Documentation of the library 
WagoAppWString 
*Release 1.0.1.1*
# Contents

## 1 Description

## 2 20 Program Organization Units

### 2.1 01 Conversions and Formatting

- Formatted Printout (snprintf-like) .......................... 3
  - WFormat1 (FUN) .................................................. 3
  - WFormat2 (FUN) .................................................. 7
  - WFormat3 (FUN) .................................................. 8
  - WFormat4 (FUN) .................................................. 8
  - WFormat5 (FUN) .................................................. 9
  - WFormat6 (FUN) .................................................. 9
  - WFormat7 (FUN) .................................................. 10
  - WFormat8 (FUN) .................................................. 11
  - WFormat9 (FUN) .................................................. 12
- WTrim (FUN) ......................................................... 12

### 2.2 02 Investigating Strings

- WFindLeft (FUN) .................................................... 13
- WFindRight (FUN) ................................................... 13
- WLength (FUN) ....................................................... 14

### 2.3 05 Non-manipulative In-place Operations

- WStrFindChar (FUN) ............................................... 14
- WStrFindLastChar (FUN) .......................................... 15
- WStrFindLeft (FUN) ............................................... 16
- WStrFindRight (FUN) .............................................. 17
- WStrLength (FUN) ................................................... 18

### 2.4 10 C99-Style

- wsnprintf (FUN) .................................................... 18

## 3 VersionHistory (GVL)

## 4 Library Reference
**Description**

This document is automatically generated. Because of this, the chapter 30 Visualization is not shown in this document. If you are interested in getting to know more about visualization, we refer to the library manager of e!Cockpit.

**Subject to Changes**

WAGO Kontakttechnik GmbH Co. KG reserves the right to provide for any alterations or modifications. WAGO Kontakttechnik GmbH Co. KG owns all rights arising from the granting of patents or from the legal protection of utility patents. Third-party products are always mentioned without any reference to patent rights. Thus, the existence of such rights cannot be excluded.

**Personnel Qualification**

All tasks that are carried out with libraries made for the e!COCKPIT software must only be performed by qualified electrical specialists instructed in PLC programming according to IEC 61131-3.

All tasks that have an effect on the properties or the behavior of automation hardware or software products must only be performed by qualified employees with a thorough knowledge of handling the products concerned.

**Intended Use of e!COCKPIT Libraries**

Libraries created for the e!COCKPIT software are used to simplify the development of application projects in the IEC 61131-3 programming languages.

For automation tasks, WAGO offers programmable logic controllers in a wide variety of performance classes. In combination with a wide range of I/O modules, the controllers can process standard types of field signals. Controllers can be implemented centrally or in decentralized configurations. The controllers offer interfaces for the most commonly used fieldbuses for use in decentralized configurations. Fieldbus independent I/O modules are then linked via fieldbus couplers. WAGO controllers offer a runtime environment for user programs called e!RUNTIME. Software projects for implementation in e!RUNTIME environments can be created in e!COCKPIT. The programming environment in e!COCKPIT is based on the established CODESYS 3 industrial standard. Users with a previous knowledge of CODESYS 3 will thus find this environment largely familiar. The following programming languages of the IEC 61131-3 standard are available:

- Structured Text (ST)
- Ladder Diagram (LD)
- Function Block Diagram (FBD)
- Instruction List (IL)
- Sequential Function Chart (SFC)
- Continuous Function Chart (CFC)

The individual programming languages can also be combined as required during the development of the software. A portfolio of prepared libraries can be accessed for many frequently used functions in order to make software development more efficient. This document provides an overview of the WagoAppWString that WAGO offers for e!COCKPIT.
Manipulation, Formatting, and Analysis of Character WStrings.  

Further library information are summarized here:

Company  WAGO
Title  WagoAppWString
Version  1.0.1.1
Categories  Application; WAGO LayerViewApp; WAGO FunctionalViewBase
Namespace  WagoAppWString
Author  WAGO / u013972
Placeholder  WagoAppWString
2.1 01 Conversions and Formatting

2.1.1 Formatted Printout (snprintf-like)

WFormat1 (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFormat1</td>
<td>WSTRING(255)</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>wsFormat</td>
<td>WSTRING(255)</td>
<td>control wstring</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>WagoTypes.vPointer</td>
<td>points to the variable</td>
</tr>
</tbody>
</table>

Function

Gives a formatted output of variables to a wstring.

Graphical Illustration

![Graphical Illustration](image)

Function Description

This function produces a formatted wstring serialization from a CODESYS variable. The general behaviour is similar to the well-established snprintf-function from C99 / Posix. While the substantial content of the result is given by that variable, the appearance of the result is controlled by a control wstring, e.g.

\[
\text{sResult := Format1( "Error-%03i", \, \text{adr}(i) ); // sResult = "Error-007" for i=7.}
\]

The construction of the control wstring is described in detail below.

