



Logix 5000 Controllers Structured Text

1756 ControlLogix, 1756 GuardLogix, 1769 CompactLogix,
1769 Compact GuardLogix, 1789 SoftLogix, 5069
CompactLogix, 5069 Compact GuardLogix, Studio 5000
Logix Emulate



Important User Information

Read this document and the documents listed in the additional resources section about installation, configuration, and operation of this equipment before you install, configure, operate, or maintain this product. Users are required to familiarize themselves with installation and wiring instructions in addition to requirements of all applicable codes, laws, and standards.

Activities including installation, adjustments, putting into service, use, assembly, disassembly, and maintenance are required to be carried out by suitably trained personnel in accordance with applicable code of practice.

If this equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

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.



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.

IMPORTANT Identifies information that is critical for successful application and understanding of the product.

Labels may also be on or inside the equipment to provide specific precautions.



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.



ARC FLASH HAZARD: Labels may be on or inside the equipment, for example, a motor control center, to alert people to potential Arc Flash. Arc Flash will cause severe injury or death. Wear proper Personal Protective Equipment (PPE). Follow ALL Regulatory requirements for safe work practices and for Personal Protective Equipment (PPE).

This manual includes new and updated information. Use these reference tables to locate changed information.

Grammatical and editorial style changes are not included in this summary.

Global changes

This table identifies changes that apply to all information about a subject in the manual and the reason for the change. For example, the addition of new supported hardware, a software design change, or additional reference material would result in changes to all of the topics that deal with that subject.

Change	Topic
Updated Legal notices.	Legal notices on page 8
Updated branding.	Throughout

Summary of changes	Studio 5000 environment	7
Preface	Additional resources	7
	Legal notices	8
	Chapter 1	
Program Structured Text	Structured Text Syntax.....	11
	Structured Text Components: Assignments.....	12
	Specify a non-retentive assignment.....	13
	Assign an ASCII character to a string data member.....	14
	Character string literals	14
	Structured Text Components: Expressions	15
	Use arithmetic operators and functions	16
	Use relational operators	17
	Use logical operators	19
	Use bitwise operators	19
	Determine the order of execution	20
	Structured Text Components: Instructions.....	20
	Structured Text Components: Constructs	21
	IF_THEN	22
	CASE_OF.....	25
	FOR_DO	27
	WHILE_DO	29
	REPEAT_UNTIL	31
	Structured Text Components: Comments	34
Index		

This manual shows how to program Logix 5000 controllers with structured text programming language.

This manual is one of a set of related manuals that show common procedures for programming and operating Logix 5000 controllers.

For a complete list of common procedures manuals, refer to the [Logix 5000 Controllers Common Procedures Programming Manual](#), publication 1756-PM001.

The term Logix 5000 controller refers to any controller based on the Logix 5000 operating system.

•

Studio 5000 environment

The Studio 5000 Automation Engineering & Design Environment® combines engineering and design elements into a common environment. The first element is the Studio 5000 Logix Designer® application. The Logix Designer application is the rebranding of RSLogix 5000® software and will continue to be the product to program Logix 5000™ 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. The Studio 5000 environment is the one place for design engineers to develop all elements of their control system.

Additional resources

These documents contain additional information concerning related Rockwell Automation products.

Resource	Description
LOGIX 5000 Controllers Program Parameters Programming Manual , publication 1756-PM021	Describes how to use program parameters when programming Logix 5000 controllers.
LOGIX 5000 Controllers General Instructions Reference Manual , publication 1756-RM003	Describes the available instructions for a Logix 5000 controller.

Resource	Description
LOGIX 5000 Controllers Process and Drives Instructions Reference Manual , publication 1756-RM006	Describes how to program a Logix 5000 controller for process or drives applications.
LOGIX 5000 Controllers Motion Instruction Set Reference Manual , publication MOTION-RM002	Describes how to program a Logix 5000 controller for motion applications.
Product Certifications website, http://ab.rockwellautomation.com	Provides declarations of conformity, certificates, and other certification details.

You can view or download publications at <http://www.rockwellautomation.com/literature>. To order paper copies of technical documentation, contact your local Rockwell Automation distributor or sales representative.

