

ТОPAZ SCRIPT EDITOR
РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Москва 2020

СОДЕРЖАНИЕ

1. О ПРОГРАММЕ	3
1.1 Основные функции программного компонента.....	3
1.2 Установка на персональный компьютер.....	3
2. ОСНОВНЫЕ ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ	4
3. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС	5
3.1 Начало работы.....	5
3.2 Элементы главного окна	5
3.3 Открытие и создание библиотеки	7
3.4 Создание новой пользовательской функции	7
3.4.1 Аргументы функции.....	9
3.5 Наполнение функции	10
3.6 Построение библиотеки	10
4. ПЕРЕМЕННЫЕ LUA	11
5. СИСТЕМНЫЕ ДОПОЛНЕНИЯ DAS	12
5.1 Общие функции работы с DAS	12
5.2 Функции чтения/записи элементов базы текущих значений.....	12
5.3 Функции работы с флагами параметра.....	13
5.4 Функции блокировки.....	14
5.5 Функции работы с буфером событий	15
5.6 Функции отладки	15
5.7 Функции работы с временем	16
6. ПРИКЛАДНЫЕ БИБЛИОТЕКИ	17
6.1 Системная библиотека прикладных задач DasComponents	17
6.1.1 Функция <i>Shutdown</i>	17
6.2 Библиотека функций для решения типовых задач DasUtil.....	17
6.2.1 Функция <i>OR</i>	17
6.2.2 Функция <i>AND</i>	18
6.2.3 Функция <i>SDPS</i>	18
6.2.4 Функция <i>UDPS</i>	18
6.2.5 Функция <i>MEAN3</i>	19
6.2.6 Функция <i>BITSPLITC2D</i>	19
6.2.7 Функция <i>OREVT</i>	19
6.2.8 Функция <i>ANDEVT</i>	20
ПРИЛОЖЕНИЕ 1	21

1. О программе

Программный компонент TOPAZ SCRIPT EDITOR является составной частью программного комплекса TOPAZ TMBuilder и предназначен для реализации возможности дополнительной обработки телемеханических сигналов путем исполнения алгоритмов пользователей (пользовательских сценариев). Функции разрабатываются на скриптовом языке программирования LUA (www.lua.org)

Программный компонент TOPAZ SCRIPT EDITOR может работать как в составе ПО TOPAZ TMBuilder, так и самостоятельно.

1.1 Основные функции программного компонента

- Реализация программного доступа к данным базы текущих значений IEC DAS
- Создание пользовательских сценариев для обработки значений телемеханических параметров
- Организация пользовательских сценариев в библиотеки.
- Интеграция с ПО TOPAZ TMBuilder для возможности загрузки и выполнения пользовательских функций в контроллерах под управление ПО TOPAZ DAS

1.2 Установка на персональный компьютер

Для установки TOPAZ SCRIPT EDITOR на персональный компьютер необходимо запустить инсталляционный файл **TOPAZ_Script_Editor_Installer_X.X.X.X.exe**, где X.X.X.X - текущая версия дистрибутива.

2. Основные термины и определения

DAS - Data Access Server - Программно-технический комплекс, включающий в себя промышленный контроллер под управлением ОС Линукс и специализированное ПО для выполнения сбора и обработки телемеханической информации.

Пользовательская функция (сценарий, прикладная задача) - законченная часть программы, имеющая уникальное (в пределах библиотеки имя) и один или более входных параметров. Пользовательская функция может исполняться как самостоятельная единица или может быть вызвана из других функций.

Библиотека - Организованный набор пользовательских функций, имеющий уникальное имя в пределах проекта TOPAZ TMBuilder. Библиотека - минимальная единица, загружаемая для исполнения в контроллер.

Исполняемый модуль - программный модуль, загружаемый в контроллер. Является подгружаемой библиотекой основного процесса iec-controls. Реализует интерпретатор сценариев языка LUA

База текущих значений DAS (база данных DAS) - Область памяти содержащая текущие значения телемеханических параметров с меткой времени последнего изменения и дополнительными флагами. Состоит из трех таблиц, по одной для каждого типа параметров:

- **дискреты** - целочисленные данные, значения от 0 до 8191 (13 бит);
- **аналоги** - данные с плавающей запятой;
- **счетчики** - целочисленные данные, значения от 0 до 4294967295 (32 бита);

Размер каждой таблицы - 65535 записей.

Каждый параметр имеет так же флаги: *недостоверность, динамика, блокировка, замещение*. Аналоги так же имеют флаг *переполнения*.