Context:

Similar to the C-snprintf(), this CODESYS-formatting tool also allows for passing 2 or more variable arguments. For that purpose, a set of similar functions is composed with function names Format2()...Format9(). Their functionality is virtually the same as Format1(). But their interfaces, however, hold more variables, according to the scheme below:
<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Type</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>sFormat</td>
<td>IN</td>
<td>WSTRING</td>
<td>Control string for formatting the output</td>
</tr>
<tr>
<td>p1</td>
<td>IN</td>
<td>POINTER</td>
<td>Address of the first value to be formatted</td>
</tr>
<tr>
<td>p2</td>
<td>IN</td>
<td>POINTER</td>
<td>Address of the 2nd value to be formatted (Format2..9()) only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p9</td>
<td>IN Return</td>
<td>POINTER</td>
<td>Address of the 9th value to be formatted (Format9() only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WSTRING(255)</td>
<td>Formatted String</td>
</tr>
</tbody>
</table>

For the sake of universality there is also a variant defined with a variable number of arguments (named ‘wsnprintf()’), but that universality results in a somewhat unhandy interface. So we recommend using Format1()...Format9() instead where possible.

**Usage:** (for explanation see below)

```plaintext
sDisplayA := Format1("Now it is: %h:%m%zt o$'Clock'", adr(typTheTime));
sDisplayB := Format2("Step %02i: X = %5.2f mm", adr(iStepNo), adr(rXcoord));
```

The first input of a FormatX()-function is a control wstring, where placeholders are embedded, just as known from the printf(3)-function in C99. Those placeholders are identified by a ‘%’ character. Also escape-sequences could be embedded, which are identified by a backslash (\). In the resulting output string of the FormatX()-function, all placeholders and escape sequences are replaced by their evaluated contents.

The second (and following) inputs are pointers to the variables, which are to be displayed. Here, content of any type can be passed to the FormatX()-function, while the real type and the way to represent it in the output is controlled by the control wstring.

**Placeholders in the control wstring**

Placeholders start with a ‘%’ sign and end with a conversion specifier. The notation is closely adopted from the C99-syntax of printf() e.g.: ‘%i’. The conversion specifier controls the general style of the output wstring.

<table>
<thead>
<tr>
<th>Conversion Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>signed integer “%i”, “%li”, ...</td>
</tr>
<tr>
<td>x</td>
<td>hexadecimal representation</td>
</tr>
<tr>
<td>f</td>
<td>floating point number</td>
</tr>
<tr>
<td>c</td>
<td>single character</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
</tr>
<tr>
<td>W</td>
<td>wstring (W -&gt; capital letter)</td>
</tr>
<tr>
<td>E</td>
<td>enumeration</td>
</tr>
<tr>
<td>t</td>
<td>time</td>
</tr>
<tr>
<td>p</td>
<td>pointer</td>
</tr>
</tbody>
</table>

**Note:** The common “u” conversion specifier is used in a different way here. Here it is used as a flag for denoting ‘unsigned’ values in contrast to signed ones. This is done to achieve good correspondence to CODESYS conventions.

Between the “%” and the terminating conversion specifying character, a number of other characters may be placed, which allows for more options according to the scheme: “%[flags][Type]<Conversion Spec.>“. E.g. “%+5.2lf” (see below)

**Type Specifier**

A Type Specifier may be inserted directly before the Conversion Specifier:
The major function of the TypeSpecifier is to carry the size of the displayed variable. (This is necessary, because type information of the variable is lost when being passed via a pointer.)

If no type specifier is given, the standard variable types according to the conversion specifier are assumed, i.e. INT, STRING, REAL, etc.

Note: The Type Specifiers differ significantly from the C99-Spec due to the special circumstances of CODESYS.

Flags

The TypeSpecifier may be preceded by several flags. Flags give more detailed control to the appearance of the output.

Flags may be used cumulatively, e.g. “%+6.2f”. They may be also be completely omitted at all, if not needed.

