшение счетчика
LD	BOOL; сброс счетчика в значение PV
PV	тип *; начальное значение счетчика
Выходные переменные	
Q	BOOL Равна 1, если CV <= 0
CV	тип * Текущее значение счетчика. Уменьшается на каждом цикле вычисления, при CD=1, пока не достигнет значения 0

CTUD	
Двунаправленный счетчик типа INT	
Входные переменные	
CU	BOOL; увеличение счетчика
CD	BOOL; уменьшение счетчика
R	BOOL; сброс CV в значение 0
LD	BOOL; сброс CV в значение PV
PV	INT; конечное значение счетчика при возрастании; начальное значение счетчика при сбросе переменной LD
Выходные переменные	

CTUD	
QU	BOOL Равна 1, если CV >= PV
QD	BOOL Равна 1, если CV <= 0
CV	INT Текущее значение счетчика. Увеличивается на каждом цикле вычисления при CU=1 пока не достигнет значения PV, уменьшается на каждом цикле вычисления, при CD=1, пока не достигнет значения 0

CTUD_*	
Двунаправленный счетчик типа DINT, LINT, ULINT, UDINT (*)	
Входные переменные	
CU	BOOL; увеличение счетчика
CD	BOOL; уменьшение счетчика
R	BOOL; сброс CV в значение 0
LD	BOOL; сброс CV в значение PV
PV	тип *; конечное значение счетчика при возрастании; начальное значение счетчика при сбросе переменной LD
Выходные переменные	
QU	BOOL Равна 1, если CV >= PV
QD	BOOL Равна 1, если CV <= 0
CV	тип * Текущее значение счетчика. Увеличивается на каждом цикле вычисления при CU=1, пока не достигнет значения PV, уменьшается на каждом цикле вычисления, при CD=1, пока не достигнет значения 0

3.2.5. Блоки таймеров

TP	
Импульс	
Входные переменные	

TP	
IN	BOOL; включение
PT	TIME; продолжительность импульса
Выходные переменные	
Q	BOOL Выход блока импульса
ET	TIME Время, нарастающее от 0 в момент включения импульса до PT, когда выход Q выключается. Счетчик сбрасывается при завершении импульса

TON	
Задержка включения	
Входные переменные	
IN	BOOL; включение
PT	TIME; длительность задержки
Выходные переменные	
Q	BOOL Выход блока задержки. Q становится равным 1 через время PT после перехода IN в состоянии 1. Если IN сбрасывается в 0, то выход блока также сбрасывается в 0
ET	TIME Время, нарастающее от 0 в момент перехода IN в 1 до PT, когда выход Q включается. Счетчик сбрасывается при переходе IN в состояние 0

TOF	
Задержка выключения	
Входные переменные	
IN	BOOL
PT	TIME; длительность задержки
Выходные переменные	
Q	BOOL Выход блока задержки. Q становится равным 0 через время PT после перехода IN в состояние 0. Если IN сбрасывается в 1, то выход блока также сбрасывается в 1

TOF	
ET	TIME Время, нарастающее от 0 в момент перехода IN в 0 до PT, когда выход Q выключается. Счетчик сбрасывается при переходе IN в состояние 1

3.2.6. Блоки работы с датой и временем

DECODE_TIME	
Разбор продолжительности	
Входные переменные	
IN	продолжительность, TIME
Выходные переменные	
DAYS	дни, UDINT
HOURS	часы, UINT
MINS	минуты, UINT
SECS	секунды, UINT
USECS	микросекунды, UDINT

DECODE_DATE	
Разбор даты	
Входные переменные	
IN	дата, DATE
Выходные переменные	
YEAR	год, UINT
MONTH	месяц, UINT
DAY	день, UINT

DECODE_TOD	
Разбор времени суток	
Входные переменные	
IN	время суток, TOD
Выходные переменные	
HOURS	часы, UINT
MINS	минуты, UINT
SECS	секунды, UINT
USECS	микросекунды, UDINT

DECODE_DT	
Разбор даты и времени	
Входные переменные	
IN	дата и время, DT
Выходные переменные	
YEAR	год, UINT
MONTH	месяц, UINT
DAY	день, UINT
HOURS	часы, UINT
MINS	минуты, UINT
SECS	секунды, UINT
USECS	микросекунды, UDINT

3.2.7. Блоки работы с аналоговыми сигналами

LAG1	
НЧ-фильтр первого порядка	
Входные переменные	
RUN	Тип BOOL. FALSE для инициализации, TRUE для реальной работы
XIN	Тип REAL. Входной сигнал
TAU	Тип TIME. Постоянная времени фильтра
CYCLE	Тип TIME. Временной интервал между вызовами функционального блока
Выходные переменные	
XOUT	Тип REAL. Отфильтрованный сигнал

INTEGRAL	
Интегрирование сигнала по времени методом прямоугольников	
Входные переменные	
RUN	Тип BOOL. FALSE для приостановки, TRUE для интегрирования
R1	Тип BOOL. TRUE для сброса блока в начальное состояние
XIN	Тип REAL. Входной сигнал
X0	Тип REAL. Начальное значение
CYCLE	Тип TIME. Временной интервал между вызовами функционального блока

INTEGRAL	
Выходные переменные	
Q	Тип BOOL. FALSE, если блок находится в состоянии сброса (not R1)
XOUT	Тип REAL. Результат интегрирования

DERIVATIVE	
Дифференцирование сигнала по времени	
Входные переменные	
RUN	Тип BOOL. FALSE для сброса, TRUE для дифференцирования
XIN	Тип REAL. Входной сигнал
CYCLE	Тип TIME. Временной интервал между вызовами функционального блока
Выходные переменные	
XOUT	Тип REAL. Результат дифференцирования

HYSTERESIS	
Гистерезис входных сигналов XIN1, XIN2 с учетом погрешности EPS	
Входные переменные	
XIN1	Тип REAL. Первый входной сигнал
XIN2	Тип REAL. Второй входной сигнал
EPS	Тип REAL. Погрешность
Выходные переменные	
Q	Тип BOOL. TRUE, если XIN1 превышает (XIN2 + EPS). FALSE, если XIN1 менее (XIN2 - EPS)

PID	
Функциональный блок ПИД-закона регулирования	
Входные переменные	
AUTO	Тип BOOL. TRUE, если управление включено
PV	Тип REAL. Выходное значение сигнала объекта управления
SP	Тип REAL. Требуемое значение выхода объекта управления. ПИД регулятор вычисляет рассогласование (ошибку) между желаемым значением (SP) и действительным значением (PV) сигнала с объекта управления. По этой ошибке и происходит управление

PID	
X0	Тип REAL. Уставка для сравнения с входным сигналом
KP	Тип REAL. Коэффициент пропорциональной составляющей
TR	Тип REAL. Постоянная интегрирования
TD	Тип REAL. Постоянная времени дифференцирования
CYCLE	Тип TIME. Временной интервал между вызовами функционального блока
Выходные переменные	
XOUT	Тип REAL. Выходная переменная

RAMP	
Изменение сигнала в зависимости от времени	
Входные переменные	
RUN	Тип BOOL. TRUE, если выходной сигнал изменяется от X0 к X1. FALSE, если блок выключен и значение сигнала остается равным X0
X0	Тип REAL. Начальное значение сигнала
X1	Тип REAL. Конечное значение сигнала
TR	Тип TIME. Продолжительность такта
CYCLE	Тип TIME. Временной интервал между вызовами функционального блока
Выходные переменные	
BUSY	Тип BOOL. TRUE, если происходит изменение выходного сигнала. FALSE, если вычисление выхода закончено, либо не начато
XOUT	Тип REAL. Выходное значение сигнала

3.2.8. Блоки среды исполнения

3.2.8.1. Блоки работы с событиями и тревогами

ADD_EVENT	
Добавляет событие	
Входные переменные	
CATEGORY	Тип DINT. Категория события (0..9999)

ADD_EVENT	
SOURCE	Тип STRING. Источник события
MESSAGE	Тип STRING. Сообщение события
TAG	Тип STRING. Дополнительные данные события
STATE	Тип INT. Состояние события (0 - исчезновение, 1 - появление)

4. БИБЛИОТЕКА ФУНКЦИЙ

4.1. Функции языка ST циклических приложений

Перечисленные в данном приложении функции реализованы как для циклических приложений, написанных на языке ST, так и для базовых блоков событийных приложений.

Помимо указанных в таблице аргументов, каждая функция имеет две дополнительные переменные логического (BOOL) типа:

EN – разрешение вычисления функции;

ENO – ошибка вычисления.

Если значение входной переменной EN истинно, то функция будет вычислена. Если в ходе выполнения функции не возникнет ошибок, то значение переменной ENO будет истинным. При возникновении ошибки значение переменной ENO станет ложным. Если же значение входной переменной EN ложно, то значение переменной ENO станет ложным и функция вычислена не будет.

4.1.1. Общие функции

4.1.1.1. Функции преобразования типа

4.1.1.1.1. Простое преобразование

***_TO_**(IN0: ANY_ELEMENTARY): ANY_ELEMENTARY**	
Преобразование значения. * - исходный тип, ** - конечный тип. Допустимы следующие преобразования: ANY_BIT, ANY_INT, STRING в BOOL, BYTE, WORD, DWORD ANY_BIT, ANY_NUM, STRING в USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT ANY_NUM, STRING, TIME в REAL, LREAL ANY_ELEMENTARY в STRING TOD, DT, STRING в DATE DATE, DT, STRING в TOD DATE, TOD, STRING в DT	
Аргументы	
IN0	Преобразуемое значение
Результат	
Преобразованное значение. В случае ошибки преобразования значение ENO станет ложным. В этом случае выходное значение будет значением по умолчанию для конечного типа (0 для числовых, пустая строка и т.п.).	

