Programming Manual

Allen-Bradley

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

Allen-Bradley • Rockwell Software

Rockwell Automation

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).

Summary of changes

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.

Subject	Reason
Updated supported controller models.	New controller models have been released.
Updated screen shots.	The Logix Designer interface has been updated.

New or enhanced features

This table contains a list of topics changed in this version, the reason for the change, and a link to the topic that contains the changed information.

Topic Name	Reason
Comments on page 38	Added information on using the **Comment text block** command.

Table of contents

Preface

Chapter 1

Program Structured Text

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 that is 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.
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

Copyright notice

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.

Other Licenses

The software included in this product contains copyrighted software that is licensed under one or more open source licenses. Copies of those licenses are included with the software. Corresponding Source code for open source packages included in this product can be located at their respective web site(s).

You may alternately obtain complete Corresponding Source code by contacting Rockwell Automation via our 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.

The following open source software is used in this product:

Software	Copyright	License Name	License Text
AngularJS	Copyright 2010-2017 Google, Inc.	MIT License	AngularJS 1.5.9 License
Bootstrap	Copyright 2011-2017 Twitter, Inc. Copyright 2011-2017 The Bootstrap Authors	MIT License	Bootstrap 3.3.7 License
jQuery	Copyright 2005, 2014 JS Foundation and other contributors	MIT License	jQuery 2.1.1 License
OpenSans	Copyright 2017 Google, Inc.	Apache License, Version 2.0	OpenSans License

Trademark Notices

Allen-Bradley, ControlBus, ControlFLASH, Compact GuardLogix, Compact I/O, ControlLogix, CompactLogix, DCM, DH+, Data Highway Plus, DriveLogix, DPI, DriveTools, Explorer, FactoryTalk, FactoryTalk Administration Console, FactoryTalk Alarms and Events, FactoryTalk Batch, FactoryTalk Directory, FactoryTalk Security, FactoryTalk Services Platform, FactoryTalk View, FactoryTalk View SE, FLEX Ex, FlexLogix, FLEX I/O, Guard I/O, High Performance Drive, Integrated Architecture, Kinetix, Logix5000, Logix 5000, Logix5550, MicroLogix, DeviceNet, EtherNet/IP, PLC-2, PLC-3, PLC-5, PanelBuilder, PowerFlex, PhaseManager, POINT I/O, PowerFlex, Rockwell Automation, RSBizWare, Rockwell Software, RSEmulate, Historian, RSFieldbus, RSLinx, RSLogix, RSNetWorx for DeviceNet, RSNetWorx for EtherNet/IP, RSMACC, RSView, RSView32, Rockwell Software Studio 5000 Automation Engineering & Design Environment, Studio 5000 View Designer, SCANport, SLC, SoftLogix, SMC Flex, Studio 5000, Ultra 100, Ultra 200, VersaView, WINtelligent, XM, SequenceManager are trademarks of Rockwell Automation, Inc.

Any Rockwell Automation logo, software or hardware product not mentioned herein is also a trademark, registered or otherwise, of Rockwell Automation, Inc.

Other Trademarks

CmFAS Assistant, CmDongle, CodeMeter, CodeMeter Control Center, and WIBU are trademarks of WIBU-SYSTEMS AG in the United States and/or other countries. Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries. ControlNet is a trademark of ControlNet International. DeviceNet is a trademark of the Open DeviceNet Vendors Association (ODVA). Ethernet/IP is a trademark of ControlNet International under license by ODVA.

Warranty

This product is warranted in accordance with the product license. The product's performance may be affected by system configuration, the application being performed, operator control, maintenance, and other related factors. Rockwell Automation is not responsible for these intervening factors. The instructions in this document do not cover all the details or variations in the equipment, procedure, or process described, nor do they provide directions for meeting every possible contingency during installation, operation, or maintenance. This product's implementation may vary among users.