Flags

<table>
<thead>
<tr>
<th>Flags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>length of a field, left-padded with blanks. [&quot;%2i&quot;, 3] -&gt; “3”</td>
</tr>
<tr>
<td>0</td>
<td>the value is left-padded with zeroes. [&quot;%02i:%02i&quot;, 12, 5] -&gt; “12:05”</td>
</tr>
<tr>
<td>-</td>
<td>the value is right-padded instead of left-padded (overrides ‘0’)</td>
</tr>
<tr>
<td>+</td>
<td>a positive number carries a “+” prefix</td>
</tr>
<tr>
<td>&lt;spc&gt;</td>
<td>a positive number is preceded by a “ “.</td>
</tr>
<tr>
<td>.1-9</td>
<td>precision specification for floating point variables (“%6.2f”)</td>
</tr>
<tr>
<td>.1-9</td>
<td>fixed-point specification for integers. [“%.2i”, 1234] -&gt; “12.34”</td>
</tr>
<tr>
<td>u</td>
<td>unsigned values “%ui”, “%udi”, “%uli”</td>
</tr>
<tr>
<td>u</td>
<td>UT reference instead of local time “%ut”, “%ult”</td>
</tr>
<tr>
<td>{...}</td>
<td>encapsulate other options, e.h. for time conversions, “%{%h:%m}zt”</td>
</tr>
<tr>
<td>#</td>
<td>alternate form, if applicable</td>
</tr>
</tbody>
</table>

Alternate Forms

With the ‘#’ flag, an alternate form of output formatting may be produced in some cases:

Alternate Forms

<table>
<thead>
<tr>
<th>Alternate Forms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%#x</td>
<td>Hexadecimal values will be preceded by “0x”:</td>
</tr>
<tr>
<td>%#t</td>
<td>If no ‘[…]’ format info given, a date format is used instead of time format.</td>
</tr>
<tr>
<td>%#c</td>
<td>if the character is non-printable, it is replaced by its ordinal, e.g. “[10]” for ‘Newline’</td>
</tr>
<tr>
<td>%#f</td>
<td>(not implemented, reserved)</td>
</tr>
<tr>
<td>other</td>
<td>Reserved for future use, undefined result at this stage of specification</td>
</tr>
</tbody>
</table>

Escape Sequences

Escape sequences are required for inline coding of special or non-printable characters, such as new-line or the percent sign.
Escape Sequences

<table>
<thead>
<tr>
<th>Seq.</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>0x0a</td>
<td>Line Feed</td>
</tr>
<tr>
<td>\r</td>
<td>0x0d</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>\t</td>
<td>0x09</td>
<td>Tab</td>
</tr>
<tr>
<td>\f</td>
<td>0x0c</td>
<td>Form Feed</td>
</tr>
<tr>
<td>\v</td>
<td>0x0d+0x0a</td>
<td>CRLF</td>
</tr>
<tr>
<td>\xnn</td>
<td>0xnn</td>
<td>Hexvalue nn</td>
</tr>
<tr>
<td>\Unnn</td>
<td>....</td>
<td>Unicode representation of 0xnnnn</td>
</tr>
<tr>
<td>\</td>
<td>0x5c</td>
<td>Backslash</td>
</tr>
<tr>
<td>?</td>
<td>0x3f</td>
<td>Question mark</td>
</tr>
<tr>
<td>$</td>
<td>0x24</td>
<td>Dollar sign (= $$, $ is reserved by CODESYS)</td>
</tr>
<tr>
<td>q</td>
<td>0x27</td>
<td>Single Quote (= $' , ' is reserved by CODESYS)</td>
</tr>
<tr>
<td>Q</td>
<td>0x22</td>
<td>Double Quote (= '' is reserved by CODESYS)</td>
</tr>
<tr>
<td>%</td>
<td>0x25</td>
<td>Percent sign</td>
</tr>
</tbody>
</table>

Encapsulated format sequence

Some conversion specifiers use a non-standard encapsulated format sequence (within curly braces), because the standard options yield too less information. This is namely:

- **Time**: (%{...}t) The encapsulated format is forwarded to the function TimeComponents_To_String() which is located at another place in this library. See there for reference.
- **Enumeration**: (%{...}e) the encapsulated format carries information about how the enumeration members are to be serialized.

Enumerations

Serialized representation of enumerations is a proprietary extension to the C99-style. The numerical value of the enumeration is converted into a string which is predefined in the control string.

The conversion specifier for this case is “e” (attn: “e” is not used for floating-points here, as it would be in C99).

By default, a 16-Bit variable is assumed, which matches the CODESYS enumeration default ‘INT’. The other regular type specifiers (b,w,d,l) are also valid - but not ‘z’, which is a special type for time formats.

The string representation is encoded in an encapsulated format sequence. A typical encapsulated format for enumerations might look like this:

```
%{standard,firstitem,seconditem,4:nextitem,*:other}e
```

The representations of the enumeration items in the format are separated by commas (",").

The numeration of the items starts with 0 and counts up. If an item starts with a number followed by a colon (":") this number identifies the enumeration value.

Attention: Although the CODESYS-Default suggests signed variables for enumeration, this snprintf()-implementation uses only unsigned enums, because the negative part is very rarely used. If an enum is indeed defined as INT:=-1, it has to be declared as 65535:...

