

SR-X Series Script Reference Rev.2.0



Symbols

This manual uses the following symbols that alert you to important messages. Be sure to read these messages carefully. Be sure to read these messages carefully.

	It indicates a hazardous situation which, if not avoided, will result in death or serious injury.
	It indicates a hazardous situation which, if not avoided, could result in death or serious injury.
	It indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.
	It indicates a situation which, if not avoided, could result in product damage as well as property damage.

	Cautions as to operation that is always performed are shown.
	Cautions as to operation that can be easily performed incorrectly are shown.

	Matters that will help the user improve understanding and useful information are shown.
--	---

The items and pages to be referred to in this manual are shown.

Introduction

This manual explains the script language described when using the edit read data function and the edit image file name function on the SR-X Series. For information on the basic functions of the SR-X Series, refer to the user's manual. This manual is written for users with basic knowledge of PC and programming experience.

Related manuals

- **Included in DVD-ROM**
 - SR-X Series User's Manual
- **Download from KEYENCE homepage**
Download the latest manual from the following web page.
BarcodeReader.com
<http://www.barcodereader.com/>

Table of Contents

General cautions	1
1-1 Overview	2
1-2 Script execution flow	2
1-3 What Script can do	2
1-4 Script file configuration	2
2-1 Naming rules	4
2-2 Variables	4
2-3 Table	5
2-4 Array	5
3-1 Calculation	6
3-2 Comparison	6
3-3 Combination of condition	6
3-4 Character string concatenation	6
3-5 Pattern matching	7
4-1 Conditional branching of process	8
4-2 Repeat process	9
5-1 Data acquisition function	10
5-2 Common functions (Basic)	11
5-3 Common functions (Advanced)	11
5-4 Controlling output terminals and triggers	12
5-5 Confirmation command function	12
5-6 Second output data setting function	13
5-7 User-defined function	13
6-1 Debug methods	13
6-2 Error message list	14
7-1 Practical sample programs	14
8-1 ASCII code table	20
8-2 Reserved words/Language elements	20
8-3 Code type	20
8-4 Troubleshooting	21
8-5 Copyright indication	21

General cautions

- KEYENCE does not guarantee the results if it is used in a manner that differs from the descriptions in this manual.
- It is prohibited to use or copy all or any part of this manual without prior approval.
- The information contained in this manual is subject to change without notice.
- The company names and product names described in this manual are registered trademarks or trademarks of each company.

1-1 Overview

Definition of Script file

The Script explained in this manual is a simple programming language operating on the SR-X Series.

Compared to using the setting software (AutoID Network Navigator), using the script enables more flexible operations for the (1) Edit output data, (2) Edit image file name, and (3) Control output terminals functions.

Related software and features

AutoID Network Navigator

Software to set the SR-X Series

FileView

Use this to send/receive the script file to/from the SR-X Series.

Web Navigator

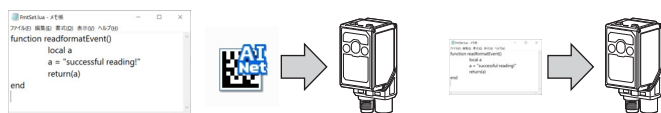
The SR-X Series has a web server feature. Settings can be configured by connecting to the server in a web browser on a computer.

* See the User's Manual for detailed operating instructions for AutoID Network Navigator, FileView and Web Navigator.

1-2 Script execution flow

Steps before executing the script

[1] Creating the script file [2] Changing the script execution setting [3] Transferring the script file



[1] Creating the script file

Create the script file (FmtSet.lua) and write the program using a text editor such as notepad.exe. (This manual describes writing methods for programming.)

[2] Changing the script execution setting

Using the AutoID Network Navigator, set the script execution setting of the SR-X Series to "Enable".

[3] Transferring the script file

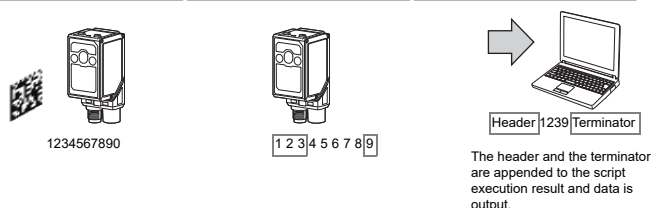
Transfer the script file (FmtSet.lua) to the SR-X Series.

* See the User's Manual for instructions on configuring settings for AutoID Network Navigator and Web Navigator.

Operations when executing the script

The script is executed when a code is read.

[1] Reading a code [2] Executing the script when reading. [3] Sending the script execution result



1-3 What Script can do

Examples of what the script can do

- Extracting arbitrary portion of read data
- Appending arbitrary character strings to read data
- Comparing data and outputting result data
- Four arithmetic operations
- Changing image file names for FTP transmission
- Comparing data and generating output from the output terminals

* "Additional information" set with AutoID Network Navigator cannot be used in combination with the script.

1-4 Script file configuration

This describes the configuration of the script file (FmtSet.lua).

Script file configuration

FmtSet.lua file, the script program handled by SR-X Series consists of the following 3 parts.

- (1) function readformatEvent()..... Edit read data
- (2) function nameformatEvent(idx) Edit image file name
- (3) User-defined function User-defined function that can be called for (1) and (2)

The process is written in the above parts in combination with variables or functions provided by the SR-X Series system.

FmtSet.lua file	
SCPVERSION="1.00"	The file version can be determined arbitrarily. (Can be omitted.)
function readformatEvent() Process end	(1) Edit read data
function nameformatEvent(idx) Process end	(2) Edit image file name
function User-defined function Process end	User-defined function that can be called for (1) and (2) (Can be omitted.) ^{*1}
--Comment	Putting 2 hyphens at the beginning enables writing comment.

*1 Multiple user-defined functions can be written as necessary. Skip this if not used.
For writing procedure of user-defined functions, refer to □ "5-7 User-defined function" (Page 13).

(1) Edit read data `function readformatEvent()`

`readformatEvent()` is activated if "Edit data using script" is enabled in the code reader settings. It is activated at the data transmission timing set in the code reader.

Format

```
function readformatEvent()
```

```
    Variable declaration
```

```
    Process
```

```
    setSecondData(variable)
```

The value entered for `setSecondData` is set as the second output data. "5-6 Second output data setting function" (Page 13)

```
    return(variable)
```

The value entered into `return` is output as "Read data".

```
end
```

Example

```
function readformatEvent()
```

```
    local a
```

```
    a="successful reading!"
```

```
    return(a)
```

```
end
```

Declares variable `a`.

Assigns "successful reading!" to variable `a`.

The value stored in `a` is output as "Read data".

* Up to 9990 characters can be assigned to `return`.

(2) Edit image file name `function nameformatEvent(idx)`

`nameformatEvent(idx)` is activated if "Edit image file name using script (only when sending by FTP)" is enabled in the code reader settings. `nameformatEvent(idx)` is activated repeatedly for each image saved after the initial trigger, and the current number of activations is stored in `idx`.

Format

```
function nameformatEvent()
```

```
    Variable declaration
```

```
    Process
```

```
    return(variable)
```

The value entered into `return` is output as "Image file name".

```
end
```

Example

```
function nameformatEvent( idx )
```

```
    local b
```

```
    b="image file"
```

```
    return(b)
```

```
end
```

Declares variable `b`.

Assigns "image file" to variable `b`.

The value stored in `b` is output as "Image file name".

Up to 180 characters can be assigned to `return`.

File names are restricted by the FTP server and OS activating the FTP server. Use the following characters for file names. If characters other than these are used, test to check if operation is normal under the operation environment.

● Usable characters

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789

Reference

- More than one "Variable" and "Process" can be written. For how to define variables, refer to [□□ "2-1 Naming rules" \(Page 4\)](#).
- A different function among functions can be called. [□□ "Calling the function" \(Page 13\)](#)
- Directories can be specified in file names.
To put a image file named "a" into the folder named [image] in the root folder, specify the value such as `image\A` in `return`.
Note if the applicable folder does not exist in the root folder, it does not operate normally.

Comment description

Format

● 1-line comment

```
-- Comment
```

Character strings written after `--` are treated as the comment.

● Multiple-line comment

```
-- [[  
    Comment  
    Comment  
]]
```

Character strings surrounded with `--[[]]` are treated as the comment.

2-1 Naming rules

This describes naming rules for variables, etc. used in the script.

Naming methods

Script file name

The script file name handled by SR-X Series is FmtSet.lua only.

Names defined in the script program

Types of Name defined in the script program are as follows.

- Variable name, array variable name
- Function name

Definitions of variable name and table name

Start the name with a 1-byte alphabet character.

1-byte alphanumeric characters and "_" (underscore) can be used.

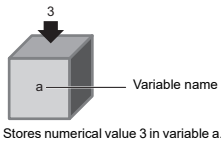
Do not use reserved words.

For reserved words, see "8-2 Reserved words/Language elements" (Page 20).

NOTICE	Do not use "_" (underscore) or a number at the beginning of the name. It may not operate normally.
--------	--

2-2 Variables

Character strings and numerical values used in the script are stored in containers called variables to be handled.



Definition of variable

Values such as numerical values, character strings or calculated results can be assigned to variables.

Declaration of variable

Format

```
local "variable name"
```

Example

<pre>local a</pre>	Declares variable a.
--------------------	----------------------

Assignment and initialization of variable

Storing a value in a variable is called "assignment".

Assigning a value at the time of variable declaration is called "Initialization".

Example

<pre>local a = 0</pre>	Initializes with 0 when declaring variable a.
<pre>a = 1</pre>	Assigns 1 to variable a.

If this is referred with no value assigned to the variable, an error will occur when executing the script.