Legal notices

Rockwell Automation publishes legal notices, such as privacy policies, license agreements, trademark disclosures, and other terms and conditions on the [Legal Notices](#) page of the Rockwell Automation website.

End User License Agreement (EULA)

You can view the Rockwell Automation End User License Agreement (EULA) by opening the license.rtf file located in your product's install folder on your hard drive.

The default location of this file is:

C:\Program Files (x86)\Common Files\Rockwell\license.rtf.

Open Source Software Licenses

The software included in this product contains copyrighted software that is licensed under one or more open source licenses.

You can view a full list of all open source software used in this product and their corresponding licenses by opening the oss_license.txt file located your product's OPENSOURCE folder on your hard drive. This file is divided into these sections:

- **Components**
Includes the name of the open source component, its version number, and the type of license.
- **Copyright Text**
Includes the name of the open source component, its version number, and the copyright declaration.

- Licenses

Includes the name of the license, the list of open source components citing the license, and the terms of the license.

The default location of this file is:

C:\Program Files (x86)\Common Files\Rockwell\Help\<product name>\Release Notes\OPENSOURCE\oss_licenses.txt.

You may obtain Corresponding Source code for open source packages included in this product from their respective project web site(s).

Alternatively, you may obtain complete Corresponding Source code by contacting Rockwell Automation via the **Contact** form on the Rockwell Automation website: <http://www.rockwellautomation.com/global/about-us/contact/contact.page>. Please include "Open Source" as part of the request text.

Program Structured Text

Structured Text Syntax

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	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	An expression is part of a complete assignment or construct statement. An expression evaluates to a number (numerical expression), a String (string expression), or to a true or false state (BOOL expression)	
Tag Expression	A named area of the memory where data is stored (BOOL, SINT, INT, DINT, REAL, String).	value1
Immediate Expression	A constant value	4
Operators Expression	A symbol or mnemonic that specifies an operation within an expression.	tag1 + tag2 tag1 >= value1
Function Expression	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 be used only in expressions. Instructions cannot be used in expressions.	function(tag1)
Instruction	An instruction is a standalone statement. An instruction uses parentheses 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 be used only in expressions.	instruction(); instruction(operand); instruction(operand1, operand2, operand3);
Construct	A conditional statement used to trigger structured text code (that is, other statements). Terminate the construct with a semi colon (;).	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT

Comment	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*/
---------	---	---

See also

[Structured Text Components: Assignments](#) on [page 12](#)

[Structured Text Components: Expressions](#) on [page 15](#)

[Structured Text Components: Instructions](#) on [page 20](#)

[Structured Text Components: Constructs](#) on [page 21](#)

[Structured Text Components: Comments](#) on [page 34](#)

Structured Text Components: 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, STRING, or REAL. Tip: The STRING tag is applicable to CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers only.	
:=	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
	SINT INT DINT REAL	Numeric
	STRING (CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers only).	String type, including string tag and string literal (CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers only).
;	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 functions, or both. Refer to Expressions for more information.



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. For more information, see [Logix 5000 Controllers Common Procedures](#), publication 1756-PM001.

You can also use Input and Output program parameters which automatically buffer the data during the Logix Designer application execution. See [LOGIX 5000 Controllers Program Parameters Programming Manual](#), publication 1756-PM021.

See also

[Assign an ASCII character to a string data member](#) on [page 14](#)

[Specify a non-retentive assignment](#) on [page 13](#)

[Structured Text Components: Expressions](#) on [page 15](#)

[Character string literals](#) on [page 14](#)

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 by using a JSR instruction.

A non-retentive assignment has this syntax:

tag **[:=]** *expression* ;

where:

Component	Description	
<i>tag</i>	Represents the tag that is getting the new value; the tag must be a BOOL, SINT, INT, DINT, STRING, or REAL. Tip: The STRING tag is applicable to CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers only.	
[:=]	Is the non-retentive assignment symbol.	
<i>expression</i>	Represents the new value to assign to the tag.	
	If tag is this data type	Use this type of expression
	BOOL	BOOL
	SINT	Numeric
	INT	
	DINT	
	REAL	

	STRING (CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers only).	String type, including string tag and string literal CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers(only)
--	---	--

See also

[Assign an ASCII character to a string data member](#) on [page 14](#)

[Structured Text Components: Assignments](#) on [page 12](#)

Assign an ASCII character to a string data member

Assign an ASCII character to a string data member

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. For example:

This is OK	This is not OK
string1.DATA[0] := 65;	string1.DATA[0] := A;
string1.DATA[0] := string2.DATA[0];	string1 := string2; Tip: This assigns all content of string2 to string1 instead of just one character.

