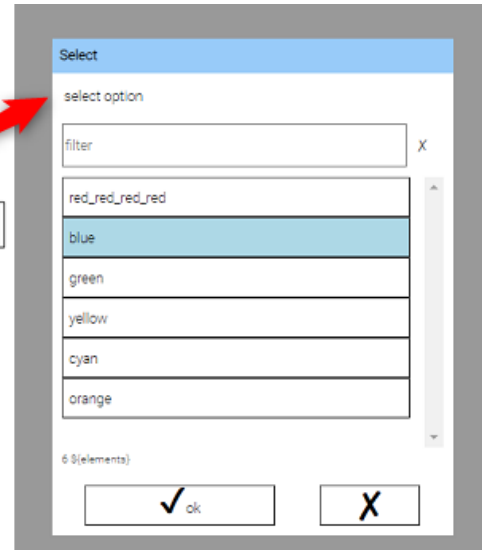
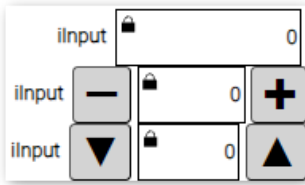
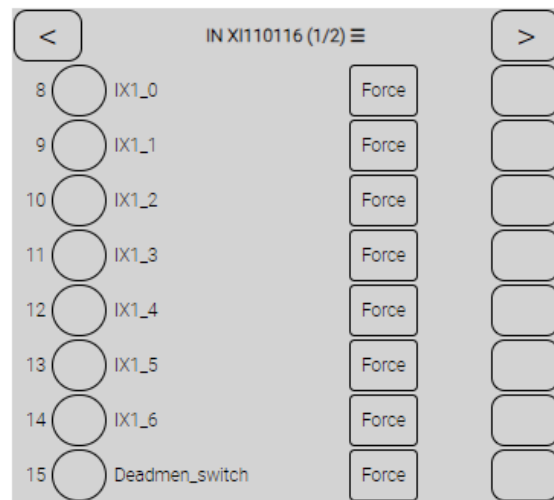
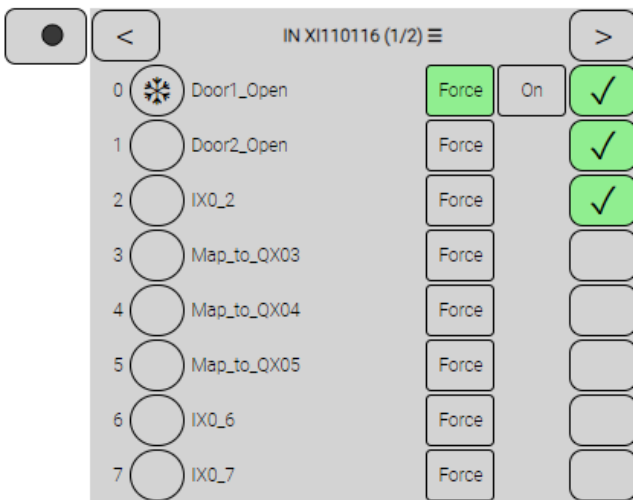
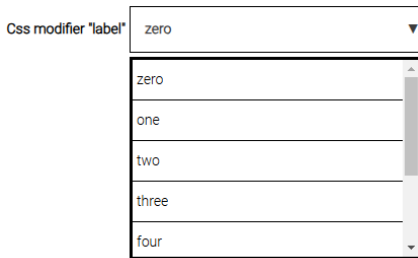


WebIQ package cxVisuals



IT ID	IT \${myName}	IT \${myX}	IT \${myY}	IT \${myZ}	IT \${myEnum}	IT \${myFlag}
000	name_0	0.100	0.20	0.3	Black	Off
001	name_1	1.100	1.20	1.3	\$(tl-white)	On
002	name_2	2.100	2.20	2.3	Blue	Off
003	name_3	3.100	3.20	3.3	\$(tl-green)	On
004	name_4	4.100	4.20	4.3	\$(tl-yellow)	Off
005	name_5	5.100	5.20	5.3	Red	On
006	name_6	6.100	6.20	6.3	\$(tl-orange)	Off
007	name_7	7.100	7.20	7.3	?TxtLst?=7	On
008	name_8	8.100	8.20	8.3	?TxtLst?=8	Off
009	name_9	9.100	9.20	9.3	?TxtLst?=9	On



Contents

1	Introduction	4
2	cx-button	5
2.1	Features	5
2.2	CSS Modifiers	5
3	cx-select	6
3.1	Features	6
3.2	CSS Modifiers	6
3.3	String item Text lists CSV format	6
3.4	String item Text lists JSON format	7
3.5	JSON object WebIQ Text lists	7
3.6	Example of lists defined as json objects	7
4	cx-input	8
4.1	Features	8
4.2	CSS Modifiers	8
5	cx-inputX	9
5.1	Features	9
5.2	CSS Modifiers	9
6	cx-table controlled by WebIQ: array structure variables	10
6.1	Features	10
6.2	Css Modifiers	10
6.3	Configuration of table	10
6.4	WebIQ Text lists	12
7	cx-table controlled by WebIQ: single variables	13
7.1	Features	13
7.2	Css Modifiers	13
7.3	Configuration of table	13
7.4	WebIQ Text lists	13
8	cx-table controlled by PLC: array structure variables	14
8.1	Features	14
8.2	CSS modifiers	14
8.3	Configuration of table	15
8.4	WebIQ Text lists	16
8.5	Implementation into the customer project	16
9	cx-table controlled by PLC: Single variables	17
9.1	Features	17
9.2	CSS modifiers	17
9.3	Configuration of table	17
9.4	WebIQ Text lists	17
9.5	Implementation into the customer project	18

10	cx-ioForce controlled by PLC	19
10.1	Features	19
10.2	CSS modifiers	19
10.3	Code generation for digital inputs/outputs.....	19
10.4	Implementation into the customer project.....	22
11	messageTop().....	23
11.1	Features	23
11.2	Implementation into the customer project.....	23

1 Introduction

This document is the help document for the WebIQ documentation project and illustrates the features of the widgets in the cx-visuals package. The associated project demonstrates the usage of the custom widgets provided in the package cxVisuals. The following widgets can only be used, with the matching PLC program:

- cx-forceIO
- cx-table.
This widget has 2 variants
 - variant 1: table management is done in WebIQ
 - variant 2: table management is done in PLC

To get familiar with the features and configuration of the widgets:

- run the WebIQ and PLC project
- check the widget configuration
- use the widgets to get to know their behavior
- click on the button "?" to read additional information, stored in this help document.

