Logix5000 Controllers Structured Text

Allen-Bradley

Catalog Numbers 1756 ControlLogix, 1769 CompactLogix, 1789 SoftLogix, 1794 FlexLogix, PowerFlex 700S with DriveLogix

Programming Manual

Allen-Bradley • Rockwell Software

Rockwell Automation

Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls (publication SGI-1.1 available from your local Rockwell Automation sales office or online at http://www.rockwellautomation.com/literature/) describes some important differences between solid state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING	Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.
ATTENTION	Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence
SHOCK HAZARD	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.
BURN HAZARD	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

Summary of Changes

This manual contains new and updated information.

IMPORTANT	RSLogix 5000 programming software is now known as Studio 5000™ Logix Designer application, a component of Studio 5000 Engineering and Design Environment.

Changes throughout this revision are marked by change bars, as shown in the margin of this page.

Notes:

Studio 5000 Engineering and Design Environment and Logix Designer Application

The Studio 5000™ Engineering and Design Environment combines engineering and design elements into a common environment. The first element in the Studio 5000 environment is the Logix Designer application. The Logix Designer application is the rebranding of RSLogix™ 5000 software and will continue to be the product to program Logix5000™ controllers for discrete, process, batch, motion, safety, and drive-based solutions.

The Studio 5000 environment is the foundation for the future of Rockwell Automation® engineering design tools and capabilities. It is the one place for design engineers to develop all the elements of their control system.

In This Manual

This manual shows how to program Logix5000 controllers with the structured text programming language. This manual is one of a set of related manuals that show common procedures for programming and operating Logix5000 controllers. For a complete list of common procedures manuals, see the *Logix 5000 Controllers Common Procedures Programming Manual*, publication 1756-PM001.

The term Logix5000 controller refers to any controller that is based on the Logix5000 operating system, such as:

- CompactLogix controllers
- ControlLogix controllers
- DriveLogix controllers
- FlexLogix controllers
- SoftLogix5800 controllers

How to Use this Manual

Some text is formatted differently from the rest of the text.

Text that is	Identifies	For example	Means
Italic	the actual name of an item that you see on your screen or in an example	Right-click *User-Defined* …	Right-click the item that is named User-Defined.
courier	information that you must supply based on your application (a variable)	Right-click *name_of_program* …	You must identify the specific program in your application. Typically, it is a name or variable that you have defined.
enclosed in brackets	a keyboard key	Press [Enter].	Press the Enter key.

Program Structured Text

Introduction

Structured text is a textual programming language that uses statements to define what to execute.

- Structured text is not case sensitive.
- Use tabs and carriage returns (separate lines) to make your structured text easier to read. They have no effect on the execution of the structured text.

Structured text is not case sensitive. Structured text can contain these components:

Term	Definition		Examples
Assignment (see page 11)	Use an assignment statement to assign values to tags. The := operator is the assignment operator. Terminate the assignment with a semi colon ";".		tag := expression;
Expression (see page 13)	An expression is part of a complete assignment or construct statement. An expression evaluates to a number (numerical expression) or to a true or false state (BOOL expression). An expression contains:		
	Tags	A named area of the memory where data is stored (BOOL, SINT,INT,DINT, REAL, string).	value1
	Immediates	A constant value.	4
	Operators	A symbol or mnemonic that specifies an operation within an expression.	tag1 + tag2 tag1 >= value1
	Functions	When executed, a function yields one value. Use parentheses to contain the operand of a function. Even though their syntax is similar, functions differ from instructions in that functions can only be used in expressions. Instructions cannot be used in expressions.	function(tag1)

Term	Definition	Examples

Instruction
 	
(see page 20)
	An instruction is a standalone statement.
	
	An instruction uses parenthesis to contain its operands.
	
	Depending on the instruction, there can be zero, one, or multiple operands.
	
	When executed, an instruction yields one or more values that are part of a data structure.
	
	Terminate the instruction with a semi colon ";".
	
	Even though their syntax is similar, instructions differ from functions in that instructions cannot be used in expressions. Functions can only be used in expressions.
		instruction();
		
		instruction(operand);
		
		instruction(operand1, operand2,operand3);

Construct (see page 21)	A conditional statement used to trigger structured text code (i.e, other statements). Terminate the construct with a semi colon ";".	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT
Comment (see page page 37)	Text that explains or clarifies what a section of structured text does. • Use comments to make it easier to interpret the structured text. • Comments do not affect the execution of the structured text. • Comments can appear anywhere in structured text.	//*comment* (**start of comment . . . end of comment**) /**start of comment . . . end of comment**/

IMPORTANT Use caution when copying and pasting components between different versions of the Logix Designer application. The application only supports pasting to the same version or newer version. Pasting to a prior version of the application is not supported. When pasting to a prior version, the paste action may succeed but the results may not be as intended.