TRUNC(IN0: ANY_REAL): ANY_INT	
Отсечение дробной части. 1.6 ⇒ 1 -1.6 ⇒ -1 1.4 ⇒ 1 -1.4 ⇒ -1	
Аргументы	
IN0	LREAL, REAL
Результат	
Значение IN0, преобразованное к типу DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT	

4.1.1.1.2. Двоично-десятичный код

***_BCD_TO_**(IN0: ANY_BIT): ANY_INT**	
Преобразование значения из двоично-десятичного кода в беззнаковое целое. * - исходный тип (BYTE, WORD, DWORD); ** - конечный тип (USINT, UINT, UDINT, ULINT)	
Аргументы	
IN0	Исходное значение
Результат	
Входное значение, преобразованное к беззнаковому целому. В случае ошибки преобразования значение ENO станет ложным. Если причиной ошибки был полубайт, значение которого превышает 9, то результат функции будет нулевым. Если исходное значение по размеру превышает конечное, то результатом будет преобразованное значение требуемого количества младших полубайт (2 для преобразования в USINT, 4 в UINT, 8 в UDINT).	

***_TO_BCD_**(IN0: ANY_INT): ANY_BIT**	
Преобразование значения из беззнакового целого в двоично-десятичный код. * - исходный тип (USINT, UINT, UDINT, ULINT); ** - конечный тип (BYTE, WORD, DWORD)	
Аргументы	
IN0	Исходное значение
Результат	
Входное значение, преобразованное к двоично-десятичному виду. Если значение входного аргумента слишком велико (например, 100 для преобразования в двоично-десятичный код типа BYTE), то значение ENO станет ложным. Результатом выполнения будет преобразованное значение, включающее в себя количество младших цифр десятичного представления исходного числа, соответствующее максимальному количеству полубайт результата (2 для преобразования в BYTE, 4 в WORD, 8 в DWORD).	

4.1.1.1.3. Форматирование

4.1.1.1.3.1. Вещественные числа

FORMAT_LREAL(VALUE: LREAL; DIGITS: UINT): STRING	
Преобразование числа типа LREAL в строку	
Аргументы	
VALUE	Преобразуемое значение
DIGITS	Количество знаков после запятой в результате

FORMAT_REAL(VALUE: REAL; DIGITS: UINT): STRING	
Преобразование числа типа REAL в строку	
Аргументы	
VALUE	Преобразуемое значение
DIGITS	Количество знаков после запятой в результате

4.1.1.1.3.2. Целые числа

FORMAT_UINT(VALUE: UINT; WIDTH: INT; FILL_ZERO: BOOL): STRING	
Преобразование числа типа UINT в строку	
Аргументы	
value	Преобразуемое значение
width	Ширина поля (количество символов в результатирующей строке)
Fill_zero	Если 1, то в результатирующей строке перед первой цифрой будут символы ‘0’, в противном случае пробелы

4.1.1.1.3.3. Дата и время

FORMAT_DT(PATTERN: STRING; IN0: DT): STRING	
Форматированное преобразование даты и времени в строку	
Аргументы	
PATTERN	Строка, описывающая формат вывода: d - день в виде одной или двух цифр (1-31);

FORMAT_DT(PATTERN: STRING; IN0: DT): STRING	
	dd - день в виде двух цифр (01-31); M - месяц в виде одной или двух цифр (1-12); MM - месяц в виде двух цифр (01-12); YY - год в виде двух цифр (00-99); YYYY - год в виде четырех цифр; h - часы в виде одной или двух цифр (0-23); hh - часы в виде двух цифр (00-23); m - минуты в виде одной или двух цифр (0-59); mm - минуты в виде двух цифр (00-59); s - секунды в виде одной или двух цифр (0-59); ss - секунды в виде двух цифр (00-59); z - миллисекунды в виде одной, двух или трех цифр (0-999); zzz - миллисекунды в виде трех цифр (000-999); u - микросекунды в виде 0-999999; uuuuuu - микросекунды в виде шести цифр (000000 to 999999)
IN0	Исходное значение
Результат	
Преобразованное значение даты и времени	

FORMAT_DATE(PATTERN: STRING; IN0: DATE): STRING	
Форматированное преобразование даты в строку	
Аргументы	
PATTERN	Строка, описывающая формат вывода: d - день в виде одной или двух цифр (1-31); dd - день в виде двух цифр (01-31); M - месяц в виде одной или двух цифр (1-12); MM - месяц в виде двух цифр (01-12); YY - год в виде двух цифр (00-99); YYYY - год в виде четырех цифр
IN0	Исходное значение
Результат	
Преобразованное значение даты	

FORMAT_TOD(PATTERN: STRING; IN0: TOD): STRING
Форматированное преобразование времени в строку
Аргументы

FORMAT_TOD(PATTERN: STRING; IN0: TOD): STRING	
PATTERN	Строка, описывающая формат вывода: h - часы в виде одной или двух цифр (0-23); hh - часы в виде двух цифр (00-23); m - минуты в виде одной или двух цифр (0-59); mm - минуты в виде двух цифр (00-59); s - секунды в виде одной или двух цифр (0-59); ss - секунды в виде двух цифр (00-59); z - миллисекунды в виде одной, двух или трех цифр (0-999); zzz - миллисекунды в виде трех цифр (000-999); u - микросекунды в виде 0-999999; uuuuuu - микросекунды в виде шести цифр (000000 to 999999)
IN0	Исходное значение
Результат	
Преобразованное значение времени	

4.1.1.2. Арифметические операции и математические функции

4.1.1.2.1. Операции

ADD(IN1, IN2, ..., IN16: ANY_MAGNITUDE): ANY_MAGNITUDE	
Сложение. Входных аргументов может быть от 0 до 16. Типы всех аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
IN1..IN16	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, TIME
Результат	
IN1 + IN2 + … + IN16, тип совпадает с типом аргументов. Выход за границы допустимого диапазона не контролируется	

MUL(IN1, IN2, ..., IN16: ANY_NUM): ANY_NUM
Умножение. Входных аргументов может быть от 0 до 16. Типы всех аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов

MUL(IN1, IN2, ..., IN16: ANY_NUM): ANY_NUM	
Входные переменные	
IN1..IN16	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
Результат	
IN1 * IN2 * … * IN16, тип совпадает с типом аргументов. Выход за границы допустимого диапазона не контролируется	

SUB(IN1, IN2: ANY_MAGNITUDE): ANY_MAGNITUDE	
Вычитание. Типы аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
IN1, IN2	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT, TIME
Результат	
IN1 - IN2, тип совпадает с типом аргументов. Выход за границы допустимого диапазона не контролируется	

DIV(IN1, IN2: ANY_NUM): ANY_NUM	
Деление. Типы аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов. Если входные аргументы целого типа, то результатом вычисления будет целочисленное деление	
Входные переменные	
IN1, IN2	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
Результат	
IN1 / IN2, тип совпадает с типом аргументов. Выход за границы допустимого диапазона не контролируется	

MOD(IN1, IN2: ANY_INT): ANY_INT	
Остаток от деления. Типы аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
IN1, IN2	DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT
Результат	

MOD(IN1, IN2: ANY_INT): ANY_INT
IN1 / IN2, тип совпадает с типом аргументов. Выход за границы допустимого диапазона не контролируется

EXPT(IN1: ANY_REAL; IN2: ANY_NUM): ANY_REAL	
Возведение в степень	
Входные переменные	
IN1, IN2	Возводимое в степень, REAL или LREAL. Степень, DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
Результат	
IN1^IN2, тип совпадает с типом аргумента IN1. Выход за границы допустимого диапазона не контролируется	

MOVE(IN: ANY_ELEMENTARY): ANY_ELEMENTARY	
Копирование входного аргумента	
Входные переменные	
IN	ANY_ELEMENTARY
Результат	
Результат совпадает с входным аргументом	

4.1.1.2.2. Математические функции

ABS(IN: ANY_NUM): ANY_NUM	
Абсолютное значение числа. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	DINT, INT, LINT, LREAL, REAL, SINT, UDINT, UINT, ULINT, USINT
Результат	
Абсолютное значение IN	

SQRT(IN: ANY_REAL): ANY_REAL	
Квадратный корень. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL
Результат	

SQRT(IN: ANY_REAL): ANY_REAL
Квадратный корень входного аргумента. Если аргумент отрицательный, то значение ENO станет ложным, а результат нулевым

LN(IN: ANY_REAL): ANY_REAL	
Натуральный логарифм. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL
Результат	
Натуральный логарифм входного аргумента. Если аргумент нулевой или отрицательный, то значение ENO станет ложным, а результат нулевым	

LOG(IN: ANY_REAL): ANY_REAL	
Десятичный логарифм. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL
Результат	
Десятичный логарифм входного аргумента. Если аргумент нулевой или отрицательный, то значение ENO станет ложным, а результат нулевым	

EXP(IN: ANY_REAL): ANY_REAL	
Экспонента. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL
Результат	
Экспонента входного аргумента (ex). При переполнении значение ENO станет ложным, а результат нулевым	

SIN(IN: ANY_REAL): ANY_REAL	
Синус. Входной аргумент задается в радианах. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL

SIN(IN: ANY_REAL): ANY_REAL
Результат
Синус входного аргумента

COS(IN: ANY_REAL): ANY_REAL	
Косинус. Входной аргумент задается в радианах. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL
Результат	
Косинус входного аргумента	

TAN(IN: ANY_REAL): ANY_REAL	
Тангенс. Входной аргумент задается в радианах. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL
Результат	
Тангенс входного аргумента	

ASIN(IN: ANY_REAL): ANY_REAL	
Арксинус. Входной аргумент должен находиться в диапазоне [-1..+1]. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL
Результат	
Арксинус входного аргумента. Если значение аргумента лежит за пределами диапазона[-1..+1], то значение ENO станет ложным, а результат нулевым	

ACOS(IN: ANY_REAL): ANY_REAL	
Арккосинус. Входной аргумент должен находиться в диапазоне [-1..+1]. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL

ACOS(IN: ANY_REAL): ANY_REAL
Результат
Арккосинус входного аргумента. Если значение аргумента лежит за пределами диапазона[-1..+1], то значение ENO станет ложным, а результат нулевым

ATAN(IN: ANY_REAL): ANY_REAL	
Арктангенс. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	REAL, LREAL
Результат	
Арктангенс входного аргумента	

4.1.1.3. Функции работы с битами

4.1.1.3.1. Битовые операции

AND(IN1, IN2, ..., IN16: ANY_BIT): ANY_BIT	
Побитовое «И». Входных аргументов может быть от 1 до 16. Типы всех аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
IN1..IN16	BOOL, BYTE, WORD, DWORD
Результат	
IN1 & IN2 & … & IN16, тип совпадает с типом аргументов. Если количество аргументов равно нулю, то значение переменной ENO станет ложным, а результат вычисления функции нулевым	

OR(IN1, IN2, ..., IN16: ANY_BIT): ANY_BIT	
Побитовое «ИЛИ». Входных аргументов может быть от 1 до 16. Типы всех аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
IN1..IN16	BOOL, BYTE, WORD, DWORD
Результат	
IN1 OR IN2 OR … OR IN16, тип совпадает с типом аргументов.	