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

See also

[Structured Text Components: Expressions](#) on [page 15](#)

[Character string literals](#) on [page 14](#)

Character string literals

Character string literals include single byte or double byte encoded characters. A single-byte string literal is a sequence of zero or more characters that are prefixed and terminated by the single quote character ('). In single byte character strings, the three-character combination of the dollar sign (\$) followed by two hexadecimal digits is interpreted as the hexadecimal representation of the eight-bit character code as shown in the following table.



Tip: Character string literals are only applicable to the CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers. Studio 5000 only supports single byte characters.

Character string literals

No.	Description	Example
1a	Empty string (length zero)	"
1b	String of length one or character CHAR containing a single character	'A'
1c	String of length one or character CHAR containing the "space" character	' '
1d	String of length one or character CHAR containing the "single quote" character	'\$'
1e	String of length one or character CHAR containing the "double quote" character	''''
1f	Support of two character combinations	'\$R\$L'
1g	Support of a character representation with '\$' and two hexadecimal characters	'\$0A'

Two-character combinations in character strings

No.	Description	Example
1	Dollar sign	\$\$
2	Single quote	'\$'
3	Line feed	\$L or \$l
4	Newline	\$N or \$n
5	Form feed (page)	\$P or \$p
6	Carriage return	\$R or \$r
7	Tabulator	\$T or \$t



Tip: The newline character provides an implementation-independent means of defining the end of a line of data for both physical and file I/O; for printing, the effect is that of ending a line of data and resuming printing at the beginning of the next line.

The '\$' combination is only valid inside single quoted string literals.

See also

[Structured Text Components: Assignments](#) on [page 12](#)

Structured Text Components: Expressions

An expression is a tag name, equation, or comparison. To write an expression, use any of the following:

- Tag name that stores the value (variable)
- Number that you enter directly into the expression (immediate value)
- String literal that you enter directly into the expression (CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers only)
- Functions, such as: ABS, TRUNC
- Operators, such as: +, -, <, >, And, Or

Follow these guidelines for writing expressions:

- Use any combination of upper-case and lower-case letter. For example, these 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.

Use these expressions for structured text:

BOOL expression: An expression that produces 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, 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`.
- Nest a numeric expression within a BOOL expression. For example, `(tag1+5)>65`.

String expression: An expression that represents a string

- A simple expression can be a string literal or a string tag

Use this table to select the operators for expressions.

If	Use
Calculating an arithmetic value	Arithmetic operators and functions
Comparing two values or strings	Relational operators
Verifying if conditions are true or false	Logical operators
Comparing the bits within values	Bitwise operators

See also

[Use arithmetic operators and functions](#) on [page 16](#)

[Use relational operators](#) on [page 17](#)

[Use logical operators](#) on [page 19](#)

[Use bitwise operators](#) on [page 19](#)

Use arithmetic operators and functions

Combine multiple operators and functions in arithmetic expressions.

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

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

The table provides examples for using arithmetic operators and functions.

Use this format	Example	
	For this situation	Write
<i>value1 operator value2</i>	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;
<i>operator value1</i>	If alarm and high_alarm are DINT tags and your specification says: 'Negate high_alarm and store the result in alarm.'	alarm:= -high_alarm;
<i>function(numeric_expression)</i>	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);
<i>value1 operator (function((value2+value3)/2))</i>	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);

See also

[Structured Text Components: Expressions](#) on [page 15](#)

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 type
Less than	<	DINT, REAL, String type
Less than or equal	<=	DINT, REAL, String type
Greater than	>	DINT, REAL, String type
Greater than or equal	>=	DINT, REAL, String type
Not equal	<>	DINT, REAL, String type