2 cx-button

with ON label only	Set = 0 long label to show line wrapping	Set = 1	Set = 2	Set = 3
with ON & OFF label	Set = 0 (ON)	Set = 1 (OFF)	Set = 2 (OFF)	Set = 3 (OFF)
with ON icon only	Set = 0	Set = 1	Set = 2	Set = 3
with ON & OFF icon	Set = 0	Set = 1	Set = 2	Set = 3

Widget to set an item and/or execute UI-Actions.

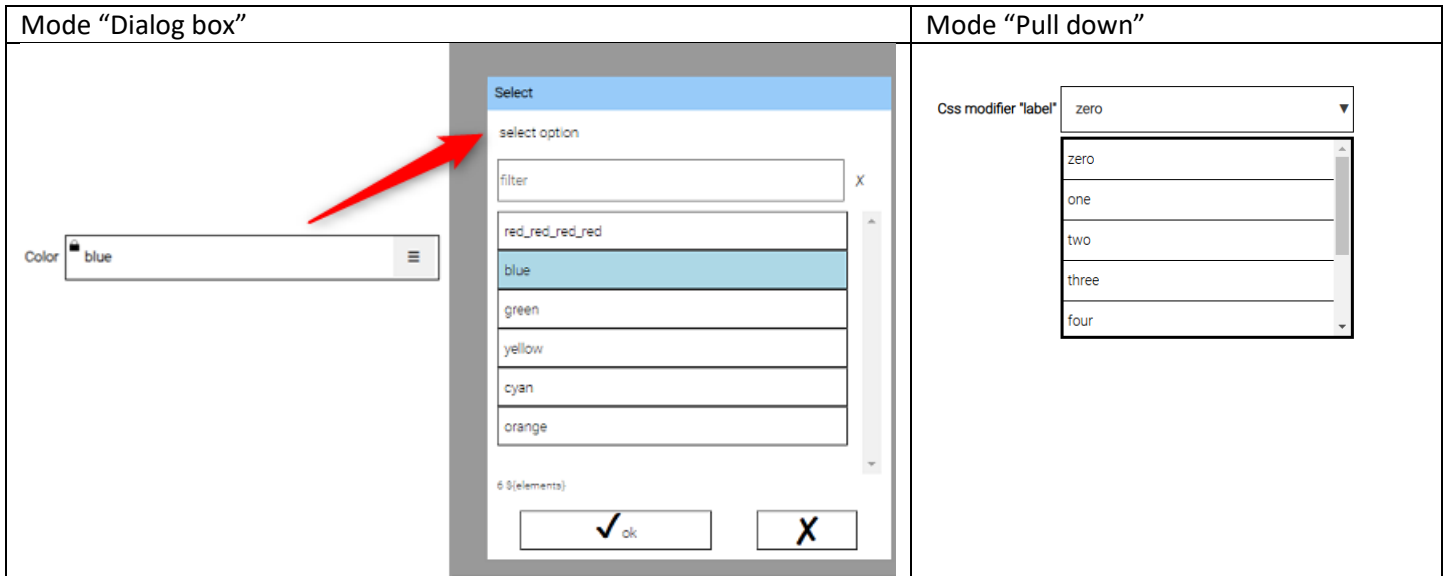
2.1 Features

- 3 modes available
 - Set: * Set ON value
 - On-Off: * onPress: Set ON value
* onRelease: Set OFF value
 - Toggle: * toggle between ON and OFF value
- Multi line button label, with word wrapping for long labels
- Good readability of disabled widgets (display lock icon instead of changing opacity)
- ON and OFF label (if OFF label is not provided ON labels is always displayed)
- ON and OFF icon (if OFF icon is not provided ON icon is always displayed)
- Selectable icon position (None, Left, Right, Top, Bottom)
- Labels and icons are optional
- When using ctrl item, the status item is optional.
A status item is often used in PLC program, as a reaction/mirror of the ctrl item.
- Optional usage of UI-Actions (onPress, onRelease, onWhilePressed) with or without usage of ctrl item.

2.2 CSS Modifiers

CSS Modifier	Description
hidden	Hides widgets
invisible	Makes widget invisible

3 cx-select



Widget to set an item by selecting text from a list.

3.1 Features

- 2 modes available
 - Dialog box: Dialog box with filter option
 - Pull down: pull down list without filter option
- Undefined values are displayed as “?TxtList?=<value>” e.g. “?TxtLst?=2”
- Supported value item types: string, number, and boolean
- Text list supported:
 - Defined as String (CSV Format and JSON) see below
 - JSON objects defined in local script cfg00_List.js
- Good readability of disabled widgets (display lock icon instead of changing opacity)

3.2 CSS Modifiers

CSS Modifier	Description
hidden	Hides widgets
invisible	Makes widget invisible
labelA : labelE	Set width of label (smallest) Set width of label (biggest)

3.3 String item Text lists CSV format

Use language variables to supported multiple languages e.g. 1=\${red}.

Syntax: ' <value1>=<label1>;<value2>=<label2> '

Example: '1=red;2=blue;3=green;4=yellow;5=cyan;6=orange '
'1=\${red};2={blue};3=\${green};4=\${yellow};5=\${cyan};6=\${orange} '

See UI-Action: **setTextListItem**

3.4 String item Text lists JSON format

The structure of the JSON object is identical to the list defined in local script `cfg00_List.js`.

Example with WebIQ localization variables	Syntax
<pre>{ "0": "\${t1-black}", "1": "\${t1-white}", "2": "\${t1-blue}", "3": "\${t1-green}", "4": "\${t1-yellow}", "5": "\${t1-red}", "6": "\${t1-orange}" }</pre>	<pre>{ "<value 0>": "<label 0>", "<value 1>": "<label 1>", "<value 2>": "<label 2>", "<value 3>": "<label 3>", "<value 4>": "<label 4>", "<value 5>": "<label 5>", "<value 6>": "<label 6>" }</pre>

3.5 JSON object WebIQ Text lists

The lists are defined as JSON objects and can contain text and images.

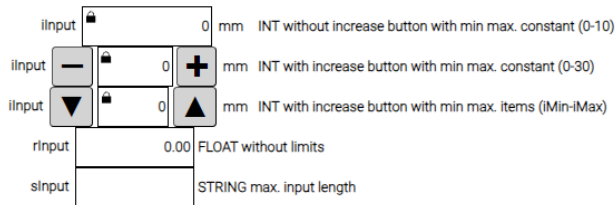
- Use localization variables to support multiple languages.
- Use HTML `` tag to display images
- See example in local script: `cfg00_List.js`

3.6 Example of lists defined as json objects

```
// predefined text lists can be overwritten
// cxt.textList["gtl-onOff"]
// cxt.textList["gtl-trueFalse"]

// define images for displaying in select box
const iHeight = "24px"
const sBlack = `
```

4 cx-input



Widget to edit a number or single line text.