OR(IN1, IN2, ..., IN16: ANY_BIT): ANY_BIT
Если количество аргументов равно нулю, то значение переменной ENO станет ложным, а результат вычисления функции нулевым

XOR(IN1, IN2, ..., IN16: ANY_BIT): ANY_BIT	
Побитовое «Исключающее ИЛИ». Входных аргументов может быть от 1 до 16. Типы всех аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
IN1..IN16	BOOL, BYTE, WORD, DWORD
Результат	
IN1 XOR IN2 XOR … XOR IN16, тип совпадает с типом аргументов. Если количество аргументов равно нулю, то значение переменной ENO станет ложным, а результат вычисления функции нулевым	

NOT(IN: ANY_BIT): ANY_BIT	
Побитовое «НЕ». Тип результата будет совпадать с типом входного аргумента	
Входные переменные	
IN	BOOL, BYTE, WORD, DWORD
Результат	
NOT IN, тип совпадает с типом аргумента	

4.1.1.3.2. Функции сдвига

SHL(IN: ANY_BIT; N: ANY_INT): ANY_BIT	
Сдвиг влево. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	BOOL, BYTE, WORD, DWORD
N	USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Значение аргумента IN, сдвинутое влево на N бит	

SHR(IN: ANY_BIT; N: ANY_INT): ANY_BIT
Сдвиг вправо. Тип результата совпадает с типом входного аргумента

SHR(IN: ANY_BIT; N: ANY_INT): ANY_BIT	
Входные переменные	
IN	BOOL, BYTE, WORD, DWORD
N	USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Значение аргумента IN, сдвинутое вправо на N бит	

ROL(IN: ANY_BIT; N: ANY_INT): ANY_BIT	
Циклический сдвиг влево. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	BOOL, BYTE, WORD, DWORD
N	USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Значение аргумента IN, циклически сдвинутое влево на N бит	

ROR(IN: ANY_BIT; N: ANY_INT): ANY_BIT	
Циклический сдвиг вправо. Тип результата совпадает с типом входного аргумента	
Входные переменные	
IN	BOOL, BYTE, WORD, DWORD
N	USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Значение аргумента IN, циклически сдвинутое вправо на N бит	

4.1.1.3.3. Дополнительные функции

EXTRACT(X: DWORD; N: INT): BOOL	
Возвращает значение бита двоичного представления числа	
Входные переменные	
X	Число, значение бита которого требуется получить
N	Номер бита (0..31)
Результат	
Возвращает TRUE, если значение бита N в двоичном представлении числа X равно 1 и 0 в противном случае	

PUTBIT(X: DWORD; N: INT; B: BOOL): DWORD	
Устанавливает значение бита в двоичном представлении числа	
Входные переменные	
X	Число, значение бита которого требуется получить
N	Номер бита (0..31)
B	Значение бита
Результат	
Число X, в котором бит N в числе X установлен в значение B	

PACK(B0, B1, ..., B7: BOOL): BYTE	
Упаковывает 8 значений типа BOOL в одно значение типа BYTE	
Входные переменные	
B0	Значение разряда 0
...	
B7	Значение разряда 7
Результат	
Число, в котором бит 0 установлен в значение B0, бит 1 в значение B1 и т.д.	

4.1.1.4. Функции выбора

SEL(G: BOOL; IN0, IN1: ANY_ELEMENTARY): ANY_ELEMENTARY	
Выбор одного из двух значений. Типы аргументов IN0, IN1 должны совпадать. Тип результата совпадает с типом аргументов IN0 и IN1	
Входные переменные	
G	Ключ выбора, BOOL
IN0	ANY_ELEMANTARY
IN1	ANY_ELEMANTARY
Результат	
Если значение G ложно, то результатом будет значение IN0. Если значение G истинно, то результатом будет значение IN1	

MAX(IN1, IN2, ..., IN16: ANY_ELEMENTARY): ANY_ELEMENTARY
Выбор максимального значения. Входных аргументов может быть от 1 до 16. Типы всех аргументов должны совпадать.

MAX(IN1, IN2, ..., IN16: ANY_ELEMENTARY): ANY_ELEMENTARY	
Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
IN1..IN16	Значения, среди которых производится выбор
Результат	
Максимальное значение среди значений аргументов IN1..IN16. Если количество аргументов равно нулю, то значение переменной ENO станет ложным, а результат вычисления функции нулевым	

MIN(IN1, IN2, ..., IN16: ANY_ELEMENTARY): ANY_ELEMENTARY	
Выбор минимального значения. Входных аргументов может быть от 1 до 16. Типы всех аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
IN1..IN16	Значения, среди которых производится выбор
Результат	
Минимальное значение среди значений аргументов IN1..IN16. Если количество аргументов равно нулю, то значение переменной ENO станет ложным, а результат вычисления функции нулевым	

LIMIT(MN, IN, MX: ANY_ELEMENTARY): ANY_ELEMENTARY	
Ограничение интервала значений. Типы всех аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов	
Входные переменные	
MN	Минимальное значение диапазона
IN	Входное значение
MX	Максимальное значение диапазона
Результат	
Если IN < MN, то результатом будет значение MN. Если IN > MX, то результатом будет значение MX. Во всех остальных случаях результатом будет значение IN	

MUX(K: ANY_INT; IN1, IN2, ..., IN16: ANY_ELEMENTARY): ANY_ELEMENTARY
Выбор одного из нескольких значений. Входных аргументов может быть от 1 до 16. Типы всех аргументов должны совпадать. Тип результата будет совпадать с типом входных аргументов

MUX(K: ANY_INT; IN1, IN2, ..., IN16: ANY_ELEMENTARY): ANY_ELEMENTARY	
Входные переменные	
K	Ключ выбора
IN1 .. IN16	Значения, среди которых производится выбор
Результат	
Значение аргумента K должно находиться в диапазоне 0..15. В противном случае значение переменной ENO станет ложным, а результат нулевым. Результатом вычисления функции будет значение аргумента INk+1	

4.1.1.5. Функции сравнения

GT(IN1, IN2, ..., IN16: ANY_ELEMENTARY): BOOL	
Больше. Входных аргументов может быть от 2 до 16. Типы всех аргументов должны совпадать. Тип результата BOOL	
Входные переменные	
IN1..IN16	Сравниваемые значения
Результат	
(IN1 > IN2) & (IN2 > IN3) & … & (INn-1 > INn) Если количество аргументов менее 2, то значение переменной ENO станет ложным, а результатом вычисления функции будет FALSE	

GE(IN1, IN2, ..., IN16: ANY_ELEMENTARY): BOOL	
Больше, либо равно. Входных аргументов может быть от 2 до 16. Типы всех аргументов должны совпадать. Тип результата BOOL	
Входные переменные	
IN1..IN16	Сравниваемые значения
Результат	
(IN1 >= IN2) & (IN2 >= IN3) & … & (INn-1 >= INn) Если количество аргументов менее 2, то значение переменной ENO станет ложным, а результатом вычисления функции будет FALSE	

EQ(IN1, IN2, ..., IN16: ANY_ELEMENTARY): BOOL
Равно. Входных аргументов может быть от 2 до 16. Типы всех аргументов должны совпадать. Тип результата BOOL

EQ(IN1, IN2, ..., IN16: ANY_ELEMENTARY): BOOL	
Входные переменные	
IN1..IN16	Сравниваемые значения
Результат	
(IN1 = IN2) & (IN2 = IN3) & … & (INn-1 = INn) Если количество аргументов менее 2, то значение переменной ENO станет ложным, а результатом вычисления функции будет FALSE	

LE(IN1, IN2, ..., IN16: ANY_ELEMENTARY): BOOL	
Меньше, либо равно. Входных аргументов может быть от 2 до 16. Типы всех аргументов должны совпадать. Тип результата BOOL	
Входные переменные	
IN1..IN16	Сравниваемые значения
Результат	
(IN1 <= IN2) & (IN2 <= IN3) & … & (INn-1 <= INn) Если количество аргументов менее 2, то значение переменной ENO станет ложным, а результатом вычисления функции будет FALSE	

LT(IN1, IN2, ..., IN16: ANY_ELEMENTARY): BOOL	
Меньше. Входных аргументов может быть от 2 до 16. Типы всех аргументов должны совпадать. Тип результата BOOL	
Входные переменные	
IN1..IN16	Сравниваемые значения
Результат	
(IN1 < IN2) & (IN2 < IN3) & … & (INn-1 < INn) Если количество аргументов менее 2, то значение переменной ENO станет ложным, а результатом вычисления функции будет FALSE	

NE(IN1, IN2: ANY_ELEMENTARY): BOOL	
Не равно. Типы аргументов должны совпадать. Тип результата BOOL	
Входные переменные	
IN1, IN2	Сравниваемые значения
Результат	
IN1 <> IN2	

4.1.1.6. Функции работы со строками

4.1.1.6.1. Стандартные функции

LEN(IN: STRING): DINT	
Длина строки	
Аргументы	
IN	Строка
Результат	
Длина строки, содержащейся в IN	

LEFT(IN: STRING; L: ANY_INT): STRING	
Подстрока слева	
Аргументы	
IN	Строка
L	Количество извлекаемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
L левых символов строки, содержащейся в IN. Если значение аргумента L превышает длину строки, то результатом будет строка, содержащаяся в IN, целиком. Если значение аргумента L меньше 1, то значение переменной ENO станет ложным, а результатом будет пустая строка	

RIGHT(IN: STRING; L: ANY_INT): STRING	
Подстрока справа	
Аргументы	
IN	Строка
L	Количество извлекаемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
L правых символов строки, содержащейся в IN. Если значение аргумента L превышает длину строки, то результатом будет строка, содержащаяся в IN, целиком. Если значение аргумента L отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