If an item starts with an asterisk followed by a colon ("*:"), this item is taken as default if the previous items did not match. Attn: This item must be the last in the sequence.

Example: In the above example, the following enumeration values would result in the following result strings:
Example: \%{standard,firstitem,seconditem,4:nextitem,*:other}\e

<table>
<thead>
<tr>
<th>Enum-Value</th>
<th>Result</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>standard</td>
<td>1st substring</td>
</tr>
<tr>
<td>1</td>
<td>firstitem</td>
<td>2nd substring</td>
</tr>
<tr>
<td>2</td>
<td>seconditem</td>
<td>3rd substring</td>
</tr>
<tr>
<td>3</td>
<td>other</td>
<td>caught by default entry '*: '</td>
</tr>
<tr>
<td>4</td>
<td>nextitem</td>
<td>explicite value</td>
</tr>
<tr>
<td>(other)</td>
<td>other</td>
<td>caught by default entry '*: '</td>
</tr>
</tbody>
</table>

Other Separators: If the first character in the format string is not a number and not an alphabetic letter, this character is taken as separator (e.g. a pipe symbol ('|') or a space or a semicolon. E.g.: \%{|standard,default|firstitem,info|seconditem,warning|4:nextitem,error|*: --other}\e

would cause a value 0 to result to the string standard, default, 1 to firstitem, info, etc.

Missing Default Value: If the variable holds a value which is not covered by any item in the encapsulated format string, it is printed out as "[ <value> ]" instead in order to get debugging information. If debugging information is unwanted, a default value ("*: ") has to be put in the format string.

Caveats

There are some behaviours of the format process, which might be unexpected:

- Padding with zeroes ("0") requires a leading digit (i.e. "0".."9" or "a".."f"). When minus or plus sign is already added, padding uses always space (" "). E.g.: "34" will be padded to "0034", but "-34" will be padded to " -34".
- The displayed data is that, which the pointer "p1" points to - with one exception: when the conversion specifier species "%p", the pointer "p1" itself is the data to be displayed.
- When a null-pointer is given as data pointer, a default value is inserted into the target wstring, rather than failing with an exception. (0 for numbers, empty strings, etc). Passing null-pointers at this place should be considered as programming error (at compile time), but once the program is running, there is no point in not behaving gracefully at fail.

WFormat2 (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFormat2</td>
<td>WSTRING(255)</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>wsFormat</td>
<td>WSTRING(255)</td>
<td>control wstring</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>WagoTypes.vPointer</td>
<td>points to the variable</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
</tbody>
</table>

Function

Gives a formatted output of variables to a wstring.

Graphical Illustration

2.1. 01 Conversions and Formatting
Function Description
This function takes exactly 2 variables. A full description is given in WFormat1().

WFormat3 (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFormat3</td>
<td>WSTRING(255)</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>wsFormat</td>
<td>WSTRING(255)</td>
<td>control wstring</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>WagoTypes.vPointer</td>
<td>points to the variable</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
</tbody>
</table>

Function
Gives a formatted output of variables to a wstring.

Graphical Illustration

Function Description
This function takes exactly 3 variables. A full description is given in WFormat1().

WFormat4 (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFormat4</td>
<td>WSTRING(255)</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>wsFormat</td>
<td>WSTRING(255)</td>
<td>control wstring</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>WagoTypes.vPointer</td>
<td>points to the first variable</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
</tbody>
</table>

Function
Gives a formatted output of variables to a wstring.

Graphical Illustration
Function Description
This function takes exactly 4 variables. A full description is given in \textit{WFormat1()}.

\textbf{WFormat5 (FUN)}

\textbf{Interface variables}

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFormat5</td>
<td>WSTRING(255)</td>
<td>control wstring</td>
</tr>
<tr>
<td>Input</td>
<td>wsFormat</td>
<td>WSTRING(255)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>WagoTypes.vPointer</td>
<td>points to the first variable</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
</tbody>
</table>

\textbf{Function}

Gives a formatted output of variables to a wstring.

\textbf{Graphical Illustration}

Function Description
This function takes exactly 5 variables. A full description is given in \textit{WFormat1()}. 

\textbf{WFormat6 (FUN)}

\textbf{Interface variables}
Function
Gives a formatted output of variables to a wstring.

Graphical Illustration

Function Description
This function takes exactly 6 variables. A full description is given in \textit{WFormat1}().

\textbf{WFormat7 (FUN)}

Interface variables

Function
Gives a formatted output of variables to a wstring.

Graphical Illustration
Function Description

This function takes exactly 7 variables. A full description is given in \textit{WFormat1()}.  

