FeSCADA – Logic programs

Introduction

A new utility called “Programs” was added to FeSCADA software. With this feature the user can setup and run logic programs.

This chapter will cover the following topics:

1. Description
2. Abstract Syntax Trees
3. If… then… else instruction
4. Operations list
5. Programs setup and run
6. Application examples
7. Conclusions

1) Description

In a classical automation application the PLC will execute the logic program to control the machine. The SCADA software will act like an HMI, to allow the operator to see indicators and to press command buttons. The PLC was built for speed and specialized for Boolean logic, timers and counters.

The possibility of running a logic program with the SCADA software can add a degree of freedom to the automation. It can help to solve more difficult tasks, or to develop more complex applications. For example, in many situations a single SCADA software station is supervising more than one machine, each one with its own PLC. In these cases a logic for synchronization and for interlocking between machines has to be implemented. This logic doesn’t need to be fast. The synchronization logic, in the past, was done with a master PLC. But nowadays it can be done with the SCADA software.

Another family of applications is using only remote terminal units for inputs and outputs (IO RTU) and no PLCs. In these cases the SCADA software can run the logic control program for inputs and outputs. Many small and medium automation applications are simple and slow enough to be implemented like that.

Examples of logic programs use within SCADA software:
- Scaling tags after mathematical equations
- Complex mathematical operations like: sin, cos, log, exp, etc.
- Special alarms implemented with mathematical algorithms
- Logic at bit level, value level and/or expression level

Two of the most important demands from a logic program in a SCADA software are to be easy to program/change it and load/reload it. The easiest way to write a program on computers is by writing text. The text program is called source code. From here there are 2 ways to go. Use a compiler or an interpreter. The compiler is taking the source code and is translating it to machine code. The user will execute the machine code. The interpreter is translating and executing the source code without translating and creating a machine code first.

FeSCADA logic programs are created by writing text and are using an interpreter to run the source code. Every program is saved as a text in a file. The text is read from the file and interpreted by a text parser. Key words, tag names, numbers and special characters are identified and separated in a list. If all tokens are recognized (no syntax errors), the next step is to create an intermediate representation, called abstract syntax trees (AST), in the memory of the computer, and to build and link the trees with pointers to: statements, expressions, operations and operands. Once this is done the program can be executed. The work to create the abstract syntax trees is done when the program starts or when the user explicitly wants to reload a modified program. After that the program is executed from memory.

2) Abstract Syntax Trees

In computer science, a tree list is a data structure that consists of one or more nodes organized in a hierarchy. The tree has one root which is the top node. All nodes, except the root, have one parent. A node without children is called a leaf node. If a node is not root or leaf is called interior node.

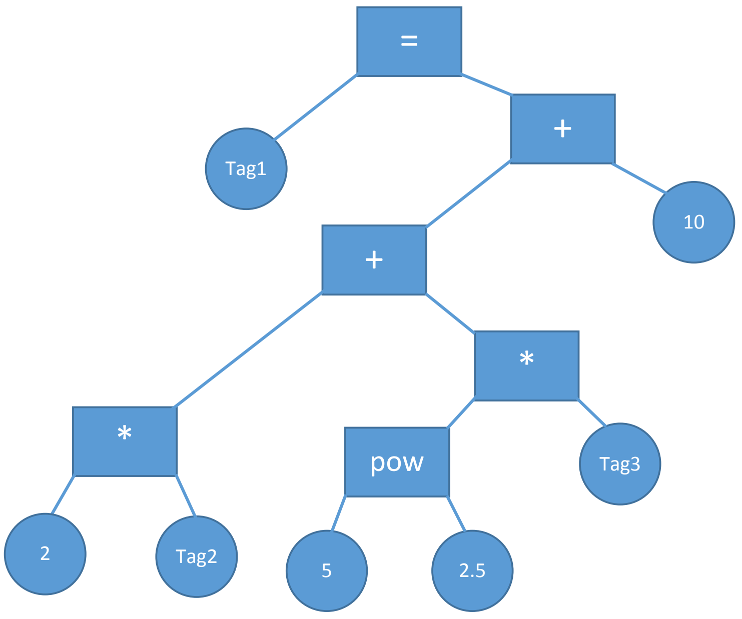
The abstract syntax tree (AST) is a tree list representation of an *abstract syntactic structure* of a text (source code) written in a *formal language*, where each interior node and the root node represent operations and the leaf nodes represent operands. The expression:

$$Tag1 = 2 * Tag2 + 5^{2.5} * Tag3 + 10$$

has the abstract syntax tree representation from the picture below. The circles represent the leaves of the tree, which are constant numbers or pointers to variables. The squares represent the root node and interior nodes, which are operations.