The table provides examples of using relational operators

Use this format	Example	
	For this situation	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...
stringtag1 operator 'character string literal'	If bar_code is a string tag and your specification says: 'If bar_code equals 'Test PASSED' then...'	IF bar_code='Test PASSED' 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	— AB < B
B	\$42	—
a	\$61	— a > B
ab	\$61\$62	

↑
l
e
s
s
e
r
 ↓
g
r
e
a
t
e
r

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

See also

[Structured Text Components: Expressions](#) on [page 15](#)

Use logical operators

Use logical operators to verify 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 this comparison	Use this operator	Optimal data type
logical AND	&, AND	BOOL
logical OR	OR	BOOL
logical exclusive OR	XOR	BOOL
logical complement	NOT	BOOL

The table provides examples of using logical operators.

Use this format	Example	
	For this situation	Use
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;

See also

[Structured Text Components: Expressions](#) on [page 15](#)

Use bitwise operators

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

The following provides an overview of the bitwise operators.

For	Use this operator	Optimal data type
bitwise AND	&, AND	DINT
bitwise OR	OR	DINT
bitwise exclusive OR	XOR	DINT
bitwise complement	NOT	DINT

This is an example.

Use this format	Example	
	For this situation	Use
<i>value1 operator value2</i>	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;

See also

[Structured Text Components: Expressions](#) on [page 15](#)

Determine the order of execution

The operations written into an expression perform in a prescribed order.

- Operations of equal order perform 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

See also

[Structured Text Components: Expressions](#) on [page 15](#)

Structured Text Components: Instructions

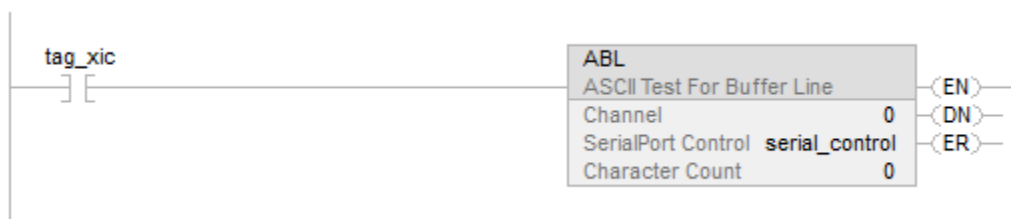
Structured text statements can also be instructions. A structured text instruction executes each time it is scanned. A structured text instruction within a construct executes every time the conditions of the construct are true. If the conditions of the construct are false, the statements within the construct 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 ladder diagram instructions that use rung-condition-in to trigger execution. Some ladder diagram instructions only execute when rung- condition-in toggles from false to true. These are transitional ladder

diagram instructions. In structured text, instructions execute when they are scanned unless pre-conditioning the execution of the structured text instruction.

For example, the ABL instruction is a transitional instruction in ladder diagram. 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 clears.



