SIEMENS

SIMATIC

WinCC
WinCC Engineering V16 - Runtime Unified

System Manual

Online help printout

11/2019
Online help printout

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER
indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING
indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION
indicates that minor personal injury can result if proper precautions are not taken.

NOTICE
indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING
Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

Document order number: Online help printout
℗ 12/2019 Subject to change

Table of contents

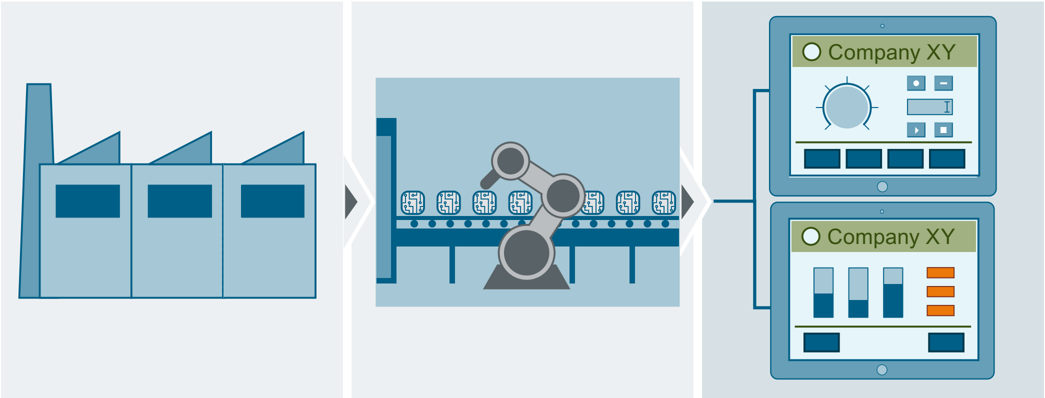
Configuring screens (RT Uni) 1

1.1 Basics (RT Uni)

1.1.1 Basics of screens (RT Uni)

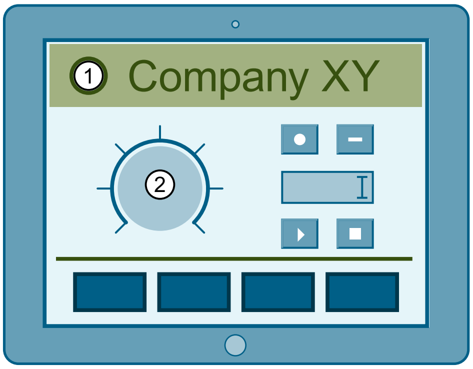
Introduction

In WinCC you create screens that a user can use to control and monitor machines and plants. When you create screens, the pre-defined object templates support you in visualizing your plant, displaying processes and specifying process values.



Structure of screens

Insert the objects you need to represent the process into your screen. Configure the objects to match the requirements of your process.



A screen can consist of static and dynamic elements:

- ① Static elements such as text or graphic objects in the screen above do not change in runtime.
- ② Dynamic elements change their status based on the process. You visualize current process values from the memory of the PLC or the HMI device. Dynamic objects include, for example, alphanumeric displays, trends and bars, as well as input fields on HMI device, such as I/O fields, switches and sliders. Process values and operator inputs are exchanged between the PLC and the HMI device by means of tags.

Start screen

The start screen is the initial screen displayed when the project is started in runtime. You must define a separate start screen for each target system. From the start screen, the user navigates to the other screens. Create the start screen in runtime settings under "General > Screen > Start screen".

Note

To compile and download a project, you need to have defined a start screen in the project. Ensure that a start screen is specified in your project.

Screen window

You display other project screens in the screen window. The screen window enhances navigation between screens and allows "screen in a screen" display.

Screen windows can, for example, be used for:

- Frequent switching between plant units
- Showing and hiding screens, for example without exiting central process visualization
- Displaying multiple plant units in one screen

Property list in the Inspector window

You can view and edit the properties of an object using the property list in the Inspector window. Changes to the size and position or dynamizations of the object are possible, for example.

The properties are displayed either in alphabetical order or in categories.

You can sort the property list as follows:

- Display of properties in alphabetical order
- Display of the properties grouped in categories

With both views, all details of the individual properties can be shown or hidden:

- All details are shown
- All details are hidden

1.1.2 Task cards (RT Uni)

Introduction

The following task cards are available in the "Screens" editor:

- Toolbox: Display and operating objects
- Layout: Aid for customizing the display
- Libraries: Administration of the project library and of the global libraries

Toolbox

The "Toolbox" task card contains objects in different panes:

- Basic objects
- Elements
- Controls
- My controls
- Graphics

You paste objects from the palettes into your screens by drag&drop or a double click. The objects available for selection are determined by the features of the HMI device you are configuring.

Layout

The "Layout" task card contains the following panes for displaying objects and elements:

- Layers: Serves to manage screen object layers The layers are displayed in a tree view and contain information about the active layer and the visibility of all layers.
- Objects out of range: Objects that lie outside the visible area are displayed with name, position and type.

Libraries

The "Libraries" task card show the following libraries in separate panes:

- Project library: The project library is stored together with the project.
- Global library: The global library is stored in a separate file in the specified path on your configuration PC.

1.1.3 Change size and position of an object (RT Uni)

Introduction

When you select an object, it is enclosed by a rectangle with handles. You have the following options for changing the size and position of an object:

- Drag the handles using the mouse.
- Configure properties in the Inspector window.

Note

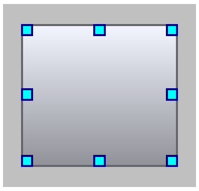
The position of the "Polygon", "Polyline" and "Line" objects can be changed by specifying positions X and Y in the Inspector window.

Requirement

You have opened the work area containing at least one object.

Change object size

1. Select the object you want to resize.
   The selection rectangle appears. The following figure shows a selected object:

2. Drag a resizing contact of the rectangle to a new position.
   The size of the object changes.

Alternatively, enter the "Height" and the "Width" of the object under "Properties".

Change object position

1. Select the object whose position you want to change.
   The selection rectangle appears.
2. Click on the object and drag it to the desired position.
   The object is moved to the new position.

Alternatively, enter the coordinates "Position X" and "Position Y" for the position of the object in relation to the screen origin under "Properties". The zero position is located at the top left-hand corner of the screen.

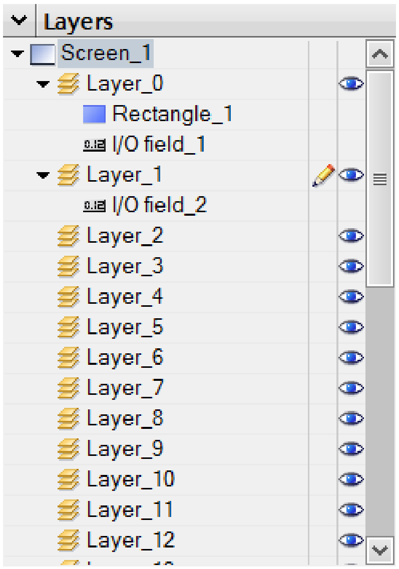
1.1.4 Using layers (RT Uni)

1.1.4.1 Basic information on using layers (RT Uni)

Layers

Use layers in order to achieve differentiated editing of the objects in a screen. Using layers, multiple objects can be combined and edited together, for example. Layers are also used to improve clarity during configuring, because multiple objects can be hidden and displayed again when required.

A screen has 32 layers. The name of the individual layers is determined by the user interface language and changes when the user interface language is changed. If you assign objects to the layers, you thereby define the screen depth. Objects of layer 0 are located in the screen background, while objects of layer 31 are located in the foreground.



The objects of a single layer are also arranged hierarchically. If you create a new object, it is arranged in the foreground. You can shift objects forwards and backwards within a layer.

Principle of the layer technique

Always one layer of the 32 layers is active. New objects you add to the screen are always assigned to the active layer. The active layer is indicated in the "Layout > Layers" task card.

When you open a screen, all 32 layers of the screen are displayed. You can hide all layers except for the active layer in the "Layout > Layers" task card. You then explicitly edit objects of the active layer.

In the "Layout > Layers" task card, you can also manage layers and objects with drag-and-drop and the shortcut menu.

Application examples

Use layers, for example, in the following cases:

- To hide the labeling of objects when editing,
- To hide individual objects, while you configure other objects

1.1.4.2 Moving objects between layers (RT Uni)

Introduction

By default, newly inserted objects are in the foreground of the active layer. You can assign an object to a different layer and change the order of objects within a layer at a later time.

Requirement

- A screen with an object is open.

Moving objects between layers

1. Select the object in the "Layout > Layers" task card.
2. Drag-and-drop the object to the required layer.

Changing the order of objects within a layer

1. Select the object in the screen.
2. Select the desired command under "Arrange" in the shortcut menu. Depending on the current position of the object, you can move it completely into the foreground, to the front, to the back or completely into the background.

Result

The object is arranged according to the selection. In the "Layout > Layers" task card, the order of the objects is displayed as follows: Objects of layer 0 are located in the screen background, while objects of layer 31 are located in the foreground. Within a layer, the objects displayed at the top of the list are in the background of the layer.

1.1.4.3 Specifying the active layer (RT Uni)

Introduction

The screen objects are always assigned to one of the 32 layers. There is always an active layer in the screen. New objects you add to the screen are always assigned to the active layer.

The active layer is indicated by the ✏ icon in the "Layout > Layers" task card.

You can activate a different layer during configuration, if necessary.

Requirement

- You have opened a screen which contains at least one object.

Procedure

1. Select "Layout > Layers" in the "Layout" task card.
2. Select the "Set to active" command from the shortcut menu of a layer.

Result

The selected layer becomes the active layer.

1.1.4.4 Hiding and showing layers (RT Uni)

Introduction

You can show or hide the layers of a screen as required.

Requirement

- The screen is opened.

Procedure

1. Select the layer that you want to hide or show in the "Layout > Layers" task card.
2. Click one of the icons next to the corresponding layer:

- 👁 A shown layer is hidden
- 👁 A hidden layer is shown

Note

You cannot hide the active layer.

Alternative procedure

1. Select "Layout > Layers" in the "Layout" task card.
2. Select the "Hide layer" or "Show layer" command from the shortcut menu of a layer.

Result

The shown layers are displayed in the engineering system.

Note

All layers are always displayed in runtime.

1.1.5 Select multiple objects (RT Uni)

Introduction

To align the object with one another or rotate them, select all affected objects. This procedure is called "multiple selection."

The Inspector window shows all the properties of the selected objects.

You have the following options to select multiple objects:

- Draw a selection frame around the objects.
- Hold down the <Shift> key, and click the required objects.

Selection frame of a multiple selection

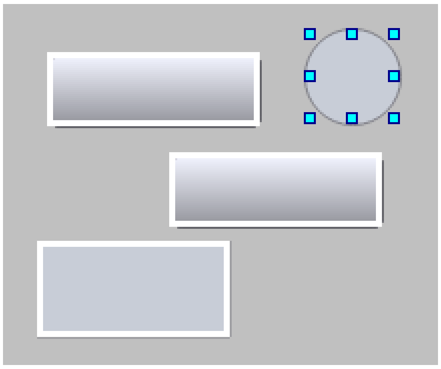
The selection frame surrounds all objects of a multiple selection. The selection frame is comparable with the rectangle that surrounds an individual object.

The selection frame is only visible as long as it is pulled up with the mouse button pressed. When you have made your multiple selection, the following frame is displayed:

- The reference object is indicated by the rectangle around it.
- The other selected objects are indicated by a frame.

Specifying a reference object

The reference object is the object upon which the other objects are oriented. The reference object is framed by a rectangle with handles. The following figure shows a reference object with three additional selected objects:



You have the following options to specify the reference object:

- Select the objects via multiple selection. The object selected first is then the reference object.
- Draw a selection frame around the objects. As a reference object, the object is automatically defined as on top in the foreground. If you wish to specify a different object within the selection as the reference object, click on the desired object. This action does not cancel your multiple selection.

Requirement

You have opened the work area containing at least two objects.

Selecting multiple objects with a selection frame

1. Position the mouse pointer in the work area close to one of the objects to be selected.
2. Hold down the mouse button, and draw a selection frame around the objects to be selected.

Selecting multiple objects using the <Shift> key

1. Hold down the <Shift> key.
2. Click the relevant objects, working in succession.
   All the selected objects are identified by frames.
   The object selected first is identified as reference object.

   Note

   To remove an object from the multiple selection, press <SHIFT>, hold it down and then click the relevant object once again.

Result

Multiple objects are selected. One of those is identified as the reference object. You can now perform the following steps:

- Moving all the objects in one group
- Aligning the objects to the reference object

1.1.6 Aligning objects (RT Uni)

Introduction

You can align the screen objects in the screen with reference to a reference object.

Aligning objects flush

The selected objects will be aligned flush to the reference object.

Icon	Description
	Aligns the selected objects to the left edge of the reference object.
	Aligns the selected objects to the vertical center axis of the reference object.
	Aligns the selected objects to the right edge of the reference object.
	Aligns the selected objects to the upper edge of the reference object.
	Aligns the selected objects to the horizontal center axis of the reference object.
	Aligns the selected objects to the lower edge of the reference object.
	Centers the selected objects to the center points of the reference object.
	Centers the selected objects vertically in the screen.
	Centers the selected objects horizontally in the screen.

Procedure

1. Select the objects via multiple selection.
2. Specify an object as the reference object.
3. Select the desired command for alignment in the toolbar or the shortcut menu.
   The selected objects are aligned.

1.1.7 Rotating an object around a pivot point (RT Uni)

Introduction

You define the rotation of an object around a pivot point. In the Inspector window of an object, specify the coordinates "X pivot point" and "Y pivot point". Specify the angle of rotation for the object under "Properties > Rotation".

The pivot point can be outside the object.

Pivot point

The pivot point can be defined as follows:

- Absolute from center: Sets the rotation to around the absolute center of the object.
- Absolute to screen: Sets the rotation to around the absolute zero point of the screen. The zero point is in the top left corner of the screen.

Rotation position

The attributes "X pivot point" and "Y pivot point" define the horizontal and vertical distance of the pivot point from the point of origin (center of the object or zero point of the screen).

The values are specified as a device-independent pixel (DIP).

The pivot point value can be outside the selection rectangle. This means that both negative values and positive values are possible.

Rotation

Defines the rotation of an object around the pivot point. Rotation is specified in degrees. The configured start point corresponds to a value of 0°. The position of an object deviates from its configured initial position by the rotation value. Both negative values and positive values are possible.

The object can also be placed outside the visible area. You can view objects outside the visible area by using the "Layout > Objects out of range" task card. In the inspector window, the position of an object can be modified under "Properties".

Example: Configuring rotation for a rectangle

1. Open the "Basic objects" palette in the "Toolbox" task card.
2. Drag the "Rectangle" object into the screen.
3. Click "Properties > Pivot point" in the Inspector window.
4. Select "Absolute from center" from the "Static value" column.
5. Enter a value of 45 for "Rotation".

The object is rotated clockwise by 45°.

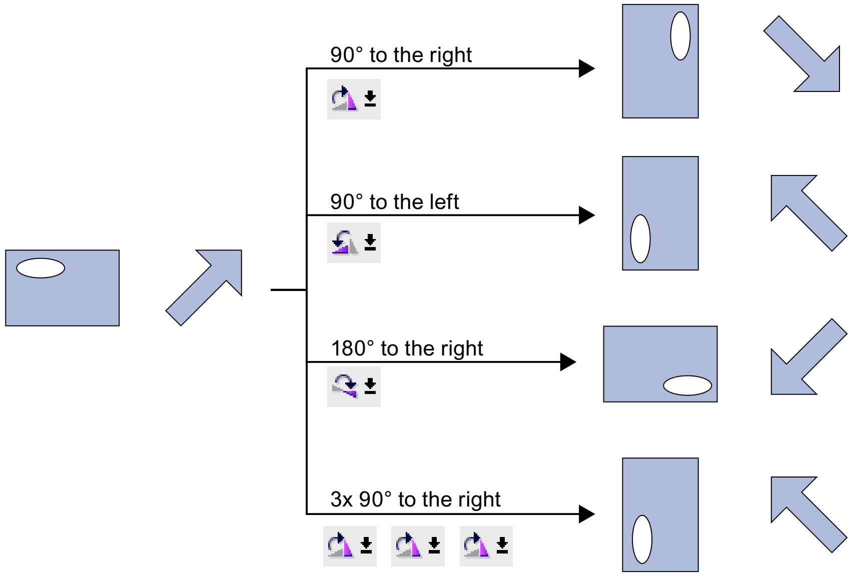
1.1.8 Rotating object (RT Uni)

Introduction

You can rotate a suitable object clockwise or counterclockwise around its center axis in steps of 90°.

You can also rotate multiple objects using the multiple selection function. Each object has its own reference point for the rotation and is rotated by its own reference point during multiple selection. Certain WinCC objects, such as Controls, cannot be rotated.

The alignment of elements in an object will change in a rotated object. The following figure shows how a rectangle and an ellipse behave under the different commands for rotating an object:



Requirement

You have opened the work area containing at least one object.

Procedure

1. Select the object that you want to rotate.
2. Click one of the following toolbar icons:
   ↻, to rotate the object clockwise around its center point. The angle of rotation is 90°.
   ↺, to rotate the object counterclockwise around its center point. The angle of rotation is 90°.
   ⟳, to rotate the object clockwise by 180°.

Alternatively, select from the shortcut menu the desired command to rotate the objects.

Result

The object is shown at its new angle.

1.1.9 Designing the fill pattern (RT Uni)

Introduction

WinCC lets you design the background color and the fill pattern of an object. The design options change in the Inspector window depending on object for which you are making the filling pattern.

For certain objects, you can not only define the color, but also a transparent background or a background with a color gradient.

Requirement

The object has been created and selected.

Designing the background color of an object

1. Click "Properties > Background color" in the Inspector window.
2. Select a color for the background of the object, for example, yellow.
   The object is filled with the selected color.

Designing the fill pattern of an object

1. Click "Properties > Background fill pattern" in the Inspector window.
2. To define a transparent background for the object, for example, select transparent.
   The object is shown as transparent.

Restriction for objects with events

Events for operator actions are only triggered if the operator action takes place in the marked, visible area of the object. If the fill pattern of an object is transparent, the user in Runtime must hit exactly the border of the object when tapping or clicking in order to trigger the events configured for the object. Select the border width so that the user can hit the border easily.