4.1 Features

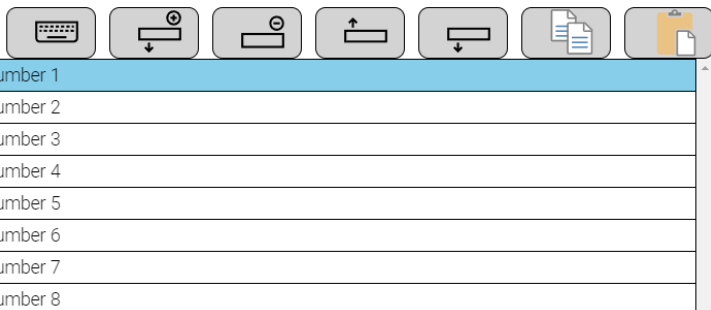
- Displays automatically the matching virtual keypad (numeric or alphanumeric) dependent on item data type
- Uses Rexroth virtual keypad (from package cxTools) instead of WebIQ virtual keypad
 - Compared to the WebIQ alphanumeric keypad, the cxTools keypad:
 - has a clearer and more intuitive layout
 - is easier to use (more edit features)
 - configuration for new languages, requires much less effort
 - it is shown on Android systems, too.
 - It is scalable & moveable at runtime.
- Number items
 - are displayed right aligned
 - has an option to show increase/decrease buttons
 - supports WebIQ unit conversion for editing
- String items:
 - are displayed left aligned
 - includes workaround for missing support of Unicode characters in ctrlX OpcUA server (since ctrlX 2.6 OpcUA server supports UNICODE for WSTRING)
 - STRING is converted to UTF-8 string for PLC
 - WSTRING is converted to UNICODE string in PLC
 - use cx-inputX to edit multi line strings
- Good readability of disabled widgets (display lock icon instead of changing opacity)

4.2 CSS Modifiers

CSS Modifier	Description
hidden	Hides widgets
invisible	Makes widget invisible
labelA : labelE	Set width of label (smallest) Set width of label (biggest)

5 cx-inputX

Multiline Editor



Widget to edit a multi-line text.

5.1 Features

- Supports 2 views. You can toggle between both.
 - View: text field
 - For usage with hardware keyboard
 - Editing in a multiline text field
 - View: table
 - Every line of the string can be edited separately
 - Features: Add, delete, move up, move dn, copy & paste row
 - Highlighting of a single row (e.g., to show active program line)
 - Normal (for display) or large (for easy touching) line height
- Good readability of disabled widgets (display lock icon instead of changing opacity)
- Uses Rexroth virtual keypad (form package cxTools) instead of WebIQ virtual keypad
 - Compared to the WebIQ alphanumeric keypad, the cxTools keypad:
 - has a clearer and more intuitive layout
 - is easier to use (more edit features)
 - configuration for new languages, requires much less effort
 - it is shown on Android systems, too.
 - It is scalable & moveable at runtime.
- Includes workaround for missing support of Unicode characters in ctrlX OpcUA server (since ctrlX 2.6 OpcUA server support UNICODE for WSTRING)
 - STRING is converted to UTF-8 string for PLC
 - WSTRING is converted to UNICODE string in PLC

5.2 CSS Modifiers

CSS Modifier	Description
hidden	Hides widgets
invisible	Makes widget invisible
bigHeight	Increase the height of the table row, making it easier to touch.
monospace	Set font style to a fixed font. E.g. to display program code

6 cx-table controlled by WebIQ: array structure variables

ID	\$myName	\$myX	\$myY	\$myZ	\$myEnum	\$myFlag
000	name_0	0.100	0.20	0.3	Black	Off
001	name_1	1.100	1.20	1.3	White	On
002	name_2	2.100	2.20	2.3	Blue	Off
003	name_3	3.100	3.20	3.3	Green	On
004	name_4	4.100	4.20	4.3	Yellow	Off
005	name_5	5.100	5.20	5.3	Red	On
006	name_6	6.100	6.20	6.3	Orange	Off
007	name_7	7.100	7.20	7.3	?TxtLst?=7	On
008	name_8	8.100	8.20	8.3	?TxtLst?=8	Off
009	name_9	9.100	9.20	9.3	?TxtLst?=9	On

Widget to display/edit WebIQ array items in a table.

6.1 Features

- Supported data types: Integer, Float, Bool, String
 - Text lists support, defined in WebIQ. See local script: cfg00_List.js
- Edit behavior (Min, Max, Editable) can be adapted by implementing the global JS function isCellEditable()
 - See local script isCellEditable.js
- Supports multiple simultaneously running web clients. The last one wins.
 - feature to highlight changed values for a moment, can be configured
- Table cells are automatically updated when a cell item is changed
- Filter, Edit, Sort can be configured
- Data items MUST be communicated via OpcUA when you want to exchange them with PLC
- Good readability of disabled widgets (display lock icon instead of changing opacity)
- Configuration is done in script editor. Format is JSON.

Why is the configuration of the widget not done in the standard way?

- The entries are so long that they would only be partially visible
- There are so many entries/rows that the definition is only partly visible

6.2 Css Modifiers

CSS Modifier	Description
hidden	Hides widgets
invisible	Makes widget invisible
bigHeight	Increase the height of the table row, making it easier to touch.
fullWidth	Set width of table to full available width
alignLeft	Left align all table cell content
alignRight	Right align all table cell content
alignCenter	Center all table cell content
singleClickEdit	Click for Select + Edit (Default is: 1st click selects 2nd click edits)
disableSort	Disable sort of all columns

6.3 Configuration of table

```
const webIqTableArray = {
  array: [0, 8],
  markChanges: true,
  id: { filter: true, sort: true },
  data: [
    { item: "arPoints[*].sName", header: "Name", sort: "S", width: "120px" },
    { item: "arPoints[*].rX", header: "X", sort: "F", format: "f2s(val,3)", width: "50px" },
  ]
}
```

```

{item:"arPoints[*].rY", header:"Y", sort:"F", format: "f2s(val,2)", width:"50px"},
{item:"arPoints[*].rZ", header:"Z", sort:"F", format: "f2s(val,3)", width:"50px"},
{item:"arPoints[*].iEnum", header:"Enum", sort:"L", width:"80px", list:"gtl-color"},
{item:"arPoints[*].bFlag", header:"Bool", sort:"L", width:"60px", list:"gtl-onOff"},
],
cmd: [
  {cmd:"copy", lbl: "Copy", cfm:""},
  {cmd:"paste", lbl: "Paste", cfm: "Confirm paste?"},
  {cmd:"mvUp", lbl: "▲", cfm: ""},
  {cmd:"mvDn", lbl: "▼", cfm: ""},
  {cmd:"clear", lbl: "Clear?", cfm: "Clear selected row?"},
  {cmd:"clearAll", lbl: "Clear all?", cfm: "Clear all rows?"},
]
}