Characters and numerical values that can be assigned to variables

3 types of values can be assigned to variables: character string, numerical value and logical value

character string	Text notation	: "ABC", "keyence"
	Character code (decimal number) notation	: "\013\010" ([CR][LF])
	Escape sequence	: "\r" (CR)
Numerical value	Integer number	: 1, -1
	Decimal number	: 2.0, -2.0
	Hexadecimal number notation	: 0x0a
Logical value	True	: true
	False	: false

* The condition that no value is assigned to the variable is called nil. If it is referred, an error will occur. Be sure to assign a value before referring to the variable.

character string

● Text notation

Expresses with the text surrounded with "". Numerical values, symbols and character strings can be written in "".

● Character code (decimal number) notation

Expresses the character string by specifying the ASCII code decimal number after \.

● Escape sequence

Expresses control characters and symbols using escape characters (\).

Example

<pre>a = "ABC"</pre>	Assigns character string "ABC" to variable a.
<pre>b = "\013\010"</pre>	Assigns character code (decimal number) notation "\013\010"([CR][LF]) to variable b.
<pre>c = "\r"</pre>	Specifies line break (\r) to variable c with escape sequence.

Numerical value

● Positive and negative integer (decimal number notation)

This expresses positive integer and negative integer as decimal number.

● Positive and negative decimal (decimal number notation)

This expresses positive decimal and negative decimal as decimal number.

▶ Important

Decimal uses double-precision floating-point number, which may have a margin of error.

● Hexadecimal number notation

This expresses 0 or positive integer by adding a hexadecimal number after 0x. The hexadecimal number notation cannot express decimal and negative integer.

Example

<pre>a = 123</pre>	Assigns decimal number integer 123 to variable a.
<pre>b = -1.5</pre>	Assigns decimal number integer -1.5 to variable b.
<pre>c = 0x0f</pre>	Assigns hexadecimal number f to variable c.

Logical value

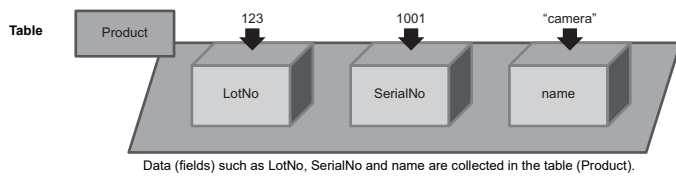
Logical value shows True or False to processed results.

Escape sequence

Pattern	Meaning
<code>\r</code>	[CR](0x0D)
<code>\n</code>	[LF](0x0A)
<code>\a</code>	[BEL](0x07)
<code>\b</code>	[BS](0x08)
<code>\f</code>	[FF](0x0C)
<code>\t</code>	[HT](0x09)
<code>\v</code>	[VT](0x0B)
<code>\\</code>	Backslash
<code>\"</code>	Double quotation
<code>\'</code>	Single quotation
<code>\000</code>	Specifies an arbitrary character code as decimal number.

2-3 Table

The table is the data configuration to collect multiple data.



Definition of table

The table can store multiple data (fields).
Fields can store numerical values and character string data as variables do.

Declaration of table

Format

```
local "table name" = {field 1, field 2, field 3}
```

Reference to data within the table

Table name.field name

Example 1) Table declaration and initialization

```
local Product = { LotNo=123 , Serial No=1001 , name="camera" }
```

Declares table Product.
LotNo=123
Serial No =1001
name ="camera"

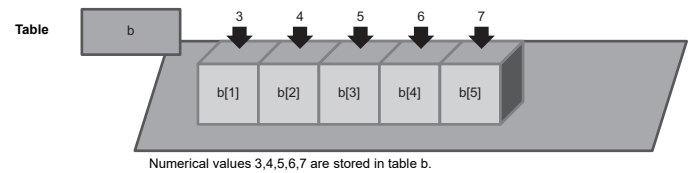
Example 2) Reference to data variables within the table

```
local a = 0  
local Product = { LotNo=123 , Serial No=1001 , name="camera" }  
a = Product.name
```

Declares variable a.
Declares table Product.
Stores the name field of
the table Product in a.

2-4 Array

The array is a type of table and a data column that can store multiple values.
Similar to normal variables, reference and assignment can be made.



Definition of array

Values such as numerical values, character strings or calculated results can be assigned to arrays.

Naming rules for array

Array names must start with a 1-byte alphanumeric character for the first character. For the second character and after, 1-byte alphanumeric characters and "_" (underscore) can be used. Array elements start from [1].

Declaration of array

Format

```
local "arrayname" = { }
```

Example

```
local b = { }
```

Declares array b.

Assignment and initialization of array variable

Assign values to array variables using { } and values. Specify multiple values by dividing "," (comma).

Example

```
local a = { }  
a = { 3, 4, 5, 6 }  
local b = { 16, 51, 55 }  
b[3] = 10  
b[4] = 11
```

Declares array a.
Assigns the 4 values 3, 4, 5, and 6 to array a.
Declares array b and initializes it with the 3 values 16, 51, 55.
Assigns 10 to the third value of array b.
Assigns 11 to the fourth value of array b (adds 1 element).

3-1 Calculation

This describes arithmetic operators used for general calculations.

Arithmetic operator

Format

Arithmetic operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder (from division)
^	Power

On the left-hand side and the right-hand side of each operator, write an expression using numerical values or variables of numerical values.

Example

```
A = B + 1
```

Assigns the result of variable B plus 1 to variable A.

Example

Examples of calculations and results (a=5.5, b=2)

Arithmetic expression	Result
a + b	7.5
a - b	3.5
a * b	11
a / b	2.75
a % b	1.5
a ^ b	30.25

Important

If character strings are assigned to variables and they cannot be converted to numerical values during calculation, an error will occur on execution.

Reference

To obtain only the integer part of division result, use arithmetic function math.floor.(Page 11)

3-2 Comparison

This describes comparative operators used when comparing numerical values or logical values.

Comparative operator

Format

Comparative operator	Meaning
<	smaller
<=	smaller or equal
>	greater
>=	greater or equal
==	equal
~=	different

On the left-hand side and the right-hand side of each operator, write numerical values, variables of numerical values or arithmetic expression.

Example

Examples of calculations and results (a=10, b=1)

Arithmetic expression	Result
a > b	true
a < b	false
a == 10	true
a ~= 10	false

Important

Comparison is also possible by assigning character strings to variables. However, when variables are evaluated by conditional expression, and if they cannot be converted to numerical values, an error will occur on execution.

3-3 Combination of condition

This describes logical operators used when combining conditions.

Logical operator

Format

Logical operator	Meaning
and	Logical AND operator: If the first argument is false or nil, the value is returned. If not, the second argument is returned.
or	Logical OR operator: If the first argument is different from false or nil, the first argument is returned. If not, the second argument is returned.
not	Logical NOT operator: If the next value is false or nil, true is returned. If not, false is returned.

Example

Examples of calculations and results

Arithmetic expression	Result
true and true	true
true or true	true
true and false	false
true or false	true
not false	true
not true	false
10 or 30	10
nil or 30	30
false or nil	nil

3-4 Character string concatenation

This describes concatenation operators used to connect one character string to another.

Concatenation operator

Format

```
variable 1..variable 2
```

Expresses using 2 periods ".." between variables.

Example

```
local a = "Keyence"
```

Declares variable a and initializes with character string "Keyence".

```
local b = "Auto-ID"
```

Declares variable b and initializes with character string "Auto-ID".

```
local c
```

Declares variable c.

```
c = a..b
```

Assigns values after a and b are connected.

```
Value of c: "KeyenceAuto-ID"
```

This section describes how to search character strings with specific patterns (Pattern matching) from character strings.

Pattern

Patterns used to search alphabets, numbers, symbols, etc. from character strings.

Meta-character

Pattern	Meaning
.	All characters (arbitrary 1 character)
%a	Character
%c	Control character (0x01 to 0x1F, 0x7F)
%d	Number
%l	Alphabet lowercase
%p	Symbol (! " # \$ % & ' () * + , - . / : ; > = < ? @ [\] ^ _ ` { })
%s	Space character (0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x20)
%u	Alphabet uppercase
%w	Alphabet and number
%x	Hexadecimal number (0 to 9, a to f, A to F)
%z	null character
% character ^{*1}	Literal character

*1 Specify characters other than alphanumeric characters

Quantifier

Pattern	Meaning
+	Matches with 1 or more times repetitions of the previous character.
*	Matches with 0 or more times repetitions of the previous character. (Longest match)
-	Matches with 0 or more times repetitions of the previous character. (Shortest match)
?	Matches with 0 or 1 time repetition of the previous character.

Range specification

Pattern	Meaning
[0-9]	Number (0 to 9)
[^0-9]	Not number (0 to 9)
[a-z]	Alphabet lowercase (a to z)
[A-Z]	Alphabet uppercase (A to Z)
[a-zA-Z]	Alphabet lowercase and alphabet uppercase (a to z and A to Z)
[0-9a-zA-Z]	Number and alphabet (0 to 9, a to z and A to Z)

* If ^ (caret) is added to the beginning of the pattern, the match is fixed to the beginning of the target character string.

If \$ (dollar) is added to the end of the pattern, the match is fixed to the end of the target character string.

4-1

Conditional branching of process

To branch processes, use if sentences.

if sentence

Format

● For 1 branch

```
if condition 1 then
    process 1
end
```

If "condition 1" is fulfilled, "process 1" is executed.
If "condition 1" is not fulfilled, no process is executed.

● For 2 branches (using else)

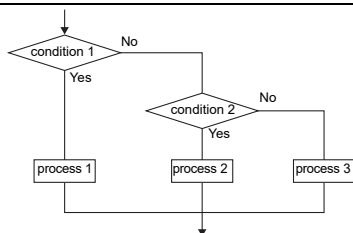
```
if condition 1 then
    process 1
else
    process 2
end
```

If "condition 1" is fulfilled, "process 1" is executed.
If "condition 1" is not fulfilled, "process 2" is executed.