MID(IN: STRING; L, P: ANY_INT): STRING
Подстрока из середины строки
Аргументы

MID(IN: STRING; L, P: ANY_INT): STRING	
IN	Строка
L	Количество извлекаемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
P	Индекс начала подстроки. Индекс первого символа в строке равен 1. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
L символов строки, содержащейся в IN, начиная с позиции P. Если значение аргумента L или P отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

CONCAT(IN1, IN2, ..., IN16: STRING): STRING	
Склейка строк. Входных аргументов может быть от 1 до 16	
Входные переменные	
IN1..IN16	Сцепляемые строки
Результат	
IN1 + IN2 + .. + IN16 Если количество аргументов менее 1, то значение переменной ENO станет ложным, а результатом вычисления функции будет пустая строка	

INSERT(IN1, IN2: STRING; P: ANY_INT): STRING	
Вставка одной строки в другую	
Аргументы	
IN1	Строка, в которую происходит вставка
IN2	Вставляемая строка
P	Индекс вставки. Индекс первого символа в строке равен 1. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Результатом вычисления функции будет строка IN1, в которую вставлена строка IN2, начиная с позиции P. Если значение P превышает длину строки IN1, то строка IN2 будет добавлена к концу строки IN1. Если значение аргумента P отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

DELETE(IN: STRING; L, P: ANY_INT): STRING
Удаление подстроки из строки

DELETE(IN: STRING; L, P: ANY_INT): STRING	
Аргументы	
IN	Строка
L	Количествоудаляемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
P	Индекс первого удаляемого символа . Индекс первого символа в строке равен 1. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Строка IN, из которой удалены L символов, начиная с позиции P. Если значение аргумента L или P отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

REPLACE(IN1, IN2: STRING; L, P: ANY_INT): STRING	
Замена подстроки в строке	
Аргументы	
IN1	Строка, в которой происходит замена
IN2	Вставляемая строка
L	Количество заменяемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
P	Индекс первого заменяемого символа . Индекс первого символа в строке равен 1. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Строка IN1, в которой подстрока длиной L символов, начиная с позиции P, заменена на строку IN2. Если значение аргумента L или P отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

FIND(IN1, IN2: STRING): DINT	
Поиск подстроки в строке	
Аргументы	
IN1	Строка, в которой происходит поиск
IN2	Искомая строка
Результат	
Результатом вычисления функции будет индекс первого вхождения строки IN2 в строку IN1. Индекс первого элемента строки равен 1. Если строка IN2 не найдена в строке IN1, то результатом вычисления функции будет 0	

4.1.1.6.2. Функции работы со строками в формате UTF8

UTF8_LEN(IN: STRING): DINT	
Длина строки	
Аргументы	
IN	Строка (в формате UTF8)
Результат	
Длина строки, содержащейся в IN	

UTF8_LEFT(IN: STRING; L: ANY_INT): STRING	
Подстрока слева	
Аргументы	
IN	Строка (в формате UTF8)
L	Количествоизвлекаемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
L левых символов строки, содержащейся в IN. Если значение аргумента L превышает длину строки, то результатом будет строка, содержащаяся в IN, целиком. Если значение аргумента L меньше 1, то значение переменной ENO станет ложным, а результатом будет пустая строка	

UTF8_RIGHT(IN: STRING; L: ANY_INT): STRING	
Подстрока справа	
Аргументы	
IN	Строка (в формате UTF8)
L	Количество извлекаемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
L правых символов строки, содержащейся в IN. Если значение аргумента L превышает длину строки, то результатом будет строка, содержащаяся в IN, целиком. Если значение аргумента L отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

UTF8_MID(IN: STRING; L, P: ANY_INT): STRING	
Подстрока из середины строки	
Аргументы	
IN	Строка (в формате UTF8)
L	Количество извлекаемых символов.

UTF8_MID(IN: STRING; L, P: ANY_INT): STRING	
	USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
P	Индекс начала подстроки. Индекс первого символа в строке равен 1. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
L символов строки, содержащейся в IN, начиная с позиции P. Если значение аргумента L или P отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

CONCAT(IN1, IN2, ..., IN16: STRING): STRING	
Склейка строк. Входных аргументов может быть от 1 до 16	
Входные переменные	
IN1..IN16	Сцепляемые строки
Результат	
IN1 + IN2 + .. + IN16 Если количество аргументов менее 1, то значение переменной ENO станет ложным, а результатом вычисления функции будет пустая строка	

UTF8_INSERT(IN1, IN2: STRING; P: ANY_INT): STRING	
Вставка одной строки в другую	
Аргументы	
IN1	Строка, в которую происходит вставка (в формате UTF8)
IN2	Вставляемая строка (в формате UTF8)
P	Индекс вставки. Индекс первого символа в строке равен 1. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Результатом вычисления функции будет строка IN1, в которую вставлена строка IN2, начиная с позиции P. Если значение P превышает длину строки IN1, то строка IN2 будет добавлена к концу строки IN1. Если значение аргумента P отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

UTF8_DELETE(IN: STRING; L, P: ANY_INT): STRING
Удаление подстроки из строки
Аргументы

UTF8_DELETE(IN: STRING; L, P: ANY_INT): STRING	
IN	Строка (в формате UTF8)
L	Количествоудаляемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
P	Индекс первого удаляемого символа . Индекс первого символа в строке равен 1. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Строка IN, из которой удалены L символов, начиная с позиции P. Если значение аргумента L или P отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

UTF8_REPLACE(IN1, IN2: STRING; L, P: ANY_INT): STRING	
Замена подстроки в строке	
Аргументы	
IN1	Строка, в которой происходит замена (в формате UTF8)
IN2	Вставляемая строка (в формате UTF8)
L	Количество заменяемых символов. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
P	Индекс первого заменяемого символа . Индекс первого символа в строке равен 1. USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT
Результат	
Строка IN1, в которой подстрока длиной L символов, начиная с позиции P, заменена на строку IN2. Если значение аргумента L или P отрицательно, то значение переменной ENO станет ложным, а результатом будет пустая строка	

UTF8_FIND(IN1, IN2: STRING): DINT	
Поиск подстроки в строке	
Аргументы	
IN1	Строка, в которой происходит поиск (в формате UTF8)
IN2	Искомая строка (в формате UTF8)
Результат	
Результатом вычисления функции будет индекс первого вхождения строки IN2 в строку IN1. Индекс первого элемента строки равен 1. Если строка IN2 не найдена в строке IN1, то результатом вычисления функции будет 0	

4.1.1.6.3. Прочие функции

TRIM(IN: STRING): STRING	
Убирает пробельные символы в начале и конце строки	
Аргументы	
IN	Исходная строка
Результат	
Строка IN, из которой убраны пробельные символы в начале и конце строки	

4.1.1.7. Функции работы с датой и временем

ADD_TIME(IN1, IN2: TIME): TIME	
Сложение интервалов времени	
Аргументы	
IN1	Первый временной интервал
IN2	Второй временной интервал
Результат	
IN1+IN2	

ADD_TOD_TIME(IN1: TOD; IN2: TIME): TOD	
Сложение времени суток и времени	
Аргументы	
IN1	Время суток
IN2	Временной интервал
Результат	
IN1+IN2	

ADD_DT_TIME(IN1: DT; IN2: TIME): DT	
Сложение даты и временного интервала	
Аргументы	
IN1	Дата и время
IN2	Временной интервал
Результат	
IN1+IN2	

SUB_TIME(IN1, IN2: TIME): TIME
Вычитание интервалов времени

SUB_TIME(IN1, IN2: TIME): TIME	
Аргументы	
IN1	Первый временной интервал
IN2	Второй временной интервал
Результат	
IN1-IN2	

SUB_DATE_DATE(IN1, IN2: DATE): TIME	
Вычитание одной даты из другой	
Аргументы	
IN1	Первая дата
IN2	Вторая дата
Результат	
IN1-IN2	

SUB_TOD_TIME(IN1: TOD; IN2: TIME): TOD	
Вычитание интервала времени из времени суток	
Аргументы	
IN1	Время суток
IN2	Интервал времени
Результат	
IN1-IN2	

SUB_TOD_TOD(IN1, IN2: TOD): TIME	
Вычитание одного времени суток из другого	
Аргументы	
IN1	Первое время суток
IN2	Второе время суток
Результат	
IN1-IN2	

SUB_DT_TIME(IN1: DT; IN2: TIME): DT	
Вычитание интервалов времени из даты	
Аргументы	
IN1	Дата и время
IN2	Интервал времени
Результат	
IN1-IN2	

SUB_DT_DT(IN1, IN2: DT): TIME	
Вычитание одной даты из другой	
Аргументы	
IN1	Первые дата и время
IN2	Вторые дата и время
Результат	
IN1-IN2	

MULTIME(IN1: TIME; IN2: ANY_NUM): TIME	
Умножение интервала времени на число	
Аргументы	
IN1	Временной интервал
IN2	USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT, REAL, LREAL
Результат	
IN1*IN2	

DIVTIME(IN1: TIME; IN2: ANY_NUM): TIME	
Деление интервала времени на число. Тип результата TIME	
Аргументы	
IN1	Временной интервал
IN2	USINT, SINT, UINT, INT, UDINT, DINT, ULINT, LINT, REAL, LREAL
Результат	
IN1 / IN2	

CONCAT_DATE_TOD(IN1: DATE; IN2: TOD): DT	
Формирует дату и время из даты и времени суток	
Аргументы	
IN1	Дата
IN2	Время суток
Результат	
Дата и время, в котором дата взята из аргумента IN1, а время - из аргумента IN2	

ENCODE_TIME(DAYS, HOURS, MINS, SECS: LREAL; USECS: UDINT): TIME
Формирование значения продолжительности

ENCODE_TIME(DAYS, HOURS, MINS, SECS: LREAL; USECS: UDINT): TIME	
Аргументы	
DAYS	Дни
HOURS	Часы
MINS	Минуты
SECS	Секунды
USECS	Микросекунды, UDINT
Результат	
Значение продолжительности	

ENCODE_DATE(YEAR, MONTH, DAY: UINT): DATE	
Формирование даты	
Аргументы	
YEAR	Год, UINT
MONTH	Месяц, UINT
DAY	День, UINT
Результат	
Значение даты	