In structured text, if writing 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;
```

Structured Text Components: Constructs

Program constructs alone or nest within other constructs.

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

Some Key Words are Reserved

These constructs are not available:

- GOTO
- REPEAT

Logix Designer application will not let you use them as tag names or constructs.

See also

[IF THEN](#) on [page 22](#)

[CASE OF](#) on [page 25](#)

[FOR DO](#) on [page 27](#)

[WHILE DO](#) on [page 29](#)

[REPEAT UNTIL](#) on [page 31](#)

IF_THEN

Use IF_THEN to complete an action when specific conditions occur.

Operands

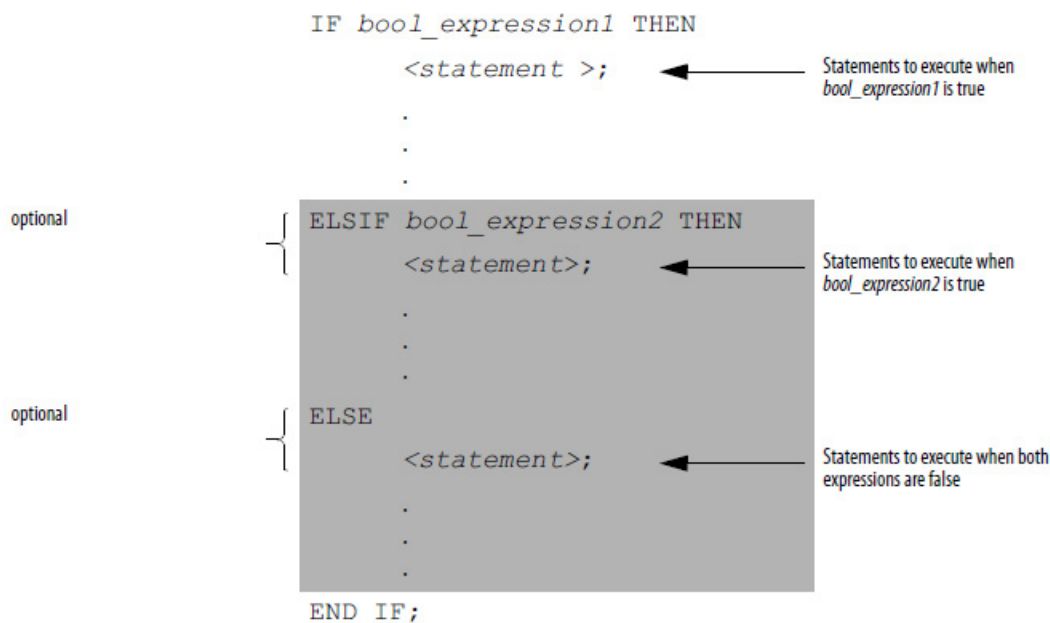
IF bool_expression THEN

<statement>;

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 described in the table.



To use ELSIF or ELSE, follow these guidelines.

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.

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

The table summarizes different combinations of IF, THEN, ELSIF, and ELSE.

If	And	Use this construct
Doing something if or when conditions are true	Do nothing if conditions are false	IF_THEN
	Do something else if conditions are false	IF_THEN_ELSE
Selecting 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

Affects Math Status Flags

No

Major/Minor Faults

None.

Examples

Example 1

IF...THEN

If performing 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 performing 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 does the following :

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 performing 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 does the following :

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 performing 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_OF to select what to do based on a numerical value.

Operands

CASE numeric_expression OF

selector1: statement;

selectorN: statement; ELSE

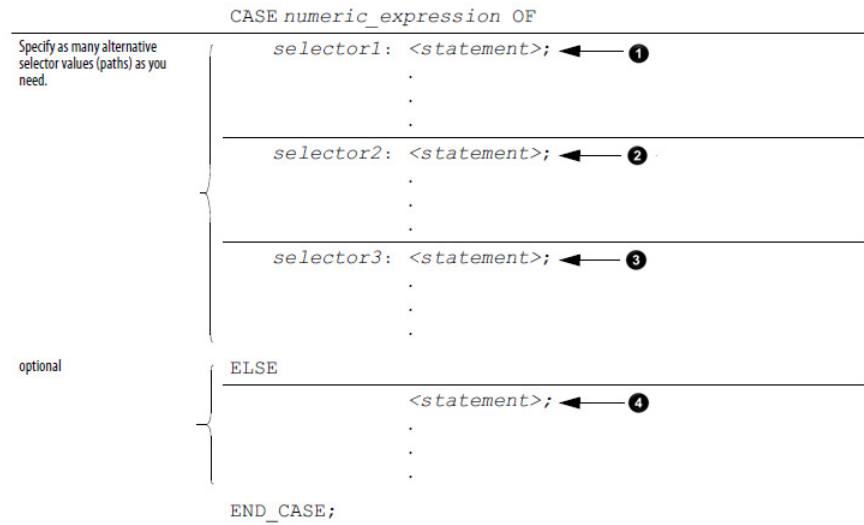
Structured Text

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 using 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 described in the table.



These are the syntax for entering the selector values.

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. With the CASE construct, the controller executes only the statements that associated with the first matching selector value. Execution always breaks after the statements of that selector and goes to the `END_CASE` statement.

Affects Math Status Flags

No

Major/Minor Faults

None

Example

If you want this	Enter this structured text
If recipe number = 1 then Ingredient A outlet 1 = open (1) Ingredient B outlet 4 = open (1)	CASE recipe_number OF 1: Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=1;
If recipe number = 2 or 3 then Ingredient A outlet 4 = open (1) Ingredient B outlet 2 = open (1)	2,3: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;
If recipe number = 4, 5, 6, or 7 then Ingredient A outlet 4 = open (1) Ingredient B outlet 2 = open (1)	4 to 7: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;
If recipe number = 8, 11, 12, or 13 then Ingredient A outlet 1 = open (1) Ingredient B outlet 4 = open (1)	8,11...13 Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=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 does the following:

Enters the RUN mode.

Leaves the step of an SFC if configuring the SFC for Automatic reset. This applies only embedding the assignment in the action of the step or using the action to call a structured text routine via a JSR instruction.

FOR_DO

Use the FOR_DO loop to perform an action a number of times before doing anything else.

When enabled, the FOR instruction repeatedly executes the Routine until the Index value exceeds the Terminal value. The step value can be positive or negative. If it is negative, the loop ends when the index is less than the terminal value.. If it is positive, the loop ends when the index is greater than the terminal value.

Each time the FOR instruction executes the routine, it adds the Step size to the Index.

Do not loop too many times in a single scan. An excessive number of repetitions causes the controller watchdog to timeout and causes a major fault.

Operands

FOR count:= initial_value TO

final_value BY increment DO

<statement>;

END_FOR;

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 Do not iterate within the loop too many times in a single scan.
The controller does not execute other statements in the routine until it completes the loop.
A major fault occurs when completing the loop takes longer than the watchdog timer for the task.
Consider using a different construct, such as IF_THEN.

Description

The syntax is described in the table.