Assignments

Use an assignment to change the value stored within a tag. An assignment has this syntax:

tag := *expression* ;

where:

Component	Description	
tag	Represents the tag that is getting the new value. The tag must be a BOOL, SINT, INT, DINT, or REAL.	
:=	Is the assignment symbol.	
expression	Represents the new value to assign to the tag.	
	If tag is this data type	**Use this type of expression**
	BOOL	BOOL expression
	SINT INT DINT REAL	Numeric expression
;	Ends the assignment.	

The tag retains the assigned value until another assignment changes the value.

The expression can be simple, such as an immediate value or another tag name, or the expression can be complex and include several operators and/or functions.

TIP I/O module data updates asynchronously to the execution of logic. If you reference an input multiple times in your logic, the input could change state between separate references. If you need the input to have the same state for each reference, buffer the input value and reference that buffer tag.

Specify a Non-retentive Assignment

The non-retentive assignment is different from the regular assignment described above in that the tag in a non-retentive assignment is reset to zero each time the controller:

- enters the Run mode.
- leaves the step of an SFC if you configure the SFC for Automatic reset. (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

A non-retentive assignment has this syntax:

tag [:=] *expression* ;

where:

Component	Description	
tag	Represents the tag that is getting the new value. The tag must be a BOOL, SINT, INT, DINT, or REAL.	
[:=]	Is the non-retentive assignment symbol	
expression	Represents the new value to assign to the tag.	
	If tag is this data type	**Use this type of expression**
	BOOL	BOOL expression
	SINT INT DINT REAL	Numeric expression
;	Ends the assignment.	

Assign an ASCII Character to a String

Use the assignment operator to assign an ASCII character to an element of the DATA member of a string tag. To assign a character, specify the value of the character or specify the tag name, DATA member, and element of the character.

This is OK	This is not OK
string1.DATA[0]:= 65;	string1.DATA[0] := A;
string1.DATA[0]:= string2.DATA[0];	string1 := string2;

To add or insert a string of characters to a string tag, use either of these ASCII string instructions.

To	Use this instruction
Add characters to the end of a string	CONCAT
insert characters into a string	INSERT

Expressions

An expression is a tag name, equation, or comparison. To write an expression, use any of these elements.

- Tag name that stores the value (variable)
- Number that you enter directly into the expression (immediate value)
- Functions, such as: ABS, TRUNC
- Operators, such as: +, -, <, >, And, Or

As you write expressions, follow these general rules.

- Use any combination of upper-case and lower-case letter. For example, these three variations of “AND” are acceptable: AND, And, and.
- For more complex requirements, use parentheses to group expressions within expressions. This makes the whole expression easier to read and ensures that the expression executes in the desired sequence.

IMPORTANT You may add user comments inline. Therefore, local language switching does not apply to your programming language.

In structured text, you use two types of expressions:

BOOL expression: An expression that produces either the BOOL value of 1 (true) or 0 (false).

- A bool expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, tag1>65.
- A simple bool expression can be a single BOOL tag.
- Typically, you use bool expressions to condition the execution of other logic.

Numeric expression: An expression that calculates an integer or floating-point value.

- A numeric expression uses arithmetic operators, arithmetic functions, and bitwise operators. For example, tag1+5.
- Often, you nest a numeric expression within a bool expression. For example, (tag1+5)>65.

Use the following table to choose operators for your expressions:

If you want to	Then
Calculate an arithmetic value	Use Arithmetic Operators and Functions on page 15.
Compare two values or strings	Use Relational Operators on page 16.
Check if conditions are true or false	Use Logical Operators on page 18.
Compare the bits within values	Use Bitwise Operators on page 19.

Use Arithmetic Operators and Functions

You can combine multiple operators and functions in arithmetic expressions.

Arithmetic operators calculate new values.

To	Use this operator	Optimal data type
Add	+	DINT, REAL
Subtract/negate	-	DINT, REAL
Multiply	*	DINT, REAL
Exponent (x to the power of y)	**	DINT, REAL
Divide	/	DINT, REAL
Modulo-divide	MOD	DINT, REAL

Arithmetic functions perform math operations. Specify a constant, a non-boolean tag, or an expression for the function.

For	Use this function	Optimal data type
Absolute value	ABS (*numeric_expression*)	DINT, REAL
Arc cosine	ACOS (*numeric_expression*)	REAL
Arc sine	ASIN (*numeric_expression*)	REAL
Arc tangent	ATAN (*numeric_expression*)	REAL
Cosine	COS (*numeric_expression*)	REAL
Radians to degrees	DEG (*numeric_expression)*	DINT, REAL
Natural log	LN (*numeric_expression*)	REAL
Log base 10	LOG (*numeric_expression*)	REAL
Degrees to radians	RAD (*numeric_expression*)	DINT, REAL
Sine	SIN (*numeric_expression*)	REAL
Square root	SQRT (*numeric_expression*)	DINT, REAL
Tangent	TAN (*numeric_expression*)	REAL
Truncate	TRUNC (*numeric_expression*)	DINT, REAL

For example:

Use this format	Example	
	For this situation	**You'd write**
value1 operator value2	If gain_4 and gain_4_adj are DINT tags and your specification says: "Add 15 to gain_4 and store the result in gain_4_adj."	gain_4_adj := gain_4+15;
operator value1	If alarm and high_alarm are DINT tags and your specification says: "Negate high_alarm and store the result in alarm."	alarm:= -high_alarm;
function(numeric_expression)	If overtravel and overtravel_POS are DINT tags and your specification says: "Calculate the absolute value of overtravel and store the result in overtravel_POS."	overtravel_POS := ABS(overtravel);
value1 operator (function((value2+value3)/2)	If adjustment and position are DINT tags and sensor1 and sensor2 are REAL tags and your specification says: "Find the absolute value of the average of sensor1 and sensor2, add the adjustment, and store the result in position."	position := adjustment + ABS((sensor1 + sensor2)/2);

Use Relational Operators

Relational operators compare two values or strings to provide a true or false result. The result of a relational operation is a BOOL value.

If the comparison is	The result is
True	1
False	0

Use these relational operators.

For this comparison:	Use this operator:	Optimal Data Type:
Equal	=	DINT, REAL, string
Less than	<	DINT, REAL, string
Less than or equal	<=	DINT, REAL, string
Greater than	>	DINT, REAL, string
Greater than or equal	>=	DINT, REAL, string
Not equal	<>	DINT, REAL, string

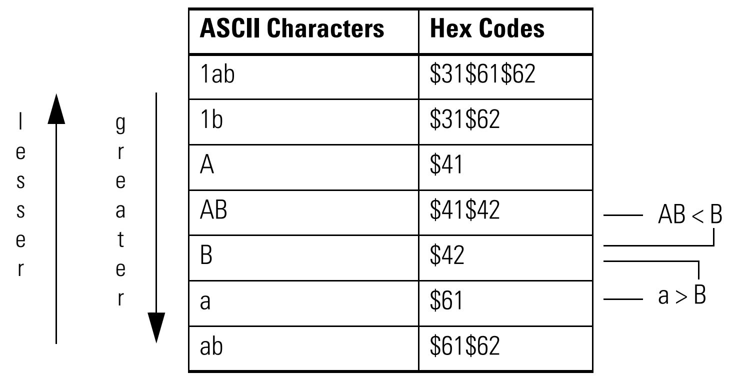
For example:

Use this format	**Example**	
	For this situation	**You’d write**
value1 operator value2	If temp is a DINT tag and your specification says: “If temp is less than 100· then…”	IF temp<100 THEN...
stringtag1 operator stringtag2	If bar_code and dest are string tags and your specification says: “If bar_code equals dest then…”	IF bar_code=dest THEN...
char1 operator char2 To enter an ASCII character directly into the expression, enter the decimal value of the character.	If bar_code is a string tag and your specification says: “If bar_code.DATA[0] equals ’A’ then…”	IF bar_code.DATA[0]=65 THEN...
bool_tag := *bool_expressions*	If count and length are DINT tags, done is a BOOL tag, and your specification says ”If count is greater than or equal to length, you are done counting.”	done := (count >= length);

How Strings Are Evaluated

The hexadecimal values of the ASCII characters determine if one string is less than or greater than another string.

- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.



ASCII Characters	**Hex Codes**
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

- Strings are equal if their characters match.
- Characters are case sensitive. Upper case “A” ($41) is *not* equal to lower case “a” ($61).

Use Logical Operators

Logical operators let you check if multiple conditions are true or false. The result of a logical operation is a BOOL value:

If the comparison is	The result is
True	1
False	0

Use these logical operators:

For	Use this operator	Data Type
Logical AND	&, AND	BOOL
Logical OR	OR	BOOL
Logical exclusive OR	XOR	BOOL
Logical complement	NOT	BOOL

For example:

Use this format	Example	
	For this situation	You'd write
BOOLtag	If photoeye is a BOOL tag and your specification says: "If photoeye_1 is on then…"	IF photoeye THEN...
NOT *BOOLtag*	If photoeye is a BOOL tag and your specification says: "If photoeye is off then…"	IF NOT photoeye THEN...
expression1 & *expression2*	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: "If photoeye is on and temp is less than 100· then…".	IF photoeye & (temp<100) THEN...
expression1 OR *expression2*	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: "If photoeye is on or temp is less than 100· then…".	IF photoeye OR (temp<100) THEN...
expression1 XOR *expression2*	If photoeye1 and photoeye2 are BOOL tags and your specification says: "If: • photoeye1 is on while photoeye2 is off or • photoeye1 is off while photoeye2 is on then…"	IF photoeye1 XOR photoeye2 THEN...
BOOLtag := *expression1* & *expression2*	If photoeye1 and photoeye2 are BOOL tags, open is a BOOL tag, and your specification says: "If photoeye1 and photoeye2 are both on, set open to true".	open := photoeye1 & photoeye2;

Use Bitwise Operators

Bitwise operators manipulate the bits within a value based on two values.