ENCODE_TOD(HOURS, MINS, SECS: UINT; USECS: UDINT): TOD	
Формирование времени суток	
Аргументы	
HOURS	Часы, UINT
MINS	Минуты, UINT
SECS	Секунды, UINT
USECS	Микросекунды, UDINT
Результат	
Значение времени суток	

ENCODE_DT(YEAR, MONTH, DAY: UINT; HOURS, MINS, SECS: UINT; USECS: UDINT): DT	
Формирование даты и времени	
Аргументы	
YEAR	Год, UINT
MONTH	Месяц, UINT
DAY	День, UINT
HOURS	Часы, UINT
MINS	Минуты, UINT

ENCODE_DT(YEAR, MONTH, DAY: UINT; HOURS, MINS, SECS: UINT; USECS: UDINT): DT	
SECS	Секунды, UINT
USECS	Микросекунды, UDINT
Результат	
Значение даты	

CLOCK(): LREAL
Возвращает значение текущей метки времени (в секундах)

NOW(): DT
Возвращает текущую локальную дату и время (откорректированную на смещение временной зоны относительно UTC)

UTC_OFFSET(): TIME
Возвращает смещение текущей временной зоны относительно UTC

TIME_AS_SECONDS(IN: TIME): LREAL
Возвращает длительность временного интервала в секундах

CURRENT_DATE_TIME(): DT
Возвращает текущие дату и время

GET_YEAR(DTV: DT): UINT	
Возвращает год, выделенный из даты и времени	
Аргументы	
date_and_time	Дата и время

GET_MONTH(DTV: DT): UINT	
Возвращает месяц, выделенный из даты и времени	
Аргументы	
date_and_time	Дата и время

GET_DAY(DTV: DT): UINT	
Возвращает день, выделенный из даты и времени	
Аргументы	
date_and_time	Дата и время

GET_HOURS(DTV: DT): UINT	
Возвращает часы, выделенные из даты и времени	
Аргументы	
date_and_time	Дата и время

GET_MINUTES(DTV: DT): UINT	
Возвращает минуты, выделенные из даты и времени	
Аргументы	
date_and_time	Дата и время

GET_SECONDS(DTV: DT): UINT	
Возвращает секунды, выделенные из даты и времени	
Аргументы	
date_and_time	Дата и время

CURRENT_TIMESTAMP(): LREAL
Возвращает значение текущей метки времени (в миллисекундах)

UTC_TO_LOCAL(VALUE: DT): DT	
Преобразование времени UTC (всемирное координированное время) в локальное время	
Аргументы	
value	Преобразуемое значение

LOCAL_TO_UTC(VALUE: DT): DT	
Преобразование локального времени в UTC (всемирное координированное время)	
Аргументы	
value	Преобразуемое значение

4.1.1.8. Функции работы с файловой системой

FILE_OPEN(PATH, MODE: STRING; ERROR: DINT): LINT	
Открывает файл и возвращает его дескриптор. Если файл открыть не удалось, то результатом работы функции будет 0	
Аргументы	
PATH	Путь к файлу
MODE	Режим открытия файла: 'r' - открыть для чтения ('rb', если файл двоичный);

FILE_OPEN(PATH, MODE: STRING; ERROR: DINT): LINT	
	'w' - открыть для записи ('wb', если файл двоичный), ранее существовавший файл при этом стирается; 'a' - открыть для дозаписи ('ab', если файл двоичный), если файла не было, то он создается
ERROR(выход)	Код ошибки, если файл открыть не удалось

FILE_READ(FD: LINT; OUT: ANY_ELEMENTARY; SIZE: DINT; EOF: BOOL): DINT	
Читает данные из файла. Возвращает размер реально прочитанных данных	
Аргументы	
FD (вход)	Дескриптор файла
OUT (выход)	Переменная, в которую будут прочитаны данные
SIZE (вход)	Размер читаемой строки (в байтах), если необходимо прочитать данные типа STRING. Во всех остальных случаях размер читаемых данных определяется реальным типом переменной OUT
EOF (выход)	TRUE, если в ходе чтения достигнут конец файла, иначе FALSE

FILE_READ_LINE(FD: LINT; EOF: BOOL): STRING	
Читает строку из текстового файла, включая символы конца строки	
Аргументы	
FD (вход)	Дескриптор файла
EOF (выход)	TRUE, если в ходе чтения достигнут конец файла, иначе FALSE

FILE_EOF(FD: LINT): BOOL	
Возвращает TRUE, если достигнут конец файла	
Аргументы	
FD	Дескриптор файла

FILE_WRITE(FD: LINT; DATA: ANY_ELEMENTARY): DINT	
Записывает строку в текстовый файл. Возвращает размер реально записанных данных	
Аргументы	
FD (вход)	Дескриптор файла
DATA (вход)	Записываемые данные

FILE_WRITE_LINE(FD: LINT; STR: STRING): BOOL	
Записывает строку в текстовый файл. Возвращает TRUE, если данные успешно записаны	
Аргументы	
FD	Дескриптор файла
STR	записываемые данные (при записи в конец строки дописываются символы конца строки)

FILE_SEEK(FD: LINT; POS: DINT; ORIGIN: INT): BOOL	
Устанавливает внутренний указатель файла в заданное положение. Возвращает TRUE, если операция успешно завершена	
Аргументы	
FD	Дескриптор файла
POS	Положение
ORIGIN	Точка отсчета, от которой отсчитывается положение: 0 - от начала файла; 1 - от текущего положения; 2 - от конца файла.

FILE_FLUSH(FD: LINT): BOOL	
Сбрасывает содержимое внутреннего буфера записи на диск. Возвращает TRUE, если операция успешно завершена	
Аргументы	
FD	Дескриптор файла

FILE_CLOSE(FD: LINT): BOOL	
Закрывает файл. Возвращает TRUE, если файл был успешно закрыт	
Аргументы	
FD	Дескриптор файла

4.1.1.9. Функции работы с аналоговыми сигналами

LINEAR(X, X1, X2, Y1, Y2: REAL; EX: BOOL): REAL	
Линейная интерполяция сигнала	
Аргументы	
X (вход)	Значение сигнала
X1 (вход)	Значение первой точки по оси абсцисс

LINEAR(X, X1, X2, Y1, Y2: REAL; EX: BOOL): REAL	
Y1 (вход)	Значение первой точки по оси ординат
X2 (вход)	Значение второй точки по оси абсцисс
Y2 (вход)	Значение второй точки по оси ординат
EX (вход)	TRUE, если осуществляется экстраполяция при выходе значения переменной X за пределы интервала [X1, X2]

4.1.1.10. Функции среды исполнения

4.1.1.10.1. Функции работы с архивами

GET_ARCHIVE_VALUE(SIG: UDINT; CELL: UINT; SIG_DT: DT; RES: INT): LREAL	
Запрашивает значение сигнала в архиве. Возвращает значение сигнала в виде вещественного числа	
Аргументы	
SIG (вход)	Идентификатор сигнала
CELL (вход)	Номер ячейки сигнала
SIG_DT (вход/выход)	Дата и время сигнала
RES (выход)	Код ошибки ядра (0, если ошибок не было)

4.1.1.10.2. Функции работы с сигналами

FIND_SIGNAL(NAME: STRING; ID: UDINT; CELL: UINT): BOOL	
Ищет сигнал в списке сигналов ядра приложения. Возвращает TRUE, если сигнал удалось найти	
Аргументы	
NAME (вход)	Имя сигнала
ID (выход)	Идентификатор сигнала
CELL (выход)	Номер ячейки сигнала

GET_SIGNAL_DATA(NAME: STRING; VALUE: ANY_ELEMENTARY; FLAGS: WORD; DTV: DT): INT	
Запрашивает значение, флаги и метку времени сигнала в микроядре	
Аргументы	
NAME (вход)	Имя сигнала
VALUE (выход)	Значение сигнала

GET_SIGNAL_DATA(NAME: STRING; VALUE: ANY_ELEMENTARY; FLAGS: WORD; DTV: DT): INT	
FLAGS (выход)	Флаги сигнала
DTV (выход)	Метка времени сигнала (UTC)

SET_SIGNAL_DATA(NAME: STRING; VALUE: ANY_ELEMENTARY; FLAGS: WORD; MASK: WORD; DTV: DT): INT	
Записывает значение, флаги (разрешённые маской) и метку времени сигнала в микроядро * данную функцию на данный момент нельзя использовать в базовых блоках	
Аргументы	
NAME (вход)	Тип STRING. Имя сигнала
VALUE (вход)	Тип ANY_ELEMENTARY. Значение сигнала
FLAGS (вход)	Тип WORD. Флаги сигнала
MASK (вход)	Тип WORD. Маска сигнала
DTV (вход)	Тип DT. Метка времени сигнала (UTC). Если метка имеет нулевое значение, то сигнал сохранится с текущей меткой времени

4.1.1.10.3. Функции работы с приложениями

CORE_AVAILABLE(CORE: STRING): BOOL	
Возвращает TRUE, если ядро-партнер доступно	
Аргументы	
CORE	Имя ядра-партнера

CORE_LATENCY(CORE: STRING): TIME	
Возвращает среднее время доступа к ядру-партнеру или -1, если ядро-партнер недоступно	
Аргументы	
CORE	Имя ядра-партнера

CORE_LATENCY_DETAILED(CORE: STRING, INDEX: INT): TIME	
Возвращает среднее время доступа к ядру-партнеру по указанному сетевому интерфейсу или -1, если ядро-партнер недоступно	
Аргументы	
CORE	Имя ядра-партнера
INDEX	Индекс сетевого интерфейса. Если значение индекса равно -1, то функция вернет наименьшее время среди всех сетевых

CORE_LATENCY_DETAILED(CORE: STRING, INDEX: INT): TIME	
	интерфейсов. Если значение 0 и больше, то функция вернет время для указанного сетевого интерфейса

ADD_LOG_MSG(MSG: STRING): BOOL	
Добавляет строку в log-файл текущего приложения	
Аргументы	
MSG	Записываемая строка