This document is current as of the time of release of the product; however, the accompanying software may have changed since the release. Rockwell Automation, Inc. reserves the right to change any information contained in this document or the software at any time without prior notice. It is your responsibility to obtain the most current information available from Rockwell when installing or using this product.

Environmental compliance

Rockwell Automation maintains current product environmental information on its website at http://www.rockwellautomation.com/rockwellautomation/about-us/sustainability-ethics/product-environmental-compliance.page

Contact Rockwell Automation

Customer Support Telephone — 1.440.646.3434

Online Support — http://www.rockwellautomation.com/support/

Chapter 1

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 do not effect on the execution of the structured text.

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) or to a true or false state (BOOL expression). An expression contains these components.		
	Tags	A named area of the memory where data is stored (BOOL, SIN, INT, DINT, REAL, String type).	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)
Instruction	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	A conditional statement used to trigger structured text code, such as 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. • Comments 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:

<table>
<tr><th>Component</th><th colspan="2">Description</th></tr>
<tr><td>Tag</td><td colspan="2">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.</td></tr>
<tr><td>:=</td><td colspan="2">Is the assignment symbol</td></tr>
<tr><td rowspan="5">Expression</td><td colspan="2">Represents the new value to assign to the tag</td></tr>
<tr><td>If tag is this data type</td><td>Use this type of expression</td></tr>
<tr><td>BOOL</td><td>BOOL expression</td></tr>
<tr><td>SINT
INT
DINT
REAL</td><td>numeric expression</td></tr>
<tr><td>STRING (CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers only).</td><td>String type, including string tag and string literal.</td></tr>
<tr><td>;</td><td colspan="2">Ends the assignment</td></tr>
</table>

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 on page 15 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 logix execution. See LOGIX 5000 Controllers Program Parameters Programming Manual , publication 1756-PM021 .

Specify a non-retentive assignment

The non-retentive assignment is different from the regular assignment previously shown because 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:

<table>
<tr><th>Component</th><th colspan="2">Description</th></tr>
<tr><td>tag</td><td colspan="2">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.</td></tr>
<tr><td>[:=]</td><td colspan="2">Is the non-retentive assignment symbol.</td></tr>
<tr><td rowspan="8">expression</td><td colspan="2">Represents the new value to assign to the tag.</td></tr>
<tr><td>If tag is this data type</td><td>Use this type of expression</td></tr>
<tr><td>BOOL</td><td>BOOL expression</td></tr>
<tr><td>SINT</td><td rowspan="4">Numeric expression</td></tr>
<tr><td>INT</td></tr>
<tr><td>DEAL</td></tr>
<tr><td>REAL</td></tr>
<tr><td>STRING (CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers).</td><td>String type, including string tag and string literal.</td></tr>
<tr><td>;</td><td colspan="2">ends the assignment</td></tr>
</table>

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.

This is OK	This is not OK
string1.DATA[0]:= 65;	string1.DATA[0] := A;
string1.DATA[0]:= string2.DATA[0];	string1 := string2; **Tip:** string1: = string2 erroneously assigns all the content of string2 to string1.

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)
- String literal that you enter directly into the expression (CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers)
- 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 letters. For example, these three variations of "AND" (AND, And, and) are acceptable.
- 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, 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.

String expression: An expression that represents a string

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

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 16.
Compare two values or strings	Use Relational Operators on page 18.
Check if conditions are true or false	Use Logical Operators on page 19.
Compare the bits within values	Use Bitwise Operators on page 20.