For	Use this operator	Optimal Data Type
Bitwise AND	&, AND	DINT
Bitwise OR	OR	DINT
Bitwise exclusive OR	XOR	DINT
Bitwise complement	NOT	DINT

For example:

Use this format	Example	
	For this situation	You'd write
value1 operator value2	If input1, input2, and result1 are DINT tags and your specification says: "Calculate the bitwise result of input1 and input2. Store the result in result1."	result1 := input1 AND input2;

Determine the Order of Execution

The operations you write into an expression are performed in a prescribed order, not necessarily from left to right.

- Operations of equal order are performed from left to right.
- If an expression contains multiple operators or functions, group the conditions in parenthesis "( )". This ensures the correct order of execution and makes it easier to read the expression.

Order	Operation
1.	( )
2.	function (…)
3.	**
4.	- (negate)
5.	NOT
6.	*, /, MOD
7.	+, - (subtract)
8.	<, <=, >, >=
9.	=, <>
10.	&, AND
11.	XOR
12.	OR

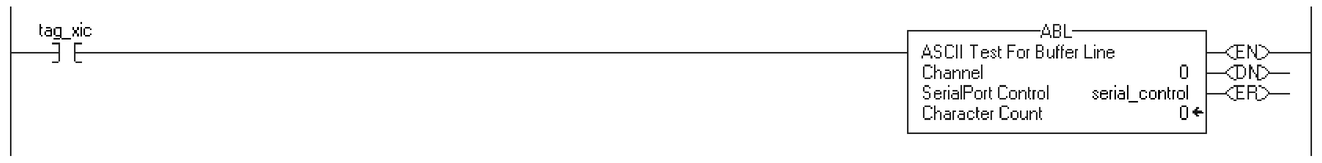
Instructions

Structured text statements can also be instructions. A structured text instruction executes each time it is scanned. A structured text instruction within a construct executes every time the conditions of the construct are true. If the conditions of the construct are false, the statements within the construct are not scanned. There is no rung-condition or state transition that triggers execution.

This differs from function block instructions that use EnableIn to trigger execution. Structured text instructions execute as if EnableIn is always set.

This also differs from relay ladder instructions that use rung-condition-in to trigger execution. Some relay ladder instructions only execute when rung-condition-in toggles from false to true. These are transitional relay ladder instructions. In structured text, instructions will execute each time they are scanned unless you pre-condition the execution of the structured text instruction.

For example, the ABL instruction is a transitional instruction in relay ladder. In this example, the ABL instruction only executes on a scan when tag_xic transitions from cleared to set. The ABL instruction does not execute when tag_xic stays set or when tag_xic is cleared.



In structured text, if you write this example as:

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

the ABL instruction will execute every scan that tag_xic is set, not just when tag_xic transitions from cleared to set.

If you want the ABL instruction to execute only when tag_xic transitions from cleared to set, you have to condition the structured text instruction. Use a one shot to trigger execution.

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);


IF (osri_1.OutputBit) THEN
        ABL(0,serial_control);
END_IF;
```

Constructs

Constructs can be programmed singly or nested within other constructs.

If you want to	Use this construct
Do something if or when specific conditions occur	IF...THEN
Select what to do based on a numerical value	CASE...OF
Do something a specific number of times before doing anything else	FOR...DO
Keep doing something as long as certain conditions are true	WHILE...DO
Keep doing something until a condition is true	REPEAT...UNTIL

Some Key Words Are Reserved for Future Use

These constructs are not available:

- GOTO
- REPEAT

The Logix Designer application will not let you use them.

IF...THEN

Use IF…THEN to do something if or when specific conditions occur.

Operands:

```
IF bool_expression THEN
        <statement>;
END_IF;
```

Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	Tag Expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

Description:

The syntax is:

