General information on the programming language

The programming language is intended for creation of programs describing functioning of “Programming language” typical block, as well as for setting global parameters and variables in the “Parameters” tab.

Using the programming language in the “Parameters” tab, one can write a program, which will be able to perform manipulations with diagram objects in course of calculation and also program video frames.

The “Parameters” tab is available on each nesting level of submodels, and dedicated local program (script) can be written for each submodel in this tab.

Identifiers (names of constants, variables, labels, functions and procedures) can contain Latin and Cyrillic characters, underlining sign “\_” and numerals. Identifier shall start with a character or underlying sign and can have a random length. Key words are not permitted to be used as identifiers (these words are automatically semi-bolded when entered). Uppercase and lowercase characters are not distinguished in identifiers by default (t and T are the same identifiers). This parameter can be changed in the system settings.

The program consists of declarations and operators. Declarations are started with key words and initialize corresponding variables and constants.

**const** – constants;

**var** – algebraic variables;

**init** – dynamic (differential) variables;

**input** – block input variables;

**output** – block output variables.

Comma-divided names of variables or constants are listed after the key word with enabled setting of initial values, for example:

**const** a=5.3,b=1.2e-4,c=10,y0=a\*b/c;

**var** x1,v=b;

**output** y=y0;

If initial value has not been set, then it assumed as equal to 0. Semicolon (;) is used as a divider of declarations and operators.

Input, output and dynamic (differential) variables shall be described in declarations. Other variables can be set automatically according to the expression assigned for a variable. It is recommended (but not is a must) to locate declarations in the beginning of program. Variable or constant can be used only after its description in declaration or assignment in the assignment operator.

The following system variables are accessible.

|  |  |  |
| --- | --- | --- |
| Variable | Data type | Purpose |
| time | Real (double) | Simulation time |
| stepsize | Real (double) | Step of integration |
| goodstep | Binary (Boolean) | “Good” step flag |
| getderi | Binary (Boolean) | Flag of calculation of derivative values |
| setstepflag | Binary (Boolean) | Flag of forced step assignment |
| newstepvalue | Real (double) | Forced step value in case setstepflag = 1 |

The program can contain comments embraced in curly brackets:

*{comment text}*

otherwise as:

*//comment text to the string end.*

Some simple examples of programs for the “Programming language” block are given below.

**Example 1.**

*Output block variable – maximum value of two input variables.*

|  |  |
| --- | --- |
|  | **input** u1,u2;  **if** u1>u2 **then** y=u1 **else** y=u2;  **output** y; |

This program can be simplified using the **max** function of the language, then the output variable calculation operator will be written as y=**max**(u1,u2);

**Example 2.**

*Output variable memorizes the maximum value of input variable.*

|  |  |
| --- | --- |
|  | **input** u;  **if** time=0 **then** y=u **els**e  **if** goodstep **then** y=**max**(y,u);  **output** y; |

time=0 value corresponds to the initial moment of simulation time. y variable practically sets the block state. Calculation of such variables is recommended to be performed only on a good step when goodstep Boolean variable is equal to 1. Otherwise, (when goodstep=0) u variable can be calculated, for example, in intermediate iteration of solution of algebraic equations, in the end of which y variable value will be incorrect.

**Example 3.**

*The block calculates the transient process period, i.e. the period, for which the absolute value of z input variable is reduced to z0 and then does not exceed this value.*

|  |  |
| --- | --- |
|  | **const** z0=0.05;  **input** z;  **output** T=0; *//0 – initial value*  **if** goodstep **and** (**abs**(z)>z0) **then** T=time; |

**Example 4.**

*The block describes Van der Pol equation.*

|  |  |
| --- | --- |
|  | **const** mu=1e6;  **init** x1=2,x2=0;  x1’=x2; *//differential equations*  x2’=mu\*((1-x1^2)\*x2-x1);  **output** x1,x2; |

Here assignment operators set differential equations and define new x1’, x2’ variables (derivatives of related variables), which, as well as other variables, can be used in mathematical expressions.

**Example 5.**

*Counter of a number of input pulses.*

|  |  |
| --- | --- |
|  | **input** u;  **output** Nimp=0;  **var** u0=1;  **if** goodstep **and** (u0<=0) **and** (u>0) **then**  **begin** Nimp=Nimp+1; u0=u **end**; |

Pulses are counted by their front edges, i.e., by the moment of simulation time, at which the input variable is increased and is becoming positive.

Tables of programming language key words can be found below. Symbols < > mark compulsory identifiers. Symbols { } mark auxiliary identifiers.