Описание языка ST в AgavaSCADA/ AgavaPLC

Редакция от 15.10.2026 · docs.kb-agava.ru

Содержание

1 Дополнительные документы 2
2 Основы синтаксиса 2
  2.1 Комментарии 2
3 Единицы организации программ (POU) 2
  3.1 PROGRAM 2
  3.2 FUNCTION 2
  3.3 FUNCTION_BLOCK 3
4 Переменные 3
  4.1 Секции переменных 4
  4.2 Типы данных 4
    4.2.1 Логический тип 4
    4.2.2 Целочисленные типы 4
    4.2.3 Вещественные типы 4
    4.2.4 Строки 4
    4.2.5 Время и дата 5
    4.2.6 Массивы 5
    4.2.7 Структуры 5
    4.2.8 Перечисления 5
5 Операторы 6
  5.1 Присваивание и сравнение 6
  5.2 Арифметические операторы 6
  5.3 Логические и побитовые операторы 7
  5.4 Литералы 7
6 Условия 7
  6.1 IF / ELSIF / ELSE 7
  6.2 CASE 7
7 Циклы 8
  7.1 WHILE 8
  7.2 REPEAT 8
  7.3 FOR 8
  7.4 EXIT и RETURN 8
8 Вызовы функций и функциональных блоков 8
9 Ограничения 9
10 Пример 9

Данное описание содержит базовое руководство по созданию программ на языке `Structured Text` (ST, IEC 61131-3) и их использованию в среде разработки Agava.

Документ предназначен для `AgavaSCADA/AgavaPLC 1.7+`

1 Дополнительные документы

- Стандартная библиотека IEC (iecstdlib).
- Описание базовых классов AgavaSCADA/AgavaPLC.
- Объектная модель AgavaSCADA/AgavaPLC.

Вы также можете ознакомиться с другими документами, размещенными в категории AgavaSCADA/AgavaPLC.

2 Основы синтаксиса

Программа на ST состоит из объявлений (типы, переменные, POU) и операторов. Операторы завершаются точкой с запятой. Присваивание выполняется оператором `:=`, сравнение на равенство — оператором `=`.

```
iCounter := iCounter + 1;
bOk := (iCounter = 10);
```

2.1 Комментарии

Поддерживаются три вида комментариев:

```
(* IEC-комментарий *)
/* C-style комментарий */
// комментарий до конца строки
```

3 Единицы организации программ (POU)

3.1 PROGRAM

Программа — единица, вызываемая задачей/хостом. Локальные переменные секции `VAR` сохраняют значения между вызовами (ретенция в рамках скана/повторных вызовов).

```
PROGRAM PumpCtrl
VAR
    iState : INT := 0;
END_VAR
    iState := iState + 1;
END_PROGRAM
```

3.2 FUNCTION

Функция — подпрограмма без собственного «состояния» между вызовами (кроме локальных экземпляров FB). Тип результата указывается после имени; результат записывается в переменную с именем функции.

```
FUNCTION Add : INT
VAR_INPUT
    a : INT;
    b : INT;
END_VAR
    Add := a + b;
END_FUNCTION
```

```
FUNCTION main : BOOL
VAR
    r : INT;
END_VAR
    r := Add(a := 2, b := 3);
    main := (r = 5);
END_FUNCTION
```

Если тип результата опущен, функция соответствует `void` (ничего не возвращает).

3.3 FUNCTION_BLOCK

Функциональный блок — экземплярный тип со входами/выходами и внутренним состоянием. Экземпляр объявляется в секции переменных и вызывается как процедура.

```
FUNCTION_BLOCK FB_Inc
VAR_INPUT
    IN : BOOL;
END_VAR
VAR_OUTPUT
    Q : BOOL;
END_VAR
VAR
    bPrev : BOOL := FALSE;
END_VAR
    Q := IN AND NOT bPrev;
    bPrev := IN;
END_FUNCTION_BLOCK
```

```
FUNCTION main : BOOL
VAR
    fb : FB_Inc;
END_VAR
    fb(IN := TRUE);
    main := fb.Q;
END_FUNCTION
```

Внутри FB поддерживаются методы Codesys-стиля `METHOD … END_METHOD`.