```
          IF bool_expression1 THEN
                  <statement >;          ◄———— Statements to execute when
                          .                    bool_expression1 is true
                          .
                          .
Optional  ELSIF bool_expression2 THEN
                  <statement>;           ◄———— Statements to execute when
                          .                    bool_expression2 is true
                          .
                          .
Optional  ELSE
                  <statement>;           ◄———— Statements to execute when
                          .                    both expressions are false
                          .
                          .
          END_IF;
```

To use ELSIF or ELSE, follow these guidelines.

1. To select from several possible groups of statements, add one or more ELSIF statements.
    - Each ELSIF represents an alternative path.
    - Specify as many ELSIF paths as you need.
    - The controller executes the first true IF or ELSIF and skips the rest of the ELSIFs and the ELSE.

2. To do something when all of the IF or ELSIF conditions are false, add an ELSE statement.

This table summarizes combinations of IF, THEN, ELSIF, and ELSE.

If you want to	And	Then use this construct
Do something if or when conditions are true	Do nothing if conditions are false	IF…THEN
	Do something else if conditions are false	IF…THEN…ESLE
Choose from alternative statements (or groups of statements) based on input conditions	Do nothing if conditions are false	IF…THEN…ELSIF
	Assign default statements if all conditions are false	IF…THEN…ELSIF…ELSE

Arithmetic Status Flags: Not affected

Fault Conditions: None

Example 1: IF…THEN

If you want this	Enter this structured text
IF rejects > 3 then	IF rejects > 3 THEN
conveyor = off (0)	conveyor := 0;
alarm = on (1)	alarm := 1;
	END_IF;

Example 2: IF…THEN…ELSE

If you want this	Enter this structured text
If conveyor direction contact = forward (1) then	IF conveyor_direction THEN
light = off	light := 0;
Otherwise light = on	ELSE
	light [:=] 1;
	END_IF;

The [:=] tells the controller to clear light whenever the controller:

- enters the Run mode.
- leaves the step of an SFC if you configure the SFC for Automatic reset. (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

Example 3: IF…THEN…ELSIF

If you want this	Enter this structured text
If sugar low limit switch = low (on) and sugar high limit switch = not high (on) then	IF Sugar.Low & Sugar.High THEN
inlet valve = open (on)	Sugar.Inlet [:=] 1;
Until sugar high limit switch = high (off)	ELSIF NOT(Sugar.High) THEN
	Sugar.Inlet := 0;
	END_IF;

The [:=] tells the controller to clear Sugar.Inlet whenever the controller:

- enters the Run mode.
- leaves the step of an SFC if you configure the SFC for Automatic reset. (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

Example 4: IF…THEN…ELSIF…ELSE

If you want this	Enter this structured text
If tank temperature > 100	IF tank.temp > 200 THEN
then pump = slow	pump.fast :=1; pump.slow :=0; pump.off :=0;
If tank temperature > 200	ELSIF tank.temp > 100 THEN
then pump = fast	pump.fast :=0; pump.slow :=1; pump.off :=0;
otherwise pump = off	ELSE
	pump.fast :=0; pump.slow :=0; pump.off :=1;
	END_IF;

CASE...OF

Use CASE to select what to do based on a numerical value.

Operands:

```
CASE numeric_expression OF
        selector1: statement;
        selectorN: statement;
ELSE
        statement;
END_CASE;
```

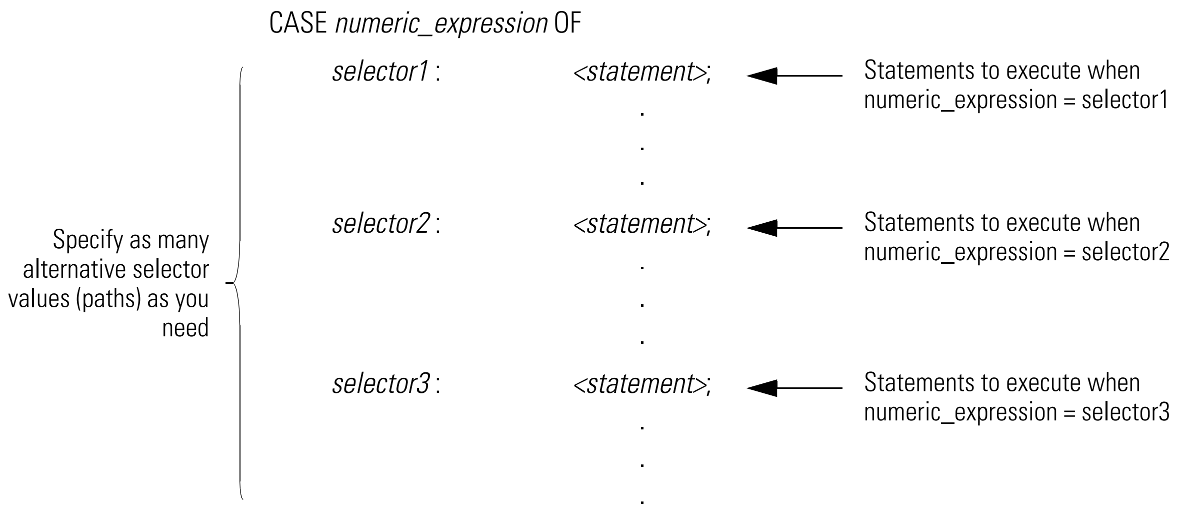
Structured Text

Operand	Type	Format	Enter
numeric_ expression	SINT INT DINT REAL	Tag Expression	Tag or expression that evaluates to a number (numeric expression)
selector	SINT INT DINT REAL	Immediate	Same type as *numeric_expression*

IMPORTANT If you use REAL values, use a range of values for a selector because a REAL value is more likely to be within a range of values than an exact match of one, specific value.

Description:

The syntax is:



```
CASE numeric_expression OF
    selector1 :          <statement>;
                              .
                              .
                              .
    selector2 :          <statement>;
                              .
                              .
                              .
    selector3 :          <statement>;
                              .
                              .
                              .
```

Statements to execute when numeric_expression = selector1
Statements to execute when numeric_expression = selector2
Statements to execute when numeric_expression = selector3
Specify as many alternative selector values (paths) as you need