● For 3 branches (using elseif)

```
if condition 1 then
    process 1
elseif condition 2 then
    process 2
else
    process 3
end
```

If "condition 1" is fulfilled, "process 1" is executed.
If "condition 1" is not fulfilled but "condition 2" is fulfilled, "process 2" is executed.
If both "condition 1" and "condition 2" are not fulfilled, "process 3" is executed.



Reference The "elseif condition then process" sentence can be written multiple times.

Conditional expression

For "condition" of if sentences, write conditional expressions using comparative operators as below.

Comparison type	Conditional expression	Description
Character string comparison	<code>if (str == "A") then</code>	If str is equal to "A"
	<code>if (str ~= "A") then</code>	If str is not equal to "A"
Numerical value comparison	<code>if (i < 10) then</code>	If i is less than 10
	<code>if (i <= 10) then</code>	If i is 10 or less
	<code>if (i == 10) then</code>	If i is equal to 10
	<code>if (i >= 10) then</code>	If i is 10 or more
	<code>if (i > 10) then</code>	If i is more than 10
Character string comparison	<code>if (i ~= 10) then</code>	If i is a number other than 10
	<code>if (bool == true) then</code>	if bool is true
	<code>if (bool ~= true) then</code>	if bool is not true

For details of comparative operators, refer to ["3-2 Comparison"](#) (Page 6).

Example 1

```
if type < 10 then
    termID = 1
end
```

If type is less than 10, assign 1 to termID.

Example 2

```
if type < 10 then
    termID = 10
elseif type < 20 then
    termID = 20
else
    termID = 30
end
```

If type is less than 10, assign 10 to termID.
If type is 10 or more and less than 20, assign 20 to termID.
If type is 20 or more, assign 30 to termID.

Reference Up to 20 nests can be made for the conditional branching.

4-2 Repeat process

This section describes processes when repeating processes.

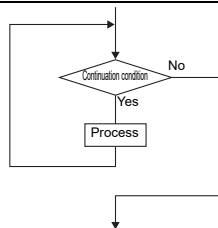
while sentence

while sentences perform repeat process by specifying continuation conditions.

Format

```
while continuation
condition do
    Process
end
```

As long as the continuation condition is fulfilled, the process is repeated.
If the continuation condition is not fulfilled, the repeat process ends.



Example

```
while (i <= 10) do
    i = i + 1
end
```

If i is 10 or less, the value for i is increased by one.

for sentence

for sentences repeat process using the counter to count the number of repetitions.

Format

● Base

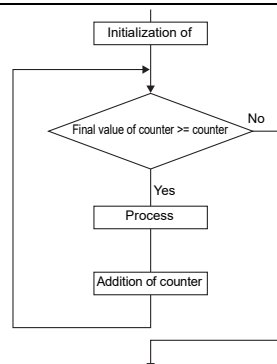
```
for Initialization expression
for counter, and final value do
    Process
end
```

- Sets the default value to the counter (variable). The process is repeated until the value exceeds the final value.
- Each time the process ends, the counter increases by one.

● Specifying "increment of counter"

```
for Initialization expression for counter,
final value and increment of counter do
    Process
end
```

- Sets the default value to the counter (variable). The process is repeated until the value exceeds the final value.
- Each time the process ends, the counter increases by positive and negative integer value specified.



Example

```
for i = 1, 5 do
    a = a + 2
end
```

If i is 5 or less, the value for a is increased by two.

repeat until sentence

repeat until sentences execute process at least once. After the last condition is checked, whether to repeat is judged.

Format

```
repeat
    Process
until condition
```

- After the process is executed once, the process is repeated until the condition is fulfilled.

Example

```
repeat
    a = a + 2
    i = i + 1
until (i <= 10)
```

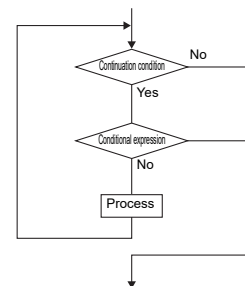
Regardless of the i value, the value for a is increased by two, and the value for i is increased by one.
If i is 10 or less, the value for a is increased by two.

break sentence

Use break when going out in the middle of repeat process of while sentences or of for sentences.

Format

```
while continuation
condition do
    Conditional expression
    Process
    break
end
end
```



* break can be written only just before end.

To call break in the middle of the sentence, write as below.

Example

```
do
    break
end
```

5-1 Data acquisition function

These functions are used to acquire information related to code read by the SR-X Series.

readResult(idx)



Acquires information related to read data

Argument	idx	: Index for read information array ^{*1}
----------	-----	--

- ^{*1} The number of arrays can be obtained using the common function "readCount()".
If idx is omitted, the first code is specified.

Functions used in readResult(idx)

These functions obtain information of read data. Use as readResult(n):***
Example) readResult(n):readData()

Function name	Argument	Return value
readData()	None	Read data
symbolType()	None	Code type (Refer to  "8-3 Code type" (Page 20).)
symbolIdentifier()	None	Symbol identifier (3 characters) (Refer to  "8-3 Code type" (Page 20).)
bankNo()	None	Bank number
regionNo()	None	Region number
scanTimes()	None	Scan count
cornerCoordinates()	None	Code top coordinates: "X1/Y1:X2/Y2:X3/Y3:X4/Y4"
centerCoordinate()	None	Code center coordinates: "X1/Y1"
eccQuality()	None	Unused error correction (0 to 100)
matchingLevel()	None	Matching Level (0-100) ^{*1}
i15415Verification()	None	ISO/IEC15415 verification result ^{*1}
aimVerification()	None	ISO/IEC TR 29158(AIM DPM-1-2006) verification result ^{*1}
i15416Verification()	None	ISO/IEC15416 verification result ^{*1}
as9132Verification()	None	SAE AS9132 verification result ^{*1}
semiT10Verification()	None	SEMI T10-070 verification result ^{*1}
i16022Verification()	None	ISO/IEC16022 verification result ^{*1}
japanMedicalVerification()	None	Verification result of the Japan ethical drug barcode ^{*1}
imageFileNames()	None	Image file name ^{*2}
readingDuration()	None	Reading duration [ms]
errCode()	None	Detail error code This is output only for burst. ^{*1}
ppc()	None	PPC
cellNum()	None	Number of cells:"Vertical/horizontal" ^{*3}
cellSize()	None	Cell size (mm) ^{*3}
codeSize()	None	Code size (mm):"Vertical/horizontal" ^{*3}
scanSkewAngle()	None	Angle (skew) ^{*4}
scanPitchAngle()	None	Angle (pitch) ^{*4}

- ^{*1} The functions of the code reader main unit must be enabled beforehand.
^{*2} When using the script for the edit image file name, the file name after the image file is changed is output. When multiple image files exist, the value of the Interdelimiter is the same value set for the SR-X Series.
^{*3} This function is designed for 2D codes.
When a barcode, not a 2D code, is read, the narrow bar width is stored during the execution of the cellSize() function. When a barcode is read, cellNum() and codeSize() cannot be executed.
^{*4} Only SR-X300/X300W is equipped.
• When using the advanced multi head, the append data of the slave is all obtained with readData().

Functions used when reading the GS1 composite

Function name	Argument	Return value
readData()	None	Read data ^{*1}
symbolIdentifier()	None	Symbol identifier ^{*1}
cornerCoordinates()	None	Code top coordinates: ^{*2} "X1/Y1:X2/Y2:X3/Y3:X4/Y4:X1/Y1:X2/Y2:X3/Y3:X4/Y4"
centerCoordinate()	None	Code center coordinates: ^{*2} "X1/Y1:X1/Y1"
readDataCompositeLen1D()	None	Data length of the GS1 composite 1D code

- ^{*1} When reading the GS1 composite (JAN/EAN/UPC), the 1D code and 2D code are connected with a composite delimiter and then output.
^{*2} When reading the GS1 composite code, values of the 1D code and 2D code are obtained separately, connected with a partition mark, and then output.

imageResult(idx)

Acquires information related to read images

Argument	idx	: Index for image name information array ^{*2}
----------	-----	--

- ^{*2} The number of arrays can be obtained using the common function "imageCount()".
If idx is omitted, the first code is specified.

Functions used in imageResult(idx)

These functions obtain information of captured images. Use as imageResult(n):***
Example) imageResult(1):bankNo()

Function name	Argument	Return value
bankNo()	None	Bank number or nil
imageFileName()	None	Image file name
readDatas()	None	Character string data of the barcode itself or nil (Concatenated when multiple data are present.)
readResultIdx()	None	Index table of append data (readResult(n)) or nil

Data entry to readResult and imageResult

Example 1

When reading in the "Standard read mode" is performed on SR-X Series by registering 3 parameter banks for which alternate reading is enabled.

• Read OK

When reading is successful with the first bank

readResult(1)

imageResult(1)

When reading is successful with the second and third banks

readResult(1)

imageResult(1)

• Read error

When reading fails

readResult(1)

imageResult(1)

imageResult(2)

imageResult(3)

5-2

Common functions (Basic)

These functions can be used in common.

Character string processing functions

Function name	Argument	Return value
<code>left(str, m)</code>	str: character string m: number of digits (1 to 9990)	Character string of m digits counted from the start of the character string (str).
<code>mid(str, s, m)</code>	str: character string s: 1 to 9990 m: 1 to 9990	Character string of m digits counted from s digit position from the start of the character string (str). If m is not specified, the range from the start position (s) to the end is specified. If the start position is outside the range of character string data, there is no return value.
<code>right(str, m)</code>	str: character string m: 1 to 9990	Character string of m digits counted from the end of the character string (str)
<code>field(str, m, sep)</code>	str: character string m: 1 to 9990 sep: character (1 character)	The mth character string after dividing the character string (str) with the separator (sep)

Reader status acquisition functions

Function name	Argument	Return value
<code>time()</code>	None	Time
<code>readCount()</code>	None	Total number of reading results ^{*1}
<code>imageCount()</code>	None	Total number of saved images ^{*1}
<code>result()</code>	None	Comparison result 0: Comparison OK, 1: Comparison NG, 2: ERROR
<code>ipAddress()</code>	None	IP address (delimited by underscore) ex)192_168_100_001
<code>masterSlaveID()</code> ^{*2}	None	Master/Slave ID
<code>masterSlaveGroupName()</code>	None	Master/Slave group name

^{*1} When the Master/Slave function is used, result data cannot be obtained correctly.