```

Root object

key	description	Array	Item
array	Defines the [first, last] index of the array to display	+	-
markChanges	true: Activate highlighting of value changes for a moment. false: disable	+	+
tableKey	Unique key to detect which table is calling function isCellEditable()		
id.filter	true: Enable filter for column id . false: disable	+	+
id.sort	true: Enable sort for column id . false: disable	+	+
id.header	Header text of column id (optional)	-	+
id.width	Width of column id (optional)	-	+
name.filter value.filter unit.filter	true: enable filter for column. false: disable	-	+
name.sort value.sort unit.sort	true: enable sort for column. false: disable	-	+
Id.header name.header value.header unit.header	Header text of column (optional). Default header is displayed, if not provided.	-	+
Id.width name.width value. width unit. width	Minimal width of column (optional). Default width is used, if not provided	-	+

Data object

data.item	Name of item to display. Placeholder for array index is a "*" e.g. arPoint[*].x	+	+
data.unit	Unit of item. Only used if item has no property unit.	+	+
data.header	Header text of column.	+	-
data.edit	Item is "+"=editable "-"=read only. Default is "+" Can be overwritten by JS function isCellEditable()	+	+
data.filter	"+"=enable filter, "-"= disable filter. Default is "+"	+	+
data.sort	Sort type: "S"=string "F"=float "I"=integer "B"=bool "L"=list. Default is "S"	+	+
data.format	Defines which function used to format the item. (optional) <ul style="list-style-type: none"> f2s mapped to cxt.float2String(rVal, iDigits = 2, iSecLen = 0) <ul style="list-style-type: none"> f2s(10.129, 2) = 10.13 f2s(10234.129, 2, 3) = 10 234.13 i2s mapped to cxt.int2String(iVal, iBase = 10, iLen = 0, iSecLen = 0, sFS = "0") <ul style="list-style-type: none"> i2s(1, 10, 3) = 001 i2s(10123,10, 0, 3) = 10 123 i2s(255, 16) = FF Application specific format functions can be created and used.	+	+

data.list	Name of the text list (defined in cfg10_Tables.js) List is used for data types: Integer, Float & Boolean. See chapter: JSON object WebIQ Text lists	+	+
data.width	Sets minimal table column width (optional)	+	+
data.min	Number: Min. input value (optional). If not provided item property is used. It can only be used to reduce the range defined in the item properties. String: Not used	+	+
data.max	Number: Max. input value (optional). If not provided item property is used. It can only be used to reduce the range defined in the item properties. String: Max. text length (optional)	+	+

Cmd object


key	description	Array	Item
cmd.cmd	Defines the features, displayed as table command buttons. Commands are not case sensitive: <ul style="list-style-type: none"> • copy: Copy all values of selected row • paste: Paste copied values to selected row • mvUp: Swap values of selected row with previous row • mvDn: Swap values of selected row with next row • clear: Clear/initializes values of selected row • clearAll: Clear/initializes values of all rows 	+	-
cmd.lbl	Label of button	+	-
cmd.cfm	Confirmation text. If defined the command must be confirmed, before execution	+	-

The default value is used when nothing is provided in the JSON configuration above.

6.4 WebIQ Text lists

See chapter: 3.5 JSON object WebIQ Text lists

7 cx-table controlled by WebIQ: single variables

↕#	↕ Name	↕ Value	↕ Unit
1	rInput	2.0	m
2	sInput	my text	
3	BOOL Input	✓	
4	INT Color	 Green	pieces
5	rInput1	23.00	

Widget to display/edit single WebIQ items in a table.

7.1 Features

- Supported data types: Integer, Float, Bool, String
 - Text lists support, defined in WebIQ. See local script: cfg00_List.js
- Edit behavior (Min, Max, Editable) can be adapted by implementing the global JS function **isCellEditable()**
 - See local script isCellEditable.js
- Supports multiple simultaneously running web clients. The last one wins.
 - feature to highlight changed values for a moment, can be configured
- Table cells are automatically updated when a cell item is changed
- Filter, Edit, Sort can be configured
- Items MUST be communicated via OpcUA when you want to exchange them with PLC
- Good readability of disabled widgets (display lock icon instead of changing opacity)
- Configuration is done in script editor. Format is JSON.
 - Why is the configuration of the widget not done in the standard way?
 - The entries are so long that they would only be partially visible
 - There are so many entries/rows that the definition is only partly visible

7.2 Css Modifiers

See chapter: 6.2 Css Modifiers

7.3 Configuration of table

```
const webIqTableItems = {
  markChanges: true,
  id: { filter: true, sort: "I", header: "#", width: "50px" },
  name: { filter: true, sort: "S", header: "Name", width: "50px" },
  value: { filter: true, sort: "S", header: "Value", width: "60px" },
  unit: { filter: true, sort: "S", header: "Unit", width: "70px" },

  data: [
    { item: "rInput", format: "f2s(val,1)" },
    { item: "sInput", unit: "string" },
    { item: "bInput", list: "gtl-checkmark" },
    { item: "iInput", list: "gtl-color" },
    { item: "rInput1", format: "f2s(val,2)" },
  ],
}
```

Meaning of the JSON keys and CSS modifiers: see chapter: **cx-table** controlled by **WebIQ**: array structure variables

7.4 WebIQ Text lists

See chapter: 3.5 JSON object WebIQ Text lists

8 cx-table controlled by PLC: array structure variables

↑↓ \$(#)	↑↓ \$(name)	\$(X-Pos)	\$(Y-Pos)	\$(Z-Pos)	↑↓ \$(enum-demo)	↑↓ \$(enum-demo)	↑↓ \$(test)
1		0.000	0.000	0.000	value A	Black	Off
2		0.000	0.000	0.000	value A	Black	Off
3		0.000	0.000	0.000	value A	Black	Off
4		0.000	0.000	0.000	value A	Black	Off
5		0.000	0.000	0.000	value A	Black	Off
6		0.000	0.000	0.000	value A	Black	Off
7		0.000	0.000	0.000	value A	Black	Off

\$(Clear)
\$(ClearAll)
\$(Copy)
\$(Paste)
\$(mvUp)
\$(mvDn)

Widget to display PLC array variables in a table.