```
           ELSE
                                  <statement>;     Statements to execute when
optional                               .           numeric_expression ¼ any
                                       .           selector
                                       .
           END_CASE;
```

The syntax for entering the selector values is:

When selector is	Enter
One value	value: statement
Multiple, distinct values	value1, value2, valueN : <statement> Use a comma (,) to separate each value.
A range of values	value1...valueN : <statement> Use two periods (..) to identify the range.
Distinct values plus a range of values	valuea, valueb, value1...valueN : <statement>

The CASE construct is similar to a switch statement in the C or C++ programming languages. However, with the CASE construct the controller executes only the statements that are associated with the first matching selector value. Execution always breaks after the statements of that selector and goes to the END_CASE statement.

Arithmetic Status Flags: Not affected

Fault Conditions: None

Example

Example

If you want this	Enter this structured text
If recipe number = 1 then	CASE recipe_number OF
Ingredient A outlet 1 = open (1)	1: Ingredient_A.Outlet_1 :=1;
Ingredient B outlet 4 = open (1)	Ingredient_B.Outlet_4 :=1;
If recipe number = 2 or 3 then	2,3: Ingredient_A.Outlet_4 :=1;
Ingredient A outlet 4 = open (1)	Ingredient_B.Outlet_2 :=1;
Ingredient B outlet 2 = open (1)	
If recipe number = 4, 5, 6, or 7 then	4..7: Ingredient_A.Outlet_4 :=1;
Ingredient A outlet 4 = open (1)	Ingredient_B.Outlet_2 :=1;
Ingredient B outlet 2 = open (1)	
If recipe number = 8, 11, 12, or 13 then	8,11..13 Ingredient_A.Outlet_1 :=1;
Ingredient A outlet 1 = open (1)	Ingredient_B.Outlet_4 :=1;
Ingredient B outlet 4 = open (1)	
Otherwise all outlets = closed (0)	ELSE
	Ingredient_A.Outlet_1 [:=]0;
	Ingredient_A.Outlet_4 [:=]0;
	Ingredient_B.Outlet_2 [:=]0;
	Ingredient_B.Outlet_4 [:=]0;
	END_CASE;

The [:=] tells the controller to also clear the outlet tags whenever the controller:

- enters the Run mode.
- leaves the step of an SFC if you configure the SFC for Automatic reset. (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

FOR…DO

Use the FOR…DO loop to do something a specific number of times before doing anything else.

Operands:

FOR *count*:= *initial_value* TO *final_value* BY *increment* DO

    <*statement*>;

END_FOR;

Structured Text

Operand	Type	Format	Description
count	SINT INT DINT	Tag	Tag to store count position as the FOR…DO executes
initial_ value	SINT INT DINT	Tag Expression Immediate	Must evaluate to a number Specifies initial value for count
final_ value	SINT INT DINT	Tag Expression Immediate	Specifies final value for count, which determines when to exit the loop
increment	SINT INT DINT	Tag Expression Immediate	(optional) Amount to increment count each time through the loop If you don't specify an increment, the count increments by 1.

IMPORTANT

Make sure that you do not iterate within the loop too many times in a single scan.

- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

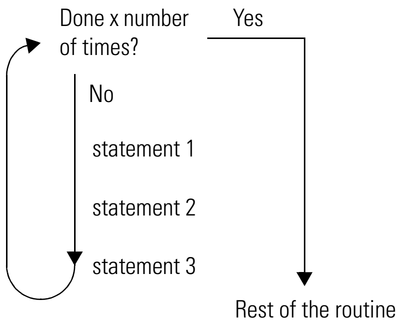
Description: The syntax is:

```
FOR count := initial_value
    TO final_value
    BY increment
    DO
    <statement>;
    IF bool_expression THEN
         EXIT;
    END_IF;
END_FOR;
```

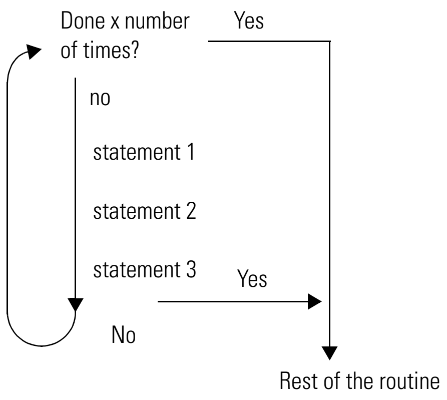
Optional { BY *increment* — If you don’t specify an increment, the loop increments by 1.

Optional { IF *bool_expression* THEN / EXIT; / END_IF; — If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

These diagrams show how a FOR...DO loop executes and how an EXIT statement leaves the loop early.



The FOR…DO loop executes a specific number of times.



To stop the loop before the count reaches the last value, use an EXIT statement.

Arithmetic Status Flags: Not affected

Fault Conditions:

A major fault will occur if	Fault type	Fault code
The construct loops too long	6	1

Example 1:

If you want this	Enter this structured text

Clear bits 0 - 31 in an array of BOOLs:
1. Initialize the subscript tag to 0.
2. Clear array[subscript] . For example, when subscript = 5, clear array[5].
3. Add 1 to subscript.
4. If subscript is £ to 31, repeat 2 and 3.
Otherwise, stop.
	For subscript:=0 to 31 by 1 do
	    array[subscript] := 0;
	End_for;

Example 2:

If you want this	Enter this structured text

A user-defined data type (structure) stores this information about an item in your inventory:
• Barcode ID of the item (string data type)
• Quantity in stock of the item (DINT data type)
An array of the above structure contains an element for each different item in your inventory. You want to search the array for a specific product (use its bar code) and determine the quantity that is in stock.
1. Get the size (number of items) of the Inventory array and store the result in Inventory_Items (DINT tag).
2. Initialize the position tag to 0.
3. If Barcode matches the ID of an item in the array, then:
a. Set the Quantity tag = Inventory[position].Qty. This produces the quantity in stock of the item.
b. Stop.
Barcode is a string tag that stores the bar code of the item for which you are searching. For example, when position = 5, compare Barcode to Inventory[5].ID.
4. Add 1 to position.
5. If position is £ to (Inventory_Items -1), repeat 3 and 4. Since element numbers start at 0, the last element is 1 less than the number of elements in the array.
Otherwise, stop.
	SIZE(Inventory,0,Inventory_Items);
	For position:=0 to Inventory_Items - 1 do
	    If Barcode = Inventory[position].ID then
	        Quantity := Inventory[position].Qty;
	        Exit;
	    End_if;
	End_for;

WHILE…DO

Use the WHILE…DO loop to keep doing something as long as certain conditions are true.