4 Переменные

Переменные объявляются в секциях `VAR*` / `VAR_GLOBAL` в виде:

```
имя {, имя} : тип [:= начальное_значение];
```

```
VAR
    a, b : INT := 0;
```

```
    sMsg : STRING := 'готово';
END_VAR
```

4.1 Секции переменных

Секция	Назначение
VAR	Локальные переменные
VAR_INPUT	Входные параметры
VAR_OUTPUT	Выходные параметры
VAR_IN_OUT	Параметры «вход-выход» (по ссылке)
VAR_TEMP	Временные локали PROGRAM (не сохраняются между вызовами)
VAR_GLOBAL	Глобальные переменные compilation unit
VAR CONSTANT / VAR_GLOBAL CONSTANT	Константы

Модификаторы RETAIN / PERSISTENT в текущей реализации не поддерживаются (выдаётся предупреждение; объявление обрабатывается как обычный VAR).

4.2 Типы данных

4.2.1 Логический тип

Тип ST	Тип AS	Значения
BOOL	bool	FALSE (0), TRUE (1)

4.2.2 Целочисленные типы

Тип ST	Тип AS	Диапазон (кратко)
SINT	int8	−128 … 127
INT	int16	−32 768 … 32 767
DINT	int	−2 147 483 648 … 2 147 483 647
LINT	int64	64-битное знаковое
USINT / BYTE	uint8	0 … 255
UINT / WORD	uint16	0 … 65 535
UDINT / DWORD	uint	0 … 4 294 967 295
ULINT / LWORD	uint64	64-битное беззнаковое

4.2.3 Вещественные типы

Тип ST	Тип AS
REAL	float
LREAL	double

Нетипизированный вещественный литерал (1.5) трактуется как REAL. Для LREAL используйте явный литерал: LREAL#3.14159.

4.2.4 Строки

Тип ST	Описание

STRING / STRING(n) / STRING[n]	Строка (UTF-8). С размером — ограничение длины при присваивании
WSTRING / WSTRING(n) / WSTRING[n]	Широкая строка (UTF-16)

Строковые литералы заключаются в одинарные или двойные кавычки. Удвоение кавычки внутри строки экранирует её. Также поддерживаются escapes через `$` (например `$N`).

Функции работы со строками (`LEN`, `LEFT`, `RIGHT`, `MID`, `CONCAT`, `FIND` и др.) описаны в документе Standard - стандартная библиотека.

4.2.5 Время и дата

Типы `TIME`, `TOD` (`TIME_OF_DAY`), `DATE`, `DT` (`DATE_AND_TIME`) и L-варианты представляются как `int64` (семантика OSCAT: TIME/TOD — миллисекунды; DATE/DT — секунды).

Литералы:

```
T#50ms
T#1h30m
DATE#2025-01-01
TOD#14:03:30
DT#2025-11-19-14:03:30
```

4.2.6 Массивы

Объявление:

```
ARR : ARRAY [0..9] OF INT;
MATRIX : ARRAY [0..2, 0..2] OF REAL;
```

Индексация с учётом нижней границы объявления. Литерал инициализации:

```
A : ARRAY [0..2] OF INT := [10, 20, 30];
```

4.2.7 Структуры

```
TYPE Person :
STRUCT
    name   : STRING;
    age    : INT;
    height : REAL;
END_STRUCT
END_TYPE
```

```
VAR
    p1 : Person := (name := 'John', age := 30, height := 1.85);
END_VAR
```

```
p1.age := 31;
```

4.2.8 Перечисления

```
TYPE TrafficLight : (Red := 1, Yellow := 2, Green := 3);
END_TYPE
```

```
VAR
    light : TrafficLight := TrafficLight#Red;
END_VAR
```

Перечисление компилируется как целочисленный тип; доступ к значениям — через `Type#Enumerator`.