^{*2} Only used for multi-drop links.

Debug function

Function name	Argument	Return value
<code>print(str)</code>	str: character string	Outputs character strings (str) when debug is executed.

* The output setting turns ON after the "SCPCBG,1" command is executed. □ "6-1 Debug methods" (Page 13)

5-3

Common functions (Advanced)

These functions can be used in common. Following is the more advanced functions.

Numeric calculation

Function name	Argument	Return value
<code>math.max(v,...)</code>	v: numerical value	Maximum value among arguments
<code>math.min(v,...)</code>	v: numerical value	Minimum value among arguments
<code>math.abs(v)</code>	v: numerical value	Absolute value
<code>math.floor(v)</code>	v: numerical value	Integer with decimals truncated
<code>math.ceil(v)</code>	v: numerical value	Integer with decimals rounded up
<code>math.pow(a,n)</code>	a: numerical value n: numerical value	a ⁿ
<code>math.sqrt(v)</code>	v: positive numerical value	√v
<code>math.deg(rad)</code>	rad: numerical value	Converts radian values to angular degrees.
<code>math.rad(deg)</code>	deg: numerical value	Converts angular degrees to radian values.
<code>math.sin(rad)</code>	rad: numerical value	sin
<code>math.cos(rad)</code>	rad: numerical value	cos
<code>math.tan(rad)</code>	rad: numerical value	tan
<code>math.asin(rad)</code>	rad: numerical value	arc sin
<code>math.acos(rad)</code>	rad: numerical value	arc cos
<code>math.atan(rad)</code>	rad: numerical value	arc tan

Character string operation

Function name	Argument	Return value
<code>string.len(str)</code>	str: character string	Character string (str) length
<code>string.lower(str)</code>	str: character string	Converts the character string (str) to lowercase characters.

Function name	Argument	Return value
<code>string.upper(str)</code>	str: character string	Converts the character string (str) to uppercase characters.
<code>string.rep(str,n)</code>	str: character string n: numerical value	Character string that repeated the character string (str) n times
<code>string.reverse(str)</code>	str: character string	Reverses the alignment sequence of character string (str).
<code>string.sub(str,i,j)</code>	str: character string i: numerical value j: numerical value	Portion of character string from ith character to jth character of the character string (str)
<code>string.byte(str,i)</code>	str: character string i: numerical value	Returns the character code for the ith character of the character string (str)
<code>string.char(c1,c2,...)</code>	c1: numerical value	Character string for the character code.
<code>string.find(str,p)</code>	str: character string p: character string or regular expression	Searches characters that match p among the character string (str), and then returns the digit number at the head.
<code>string.match(str,p)</code>	str: character string p: character string or regular expression	Returns characters that match p among the character string (str).
<code>string.gmatch(str,p)</code>	str: character string p: character string or regular expression	Used for for loop. Each time the characters that match p among the character string (str), and then returns the digit number at the head.
<code>string.gsub(str,p,r,n)</code>	str: character string p: character string or regular expression r: character string n: numerical value (Can be omitted)	Replaces characters that match p among the character string (str) with r. If n is specified, the first n characters in the character string are replaced.

Array operation

Function name	Argument	Return value
<code>table.concat(tbl,sep,i,j)</code>	tbl: array name sep: character (1 character) i: numerical value j: numerical value	Character strings combined after the data from tbl[i] to tbl[j] in the array (tbl) is delimited by sep.
<code>table.insert(tbl,i,v)</code>	tbl: array name i: numerical value v: numerical value	Inserts v into the tbl[i] position in the array (tbl). Elements following the ith position are moved backward by one. No return value.
<code>table.maxn(tbl)</code>	tbl: array name	Returns the largest index value in the array (tbl).
<code>table.remove(tbl,i)</code>	tbl: array name i: numerical value	Deletes the element at the tbl[i] position in the array (tbl). Elements following the ith position are moved forward by one. No return value.
<code>table.sort(tbl)</code>	tbl: array name	Sorts the elements in the array (tbl). No return value.

Bit calculations

Function name	Argument	Return value
<code>bitAnd(a,b)</code>	a: numerical value b: numerical value	Logical disjunction (OR) of a and b
<code>bitNot(a)</code>	a: numerical value	Negation of a
<code>bitOr(a,b)</code>	a: numerical value b: numerical value	Logical conjunction (AND) of a and b
<code>bitXor(a,b)</code>	a: numerical value b: numerical value	Exclusive disjunction (exclusive OR) of a and b
<code>bitLeftShift(a,b)</code>	a: numerical value b: numerical value	Shifts a to the left by b bits
<code>bitRightShift(a,b)</code>	a: numerical value b: numerical value	Shifts a to the right by b bits
<code>bitLeftRotate(a,b)</code>	a: numerical value b: numerical value	Rotates a to the left by b bits
<code>bitRightRotate(a,b)</code>	a: numerical value b: numerical value	Rotates a to the right by b bits
<code>bitExtract(a,b)</code>	a: numerical value b: numerical value	Returns the bit in position b from the lowest order position of a
<code>bitReplace(a,b,c)</code>	a: numerical value b: numerical value c: numerical value	The value in which the bit in position b from the lowest order position of a is replaced with bit c

Bit calculation functions are calculations performed in units of 32 bits.

Others

Function name	Argument	Return value
<code>assert(v, mes)</code>	v: judgment mes: character string	If v is False, mes is output. The program is force-quit.
<code>error("", 1)</code>	-	Errors (ScriptError) out explicitly and terminates the program.
<code>print(mes)</code>	mes: character string	Outputs messages for debug.
<code>tonumber(v)</code>	v: value	Converts to numerical values. Returns nil if value conversion is not possible.
<code>tostring(v)</code>	v: value	Converts to character strings.
<code>type(v)</code>	v: value	Variable type

5-4

Controlling output terminals and triggers

Controlling output terminals

Function name	Argument	Return value
outonEvent (n)	n: output terminal number	None (The corresponding output terminal turns on.)

When using a script to control OUT terminals, the following settings need to be configured after setting "Enable script features".

(This is activated when "SCRIPT CONTROL" is checked only for the OUT terminals to be controlled by the script.)

Example settings: When using a script to control all OUT terminals

The screenshot shows the 'Output Terminal' configuration window. Under the 'Option' section, the 'SCRIPT CONTROL' checkbox is checked for all three output functions (OUT1, OUT2, and OUT3). The 'Output Duration' is set to 500ms for all three, and the 'Output polarity' is set to 'Norm. open'.

Controlling triggers

Function name	Argument	Return value
invokeNextShot (n)	n: bank number	Instruction to continue capturing images If n is left blank, the SR operates with a bank on which alternating is enabled.

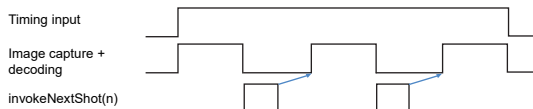
If you turn off the timing input, reading will end without capturing the next image regardless of whether this function is executed or not.

Timing diagrams

• Normal



• When using invokeNextShot(n)



Trigger status

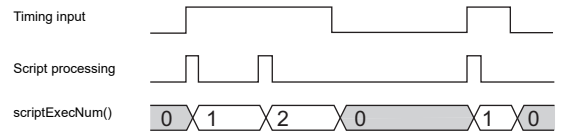
Function name	Argument	Return value
isTriggerActive ()	None	Trigger status 0: triggers are off 1: triggers are on
scriptExecNum ()	None	Number of times the script was called while the trigger was applied

Timing diagrams

• isTriggerActive



• scriptExecNum()



Trigger control precautions

Limits when performing trigger control

	Script processing	Out signals			LED ²	EtherNet/IP, PROFINET		Saving images
		OK	ERROR	SCRIPT CONTROL		Completion bit ³	Read error bit	
When capturing images	✓			✓				
When reading is judged to be successful ¹		✓			Green			✓
When outputting data						✓		

- *1 This refers to the case in which the conditions for successful reading specified with the SR-X Series are met. When using script triggers, reading is not judged as resulting in an error.
- *2 This is the color displayed by the OK/ERROR LED (green/red) of the SR-X Series main unit. When using script triggers, this LED does not light in red.
- *3 This indicates "Read Complete" or "Ext. Request Complete" when using EtherNet/IP or PROFINET with the SR-X Series.

5-5

Confirmation command function

You can obtain the reply result of SR-X Series confirmation commands (RN/RP/RC/RB/RD).

For details on the command numbers and the reply details, see the user's manual.

Function name	Argument	Return value
execCmd ("n")	n: confirmation command	Confirmation command reply result
Applicable commands		RN/RP/RC/RB/RD, KEYENCE, EMAC

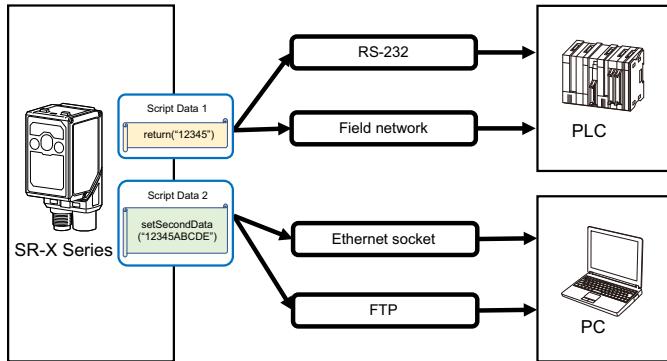
- * You cannot use commands other than those listed above.