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. See the following table to 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 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 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

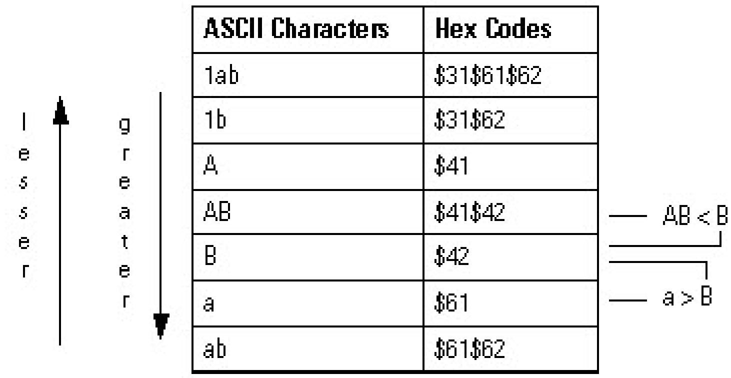
The table shows some examples.

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.



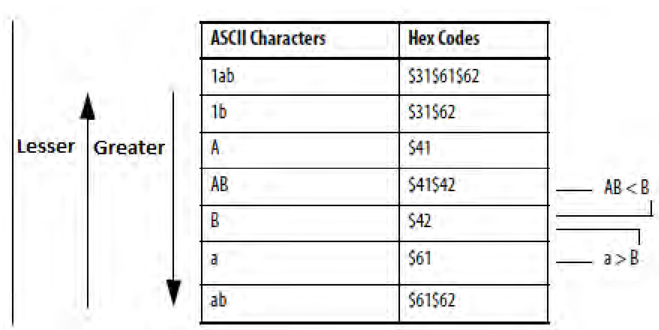
ASCII Characters	Hex Codes
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

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

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, 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. Uppercase ‘A’ ($41) is not equal to lowercase ‘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 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.

See the following table for 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

For	Use this operator	Optimal Data Type
Bitwise complement	NOT	DINT

For example:

Use this format	Example	
value1 operator value2	**For this situation**	**You write**
	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.

The following table lists order of operation.

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:

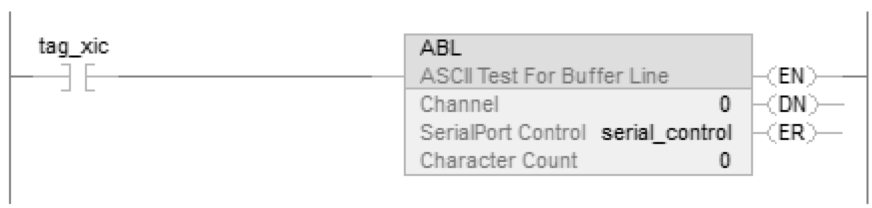
- Executes each time it is scanned.
- Within a construct executes every time the conditions of the construct are true.

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

You can program constructs singly or nested within other constructs.

Follow this table to use the appropriate construct.

If you want to	Use this construct
Do something if or when specific conditions occur	IF...THEN on page 23
Select what to do based on a numerical value	CASE...OF on page 26
Do something a specific number of times before doing anything else	FOR…DO on page 29
Keep doing something as long as certain conditions are true	WHILE…DO on page 32
Keep doing something until a condition is true	REPEAT…UNTIL on page 35

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**	**Tag Expression**
bool_expression	BOOL	Tag Expressio n	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

Description:

The syntax is:

```
            IF bool_expression1 THEN
               <statement >; 1
                                   .
                                   .
                                   .
            ELSIF bool_expression2 THEN
Optional
               <statement>; 2
                                   .
                                   .
                                   .
            ELSE
Optional
               <statement>; 3
                                   .
                                   .
                                   .
            END_IF;
```

1. Statements to execute when bool_expression1 is true.
2. Statements to execute when bool_expression2 is true.
3. Statements to execute when both expressions are false.

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…ELSE
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

Affects Math 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 through 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 through 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. Use CASE...OF in a Logix program to determine the next process to run based on the evaluation of a numerical input value.

Operands

CASE numeric_expression OF

selector1: statement;

selectorN: statement; ELSE

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:	As a best practice, use a range of values for a selector when evaluating numeric expressions with REAL data types.