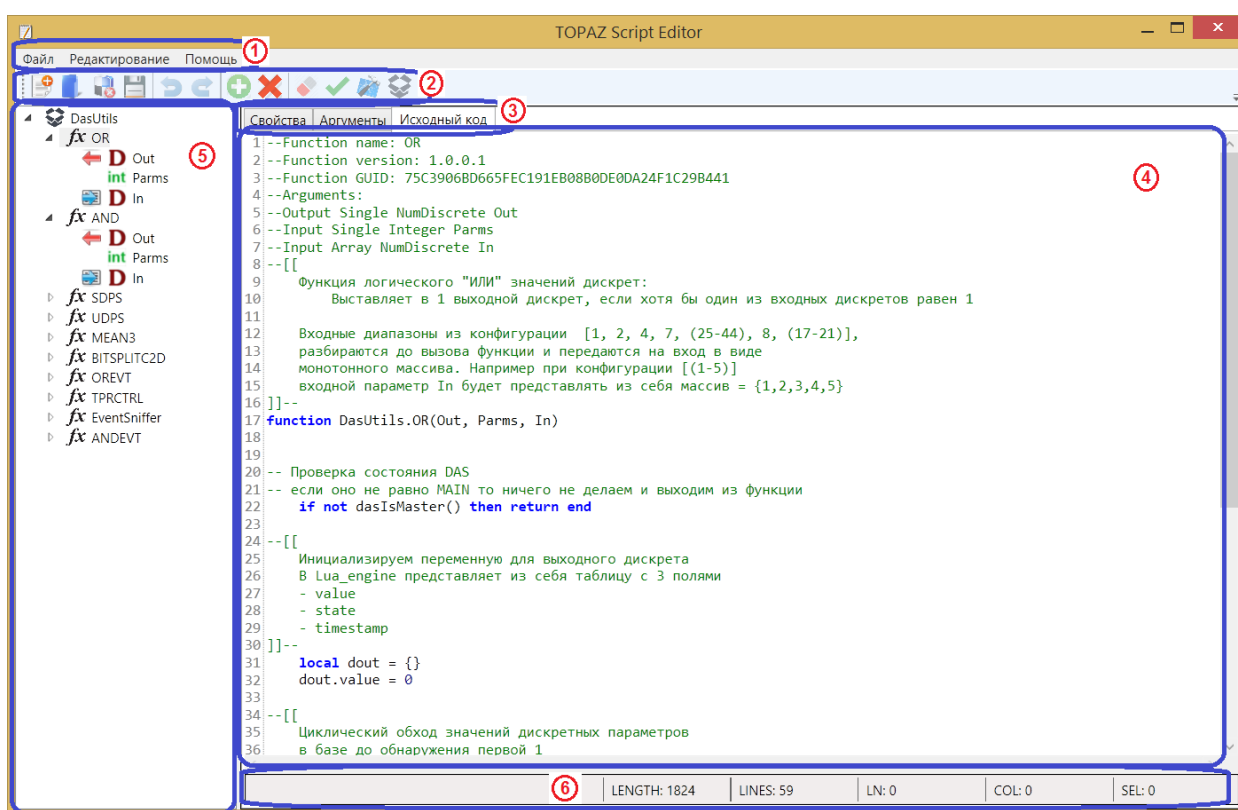
3. Пользовательский интерфейс

3.1 Начало работы

TOPAZ SCRIPT EDITOR может запускаться как самостоятельное приложение или вызываться из TOPAZ TMBUILDER.













При запуске TOPAZ SCRIPT EDITOR как самостоятельного приложения, необходимо открыть заранее сохраненную библиотеку или создать новую. При запуске из TOPAZ TMBUILDER'a автоматически откроется указанная библиотека конкретного проекта.

3.2 Элементы главного окна



1. Панель меню - служит для доступа ко всем функциям. Содержит следующие элементы:

Пункт меню	Подпункт меню	Описание
Файл	Новая библиотека (Ctrl-N)	Вызвать диалоговое окно создания новой библиотеки
	Открыть библиотеку (Ctrl-O)	Вызвать диалоговое окно для открытия библиотеки
	Закреть библиотеку	Закреть текущую библиотеку
	Сохранить библиотеку (Ctrl-S)	Сохранить текущую библиотеку
	Открыть сессию	Открыть ранее сохраненную


Пункт меню	Подпункт меню	Описание
		сессию
	Закреть сессию	Закреть ранее сохраненную сессию
	Недавние файлы	Открыть список ранее сохраненных файлов
	Выход 	Выход из TOPAZ SCRIPT EDITOR
Редактирование	Отмена 	Отмена последней операции
	Повтор 	Повтор последней операции
	Вырезать	Вырезать выделенный фрагмент
	Копировать	Копировать выделенный фрагмент
	Вставить	Вставить выделенный фрагмент
	Удалить	Удалить выделенный фрагмент
	Выделить все	Выделить все
	Добавить функцию 	Вызов мастера создания новой пользовательской функции
	Удалить функцию 	Удаление функции из библиотеки
	Обновить заголовок функции	Обновить заголовок функции
	Очистить 	Очистить каталог вывода библиотеки
	Проверить 	Проверить библиотеку
	Построить библиотеку 	Построение библиотеки
	Открыть папку сборки 	Открыть папку с построенной библиотекой
	Схема библиотеки 	Отображает/скрывает панель схемы библиотеки
Помощь	Справка 	Открыть справку по LUA
	О приложении 	Вызов окна "О программе"

2. *Панель инструментов* - содержит элементы для доступа к часто используемым операциям из панели меню
3. *Закладки редактора* - Закладки переключают режим окна редактора функции:
 - Свойства - режим редактирования общих свойств функции
 - Аргументы - режим редактирования аргументов функции
 - Исходный код - режим редактирования исходного кода функции
4. *Окно редактора*
5. *Структура проекта* - отображает структуру текущего проекта в виде дерева: Библиотеки, функции, аргументы функций.
6. *Статусная строка*