Note

Objects for which the "Opacity" property has the value "0" are also not visible in Runtime and do not trigger events.

1.1.10 Defining color gradients (RT Uni)

Introduction

For the objects in WinCC, color gradients can be specified as background for various surfaces.

Change the category in the Inspector window, depending on which surface you fill with a color gradient. The procedure remains the same.

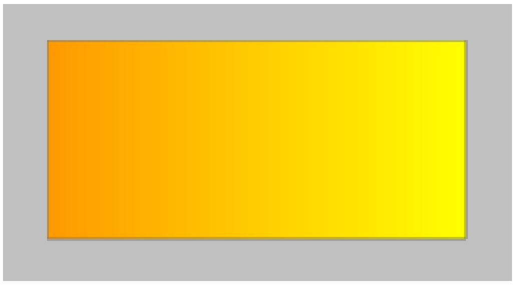
The following section describes how to configure the color gradient of a rectangle.

Horizontal color gradient with two colors

1. Select an object, for example a rectangle.
2. In the Inspector window, select "Horizontal gradient" under "Properties > Background fill pattern".
3. Go to "Properties > Fill direction" and select the direction in which the color is to run, for example "From left to right".
4. Select a background color for the horizontal color gradient, for example orange.
5. Select the second color for the gradient under "Alternative background color", for example yellow.

Result

The background of the rectangle is displayed with a color gradient of orange to yellow.



1.2 Advanced design (RT Uni)

1.2.1 Configuring toolbar and status bar (RT Uni)

Introduction

You operate the controls in runtime using the buttons in the toolbar. The status bar displays status messages from the control. During configuration, set the content of the toolbar and status bar.

Requirement

- The control is selected in the screen
- The Inspector window is open

Configuring the toolbar

1. In the Inspector window, configure the general properties of the toolbar, such as orientation and background color or displayed buttons, under "Properties > Toolbar".
2. In the Inspector window, enable the buttons you need in runtime under "Properties > Toolbar > Elements".
3. If required, configure the button display, for example background color, border and maximum and minimum size.
4. If necessary, select the authorization needed to operate the buttons in runtime.
5. If a button is not to be operated in Runtime, deselect "Allow operator control".
   You can reactivate a deactivated a button using a script in runtime, for example.

Configuring the status bar

1. In the Inspector window, configure the general properties of the status bar such as orientation and background color under "Properties > Status bar".
2. In the Inspector window, select the elements you need in runtime such as date and time under "Properties > Status bar > Elements".
3. You can adjust the display of an element in the status bar under "Properties > Status Bar > Elements" for the respective element.

1.2.2 Configuring flashing (RT Uni)

Introduction

You have the option of displaying the objects as flashing in runtime. You select the flashing frequency, the condition and the colors for the object.

You configure flashing characteristics for each color setting of an object that supports flashing.

Note

No change to the color value through flashing

If the property is displayed flashing in Runtime, this does not change the color value configured for the property.

You choose from the following conditions:

- "Never": Flashing is not active.
- "Always": The configured property always flashes in runtime.
- "Range violation": The property only flashes if the configured range has been exceeded.

The following options are available for the rate:

- "Slow"
- "Medium"
- "Fast"

Requirement

You have opened the work area containing at least one object.

Procedure

1. Select the object that is to flash, e.g. an I/O field.
   In the Inspector window under "Properties", select the property for which you want to define the flashing characteristics, for example "Background color".
2. Select "Flashing" in the "Dynamization" column.
   The "Flashing" page will appear.
3. Select the color and the alternative color for the states "On" and "Off".

   Note

   Flashing is only visible in runtime when there is a difference between the two colors.
4. Select the condition for the object flashing in runtime, for example "Always".
5. Select the flash rate, for example "Fast".

Result

In runtime, the object flashes in the selected colors and at the set rate.

1.2.3 Connecting tags and text lists in the text (RT Uni)

Introduction

You can connect tags or text lists with the following objects:

- Text field
- Button
- Check box

- List box
- Switch

You configure the connection of tags and text lists for each text property of an object that supports this.

Requirement

- The object has been created and selected

Connecting tags

1. In the Inspector window, select "Properties > Text".
2. Select "Tag" in the "Dynamization" column.
   The "Tag" page will open.
3. Select an existing tag with "Tag > Process > Tag" or create a new tag using the "Add" button.
4. Confirm your entries.

The name of the connected tag is displayed in the text field of the object.

Connecting the text list

1. In the Inspector window, select "Properties > Text".
2. Select "Resource list" in the "Dynamization" column.
   The "Resource list" page will open.
3. Select the text list from which a text entry is displayed.
4. Confirm your entries.

The reference to the text list entry is displayed in the text field of the object.

Note

The number of references to text list entries that in turn include references to text list entries or tags is limited to three layers.

1.2.4 External graphics (RT Uni)

Introduction

You can use graphics created with an external graphic program in WinCC. To use these graphics you store them in the project graphics of the WinCC project.

You can save graphics in the project graphics:

- When you drag-and-drop graphics objects from the "Graphics" pane into the work area, these are stored automatically in the project graphics. The graphic names are numbered in the order of their creation, for example, "Graphic_1." Use the <F2> function key to rename the graphic.
- As a graphic file with the following formats:
  *.bmp, *.ico, *.emf, *.wmf, *.gif, *.tif, *.png, *.svg, *.jpeg or *.jpg
- As an OLE object that is embedded in WinCC and is linked to an external graphic editor. In the case of an OLE link, you open the external graphic editor from WinCC. The linked object is edited using the graphic editor. An OLE link only works if the external graphic editor is installed on your PC, and supports OLE.

Use of graphics from the project graphics

Graphics from the project graphics are used in your screens:

- In a graphic view
- In a graphic list
- As labeling for a button

To use a graphic in the screen or in the screen object, drag-and-drop the desired graphic to the screen or the screen object. Alternatively, select the graphic from the drop-down list in the "Graphic" property in the Inspector window.

Transparent graphics

In WinCC you also use graphics with a transparent background. When a graphic with a transparent background is inserted into a graphic object of WinCC, the transparency is replaced by the background color specified for the graphic object. The selected background color is linked firmly with the graphic. If you use the graphic in another graphic object of WinCC, this object is displayed with the same background color as the graphic object that was configured first. If you want to use the graphic with different background colors, include this graphic in the project graphics again under a different name. The additional background color is configured when the graphic is used at the corresponding graphic object of WinCC.

Managing graphics

An extensive collection of graphics, icons and symbols is installed with WinCC. In the Toolbox window of the "Graphic" pane the graphic objects are structured by topic in the "WinCC graphics folder." The link to the WinCC graphics folder cannot be removed, edited or renamed.

The "Graphics" pane is also used to manage the external graphics. The following possibilities are available:

- Creating links to graphics folders
  The external graphic objects in this folder, and in the subfolders, are displayed in the toolbox and are thus integrated in the project.
- Editing folder links
- You open the program required for editing of the external graphic in WinCC.

Restrictions on SVG graphics

The SVG graphics are converted to Siemens SVG Standard. Note the following restrictions when using SVG graphics:

- The CSS definitions are converted to inline attributes.
- The embedded scripts and non-local URL links are not supported in the SVG graphics and are removed from the original graphics during conversion.
- The use of SVG graphics with embedded graphics and animations is not supported.
- The use of large SVG graphics affects performance due to the load associated with the increased characters.
- Migration of SVG graphics from WinCC V7 to TIA Portal is not supported.
- The following SVG characteristics are not supported:
  - Scripting - no Inline-JavaScript
  - Interactivity
  - Styling
  - Expandability - no ForeignObjects
  - Animations

The representation of SVG graphics depends on the browser used:

- The attributes "Width" and "Height" must be set for the correct display of an SVG graphic in the Firefox browser.

Note

Scaled SVG graphics in Chrome

Elements using an SVG graphic that was scaled in the engineering system as background graphic are not displayed correctly in Chrome in Runtime.

Editing SVG graphics

It is not possible to open SVG graphics in an external editor using the "Edit" command.

1.2.5 Managing external graphics (RT Uni)

Introduction

External graphics that you want to use in WinCC are managed in the "Screens" editor by using the "Tools" task card in the "Graphics" pane.

Requirement

- The "Screens" editor is open.
- The "Tools" task card is open.
- The graphics are available.
- The graphics have the following formats:
  *.bmp, *.ico, *.emf, *.wmf, *.gif, *.tif, *.svg, *.jpeg or *.jpg

Creating a folder link

1. Click "My graphics folder."
2. Select "Link" in the shortcut menu.
   The "Create link to folder" dialog is opened. The dialog suggests a name for the folder link.
3. Edit the name as required. Select the path containing the graphic objects.
4. Click "OK" to confirm your input.
   The new folder link is added to the "Graphics" object group. The external graphics that are located in the target folder and in sub-folders are displayed in the toolbox.

Editing folder links

1. Select the folder link to edit.
2. Select the "Edit link..." command from the shortcut menu.
   The "Create link to folder" dialog is opened.
3. Edit the name and path of the folder link as required.
4. Click "OK" to confirm your input.

Renaming the folder link

1. Select the folder link to rename.
2. Select "Rename" from the shortcut menu.
3. Assign a name to the new folder link.

Removing a folder link

1. Select the folder link you want to delete.
2. Select "Remove" in the shortcut menu.

Edit external graphics

1. Select the graphic you want to edit.
2. Select the "Edit graphic" command from the shortcut menu.
   This opens the screen editor associated with the graphic object file.

Editing graphics folders from WinCC

1. Select the graphic you want to edit.
2. Select "Open folder" from the shortcut menu.
   The Windows Explorer opens.

1.2.6 Defining the output format (RT Uni)

Introduction

You can customize the output format for the displayed values in many objects or define it yourself. The process value that is displayed in the object can be processed and output in different notations.

You define the output in the "Output format" property of a screen object for the following data:

- Binary
- Hexadecimal
- Decimal
- Text
- Floating-point numbers
- Duration, date, time
- Numerical values

The definition of the output format is based on UNICODE CLDR. You can find additional information on the CLDR project and on the definitions on the Internet at http://cldr.unicode.org/ (http://cldr.unicode.org/)

Defining the output format

The definition for a format pattern can be rewritten as a sequence of formatting codes. The formatting codes act as placeholders for a specific group of characters. For example, if a formatting code for which only the display of the numbers 0 to 9 is preset for a specific position in the display of an I/O field, no letters can be input at this position. The definitions for the output format are independent of the language. The output format can be language-specific and thus take linguistic differences into account, for example, for output of the date.

You can define and combine different format patterns yourself. The tables below show examples for the definition of output formats that are frequently used.

Binary data format

You use the "Binary" data format to display binary values. The "Binary" data format requires three inputs in the output format: The sign "B" followed by the optional number of digits and optional information on forming blocks.

Output format example	Mindigits	Block size	Tag value	Result
{B}	Default	-	16	1 0000
{B8}	8	-	16	0001 0000
{B8}	8	-	80	0101 0000
{B8,4}	8	4	80	0101 0000
{B,2}	Default	2	80	1 01 00 00
{B}	Default	-	-1	1111 1111

mindigits	Number of digits (optional)	Minimum: 1	Maximum: 64	Default value: 1
blocksize	Number of digits in front of the separator (optional)	Minimum: 0 (none)	Maximum: 8	Default value: 4

Hexadecimal data format

You use the "Hexadecimal" data format to display hexadecimal values. The "Hexadecimal" data format requires three inputs in the output format: The sign "H" followed by the optional number of digits and optional information on forming blocks.

Output format example	Mindigits	Block size	Tag value	Result
{H}	Default	-	1	1
{H}	Default	-	15	F
{H}	Default	-	45054	AFFE
{H4,2}	4	2	45054	AF FE
{H,2}	Default	2	45054	AF FE

mindigits	Minimum number of digits (optional)	Minimum: 1	Maximum: 16	Default value: 1
blocksize	Number of digits in front of the separator (optional)	Minimum: 0 (none)	Maximum: 8	Default value: 4

Integer data format

You use the "Integer" data format to display decimal values. The "Integer" data format requires three inputs in the output format: The sign "I" followed by the number of digits. A plus or minus symbol can be placed in front of the sign.

Output format example	Mindigits	Tag value	Result
{I}	Default	9	9
{I4}	4	9	0009
{I2}	2	123	123
{I}	Default	1.6	1
{+I}	Default	1	+1
{I1}	1	123456789	123456789

+/-	Sign (optional)			Default value: None
mindigits	Minimum number of digits (optional)	Minimum: 1	Maximum: 16	Default value: 1

String data format

You use the "String" format to display texts. The "STRING" data format requires three inputs in the output format: The sign "S" followed by the optional number of characters and optional formatting parameter.

Output format example	Maxchars	String format	Tag value	Result
{S}	Default	-	Motor	Motor
{S4}	4	-	Motor	Moto
{S,trim}	Default	trim	Motor	Motor
{S,upper}	Default	upper	Motor	MOTOR
{S,lower}	Default	lower	Motor	motor
{S,trim,upper}	Default	trim, upper	Motor	MOTOR
{S3,trim,upper}	3	trim, upper	Motor	MOT

maxchars	Number of characters (optional)	Minimum: 1	Maximum: 99	Default value: Complete input
stringformat	Parameter for formatting of the input (optional)	trim: Outputs the input string without spaces upper: Outputs the input string in uppercase letters lower: Outputs the input string in lowercase letters		

"Duration" data format

The accuracy of the duration inputs is limited to 1 ms. All inputs of less than 1 ms are shown as 0 in runtime.

To display fractions of a second, use .S, .SS or .SSS according to the pattern for the duration.

The "Duration" data format requires three inputs in the output format: The sign "P" followed by the optional units of time.

Output format example	Tag value (ns)	Result
{P,s}	10000000	1
{P,s}	-10000000	-1
{P,s}	10000000	01
{P,m:ss}	35990000000	59:59
{P,h:mm:ss}	36000000000	1:00:00
{P,hh:mm:ss}	36000000000	01:00:00
{P,D hh:mm:ss}	864000000000	1D 00:00:00
{P,DD hh:mm:ss}	864000000000	01D 00:00:00
{P,s.S}	10000	0.0
{P,s.SS}	10000	0.00
{P,s.SSS}	10000	0.001
{P,s.SSS}	9999	0.000

durationunit	Duration

The output format {P} enables automatic mode. In the mode the result with the smallest necessary unit of time is written. The table below shows some examples for tag values and their output in automatic mode.

Tag value (ns)	Result	Meaning
9999	0	0.9999 ms
10000	0.001	1ms
9990000	0.999	999ms
10000000	1	1s
10010000	1.001	1s 1ms
600000000	1:00	1m
700000000	1:10	1m 10s
35999990000	59:59.999	59m 59s 999ms
36000010000	1:00:00.001	1h 1ms
863999990000	23:59:59.999	23h 59m 59s 999ms
937845670000	1D 02:03:04.567	1D 2h 3m 4s 567ms
86400000000000	100D	100D

Localized output

Some data formats can be localized. In this case the output format depends on the system language. The table below shows the examples for the output formats for "German".

Data format	Permitted values and default values	Output format example	Tag value	Result
Floating-point numbers: {FAccuracy}	Accuracy: from 1 to 9 Default value: 2	{F}	123.456	123.46
		{F3}	123.123	123,123
		{+F}	123.1	+123.10
Number: {NAccuracy}	Accuracy: from 0 to 9 Default value: 2	{N}	123	123.00
		{+N}	123	+123.00
		{N1}	123	123.0
{#|0|,|.}	User-defined numerical values to CLDR	{#,##0.###}	1234.567	1,234.567
		{#,##0.##}	1234.123	1234.12
		{#,###.#}	1234.123	1,234.1
Exponential: {EAccuracy}	Accuracy: from 0 to 9 Default value: 2	{E}	1123	1.12E+3
		{E1}	1123	1.1E+3
		{E3}	1123	1.123E+3
		{+E}	1123	+1.12E+3
		{E0}	1123	1E+3
Date: {D,Length}	Length: short medium or code according to CLDR format with the sign @ Default value: short	{D}	2019-06-15T13:45:30	15.06.19
		{D,medium}	2019-06-15T13:45:30	15.06.2019
		{D,@y}	2019-06-15T13:45:30	2019
		{D,@EEEE, MMMM d, yy}	1996-07-10	Sa., July 10, 96
Time: {T,Time format}	Time format: short medium medium.S medium.SS medium.SSS or code according to CLDR format with the sign @ Default value: medium	{T}	2019-06-15T13:45:30	1:45:30
		{T,short}	2019-06-15T13:45:30	1:45
		{T,@h:mm a}	09:08:15	9:08 AM
		{T,@hh:mm a}	09:08:15	09:08 AM
		{T,@HH:mm:ss}	09:08:15	19:08:15
		{T,@HH:mm:ss.SSS}	09:08:15.1230	19:08:15.123
		{T,medium.SSS}	09:08:15.1239	19:08:15.123

Combined output

You have the option of combining output formats.

Description	Output format example	Tag value	Result
Date and time	{D} – {T}	2019-06-15T13:45:30	15.06.2019 – 1:45:30 PM
	{D} {T}	2019-06-15T13:45:30	15.06.2019 1:45:30 PM

Description	Output format example	Tag value	Result
Date and time with line break	{D}\n{T}	2019-06-15T13:45:30	15.06.2019 1:45:30 PM
Two numerical values	hex {H} – dec {I}	45054	hex AFFE – dec 45054
Text with prefix	myMotor {S}	motor34	myMotor motor34

See also

Configuring an alarm control (Page 253)

http://cldr.unicode.org/ (http://cldr.unicode.org/)

1.2.7 Example: Configuring a rectangle (RT Uni)

Task

In this example you configure a rectangle:

- Color = red
- Black frame 2 pixels wide
- Position = (20, 20)
- Size = (100,100)

Changing the color of the rectangle

To change the color of the rectangle:

1. Select the rectangle.
2. Specify the background color in the Inspector window "Properties > Background color".
3. Select the "Solid" option under "Background fill pattern".
4. Define the color for the border in the Inspector window.
5. Enter the value "2" for "frame width".

Interim result

The rectangle is red and has a black frame with a width of two pixels.

Repositioning and resizing the rectangle

To change the position and size of the rectangle:

1. Select the rectangle.
2. In each case set the value "20" for the positions "X position" and "Y position" in the Inspector window.
3. Set "100" for the height and for the width.