\textbf{WFormat8 (FUN)}

\textbf{Interface variables}

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFormat8</td>
<td>WSTRING(255)</td>
<td>control wstring</td>
</tr>
<tr>
<td></td>
<td>wsFormat</td>
<td>WSTRING(255)</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>p1</td>
<td>WagoTypes.vPointer</td>
<td>points to the first variable</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p6</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
</tbody>
</table>

\textbf{Function}

Gives a formatted output of variables to a wstring.

\textbf{Graphical Illustration}

Function Description

This function takes exactly 8 variables. A full description is given in \textit{WFormat1()}.  

2.1. 01 Conversions and Formatting
WFormat9 (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFormat9</td>
<td>WSTRING(255)</td>
<td>control wstring</td>
</tr>
<tr>
<td>Input</td>
<td>p1</td>
<td>WagoTypes.vPointer</td>
<td>points to the first variable</td>
</tr>
<tr>
<td>Input</td>
<td>p2</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td>Input</td>
<td>p3</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td>Input</td>
<td>p4</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td>Input</td>
<td>p5</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td>Input</td>
<td>p6</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td>Input</td>
<td>p7</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td>Input</td>
<td>p8</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
<tr>
<td>Input</td>
<td>p9</td>
<td>WagoTypes.vPointer</td>
<td>points to a further variable</td>
</tr>
</tbody>
</table>

Function

Gives a formatted output of variables to a wstring.

Graphical Illustration

![Function Diagram](image)

Function Description

This function takes exactly 9 variables. A full description is given in WFormat1().

2.1.2 WTrim (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WTrim</td>
<td>WSTRING(255)</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>wsTr</td>
<td>WSTRING(255)</td>
<td>the wstring which is to be trimmed</td>
</tr>
</tbody>
</table>

Function

Removes leading and trailing spaces and control characters from a wstring.

Graphical Illustration

![Function Diagram](image)
Function Description

Example:

\[
\text{WTrim(" Zeile aus Datei\$r\$n") = "Zeile aus Datei"}
\]

## 2.2 02 Investigating Strings

### 2.2.1 WFindLeft (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFindLeft</td>
<td>UDINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>wsBuffer</td>
<td>WSTRING(255)</td>
<td>the string which is to be investigated</td>
</tr>
<tr>
<td></td>
<td>wsProbe</td>
<td>WSTRING(255)</td>
<td>the substring which is to be searched</td>
</tr>
<tr>
<td></td>
<td>udiBegin</td>
<td>UDINT</td>
<td>first position of the search (first position = 1)</td>
</tr>
</tbody>
</table>

**Function**

Finds the first occurrence of a subwstring in a larger wstring and returns its position.

**Graphical Illustration**

![Function Diagram](FunctionDiagram.png)

**Function Description**

The position at the begin of the wstring is denoted by index 1, as is the convention in a CODESYS environment. If the subwstring is not found, 0 is returned.

In contrast to the standard FIND function, this one is able to consecutively find multiple occurrences of a subwstring in a wstring. This is done by starting the search process not at the left but at a given position behind the result of the previous search.

If 0 is given for \textit{udiBegin}, the search starts at the beginning of the string.

### 2.2.2 WFindRight (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WFindRight</td>
<td>UDINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>wsBuffer</td>
<td>WSTRING(255)</td>
<td>the string which is to be investigated</td>
</tr>
<tr>
<td></td>
<td>wsProbe</td>
<td>WSTRING(255)</td>
<td>the substring which is to be searched</td>
</tr>
<tr>
<td></td>
<td>udiBegin</td>
<td>UDINT</td>
<td>first position of the search (first position = 1, rightmost position =0)</td>
</tr>
</tbody>
</table>

**Function**

Finds the rightmost occurrence of a subwstring in a larger wstring and returns its position.
Graphical Illustration

```
Function
WFindRight
wsBuffer  STRING  UDINT  WFindRight
wsProbe   STRING
udiBegin  UDINT
```

Function Description

The position at the begin of the wstring is denoted by index 1, as is the convention in a CODESYS environment. If the subwstring is not found, 0 is returned.

In contrast to the standard FIND function, this one searches in reverse order and it is able to consecutively find multiple occurrences of a subwstring in a wstring by starting the search process not at the right but at a given position before the result of the previous search.

If 0 is given for udiBegin, the search starts at right end of the string.

2.2.3 WLength (FUN)

Interface variables

```
Scope | Name | Type       | Comment
------|------|------------|--------
Return| WLength | UDINT     |        
Input | wsTr | WSTRING(255) | the string which is to be investigated
```

Function

Returns an unsigned (natively non-negative) length of a wstring.