Description

The following table depicts how the CASE syntax is evaluated.

```
                 CASE numeric_expression OF

Optional         selector1: <statement>;  1
                                                .
                                                .
                                                .
Optional         selector2: <statement>;  2
                                                .
                                                .
                                                .
Optional         selector3: <statement>;  3
                                                .
                                                .
                                                .
Optional         ELSE>
                          <statement>  4
                                                .
                                                .
                                                .
                 END_CASE;
```

1 Statements to execute when numeric_expression = selector1

2 Statements to execute when numeric_expression = selector2

3	Statements to execute when numeric_expression = selector3
4	Statements to execute when numeric_expression ≠ selector1

Use the following to help determine 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. 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.

Affects Math Status Flags

No

Fault Conditions

None.

Example

The following table provides examples that illustrate how to translate a functional requirement into structured text using the standard syntax of CASE ... OF, and modifying it with the requirement variables.

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…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 clear the outlet tags whenever the controller does the following:

- Enters the RUN mode.
- Leaves the step of an SFC if you configure the SFC for Automatic reset.

FOR...DO

Use a FOR...DO loop to perform an evaluation process a specific number of times before continuing on to the next instruction in the sequence.

Operands

FOR count:= initial_value TO

final_value BY increment DO

<statement>;

END_FOR;

Structured Text

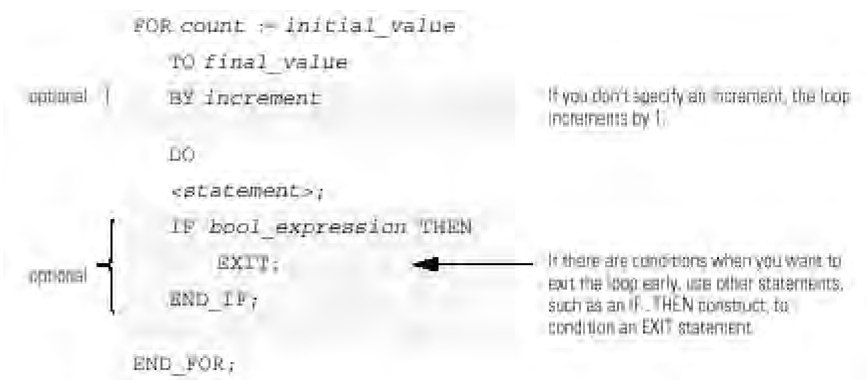
Operand	Type	Format	Description
count	SINT INT DINT	Tag	Tag to store count position as the FOR…DO executes
initial_ value	SINT INT DINT	Tag expression Immediate	Must evaluate to a number Specifies initial value for count
final_ value	SINT INT DINT	Tag expression Immediate	Specifies final value for count, 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.

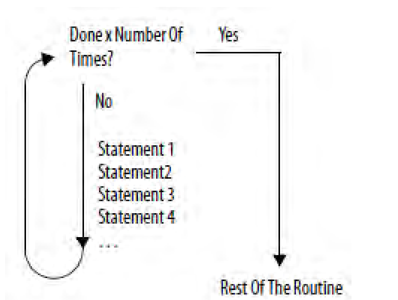
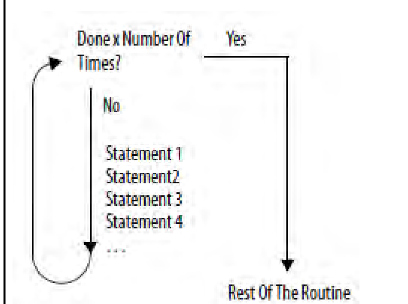
Important: The controller does not execute any other statements in the routine until it completes the loop.
Make sure that you do not iterate within the loop too many times in a single scan.
If the time that it takes to complete the loop is greater than the Watchdog timer for the task, a major fault occurs. If you encounter this fault, consider using a different construct, such as IF...THEN.

Description

The following table depicts how the FOR...DO syntax is evaluated.