Result

The rectangle is positioned at the coordinates (20, 20), and has a width and height of 100 pixels.

1.2.8 Example: Configuring an I/O field (RT Uni)

Task

In this example, you configure an I/O field and connect it to a tag:

- Color = blue
- Border color = gray
- Mode = Input/output
- HMI tag = MyTag

Configuring an I/O field

1. Open the "Elements" palette in the "Toolbox" task card.
2. Drag the "I/O field" object to the screen.
3. In the Inspector window, select "Properties > Background color".
4. Select another color, for example blue, from the "Static value" column.
5. Select another border color, for example gray.
6. Select "Input/output" mode under "Properties > Mode".

Connecting the I/O field to a tag

1. In the Inspector window, click "Properties > Process value" in the "Dynamization" column.
2. Select the entry "Tag" from the list.
   The "Tag" page will open.
3. Click on the selection button under "Tag > Process > Tag".

4. Select the required tag "MyTag" from the list.
   Confirm your selection.
5. Go to "Properties > Reaction to input" and set how the values are to be handled in runtime, for example "Accept value after exit".

Result

The I/O field is now configured in accordance with the settings and connected to the tag. In runtime, you see the current value of the tag in the I/O field and can also enter the value for the tag. The value is applied to the tag.

1.2.9 Example: Set values (RT Uni)

Introduction

You want to display specific process values in a screen in runtime, enter values yourself and correct them, if necessary.

In the steps below you visualize the display of the velocity actual value of a motor and regulate the velocity yourself.

- You configure 2 I/O fields for input and output of the process values.
- You configure text fields for a clearly structured display in the screen.
- You configure a slider to display and adjust the values.

Requirement

- A project is open.
- A screen has been configured.
- The tags "SetValue" and "ActualValue" have been created as process values for the motor speed.

Display and input of values

With an I/O field you are displaying the current value of the tags in your screen and, if necessary, enter the values for the process yourself.

1. Add the "I/O field" object to the screen from the "Toolbox" task card.
2. Go to "Properties" and set the required height, width and position of the object.
3. Under "Properties > Mode" specify the "Input/output" mode in the "Static value" column.
4. Click under "Properties > Process value".
5. In the "Dynamization" column, select the entry "Tag".
   The "Tag" dialog opens on the right in the Inspector window.

6. Under "Tag" specify the "SetValue" tag whose values you want to display and change in runtime.
7. Specify an additional I/O field for the "ActualValue" tag in the "Output" mode.
8. Configure two text fields "Actual value" and "Set value" for labeling the I/O fields.

Tips for effective procedure

- You can also create a new I/O field by dragging a configured tag from the Details view to an HMI screen.
  An I/O field is created automatically and connected to the desired tag.

Adjusting values

You use a slider to intervene in the process and correct the displayed process value.

1. Add the "Slider" object to the screen from the "Toolbox" task card.
2. Go to "Properties" and set the required height, width and position of the object.
3. Under "Properties > Process value indicator mode" specify the "Detailed indicator" mode in the "Static value" column.
4. Click under "Properties > Process value".
5. In the "Dynamization" column, select the entry "Tag".
   The "Tag" dialog opens on the right in the Inspector window.
6. Under "Tag" specify the "SetValue" tag whose values you want to display and change in runtime.

Result

In runtime the motor speed is displayed in the I/O field. You can transfer the speed to the motor in the "Set value" I/O field.

Using the slider, you can read the actual velocity and control the speed yourself by moving the slider.

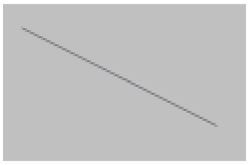
1.3 Configuring objects (RT Uni)

1.3.1 Basic objects (RT Uni)

1.3.1.1 Line (RT Uni)

Application

The "Line" object is an open object. The line length and gradient slope are defined by the height and width of the rectangle enclosing the object.



Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "Dash type"
- "Line start" and "Line end":

Dash type

You specify the line display under "Properties > Dash type" in the Inspector window. The line is shown without interruption if you select "Solid", for example.

Note

The dash types available depend on the HMI device selected.

Line start and end

You define the start and end points of the line under "Properties" in the Inspector window.

Use arrow points, for example, as the start and end points of the line. The available start and end points depend on the device.

1.3.1.2 Polyline (RT Uni)

Application

The "Polyline" is an open object. Use the "Polygon" object if you want to fill the object with color.

Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "Line start" and "Line end": Specifies the type of line start and line end.
- "Points": Modifies, deletes or adds corners.

Line start and end

Define the start and end of the line in the "Properties" Inspector window. Use arrow point, for example, as start and end point. The available start and end points depend on the device.

Points

The corner points are numbered in the order of their creation. You can change, delete, or add more corner points:

1. Select "Properties > Points" in the Inspector window.
2. Select the required corner point. Enter a value for "X coordinate" and "Y coordinate".
3. Click on the selection button in the "Static value" column to add or delete a corner point. A dialog opens.
4. Use the "Add" command to create a new point. You can delete corner points by selecting the corresponding row in the dialog and selecting "Delete" from the shortcut menu for the row.

Configuring rotation in runtime

You configure the "Polyline" object so that it rotates about a reference point in runtime.

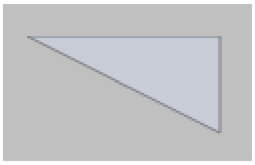
Enter the values for rotation in degrees:

1. Select "Properties" in the Inspector window.
2. Enter the required values for the following attributes:
   - Pivot point
   - Rotation
   - X pivot point
   - Y pivot point

1.3.1.3 Polygon (RT Uni)

Application

The "Polygon" is a closed object which you can fill with a background color.



Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. In particular, you can customize the following property:

- "Points": Modifies, deletes or adds corners.

Points

The corner points are numbered in the order of their creation. You can change, delete, or add more corner points:

1. In the Inspector window, select "Properties > Points".
2. Select the required corner point. Enter a value for "X coordinate" and "Y coordinate".
3. Click on the selection button in the "Static value" column to add or delete a corner point. A dialog opens.
4. Use the "Add" command to create a new point. You can delete corner points by selecting the corresponding row in the dialog and selecting "Delete" from the shortcut menu for the row.

Configuring rotation in runtime

You configure the "Polygon" object so that it rotates about a reference point in runtime.

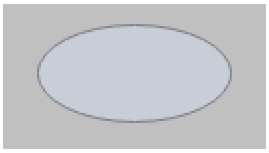
Enter the values for rotation in degrees.

1. In the Inspector window, select "Properties".
2. Enter values for the following attributes in the "Rotation" area.
   - Pivot point
   - Rotation
   - X pivot point
   - Y pivot point

1.3.1.4 Ellipse (RT Uni)

Application

The "Ellipse" is an enclosed object that can be filled with a color or pattern.



Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "X radius": Specifies the horizontal radius of the elliptical object.
- "Y radius": Specifies the vertical radius of the elliptical object.

X radius

The horizontal radius of the "Ellipse" object is specified in the Inspector window. The value is entered in pixels.

1. Click "Properties" in the Inspector window.
2. Enter a value of between 0 and 2500 under "X radius."

Y radius

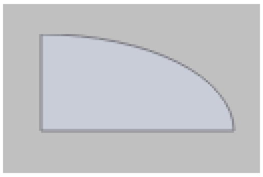
The vertical radius of the "Ellipse" object is specified in the Inspector window. The value is entered in pixels.

1. Click "Properties" in the Inspector window.
2. Enter a value of between 0 and 2500 for "Y radius".

1.3.1.5 Ellipse segment (RT Uni)

Use

The "Ellipse segment" is a closed object that you can fill with a color or pattern. By default, an ellipse segment is a quarter ellipse. It can be customized as required.



Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "X radius" and "Y radius": Specifies the horizontal and vertical radius of the elliptical object.
- "Angle start" and "Angle range": Specify where the start and end point lie on a virtual circle of 360°.

Defining the radius

Define the horizontal and vertical radius of the "Ellipse segment" object in the Inspector window. Enter the values using Pixels as the unit:

1. Click "Properties" in the Inspector window.
2. Enter one value each for "X radius" and "Y radius".

Defining the start angle and angle range

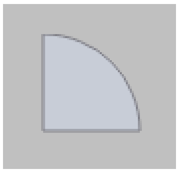
Set the size of the ellipse segment using the "Start angle" and "Angle range" attributes. Enter the values using Degrees as the unit:

1. Click "Properties" in the Inspector window.
2. Enter one value each for "Start angle" and "Angle range".

1.3.1.6 Circle segment (RT Uni)

Use

The "Circle segment" is a closed object that you can fill with a color or pattern. By default, a circle segment is a quarter circle. It can be customized as required.



Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "Radius": Define the size of the circle segment.
- "Angle start" and "Angle range": Specify where the start and end angle lie on a virtual circle of 360°.

Radius

You define the radius of the "Circle segment" object in the Inspector window. Enter the value using Pixels as the unit.

1. Click "Properties" in the Inspector window.
2. Enter a value for "Radius".

Defining the start angle and angle range

Set the size of the circle segment using the "Start angle" and "Angle range" attributes. Enter the values using Degrees as the unit.

1. Click "Properties" in the Inspector window.
2. Enter one value each for "Start angle" and "Angle range".

1.3.1.7 Elliptical arc (RT Uni)

Use

The "Elliptical arc" is an open object. Use the "Ellipse segment" object if you want to fill the object with color. By default, an elliptical arc is a quarter ellipse. It can be customized as required.

Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "X radius" and "Y radius": Specifies the horizontal and vertical radius of the elliptical object.
- "Angle start" and "Angle range": Specify where the start and end point lie on a virtual circle of 360°.

Defining the radius

Define the horizontal and vertical radius of the "Elliptical arc" object in the Inspector window. Enter the values using Pixels as the unit.

1. Click "Properties" in the Inspector window.
2. Enter one value each for "X radius" and "Y radius".

Defining the start angle and angle range

Set the length of the elliptical arc using the "Start angle" and "Angle range" attributes. Enter the values using Degrees as the unit.

1. Click "Properties" in the Inspector window.
2. Enter one value each for "Start angle" and "Angle range".

1.3.1.8 Circular arc (RT Uni)

Use

The "Circular arc" is an open object. Use the "Circle segment" object if you want to fill the object with color. By default, a circular arc is a quarter circle. It can be customized as required.

Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "Radius": Define the size of the circular arc.
- "Angle start" and "Angle range": Specify where the start and end angle lie on a virtual circle of 360°.

Defining the radius

You define the radius of the "Circular arc" object in the Inspector window. Enter the value in pixels.

1. Click "Properties" in the Inspector window.
2. Enter a value for "Radius".

Defining the start angle and angle range

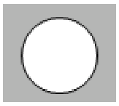
You set the length of the elliptical arc using the "Start angle" and "Angle range" attributes. Enter the values using Degrees as the unit.

1. Click "Properties" in the Inspector window.
2. Enter one value each for "Start angle" and "Angle range".

1.3.1.9 Circle (RT Uni)

Application

The "Circle" object is a closed object which can be filled with a color or pattern.



Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "Radius": Specifies the size of the circle.

Radius

The radius of the "Circle" object is specified in the Inspector window. The value is entered in pixels.

1. Click "Properties" in the Inspector window.
2. Enter a value of between 0 and 2500 for "Radius".

1.3.1.10 Rectangle (RT Uni)

Application

The "Rectangle" is a closed object which you can fill with a color.

Layout

In the Inspector window you can customize the settings for the position, geometry and color of the object. You can adapt the following properties in particular:

- "Corner radius": Specifies the horizontal and vertical distance between the corner of the rectangle and the start point of a rounded corner.

Corner radius

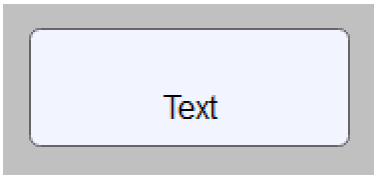
The corners of the "Rectangle" object can be rounded to suit your requirements. If the "Radius" property for all four corners is set to 0, a standard rectangle without rounded corners is displayed.

1. Select "Properties > Corners" in the Inspector window.
2. Enter the radius for each corner.

1.3.1.11 Text box (RT Uni)

Use

The "Text box" is a closed object which you can fill with a color.



Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can adapt the following properties in particular:

- "Text": Specifies the text for the text box.
- "Text trimming": Specifies whether ellipsis is to be displayed after a line break for a long text.
- "Text break": Specifies whether the next word is to be automatically moved to the next row for a long text.

Text

Specify the text for the text box in the Inspector window.

1. Click "Properties" in the Inspector window.
2. Enter a text.

Trimming text

You can specify ellipsis for a text that cannot be displayed in full in a text box.

1. Click "Properties > Text trimming" in the Inspector window.
2. Select the option "With character ellipsis" in the "Static value" column.

The text displayed is truncated with ellipsis.

Enabling line breaks

You can enable line breaks for a text that cannot be displayed in full in a text box. If you find that the text box is large enough for display with a line break:

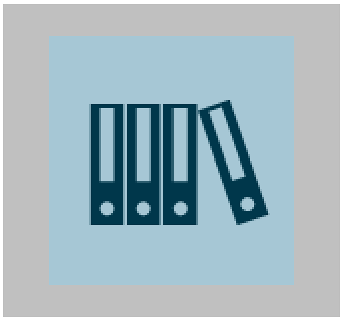
1. Click "Properties > Text break" in the Inspector window.
2. Select the option "Word wrap" in the "Static value" column.

The text is displayed in full with line breaks.

1.3.1.12 Graphic view (RT Uni)

Application

The "Graphic view" object is used to display graphics.



Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "General > Graphic": Specifies the graphic file that is displayed in the object.
- "Format > Scale background graphic": Specifies how the graphic is scaled.

Scale background graphic

The following modes for scaling graphics are available:

- None
  The graphic is inserted centered into the graphic view. If the graphic is larger than the graphic view, the graphic is displayed incompletely.
- Fill
  The graphic fills the graphic view. This mode can lead to a distortion of the graphic.
- Uniform
  The graphic is fully displayed and without distortion in the graphic view.
- Stretch to fit
  The graphic is adjusted to the size of the graphic view without distortion. This may cause the graphic to be displayed incompletely.
- Tiled
  The graphic is displayed in original size, multi-tiled until the graphic view is filled.

To select a mode for scaling the graphic, proceed as follows:

1. Click "Format > Scale background graphic" in the Inspector window.
2. Select the desired mode in the "Static value" column.

Inserting graphics

Use the following image formats in the "Graphic Display" object: *.bmp, *.tif, *.png, *.ico, *.emf, *.wmf, *.gif, *.svg, *.jpg or *.jpeg. You can also use graphics as OLE objects in the Graphic view .

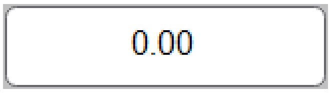
1. In the Inspector window, click "General > Graphic".
2. Select the graphic that you wish to insert.
   The graphic preview is shown in the right pane.
   You have the option to create the graphic from a file using the button, or to create a new graphic from an OLE object using the button.
3. Click "Apply" to insert the graphic in the Graphic view .

1.3.2 Elements (RT Uni)

1.3.2.1 I/O field (RT Uni)

Application

The "I/O field" object is used to enter and display process values.



Layout

In the Inspector window, you customize the position, shape, style, color and font types of the object. You can adapt the following properties in particular:

- "Mode": Specifies whether the values and entered and displayed in runtime or displayed only.
- "Behavior during input": Specifies the response of the object in runtime.
- "Hidden input": Specifies whether the input value is displayed normally or encrypted during input.

Note

Reports

In reports, I/O fields only output data. "Output" mode is preset. Properties for configuring input are not available, for example "Hidden input".

Mode

The response of the I/O field is specified in the Inspector window under "Properties".

Mode	Description
"Input/output"	Values can be input and output in the I/O field in runtime.
"Output"	The I/O field is used for the output of values only.

Hidden input

In runtime, the input can be displayed normally or encrypted, for example for hidden input of a password. A "*" is displayed for each character in hidden input. The data format of the value entered cannot be recognized.

1. In the Inspector window, select "Properties > Reaction to input".
2. Select "Hidden input".

1.3.2.2 Button (RT Uni)

Application

The "Button" object allows you to configure an object that the operator can use in runtime to execute any configurable function.

Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can adapt the following properties in particular:

- "Format > Content > Content mode": Defines the graphic representation of the object.
- "Format > Content > Scale background graphic": Specifies how the graphic is scaled.

Content mode

The button display is specified under "Format > Content > Content mode" in the Inspector window.

Content mode	Description
"Text"	The button is displayed with text. This text explains the function of the button.
"Graphic"	The button is displayed with a graphic. This graphics represents the function of the button.
"Graphics or text"	The button is displayed with text or graphics. If the graphics cannot be displayed, the corresponding text is displayed.
"Graphics and text"	The button is displayed with text and graphics.

Different options are available depending on the device.

Scale background graphic

The following modes for scaling graphics in buttons are available:

- None
  The graphic is inserted centered inserted into the button. If the graphic is larger than the button, the graphic will be displayed incompletely.
- Fill
  The graphic fills the button. This mode can lead to a distortion of the graphic.
- Uniform
  The graphic is fully displayed and without distortion in the button.
- Stretch to fit
  The graphic is adjusted to the size of the button without distortion. This may cause the graphic to be displayed incompletely.

To select a mode for scaling the graphic, proceed as follows:

1. In the Inspector window, click "Format > Content > Scale background graphic".
2. Select the desired mode in the "Static value" column.

Text / Graphic

The "Content mode" property settings are used to define whether the display is static or dynamic. The display is defined under "Properties > Text" or "Graphic" in the Inspector window.

You can, for example, select the following options for the "Graphic" or "Text" type.

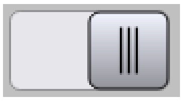
Type	Description
"Graphic"	Use "Graphic with pressed button" to specify a graphic displayed in the button in the "ON" state.
"Text"	Use "Text with pressed button" to specify the text displayed in the button for the "ON" state.

1.3.2.3 Switch (RT Uni)

Application

The "Switch" object is used to configure a switch that is used to switch between two predefined states in runtime. The current state of the "Switch" object can be visualized with either a label or a graphic.

The following figure shows a "Switch" type switch.



Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. In particular, you can customize the following property:

- "Content mode" Defines the graphic representation of the object.

Type

Switch display is specified under "Properties > Content > Content mode" in the Inspector window.