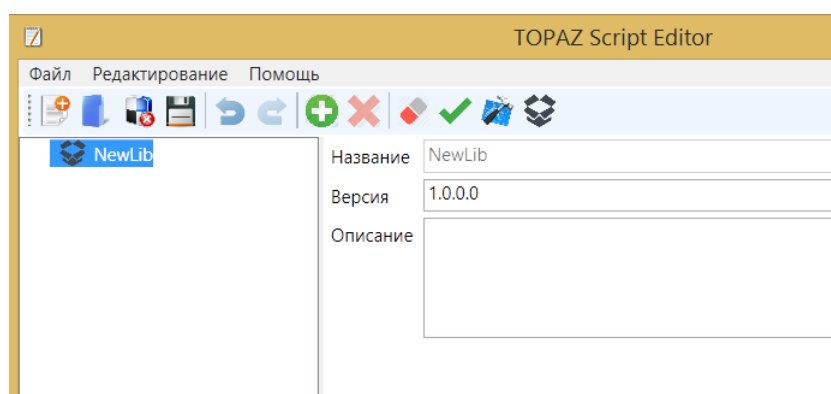
3.3 Открытие и создание библиотеки

Если TOPAZ SCRIPT EDITOR вызывается из TOPAZ TMBuilder, то никаких дополнительных действий делать не надо, будет автоматически открыта библиотека выбранная в TOPAZ TMBuilder. (см. руководство пользователя TOPAZ TMBuilder, часть 2.2, пункт 3.8)


Если TOPAZ SCRIPT EDITOR запускается как самостоятельное приложение, то становится доступна возможность создания и загрузки ранее сохраненной библиотеки.

Для создания новой библиотеки надо нажать кнопку  на панели инструментов или выбрать меню "Файл"-"Новая библиотека". Откроется диалоговое окно выбора названия и расположения создаваемой библиотеки. Имя библиотеки, заданное при создании, изменить потом нельзя.

После этого в дереве проекта появится созданная библиотека, щелкнув на нее, можно перейти к редактированию ее свойств:




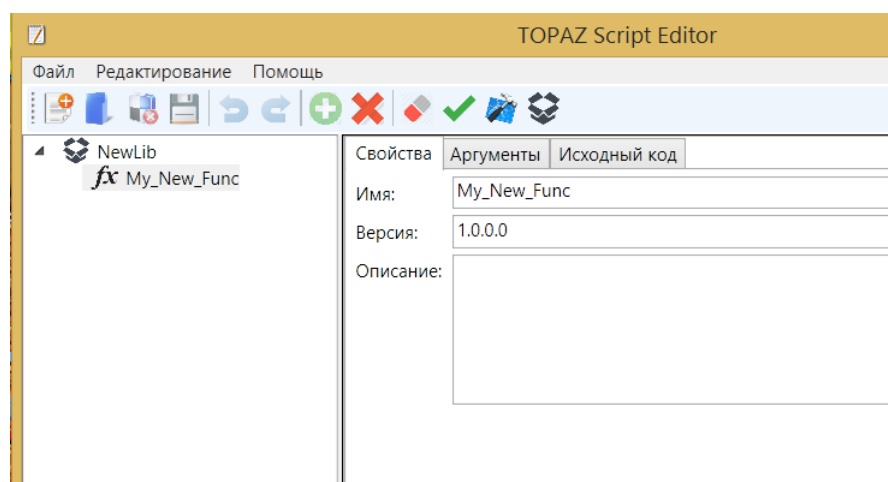
Здесь можно изменить версию библиотеки и задать ее описание.

Если библиотека была создана ранее и сохранена на диске, ее можно открыть, нажав на кнопку  на панели инструментов или выбрать меню "Файл"-"Открыть библиотеку" и указать путь к рабочему каталогу библиотеки.

Во время работы можно создавать или открывать несколько библиотек одновременно.

3.4 Создание новой пользовательской функции

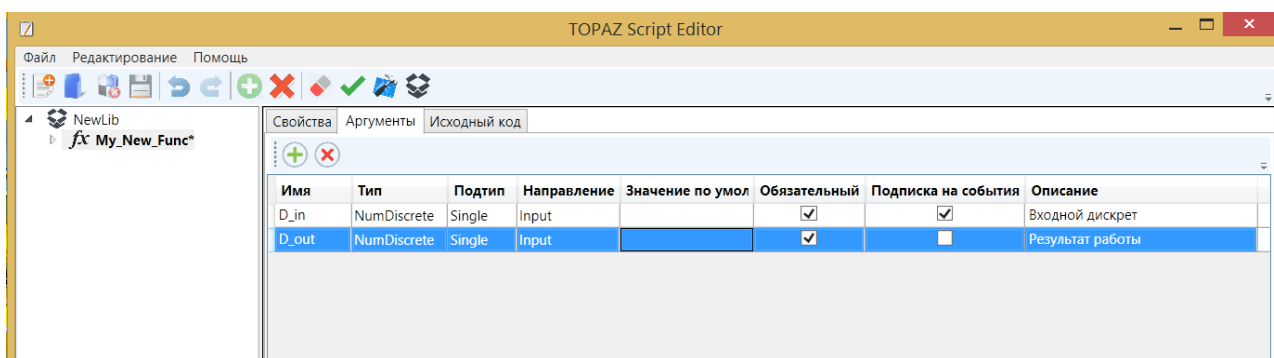
Для создания новой пользовательской функции надо нажать кнопку  на панели инструментов или выбрать соответствующий пункт меню. После этого в текущей библиотеке будет создана новая функция. Щелкнув на нее, ее свойства будут доступны в окне редактора.



На закладке "Свойства" надо задать основные свойства функции:

- **Имя** - оно должно быть уникально для данной библиотеки и состоять только из латинских букв, цифр и знака подчеркивания
- **Версия** - Номер версии функции
- **Описание** - подробное описание функции (не обязательно)

На закладке "Аргументы" можно добавлять/редактировать аргументы функции:



Кнопки соответственно добавляют и удаляют аргументы

Аргументы имеют следующие свойства (подробное описание см. ниже):

- **Имя** - уникальное для данной функции имя аргумента (регистрозависимое, должно состоять только из латинских букв, цифр и знака подчеркивания)
- **Тип** - Тип данных данного аргумента
- **Подтип** аргумента (Argument Subtype) - может принимать значения: Single - одиночный элемент; Array - список или диапазон элементов; Unknown - незадаанный тип.
- **Направление** потока данных - может принимать значения: **Input** - входные данные; **Output** - выходные данные; **Unknown** - незадаанный.
- **Обязательный** аргумент - если этот чекбокс снят, то данный аргумент считается необязательным.
- **Подписка на события** - имеет смысл, если данный аргумент описывает номер в базе DAS. Если чекбокс отмечен, то значения дискрета вычитывается из очереди событий, иначе происходит циклический опрос значения дискрета.
- **Значение по умолчанию (необязательно)** - значение, которым будет проинициализирован данный аргумент.