The order of construction is important. A precedence number is assigned to different types of operations. For example, the multiplication is done before addition. The assignment operation, of the result on the right of “=” to *Tag1*, is the last operation, with the lowest preceding level.

The evaluation of the expression above starts from the top of the tree. The query is going down on the branches of the tree, to the leaves level. The operations are performed from bottom to top. Once an operation is completed, the result is ready, as a left or right operand, for the next operation above.



Very complicated expressions can be represented as abstract syntax trees in the memory of the computer. The execution is fast because all operations are executed in the memory.

3) If… then… else… instruction

if… then… else…; is the general form for a logic instruction. The instruction is composed of 3(three) statements, *if…A…* statement, *then…B…* statement and *else… C…* statement. If the value of expression *A* is positive (not 0 and not negative) then the expression *B* will be executed, else the expression *C*.

A program consists of one or more consecutive instructions. Consecutive instructions are separated by semicolon, ";". For any instruction the *if…* and *else…* statements can be omitted. In these cases the expression is interpreted as a *then…* statement that is always executed. No *then…* keyword is necessary.

A special case is the use of curly brackets, "{ }". If an open curly bracket is detected after an *if…* statement then the next instructions, up to the next close bracket, "}", are executed if the previous *if…* statement expression is true (positive), and are not executed if the expression is false (zero or negative).

4) Operations list

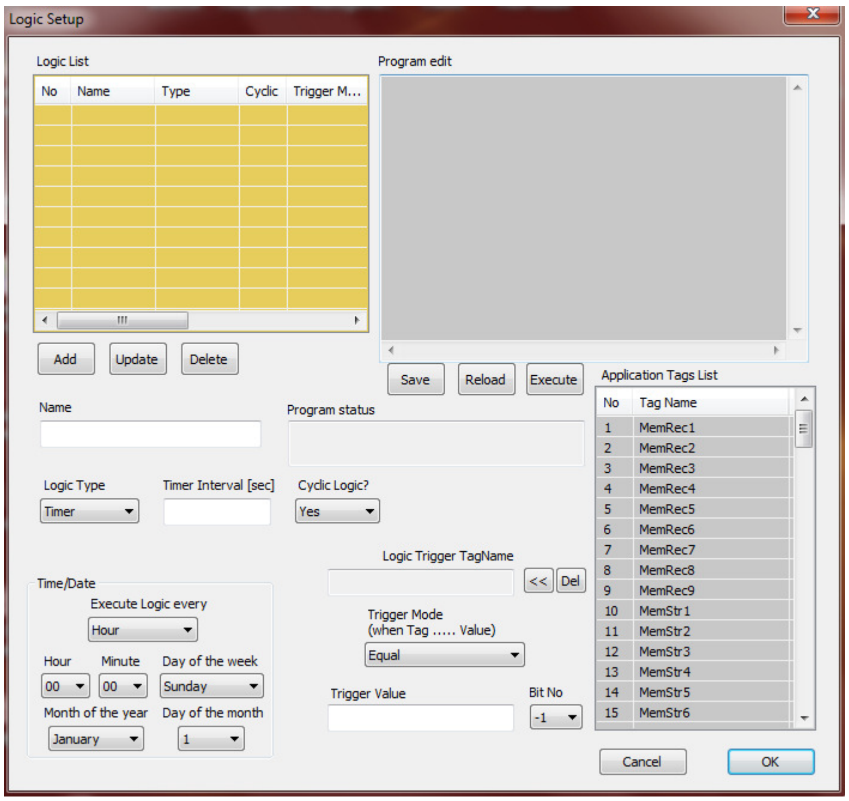
The operations supported in the FeSCADA logic programs are listed in the table below. The precedence number, from 1 to 8, is indicated for each operation. Eight is the first operation evaluated and 1 is the last. In case of an open parenthesis, "(", then the operations inside the round brackets will have a higher precedence, with a level up, 8+x, where x is the typical precedence number. Each open round bracket will increase the level of precedence, (N+1)*8+x. Each close round bracket will decrease the level, (N-1)*8+x. The operations can be used in any of the 3 statements of the *if… then… else… ;* instruction.