5 Операторы

5.1 Присваивание и сравнение

Оператор	Символ	Пример	Операция
Присваивание	:=	x := y	Присвоить y переменной x
Равно	=	x = y	TRUE, если x равно y
Не равно	<>	x <> y	TRUE, если x не равно y
Меньше	<	x < y	TRUE, если x меньше y
Больше	>	x > y	TRUE, если x больше y
Меньше или равно	<=	x <= y	TRUE, если x ≤ y
Больше или равно	>=	x >= y	TRUE, если x ≥ y

5.2 Арифметические операторы

Оператор

Символ

Пример

Сложение

+

x + y

Вычитание

-

x − y

Умножение

*

x * y

Деление

/

x / y

Остаток

MOD

x MOD y

5.3 Логические и побитовые операторы

Оператор	Символ	Примечание
НЕ	NOT	Для BOOL — логическое; для целых — побитовое
И	AND / &	Для BOOL — логическое; для целых — побитовое
ИЛИ		Для BOOL — логическое; для целых — побитовое
ИСКЛ. ИЛИ	XOR	Для BOOL — логическое; для целых — побитовое

Сдвиги и ротации — стандартные функции IEC: `SHL`, `SHR`, `ROL`, `ROR` (см. Standard - стандартная библиотека).

5.4 Литералы

- Целые и вещественные: `42`, `3.14`, `1e-3`; допускается разделитель `_` в числах.
- Базированные: `2#1010`, `8#77`, `16#FF`.
- Типизированные: `DINT#0`, `REAL#1.5`, `BYTE#255`.
- BOOL: `TRUE`, `FALSE`.
- Доступ к биту: `dwordVar.3`.

6 Условия

6.1 IF / ELSIF / ELSE

```
IF iState = 0 THEN
    iState := 1;
ELSIF iState = 1 THEN
    iState := 2;
ELSE
    iState := 0;
END_IF
```

6.2 CASE

```
CASE selector OF
    0:
        result := 10;
    1, 3:
        result := 20;
    4..7:
        result := 30;
ELSE
    result := 0;
END_CASE
```

Ветки без «проваливания» (fall-through). При пересечении диапазонов выбирается первая подходящая ветка.

7 Циклы

7.1 WHILE

```
i := 0;
WHILE i < 5 DO
    i := i + 1;
END_WHILE
```

7.2 REPEAT

```
i := 0;
REPEAT
    i := i + 1;
UNTIL i >= 5
END_REPEAT
```

Тело `REPEAT` выполняется хотя бы один раз.

7.3 FOR

```
FOR i := 0 TO 10 BY 2 DO
    sum := sum + i;
END_FOR
```

`BY` опционален (по умолчанию +1). Для обратного счёта: `BY -1`.

7.4 EXIT и RETURN

- `EXIT;` — выход из ближайшего цикла.
- `RETURN;` / `RETURN expr;` — выход из функции (без выражения в FUNCTION возвращается переменная результата).

Оператор `CONTINUE` как statement не поддерживается; пропуск итерации оформляйте через `IF`.

8 Вызовы функций и функциональных блоков

Позиционные и именованные аргументы:

```
r := Add(2, 3);
r := Add(a := 2, b := 3);
```

Для выходов FB/функций используется `=>`:

```
TestOUT(in1 := 15, iOut => intVal, bOut => boolVal);
```

Вызов экземпляра FB:

```
ton1 : TON;
ton1(IN := TRUE, PT := T#50ms);
ton1();   (* повторный опрос с сохранённым состоянием *)
```

Опущенные входы FB сохраняют предыдущие значения экземпляра (поведение IEC / CoDeSys).

Стандартные FB (`TON`, `TOF`, `TP`, `RS`, `SR`, `CTU`, …) и функции IEC описаны в документе Библиотека Standard AgavaSCADA/AgavaPLC.

9 Ограничения

- Указатели (`POINTER TO`, `ADR`, оператор `^`) не поддерживаются.
- `RETAIN` / `PERSISTENT` — не поддерживаются, работают как обычные `VAR`.
- `PROGRAM` нельзя использовать как экземплярный FB из другого POU, вызов возможен только из задачи.

10 Пример

```
FUNCTION_BLOCK FB_Counter
VAR_INPUT
    CU : BOOL;
    R  : BOOL;
END_VAR
VAR_OUTPUT
    CV : INT;
    Q  : BOOL;
END_VAR
    IF R THEN
        CV := 0;
    ELSIF CU THEN
        CV := CV + 1;
    END_IF
    Q := (CV >= 10);
END_FUNCTION_BLOCK
```

```
FUNCTION main : BOOL
VAR
    fb : FB_Counter;
    i  : INT;
END_VAR
    FOR i := 1 TO 10 DO
        fb(CU := TRUE, R := FALSE);
        fb(CU := FALSE, R := FALSE);
    END_FOR
    main := fb.Q AND (fb.CV = 10);
END_FUNCTION
```