8.1 Features

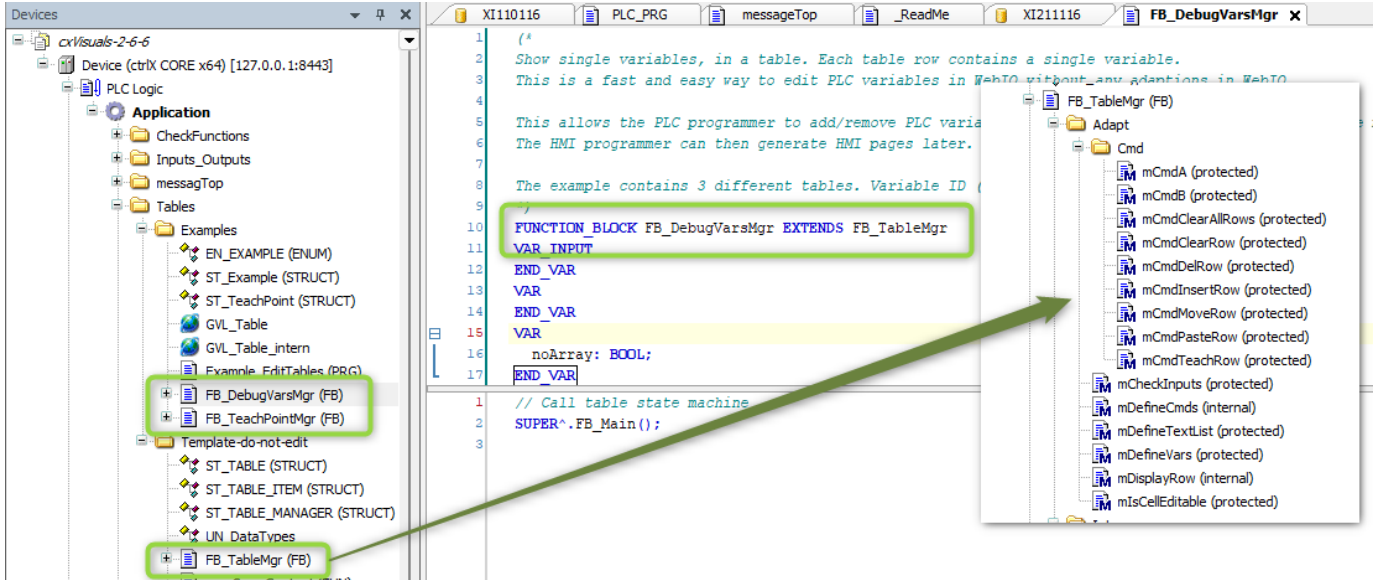
- Configuration completely done in PLC, therefore it is only working, when the PLC code is running. See FB_TeachPointMgr for example implementation.
- Supported data types: Integer, Float, Bool, String
- Data exchange is done via 4 variables. Plc array data must NOT be communicated via OpcUA
- Display multiple arrays in table is supported. The array must have:
 - Identical min. and max. index (e.g. arTest[1..10] of ST_Test1; arTest2[1..10] of ST_Test2;)
- Editors: numeric & alpha numeric keypad, cx-select user defined editors
- Text list supported. Text list can be defined in PLC and/or HMI.
 - PLC: See mDefineTextList(). Used only by cx-table
 - WebIQ: See local script: cfg00_List.js. Used by cx-table and cx-select
- Filter, Sort, Edit can be configured
- Simultaneous editing by several clients is possible. The last one wins.
 - Highlight of changed values not available
- Table cells are updated when any client edits a value.
- Good readability of disabled widgets (display lock icon instead of changing opacity)

8.2 CSS modifiers

Css Modifier	Description
singleClickEdit	Selects row and start edit, with a single click. Otherwise, the 1 st click selects the row, the 2 nd click start edit.
disableSort	Disables sort feature of table
fullWidth	Stretches table to full width
bigHeight	Set big height of rows, for easy touching
alignLeft	Align all cells to left
alignCenter	Align all cells to center

8.3 Configuration of table

Configuration is completely done in PLC code of folder Tables. There are 2 table examples in the subfolder Table>Examples, which extends FB_TableMgr.



Below is an incomplete list of the methods that exist for configuring the table. The complete list can be found in the PLC project, which also describes the use with comments. You can overwrite all methods in the extended FB, to adapt them to the application needs.

Define methods	Description
mDefineVars	Definition of the variables that are displayed in the table. <ul style="list-style-type: none"> Supported types: String, WString, Integers, Floats and Bool Integers and Booleans can be displayed with a text list Features: <ul style="list-style-type: none"> Title, width & alignment (left,right,center) of column format function (e.g. number of decimal points) text list name (to display text instead of numbers) Configurable features: Filter, sort & editable Unit conversion (to activate the function, enter the name of a WebIQ element with activated unit conversion) Limit input range for numbers (min./max. value) Limit editable string length (max. value)
mDefineCmds	Definition of the command buttons that are displayed below the table.
mDefineTextList	Definition of text list, which can only be used by this specific table. Alternatively, you can also define global text lists in WebIQ, specific text lists for a certain table in PLC. Global text lists can be used by all widgets of type cxTable and cxSelect.
mIsCellEditable	Here you can overwrite the default edit features of any variable. E.g. <ul style="list-style-type: none"> Disable edit in Automatic mode, adapt min./max. limits dependent on a condition.

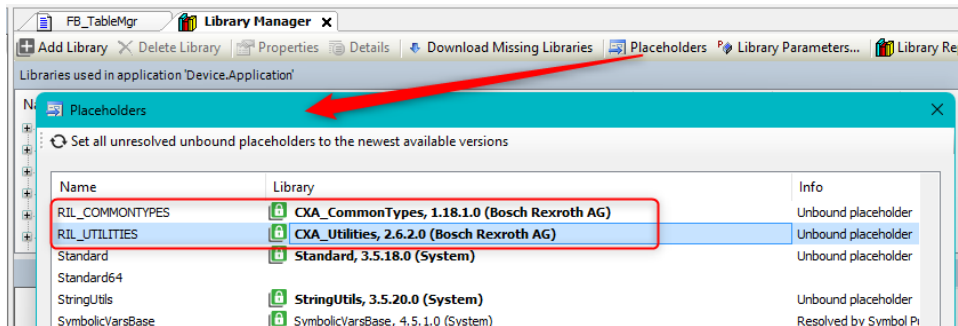
Cmd methods	Description
mCmdA	First free usable command
mCmdB	Second free usable command
mCmdClearAllRows	Clears/Initializes all table row variables
mCmdClearRow	Clears/Initializes selected row variables
mCmdMoveRow	Swaps content of selected row with previous or next row
mCmdPasteRow	Paste content of previously copied row into selected row
mCmdTeachRow	Copy x,y,z coordinates to selected row