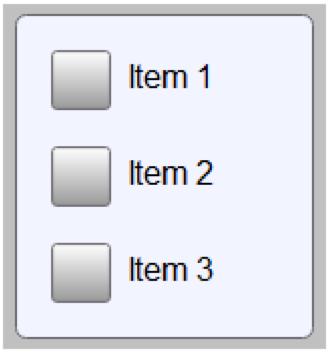
Type	Description
"Graphic"	The current state of the switch is shown with a graphic. In runtime click on the button to actuate the switch.
"Text"	The current state of the switch is shown with a label. In runtime click on the button to actuate the switch.
"Graphics or text"	The switch displays graphics or a text. If the graphics are not available, the text is displayed.
"Graphics and text"	The switch displays graphics and a text.

1.3.2.4 Check box (RT Uni)

Application

You use the "Check box" object to display and select multiple entries. You activate a selection item by default so that the operator only changes the preset value if necessary. The operator can select several options in runtime. You can specify a text or a graphic for each option.

To integrate the check box into the process, dynamize the corresponding properties.



Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can adapt the following properties in particular:

- "Number of check boxes": Defines the number of options.
- "Selection of check boxes": Defines which options are displayed as activated by default.

Note

Default standard height

The item height option of the radio button is set to "0" during the creation of a new object. This value does not represent the actual value 0, but a default setting.

Specify number of check boxes

1. Click "Properties > Selection items" in the Inspector window.
2. Click on the selection button in the "Static value" column.
   A dialog opens.
3. Specify the desired number of check boxes with "Add".
   To delete entries, click in the corresponding line and press the <DEL> key.

Using graphics and texts in the selection items

You can mark the selection items with texts or graphics. The following modes are available:

- "Graphic and text": The selection item shows text and graphic.
- "Graphic or text" The selection item is visualized either by a graphic or a text. If the graphic is not available, the text is displayed.
- "Graphic": The selection item is visualized with a graphic.
- "Text": The selection item is visualized with an inscription.

To configure the CheckBox contents, follow these steps:

1. Under "Properties > Content > Content mode" select the mode for display of the selection items, e.g. "Graphic and text".
2. Under "Selection items > Selection item [N] > Text" enter the text that is to be shown in the check box as selection item.
3. Under "Selection items > Selection item [N] > Graphic" open the drop-down list and select the corresponding graphic.

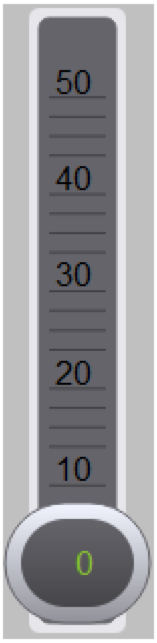
Specify default of the check box

Use the "Select item" property of a selection item to define whether it is to be shown as enabled in a check box list. You can activate multiple options.

1.3.2.5 Bar (RT Uni)

Use

The tags are displayed graphically using the "Bar" object. The bar graph can be labeled with a scale of values.



Layout

In the Inspector window, you customize the settings for the position, shape, style, color, and font types of the object. You can adapt the following properties in particular:

- "Show trend indicator": Shows whether the current value is higher or lower than the previous value.
- "Bar mode": Defines the gradations on the bar scale.
- "Linear scale": Specifies the properties for the bar scale.
- "Process value indicator mode": Specifies how the process value is displayed in the bar chart.

Bar mode

You define how the color change is represented in "Properties > Bar mode" in the Inspector window.

Color transition	Description
"Segmented"	The bar changes to the predefined color segment by segment from the start value to the process value. With segment by segment representation, you visualize, for example, which limits are exceeded by the displayed value.
"Unicolor"	The entire bar changes to the predefined color from the start value to the process value.
"Segmented static"	The bar background color changes segment by segment from the minimum scale value to the maximum scale value. With segment by segment representation, you visualize, for example, which limits are exceeded by the displayed value.
"Unicolor static"	The bar background changes to a predefined color from the minimum scale value to the maximum scale value.

Displaying the process value indicator

You use the property "Process value indicator mode" to select the process value of the selected tag in the bar in runtime:

1. In the Inspector window, select "Properties > Process value indicator mode".
2. Select another "Indicator" mode in the "Static value" column.
3. Go to "Foreground color process value indicator" and select the display color for the process value.

Define bar segments

You define the settings for the bar scale under "Properties > Linear scale":

- "Scaling type": Specifies how the bar scale is calculated, for example "Linear".
- "Alignment": Specifies whether the bar is displayed horizontally or vertically.
- "Scale mode": Specifies whether the scale is subdivided with ticks or labels or not at all.
- "Maximum scale value" and "Minimum scale value": Specifies the start and end value displayed on the scale.

Defining scale gradation

Use the "Division count" property to define the subdivision count for the bar scale divisions.

The "Subdivision count" property defines the number of ticks between the division marks.

1. Click "Properties > Linear scale" in the Inspector window.
2. Enable "Show scale".
3. Enter the required values for the "Division count" and "Subdivision count".

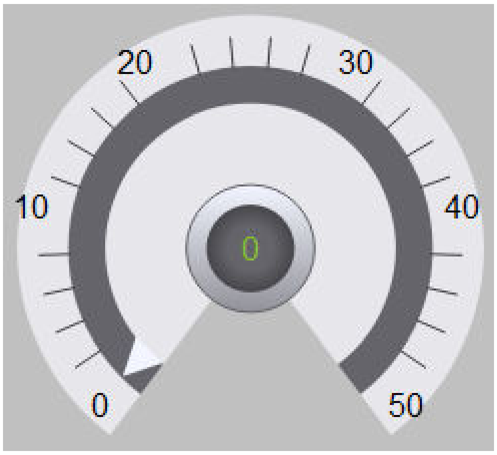
Note

The division count can only be changed if "Automatic scaling" is disabled.

1.3.2.6 Gauge (RT Uni)

Use

The "Gauge" object shows numeric values in the form of an analog gauge. For example, a glance in runtime is enough to note that the boiler pressure is in the normal range. The gauge is for display only and cannot be controlled by the operator.



Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can adapt the following properties in particular:

- "Peak indicator": Specifies whether the measurement range is indicated with a peak indicator.
- "Maximum scale value" and "Minimum scale value": Specifies the top and bottom values of the scale.
- "Color of normal range": Specifies the color in which the normal range is displayed.
- "Scale": Specifies various settings for the scale view.

Display peak value

The "Peak indicator" property can be used to enable a marker function for the maximum or minimum pointer movement in runtime.

1. Click "Properties > Peak indicator" in the Inspector window.
2. Select the option "High" or "Low" in the "Static value" column.

Maximum and minimum scale value

You can set the top and bottom end values of the scale in the Inspector window.

1. Click "Properties > Scale" in the Inspector window.
2. Enter a number at "Maximum scale value" and "Minimum scale value".
   If you select a tag as the end value of the scale, the number will be no longer available.

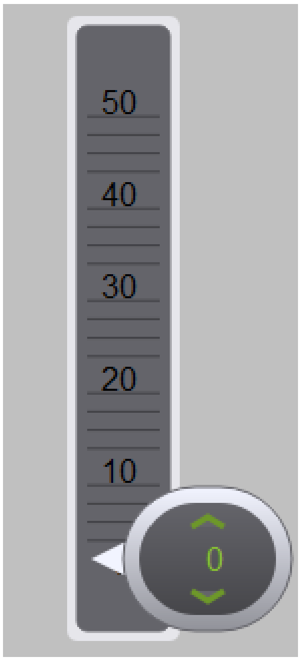
Configuring a scale

1. Click "Properties > Scale" in the Inspector window.
2. Under "Start angle", specify the angle at which the scale is to start. The angle is specified in degrees, starting at the zero position.
   The scale runs clockwise. A starting value of 0 corresponds to a display of 3 o'clock.
3. Under "Angle range", specify the range in degrees to be covered by the scale.
4. Under "Scale mode", specify whether the divisions are displayed as ticks or numbers.

1.3.2.7 Slider (RT Uni)

Use

Process values are monitored and adapted within a defined range with the "Slider" object. The monitored range is visualized in the form of a slider. By adjusting the slider, you intervene in the process and correct the displayed process value.



Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can, in particular, adapt the following properties as required:

- "Maximum scale value" and "Minimum scale value": Specifies the top and bottom values of the scale.
- "Process value indicator mode": Specifies how the current process value is displayed in the slider.
- "Show trend indicator": Specifies how the current value has changed compared to the previous values.

Maximum and minimum scale value

The top and bottom end values of the scale are specified in the Inspector window.

1. Click "Properties > Linear scale" in the Inspector window.
2. Enter a number at "Maximum scale value" and "Minimum scale value". If you select a tag as the end value of the scale, the number will be no longer available.

Show value

Specify that the value of the current position is displayed below the slider in the Inspector window.

1. Click "Properties" in the Inspector window.
2. Select "Show value".

Process value indicator mode

Specify a mode for process value display:

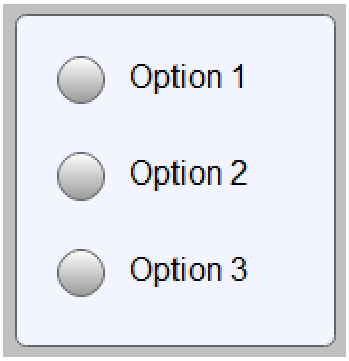
Mode	Description
Bar	Displays the bar with the process value indicator.
Indicator	Shows the process value indicator as a position on the bar.
Detailed indicator	Shows the process indicator in the bar.
Bar with detailed indicator	Shows the current process value and its position in the slider bar.

1. In the Inspector window, activate "Properties > Properties > Layout".
2. Deactivate "Display bar".

1.3.2.8 Radio button (RT Uni)

Application

You use the "Option buttons" object to display and select various options. Only one of these options can be selected by the operator. Enable one of the options by default so that the operator only changes the default value if necessary. To incorporate an option button into the process, dynamize the corresponding attribute.



Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "Selection items": Defines the options.
- "Select item": Specifies which entries are displayed as activated.

Note

Default standard height

The item height option of the radio button is set to "0" during the creation of a new object. This value does not represent the actual value 0, but a default setting.

Defining the number of entries

You specify the number of entries in the Inspector window:

1. Click "Properties > Selection items" in the Inspector window.
2. Click on the selection button in the "Static value" column.
   A dialog opens.
3. Specify the required number of entries.

Using graphics and texts in the selection items

You can mark the selection items with texts or graphics. The following modes are available:

- "Graphic and text": The selection item shows text and graphic.
- "Graphic or text" The selection item is visualized either by a graphic or a text. If the graphic is not available, the text is displayed.
- "Graphic": The selection item is visualized with a graphic.
- "Text": The selection item is visualized with an inscription.

To configure the check box contents, follow these steps:

1. Under "Properties > Content > Content mode" select the mode for display of the selection items, e.g. "Graphic and text".
2. Under "Selection items > Selection item [N] > Text" enter the text that is to be shown in the check box as selection item.
3. Under "Selection items > Selection item [N] > Graphic" open the drop-down list and select the corresponding graphic.

Specifying the default setting of the option buttons

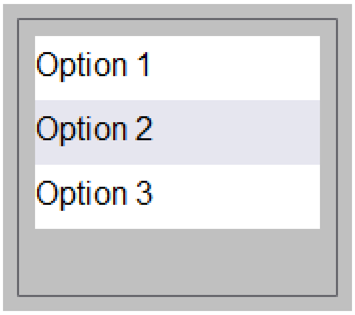
Use the "Select item" property to specify which option button entry is to be shown as enabled. Go to "Properties > Selection items" and enable the property "Select item" for an entry to make its default status enabled. Only one entry can be enabled at any one time.

1.3.2.9 List box (RT Uni)

Application

You use the "List box" object to present and select multiple list entries. You activate list entries by default so that the operator only changes the preset entry if necessary. If the list box is larger than the selection rectangle, WinCC automatically adds a scroll bar to the right margin.

To incorporate list fields into the process, dynamize the corresponding properties.



Layout

In the Inspector window, you customize the position, style, colors and font type settings of the object. You can adapt the following properties in particular:

- "Selection items": Defines the list entries.
- "Selection of entries": Defines which entry is displayed as activated by default.
- "Selection mode": Specifies whether only one entry or multiple entries can be selected.

Defining the number of entries

1. Click "Properties > Selection items" in the Inspector window.
2. Click on the selection button in the "Static value" column.
   A dialog opens.
3. Specify the required number of selection items.

Specifying the default setting of the list boxes

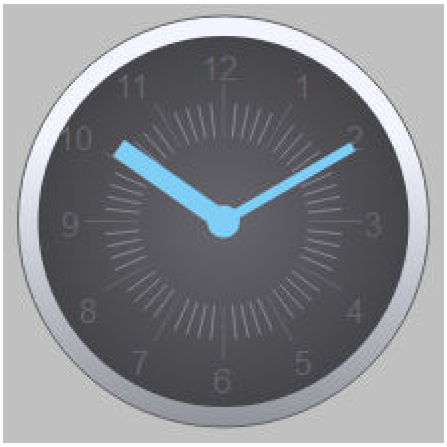
Use the "Select item" property of a selection item to specify which list item is to be shown as enabled.

To do so, select the check box in the "Static value" column of the "Select item" property of the respective selection item.

1.3.2.10 Clock (RT Uni)

Application

The "Clock" object displays the time.



Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can adapt the following properties in particular:

- "Clock face mode": Specifies whether the hour marks of the analog clock are displayed as ticks or numbers.
- "Show hours pointer", "Show minutes pointer" and "Show seconds pointer": Specifies whether the hour hand, minute hand and second hand are displayed on the clock.

Configuring the clock face

In the Inspector window, you can specify how the hour marks are displayed.

1. Click "Properties > Clock face mode" in the Inspector window.
2. Select "Ticks" to display hours as ticks.
   Alternatively, select "Numbers" for a numerical display of the hours.

1.3.2.11 Symbolic I/O field

Application

The "Symbolic I/O field" object can be used to configure a selection list for input and output of texts in Runtime.

Layout

In the Inspector window, you customize the position, shape, style, color and font types of the object. You can customize the following properties in particular:

- Mode: Specifies the response of the object in Runtime.
- Text list: Specifies the text list that is linked to the object.

Mode

The response of the symbolic I/O field is specified in the Inspector window in "Properties > Properties > General > Type".

Mode	Description
"Output"	The symbolic I/O field is used to output values.
"Input"	The symbolic I/O field is used to input values.
"Input/output"	The symbolic I/O field is used for the input and output of values.
"Two states"	The symbolic I/O field is used only to output values and has a maximum of two states. The field switches between two predefined texts. This is used, for example, to visualize the two states of a valve: closed or open.

Text list

In the Inspector window you specify which text list is linked to the symbolic I/O field.

1. In the Inspector window, select "Properties > Properties > General":
2. Open the selection list for "Text list".
3. Select a text list.

1.3.2.12 Touch area (RT Uni)

Application

The "Touch area" object allows you to configure an object that the operator can use in runtime to execute any configurable function. A gesture on the user interface starts the execution of the function.

Note

A gesture is recognized in the area where it begins.

Layout

In the Inspector window you can customize the settings for the position, geometry and color of the object.

Available gestures

The operator can select between four predefined gestures in the Inspector window under "Properties > Events":

- 1-finger swipe downward
- 1-finger swipe left
- 1-finger swipe right
- 1-finger swipe upward

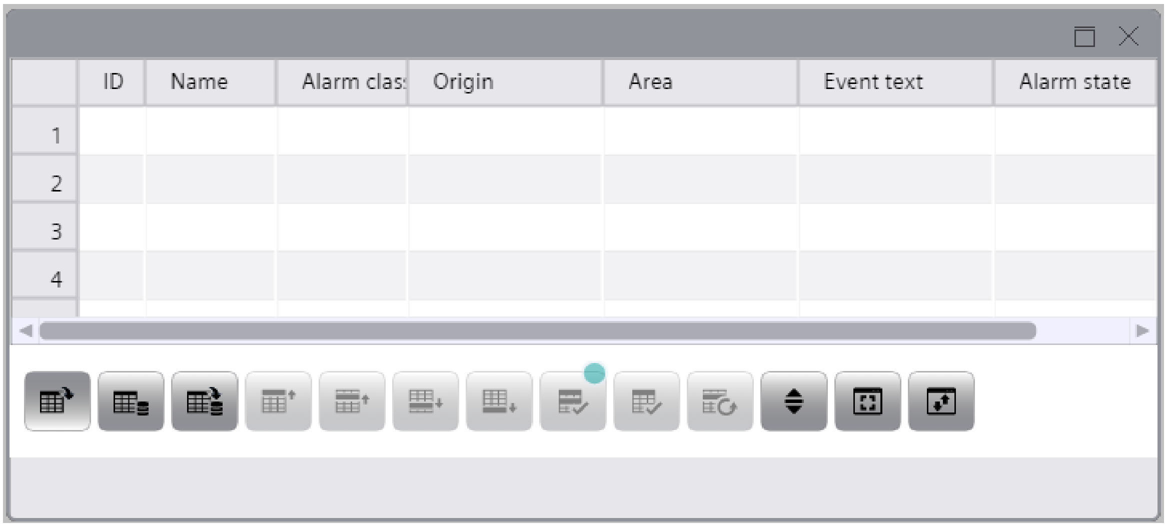
1.3.3 Controls (RT Uni)

1.3.3.1 Alarm control (RT Uni)

Use

The "Alarm view" object displays alarms that occur during the process in a plant. You also use the alarm view to visualize alarms in list format.

WinCC offers various views, such as "Current alarms" or "Logged alarms".



	ID	Name	Alarm clas	Origin	Area	Event text	Alarm state
1							
2							
3							
4							

Layout

You change the settings for the position, geometry, style, color, and font of the object in the Inspector window. You can adapt the following properties in particular:

- "Toolbar": Defines the control elements of the alarm view.
- "Status bar": Defines the elements in the status bar.
- "Text format": Defines the font and font size.

- "Allow sorting": Defines whether the alarms are sorted in runtime.
- "Alarm view": Defines the different properties for display of the alarms, e.g. background color and row height.

Configuring output of alarms

- "Alarm source": Defines which alarms are displayed in this alarm view.
- "Show recent": Defines whether it is always the latest alarm that is selected in the alarm view.

Operator controls

You define the control elements for the alarm view in runtime, and their operator authorizations, under "Properties > Toolbar" in the Inspector window. Some buttons are enabled by default. To display additional buttons in the control, activate the "Visibility" property in the settings of the corresponding button.