- **Описание аргумента (необязательно)** - подробное описание аргумента

После описания всех аргументов можно перейти на закладку "исходный код" и приступить к редактированию кода функции.

3.4.1 Аргументы функции

Аргумент пользовательской функции - заранее определенный набор переменных, имеющий определенные свойства, для передачи данных в функцию и получения из нее результатов работы.

3.4.1.1 Типы и подтипы аргументов

Типы аргументов:

- **Float** - переменная с плавающей точкой;
- **Integer** - целочисленная переменная;
- **Boolean** - логическая переменная;
- **String** - строковая переменная
- **NumAnalog** - номер элемента в базе аналогов DAS
- **NumDiscret** - номер элемента в базе дискретов DAS
- **NumCounter** - номер элемента в базе счетчиков DAS
- **NumTC** - номер элемента в базе телеуправлений DAS

Подтипы аргументов:

- **Single** - одиночный элемент;
- **Array** - массив элементов;

3.4.1.2 Направление потока данных

Для аргумента функции должно быть выбрано направление потока:

- **Input** - входные данные - передаются в функцию;
- **Output** - выходные данные - передаются из функции;

Важно понимать, аргументы - номера элементов в базе DAS, являются **входными** данными, независимо от того читает функция из базы DAS или пишет в нее.

3.4.1.3 Дополнительные параметры аргументов

Обязательный (Mandatory) - если установлено это свойство, то данный аргумент является обязательным для работы функции. TM Builder будет выдавать ошибку построения конфигурации, если этот параметр не будет задан.

Подписка на события (Event subscription) - это свойство задается только для аргументов типа *NumDiscret*. Если оно установлено, то значения дискретов из базы DAS

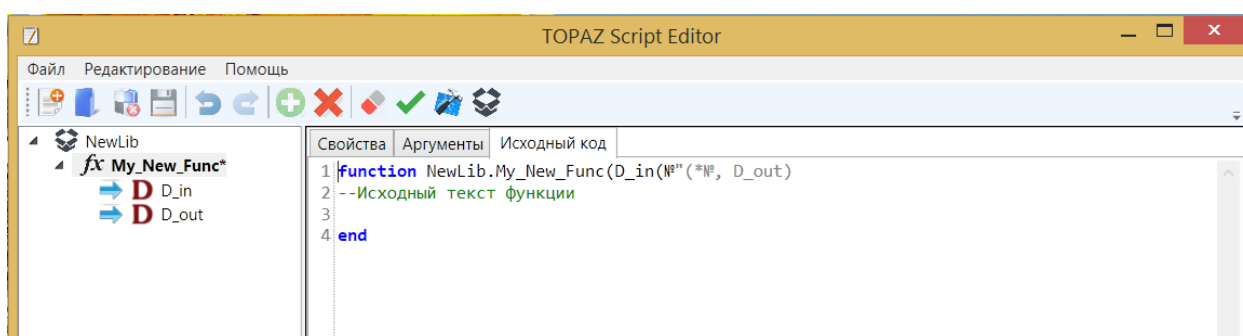
передаются в функцию из очереди событий, а не по циклическому опросу. Используется для параметров, которые могут менять свое значение чаще, чем цикл выполнения функции (см. п.5.5)

Значение по умолчанию - Начальное значение аргумента, если другое не задано в конфигурации.

Описание аргумента - произвольное текстовое описание

3.5 Наполнение функции

При переходе на закладку "исходный код", код текущей функции будет отображен на панели редактора.




В коде могут использоваться стандартные операторы и выражения языка LUA (см. приложение 1), а так же системные дополнения для работа с базой текущих параметров DAS и другими свойствами DAS (см. п. 4)

При нажатии на клавишу ESC появится подсказка, со списком всех доступных операторов и функций.

Т.к. пользовательские сценарии вызываются системой циклически, с заданным интервалом, то важно, при написании своих функций, избегать бесконечного зацикливания внутри сценария.

Использование пользовательских функций в системе конфигурирования Topaz DAS Access Server описаны в руководстве пользователя Topaz DAS Access Server, часть 2.1, глава 11.

3.6 Построение библиотеки

После того, как код будет написан, необходимо построить библиотеку. Для этого надо нажать кнопку  на панели инструментов или выбрать пункт меню "Редактирование"->" Построить библиотеку " - будет сформирован файл библиотеки и помещен в выходную папку. Только после этого файл библиотеки будет доступен для загрузки в контроллер из Topaz TMBUILDER.

Перейти в выходную папку можно нажав кнопку  (открыть папку сборки)

4. Переменные LUA

Переменные делятся на 2 типа - локальные и глобальные.

Локальные сохраняют свое значение только на время выполнения функции и не доступны за ее пределами. Локальные переменные должны быть явно объявлены с использованием ключевого слова **local**. Объявить локальную переменную можно в любом месте скрипта. Объявление может включать в себя присваивание переменной начального значения. Если значение не присвоено, переменная содержит nil

Объявление переменной dout типа таблица, присвоение полю value значения 0:

```
local dout = {}  
dout.value = 0
```

Для сохранения значений между вызовами пользовательской функции, можно воспользоваться **глобальными** переменными.