8.4 WebIQ Text lists

See chapter 3.5 JSON object WebIQ Text lists

8.5 Implementation into the customer project

- Start PLC Engineering and open example project archive
- Start a 2nd instance of PLC Engineering and open your project
- Copy & Past folders **Table** & **msgTop** from example into your project
- Add libraries:
 - App_CommonData
 - App_Tools
 - SysMem
 - StringUtils
- Assign libraries for placeholders:
 - RIL_COMMONTYPES => CXA_CommonTypes
 - RIL_UTILITIES => CXA_Utilities



- Call example program Tables>Examples>Example_EditTables() from your main program.
- While the PLC program is working with fixed string length, you must adapt the size of the string, used for communication with HMI, in **GVL_Table_intern**
 - TABLE_DATA_SIZE: WORD:= TABLE_PAGE_ROWS * TABLE_COL_MAX * TABLE_COL_LEN;

9 cx-table controlled by PLC: Single variables

ID	Variable name	Value
1	Select Debug table	debug table 0
2	ITestCounter (Read)	1865
3	bTest	?TxtLst?=0
4	ITest	10
5	rTest	33.3

Widget to display PLC single variables in a table.

9.1 Features

- Configuration done in PLC code, therefore it is only working, when the PLC code is running. See FB_DebugVarsMgr for example implementation. For detailed information see comments in source code.
- Supported data types: Integer, Float, Bool, String
- Data exchange is done via 4 variables. Plc array data must NOT be communicated via OpcUA
- Editors: numeric & alpha numeric keypad, cx-select and user defined editors
- Text list supported. Text list can be defined in PLC and/or HMI.
 - PLC: See mDefineTextList(). Used only by cx-table
 - WebIQ: See local script: cfg00_List.js. Used by cx-table and cx-select
- Edit can be configured. **Filter, Sort not supported???**
- Simultaneous editing by several clients is possible. The last one wins.
 - Highlight of changed values not available
- Table cells are updated when any client edits a value.
- Good readability of disabled widgets (display lock icon instead of changing opacity)
- Option to refresh complete table cyclically, which should only be used for small tables on performant hardware. Just verify the performance on your hardware.
- This table variant was implemented specifically for editing individual PLC variables at commissioning. This allows you to view and edit any PLC variable without making any adjustments in WebIQ.

9.2 CSS modifiers

Css Modifier	Description
singleClickEdit	Selects row and start edit, with a single click. Otherwise, the 1 st click selects the row, the 2 nd click start edit.
disableSort	Disables sort feature of table
fullWidth	Stretches table to full width
bigHeight	Set big height of rows, for easy touching
alignLeft	Align all cells to left
alignCenter	Align all cells to center

9.3 Configuration of table

Below is an incomplete list of the methods that exist for configuring the table. The complete list can be found in the PLC project, which also describes the use with comments.

Define methods	Description
mDefineVars	Definition of the variables that are displayed in the table. Supported types: String, WString, Integer, Float and Bool. Integers and Booleans can be displayed with a text list.
mDefineTextList	You can either define global text lists in WebIQ, specific text lists for a single table. Global text lists, can be used by all widgets of type cxTable and cxSelect.

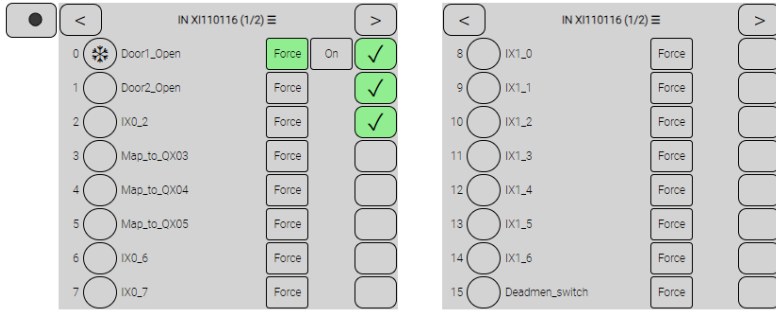
9.4 WebIQ Text lists

See chapter: 3.5 JSON object WebIQ Text lists

9.5 Implementation into the customer project

See chapter: 8.5 Implementation into the customer project

10 cx-ioForce controlled by PLC



Widget to display PLC bits of digital IO modules.

10.1 Features

- Displays status of digital IO bits up to 64 bits
- Forcing of digital IO's from WebIQ
- Checkbox for successful verified wiring of IO's at commissioning
- Automatic code generation in PLC Engineering
- No additional effort on WebIQ side, when adding/removing IO's on PLC.

10.2 CSS modifiers

Css Modifier	Description
hideForce	true: hide force button
hideIO_verified	true: Hide io verified checkbox

10.3 Code generation for digital inputs/outputs

The template includes a mechanism that allows forcing of inputs and outputs via the HMI. The code required for this feature is generated by a Python script in PLC Engineering. No additional work, like adding OPC UA variables or HMI screens, is required. So, you can save a lot of implementation time.

By default, digital IO is automatically detected, and the code is generated. Alternatively, you can still manually export the digital IO configuration.

You must accomplish these 5 steps to use this mechanism:

Step 1: Copy python code into a file e.g., makeCode_Forcelo.py (only once)

Step 2: Definition of the variables for the digital inputs/outputs

(Step 3: **Only manual export:** Export every single digital IO into a separate csv file)

Step 4: Start the Python script

Step 5: Adapt `bDeadmenSwitch` in `MachinePrg` (only once)

Step 1: Copy python code into a file

Creating the script file must be done only once. Afterwards the script can be used again and again.

- Open the POU `makeCode_Forcelo` in PLC engineering
- Copy complete python code into a text file (e.g., makeCode_Forcelo.py)
- Optional:
 - Set variable `iMapAllIOs`.
0=do not map IOs without name
1=map all IOs even the unnamed IOs. When no IO name is defined the bit address is used as IO name.
e.g., IX50_0

Manual export of CSV files:

- To enable manual export in script: Set variable `iAutoDetectIO=0`.

- Set the variable **strCsvDir** to the directory where you saved the CSV files. Therefore no dialog asks you to select the files.