The following control elements are available for the alarm view :

Button		Function
	Show active alarms	Shows the currently active alarms.
	Show logged alarms	Shows the logged alarms.
	Show and update logged alarms	Updates the logged alarms and shows them.
	Show defined alarms	Shows the alarms configured in the system.
	Alarm annunciator	Shows all alarms for which the alarm annunciator was configured. The alarm annunciator is a visible or sound signal, for example a horn or warning light, that is displayed in addition to the alarm view in the system.
	First line	Selects the first of the active alarms. The visible area of the alarm view moves, if necessary. This button can only be used if the "Show recent" function is disabled.
	Previous line	Selects the previous alarm in relation to the currently selected alarm. The visible area of the alarm view moves, if necessary. This button can only be used if the "Show recent" function is disabled.
	Next line	Selects the next alarm in relation to the currently selected alarm. The visible area of the alarm view moves, if necessary. This button can only be used if the "Show recent" function is disabled.
	Last line	Selects the last of the active alarms. The visible area of the alarm view moves, if necessary. This button can only be used if the "Autoscroll" function is disabled.

Button		Function
	Move to next acknowledgeable alarm	Selects the next alarm in relation to the currently selected alarm. The visible area of the alarm view moves, if necessary. This button can only be used if the "Autoscroll" function is disabled.
⏪	Previous page	Navigates to the next page
⏩	Next page	Moves to the previous page
	Single acknowledgment	Acknowledges an individual alarm. A counter shows how many alarms are not acknowledged. The counter includes all connected servers, but no filters.
	Group acknowledgment	Acknowledges all active visible alarms in the alarm view that require acknowledgment, unless they are subject to single acknowledgment.
	Single confirm	Resets the alarm. Relevant for alarms with the state machine "Alarm with acknowledgment and confirmation" that have already been acknowledged and are outgoing.
⬍	Show recent	Defines whether it is always the latest alarm that is selected in the alarm view. The visible area of the alarm view moves, if necessary. You can only select individual alarms if "Autoscroll" is not active.
ℹ	Info text configuration	Opens a dialog to display an infotext.
	Disable alarm	Disables an alarm in the current alarm list and in the alarm log lists. The alarm is added to the display "Disabled alarms."
	Enable alarm	Shows an alarm once again.
	Shelve alarm	Shelves an alarm to prevent, for example, that an error alarm affects the effectivity of your system. The alarm appears in the "Shelved alarms" display.
	Unshelve alarm	Unshelves the respective alarm.
	Copy lines	Copies the selected alarms.
	Time base setup	Opens a dialog for setting the time zone for the time information shown in alarms.
	Selection display	Opens a dialog for filtering alarms. Here you define the filter criteria directly or filter the alarms by criteria defined in the engineering system.

Button		Function
	Sorting setup	Opens a dialog for setting user-defined sort criteria for the displayed alarms.
	Display options setup	Opens a dialog for configuring the display options of the alarm view. Here you define which alarms are displayed, for example, only shelved alarms or all alarms.
	Disabled alarms setup	Opens a dialog for configuring the display options of the locked alarms.
	Print	Starts printing the alarms displayed in the alarm view.
	Export	Starts exporting the alarms to a .CSV file.

Status bar

You define which of the status bar elements are displayed using "Properties > Status bar" in the Inspector window.

Access protection in runtime

Configure access protection with the properties "Allow operator control" and "Authorization" under "Properties" in the Inspector window. If a logged-on user has the required authorization, he can acknowledge, and edit alarms using the control elements in the alarm view.

Set up sort order

1. Click "Properties > Alarm view > Columns" in the Inspector window.
2. Select the sort criteria and sorting order for the individual columns.

Note

You define the sorting direction of the alarms in the alarm view under "Properties > Default sorting direction", e.g. "Ascending".

Displaying current alarms

When you select the property "Show recent", it is always the latest alarm that is selected in the alarm view. The visible area of the alarm view moves, if necessary.

You can only select individual alarms when the "Show recent" property is not enabled.

Set to a table in the alarm view

1. Click "Properties > Alarm view" in the Inspector window.
2. Define the settings for the rows and cells.
   - "Row height": Defines the height of the rows in the alarm view.
   - "Trim cell content": If the text is longer than the cell, the text is displayed truncated.
3. Under "Header settings" you define the settings for the headers:
   - "Row header": Defines whether or not each row has a header.
   - "Column header": Defines the display of the column header.
4. Specify the width and color of the grid lines.
5. Specify the use of scroll bars.

See also

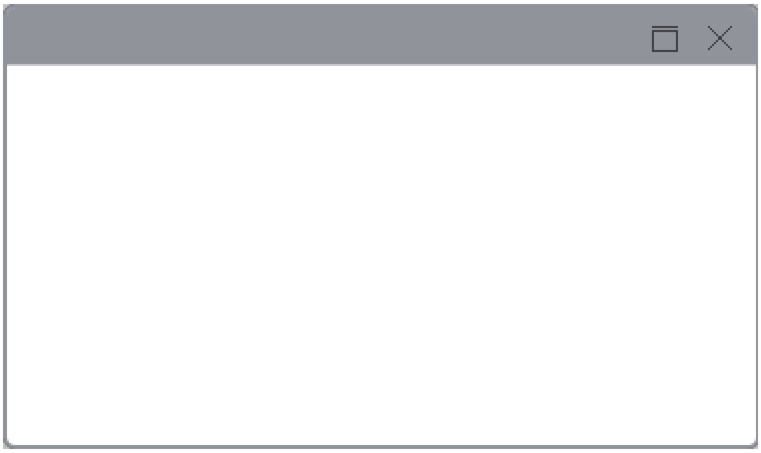
Configuring an alarm control (Page 253)

1.3.3.2 Screen window (RT Uni)

Application

You use the "Screen window" object to represent other screens from the project in the current screen. You can make the object dynamic to constantly update the content of a screen window, for example.

You can also use independent screen windows independently of the screen in question. With appropriate hardware equipment and support by the operating system you can also control multiple monitors and map processes in a more comprehensive and differentiated manner.



Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "Zoom factor": Defines the size of the embedded screen.
- "Fit to size": Specifies whether the embedded screen is to be adjusted to the screen window size or the screen window to the size of the embedded screen. If the embedded screen is larger than the screen window, you configure scroll bars for the screen window.

Matching the size of the embedded screen and screen window

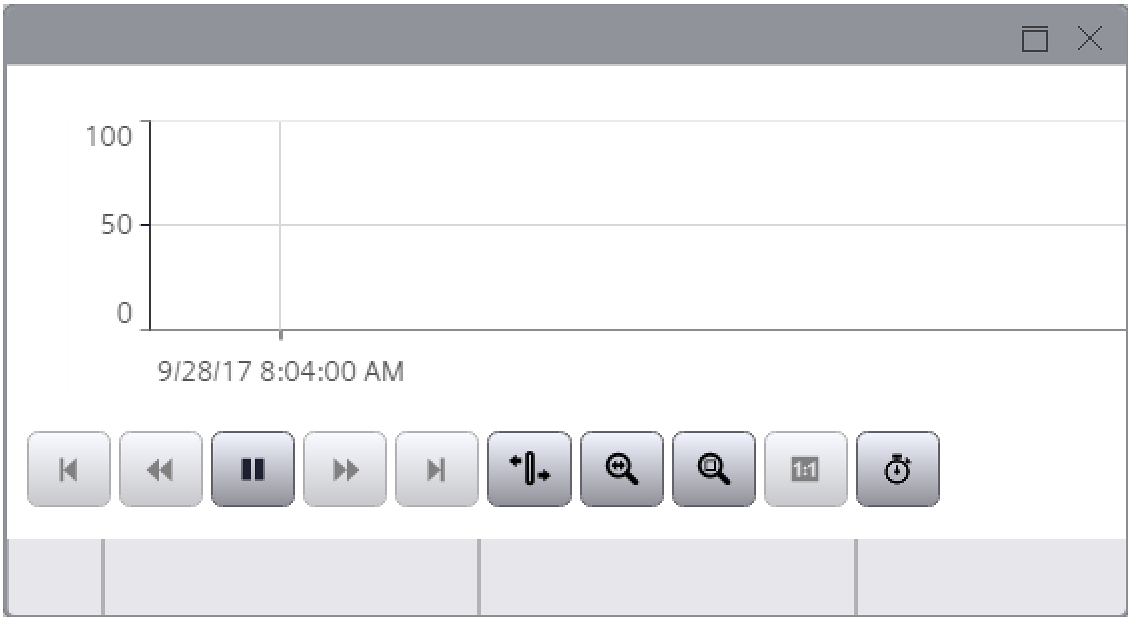
You can match the size of the embedded screen to the size of the screen window in the following ways:

- You want the embedded screen to appear smaller.
  Enter the required zoom factor under "Properties" in the Inspector window.
- You want to scroll to a section of the embedded screen.
  In the "Properties" Inspector window, show the horizontal and vertical scroll bars and specify their position.
  The user can scroll to details of the embedded screen in runtime.
- You can adapt the embedded screen to the size of the screen window, or vice versa.
  Select either "Fit window to screen", or "Fit screen to window" under "Properties > Fit to size" in the Inspector window.
  Choose between two options in scaling mode for the "Fit screen in window" setting:
  The aspect ratio is retained with both options.

1.3.3.3 Trend control (RT Uni)

Application

You use the "Trend view" object to display tag values from the current process or from the log in the form of trends as a function of the time.



Layout

In the Inspector window, you customize the position, geometry, style, colors and font types of the object. You can adapt the following properties in particular:

- "Configure trend area"
- "Configuring trends"
- "Define buttons in the toolbar"

Configure trend area

Configure the display of trends under "Properties > Trend areas":

- Common or individual trend areas
- Common or separate axes
- Writing direction of all trends

The first trend area [0] is by default already available in the control. You can create more trend areas using the selection button in the "Static value" column.

1. Configure the value axes and the time axes.
2. Open the settings of the time axis under "Properties > Time axis bottom > Time axis [0]".

3. Configure the "Time range" of the trend display.
   - "Time interval": You define the time range using a starting time and a following time interval.
   - "Start time and end time": You define the time range using a starting time and an end time.
   - "Measuring points": You define the time range using a starting time and a number of measuring points.
4. Open the settings of the value axis under "Properties > Left value axis > Value axis [0]".
5. If required, configure the "Value range", "Format" and "Scaling" of the value axis.
6. If required, specify the use of user-defined axis segments and add the segments.
7. Go to "Properties > Trend areas > Trends" and configure the trends for the trend area.

Configuring trends

You can configure the trends for each trend area in the Inspector window by expanding the index number for each trend required.

1. Select the data supply for a given trend under "Properties > Trend areas > Trends > Trend [0] > Data source Y > Source".
   - "Logging tag": The trend view is supplied with values from a data log.
   - "HMI_Tag": The trend view is supplied with values of a tag.
2. Select the data supply for the tag under "Properties > Trend areas > Trends > Trend [0] > Data source Y > Tag".
   - In the case of a HMI tag, specify the name of the tag in the "Static value" column.
   - In the case of a logging tag, first enter the name of the HMI tag in the "Static value" column and then the name of the associated logging tags separated by a colon, for example, "HMITag_1:LoggingTag_1".
3. Configure the display mode for trends under "Trend mode".

Toolbar

You define the operator controls of the trend view in runtime under "Properties > Properties > Toolbar" in the Inspector window. Some buttons are enabled by default. To display additional buttons in the control, activate the "Visibility" property in the settings of the corresponding button.

The following operator controls are available for the trend view:

Button	Name	Function
⏮	First record	Shows the trend direction starting with the first logged value.
⏪	Previous data record	Shows the trend direction of the previous time interval.

Button	Name	Function
⏸	Start/stop	Stops and starts the trend update. The values are buffered and updated as soon as you start trend update again.
⏩	Next record	Shows the trend direction of the next time interval.
⏭	Last record	Shows the trend direction up to the last logged value.
📈	Previous trend	Displays the previous trend in the foreground.
📈	Next trend	Displays the next trend in the foreground.
↔	Show/hide ruler	Determines the coordinates of a point of the trend.
🔍	Zoom time axis +/-	Enlarges or reduces the time axis display.
🔍	Zoom value axis +/-	Enlarges or reduces the value axis display.
🔍	Zoom area	Increases the size of any section of the trend window.
🔍	Zoom +/-	Enlarges or reduces the view in the trend window.
✥	Move trend area	Moves the display in the trend area.
↕	Move axes area	Moves the display in the axes area.
1:1	Original view	Switches from the magnified trend view back to the normal view.
⏱	Select time range	Opens the dialog for setting the time range displayed in the trend window.
📈	Select trends	Opens the dialog for setting the visibility of trends.

Button	Name	Function
	Select data connection	Opens the dialog for selecting the logs and tags that serve as data source for the trend view.
[∿]	Statistics area	Enables you to define a time range for which statistical values are determined. Vertical lines which you use to set the time range are displayed in the trend window.
∫[x]	Calculate statistics	Opens a statistics window to display the minimum, maximum, means, and standard deviation for the selected time range and the selected trend.
🖶	Print	Starts printing the trends shown in the trend window.
	Export	Opens the dialog for saving the trend data in CSV format.

The order of the buttons is fixed. Under "Hotkey" you can set up individual key assignments, and shortcuts for every button on the toolbar.

See also

Configuring a trend control (Page 193)

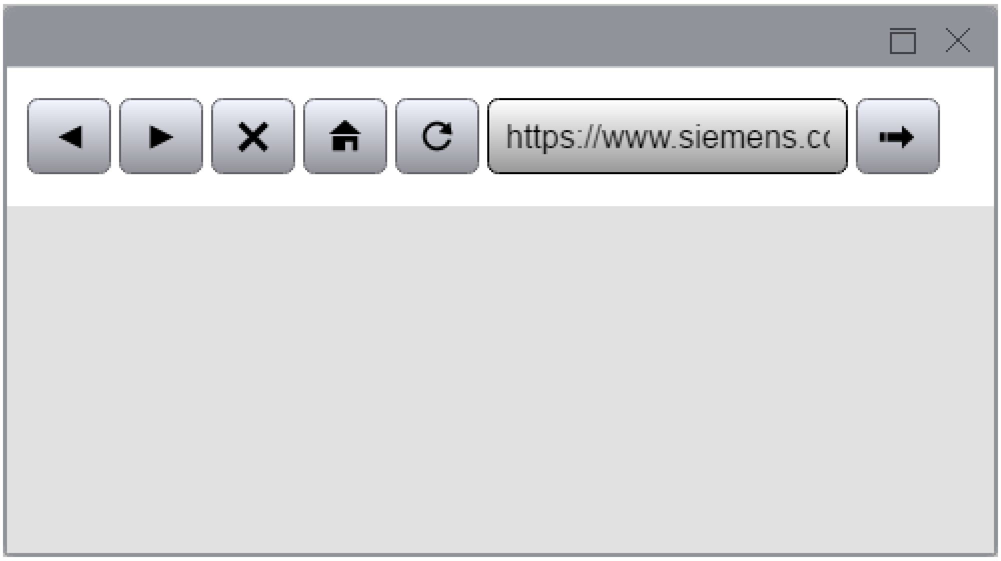
Configuring toolbar and status bar (Page 199)

Defining the data source (Page 200)

1.3.3.4 Browser (RT Uni)

Use

The "Web control" object is designed for the visualization of simple HTML pages. This function allows you to draw up machine-specific descriptions which are stored centrally and which can then be displayed from different HMI devices.



Layout

Customize the object position and size in the Inspector window. In particular, you can customize the following property:

- "URL": Specifies which Internet address is opened in the HTML Browser.
- "Toolbar": Specifies whether a navigation toolbar is shown.

Address

You set the Internet address in the Inspector window under "Properties > URL".

Displayed contents

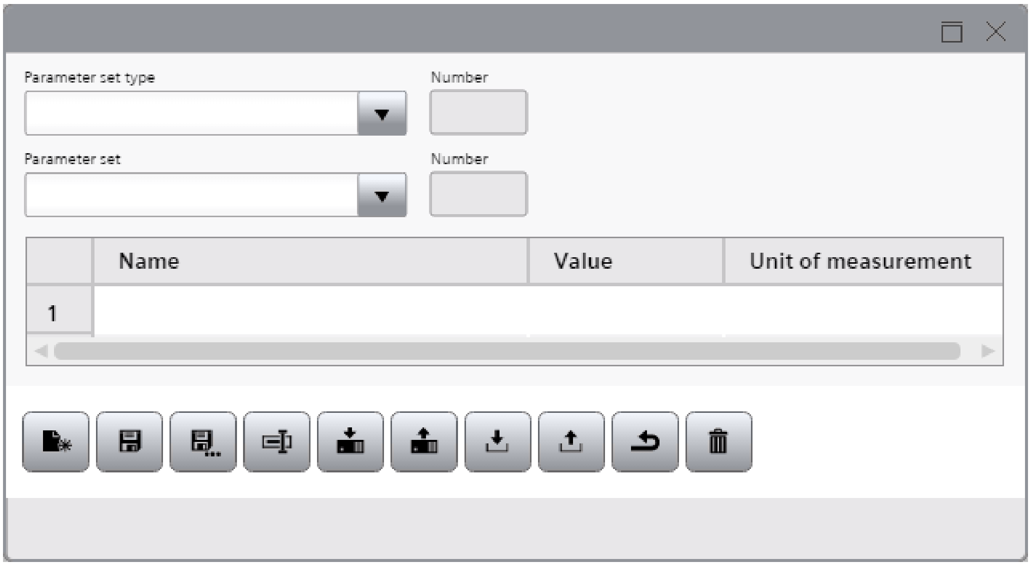
Note the following information when using the control:

- The "Browser" control only displays content that is supported by the web browser in which runtime is open.
- The control is implemented as IFrame. Pages with X-frame option settings that prevent display in an IFrame are not displayed in the control.
- Compared to a standard browser, the "Browser" control has a limited range of functions:
  - Navigation from the "Browser" control is not supported (top level navigation).
  - Calls of queries and dialogs are not supported (pop-ups).

1.3.3.5 Parameter set control (RT Uni)

Use

The parameter set control is used to display parameter sets in runtime, to manage them and to exchange them with the control system.