Graphical Illustration

```
Function
WLength
wsTr  STRING  UDINT  WLength
```

Function Description

This does effectively the same as the native CODESYS function, but returns a more appropriate type.

Attention: This function is restricted to wstring lengths of 255 or less, because it accepts a WSTRING(255) as input. For use with unrestricted strings, please use StrLength instead.

2.3 05 Non-manipulative In-place Operations

These functions use in-place mechanisms and may operate on buffer sizes larger than 255 bytes, although they do not alter the buffer. They just analyze the buffer.

2.3.1 WStrFindChar (FUN)

Interface variables
WagoAppWString, Release 1.0.1.1

Function

Finds a character in a larger wstring.

Graphical Illustration

```
Function

wsBuffer   WSTRING   UDINT
udiSize    UDINT
wProbe     WORD
udiBegin   UDINT

WStrFindChar
```

Function Description

The first parameter is the ‘buffer’-wstring which is to be investigated. For technical reasons, the IN_OUT-parameter is typed as WSTRING(1), but in fact you may pass wstrings of any size here.

The second parameter must hold the actual size of that wstring variable or zero if the input wstring is guaranteed to be zero-terminated.

The index of the first occurrence of a character in the wstring is returned. If no occurrence is found, 0 is returned. The parameter udiBegin denotes the start position of the search. This parameter allows for consecutively finding multiple occurrences of a character.

A search for 0x00 returns the length of the wstring if a terminator could be found, or zero if no terminator could be found.

Example:

```
wBuffer := "Hello World!";
udiPos := WStrFindChar(wsBuffer, SizeOf(sBuffer), "o", 1); // udiPos=5
udiPos := WStrFindChar(wsBuffer, SizeOf(sBuffer), "o", 6); // udiPos=8
```

2.3.2 WStrFindLastChar (FUN)

Interface variables

```
Scope | Name         | Type    | Comment
-----|--------------|---------|---------
Return| WStrFindLastChar | UDINT   |         
Inout | wsBuffer     | WSTRING(1) | the buffer string which is to be analyzed 
Input | udiSize      | UDINT | The size of buffer string. Use SizeOf(...) 
wProbe | WORD | The char which is to be searched 
udiBegin | UDINT | Start position of the search (1=first position, 0=rightmost) 
```

Function

Finds a character in a larger wstring (starts from the end).

Graphical Illustration
Function Description

The first parameter is the ‘buffer’-wstring which is to be investigated. For technical reasons, the IN_OUT-parameter is typed as WSTRING(1), but in fact you may pass wstrings of any size here.

The second parameter must hold the actual size of that wstring variable or zero if the input wstring is guaranteed to be zero-terminated.

The index of the last occurrence of the character is returned. If no occurrence is found, 0 is returned.

The parameter udiBegin denotes the begin of the search. This parameter allows for consecutively finding multiple occurrences of a character.

This function is quite similar to StrFindChar, except for the search direction, which is from right to left in this case. When 0 is passed as parameter udiBegin, the search starts at the end of the wstring.

Example:

wsBuffer := "Hello World!";
udiPos := WStrFindLastChar(wsBuffer, SizeOf(wsBuffer), "o", 0); // udiPos=8
udiPos := WStrFindLastChar(wsBuffer, SizeOf(wsBuffer), "o", 7); // udiPos=5

Note: When 0 is passed as char-to-be-searched, the length of the wstring is returned, if udiBegin is 0 or greater than the length of the probe wstring (outside of the wstring). If udiBegin points to somewhere inside the wstring instead, 0 is returned, because we do not have any zeroes inside a regular wstring.

2.3.3 WStrFindLeft (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WStrFindLeft</td>
<td>UDINT</td>
<td>the buffer-wstring which is to be analyzed</td>
</tr>
<tr>
<td></td>
<td>wsBuffer</td>
<td>WSTRING(1)</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>udiSize</td>
<td>UDINT</td>
<td>The size of buffer-wstring. Use SizeOf(...)</td>
</tr>
<tr>
<td></td>
<td>wsProbe</td>
<td>WSTRING(255)</td>
<td>The subwstring which is to be found</td>
</tr>
<tr>
<td></td>
<td>udiBegin</td>
<td>UDINT</td>
<td>Start of the search (1=first position)</td>
</tr>
</tbody>
</table>

Function

Finds a subwstring in a larger wstring.

Graphical Illustration
Function Description

The first parameter is the ‘buffer’-wstring which is to be investigated. For technical reasons, the IN_OUT-parameter is typed as WString(1), but in fact you may pass wstrings of any size here.

The second parameter must hold the actual size of that wstring variable or zero if the input wstring is guaranteed to be zero-terminated.

The index of the first occurrence of a subwstring in a wstring is returned. If no occurrence is found, 0 is returned.