No	Precedence	Operation	Explanations
1	5	+	Addition
2	5	-	Subtraction
3	6	*	Multiplication
4	6	/	Division
5	6	%	Modulo division
6	7	$	Power, a$x -> a^x
7	7	<<	Shift Left – bit level, example: a<<2
8	7	>>	Shift Right – bit level, example: a>>1
9	7	@	Bit number, a@3 is the value of bit number 3
10	4	<	Less – comparison (result 0 or 1), example: if (a < 10) then… ;
11	4	<=	Less or equal – comparison
12	4	>	Greater – comparison
13	4	>=	Greater or equal – comparison
14	4	==	Equal – comparison
15	4	!=	Not equal – comparison
16	4	&	AND at bit level, example: if (a & 3) > 0 then… ;
17	4	|	OR at bit level
18	4	~	NOT at bit level, example: ~3 as a byte is 11111100
19	4	^	XOR at bit level
20	2	AND	AND logic, example: if (Tag1>10 AND Tag1<20) then Tag2=2*Tag1;
21	2	OR	OR logic
22	3	NOT	NOT logic, one operand, example: if NOT Tag1>10 then… ;

23	1	=	Assignment, example: Tag1 = 2+5*Tag2;
24	8	SIN	sin(x), the next operations have only one operand
25	8	COS	cos(x), example: Tag1 = 2*COS(Tag2/180*3.1415);
26	8	EXP	exp(x), e^x
27	8	SQRT	Sqrt(x), $\sqrt{x}$
28	8	LOG	log(x)
29	8	LOG10	log10(x)
30	8	ABS	abs(x)
31	8	TAN	tan(x)
32	8	ATAN	atan(x)
33	8	ASIN	asin(x)
34	8	ACOS	acos(x)
35	8	RAND	RAND(X) is generating a random number between 0 and X. If X is integer the result is integer. If X is a real number the result is a real number.
36	8	BCD	BCD(X) is returning an integer Binary Coded Decimal (1234 => 0001 0010 0011 0100)
37	8	DCB	DCB(X) is the reverse of BCD, returning an integer Decimal Coded Binary (0000 0011 0100 0010 => 0342)

5) Programs setup and run

To open the **Programs** setup dialog window, select from the menu **Utils->Programs**. A notification message will inform if no data was previously configured. Press **OK** to open the dialog window. The **Logic List** is on the top left of the window. The **Program Edit** text box is on the top right corner. The **Application Tags List** is in the bottom right corner.



Logic Type: Timer
Timer Interval [sec]:
Cyclic Logic?: Yes
Execute Logic every: Hour
Hour: 00 Minute: 00 Day of the week: Sunday
Month of the year: January Day of the month: 1
Trigger Mode (when Tag ..... Value): Equal
Trigger Value:
Bit No: -1

Application Tags List

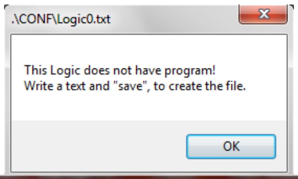
No	Tag Name
1	MemRec1
2	MemRec2
3	MemRec3
4	MemRec4
5	MemRec5
6	MemRec6
7	MemRec7
8	MemRec8
9	MemRec9
10	MemStr1
11	MemStr2
12	MemStr3
13	MemStr4
14	MemStr5
15	MemStr6

To setup a logic program first type a name in the **Name** text box. Then choose between *Timer* and *Time/Date* for the **Logic Type**.

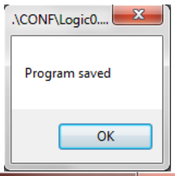
If the choice is *Timer*, a trigger tag can be selected from the **Application List** with the **<<** button.

If no trigger tag is selected the program will be executed by default at maximum speed, 10 times per second (100ms cycle).

If a trigger tag is selected then a **Timer Interval [sec]** is required. The timer is started when the trigger tag evaluation is true in comparison with the **Trigger Value**. After the **Time Interval [sec]** is passing the program will be executed. If the **Cyclic Logic?** is *Yes* the execution will repeat at this time interval. If it is *No*, it will execute only once. The trigger tag has to change its value and then retake the trigger value again for the program to execute again.