Note

The "Parameter set control" object is supported with version V16 exclusively for Unified PC. If the user uses the object under Unified Comfort Panel, an error message of the compiler is returned. Existing projects under Unified Comfort Panel that have configured the object, must delete the object before compiling to version V16.

Layout

You change the settings for the position, geometry, style, color, and font of the object in the Inspector window. You can adapt the following properties in particular:

- Display selection list: If you clear the check box of the property, the Parameter set type field and the Parameter set type ID field with the associated labels are hidden.

  Note

  If you hide the two fields and do not select a parameter set type under "Properties > Fixed parameter set type", the Parameter set type field is disabled in runtime. In addition, no parameter set ID is displayed in the Parameter set ID field in runtime.

- "Parameter view": Defines the display of the parameter table in the control.
- "Toolbar": Defines the operator controls of the parameter set control.
- "Status bar": Specifies the display of the status line.

  Note

  The "Status Text" element is the only status line element of the parameter set display. Status messages are displayed in this element in runtime.

Using a parameter set type.

If you only want to use a particular parameter set type with its parameter sets in runtime, select the desired parameter set type under "Properties > Fixed parameter set type".

Operator controls

You define the operator controls for the parameter set control in runtime, and their operator authorizations, under "Properties > Toolbar" in the Inspector window. By default, all buttons are displayed in the toolbar. To hide specific buttons, deactivate the "Visibility" property in the settings of the corresponding button.

The following operator controls are available for the parameter set control:

	Button	Function
	Create	Creates a new parameter set.
	Save	Saves a parameter set.
	Save as	Saves an existing parameter set under a new name and new ID.
	Rename	Renames the selected parameter set.

	Button	Function
⬇	Write to PLC	Writes the values of the selected parameter set to the PLC.
⬆	Read from PLC	Writes the values of the selected parameter set from the PLC.
⬇	Import	Imports parameter sets from a "*.tsv" file.
⬆	Export	Exports parameter sets to a "*.tsv" file.
↶	Cancel	Cancels the process.
🗑	Delete	Deletes the selected parameter set.

Note

A "*.tsv" file is a text file that uses the tabulator as a list separator.

Enabling/disabling operator controls

In "Properties > Editing mode", configure the activation status of the toolbar buttons "Create", "Save", "Save as", "Rename" and "Delete". These toolbar buttons are used to edit parameter sets.

You can select between the following settings:

- "None": Deactivates all buttons.
- "Update": Activates the "Save" and "Rename" buttons.
- "Create": Activates the "Create" and "Save as" buttons
- "Delete": Activates the "Delete" button

Configuring a status bar

1. Configure the general properties of the status bar such as the font or the background color under "Properties > Status bar".
2. To adjust the size of the "Status Text" element, activate the "Customized" property under "Properties > Status bar > Elements > Control bar label [0]".
3. You can enter a pixel value for the width and height.

Note

Status messages are displayed in runtime in the "Status text" element.

Configuring a time zone

Under "Properties > Time zone", you set the desired time zone in which you enter a numerical value.

The numerical value stands for a time zone, for example:

- "-1" stands for UTC-1h (Central European Time, standard time)
- "1" stands for UTC-12h (International Date Line West)
- "2" stands for UTC-11h (Hawaii)

See also

Configuring the parameter set view (Page 805)

1.3.3.6 Faceplate container (RT Uni)

Application

The faceplate container is used to display faceplates in runtime. If a faceplate type has been instantiated in the container, the desired faceplate type is specified in the "Contained type" property.

You can find detailed information on configuring faceplates in the section "Configuring faceplates".

Layout

You change the settings for the position, geometry, style, color, and font of the object in the Inspector window.

You can adapt the following properties in particular:

- "Contained type": Defines the faceplate type that is instantiated in the faceplate container.
- "Properties": When a faceplate type is instanced in the faceplate container, you supply the tags of the interface in the properties.

See also

Basics of faceplates (Page 103)

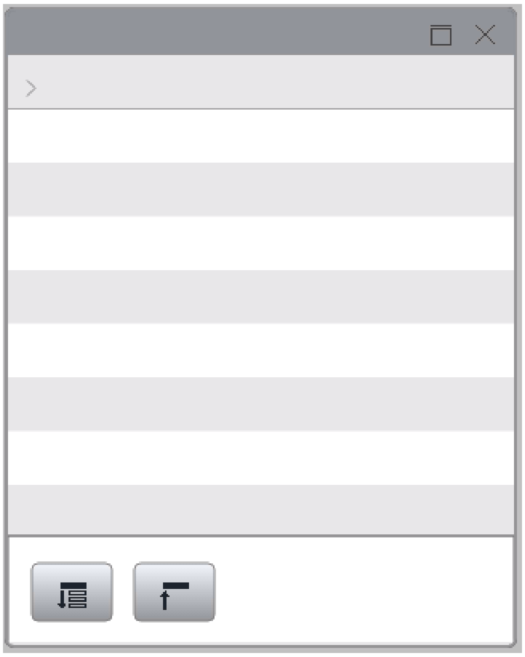
1.3.3.7 Plant overview (RT Uni)

Application

You use the "Plant overview" object to display the configured plant view in runtime.

You use it to navigate to the plant objects within the plant structure and get an overview of your plant at one glance.

If you have configured screens or alarms for the lower-level plant objects and have linked them to the "Plant overview" object, navigate to these pictures and alarms and display them.



Note

The "Plant overview" object is supported with version V16 exclusively for Unified PC. If the user uses the object under Unified Comfort Panel, an error message of the compiler is returned. Existing projects under Unified Comfort Panel that have configured the object, must delete the object before compiling to version V16.

Layout

You change the settings for the position, geometry, style, color of the object in the Inspector window.

To enable navigation between the screens of the plant objects, configure the corresponding controls under "Properties > Accompanying controls".

Operator controls

The following operator controls are available for the "Plant overview" object in runtime:

Button	Name	Function
↓☰	Expand	Expands the plant view with the lower-level plant objects.
↑▔	Collapse	Collapses the plant view with the lower-level plant objects.

1.3.3.8 Reports (RT Uni)

Use

The "Reports" object is used to create and manage report tasks in runtime. It also gives you access to the reports generated by the report tasks.

You can find detailed information on configuring the object in engineering in the section Configuring production reports in the engineering system (Page 933).

You can find detailed information on configuring report tasks in Runtime in the section Working with production logs in runtime (Page 962).

Note

The "Reports" object is supported with version V16 exclusively for Unified PC. If the user uses the object under Unified Comfort Panel, an error message of the compiler is returned. Existing projects under Unified Comfort Panel that have configured the object, must delete the object before compiling to version V16.

Layout

In the Inspector window, you change general settings of the object such as the position, height, width, label and window settings.

See also

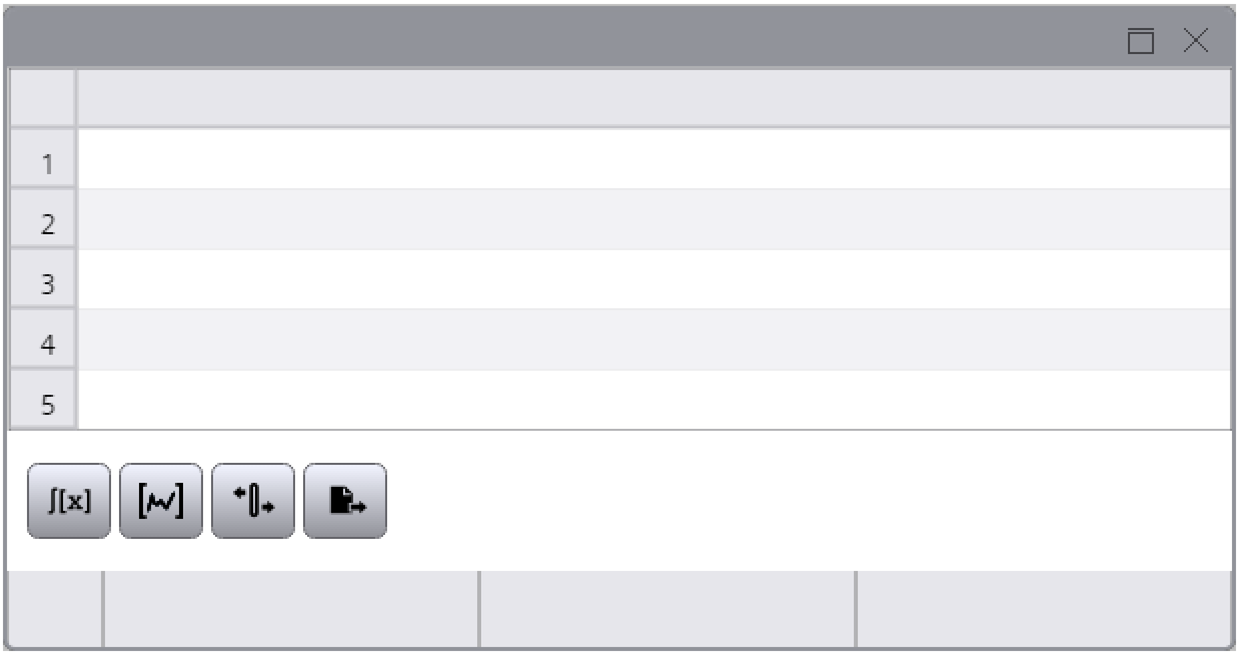
Basics of Reporting (Page 931)

The user interface of the "Reports" control (Page 962)

1.3.3.9 Trend companion (RT Uni)

Application

You use the "Value table" object to show evaluated data and statistics in a table.



- Trend view
- f(x) trend view

Note

The "Value table" object is supported with version V16 exclusively for Unified PC. If the user uses the object under Unified Comfort Panel, an error message of the compiler is returned. Existing projects under Unified Comfort Panel that have configured the object, must delete the object before compiling to version V16.

Layout

In the Inspector window, you customize the position, geometry, style, colors and font types of the object. You can adapt the following properties in particular:

- "Data source for displaying the values"
- "Mode"
- "Toolbar"

Data source for displaying the values

You define which values are displayed in the value table in the "Properties > Data source" Inspector window.

To adjust display to the corresponding control, enable the options "Use data source background color" and "Use data source font color".

By default, the configuration for the display format, the format is taken from the connected control. The size, value range and zoom factor of the control are taken into account to display the optimum number of decimal places. You can configure the display formats for individual values in the Inspector window of the value table yourself, for example, to show a precise number of decimal places.

Mode

You define the mode in the "Properties > Value table mode" Inspector window. You have a choice of three different types depending on the data source.

- The ruler window shows the coordinate values of the trends on the ruler or values of a selected line in the table.
- The statistics area window shows the values of the lower limit and upper limit of the trends between two rulers or the selected area in the table. The statistics area window is not provided for the f(x) trend view object.
- The statistics window shows the statistical evaluation of the trends between two rulers or the selected values in the table. The statistics window is not provided for the f(x) trend view object.

Toolbar

You define the operator controls for the value table in runtime in the "Properties > Toolbar > Elements" Inspector window. Some buttons are enabled by default. To display additional buttons in the control, activate the "Visibility" property in the settings of the corresponding button.

The following control elements are available for the recipe data:

	Brief description	Description
∫[x]	Calculate statistics	The button shows the statistical values in the statistics window. The displayed values refer to a selected trend with the configured calculation time period. The button can only be pressed if a statistics window is connected with a trend view.
	Statistics area	Enables you to define a time range for which statistical values are determined.
	Ruler window	You query the coordinate points of a trend with the button. The trend data are displayed in the ruler window.
	Printing	Start the print-out of the values shown in the table.
	Export	This button is used for exporting all or the selected runtime data into a "CSV" file.

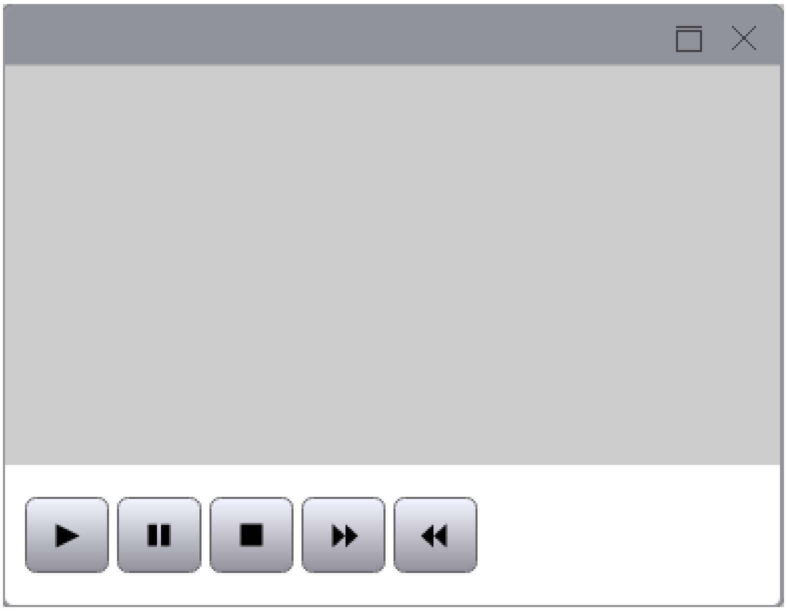
See also

Configuring the trend companion (Page 198)

1.3.3.10 Media Player (RT Uni)

Use

In runtime, the Media Player is used to play multimedia files.



Layout

You can set the following properties in the Inspector window:

- "Display status bar": Determines whether to display the status bar.
- "Toolbar" > "Elements": Specifies the operator controls in runtime.

Operator controls

The operator controls that can be used to control the Media Player in runtime are specified in the Inspector window under "Properties > Toolbar > Elements".

Supported file formats

All file formats that are supported by the browser used are also supported in the Media Player.

Note

Playing back multimedia files in the control depends on the video and audio codecs installed on the PC, as well as on the file format.

Note

Data loss when copying the project

If you copy the project to another PC, keep the following in mind:

Files indicated in the WinCC Media Control are not copied along with the other files if they are dynamically linked and no UNC path is specified. You have to load the files into the project again.

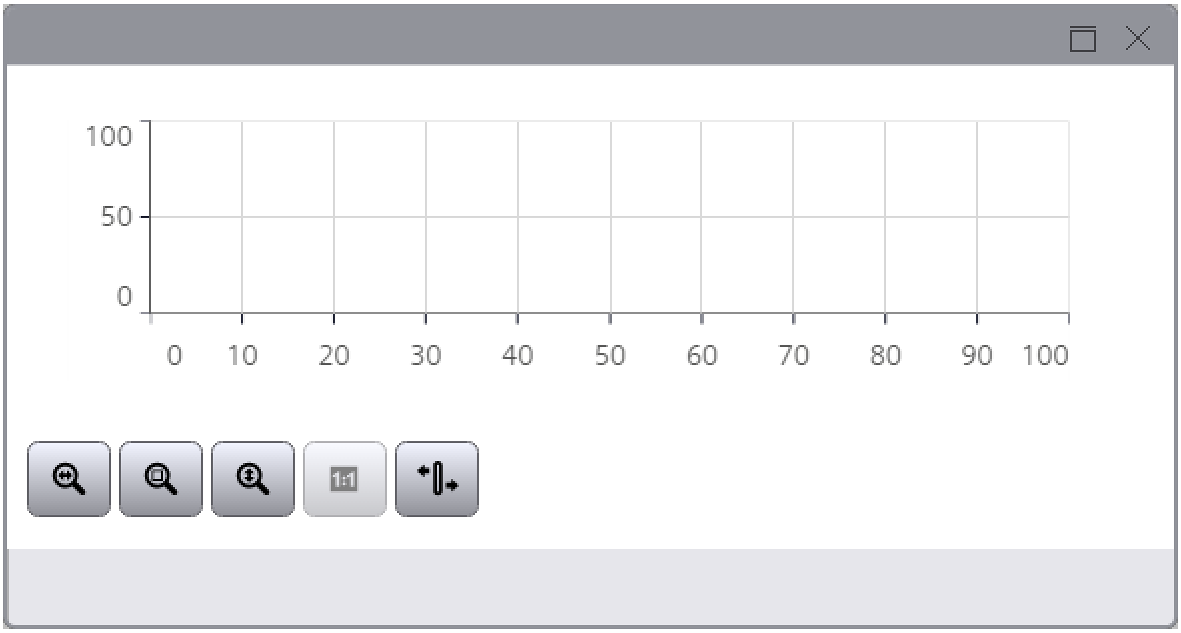
See also

http://support.automation.siemens.com (http://support.automation.siemens.com/WW/view/en/62101921)

1.3.3.11 Function trend control (RT Uni)

Use

You use the "f(x) trend view" object to represent the values of a tag as a function of another tag. This means that you can present temperature trends as a function of the pressure, for example. You can also compare the trend to a setpoint trend.



Layout

In the Inspector window, you customize the position, geometry, style, colors and font types of the object. You can adapt the following properties in particular:

- "Function trend area"
- "Adding and configuring trends"
- "Toolbar"

Configuring trends

1. Select the data supply for the function trend under "Properties > Function trend area > Function trends > Function trend [0] > Data source X > Source type".
   - "Log": The trend view is supplied with values from a data log.
   - "Undefined": The trend view is supplied in runtime with user-defined scripts.
   - "Online": The trend view is supplied with values of a tag.
2. Select the tag name under "Properties > Function trend area > Function trends > Function trend [0] > Data source X > Tag".
   - In the case of a HMI tag, specify the name of the tag in the "Static value" column.
   - In the case of a logging tag, first enter the name of the HMI tag in the "Static value" column and then the name of the associated logging tags separated by a colon, for example, "HMITag_1:LoggingTag_1".
3. Configure the data supply for "Data source Y".
4. Configure the trend display for the "Time range".
   - "Time interval": You define the time range using a starting time and a following time interval.
   - "Start time and end time": You define the time range using a starting time and an end time.
   - "Measuring points": You define the time range using a starting time and a number of measuring points.
5. Configure the value range of the trend display under "Left value axis" and "Bottom value axis".
   - "Automatically adapt value range" The displayed value range is automatically adapted to the current values.
   - "Minimum scale value" / "Maximum scale value": You define the minimum value and maximum value for the value range.

Toolbar

You define the control elements of the f(x) trend view in runtime under "Properties > Toolbar > Elements" in the Inspector window. Some buttons are enabled by default. To display additional buttons in the control, activate the "Visibility" property in the settings of the corresponding button.