The parameter udiBegin denotes the begin of the search. This parameter allows for consecutively finding multiple occurrences of a subwstring.

Example:

```pascal
wsBuffer := "the quick brown fox jumped over the lazy old dog."
udiPos := WStrFindLeft(wsBuffer, SizeOf(wsBuffer), "the", 1); // udiPos = 1
udiPos := WStrFindLeft(wsBuffer, SizeOf(wsBuffer), "the", 10); // udiPos = 33
udiPos := WStrFindLeft(wsBuffer, SizeOf(wsBuffer), "the", 34); // udiPos = 0
```

2.3.4 WStrFindRight (FUN)

### Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WStrFindRight</td>
<td>UDINT</td>
<td></td>
</tr>
<tr>
<td>Inout</td>
<td>wsBuffer</td>
<td>WSTRING(1)</td>
<td>the buffer-wstring which is to be analyzed</td>
</tr>
<tr>
<td>Input</td>
<td>udiSize</td>
<td>UDINT</td>
<td>The size of buffer-wstring. Use SizeOf(...)</td>
</tr>
<tr>
<td></td>
<td>wsProbe</td>
<td>WSTRING(255)</td>
<td>the subwstring which is to be found</td>
</tr>
<tr>
<td></td>
<td>udiBegin</td>
<td>UDINT</td>
<td>Start of the search(1=leftmost, 0=rightmost)</td>
</tr>
</tbody>
</table>

Function

Finds a wsubstring in a larger wstring.

Graphical Illustration

```
Function
WStrFindRight

wsBuffer WSTRING UDIINT WStrFindRight
udiSize UDIINT
wsProbe WSTRING
udiBegin UDIINT
```

Function Description

The first parameter is the ‘buffer’-wstring which is to be investigated. For technical reasons, the IN_OUT-parameter is typed as WSTRING(1), but in fact you may pass wstrings of any size here.

The second parameter must hold the actual size of that wstring variable or zero if the input wstring is guaranteed to be zero-terminated.

The index of the last occurrence of the subwstring is returned, (leftmost = 1). If no occurrence is found, 0 is returned.

The parameter udiBegin denotes the begin of the search. This parameter allows for consecutively finding multiple occurrences of a subwstring.

This function is quite similar to WStrFindLeft, except for the search direction, which is from right to left in this case. When 0 is passed as parameter udiBegin, the search starts at the end of the wstring.

2.3. 05 Non-manipulative In-place Operations
2.3.5 WStrLength (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WStrLength</td>
<td>UDINT</td>
<td></td>
</tr>
<tr>
<td>Inout</td>
<td>wsBuffer</td>
<td>WSTRING(1)</td>
<td>the buffer-wstring which is to be analyzed</td>
</tr>
<tr>
<td>Input</td>
<td>udiSize</td>
<td>UDINT</td>
<td>The size of buffer-wstring. Use SizeOf(...) (0=no limit)</td>
</tr>
</tbody>
</table>

Function
Determines the length of a given wstring.

Graphical Illustration

Function Description
The first parameter is the ‘buffer’-wstring which is to be investigated. For technical reasons, the IN_OUT-parameter is typed as WSTRING(1), but in fact you may pass wstrings of any size here.

The second parameter must hold the actual size of that wstring variable or zero if the input wstring is guaranteed to be zero-terminated.

In contrast to the standard LEN function, this function is not restricted to short wstrings. For robustness reasons, the length of the reserved wstring can be passed with the udiSize parameter. If 0 is passed, no size check takes place.

If the wstring is longer than the given buffersize, the buffersize is returned (which is the maximum possible length of the wstring +1).

2.4 10 C99-Style

2.4.1 wsnprintf (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>wsnprintf</td>
<td>WagoTypes.eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>pResult</td>
<td>POINTER TO WORD</td>
<td>buffer for result wstring</td>
</tr>
<tr>
<td></td>
<td>udiResultSize</td>
<td>UDINT</td>
<td>size of the result wstring. Use SizeOf(...)</td>
</tr>
<tr>
<td></td>
<td>pFormat</td>
<td>POINTER TO WORD</td>
<td>control wstring</td>
</tr>
<tr>
<td></td>
<td>pVarList</td>
<td>POINTER TO WagoTypes.aPointerArray</td>
<td>list of arguments</td>
</tr>
<tr>
<td></td>
<td>udiListSize</td>
<td>UDINT</td>
<td>number of arguments</td>
</tr>
</tbody>
</table>

Function
Formatted output conversion.

Graphical Illustration
Function Description

wsnprintf() produces a formatted output wstring, of which location and maximum size are given by pResult and udiResultSize.