5-6

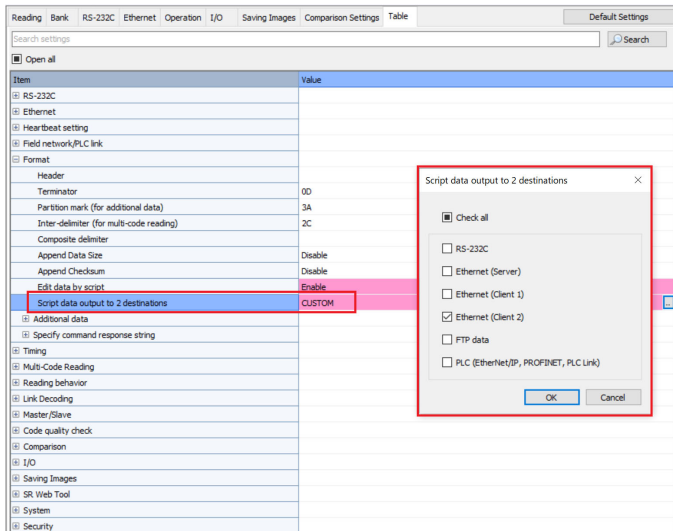
Second output data setting function

The SR-X Series allows two types of output data to be set for different data output destinations.

Function name	Argument	Return value
setSecondData(str)	str: Character string	None



When using a script to set a second output data, the following settings need to be configured after setting "Enable script features".



5-7

User-defined function

This section describes functions that can be uniquely defined by users.

User-defined function

Format

● Definition of function

```
function function name (argument)
    Variable declaration
    Process
    return (variable)
end
```

The value entered into return is the return value.

● Calling the function

Function name (value)

Example 1

```
function readformatEvent()
    local a
    a = test_1()
    a = a .. "ABC"
    return (a)
end

function test_1()
    local c
    c = 123
    return (c)
end
```

Declares variable a.
Stores the result of test_1() in variable a.
Appends character string "ABC" to variable a.
Outputs the result.

Defines the user-defined function test_1().
Declares variable c.
Assigns "123" to variable c.
The value of variable c is the return value.

Execution result 123ABC

6-1

Debug methods

This section describes debug methods when the script is executed.

Operation check procedure

- 1 Send the FmtSet.lua file to SR-X Series using AutoID Network Navigator.
- 2 Start the AutoID Network Navigator [Terminal] and click [LON] to start reading.
- 3 Read with the code reader and check if the result is correctly reflected.
 - For editing read data
→ Check if the output data is correct.
 - For editing image file name
→ When using the setting to send image files via FTP, check the name of image file uploaded on the specified FTP server.

* While reading, if an error message such as "Load Error" is displayed, debug according to the following debug methods.

Error message	Description
LoadError	Script file loading failed.
ReadResultScriptError	An error has occurred in readformatEvent.
ImageResultScriptError	An error has occurred in nameformatEvent.
TimeoutError	Script timeout error. The script execution took too long. *1
ScriptError	Other errors.

*1 The time for script timeout should be 3 seconds.

Debug methods

Error details check

- 1 Send the "SCPERR" command to SR-X Series with AutoID Network Navigator [Terminal]. (Executing this command will display the error details.)
- 2 "error ~~~ FmtSet.lua: ○ : ~~~" is displayed. The number applied to ○ is the corresponding row in error.
Open FmtSet.lua with the text editor and repair the source code.

print(mes) display (Page 11)

To check (debug output) the variable value in the source code of FmtSet.lua, use the print function.

- 1 Write the print function on where the variable value is to be checked in the source code of FmtSet.lua.
- 2 Send the "SCPDBG,1" command to SR-X Series with AutoID Network Navigator [Terminal]. (Executing this command enables debug output when reading.)

- 3 Click [LON] to start reading.
The content specified as the argument of the print function is output.
Referring to this output result, open FmtSet.lua with the text editor and repair the source code.
- 4 After the source code is repaired, send the "SCPDBG,0" command and disable debug output.

6-2 Error message list

After the "SCPDBG,1" command is executed, if the script file is sent to the code reader, the following error messages are displayed.

Error message list

Load errors

Error message	Description
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: unfinished long comment near '<eof>'	"]]" of the comment "-[[]]" is missing. Add "]]" to the end of the comment.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: 'end' expected (to close 'function' at line 1) near '<eof>'	end is missing for "function" on the first line. Write end at the correct position.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: 'end' expected (to close 'for' at line 2) near '<eof>'	end is missing for "for" on the second line. Write end at the correct position.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: ')' expected (to close '(' at line 1) near '<eof>'	")" for "(" is missing. Write ")" at the correct position.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: <name> or '...' expected near 'end'	When declaring function, ")" for "(" is missing. Write ")" at the correct position.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: '=' expected near '+'	Grammatical error (Arithmetic expression error). Check if the arithmetic expression is correct.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: '=' expected near '=='	Grammatical error (Is == used instead of =?) Correct == to =.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: 'then' expected near '='	Grammatical error (Is = used instead of ==?) Correct = to ==.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: unexpected symbol near '='	Grammatical error (Is a value assigned to the reserved word?) Do not assign any value to the reserved word.
error loading module 'FmtSet' from file 'FmtSet.lua': FmtSet.lua:0: malformed number near '1.a'	Numerical value writing error. Write numerical values correctly.

Execution errors

Error message	Description
FmtSet.lua:0: attempt to perform arithmetic on a string value	Was the arithmetic expression written with character strings? Write it correctly.
FmtSet.lua:0: attempt to perform arithmetic on global 'a' (a string value)	
FmtSet.lua:0: attempt to concatenate a nil value	Concatenation failed. Check if values other than character strings are not specified.
FmtSet.lua:0: attempt to concatenate global 'a' (a table value)	
FmtSet.lua:0: attempt to call global 'a' (a nil value)	Undeclared functions were called. Check if the name of the function to call is correct.
FmtSet.lua:0: attempt to call global 'a' (a number value)	
FmtSet.lua:0: attempt to index global 'a' (a nil value)	Undeclared tables were called. Check if the name of the table to call is correct.
FmtSet.lua:0: attempt to index global 'a' (a string value)	
FmtSet.lua:0: bad argument #1 to 'cos' (number expected, got string)	Wrong argument.
FmtSet.lua:0: bad argument #2 to 'max' (number expected, got nil)	Specify the correct argument.

Other errors

Error message	Description
not enough memory	Memory shortage has occurred. Review the program.
attempt to call global 'readformatEvent' (a nil value)	The readformatEvent function does not exist. Write the readformatEvent function.
attempt to call global 'nameformatEvent' (a nil value)	The nameformatEvent function does not exist. Write the nameformatEvent function.
FmtSet.lua:0: cancel	Aborted while the program is running. <ul style="list-style-type: none"> Aborted by the CANCEL command Aborted due to the timeout of the program execution time Check if the program is not supposed to go into the infinite loop or the process time of the program is too long.

Sample programs

1 Editing read data

- 1-1 Extracting the first 1 character and the last 2 characters of read data.
- 1-2 Deleting the first 1 character of read data.
- 1-3 Deleting all unnecessary zeros at the beginning of barcode.

2 Count

- 2-1 Counting the number of readings

3 Conditional branching

- 3-1 Outputting the type of read codes
- 3-2 Branching according to the read results (OK/NG/ERROR)
- 3-3 Branching after looking at the first 1 character of read data

4 Comparison

- 4-1 Comparison when specific characters are included in read data
- 4-2 Comparison when control characters are included in read data
- 4-3 Deleting control characters when they are included in read data
- 4-4 Deleting <LF>(0x0A) when it is included in read data
- 4-5 Date comparison

5 Calculations

- 5-1 Obtaining angular degrees of a tilted 2D code
- 5-2 Obtaining angular degrees of a tilted barcode
- 5-3 Bit calculations

6 Editing read image file names

- 6-1 Appending read data to the read image file name
- 6-2 Appending date and time to the read image file name
- 6-3 Appending date and time and read data to the read image file name

7 Controlling output terminals

- 7-1 Controlling output terminals (OK/NG/ERROR)
- 7-2 Preventing duplicate reading

8 Controlling triggers with scripts

- 8-1 Continuing reading until the character "S" is read at the start of the data
- 8-2 Detecting timing input turning off

9 Confirmation commands

- 9-1 MAC address confirmation command
- 9-2 Reader explanation confirmation command

1 Editing read data

1-1 Extracting the first 1 character and the last 2 characters of read data.

```
function readformatEvent()
    local read_data
    read_data = readResult():readData() -- Obtains the code reading result.
    read_data = left(read_data,1)..right(read_data,2) -- Extracts 1 character on the left
                                                    and 2 characters on the right.
    return (read_data)
end
```

Execution result

Read data : keyence



Execution result : kce

1-2 Deleting the first 1 character of read data.

```
function readformatEvent()
    local read_data
    local datalength
    read_data = readResult():readData() -- Stores read data in the variable.
    datalength = string.len(read_data) -- Obtains the length of read data
    read_data = mid(read_data,2,datalength-1) -- Extracting character strings from
                                                    the second character to the end.
    return (read_data)
end
```

Execution result

Read data : keyence



Execution result : eyence

1-3 Deleting all unnecessary zeros at the beginning of barcode.

```
function readformatEvent()
    local read_data = readResult():readData() -- Obtains the code reading result.
    local data
    data = string.gsub(read_data,"^0+","") -- Remove consecutive zeros from
                                                    the start.
    return (data)
end
```

Execution result

Read data : 00000143



Execution result : 143

2 Count

2-1 Counting the number of readings

```
local count = 0 -- Retains count.*1
function readformatEvent()
    local maxCount = 5 -- If codes are read 5 times or more,
    local data -- "5 count over" is returned.
    while count < maxCount do
        count = count + 1
        data = readResult():readData() .. ":count_data is"..count
        return data
    end
    return "5 count over"
end
```

Execution result

Read data : keyence



Number of readings : n (n=1 to 5)

n=6 and more

Execution result : keyence:count_data is n

5 count over


*1 The variable value defined here is initialized when the script file is sent again or the SR main unit is reset.