The following control elements are available for the f(x) trend view:

Button	Name	Function
🔍	Zoom +/-	Enlarges and/or shrinks the trends in the trend window.
🔍	Zoom area	Increases the size of any section of the trend window.

Button	Name	Function
🔍	Zoom X axis	Enlarges and/or reduces the X axis in the trend window.
🔍	Zoom Y axis	Enlarges and/or reduces the Y axis in the trend window.
1:1	Original view	Switches from the magnified trend view back to the normal view.
📈	Previous trend	Displays the previous trend in the foreground.
📈	Next trend	Displays the next trend in the foreground.
↔	Ruler	Determines the coordinates of a point of the trend.
✥	Move trend area	You can move the trends in the trend window along the X axis and the Y axis using the button.
↕	Move axes area	You can move the trends in the trend window along the value axis using the button.
📈	Select trends	Opens a dialog for setting the visibility of trends.
▥	Select data connection	Opens a dialog for selecting logs and tags.
🖶	Print	Click this button to print the trend shown in the trend window. The print job used during printing is defined in the configuration dialog in the "General" tab.
📈	Export data	This button is used for exporting all or the selected runtime data to a csv file.

The order of the buttons is fixed. The operator can set up individual key assignments, and shortcuts for every button on the toolbar.

See also

Configuring the function trend control (Page 195)

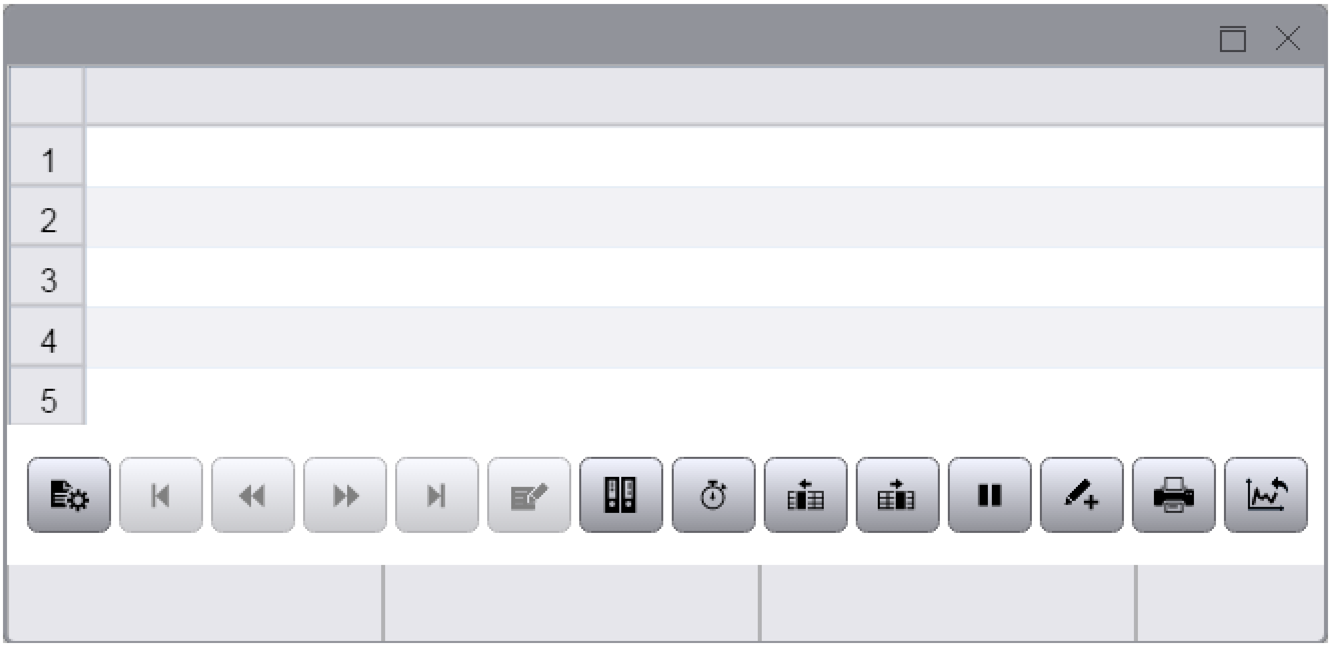
Configuring toolbar and status bar (Page 199)

Defining the data source (Page 200)

1.3.3.12 Process control (RT Uni)

Application

You use the "Process control" object to display the tag values in a table. You can display current, or logged values in the table. You can configure up to nine value columns. The first column is reserved for the time column.



Layout

In the Inspector window, you customize the position, geometry, style, colors and font types of the object. You can adapt the following properties in particular:

- "Configuring data source and columns"
- "Configuring a table"
- "Configuring a toolbar"

Configuring data source and columns

1. Go to "Properties > Process view > Columns > Time range column [0]" and open the settings of the time column.
2. Select the time range of the table under "Properties > Process view > Columns > Time range column [0] > Time range":
   - "Time interval": You define the time range using a starting time and a following time interval.
   - "Start time and end time": You define the time range using a starting time and an end time.
   - "Measuring points": You define the time range using a starting time and a number of measuring points.
3. Go to "Properties > Process view > Columns" and open the settings of the respective value column.
4. Under "Data source" select the type of data source and the tag that supplies the column with values.
5. Under "Sorting order" define the sorting order of the columns in the process control.
6. If necessary, define the "Sorting direction" in which the values are sorted.

Toolbar

You define the operator controls of the process control in runtime in the "Properties > Properties > Toolbar" inspector window. Some buttons are enabled by default. To display additional buttons in the control, activate the "Visibility" property in the settings of the corresponding button.

The following operator controls are available for the process control:

Button	Name	Function
⏮	First record	Shows the tag values starting with the first logged value.
⏪	Previous data record	Shows the tag values in the previous time interval.
⏸	Start/stop	Stops and starts the column update. The values are buffered and updated as soon as you start column update again.
⏩	Next record	Shows the tag values in the next time interval.
⏭	Last record	Shows the tag values up to the last logged value.
✎	Edit	Allows the editing of data in any table field that is opened when the user double-clicks it.

Button	Name	Function
←	Previous column	Displays the previous column in the foreground
→	Next column	Displays the next column in the foreground
⏱	Select time range	Opens the dialog for setting the time range displayed in the process control.
▮▮	Select data connection	Opens the dialog for selecting the logs and tags that serve as data source for this process control.
✎+	Create archive value	Creates an archived value.
🖨	Print	Starts printing the columns displayed in the process control.
➜	Export	This button is used for exporting all or the selected runtime data into a "CSV" file.

See also

Configuring the process control (Page 196)

1.3.4 My Controls (RT Uni)

Use

You can use My Controls in WinCC that have been created externally. My Controls are freely programmable and serve as a specific solution that goes beyond the functionalities of the toolbox provided. Like all other tools, My Controls are used within screens and displayed in runtime.

Requirement

- A WinCC project has been created
- A screen has been created

Procedure

To use My Controls, proceed as follows:

1. Open the directory of your project.
2. Open the "UserFiles" subfolder.
3. Create a folder with the name "CustomControls".
4. Store the created program as *.zip archive in the "CustomControls" folder.

Note

Update

To use My Controls in the tool list, you need to close and restart the WinCC application.

1.4 Configuring faceplates (RT Uni)

1.4.1 Basics (RT Uni)

1.4.1.1 Basics of faceplates (RT Uni)

Introduction

Faceplates are user-defined groups of display and operating objects that are stored, managed and edited centrally in the project.

Faceplates can be scripted and can therefore also open other faceplates in a pop-up window.

Depending on design and configuration, faceplates can be used universally and easily integrated into existing projects and employed several times.

Use

You use faceplates to create individually configured display and operating objects. You use faceplates several times in the project. All changes to the faceplate in the project can be changed centrally in the faceplate type. This reduces the configuration effort.

Depending on the application, a faceplate is a user-defined simple screen item or a detailed representation of a complex plant component.

Ideally, you should use faceplates for plant objects or parts that you use several times and that have identical data structures.

Type/instance concept

In order to support central changeability, faceplates are based on a type-instance model:

- You create properties for faceplate instances centrally in the faceplate type.
- The instances represent local point of use of the faceplate type.

Faceplate type

- You create a faceplate type with basic objects and elements in the "Unified Faceplate Types" editor to match your needs.
- You specify the tags and user data types (UDTs) that will be accessible in the faceplate instance.
- You configure interface properties which you can define differently at each faceplate instance without changing the faceplate type.
- The properties of the faceplate type and the basic objects used can be dynamized.
- You configure reactions to events on the basic objects using a script. This can also be used to create pop-ups, for example.

Faceplate container

The faceplate container is an independent object in which a faceplate type is instantiated.

- Each instance is connected to the faceplate type that has been used.
  This means that if you change a property or the data structure of a faceplate type, this property change immediately affects all faceplate instances that are based on faceplate type.
- A faceplate container is used in screens just like any other display and operating object.
- If a faceplate type has been instantiated in the container, the corresponding faceplate type is specified in the "Contained type" property.
- The tags and interface properties configured in the faceplate type are linked in the faceplate container.

Example

If you use multiple valves within your project, you typically always use the same data structures to control and query the status of these valves. Therefore, it makes sense to use the same display and operating objects for the visualization of these valves.

1. In a faceplate type, you configure how the valve is displayed and which input and output tags the valve has in the form of tags and UDT structures.
2. If required, configure another faceplate type that contains the same data structure and functions as a pop-up window.
   This pop-up window can be called by the first faceplate type using a script.
3. For each valve in the system with the same data structure, instantiate the desired faceplate type and link its tags and UDT structures with the corresponding ones in the system.

1.4.1.2 Device dependency of faceplates (RT Uni)

Not all displays and HMI devices support faceplates. The screen and operating objects that are not available in the respective HMI device are not displayed when using the faceplate.

Independent of the device, all properties are offered for the configuration during the generation of faceplates. When using a faceplate container in a screen, only the properties supported by the configured device are available.

Devices

The following devices support faceplates:

Runtimes

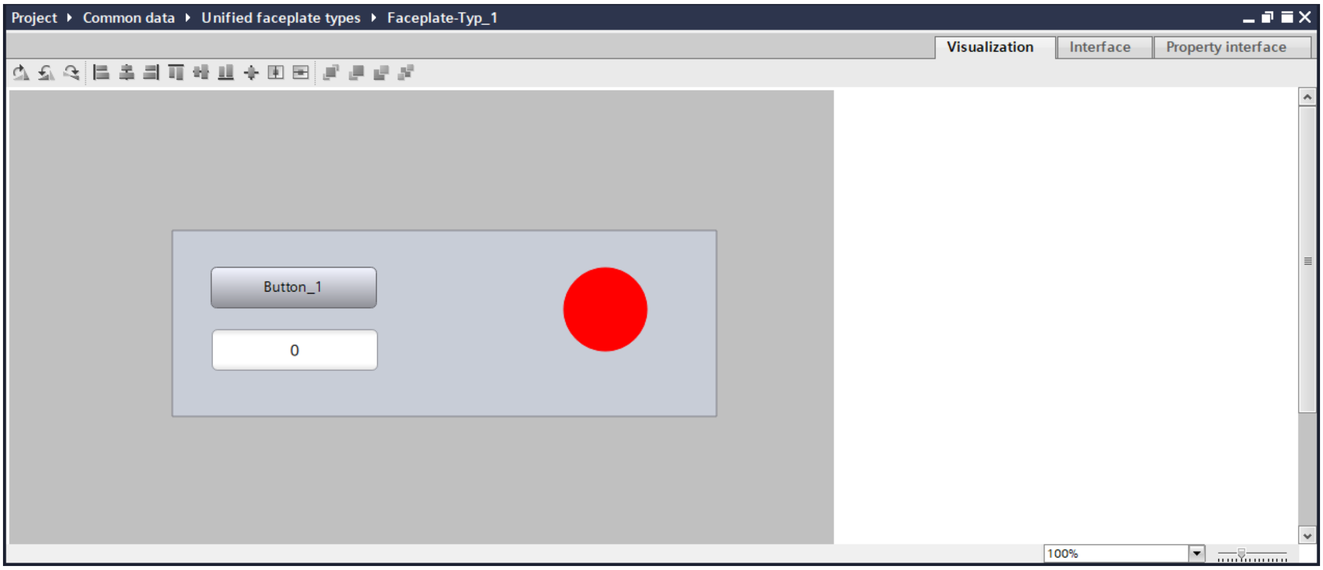
- WinCC Unified Scada RT

Comfort Panels

- Unified Comfort Panel

1.4.1.3 "Unified Faceplate Types" editor (RT Uni)

Layout


Project ▸ Common data ▸ Unified faceplate types ▸ Faceplate-Typ_1
Visualization
Interface
Property interface
Button_1
0
100%

The editor for faceplate types is divided into 3 main areas:

- Visualization
  You can visually design the faceplate type here.
  In the Inspector window, you define the properties of the faceplate type in the area "Properties > Properties". Here, you define display name, appearance and size, for example.
- Tags of the interface
  Here you configure tags and specify existing user data types that you need in the faceplate type.
- Interface properties
  Here, you define the interface properties of the faceplate type.

"Visualization" tab

In the "Visualization" tab, you place the objects you need in the faceplate type, similar to the familiar procedure in the "Screens" editor.

You insert new objects from the "Toolbox" task card under "Basic objects" and "Elements". Edit these objects according to your requirements.

"Interface tags" tab

In the "Interface tags" tab, you configure the tags of the faceplate type.

Here, you also use user data types (UDTs) that you have previously created in controllers (PLC).

"Interface properties" tab

In the "Interface properties" tab, you configure the interface properties of the faceplate type.

You create interface properties of the following data types:

- Color
- Resource list

1.4.2 Creating and managing faceplates (RT Uni)

1.4.2.1 Creating a faceplate type (RT Uni)

Creating a faceplate type

1. In the project tree in the "Devices" tab, expand the "Shared data" > "Unified faceplate types" folder.
2. Double-click the "Add new faceplate type" entry.
   The "Unified faceplate types" editor opens.
3. Add objects from the "Toolbox" task card, which you are familiar with from the "Screens" editor.
   You can currently use objects from the following categories:
   - Basic objects
   - Elements

Editing properties

1. Configure the properties for the faceplate type and the objects used in the Inspector window under "Properties > Properties".
2. Configure events on the relevant objects.
3. Dynamize the object properties as required.

Configuring tags and using user data types

1. Switch to the "Interface tags" tab.
2. Select "<Add>".
3. Specify the data type for the tag.
4. Configure all required tags.
5. When you use a user data type, select "Struct" as the data type and then select the corresponding user data type.

Use the buttons ⭱ "Moves selected element up" and ⭳ "Moves selected element down" to change the order of the tags.

Configuring interface properties

1. Switch to the "Interface properties" tab.
2. Select "<Add>".
3. Specify a data type:
   - Color
   - Resource list

Use the buttons ↑ "Moves selected element up" and ↓ "Moves selected element down" to change the order of the properties.

You use the configured interface properties to dynamize properties within the faceplate. Use the "Interface properties" entry in the "Dynamization" column for this purpose.

You dynamize the "Text list" property of a symbolic I/O field with an interface property of the "Resource list" data type.

Note

If you change the data type of an interface property that you have already used to dynamize a property in the "Visualization" tab, you need to adapt the dynamization of the property.

Result

- The faceplate type created can be instantiated in one or more faceplate containers.
- The configured tags of the faceplate type can be used within the faceplate type. If the faceplate type is used in a plant object type, these tags can be linked to interface tags of the plant object type.
- To dynamize colors and resource lists within the faceplate type, use the configured interface properties. When you instantiate the faceplate type, you specify the values for the interface properties. This allows you to use different colors and resource lists for each instantiated faceplate container.

See also

Basics of screens (Page 21)

Configuring a faceplate type (Page 112)

1.4.2.2 Link faceplate type to a plant object type (RT Uni)

Introduction

Faceplate types can be linked to plant object types. The tags of the faceplate type are linked to the interface tags of the plant object type.

Requirement

- A plant object type is configured.
- Interface tags are defined in the plant object type.
- A faceplate type with appropriately configured visualization, tags and interface properties has been created.

Procedure

1. Open the editor of the plant object type.
2. Switch to the "Visualization" tab.
3. Drag-and-drop the faceplate type you want to connect to the plant object type from the device view into the work area on "Drop faceplate here".
4. Expand the faceplate type.
5. Connect the faceplate tags to the respective interface tags of the plant object under "Interface connection".
   All interface tags do not have to be assigned.

Note

In TIA Portal V16, only one faceplate type can be linked.

Result

You have connected a plant object type to a faceplate type and assigned the corresponding tags.

See also

Creating a faceplate type (Page 107)

1.4.2.3 Creating a faceplate instance (RT Uni)

Introduction

Faceplate types are stored in the project tree in the "Devices" tab under "Shared data" > "Unified Faceplate types".

When you use the faceplate type in a screen, you create an instance of the faceplate type.

Note

The number of faceplate instances in a screen is basically not limited.

However, overall performance decreases more sharply

- When more faceplate instances are used in a screen.
- When more scripts are used in the instanced faceplate type.

Note

If you want to copy a faceplate instance from one project to another:

1. First copy the faceplate type into the target project.
2. Then copy the faceplate instances into the target project.

Requirements

- The project contains at least one faceplate type.
- A screen is open.

Procedure using drag-and-drop

1. In the project tree, open the node "Common data" > "Unified Faceplate Types" in the "Devices" tab.
2. Drag the desired faceplate type from the project tree into the screen.
   The faceplate container with the faceplate instance is added to the screen.
3. Open the Inspector window under "Properties > Properties > General > Interface".
4. Connect the faceplate tags to project tags.
5. Specify colors and resource lists for the interface properties.

Procedure via task card

1. Open the "Toolbox > Controls" task card.
2. Drag-and-drop the "Faceplate container" control into the screen.
   An instance of the "Faceplate container" control is configured, but no faceplate type is linked yet.
3. Select the faceplate container.
4. Open the Inspector window under "Properties > Properties > General".
5. Select the desired faceplate type under "Contained type" in the "Static value" column.
6. Open the Inspector window under "Properties > Properties > General > Interface".
7. Connect the faceplate tags to project tags.
8. Specify colors and resource lists for the interface properties.

Procedure via plant objects

If you have linked the faceplate type to a plant object type, proceed as follows:

1. Open the "Plant objects" tab in the project tree.
   The plant view is displayed.
2. Drag-and-drop the required plant object into the screen.
   The faceplate container with the faceplate instance is added to the screen.
   The tags and interface properties defined in the faceplate type are transferred.
3. Open the Inspector window under "Properties > Properties > General > Interface".
4. Specify colors and resource lists for the interface properties.