This is a helper for implementing WFormat1(). WFormat9(). In contrast to the latter, wsnprintf could process a variable number of inputs and an unlimited output size, but is more complicated to use.

Please, look at ‘WFormat1()’ for a comprehensive description of the format control string.

Example:

```pascal
VarList : ARRAY[0..1] OF POINTER TO WORD;
sResult : WSTRING;
VarList[0] := adr(sFirstvar);
VarList[1] := adr(sSecondVar);

eResultCode := wsnprintf( adr(sResult), SizeOf(sResult), adr("Var1=%s Var2=%s\n"),
                          adr(VarList), 2 );
```

<table>
<thead>
<tr>
<th>result codes</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>EINVAL</td>
<td>output buffer has invalid size (zero)</td>
</tr>
<tr>
<td>ENODATA</td>
<td>a number of one or more variables are indicated, but the variable list does not exist</td>
</tr>
<tr>
<td>ENOSPC</td>
<td>There is no space left in the output buffer</td>
</tr>
</tbody>
</table>
### VersionHistory (GVL)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WagoSysVersion.ProjectInfo</td>
<td></td>
</tr>
</tbody>
</table>

WagoAppWString.library

<table>
<thead>
<tr>
<th>date</th>
<th>version</th>
<th>author</th>
<th>change</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.01.2019</td>
<td>1.0.1.1</td>
<td>WAGO / u013972</td>
<td>Buf Fix: Problem with the placeholder resolution</td>
</tr>
<tr>
<td>06.09.2017</td>
<td>1.0.0.1</td>
<td>WAGO / u013972</td>
<td>Bug Fix: Library resolution</td>
</tr>
</tbody>
</table>

**Release Notes:**

**Known Bugs:**
This is a dictionary of all referenced libraries and their name spaces.

**Standard64**

*Library Identification:*
- Placeholder: Standard64
- Default Resolution: Standard64, * (System)
- Namespace: Standard64

*Library Properties:*
- LinkAllContent: False
- Optional: False
- QualifiedOnly: True
- SystemLibrary: False
- PublishSymbolsInContainer: True

**WagoAppString**

*Library Identification:*
- Placeholder: WagoAppString
- Default Resolution: WagoAppString, * (WAGO)
- Namespace: WagoAppString

*Library Properties:*
- LinkAllContent: False
- QualifiedOnly: True
- SystemLibrary: False
- Optional: False

**WagoAppTime**

*Library Identification:*
- Placeholder: WagoAppTime
- Default Resolution: WagoAppTime, * (WAGO)
- Namespace: WagoAppTime
Library Properties:

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

WagoSysErrorBase

Library Identification:
Placeholder: WagoSysErrorBase
Default Resolution: WagoSysErrorBase, *(WAGO)
Namespace: WagoSysErrorBase

Library Properties:

- LinkAllContent: False
- QualifiedOnly: True
- SystemLibrary: False
- Optional: False

Library Parameter:
Parameter: RES_LOG_MAX_FILESIZE = 2000
Parameter: RES_LOG_MAX_FILES = 1
Parameter: RES_LOG_MAX_ENTRIES = 200
Parameter: RES_LOG_NAME = ‘WagoAppResultLogger’

WagoSysPlainMem

Library Identification:
Placeholder: WagoSysPlainMem
Default Resolution: WagoSysPlainMem, *(WAGO)
Namespace: WagoSysPlainMem

Library Properties:

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

WagoSysTypedefs_Internal_32Bit

Library Identification:
Placeholder: WagoSysTypedefsInternal
Default Resolution: WagoSysTypedefs_Internal_32Bit, *(WAGO)
Namespace: WagoTypesInternal

Library Properties:
• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

**WagoSysVersion**

*Library Identification:*
Name: WagoSysVersion
Version: 1.0.0.0
Company: WAGO
Namespace: WagoSysVersion

*Library Properties:*

• LinkAllContent: False
• QualifiedOnly: True
• SystemLibrary: False
• Optional: False

**WagoTypesCommon**

*Library Identification:*
Placeholder: WagoTypesCommon
Default Resolution: WagoTypesCommon, * (WAGO)
Namespace: WagoTypes

*Library Properties:*

• LinkAllContent: False
• Optional: False
• QualifiedOnly: False
• SystemLibrary: False
• PublishSymbolsInContainer: True

© WAGO Kontakttechnik GmbH & Co. KG, Germany 2018 – All rights reserved. For the avoidance of doubt, this copyright notice does not only apply to the information above but also and primarily to the described library itself. Please note that third-party products are always mentioned without reference to intellectual property rights, including patents, utility models, designs and trademarks, accordingly the existence of such rights cannot be excluded. WAGO is a registered trademark of WAGO Verwaltungsgesellschaft mbH.