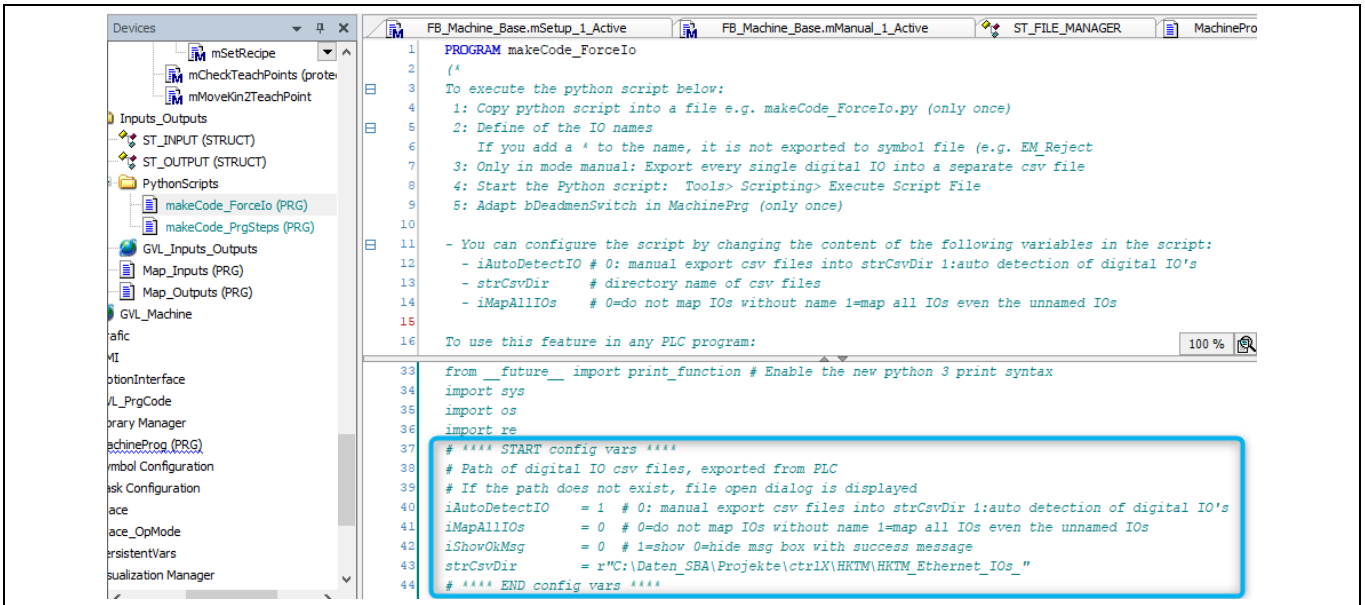


Figure 1: PLC engineering Python script

Step 2: Definition of the variables for the digital inputs/outputs

Open the configuration mapping page. Enter the variable name of inputs/outputs into the description field of the input/output page. Keep in mind to create valid PLC variable names. If you add a * to the name (e.g., EM_Reject*), it is not exported to the symbol file.

The column **Variable** must not be filled in. This prevents writing to the IOs via 2 different paths.

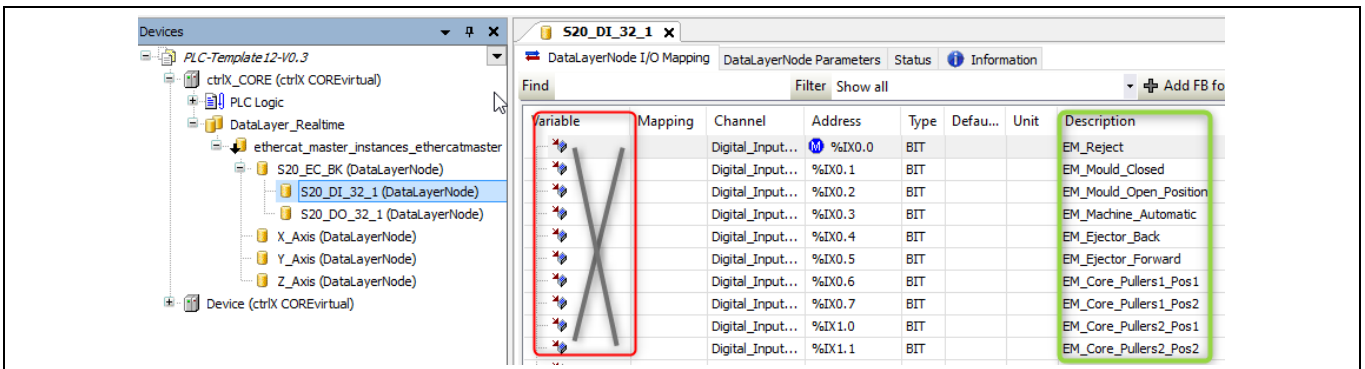


Figure 2: Names of I/O

(Step 3: Only manual export: Export every single digital IO into a separate csv file)

This step is not necessary when script run in default (=auto) mode, where the digital IOs are automatically recognized. It is only intended as a backup in case the automatic detection does not work.

Right click digital IO, select **Export mappings to CSV...** and provide a valid file name.

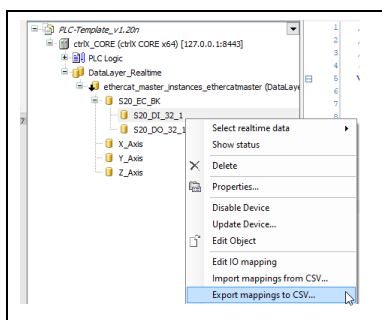


Figure 3: Digital IO nodes

Hint: All csv files must be in the same folder
Do not export any other module than digital IO's.

Manual export of CSV files:

- To enable manual export in script: Set variable **iAutoDetectIO=0**.
- Set the variable **strCsvDir** to the directory where you saved the CSV files. Therefore no dialog asks you to select the files.