```

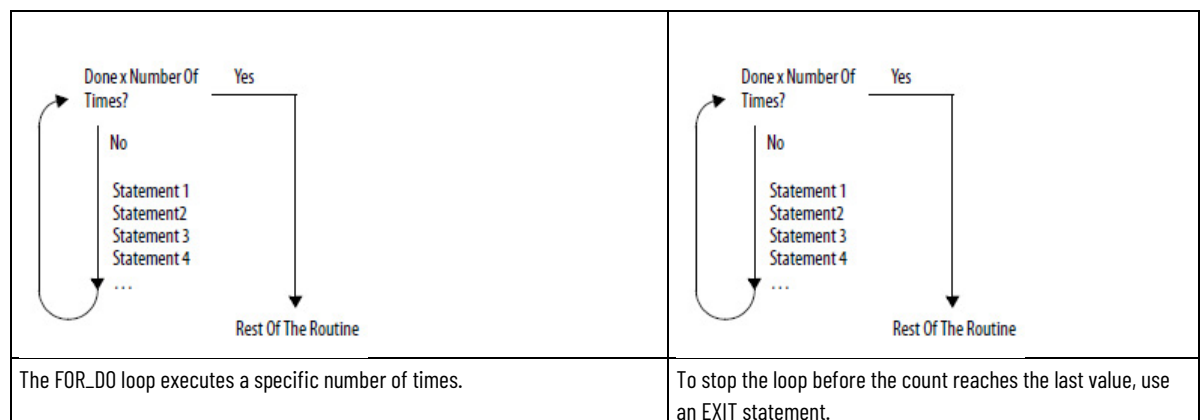
FOR count := initial_value
    TO final_value
optional [ BY increment
DO
    <statement>;
optional { IF bool_expression THEN
            EXIT;
            END_IF;
END_FOR;

```

If you don't specify an increment, the loop increments by 1.

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.

This diagrams illustrates how a FOR_DO loop executes, and how an EXIT statement leaves the loop early.



Affects Math Status Flags

No

Major/Minor Faults

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

Example 1

If performing the following,	Enter this structured text
Clear bits 0...31 in an array of BOOLs: Initialize the subscript tag to 0. Clear i . For example, when subscript = 5, clear array[5]. Add 1 to subscript. If subscript is $\leq$ to 31, repeat 2 and 3. Otherwise, stop.	For subscript:=0 to 31 by 1 do array[subscript] := 0; End_for;

Example 2

If performing the following,	Enter this structured text
A user-defined data type (structure) stores the following 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 2. Inventory_Items (DINT tag). Initialize the position tag to 0. 3. If Barcode matches the ID of an item in the array, then: Set the Quantity tag = Inventory[position].Qty. This produces the quantity in stock of the item. 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 $\leq$ 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 continue performing an action while certain conditions are true.

Operands

WHILE bool_expression DO

<statement>;

Structured Text

Operand	Type	Format	Description
<i>bool_expression</i>	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value

IMPORTANT

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.

A major fault occurs when completing the loop takes longer than the watchdog timer for the task.

Consider using a different construct, such as IF_THEN.

Description

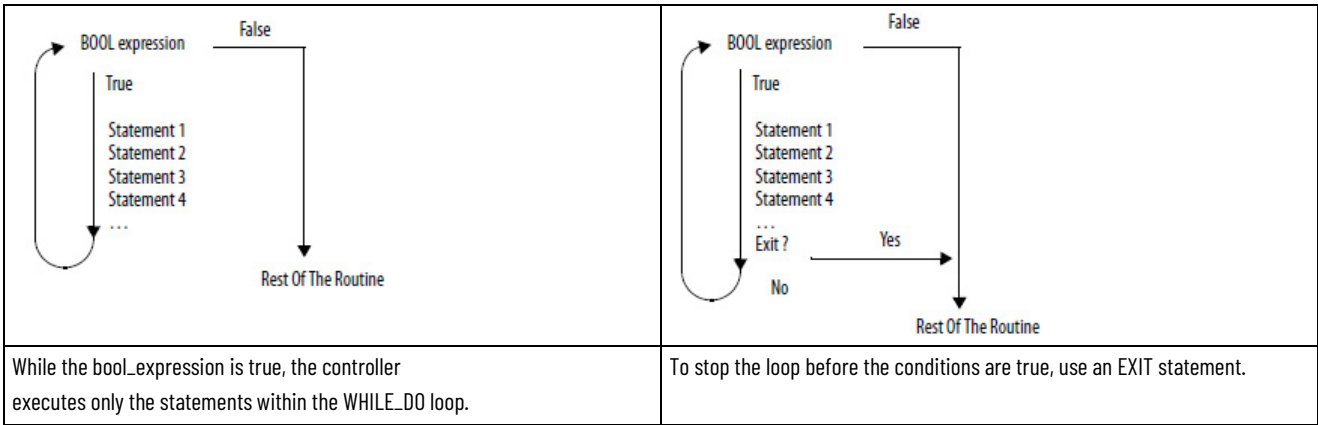
The syntax is:

```
WHILE bool_expression1 DO
    <statement>;
    IF bool_expression2 THEN
        EXIT;
    END_IF;
END_WHILE;
```

statements to execute while *bool_expression1* is true

optional { 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.

The following diagrams illustrate how a WHILE_DO loop executes, and how an EXIT statement leaves the loop early.



Affects Math Status Flags

No

Fault Conditions

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

Example 1

If performing the following,	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.	<pre>pos := 0; While ((pos <= 100) & structarray[pos].value <> targetvalue)) do pos := pos + 2; String_tag.DATA[pos] := SINT_array[pos]; end_while;</pre>

Example 2

If performing the following,	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. Initialize Element_number to 0. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag). If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop. Set String_tag[element_number] = the character at SINT_array[element_number]. Add 1 to element_number. This lets the controller check the next character in SINT_array. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.) If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)	<pre>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;</pre>

REPEAT_UNTIL

Use the REPEAT_UNTIL loop to continue performing an action until conditions are true.

Operands

REPEAT

<statement>;

Structured Text

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

IMPORTANT Do not iterate within the loop too many times in a single scan.
The controller does not execute other statements in the routine until it completes the loop.
A major fault occurs when completing the loop takes longer than the watchdog timer for the task.
Consider using a different construct, such as IF_THEN.

Description

The syntax is:

```

REPEAT
    <statement>;
    optional {
        IF bool_expression2 THEN
            EXIT;
        END_IF;
    }
UNTIL bool_expression1
END_REPEAT;