* When the Master/Slave function is used, the count is not correct.

3-1 Outputting the type of read codes


```
function readformatEvent()
  local data = readResult():symbolType() -- Stores code types.
  if data == 1 then -- Conditionally branches according
    return "QR" -- to the type of code.
  elseif data == 2 then
    return "DataMatrix"
  elseif data == 5 then
    return "GS1 DataBar"
  elseif data == 6 then
    return "CODE39"
  elseif data == 7 then
    return "ITF"
  elseif data == 9 then
    return "Codabar"
  elseif data == 10 then
    return "JAN/EAN/UPC"
  elseif data == 11 then
    return "CODE128"
  elseif data == 0 then
    return "no data"
  end
  return "other"
end
```

Execution result	
Read data	: keyence 
Execution result	: DataMatrix

* The symbolType function in  "5-1 Data acquisition function" (Page 10) is used.

3-2 Branching according to the read results (OK/NG/ERROR)

```
function readformatEvent() -- Branches according to the read
  local r_data = result() -- results (OK/NG/ERROR).
  local data = readResult():readData()
  if r_data == 0 then -- OK image
    return "OK: "..data
  elseif r_data == 1 then -- Comparison NG
    return "NG: "..data
  elseif r_data == 2 then -- READ ERROR
    return "ERROR: "..data
  end
  return "result has no return"
end
```

Execution result	
Read data	: keyence 
Execution result	: OK:keyence (Reading success)

3-3 Branching after looking at the first 1 character of read data

```
function readformatEvent()
  local data = readResult():readData() -- Obtains the code reading result.
  local header = left(data,1) -- Obtains the first 1 character.
  if header == "1" then
    return "A"
  elseif header == "2" then
    return "B"
  else
    return "Z"
  end
  return "other"
end
```

Execution result	
Read data	: 1SR-D100  2SR-D110  3SR-D100HA 
Execution result	: A B Z

4-1 Comparison when specific characters are included in read data


```
function readformatEvent()
  local o_data = ""
  if string.match(readResult():readData(),"456") then -- When "456" is included in read data
    o_data = readResult():readData() .. "_OK" -- Appends "_OK" to the end of read data.
  else -- Other cases
    o_data = readResult():readData() .. "_NG" -- Appends "_NG" to the end of read data.
  end
  return (o_data)
end
```

Execution result	
Read data	: 123456789 
Execution result	: 123456789_OK

 "3-5 Pattern matching" (Page 7) is used.)

4-2 Comparison when control characters are included in read data

```
function readformatEvent()
  local o_data = ""
  if string.match(readResult():readData(),"%c") then -- When control characters are included in read data
    o_data = "OK" -- Outputs "OK".
  else -- Other cases
    o_data = "NG" -- Outputs "NG".
  end
  return (o_data)
end
```

Execution result	
Read data	: ABC<HT>123 
Execution result	: OK

 "3-5 Pattern matching" (Page 7) is used.)

4-3 Deleting control characters when they are included in read data

```
function readformatEvent()
  local read_data
  read_data = readResult():readData()
  read_data = string.gsub(read_data,"%c","") -- Deletes control characters
  read_data = string.gsub(read_data,"%z","") -- Deletes NULL<0x00>
  return (read_data)
end
```

Execution result	
Read data	: <SOH><SOH>123 
Execution result	: 123

 "3-5 Pattern matching" (Page 7) is used.)

4-4 Deleting <LF>(0x0A) when it is included in read data

```
function readformatEvent()
  local read_data
  read_data = readResult():readData()
  read_data = string.gsub(read_data,"\010","") -- Deletes [LF]<0x0A>(010 in decimal number)
  return (read_data)
end
```



Execution result	
Read data	: ABC<LF>123 
Execution result	: ABC123

4-5 Date comparison

```
function readformatEvent()
    local year = 2012
    local month = 12
    local date = 22
    local code_Year
    local code_Month
    local code_Date
    local read_data = readResult():readData()
    if string.len(read_data) < 8 then
        return("reading data is not correct")
    end
    code_Year = tonumber(left(read_data,4)) -- Extracts date information and
    code_Month = tonumber(mid(read_data,5,2)) -- converts it to numerical values.
    code_Date = tonumber(mid(read_data,7,2))

    if code_Year < year then
        return "The year is not correct"
    elseif code_Month < month then
        return "The month is not correct"
    elseif code_Date < date then
        return "The date is not correct"
    end
    return ("OK") -- Returns OK.
end
```

Execution result

Read data : 22001225.  20111222 
Execution result : OK The year is not correct



This is used to control expiration dates, etc. The date data in the barcode is compared with the reference date and output.

5 Calculations

5-1 Obtaining angular degrees of a tilted 2D code

```
function readformatEvent()
    if result() == 2 then -- Displays the message at read
        return "READ_ERROR" error.
    end
    local o_data
    local data = readResult():cornerCoordinates() -- Obtains the coordinates at 4 positions of code.
    local x1_y1 = field(data,1,":") -- x1/y1:x2/y2:x3/y3:x4/y4
    local x2_y2 = field(data,2,":")
    local x1 = tonumber(field(x1_y1,1,"/"))
    local y1 = tonumber(field(x1_y1,2,"/"))
    local x2 = tonumber(field(x2_y2,1,"/"))
    local y2 = tonumber(field(x2_y2,2,"/"))
    if (x1 - x2) == 0 then
        o_data = "90".."[deg]_"..readResult():readData()
        return (o_data)
    elseif (y1 - y2) == 0 then
        o_data = "0".."[deg]_"..readResult():readData()
        return (o_data)
    else
        o_data = (y1 - y2) / (x2 - x1) -- Obtains arctan.
        o_data = math.atan(o_data) -- Radian --> Converted to deg(°).
        o_data = math.deg(o_data) -- Truncates decimals.
        o_data = math.floor(o_data)
        o_data = o_data.."[deg]_"..readResult():readData() -- Appends read data to output.
        return (o_data)
    end
end
```

Execution result

Read Data :  
Execution result : 45[deg]_keyence 0[deg]_keyence

Angular degrees when a workpiece is read from the front are obtained.

- * If it is read from angles other than the front, correct angular degrees cannot be obtained.
- * Set the "partition mark" for SR to ":" (colon).

5-2 Obtaining angular degrees of a tilted barcode

```
function readformatEvent()
    local o_data_u
    local o_data --Output data

    if result() == 2 then --Displays the message at read
        return "READ_ERROR" error.
    end

    local data = readResult():cornerCoordinates() -- Obtains the coordinates at 4 positions of code.
    local x1_y1 = field(data,1,":") --x1/y1:x2/y2:x3/y3:x4/y4
    local x2_y2 = field(data,2,":")
    local x3_y3 = field(data,3,":")
    local x4_y4 = field(data,4,":")
    local x1 = tonumber(field(x1_y1,1,"/")) --Converts it to numerical values
    local y1 = tonumber(field(x1_y1,2,"/")) and assigns the value to the
    local x2 = tonumber(field(x2_y2,1,"/")) variable.
    local y2 = tonumber(field(x2_y2,2,"/"))
    local x3 = tonumber(field(x3_y3,1,"/"))
    local y3 = tonumber(field(x3_y3,2,"/"))
    local x4 = tonumber(field(x4_y4,1,"/"))
    local y4 = tonumber(field(x4_y4,2,"/"))

    if (x3 - x2) == 0 then
        if (y3 - y2) > 0 then
            o_data_u = 0
        else
            o_data_u = 180
        end
    elseif (y3 - y2) == 0 then
        if (x3 - x2) > 0 then
            o_data_u = -90
        else
            o_data_u = 90
        end
    else
        o_data_u = (y3 - y2) / (x3 - x2)
        o_data_u = math.atan(o_data_u)
        o_data_u = math.deg(o_data_u)
        o_data_u = math.floor(o_data_u)

        if (x3 > x2) then
            o_data_u = -90 + o_data_u
        else
            o_data_u = 90 + o_data_u
        end
    end

    o_data = o_data_u.."[deg]_"..readResult():readData() --Appends read data to output.
    return (o_data)
end
```

Execution result

Read Data :  
Execution result : 30[deg]_SR 0[deg]_SR


Angular degrees when a workpiece is read from the front are obtained.

- * If it is read from angles other than the front, correct angular degrees cannot be obtained.
- * Set the "partition mark" for SR to ":" (colon).

5-3 Bit calculations

```
function readformatEvent()
    local a = {}
    a[1] = bitAnd(0,0)
    a[2] = bitAnd(1,0)
    a[3] = bitAnd(0,1)
    a[4] = bitAnd(1,1)
    return (a[1]..a[2]..a[3]..a[4])
end
```


Execution result

Read Data : P002 
Execution result : 0001

6-1 Appending read data to the read image file name

```
function nameformatEvent()
    local read_data
    read_data = readResult():readData() -- Obtains the code reading result.
    return (read_data)
end
```

Execution result

Read data : keyence 

Execution result : keyence.bmp

* The read data is specified as the image file name. If a code with exactly the same data contents is read, the same image file name is adopted. In this case, the old file is overwritten.

6-2 Appending date and time to the read image file name

```
function nameformatEvent()
    return (time()) -- Obtains date information.
end
```

Execution result

Read data : keyence 

Execution result : 20121222120001.bmp

* Sending settings from AutoID Network Navigator to the SR-X Series synchronizes the time of the PC to that of the SR-X Series.