Все глобальные переменные являются полями обычной таблицы, называемой глобальным окружением. Эта таблица доступна через глобальную переменную **_G**

Глобальная переменная существует до тех пор, пока существует среда исполнения скрипта и доступна любому Lua-коду, выполняемому в этой среде

Глобальная переменная появляется в момент присваивания ей первого значения. До присваивания первого значения обращение к глобальной переменной даёт nil

Проверка на существование глобальной переменной var1, если она не существует, создание ее с присвоением значения 0:

```
if _G[var1] == nil then  
  _G[var1] = 0  
end
```

5. Системные дополнения DAS

5.1 Общие функции работы с DAS

dasGetSizeBaseD() - возвращает размер базы дискретов

dasGetSizeBaseA() - возвращает размер базы аналогов

dasGetSizeBaseC() - возвращает размер базы счетчиков

Размер базы - это максимальный номер элемента, который может быть использован. Задается в конфигурации DAS. Может принимать целочисленные значения 1 до 65535.

Попытка чтения/записи вне этого диапазона приведет к аварийному завершению работы DAS.

dasIsMaster() - возвращает true, если текущий режим работа контроллера "основной" и false, если "резервный".

5.2 Функции чтения/записи элементов базы текущих значений

dasDiscretRead(*n*) - чтение дискрета с номером *n* из базы текущих значений DAS

dasAnalogRead(*n*) - чтение аналога с номером *n* из базы текущих значений DAS

dasReadCounter(*n*) - чтение счетчика с номером *n* из базы текущих значений DAS

Аргумент *n* - целочисленная переменная, может принимать значения от 1 до 65535

Все функции чтения данных из базы текущих значений DAS возвращают структуру (таблицу) из трех полей:

- **value** - значение параметра (тип integer для счетчиков и дискретов, float для аналогов);
- **state** - флаги параметра (см. ниже);
- **timestamp** - метка времени в виде структуры из двух полей:
 - **timestamp.sec** - время в секундах (от 1 января 1970г.)
 - **timestamp.nsec** - количество наносекунд

dasDiscretWrite(*n*, *out*) - запись дискрета *out* в базу текущих значений, с индексом *n*

dasAnalogWrite(*n*, *out*) - запись аналога *out* в базу текущих значений, с индексом *n*

dasWriteCounter(*n*, *out*) - запись счетчика *out* в базу текущих значений, с индексом *n*

Аргумент *n* - целочисленная переменная, может принимать значения от 1 до 65535

Аргумент *out* - структура из трех полей:

- **value** - значение параметра (тип integer для счетчиков и дискретов, float для аналогов);
- **state** - флаги параметра (см. ниже);
- **timestamp** - метка времени (в UNIX формате);

5.3 Функции работы с флагами параметра

Параметр в базе DAS независимо от своего типа, кроме самого значения, имеет еще четыре битовых флага:

- Недостоверность
- Динамика
- Блокировка
- Замещение

Параметры типа аналог, кроме того имеют флаг *переполнения*

Для работы с ними предусмотрены специальные функции:

Проверка флагов:

dasIsUndefD(discret) - Возвращает **true**, если аргумент **discret**, имеет взведенный флаг недостоверности;

dasIsDynD(discret) - Возвращает **true**, если аргумент **discret**, имеет взведенный флаг динамики;

dasIsLockD(discret) - Возвращает **true**, если аргумент **discret**, имеет взведенный флаг блокировки;

dasIsReplasedD(discret) - Возвращает **true**, если аргумент **discret**, имеет взведенный флаг замещения;

dasIsUndefA(analog) - Возвращает **true**, если аргумент **analog**, имеет взведенный флаг недостоверности;

dasIsDynA(analog) - Возвращает **true**, если аргумент **analog**, имеет взведенный флаг динамики;

dasIsLockA(analog) - Возвращает **true**, если аргумент **analog**, имеет взведенный флаг блокировки;

dasIsReplasedA(analog) - Возвращает **true**, если аргумент **analog**, имеет взведенный флаг замещения;

dasIsOverflowA(analog) - Возвращает **true**, если аргумент **analog**, имеет взведенный флаг переполнения аналога

dasIsUndefC(counter) - Возвращает **true**, если аргумент **counter**, имеет взведенный флаг недостоверности;

dasIsDynC(counter) - Возвращает **true**, если аргумент **counter**, имеет взведенный флаг динамики;

dasIsLockC(counter) - Возвращает **true**, если аргумент **counter**, имеет взведенный флаг блокировки;

dasIsReplasedC(counter) - Возвращает **true**, если аргумент **counter**, имеет взведенный флаг замещения;

Установка флагов:

discret = dasSetUndefD(discret) - Вводит у переменной **discret** флаг недостоверности;

discret = dasSetDynD(discret) - Снимает у переменной **discret** флаг динамики;

discret = dasSetLockD(discret) - Снимает у переменной **discret** флаг блокировки;

discret = dasSetReplasedD(discret) - Снимает у переменной **discret** флаг замещения;

analog = dasSetUndefA(analog) - Вводит у переменной **analog** флаг недостоверности;
analog = dasSetDynA(analog) - Снимает у переменной **analog** флаг динамики;
analog = dasSetLockA(analog) - Снимает у переменной **analog** флаг блокировки;
analog = dasSetReplasedD(analog) - Снимает у переменной **analog** флаг замещения;

couner = dasSetUndefC(couner) - Вводит у переменной **couner** флаг недостоверности;
couner = dasSetDynC(couner) - Снимает у переменной **couner** флаг динамики;
couner = dasSetLockC(couner) - Снимает у переменной **couner** флаг блокировки;
couner = dasSetReplasedC(couner) - Снимает у переменной **couner** флаг замещения;