4.1.1.10.4. Функции работы с событиями и тревогами

CREATE_EVENT(MESSAGE: STRING; CATEGORY: DINT; STATE: INT; SOURCE, ALARM_ID: STRING; USER_DT: DT; TAG: STRING): BOOL	
Добавляет событие	
Аргументы	
MESSAGE	Сообщение события
CATEGORY	Категория события
STATE	Состояние события
SOURCE	Источник события
ALARM_ID	Идентификатор тревоги
USER_DT	Пользовательские дата и время события
TAG	Дополнительные данные события

ADD_INFO(TEXT: STRING): BOOL	
Формирует информационное событие системы	
Аргументы	
text	Текст события

ADD_WARNING(TEXT: STRING): BOOL	
Формирует событие - предупреждение	
Аргументы	
text	Текст события

ADD_CRITICAL(TEXT: STRING): BOOL
Формирует критическое событие системы
Аргументы

ADD_CRITICAL(TEXT: STRING): BOOL	
text	Текст события

ADD_EVENT(CATEGORY: DINT; SOURCE, MESSAGE, TAG: STRING; STATE: INT): BOOL	
Формирует критическое событие системы	
Аргументы	
category	Категория события (0..9999)
source	Источник события
message	Сообщение события
tag	Дополнительная информация события
state	Состояние события (0 - исчезновение, 1 - появление)

4.1.1.11. Прочие функции

SYSEXEC(CMD: STRING): DINT	
Запускает системную команду и ждет завершения ее выполнения. Возвращает Id порожденного процесса	
Аргументы	
cmd	Запускаемая команда

SYSSPAWN(CMD: STRING): DINT	
Запускает системную команду и завершается, не ожидая завершения выполнения команды. Возвращает Id порожденного процесса	
Аргументы	
cmd	Запускаемая команда

4.1.2. Функции графических приложений

4.1.2.1. Функции среды исполнения

4.1.2.1.1. Безопасность и работа с пользователями

CHECK_RIGHTS(DATA: STRING): BOOL
Проверяет права текущего пользователя.

CHECK_RIGHTS(DATA: STRING): BOOL	
Возвращает TRUE, если строка data есть среди прав пользователя	
Аргументы	
data	Проверяемая строка

LOGOUT(): BOOL
Принудительно завершает работу пользователя в системе

USER_NAME(): STRING
Возвращает текущее имя пользователя, прошедшего регистрацию в системе

USER_LOG_ON(): BOOL
Возвращает TRUE, если какой-либо пользователь зарегистрировался в системе

5. ПОЛЕЗНЫЕ ПРИМЕРЫ

5.1. Реализация работы с событиями в Lua алгоритмах

В данном примере будер рассмотрен Lua алгоритм, в котором описана реализация работы с событиями.

Создадим приложение с типом APPLICATION.LUA, назовём его EventsLUA и создадим в интерфейсе четыре переменные: LSignal1, LSignal2, LSignal3 и LSignal4, тип для переменных выберем INT (см. рис. 5.1).

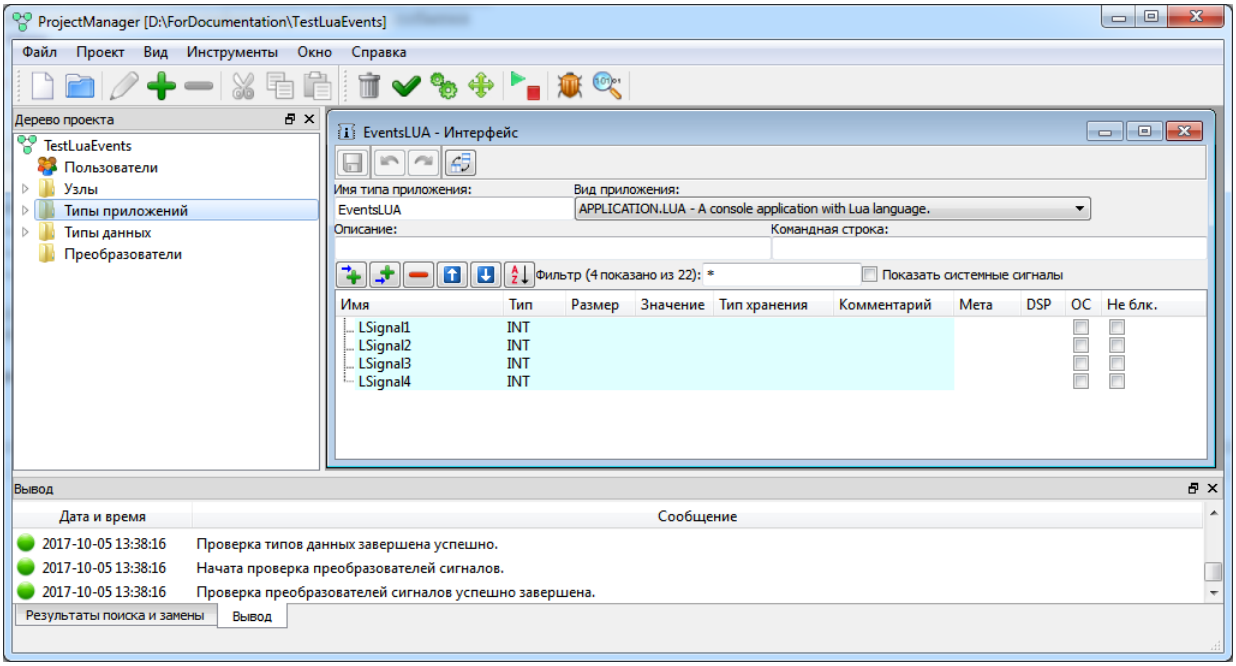
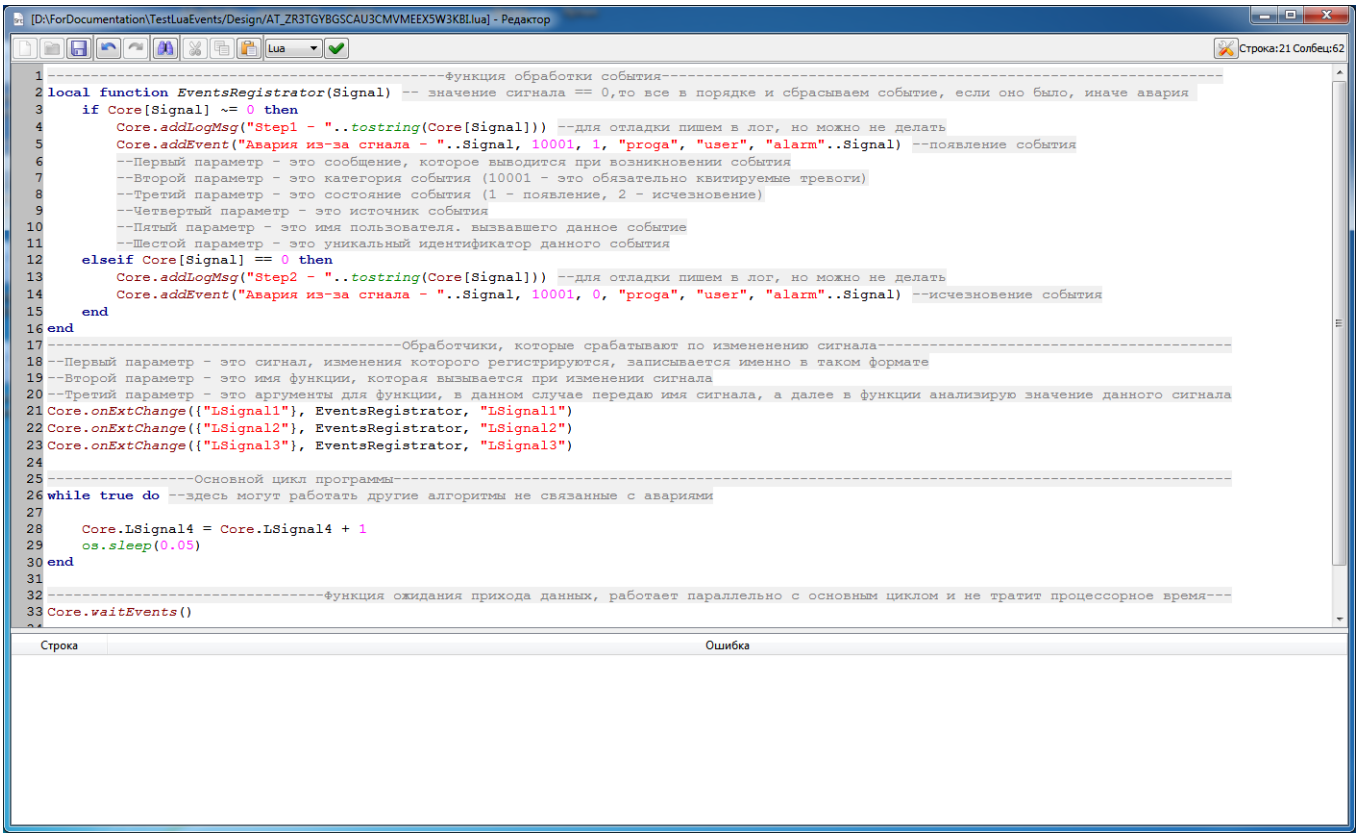


Рисунок 5.1 - Интерфейс нашего приложения APPLICATION.LUA

Далее напишем следующий код для этого приложения (см. рис. 5.2).




```lua
----------------------------------------функция обработки события----------------------------------------------------------------
local function EventsRegistrator(Signal) -- значение сигнала == 0,то все в порядке и сбрасываем событие, если оно было, иначе авария
    if Core[Signal] ~= 0 then
        Core.addLogMsg("Step1 - "..tostring(Core[Signal])) --для отладки пишем в лог, но можно не делать
        Core.addEvent("Авария из-за сгнала - "..Signal, 10001, 1, "proga", "user", "alarm"..Signal) --появление события
        --Первый параметр - это сообщение, которое выводится при возникновении события
        --Второй параметр - это категория события (10001 - это обязательно квитируемые тревоги)
        --Третий параметр - это состояние события (1 - появление, 2 - исчезновение)
        --Четвертый параметр - это источник события
        --Пятый параметр - это имя пользователя. вызвавшего данное событие
        --Шестой параметр - это уникальный идентификатор данного события
    elseif Core[Signal] == 0 then
        Core.addLogMsg("Step2 - "..tostring(Core[Signal])) --для отладки пишем в лог, но можно не делать
        Core.addEvent("Авария из-за сгнала - "..Signal, 10001, 0, "proga", "user", "alarm"..Signal) --исчезновение события
    end
end
----------------------------------------Обработчики, которые срабатывают по изменению сигнала-----------------------------------
--Первый параметр - это сигнал, изменения которого регистрируются, записывается именно в таком формате
--Второй параметр - это имя функции, которая вызывается при изменении сигнала
--Третий параметр - это аргументы для функции, в данном случае передаю имя сигнала, а далее в функции анализирую значение данного сигнала
Core.onExtChange({"LSignal1"}, EventsRegistrator, "LSignal1")
Core.onExtChange({"LSignal2"}, EventsRegistrator, "LSignal2")
Core.onExtChange({"LSignal3"}, EventsRegistrator, "LSignal3")

----------------Основной цикл программы-----------------------------------------------------------------------------------------
while true do --здесь могут работать другие алгоритмы не связанные с авариями

    Core.LSignal4 = Core.LSignal4 + 1
    os.sleep(0.05)
end

--------------------------------функция ожидания прихода данных, работает параллельно с основным циклом и не тратит процессорное время---
Core.waitEvents()
```

Рисунок 5.2 - Алгоритм реализации работы с событиями в Lua

Данный алгоритм включает событийную и циклическую модели работы.