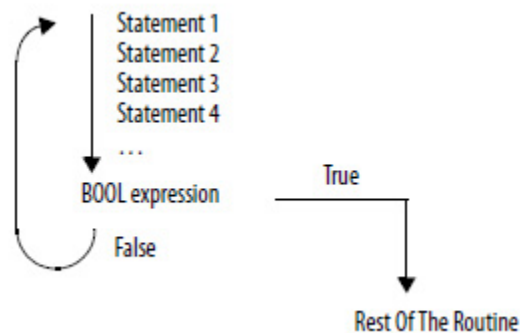
```

statements to execute while *bool_expression1* is false

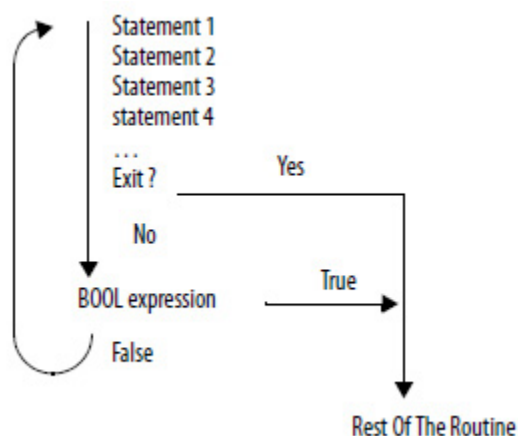
If there are conditions when you want to exit the loop early, use other statements, such as an IF_THEN construct, to condition an EXIT statement.

The following 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.



Affects Math Status Flags

No

Fault Conditions

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

Example 1

If performing the following,	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.	<pre> pos := -1; REPEAT pos := pos + 2; UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue)) end_repeat; </pre>

Example 2

If performing the following,	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. Initialize Element_number to 0. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).	<pre> 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 </pre>

Set String_tag[element_number] = the character at SINT_array[element_number]. Add 1 to element_number. This lets the controller check the next character in SINT_array. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.) If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.) If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.	exit;
	end_if;
	Until SINT_array[element_number] = 13
	end_repeat;

Structured Text

Components: 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.

To add comments to your structured text:

To add a comment	Use one of these formats
on a single line	//comment (*comment*)
at the end of a line of structured text	/*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);

Assignments 12

structured text expression 20

Index

W

WHILE?DO 29

A

arithmetic operators 16

ASCII character 14

assign ASCII character 14

B

bitwise operators 19

C

CASE 25

comments 34

E

evaluation in structured text 17

evaluation of strings 17

F

FOR?DO 27

functions 16

I

IF...THEN 22

L

logical operators 19

N

non-retentive 13

non-retentive assignment 13

R

relational operators 17

REPEAT?UNTIL 31

S

structured text 13, 16, 17, 19, 20, 34

structured text assignment 14

Structured Text Components

Rockwell Automation support

Use these resources to access support information.

Technical Support Center	Find help with how-to videos, FAQs, chat, user forums, and product notification updates.	rok.auto/support
Knowledgebase	Access Knowledgebase articles.	rok.auto/knowledgebase
Local Technical Support Phone Numbers	Locate the telephone number for your country.	rok.auto/phonesupport
Literature Library	Find installation instructions, manuals, brochures, and technical data publications.	rok.auto/literature
Product Compatibility and Download Center (PCDC)	Get help determining how products interact, check features and capabilities, and find associated firmware.	rok.auto/pcdc

Documentation feedback

Your comments help us serve your documentation needs better. If you have any suggestions on how to improve our content, complete the form at rok.auto/docfeedback.

Waste Electrical and Electronic Equipment (WEEE)



At the end of life, this equipment should be collected separately from any unsorted municipal waste.

Rockwell Automation maintains current product environmental information on its website at rok.auto/pec.

Allen-Bradley, expanding human possibility, Logix, Rockwell Automation, and Rockwell Software are trademarks of Rockwell Automation, Inc.

EtherNet/IP is a trademark of ODVA, Inc.

Trademarks not belonging to Rockwell Automation are property of their respective companies.

Rockwell Otomasyon Ticaret A.Ş. Kar Plaza İş Merkezi E Blok Kat:6 34752, İçerenköy, İstanbul, Tel: +90 (216) 5698400 EEE Yönetmeliğine Uygundur

Connect with us.

rockwellautomation.com — expanding **human possibility**™

AMERICAS: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444

EUROPE/MIDDLE EAST/AFRICA: Rockwell Automation NV, Pegasus Park, De Kleetlaan 12a, 1831 Diegem, 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