Сброс флагов:

discret = dasClearUndefD(discret) - Снимает у переменной **discret** флаг недостоверности;
discret = dasClearDynD(discret) - Снимает у переменной **discret** флаг динамики;
discret = dasClearLockD(discret) - Снимает у переменной **discret** флаг блокировки;
discret = dasClearReplasedD(discret) - Снимает у переменной **discret** флаг замещения;

analog = dasClearUndefA(analog) - Снимает у переменной **analog** флаг недостоверности;
analog = dasClearDynA(analog) - Снимает у переменной **analog** флаг динамики;
analog = dasClearLockA(analog) - Снимает у переменной **analog** флаг блокировки;
analog = dasClearReplasedA(analog) - Снимает у переменной **analog** флаг замещения;

couner = dasClearUndefC(couner) - Снимает у переменной **couner** флаг недостоверности;
couner = dasClearDynC(couner) - Снимает у переменной **couner** флаг динамики;
couner = dasClearLockC(couner) - Снимает у переменной **couner** флаг блокировки;
couner = dasClearReplasedC(couner) - Снимает у переменной **couner** флаг замещения;

5.4 Функции блокировки

Блокировка элемента базы текущих значений DAS, позволяет заблокировать изменение элемента из других приложений. Т.е. между выполнением функций блокировки и разблокировки, значение данного элемента базы останется гарантированно неизменным.

dasLockDiscrete(n) - блокирует дискрет с индексом **n**;
dasUnlockDiscrete(n) - разблокирует дискрет с индексом **n**;
dasGetDiscreteLockState(n) - возвращает статус блокировки дискрета с индексом **n**;

dasLockAnalog(n) - блокирует аналог с индексом **n**;
dasUnlockAnalog(n) - разблокирует аналог с индексом **n**;
dasGetAnalogLockState(n) - возвращает статус блокировки аналога с индексом **n**;

dasLockCounter(n) - блокирует счетчик с индексом **n**;
dasUnlockCounter(n) - разблокирует счетчик с индексом **n**;
dasGetCounterLockState(n) - возвращает статус блокировки счетчика с индексом **n**;

5.5 Функции работы с буфером событий

В текущей реализации существует 2 способа получения данных из базы текущих значений DAS: циклический опрос и подписка на события. Функции описанные в п. 5.2 предназначены для циклического опроса.

Но если значения в базе DAS могут меняться чаще, чем цикл выполнения пользовательских сценариев, т.е. существует вероятность потерять некоторые данные, следует использовать второй механизм - подписку на события.

Принцип работы с событиями лучше рассмотреть на конкретном примере:

Ниже приведен пример получения данных из буфера событий и распечатки их в консоль:

```
-- объявляем функцию с аргументом In типа "список номеров дискретов"
function UserLib1.PrintEvent(In)

    --Получаем идентификатора задачи
    local taskId = _G["DAS_LUA_TASK_ID"]

    -- Подписка функции на события дискретов в In
    DasServices.RegisterEventHandler(taskId, In)

    -- Перебираем все события в цикле
    for key, event in pairs(_G["IecEventDb"]["observers"][taskId]["events"]) do

        -- Получаем очередное событие
        local event = _G["IecEventDb"]["observers"][taskId]["events"][key]

        -- считываем дискрет из события
        local evtD = event.discrete

        -- печатаем номер дискрета для которого пришло событие
        print("Discret No:",event.basenum)

        -- печатаем значение дискрета
        print("val=",tostring(evtD.value))

        -- печатаем флаги дискрета
        print("st=",tostring(evtD.state))

        -- печатаем метку времени
        print("timestamp=",os.date("%Y-%m-%d %X", evtD.timestamp.sec) .. ":" ..tostring(evtD.timestamp.nsec))

    end

end

end
```

5.6 Функции отладки

dasWriteToLog(msg) - запись в ЛОГ-фал произвольного текстового сообщения **msg**

echoD(discrete) - Печать дискрета в консоль

echoA(analog) - Печать аналога в консоль

echoC(counter) - Печать счетчика в консоль

print(msg) - Печать в консоль произвольного текстового сообщения **msg**

5.7 Функции работы с временем

dasGetMS() - возвращает время в миллисекундах (циклический счетчик) принимает значения от 0 до 4294967295, после чего сбрасывается в 0;

sysGetTimeStamp() - возвращает текущую метку времени, как таблицу из 2х полей:

- **.sec** - время в секундах (от 1 января 1970г.)
- **.nsec** - количество наносекунд

getMsDiff(start, stop) - возвращает разницу между временем start и stop с учетом переполнения (однократного) счетчика

CheckTimestampIdentity(T1, T2) - возвращает true, если метки времени T1 и T2 равны. Где T1 и T2 - таблица из 2х полей .sec и .nsec

6. Прикладные библиотеки

При установке TOPAZ SCRIPT EDITOR, так же будут установлены прикладные библиотеки с часто употребляемыми пользовательскими сценариями. Исходный код этих сценариев доступен для просмотра.