Событийная модель реализована пользовательской функцией **EventsRegistartor** и стандартными функциями **Core.onExtChange** и **Core.waitEvents**. Функция **Core.waitEvents()** нужна для регистрации всех событий в системе и работает параллельно с основным циклом программы. Её местоположение в коде должно быть именно после основного цикла. Далее функция **Core.onExtChange({"LSignal1"}, EventsRegistrator, "LSignal1")** срабатывает по изменению сигнала, указанного первым параметром, т.е. LSignal1, по его изменению срабатывает функция, указанная вторым параметром, т.е. EventsRegistrator и в третьем параметре передаются для функции EventsRegistrator её аргументы, в данном случае имя сигнала LSignal1(тип переданного значения аргумента String). Функция **EventsRegistrator(Signal)**, получает имя изменившегося сигнала и далее в условиях смотрит чему равно текущее значение. Если текущее значение 1, то регистрирует появление события, если 0, то регистрирует исчезновение события. Аналогично работают **Core.onExtChange({"LSignal2"}, EventsRegistrator, "LSignal2")** и **Core.onExtChange({"LSignal3"}, EventsRegistrator, "LSignal3")**. Таких функций может быть большое количество и большим преимуществом является то, что они срабатывают только при изменении своего сигнала.

Циклическая модель реализована бесконечным циклом **while**. В данном примере в нём каждый цикл увеличивается на 1 значение сигнала Core.LSignal4.

Далее, чтобы проверить работу данного алгоритма, создайте приложение с типом APPLICATION.IEC.WINDOW(графическое приложение) и создайте следующий алгоритм из функциональных блоков EventViewer, TButton и AlarmViewer (см. рис. 5.3).

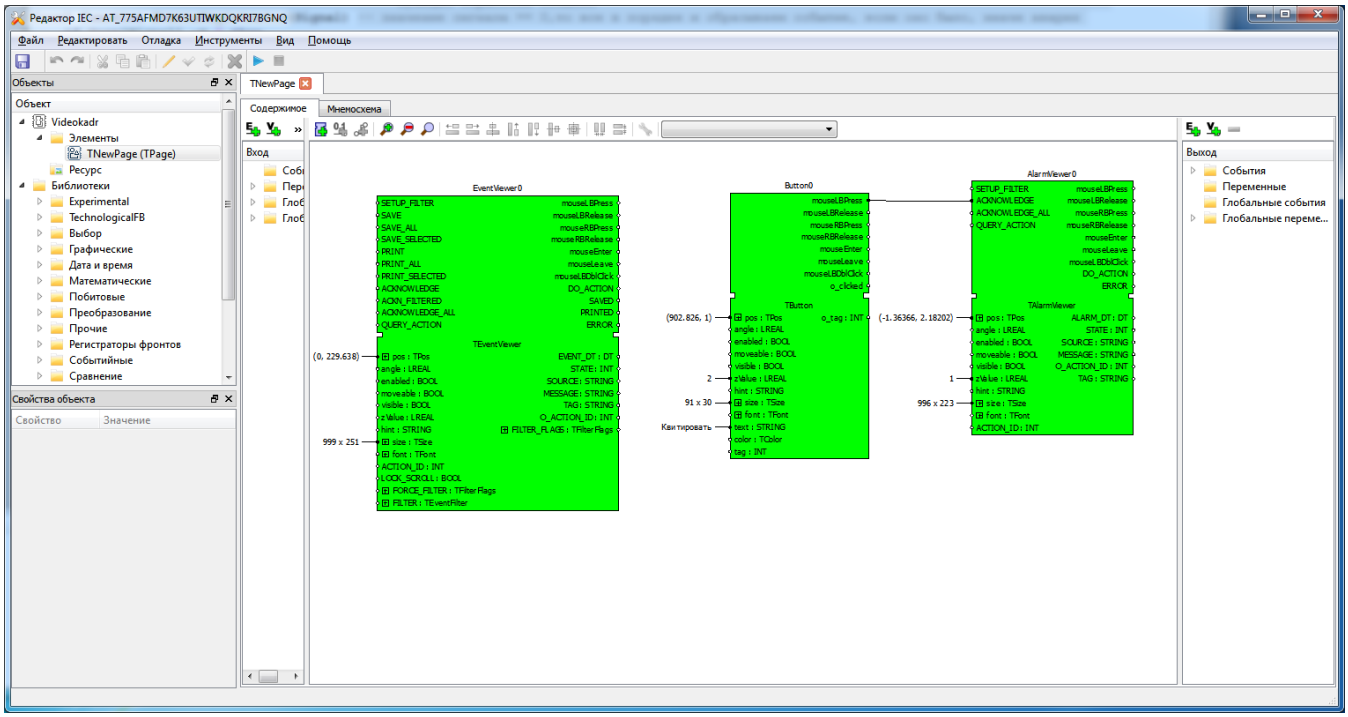


Рисунок 5.3 - Алгоритм видеокадра

Графически этот алгоритм выглядит следующим образом (см. рис. 5.4).

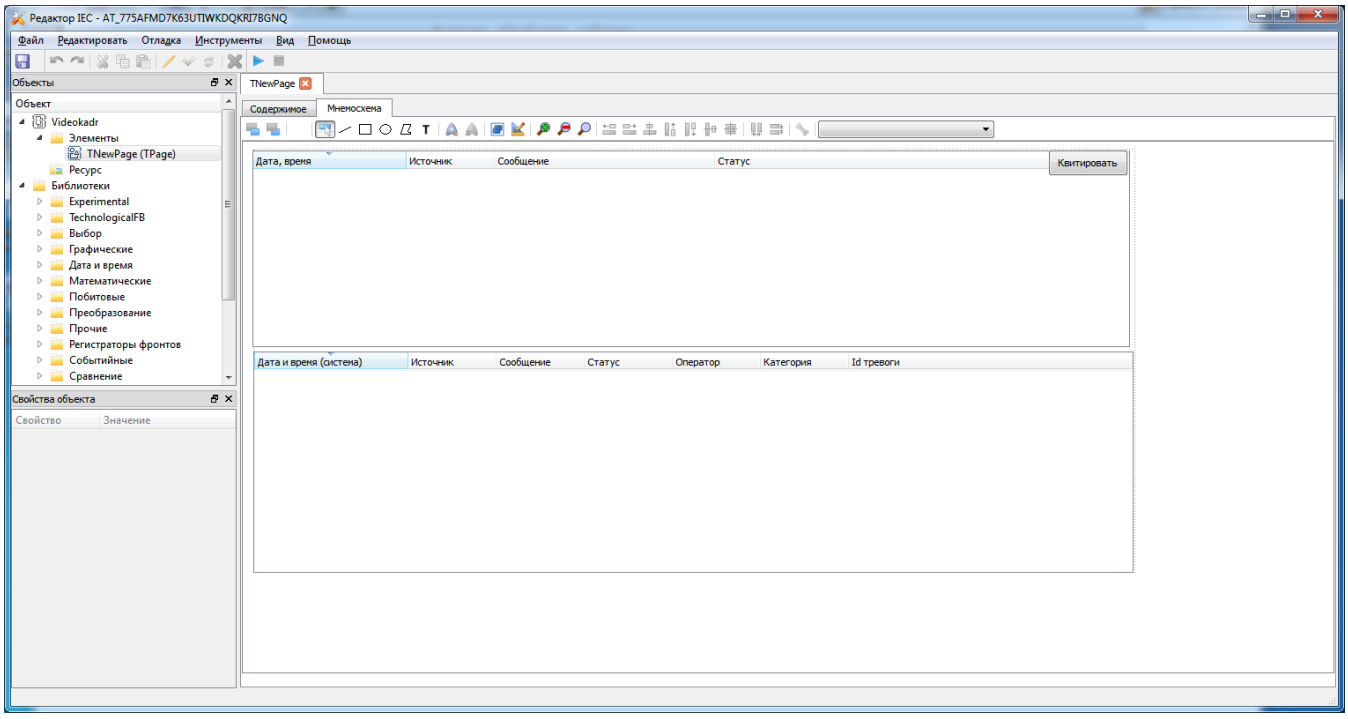


Рисунок 5.4 - Графическое представление видеокадра

Далее создадим узел, на котором будем запускать проект, и добавим в него необходимые приложения (см. рис. 5.5). Обязательно длля работы с событиями создайте и добавьте приложение с типом EVENT LOGGER, это приложение хранящее все события в системе.