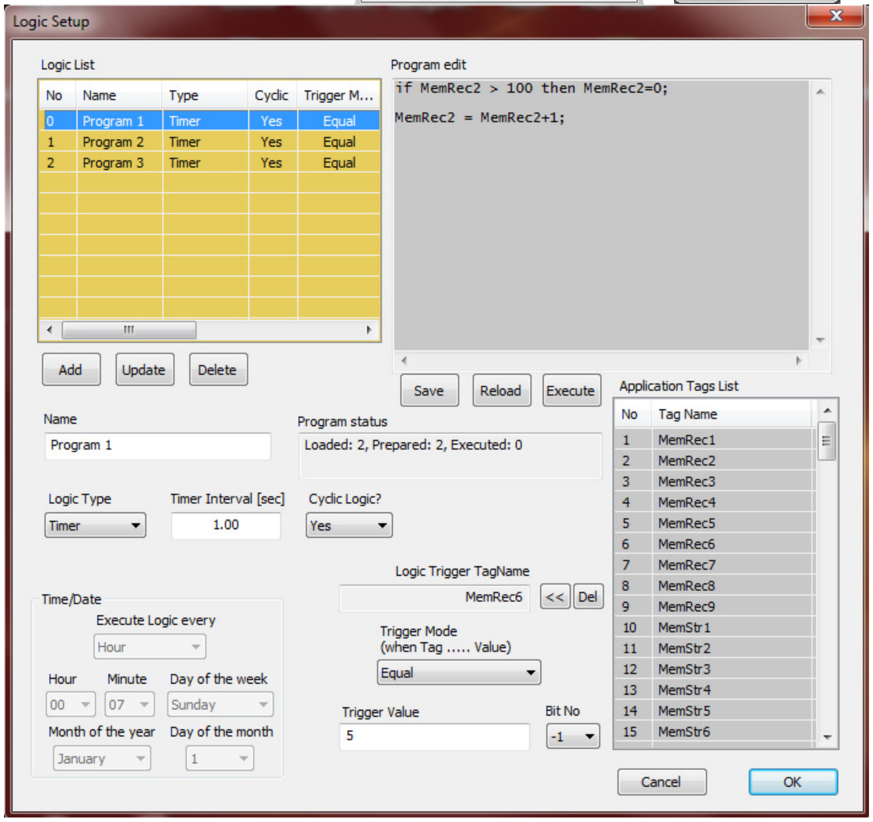
This Logic does not have program!
Write a text and "save", to create the file.



Program saved

If the **Logic Type** choice is *Date/Time*, the user can select to execute the logic program every: *Hour*, *Day*, *Week*, *Month*, *Year*.

Press **Add** to insert the new logic in the **Logic List** table. Select the logic with the mouse. A warning window will inform that there is no program saved for this logic. Press OK and write a logic program in the **Program Edit** text box. Press **Save** to save the program in a text file. A pop up window will confirm.



No	Name	Type	Cyclic	Trigger M...
0	Program 1	Timer	Yes	Equal
1	Program 2	Timer	Yes	Equal
2	Program 3	Timer	Yes	Equal

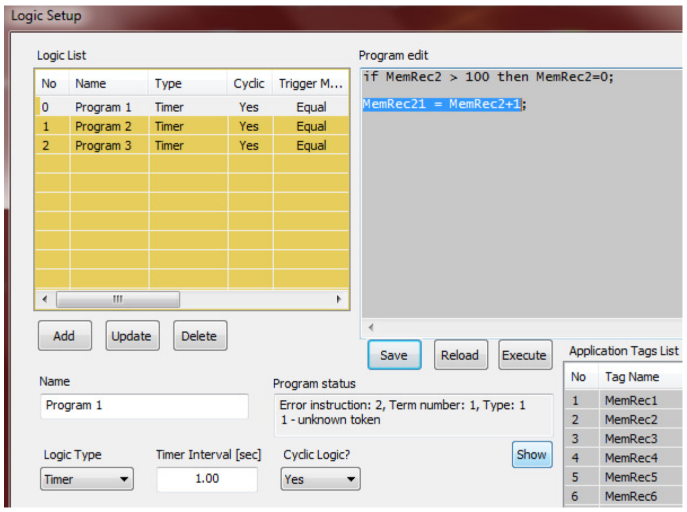
```
if MemRec2 > 100 then MemRec2=0;
MemRec2 = MemRec2+1;
```