6.1 Системная библиотека прикладных задач DasComponents

6.1.1 Функция *Shutdown*

Назначение:

Отключение контроллера по состоянию одного или нескольких дискретов, например "низкий заряд батареи"

Аргументы:

- **Signals** - список номеров контролируемых дискретов
- **Time** - время (в секундах) выключения при отсутствии обратной связи от sql менеджера

Описане:

Если Все достоверные дискреты из входного списка имеют значение "1", то запускается процедура отключения контроллера. Резервные контроллер отключается сразу, основной - после завершения записи всех данных в базу данных или по истечении заданного таймаута **Time**

6.2 Библиотека функций для решения типовых задач DasUtil

В этой библиотеке собраны наиболее часто востребованные задачи обработки телемеханических параметров.

6.2.1 Функция *OR*

Назначение:

Сложение нескольких дискретов через логическое ИЛИ

Аргументы:

- **In** - список номеров дискретов
- **Out** - номер дискрета в который будет записан результат
- **Parms** - Целочисленная константа, определяющая режим работы функции

Описане:

Если хотя бы один из входных дискретов достоверен и имеет значение 1, то в выходной дискрет будет записан 1, иначе 0.

6.2.2 Функция AND

Назначение:

Сложение нескольких дискретов через логическое И

Аргументы:

- **In** - список номеров дискретов
- **Out** - номер дискрета в который будет записан результат
- **Parms** - Целочисленная константа, определяющая режим работы функции

Описане:

Если все достоверные входные дискреты имеют значение 1, то в выходной дискрет будет записан **1**, иначе **0**. (При Parms=1, выходной дискрет будет помечен признаком недостоверности, если хотя бы один входной дискрет недостоверен)

6.2.3 Функция SDPS

Назначение:

Функция преобразования двух однопозиционных выключателя (SPS) в один двухпозиционный (DPS)

Аргументы:

- **In1** - номер дискрета, который соответствует значению "включен" 2 bit (On)
- **In2** - номер дискрета, который соответствует значению "отключен" 1 bit (Off)
- **Out** - номер дискрета в который будет записан результат
- **Parms** - Целочисленная константа, определяющая режим работы функции

Описане:

Функция преобразует два однопозиционных выключателя (0/1) в один двухпозиционный (0/1/2/3): 0 - обрыв (00); 1 - отключен (01); 2 - включен (10); 3 - промежуточное положение (11)

Если один из входных дискретов недостоверен, то расчет производится по одному, достоверному.

6.2.4 Функция UDPS

Назначение:

Функция преобразования одного двухпозиционного выключателя (DPS) в два однопозиционных (SPS)

Аргументы:

- **In** - номер входного дискрета, в котором хранится двухпозиционное положение выключателя.
- **Out1** - номер выходного дискрета, который соответствует значению "включен"
- **Out2** - номер выходного дискрета, который соответствует значению "отключен"

Описане:

Если входной дискрет $In=0$, то $Out1=0$, $Out2=0$; Если $In=1$, то $Out1=0$, $Out2=1$; Если $In=2$, то $Out1=1$, $Out2=0$; Если $In=3$, то $Out1=1$, $Out2=1$

6.2.5 Функция *MEAN3*

Назначение:

Расчет среднего по трем аналогам

Аргументы:

- **In1** - номер входного аналога 1
- **In2** - номер входного аналога 2
- **In3** - номер входного аналога 3
- **Out** - номер выходного аналога
- **kof** - Коэффициент минимального значения
- **Parms** - Целочисленная константа, определяющая режим работы функции

Описане:

Функция вычисляет среднее значение по трем переменным $In1$, $In2$ и $In3$ из базы данных аналоговых переменных.

Коэффициент kof является порогом: если переменная $In1$, $In2$ или $In3$ превышает порог ($>$), то она используется в расчетах (иначе игнорируется).

Если переменная недостоверна, то эта переменная в расчете среднего не участвует.

6.2.6 Функция *BITSPLITC2D*

Назначение:

Побитовый разбор счетчика

Аргументы:

- **inpC** - номер входного параметра в базе счетчиков
- **bit0OutD** - номер выходного дискрета в который будет записано состояние бита 0
- **bit1OutD** - номер выходного дискрета в который будет записано состояние бита 1
- ...
- **bit15OutD** - номер выходного дискрета в который будет записано состояние бита 15

Описане:

Если задан номер выходного дискрета, то в него будет записано состояние соответствующего бита значения входного счетчика.

6.2.7 Функция *OREVT*

Назначение:

Сложение нескольких дискретов через логическое ИЛИ. Получение данных через буфер событий с учетом флага блокировки дискрета

Аргументы:

- **In** - список номеров входных дискретов
- **Out** - номер дискрета в который будет записан результат
- **Parms** - Целочисленная константа, определяющая режим работы функции
- **lockIn** - номер входного дискрета блокировки
- **lockOut** - номер выходного дискрета блокировки

Описане:

Если хотя бы один из входных дискретов достоверен и имеет значение 1, то в выходной дискрет будет записан **1**, иначе **0**.

Если хотя бы один дискрет имеет флаг блокировки, то в выходной дискрет lockOut будет записана единица, иначе будет записано значение дискрета lockIn.

6.2.8 Функция *ANDEVT*

Назначение:

Сложение нескольких дискретов через логическое И. Получение данных через буфер событий с учетом флага блокировки дискретаю

Аргументы:

- **In** - список номеров входных дискретов
- **Out** - номер дискрета в который будет записан результат
- **Parms** - Целочисленная константа, определяющая режим работы функции
- **lockIn** - номер входного дискрета блокировки
- **lockOut** - номер выходного дискрета блокировки

Описане:

Если все достоверные входные дискреты имеют значение 1, то в выходной дискрет будет записан **1**, иначе **0**.

Если хотя бы один дискрет имеет флаг блокировки, то в выходной дискрет lockOut будет записана единица, иначе будет записано значение дискрета lockIn.