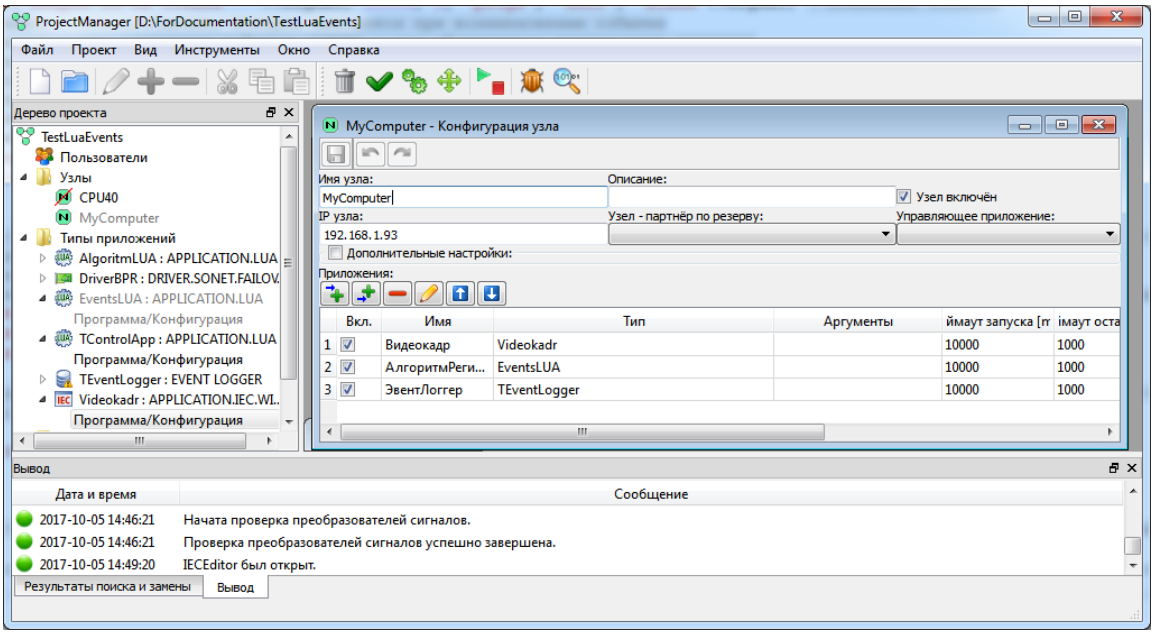


Рисунок 5.5 - Узел нашего проекта

Скомпилируйте проект, разошлите и запускайте. С помощью программы SIGNAL VIEWER (Просмотрщик сигналов) выберите у приложения EventsLUA сигналы LSignal1, LSignal2, LSignal3 и LSignal4 (см. рис. 5.6).

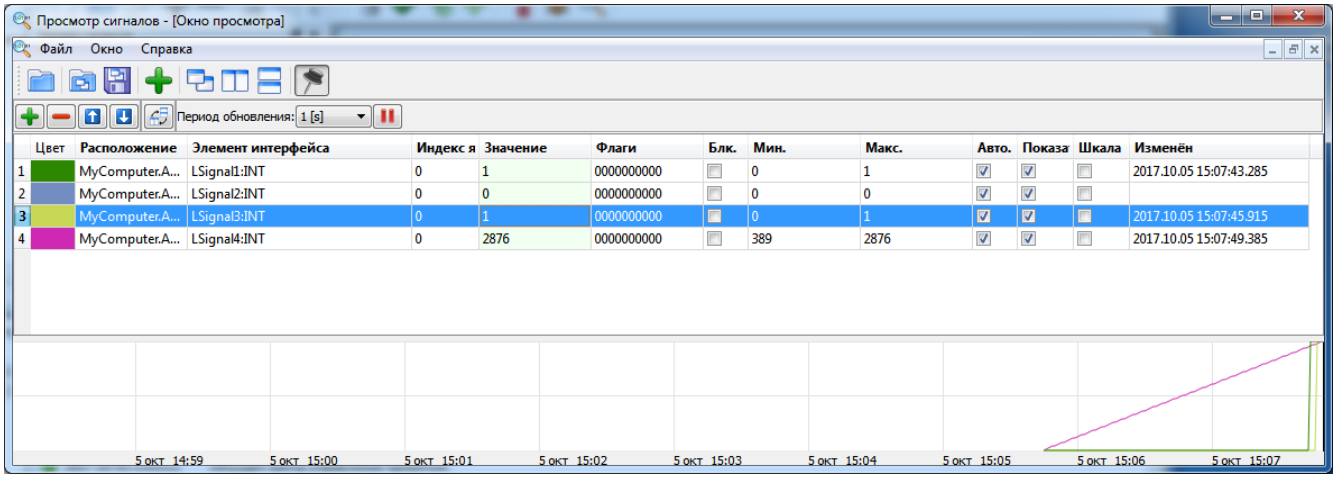


Рисунок 5.6 - Просмотрщик сигналов

В Просмотрщике сигналов будет видно, что сигнал LSignal4 постоянно увеличивается, что говорит о работе основного цикла программы и, если изменять значения других сигналов, то на видеокадре будут отображаться соответствующие события (см. рис. 5.7).

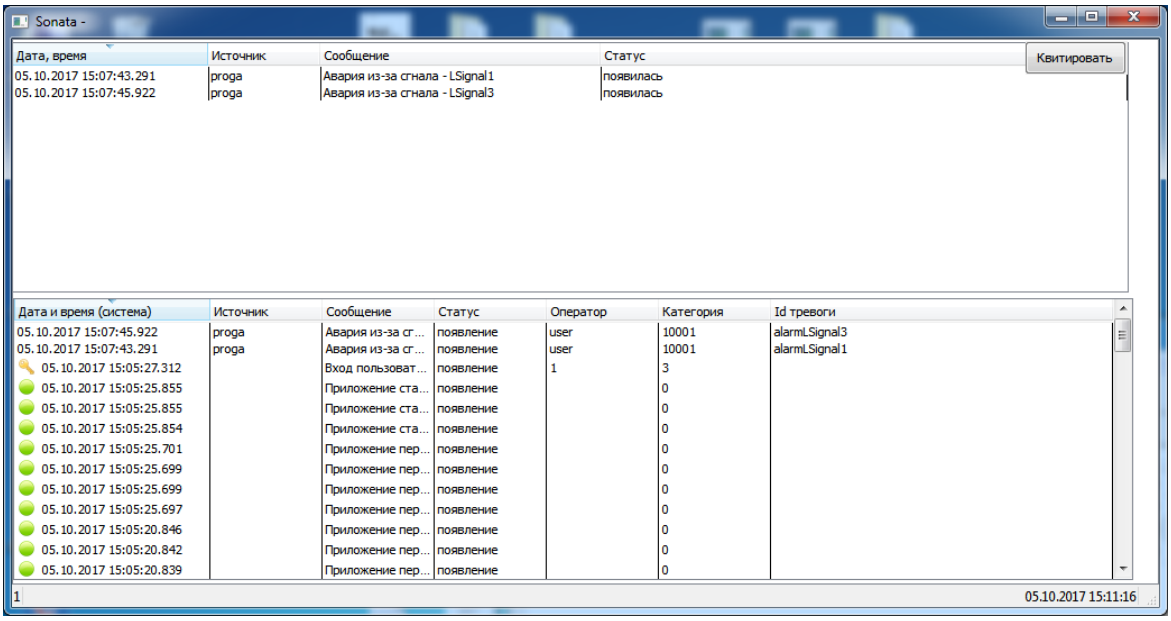


Рисунок 5.7 - Работающий видеокадр

5.2. Как в Lua узнать работает другой узел или остановлен

Состояние узла можно узнать по системному сигналу @NODE_RUN (тип BOOLEAN) в приложении Loader, которое присутствует на каждом узле. Его значение будет равно TRUE, если все приложения на узле запустились без ошибок (отображаются зеленым кружочком в Центре управления) или могут быть запущенные приложения с ошибками (отображаются жёлтым кружочком в Центре управления). В случае, если узел остановлен или есть приложения, которые не запустились на узле (отображаются красным кружочком в Центре управления), значение будет FALSE.

В приложении Lua можно написать строку следующего типа:

local result, value = Core.directGet("Имя_узла.Loader", 0.01, "@NODE_RUN", 0)

В локальную переменную **value** будет помещено значение системной переменной @NODE_RUN с узла Имя_узла.Loader. На основе этих данных можно писать свой алгоритм.

Более подробно о команде **Core.directGet** прочитайте в её описании.

ПРИЛОЖЕНИЕ А

КОДЫ ОШИБОК, ИСПОЛЬЗУЕМЫЕ ДЛЯ ДИАГНОСТИКИ В СКАДА-СИСТЕМЕ "СОНАТА"

Данные коды ошибок и сообщения используются при анализе логов работающих или неработающих приложений и в функциях языка **LUA**:

0 - ошибок нет;
1 - неизвестная ошибка;
2 - неверный аргумент функции;
3 - не удалось запустить или остановить служебный поток;
4 - не удалось инициализировать сокет;
5 - ядро уже запущено. Данная операция не может быть выполнена;
6 - тайм-аут ожидания;
7 - не найден преобразователь сигнала;
8 - такой преобразователь сигнала уже существует;
9 - преобразователь сигнала используется и не может быть изменён или удалён;
10 - запрашиваемое ядро-партнёр не найдено;
11 - описание ядра-партнёр уже существует;
12 - ядро-партнёр используется и не может быть изменено или удалено;
13 - потеряна связь с ядром-партнёром;
14 - восстановлена связь с ядром-партнёром;
15 - синхронизированы значения сигналов с ядром-партнёром;
16 - передача значения сигнала с узла наружу заблокирована;
17 - запрашиваемый сигнал не найден;
18 - выход за границы массива сигнала;
19 - такой сигнал уже существует;
20 - невозможно преобразовать значение сигнала. Ошибка в DSP;
22 - отсутствие значения сигнала;
23 - несовпадение типа сигнала с типом запрашиваемого значения;
24 - выход за допустимый диапазон дат.

ПРИЛОЖЕНИЕ В

Таблица В.1 - Флаги сигнала

Бит	**Описание**
0-9	Пользовательские флаги сигнала
10-12	Не используются (зарезервированы)
13	Флаг невалидности (недостоверности) сигнала
14	Изменение сигнала заблокировано (значение сигнала изменить нельзя)
15	Игнорировать изменение флагов

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в докум.	№ документа	Входящий № сопр. документа и дата	Подп.	Дата
Изм.	измененных	замененных	новых	аннулированных					
1		все			320	35001-19			02.19