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

<table>
<tr><td>
</td><td>
</td></tr>
<tr><td>The FOR…DO loop executes a specific number of times.</td><td>To stop the loop before the count reaches the last value, use an EXIT statement.</td></tr>
</table>

Affects Math Status Flags

No

Fault Conditions

A major fault will occur if	Fault type	Fault code
If the time that it takes to complete the loop is greater than the Watchdog timer for the task.	6	1

Examples

The following tables provide examples that illustrate how to translate a functional requirement into structured text using the standard syntax of FOR...DO and then modifying it with the requirement variables.

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 i . 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;

If you want this	Enter this structured text
A user-defined data type (structure) stores the following information about an item in your inventory:	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;

• 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 unique 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: • Set the Quantity tag = Inventory[position].Qty. This produces the quantity in stock of the item, or • 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.	END_FOR;

WHILE...DO

Use a WHILE...DO loop to continue performing a process until the specified condition is false before continuing on to the next instruction in the sequence.

Operands

WHILE bool_expression DO

<statement>;

END_WHILE;

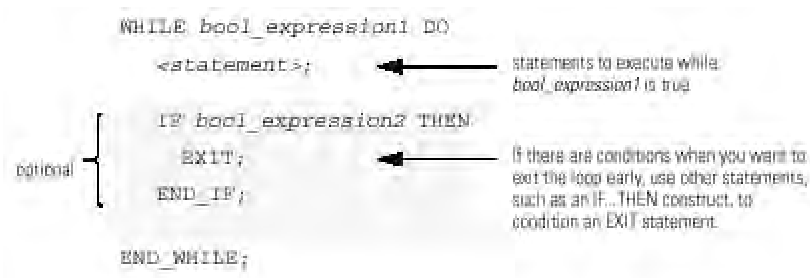
Structured Text

Operand	BOOL	Format	Description
bool_expression	BOOL	tag expressio n	BOOL tag or expression that evaluates to a BOOL value

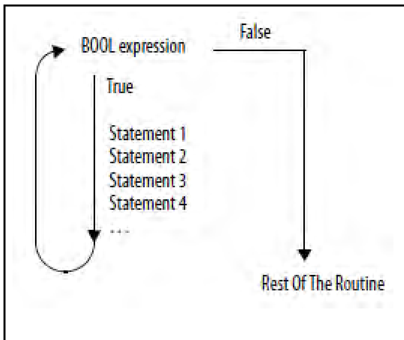
Important:	The controller does not execute any other statements in the routine until it completes the loop. Make sure that you do not iterate within the loop too many times in a single scan. If the time that it takes to complete the loop is greater than the Watchdog timer for the task, a major fault occurs. If you encounter this fault, consider using a different construct, such as IF...THEN.

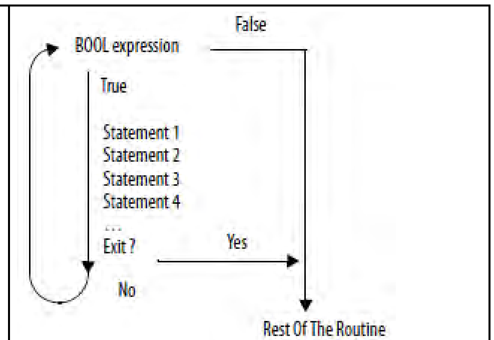
Description

The following table depicts how the WHILE..DO syntax is evaluated.



The following 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.

Affects Math Status Flags

No

Fault Conditions

A major fault will occur if	Fault type	Fault code
If the time that it takes to complete the loop is greater than the Watchdog timer for the task.	6	1

Examples

The following tables provide examples that illustrate how to translate a functional requirement into structured text using the standard syntax of WHILE...DO and then modifying it with the requirement variables.

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;

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] =	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;

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 step 3.	end_while;

REPEAT...UNTIL

Use a REPEAT...UNTIL loop to repeat an evaluation process until the specified condition is true before continuing on to the next instruction in the sequence.