Приложение 1

Язык Lua. Краткое описание

1. Ключевые слова языка:

<i>and</i>	<i>function</i>	<i>repeat</i>
<i>break</i>	<i>goto</i>	<i>return</i>
<i>do</i>	<i>if</i>	<i>then</i>
<i>else</i>	<i>in</i>	<i>true</i>
<i>elseif</i>	<i>local</i>	<i>until</i>
<i>end</i>	<i>nil</i>	<i>while</i>
<i>false</i>	<i>not</i>	
<i>for</i>	<i>or</i>	

2. Зарезервированные лексемы:

+	<=	[
-	>=]
*	<	::
/	>	;
%	=	:
^	(,
#)	.
==	{	..
~=	}	..

3. Однострочные комментарии:

-- *пример комментария*

4. Многострочный комментарий:

--[[*многострочный комментарий*]]

5. Переменные бывают 3 типов: глобальные переменные, локальные переменные и поля таблиц (объектов). Одиночное имя может обозначать: глобальную переменную, локальную переменную, а также формальный параметр функции).

var ::= Name

Любая объявленная переменная по умолчанию является глобальной, если только не объявлена как локальная с помощью модификатора видимости ***local***.

6. Тип данных «таблица» представляют собой массив записей «ключ-значение». В качестве ключа может использоваться строка или число.

Пример:

var = {} –объявление объекта

var["value"] = 25 – инициализация поля «value» и запись в него значения 25

-- доступ к данному полю таблицы можно осуществить 2 способами

print (var["value"]) -->напечатает на экране 25

print (var.value) -->напечатает на экране 25

7. Единицей исполнения в Lua является блок операторов, которые выполняются последовательно.

8. Lua позволяет одновременное множественное присваивание:

stat ::= varlist '=' explist

varlist ::= var {' ' var}

explist ::= exp {' ' exp}

В начале выполняются все вычисления и только после их окончания, выполняется присваивание.

i = 3

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

9. Управляющие структуры языка

Управляющие структуры **if**, **while**, и **repeat** имеют обычное значение синтаксис:

stat ::= while exp do block end

stat ::= repeat block until exp

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

Все значения, отличные от **nil** и **false**, считаются **true** (в частности число **0** и пустая строка также являются **true**)

goto передает управление в точку, помеченную соответствующей меткой.

stat ::= goto Name

stat ::= label

label ::= '::' Name '::'

Метка видна во всем блоке, где она была объявлена, за исключением внутренних, вложенных блоков и функций, где были объявлены метки с такими же именами.

break прекращает выполнение **while**, **repeat**, **for** циклов, передавая управление оператору, следующему за меткой. Оператор **break** прерывает выполнение близлежащего цикла.

stat ::= break

10. Оператор **for** имеет 2 формы: числовую и обобщенную.

Числовой оператор **for** выполняет код в цикле, в то время как управляющая переменная используется в операции арифметической прогрессии.

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

Более точный синтаксис оператора:

for v = e1, e2, e3 do block end

где **e1** – начальное значение переменной цикла

e2 – конечное значение переменной цикла

e3 – шаг наращивания переменной цикла

Данный блок эквивалентен:

```
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
    block
    var = var + step
  end
end
```

Заметки:

- все 3 выражения вычисляются перед началом выполнения цикла
- переменные *var*, *limit*, *step* – невидимые
- если 3 выражение (шаг наращивания) отсутствует, то используется 1
- можно использовать **break** для выхода из цикла
- переменная цикла *v* является локальной для цикла; ее нельзя использовать после окончания цикла

11. Локальные переменные могут быть объявлены в любом месте внутри блока. Объявление может включать инициализацию.

stat ::= local namelist ['='] explist

12. Базовые выражения языка:

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

Примеры выражений

```
f()          -- adjusted to 0 results
g(f(), x)    -- f() is adjusted to 1 result
g(x, f())    -- g gets x plus all results from f()
a,b,c = f(), x -- f() is adjusted to 1 result (c gets nil)
a,b = ...    -- a gets the first vararg parameter, b gets
              -- the second (both a and b can get nil if there
              -- is no corresponding vararg parameter)

a,b,c = x, f() -- f() is adjusted to 2 results
a,b,c = f()   -- f() is adjusted to 3 results
return f()    -- returns all results from f()
return ...    -- returns all received vararg parameters
return x,y,f() -- returns x, y, and all results from f()
```

f() -- creates a list with all results from *f()*
{...} -- creates a list with all vararg parameters
f(), nil -- *f()* is adjusted to 1 result

13. Lua поддерживает автоматическое преобразование строки в число. Любая арифметическая операция, применяемая к строке, вызывает фоновую конвертацию строки в число.

14. Операции сравнения:

== ~= < > <= >=

Данные операторы всегда возвращают **true** или **false**

15. Логические операции.

Логическими операторами в Lua являются **and**, **or** и **not**. Также как и управляющие структуры, все логические операторы возвращают **false**, если значение **nil** или **false**, иначе **true**. Оператор **not** всегда возвращает **false** или **true**. Оператор **and** возвращает первый аргумент, если значение **nil** или **false**, иначе возвращается второй аргумент. Оператор **or** возвращает первый аргумент, если значение отличается от **nil** или **false**, иначе возвращается второй элемент. Пример использования:

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*

16. Для объединения 2 строк в одну, используется оператор «...».
print("Hello ".."World!")

17. Для получения размера объекта используется оператор **#**

18. Приоритет операторов

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

19. Синтаксис объявления функций

functiondef ::= function funcbody
funcbody ::= (' [parlist] ') block end