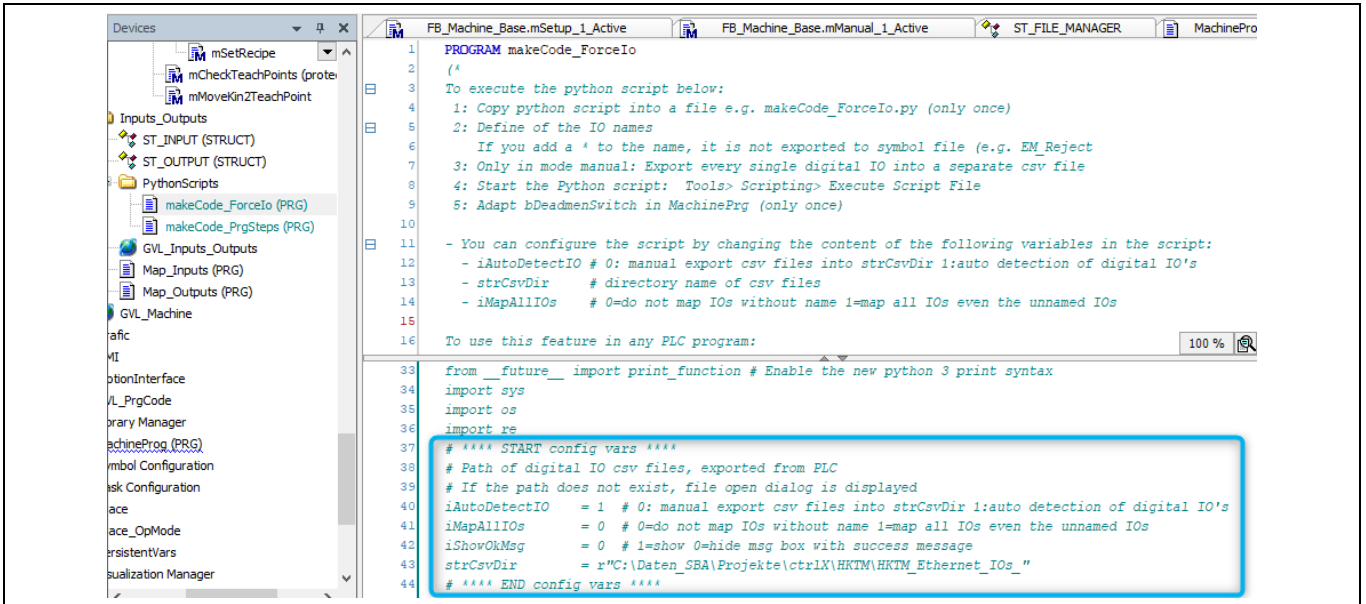


Figure 4: PLC engineering Python script

Step 4: Start the Python script

In PLC Engineering

- Click on Tools/Scripting/Execute Script File

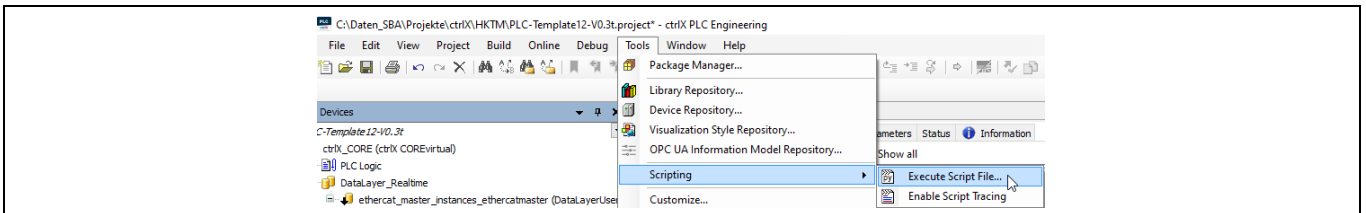


Figure 5: Execute script file

- Start the python script, by selecting it within the file browser
- Manual export only:
If you haven't set **strCsvDir**, select any of the exported csv files and confirm with **Open**
- In the message window, check whether the code generation was successful and which **DUTs, GVLs and POUs** were updated.
- Compile the program. When compile errors are thrown, fix the invalid variable names. Start creation process again.

10.4 Implementation into the customer project

- Start PLC Engineering and open example project archive
- Start a 2nd instance of PLC Engineering and open your project
- Copy & Past folder IO from example into your project
- Import both libraries into your project
- Assign libraries for placeholders: (see chapter 8.5 Implementation into the customer project)
 - RIL_COMMONTYPES => CXA_CommonTypes
 - RIL_UTILITIES => CXA_Uilities
- Call Map_Inputs() as first statement and Map_Outputs() as last statement in your main program

```
// call as first statement of PLC main program
Map_inputs();

// call your main state machine
fbMainStateMachine();

// call as last statement of PLC main program
Map_Outputs();
```

11 messageTop()

The screenshot shows five stacked message boxes on an HMI interface. From top to bottom: 1. A yellow warning box with a warning icon, titled 'my warning', containing the text 'Must be closed manually' and a close button 'X'. 2. A red error box with a stop sign icon, titled 'my error', containing the text 'Must be closed manually' and a close button 'X'. 3. A red bug box with a bug icon, titled 'my bug', containing the text 'message can not be closed' and 'Reload browser to remove message'. 4. A light blue information box with an 'i' icon, titled 'my title', containing the text 'opens only once. msg1 is replaced by msg2' and 'Closes automatically after some seconds' and a close button 'X'. 5. A green information box with an 'i' icon, titled 'my information', containing the text 'automatically closed afert some seconds' and a close button 'X'. To the left of the messages, there is a code snippet: `cxt.msgTop()` which displays the message at the top. Below the messages, there is a list of available message types: 'info', 'warning', 'error', 'bug', and 'title'. At the bottom, there are three bullet points: '- Interface to use it from PLC side with the PLC function messageTop()', '- Interface uses JSON string', and '- PLC must run'.

The Plc program can display messages on the HMI. The PLC function `messageTop()` sends the message via a JSON string to the JS function `cxt.msgTop()`, which displays the message on the HMI device.

11.1 Features

Available message types:

- "info" closes automatically
- "warning" must be confirmed
- "error" must be confirmed
- "bug" cannot be closed
- "title" Only a single instance is displayed.
That means if you call it a 2nd time, the 1st message is replaced by the 2nd
All other types open a further instance of the message box

11.2 Implementation into the customer project

PLC Engineering

- Start PLC Engineering and open example project archive
- Start a 2nd instance of PLC Engineering and open your project
- Copy & Past folders **messageTop** from example into your project
- Add libraries:
 - App_CommonData
 - App_Tools
- Assign libraries for placeholders: (see chapter 8.5 Implementation into the customer project)
 - RIL_COMMONTYPES => CXA_CommonTypes
 - RIL_UTILITIES => CXA_Uilities

WebIQ

- Start WebIQ Designer and open your project
- Upload and install package cx-tools
- In "Code Manager" create a local script with the name "start-up". Add the code with yellow background.
- Alternatively the code, can be added to an existing local script.

```

module.run = function (self) {
    // enable msgTop from PLC
    // if you change the item name of array size,
    // adapt constants MSG_TOP_*
    const MSG_TOP_ITEM = "GVL_msgTop.arMsgTop.*"
    const MSG_TOP_COUNT = 3 // [0-MSG_TOP_COUNT]
    let tokMsgTop = cxt.msgTopSubscribe(MSG_TOP_ITEM, MSG_TOP_COUNT)

    /* called when this local-script is disabled */
    self.onDisable = function () {
        self.run = false; /* from original .onDisable function of LocalScript control */
        // unsubscribe when panel is closed
        tokMsgTop.unlisten()
    };
};

```

- Call local script from the projects main panel, which must always be open, while the HMI is running

12 Unit classes()