Name: Program 1
Program status: Loaded: 2, Prepared: 2, Executed: 0
Logic Type: Timer
Timer Interval [sec]: 1.00
Cyclic Logic?: Yes
Logic Trigger TagName: MemRec6
Trigger Mode (when Tag ..... Value): Equal
Trigger Value: 5
Bit No: -1
Execute Logic every: Hour
Hour: 00
Minute: 07
Day of the week: Sunday
Month of the year: January
Day of the month: 1

No	Tag Name
1	MemRec1
2	MemRec2
3	MemRec3
4	MemRec4
5	MemRec5
6	MemRec6
7	MemRec7
8	MemRec8
9	MemRec9
10	MemStr1
11	MemStr2
12	MemStr3
13	MemStr4
14	MemStr5
15	MemStr6

If the program was just added to the **Logic List** table, the changes are active after closing **Logic Setup** window by pressing OK, and after FeSCADA is restarted. After restart, if the program is modified and saved, it can be reloaded with the **Reload** button. No restart is necessary. Also the program can be executed by pressing **Execute** button, even if the execution condition was not met yet.

Program Status is showing the number of program instructions loaded and prepared for execution. The last information indicates how many instructions were executed at the last run.



No	Name	Type	Cyclic	Trigger M...
0	Program 1	Timer	Yes	Equal
1	Program 2	Timer	Yes	Equal
2	Program 3	Timer	Yes	Equal

```
if MemRec2 > 100 then MemRec2=0;
MemRec21 = MemRec2+1;
```

Name: Program 1
Program status: Error instruction: 2, Term number: 1, Type: 1
1 - unknown token
Logic Type: Timer
Timer Interval [sec]: 1.00
Cyclic Logic?: Yes

No	Tag Name
1	MemRec1
2	MemRec2
3	MemRec3
4	MemRec4
5	MemRec5
6	MemRec6

If an instruction in the program has any syntax error, the program is not executed. **Program Status** will specify the instruction number, the term number in instruction and an error type. The **Show** button will become visible. If the user is pressing the **Show** button, the instruction with error is highlighted in blue. In the picture on the right the tag *MemRec21* was not found in the **Application Tags List**.

The user can correct the error, save the program and reload it. If the program is correct it will start running immediately.

The table below shows the syntax errors detected by FeSCADA interpreter.

Error type	Explanations
1	unknown token
2	no left operand
3	no right operand
4	no left variable for assign (=) operation
5	variable type is not integer or float
6	too many open parenthesis (
7	too many close parenthesis)

6) Application examples

There are many possible applications that can be done using the **Programs** utility of FeSCADA software. In the table below some examples are provided.

Description	Program example

Boolean logic	if sensor1 AND sensor2 AND NOT sensor3
	  then Start_Pump = 1
	  else Start_Pump = 0;

Scaling tags after mathematical equations
 	
(with other logic decisions for the limits of the result)
	Tag1 = 10*EXP(Tag2) +10*LOG(Tag3);
	if Tag1 > 100
	{
	  Tag1 = 100;
	  Tag2 = Tag2 – 0.1;
	  if Tag3 > 0 then Tag3 = -Tag3;
	}

Complex mathematical operations	Angle1 = ASIN(SIN(tagA)*COS(tagB) + COS(tagA)*SIN(tagB));

Special alarms
 	
If the user is defining the integer memory tags: *SysHour*, *SysMinute*, *SysSecond*, *SysDayOfMonth*, *SysDayOfWeek*, the program will write in them the actual values read from the real time clock of the computer. These tags can be used in numerous logic programs.
	if SysDayOfWeek >=1 AND SysDayOfWeek <=5
	{
	  if SysHour < 8 OR SysHour > 16
	  {
	    hours = 0;
	    Production_Alarm = 0;
	  }
	  if SysHour >= 8 AND SysHour <= 16
	  {
	    hours = SysHour - 7;
	    Planed_Production = 1000*hours;
	    if (Planed_Production – Current_Production) > 500
	    then Production_Alarm = 1
	    else Production_Alarm = 0;
	  }
	}

Logic at bit level	if Temperature1 > 100 then Alarms = Alarms | (1<<5);
	if Temperature1 < 95 then Alarms = Alarms & (~(1<<5));

7) Conclusions

With FeSCADA software it is possible to setup and run logic programs. The programs can be run continuously, at special time/dates, or as a logic comparison result of a trigger tag value with a constant. Tags can be used in any logic expression, at the bit level or value level, and the result of complex expressions can be assigned to any numeric tag.