```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

Operands:

Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	Tag Expression	BOOL tag or expression that evaluates to a BOOL value

IMPORTANT Make sure that you do not iterate within the loop too many times in a single scan.

- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

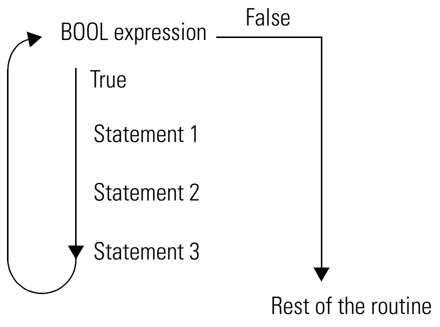
Description:

The syntax is:

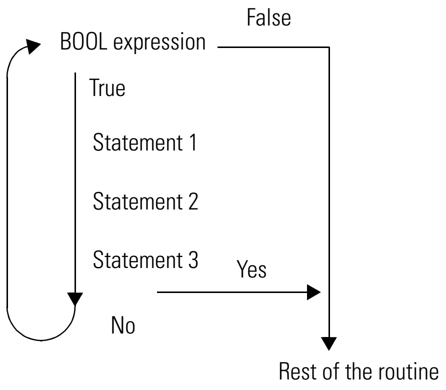
```
WHILE bool_expression1 DO
    <statement>;            ◄──── Statements to execute while
                                  bool_expression1 is true
    IF bool_expression2 THEN
        EXIT;               ◄──── If there are conditions when you want to
    END_IF;                       exit the loop early, use other statements,
                                  such as an IF...THEN construct, to
                                  condition an EXIT statement.
END_WHILE;
```

Optional (IF bool_expression2 THEN … END_IF;)

These diagrams show how a WHILE...DO loop executes and how an EXIT statement leaves the loop early.



While the *bool_expression* is true, the controller executes only the statements within the WHILE…DO loop.



To stop the loop before the conditions are true, use an EXIT statement.

Arithmetic Status Flags:

Not affected

Fault Conditions:

A major fault will occur if	Fault type	Fault code
The construct loops too long	6	1

Example 1:

If you want this	Enter this structured text

The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop.
 	
This differs from the REPEAT...UNTIL loop because the REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.
	`pos := 0;`
	`While ((pos <= 100) & structarray[pos].value <> targetvalue)) do`
	`    pos := pos + 2;`
	`    String_tag.DATA[pos] := SINT_array[pos];`
	`end_while;`

Example 2:

If you want this	Enter this structured text

Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.
 	
1. Initialize Element_number to 0.
2. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).
3. If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.
4. Set String_tag[element_number] = the character at SINT_array[element_number].
5. Add 1 to element_number. This lets the controller check the next character in SINT_array.
6. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.)
7. If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)
8. Go to 3.
	element_number := 0;
	SIZE(SINT_array, 0, SINT_array_size);
	While SINT_array[element_number] <> 13 do
	    String_tag.DATA[element_number] := SINT_array[element_number];
	    element_number := element_number + 1;
	    String_tag.LEN := element_number;
	    If element_number = SINT_array_size then
	        exit;
	    end_if;
	end_while;

REPEAT…UNTIL

Use the REPEAT…UNTIL loop to keep doing something until conditions are true.