6-3 Appending date and time and read data to the read image file name

```
function nameformatEvent()
    local read_data
    read_data = readResult():readData() -- Obtains the code reading result.
    read_data = time().. "_" .. read_data -- Appends date information.
    return (read_data)
end
```

Execution result

Read data : keyence 

Execution result : 20121222120001_keyence.bmp

* Sending settings from AutoID Network Navigator to the SR-X Series synchronizes the time of the PC to that of the SR-X Series.

When using a script to control OUT terminals, the following settings need to be configured after setting "Enable script features".

(This is activated when "SCRIPT CONTROL" is checked only for the OUT terminals to be controlled by the script.)

Example settings: When using a script to control all OUT terminals


Reading	Bank	RS-232C	Ethernet	Operation	I/O	Saving Images	Comparison Settings	Table
Input Terminal								
		IN1 Function		IN2 Function				
Function		Timing		Disable				
Details		On:Start, Off:Stop						
Power-on trigger		Off		Off				
Input Polarity		Norm. open		Norm. open				
Required Input Duration		1ms		1ms				
Output Terminal								
		OUT1 Function		OUT2 Function		OUT3 Function		
Function		One-shot output		One-shot output		One-shot output		
Option		<input type="checkbox"/> OK <input type="checkbox"/> STABLE <input type="checkbox"/> UNSTABLE <input type="checkbox"/> ERROR <input type="checkbox"/> PRESET OK <input type="checkbox"/> TUNING OK <input checked="" type="checkbox"/> SCRIPT CONTROL		<input type="checkbox"/> OK <input type="checkbox"/> STABLE <input type="checkbox"/> UNSTABLE <input type="checkbox"/> ERROR <input type="checkbox"/> PRESET OK <input type="checkbox"/> TUNING OK <input checked="" type="checkbox"/> SCRIPT CONTROL		<input type="checkbox"/> OK <input type="checkbox"/> STABLE <input type="checkbox"/> UNSTABLE <input type="checkbox"/> ERROR <input type="checkbox"/> PRESET OK <input type="checkbox"/> TUNING OK <input checked="" type="checkbox"/> SCRIPT CONTROL		
		<input type="checkbox"/> Comparison NG <input type="checkbox"/> TRIGGER OVERRUN		<input type="checkbox"/> Comparison NG <input type="checkbox"/> TRIGGER OVERRUN		<input type="checkbox"/> Comparison NG <input type="checkbox"/> TRIGGER OVERRUN		
Output Duration		Output duration 500ms		Output duration 500ms		Output duration 500ms		
Details								
Output polarity		Norm. open		Norm. open		Norm. open		

7-1 Controlling output terminals (OK/NG/ERROR)

```
function readformatEvent()
    local r_data = result()
    local data = readResult():readData()

    if r_data == 0 then --If the result is OK, turn on OUT1.
        outonEvent(1)
    elseif r_data == 2 then --If the result is ERROR, turn on OUT2.
        outonEvent(2)
    else --In all other situations, turn on OUT1 and OUT2.
        outonEvent(1)
        outonEvent(2)
    end
    return (data)
end
```

Execution result

Read data : keyence 

Execution result : OUT1 turns on.

7-2 Preventing duplicate reading

```
local g_latestData = ""

function readformatEvent()
    local r_data = result() --Branching depending on the result (OK/ERROR)
    local readData = readResult():readData() --Read data
    if r_data == 0 then --OK
        if g_latestData == readData then
            return ""
        end
        g_latestData = readData
        outonEvent(1)
        return readData
    elseif r_data == 2 then --ERROR
        outonEvent(2)
        return "ERROR"
    end
end
```

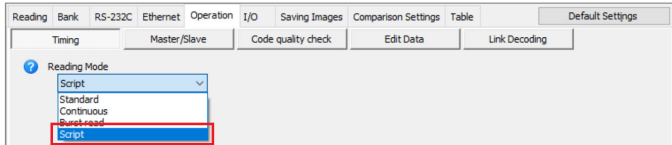
Execution result

Read data : keyence 

Execution result : keyence (The same data is not output twice.)

8 Controlling triggers with scripts

To control triggers with scripts, you have to set "Enable scripts," and then configure the settings as shown below.



8-1 Continuing reading until the character "S" is read at the start of the data

```
function readformatEvent()  
  local i  
  local read_data  
  local output_data  
  
  for i = 1, readCount() do  
    read_data = readResult(i):readData()  
    if left(read_data,1) == "S" then  
      return("S data is found!")  
    elseif result() == 0 then  
      output_data = "This data isn't S"  
      invokeNextShot() --Continue reading.  
    else  
      output_data = "Read ERROR!"  
      invokeNextShot() --Continue reading.  
    end  
  end  
  return (output_data)  
end
```

Execution result

Read data : S001  P002 
Execution result : "S data is found!" "Read ERROR!"

8-2 Detecting timing input turning off

```
function readformatEvent()  
  invokeNextShot() --Continue reading.  
  if isTriggerActive() == 0 then  
    return("Trigger off") --Detect the trigger being turned off.  
  else  
    return(scriptExecNum()) --Count the number of times the  
    script has been executed.  
  end  
end
```

Execution result

Read data : S001 
Execution result : 1
2
3
Trigger OFF

9 Confirmation commands

9-1 MAC address confirmation command

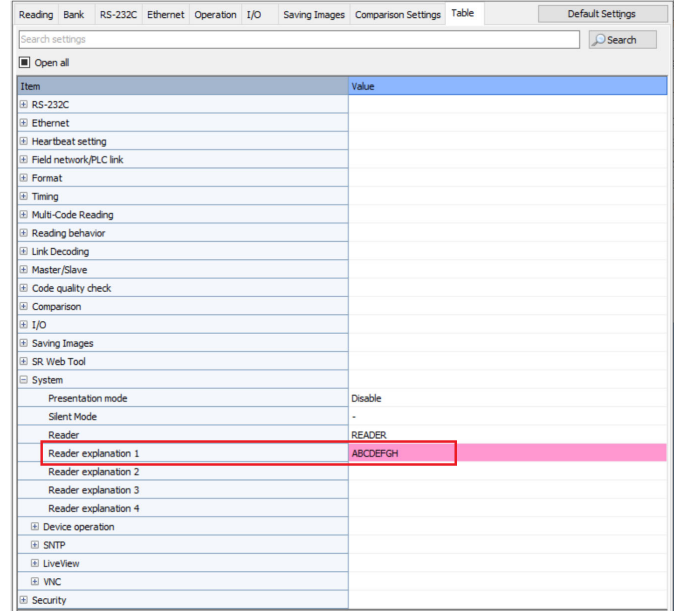
```
function readformatEvent()  
  local output_data  
  output_data = execCmd("EMAC") --Execute the command.  
  return(output_data)  
end
```

Execution result

Execution result: OK,EMAC,0001FC2D8206

9-2 Reader explanation confirmation command

You can use the SR-X Series to obtain the values set below.



```
function readformatEvent()  
  local output_data  
  output_data = execCmd("RP,620") --Execute the command.  
  return(output_data)  
end
```

Execution result

Execution result : OK,RP,ABCDEF GH

8-1 ASCII code table

			High-order 4 bits							
Hexadecimal number			0	1	2	3	4	5	6	7
	Binary number		0000	0001	0010	0011	0100	0101	0110	0111
Low-order 4 bits	0	0000	NUL 000	DLE 016	(SP) 032	0 048	@ 064	P 080	` 096	p 112
	1	0001	SOH 001	DC1 017	! 033	1 049	A 065	Q 081	a 097	q 113
	2	0010	STX 002	DC2 018	" 034	2 050	B 066	R 082	b 098	r 114
	3	0011	ETX 003	DC3 019	# 035	3 051	C 067	S 083	c 099	s 115
	4	0100	EOT 004	DC4 020	\$ 036	4 052	D 068	T 084	d 100	t 116
	5	0101	ENQ 005	NAK 021	% 037	5 053	E 069	U 085	e 101	u 117
	6	0110	ACK 006	SYN 022	& 038	6 054	F 070	V 086	f 102	v 118
	7	0111	BEL 007	ETB 023	' 039	7 055	G 071	W 087	g 103	w 119
	8	1000	BS 008	CAN 024	(040	8 056	H 072	X 088	h 104	x 120
	9	1001	HT 009	EM 025) 041	9 057	I 073	Y 089	i 105	y 121
	A	1010	LF 010	SUB 026	* 042	: 058	J 074	Z 090	j 106	z 122
	B	1011	VT 011	ESC 027	+ 043	; 059	K 075	[091	k 107	{ 123
	C	1100	FF 012	FS 028	, 044	< 060	L 076	\ 092	l 108	 124
	D	1101	CR 013	GS 029	- 045	= 061	M 077] 093	m 109	} 125
	E	1110	SOH 014	RS 030	. 046	> 062	N 078	^ 094	n 110	~ 126
	F	1111	SI 015	US 031	/ 047	? 063	O 079	_ 095	o 111	DEL 127

8-2 Reserved words/Language elements

Reserved word list

```
and    break  do      else    elseif  end    false  for     function
goto   if      in      local  nil     not    or      repeat return
then   true    until   while
```

* Do not use variables/functions starting with "_" (underscore).