Operands

REPEAT

<statement>;

END_REPEAT;

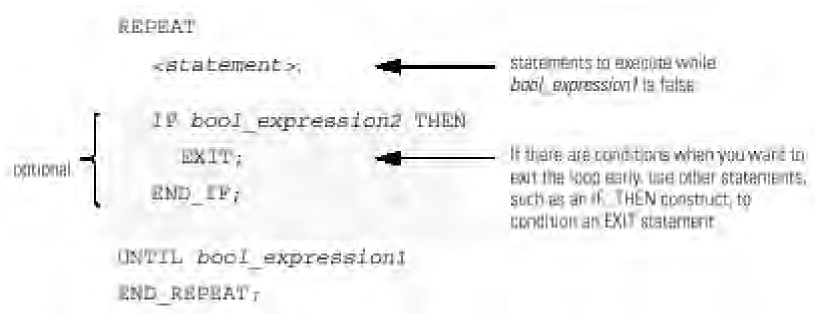
Structured Text

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

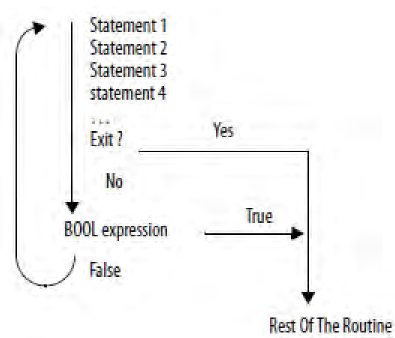
Important:	The controller does not execute any other statements in the routine until it completes the loop. Make sure that you do not iterate within the loop too many times in a single scan. If the time that it takes to complete the loop is greater than the Watchdog timer for the task, a major fault occurs. If you encounter this fault, consider using a different construct, such as IF...THEN.

Description

The following table depicts how the REPEAT...UNTIL syntax is evaluated.



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

<table>
<tr><td></td><td>
</td></tr>
<tr><td>While the bool_expression is false, the controller executes only the statements within the REPEAT…UNTIL loop.</td><td>To stop the loop before the conditions are false, use an EXIT statement.</td></tr>
</table>

Affects Math Status Flags

No

Fault Conditions

A major fault will occur if	Fault type	Fault code
If the time that it takes to complete the loop is greater than the Watchdog timer for the task.	6	1

Examples

The following tables provide examples that illustrate how to translate a functional requirement into structured text using the standard syntax of REPEAT...UNTIL, and then modifying it with the requirement variables.

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;

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.
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).
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.
	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

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.	end_repeat;

Comments

You can add comments to make your structured text easier to interpret. Comments:

- Let you use plain language to describe how your structured text works.
- Do not affect the execution of the structured text.
- Download into the controller memory and are available for upload.

Follow this table 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);

Tip: On the main menu, select **Edit > Comment Text Block** to add comment text. Additionally, use the **Comment Text Block** command to:

- Mark the entire selection as a multi-line comment.
- Add a single line comment to the line that contains the cursor when no text is selected.
- Add "//" comment delimiter to the beginning of each line when using line selection to select individual lines.
- Add "(*" and "*)" delimiters to the beginning and the end of the text selection, respectively, when selecting a portion of text that spans multiple lines, either entire lines or portions of them.

Index

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 and application notes, sample code, and links to software service packs. You can also visit our Support Center at https://rockwellautomation.custhelp.com for software updates, support chats and forums, technical information, FAQs, and to sign up for product notification updates.

In addition, we offer multiple support programs for installation, configuration, and troubleshooting. For more information, contact your local distributor or Rockwell Automation representative, or visit http://www.rockwellautomation.com/services/online-phone.

Installation assistance

If you experience a problem 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 available at http://www.rockwellautomation.com/locations, 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 the feedback form, publication RA-DU002.

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

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

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

Rockwell Automation Publication 1756-PM007G-EN-P - February 2018

Supersedes Publication 1756-PM007F-EN-P - December 2016