Operands:

```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	Tag Expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

IMPORTANT Make sure that you do not iterate within the loop too many times in a single scan.

- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

Description: The syntax is:

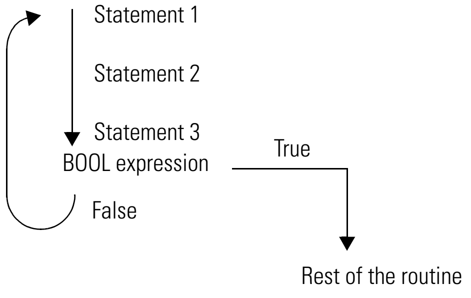
```
REPEAT
    <statement>;

    IF bool_expression2 THEN
        EXIT;
    END_IF;

UNTIL bool_expression1
END_REPEAT;
```

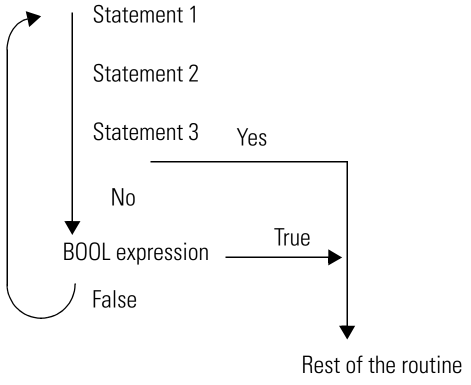
Optional

Statements to execute while bool_expression1 is false

If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

These diagrams show how a REPEAT...UNTIL loop executes and how an EXIT statement leaves the loop early.





While the *bool_expression* is false, the controller executes only the statements within the REPEAT…UNTIL loop.

To stop the loop before the conditions are false, use an EXIT statement.

Arithmetic Status Flags: Not affected

Fault Conditions:

A major fault will occur if	Fault type	Fault code
The construct loops too long	6	1

Example 1:

If you want this	Enter this structured text

The REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again.
 	
This differs from the WHILE...DO loop because the WHILE...DO The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.
	pos := -1;
	REPEAT
	    pos := pos + 2;
	UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))
	end_repeat;

Example 2:

If you want this	Enter this structured text

Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.
 	
1. Initialize Element_number to 0.
 	
2. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).
 	
3. Set String_tag[element_number] = the character at SINT_array[element_number].
 	
4. Add 1 to element_number. This lets the controller check the next character in SINT_array.
 	
5. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.)
 	
6. If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)
 	
7. If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.
 	
Otherwise, go to 3.
	element_number := 0;
	SIZE(SINT_array, 0, SINT_array_size);
	Repeat
	    String_tag.DATA[element_number] := SINT_array[element_number];
	    element_number := element_number + 1;
	    String_tag.LEN := element_number;
	    If element_number = SINT_array_size then
	        exit;
	    end_if;
	Until SINT_array[element_number] = 13
	end_repeat;

Comments

To make your structured text easier to interpret, add comments to it.

- Comments let you use plain language to describe how your structured text works.
- Comments do not affect the execution of the structured text.

Structured text comments are downloaded into controller memory and are available for upload. To add comments to your structured text:

To add a comment	Use one of these formats
On a single line	//*comment*
At the end of a line of structured text	
	(**comment**)
	/**comment**/
Within a line of structured text	(**comment**)
	/**comment**/
That spans more than one line	(**start of comment . . . end of comment**)
	/**start of comment . . . end of comment**/

For example:

Format	Example

//comment	**At the beginning of a line**
	
	//Check conveyor belt direction
	
	IF conveyor_direction THEN...
	
	At the end of a line
	
	ELSE //If conveyor isn't moving, set alarm light
	
	light := 1;
	
	END_IF;

(**comment**)	Sugar.Inlet[:=]1;(*open the inlet*)
	
	IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN...
	
	(*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*)
	
	IF tank.temp > 200 THEN...

/**comment**/	Sugar.Inlet:=0;/*close the inlet*/
	
	IF bar_code=65 /*A*/ THEN...
	
	/*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/
	
	SIZE(Inventory,0,Inventory_Items);

Rockwell Automation Support

Rockwell Automation provides technical information on the Web to assist you in using its products. At http://www.rockwellautomation.com/support/, you can find technical manuals, a knowledge base of FAQs, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools.

For an additional level of technical phone support for installation, configuration, and troubleshooting, we offer TechConnect support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit http://www.rockwellautomation.com/support/.

Installation Assistance

If you experience an anomoly within the first 24 hours of installation, review the information that is contained in this manual. You can contact Customer Support for initial help in getting your product up and running.

United States or Canada	1.440.646.3434
Outside United States or Canada	Use the Worldwide Locator at http://www.rockwellautomation.com/support/americas/phone_en.html, or contact your local Rockwell Automation representative.

New Product Satisfaction Return

Rockwell Automation tests all of its products to ensure that they are fully operational when shipped from the manufacturing facility. However, if your product is not functioning and needs to be returned, follow these procedures.

United States	Contact your distributor. You must provide a Customer Support case number (call the phone number above to obtain one) to your distributor to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for the return procedure.

Documentation Feedback

Your comments will help us serve your documentation needs better. If you have any suggestions on how to improve this document, complete this form, publication RA-DU002, available at http://www.rockwellautomation.com/literature/.

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-PM007D-EN-P - November 2012

Supersedes Publication 1756-PM007C-EN-P - October 2009