Priority of operator

High	^
	not # - (unary)
	* / %
	+ -
	..
	< > <= >= ~= ==
	and
Low	or

Pattern matching

Pattern	Meaning
.	All characters (arbitrary 1 character)
%a	Character
%c	Control character (0x01 to 0x1F, 0x7F)
%d	Number
%l	Alphabet lowercase
%p	Symbol (! " # \$ % & ' () * + , - . / : ; > = < ? @ [\] ^ _ ` {)
%s	Space character (0x09,0x0a,0x0b,0x0c,0x0d,0x20)
%u	Alphabet uppercase
%w	Alphabet and number
%x	Hexadecimal number (0 to 9, a to f, A to F)
%z	null character
% character ^{*1}	Literal character

*1 Specify characters other than alphanumeric characters

8-3 Code type

Code type list used for symbolType of readResult(idx)

Code type	Return value of symbolType
Reading error	0
QR	1
DataMatrix	2
PDF417/MicroPDF	3
MaxiCode	4
GS1 DataBar	5
CODE39	6
ITF	7
2of5	8
NW-7(Codabar)	9
JAN/EAN/UPC	10
CODE128	11
COOP2of5	12
CODE93	13
CC-A/B(GS1 DataBar)	14
CC-A/B(JAN/EAN/UPC)	15
CC-A/B/C(GS-128)	16
Postal	17
Pharmacode	18
DotCode	20
Aztec Code	21

Code type list used for symbolIdentifier of readResult(idx)

Code type	Detail	Symbol ID
QR	: Model 1	JQ0
	: Model 2, ECI not applied	JQ1
	: Model 2, ECI applied	JQ2
	: Model 2, ECI not applied, FNC1 (1st)	JQ3
	: Model 2, ECI applied, FNC1 (1st)	JQ4
	: Model 2, ECI not applied, FNC1 (2nd)	JQ5
	: Model 2, ECI applied, FNC1 (2nd)	JQ6
DataMatrix	: ECC 200	Jd1
	: ECC 200, FNC1 (1st)	Jd2
	: ECC 200, FNC1 (2nd)	Jd3
	: ECC 200, ECI applied	Jd4
	: ECC 200, ECI applied, FNC1 (1st)	Jd5
	: ECC 200, ECI applied, FNC1 (2nd)	Jd6
	DMRE	Jd7
	DMRE, FNC1 (1st or 5th)	Jd8
	DMRE, FNC1 (2nd or 6th)	Jd9
	DMRE, ECI	JdA
	DMRE, ECI, FNC1 (1st or 5th)	JdB
	DMRE, ECI, FNC1 (2nd or 6th)	JdC
CODE39	No check digit validation	JA0
	Check digit is validated and transmitted.	JA1
	Check digit is validated but not transmitted.	JA3
	FullASCII, no check digit validation	JA4
	FullASCII, check digit is validated and transmitted.	JA5
	FullASCII, check digit is validated but not transmitted.	JA6
ITF	No check digit validation	Jl0
	Check digit is validated and transmitted.	Jl1
	Check digit is validated but not transmitted.	Jl3
NW-7(Codabar)		JF0
JAN/EAN/UPC	UPC-A, UPC-E, JAN/EAN13	JE0
	JAN/EAN8	JE4
	UPC-A, UPC-E, JAN/EAN13 Addon 2, addon 5	JE3
CODE128	FNC1 not included.	JC0
	FNC1 on the first digit (GS1-128).	JC1
	FNC1 on the second digit.	JC2
GS1 Databar		Je0
PDF417, MicroPDF417	Standard	JL0
	Extended channel interpretation	JL1
	Basic channel interpretation	JL2
CODE93		JG0
2of5		JS0
COOP2of5		JX0
Trioptic CODE39		JA8
Postal	JAPAN POST	JX1
	Intelligent Mail	JX8
DotCode	generic data	JJ0
	GS1 format (GS1 DotCode)	JJ1
	Application Specific	JJ2
	generic data + ECI	JJ3
	GS1 format (GS1 DotCode) + ECI	JJ4
	Application Specific + ECI	JJ5
Aztec Code		Jz0
	FNC1(1st)	Jz1
	FNC1(An initial letter or pair of digits)	Jz2
	ECI	Jz3
	FNC1(1st) + ECI	Jz4
	FNC1(An initial letter or pair of digits) + ECI	Jz5
	Aztec Rune	JzC


8-4 Troubleshooting

The script does not operate.


If codes can be read normally but the script does not operate, the script execution setting of the SR-X Series may not be set to "Enable". Using the AutoID Network Navigator, set the script execution setting to "Enable".

* For using the AutoID Network Navigator, refer to the user's manual.

Expected results cannot be achieved.

Debug according to  "6-1 Debug methods" (Page 13)

An error occurs when executed.

See the applicable error message on  "6-2 Error message list" (Page 14), and revise the program.

Barcodes/2D codes cannot be read.

Reading setting may not be correctly made on your code reader. Using the AutoID Network Navigator, make the reading setting for the SR-X Series.

* For using the AutoID Network Navigator, refer to the user's manual.

8-5 Copyright indication

This software uses the following libraries:

Lua:

Copyright (c) 1994-2021 Lua.org, PUC-Rio.

Revision History

Date of printing	Version	Revision contents
January 2022	First Edition	
June 2023	Second Edition	

WARRANTIES AND DISCLAIMERS

- (1) KEYENCE warrants the Products to be free of defects in materials and workmanship for a period of one (1) year from the date of shipment. If any models or samples were shown to Buyer, such models or samples were used merely to illustrate the general type and quality of the Products and not to represent that the Products would necessarily conform to said models or samples. Any Products found to be defective must be shipped to KEYENCE with all shipping costs paid by Buyer or offered to KEYENCE for inspection and examination. Upon examination by KEYENCE, KEYENCE, at its sole option, will refund the purchase price of, or repair or replace at no charge any Products found to be defective. This warranty does not apply to any defects resulting from any action of Buyer, including but not limited to improper installation, improper interfacing, improper repair, unauthorized modification, misapplication and mishandling, such as exposure to excessive current, heat, coldness, moisture, vibration or outdoors air. Components which wear are not warranted.
- (2) KEYENCE is pleased to offer suggestions on the use of its various Products. They are only suggestions, and it is Buyer's responsibility to ascertain the fitness of the Products for Buyer's intended use. KEYENCE will not be responsible for any damages that may result from the use of the Products.
- (3) The Products and any samples ("Products/Samples") supplied to Buyer are not to be used internally in humans, for human transportation, as safety devices or fail-safe systems, unless their written specifications state otherwise. Should any Products/Samples be used in such a manner or misused in any way, KEYENCE assumes no responsibility, and additionally Buyer will indemnify KEYENCE and hold KEYENCE harmless from any liability or damage whatsoever arising out of any misuse of the Products/Samples.
- (4) **OTHER THAN AS STATED HEREIN, THE PRODUCTS/SAMPLES ARE PROVIDED WITH NO OTHER WARRANTIES WHATSOEVER. ALL EXPRESS, IMPLIED, AND STATUTORY WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY RIGHTS, ARE EXPRESSLY DISCLAIMED.**
IN NO EVENT SHALL KEYENCE AND ITS AFFILIATED ENTITIES BE LIABLE TO ANY PERSON OR ENTITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, PUNITIVE, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING, WITHOUT LIMITATION, ANY DAMAGES RESULTING FROM LOSS OF USE, BUSINESS INTERRUPTION, LOSS OF INFORMATION, LOSS OR INACCURACY OF DATA, LOSS OF PROFITS, LOSS OF SAVINGS, THE COST OF PROCUREMENT OF SUBSTITUTED GOODS, SERVICES OR TECHNOLOGIES, OR FOR ANY MATTER ARISING OUT OF OR IN CONNECTION WITH THE USE OR INABILITY TO USE THE PRODUCTS, EVEN IF KEYENCE OR ONE OF ITS AFFILIATED ENTITIES WAS ADVISED OF A POSSIBLE THIRD PARTY'S CLAIM FOR DAMAGES OR ANY OTHER CLAIM AGAINST BUYER. In some jurisdictions, some of the foregoing warranty disclaimers or damage limitations may not apply.

BUYER'S TRANSFER OBLIGATIONS:

If the Products/Samples purchased by Buyer are to be resold or delivered to a third party, Buyer must provide such third party with a copy of this document, all specifications, manuals, catalogs, leaflets and written information provided to Buyer pertaining to the Products/Samples.

E 1101-3

BarcodeReader.com

www.barcodereader.com

You can download technical documents useful for BL/SR introduction and operation.

KEYENCE CORPORATION

1-3-14, Higashi-Nakajima, Higashi-Yodogawa-ku,
Osaka, 533-8555, Japan
PHONE: +81-6-6379-2211

www.keyence.com/glb

AUSTRIA Ph: +43 (0)2236 378266 0	HONG KONG Ph: +852-3104-1010	NETHERLANDS Ph: +31 (0)40 206 6100	TAIWAN Ph: +886-2-2721-1080
BELGIUM Ph: +32 (0)15 281 222	HUNGARY Ph: +36 1 802 7360	PHILIPPINES Ph: +63-(0)2-8981-5000	THAILAND Ph: +66-2-078-1090
BRAZIL Ph: +55-11-3045-4011	INDIA Ph: +91-44-4963-0900	POLAND Ph: +48 71 368 61 60	UK & IRELAND Ph: +44 (0)1908-696-900
CANADA Ph: +1-905-366-7655	INDONESIA Ph: +62-21-2966-0120	ROMANIA Ph: +40 (0)269 232 808	USA Ph: +1-201-930-0100
CHINA Ph: +86-21-3357-1001	ITALY Ph: +39-02-6688220	SINGAPORE Ph: +65-6392-1011	VIETNAM Ph: +84-24-3772-5555
CZECH REPUBLIC Ph: +420 220 184 700	KOREA Ph: +82-31-789-4300	SLOVAKIA Ph: +421 (0)2 5939 6461	
FRANCE Ph: +33 1 56 37 78 00	MALAYSIA Ph: +60-3-7883-2211	SLOVENIA Ph: +386 (0)1 4701 666	
GERMANY Ph: +49-6102-3656-0	MEXICO Ph: +52-55-8850-0100	SWITZERLAND Ph: +41 (0)43 455 77 30	

Specifications are subject to change without notice.

A6WW1-MAN-2033

Copyright (c) 2022 KEYENCE CORPORATION. All rights reserved.
193308GB 2053-2 [C88GB] Printed in Japan