Result

- The faceplate type is instantiated in a faceplate container.
- The objects configured in the faceplate type are visible in the faceplate container.
- Faceplate tags and interface properties are defined for the faceplate container.
- If required, the properties of the faceplate container can be configured and dynamized in the inspector window.

1.4.2.4 Copying faceplate types and faceplates to other projects (RT Uni)

Introduction

Faceplates and faceplate types can also be transferred to other projects.

Requirements

- The target project contains the devices on which faceplates can be used.
- Faceplate types: If user data types are used in the faceplate type, the same user data types must be available in the target project.
- Faceplate instances: It must also be possible to integrate the tags of the used faceplate types into the target project.
- Both projects (source and target) are open.

Procedure

Note

Dependencies

Hierarchical dependencies exist between user data types, faceplate type and faceplate instances:

1. Faceplate instances use faceplate types.
2. Faceplate types use user data types where necessary.

Therefore, note the order:

1. Configure the user data types.
2. Copy the faceplate type.
3. Copy the faceplate instances.

Configuring user data types

1. Switch to the project from which you want to copy the faceplates.
2. Check the faceplate type to be copied for any user data types that might be used.

3. Go to the target project.
4. In the target project, configure the user data types required in the faceplate type that is to be copied.

Copying a faceplate type

1. Switch to the project from which you want to copy the faceplates.
2. Copy the desired faceplate type.
3. Go to the target project.
4. Insert the faceplate type into the target project.
5. Integrate the required user data types into the new faceplate type.

Copying a faceplate instance

1. Switch to the project from which you want to copy the faceplate instance.
2. Copy the faceplate container that the previously copied faceplate type uses.
3. Go to the target project.
4. Open the screen in which the faceplate instance is to be inserted.
5. Insert the faceplate container and check whether it uses the desired faceplate type .
6. Link the required tags and user data types in the faceplate container with those in the project.

See also

Basics of faceplates (Page 103)

1.4.3 Editing faceplate types (RT Uni)

1.4.3.1 Configuring a faceplate type (RT Uni)

Introduction

In the Inspector window under "Properties" you define the properties of the faceplate type and the objects used in it.

In the "Tags Interface" tab of the faceplate editor, tags can be defined that are linked in the faceplate instance to the tags in the configuration. Existing user data types can also be used instead of using tags.

In the "Interface properties" tab, interface properties can be defined that can be used within the faceplate type to dynamize properties. To do this, use the "Interface properties" entry in the "Dynamization" column of the respective property. When you instantiate the faceplate type, you specify the values for the interface properties. This allows you to use different colors and resource lists for each instantiated faceplate container.

Requirement

- The faceplate type is created.
- The faceplate type is open for editing in the "Unified Faceplate Types" editor.
- The "Properties" tab is opened in the Inspector window .

Procedure

The faceplate type can be configured in many ways:

- Edit properties of the faceplate type
- Define tags and re-use existing user data types
- Create interface properties in the "Interface properties" tab and use for the dynamization of properties
- Configure reactions to events in the faceplate type
- Dynamize properties in the faceplate type
- Configure scripts for events and for dynamizing properties

Note

Faceplate type tags and interface properties are referenced. When tags or interface properties are added, deleted, or renamed, they are displayed updated at each use of the faceplate type.

See also

Creating a faceplate type (Page 107)

Editing properties of a faceplate type (Page 114)

Configuring tags in the faceplate type (Page 114)

Basics for the dynamization of faceplates (Page 116)

Basics of dynamizing screens (Page 131)

1.4.3.2 Editing properties of a faceplate type (RT Uni)

Adapting properties

Both the properties of the faceplate type and the properties of the objects used are edited in the Inspector window under "Properties" > "Properties".

1. Select the required object.
   - If you want to adapt the properties of the faceplate type, click in a free area of the "Unified Faceplate Type" editor.
   - If you want to adapt the properties of a used object, click on it.
     The displayed handles indicate the selected object.
2. Open the shortcut menu with a right-click and select the "Properties" entry.
   The Inspector window displays the properties of the object or faceplate type.
3. Edit the properties according to your needs.

See also

Configuring tags in the faceplate type (Page 114)

1.4.3.3 Configuring tags in the faceplate type (RT Uni)

Introduction

In the faceplate type you configure tags with which you can dynamize the properties of the objects contained in the faceplate type or which you can embed in scripts.

The tags of a faceplate type are exclusively linked to the project tags via the faceplate container.

Note

The use of special characters such as "." and "@" is not permitted in tag names.

- Make sure that no special characters appear in the tag names.

Requirements

- The faceplate type has been created and open for editing.
- The "Tags Interface" tab is open in the "Unified Faceplate Types" editor.

General

The integration of existing UDT structures and the creation of new faceplate type tags is basically identical.

Defining tags and re-using existing user data types

Note

Only tags of the faceplate type are displayed within the "Unified Faceplate Types" editor.

1. Click the "Add Tag" button.
2. Click on the name of the tag and assign a name.
3. Select the desired data type.
   If you want to specify a user data type:
   - Select the "Struct" data type.
   - Specify the required user data type.
     All tags from the user data type are available in the faceplate type.
4. Repeat the procedure for all other required tags.
5. In the "Unified Faceplate Types" editor, go back to the "Visualization" tab.

Result

You have configured the tags and user data types required for the faceplate type.

The tags defined in the faceplate type are accessible in the corresponding faceplate instances and can be used for dynamization and for creating scripts within the faceplate type.

See also

Configuring an event in the faceplate type (Page 115)

1.4.3.4 Configuring an event in the faceplate type (RT Uni)

Introduction

In the Inspector window of the "Unified Faceplate Types" editor you define the reaction to events for the objects that are used in the faceplate type.

Use the tags that were previously created in the faceplate type to trigger the reaction to events in the faceplate instance.

Requirement

- A faceplate type has been created.

Procedure

Note

Using tags

Only use tags that are defined within the faceplate type.

Tags are used in the following cases:

- Dynamizing properties through a tag
- Dynamize properties with a script (if tags other than those defined in the script were used).

You link the tags defined in the faceplate type with the project tags in the faceplate instance.

- The reaction to events on objects in the faceplate type is configured in the inspector window under "Properties" > "Events".
- For creating scripts there are special snippets, which can be reached in the shortcut menu under "Snippets" > "Faceplates".
- Creating scripts in the faceplate type works the same way as creating scripts on objects in screens.

See also

Configuring tags in the faceplate type (Page 114)

Configuring faceplate scripts (Page 119)

Runtime scripting (Page 355)

1.4.4 Dynamizing faceplates (RT Uni)

1.4.4.1 Basics for the dynamization of faceplates (RT Uni)

General

Both the properties of the faceplate type and the properties of the objects used are dynamized in the Inspector window under "Properties" > "Properties".

Application

You can dynamize events and properties of faceplates at two levels.

1. Dynamizing objects in faceplate type
2. Dynamizing a faceplate instance

Dynamize properties of objects in faceplate type

Note

Using tags

Only use tags that are defined within the faceplate type.

Tags are used in the following cases:

- Dynamizing properties through a tag
- Dynamize properties with a script (if tags other than those defined in the script were used).

You link the tags defined in the faceplate type with the project tags in the faceplate instance.

Properties of objects in the faceplate type can be dynamized in the "Unified Faceplate Types" editor. You configure the individual objects as in the "Screens" editor.

- To dynamize properties and events, the following methods are available in the "Unified faceplate types" editor:
  - Tag
  - Script
  - Flashing (for colors)
  - Interface properties (for colors and text lists)
    With this method, you use the interface properties configured in the "Interface properties" tab.

  Depending on the property, only certain methods are available.
- You do not have access to the tags and scripts of the project within the faceplate type.
  To do this, you must configure tags in faceplate type, which you link to the tags of the project in the faceplate instance.
- Each faceplate instance created with the faceplate type has the same objects with identical dynamization.
  You edit this dynamization exclusively in the "Unified Faceplate Types" editor.

Dynamizing a faceplate instance

You configure the events or dynamic properties individually on the faceplate container. This dynamization refers exclusively to the entire faceplate container.

Properties of the objects used in the faceplate type cannot be dynamized directly. For this purpose, faceplate type tags must be defined in order to trigger a dynamization.

See also

Configuring faceplate scripts (Page 119)

Basics of dynamizing screens (Page 131)

1.4.4.2 Dynamizing a faceplate instance (RT Uni)

Introduction

Note

A faceplate type is always instanced in a faceplate container.

You dynamize properties of the faceplate instance in exactly the same way as you dynamize properties of another object in the "Screens" editor.

In the "Screens" editor, you can connect the dynamic properties of the faceplate container with a tag or a script that provides the property with values in runtime.

You have previously created the tags and scripts in the project.

Requirement

- A faceplate container with a faceplate instance is inserted in the screen.

Procedure

1. Select the relevant faceplate instance.
2. Open the shortcut menu of the faceplate instance (right-click) and select "Properties".
3. In the "Dynamization" column of the Inspector window, select the menu of the property that you want to dynamize.
4. Select the required method:
   - Tag
   - Script
   - Resource list (for strings)
   - Flashing (for colors)

   Depending on the property, only certain methods are available.

See also

Basics for the dynamization of faceplates (Page 116)

Basics of dynamizing screens (Page 131)

1.4.4.3 Configuring faceplate scripts (RT Uni)

Introduction

In the configuration area of the "Unified Faceplate Types" editor, you create scripts which you only use within a faceplate type. You can only refer to tags of the faceplate type or properties of the contained objects within the script. The script is used as a copy in the instance of the faceplate type.

Requirement

- The faceplate type has been created and opened in the "Unified Faceplate Types" editor.
- Faceplate tags or dynamic properties are created.

Procedure

Note

Using tags

Only use tags that are defined within the faceplate type.

You link the tags defined in the faceplate type with the project tags in the faceplate instance.

Creating scripts for objects in faceplate types works the same way as in the "Screens" editor.

See also

Basics for the dynamization of faceplates (Page 116)

Runtime scripting (Page 355)

1.4.5 Example: Creating and using faceplates (RT Uni)

1.4.5.1 Example: Configuring a faceplate (RT Uni)

In the following example, you create a faceplate type to specify a motor speed and then use an instance of this faceplate type in the project.

Procedures overview

The example is divided into the following steps:

1. Creating a faceplate type
2. Configuring tags in the faceplate type
3. Instead of tags: Use PLC data types (UDT) and thus minimize configuration workload

4. Configuring interface properties in the faceplate type
5. Creating a local script in the faceplate type
6. Creating a faceplate instance and integrating it in the project

See also

Example: Creating a faceplate type (Page 120)

Example: Configuring tags in the faceplate type (Page 122)

Instead of tags: Using the user data type (UDT) in the faceplate type (Page 123)

Example: Creating a local script in the faceplate type (Page 127)

Example: Creating a faceplate instance and integrating it in the project (Page 128)

1.4.5.2 Example: Creating a faceplate type (RT Uni)

Task

You create a faceplate type.

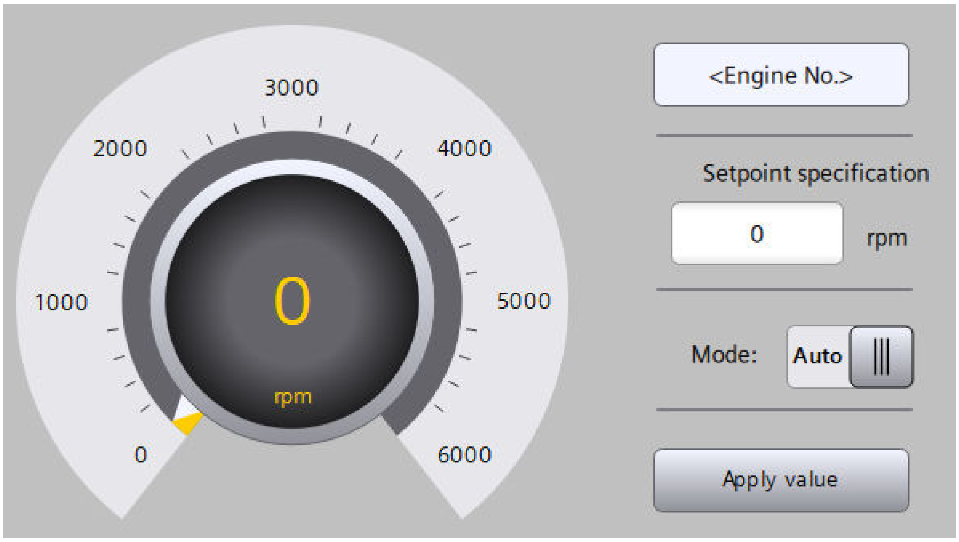
Requirement

- A project has been created.

Procedure

1. Create a new faceplate type and give it a meaningful name.
2. Open the faceplate type.
3. In the "Unified Faceplate Types" editor, go to the "Visualization" view.

4. Open the "Toolbox" task card.
5. Drag the required objects to the faceplate type; see table below.
   Arrange the objects according to your needs. This could look like this, for example:



Objects in the faceplate type

Object	Object name	Meaning/Properties
Gauge	Revolutions_Gauge	Display of the current motor speed
Text field	EngineName_Textbox	Text: Motor identification
Text field	SetPoint_Textbox	Text: Default setpoint
Text field	Unit_Textbox	Text: Unit for speed
Text field	Mode_Textbox	"Mode" text:
I/O field	Revs_InputField	Input field for the speed setpoint
Switch	InputMode_Switch	Mode selection for motor Text ON: Auto Text OFF: Man
Button	ApplyValue_Button	When the button is pressed, the speed setpoint and mode are transferred to the corresponding tags.

Result

The objects required in the faceplate type have been configured.

See also

Creating a faceplate type (Page 107)

1.4.5.3 Example: Configuring tags in the faceplate type (RT Uni)

Task

You configure tags in the faceplate type with which you can dynamically control properties and which are required for data exchange in the project.

Configuring tags

Note

Tags are required to exchange values between the faceplate instance and project.

When used in screens, access to individual object properties in faceplates is possible exclusively via faceplate tags.

If you follow the example with PLC data types, you use the tags from the PLC data type accordingly.

1. Switch to the "Interface tags" tab in the "Unified faceplate types" editor.
2. Configure the required tags. To do this, click the ▤ icon.
   The required tags are listed in the table below.
3. Switch to the "Visualization" tab in the "Unified faceplate types" editor.

Tags in the faceplate type

Tag	Data type	Meaning	Source	Target
rpmInputTag	Int	Speed setpoint	Input field (Faceplate)	Project
rpmCurrTag	Int	Current speed value (actual value)	Project	Gauge (Faceplate)
engineNameTag	WString	Motor name (verbal identification)	Project	Text field (Faceplate)
modeSwitchTag	Bool	Mode (automatic/manual)	Switch (Faceplate)	Project

Assigning tags

1. Select the pointer instrument "Revolutions_Gauge".
2. Open the shortcut menu with a right-click and select "Properties".

3. Set the "tag" value under "General > Process value" in the "Dynamization" column.
   A screen for selecting the tag is shown on the right-hand side of the Inspector window.
4. Assign the "rpmCurrTag" tag to the "Process value" property.
   When the "rpmCurrTag" tag is linked in the project, the pointer instrument shows the corresponding values.

   Note

   All other tags in the faceplate type are later used in the script and linked in the project.

   If you follow the example with PLC data types, you assign the tags from the PLC data type accordingly.

See also

Configuring tags in the faceplate type (Page 114)

Instead of tags: Using the user data type (UDT) in the faceplate type (Page 123)

1.4.5.4 Instead of tags: Using the user data type (UDT) in the faceplate type (RT Uni)

Introduction

You have configured multiple motors of the same type. To do so, use data blocks based on a user data type (UDT).

You also want to use this user data type in the faceplate type, which displays some motor data and allows data input.

The following advantages arise from reusing the user data type:

- You minimize the configuration effort.
- You reduce the consumption of resources.
- They ensure unique and consistent naming of tags in data blocks and faceplates and hereby significantly reduce the probability of configuration errors.

Task

You use the existing user data type in the faceplate type.

Then link the tags of the data blocks defined in the user data type with those in the faceplate instance.

Requirement

- A controller is configured that supports user data types, e.g. from the SIMATIC S7-1200 or SIMATIC S7-1500 series.
- A suitable user data type for the motor type is configured.

- A data block is configured for each motor based on the user data type.
- The "Unified Faceplate Types" is opened with the created faceplate type.

Procedure

1. Adapt the faceplate type so that the user data type for the motors is also used in this faceplate type.
   - Switch to the "Interface tags" view.
   - You use the defined user data type (UDT) instead of the tag.
     The user data type now supplies the faceplate type with the corresponding tags.
   - Go back to the "Visualization" view.
   - Take into account that tag access in the faceplate type now takes place via the user data type.
     This affects, for example, the use of faceplate tags in dynamized object properties and in local scripts.
2. From the faceplate type, create the same number of faceplate instances as the number of motors you have configured.
3. Link the tags from the data blocks and faceplate instances with each other.
   - The tag names in data blocks and faceplate instance are hierarchically structured.
   - This means that the tags from the individual faceplate instances can always be assigned uniquely to the tags of the respective data block.
   - If the user data type contains tags that you do not require in the visualization, ignore these tags.

See also

Configuring tags in the faceplate type (Page 114)

1.4.5.5 Example: Configuring interface properties in the faceplate type (RT Uni)

Introduction

You configure interface properties in the faceplate type with which some properties can be dynamically controlled. Interface properties allow you to use different colors and resource lists in each faceplate instance.

Task

Design the background color of the EngineName_Textbox text box with interface properties.

Configure interface property

1. Switch to "Interface properties" tab in the "Unified faceplate types" editor.
2. Click the icon.
3. Assign the name "backEngineName".
4. Switch to the "Visualization" tab in the "Unified faceplate types" editor.

Assign interface property

1. Select the "EngineName_Textbox" text box.
2. Open the shortcut menu with a right-click and select "Properties".
3. Set the "Interface properties" under "Appearance > Background color" in the "Dynamization" column.
   A screen for selecting the interface property is shown on the right side of the Inspector window.
4. Assign the "backEngineName" interface property to the "Background color" property.
   If the value of the "backEngineName" interface property is defined for the faceplate instance, the background of the "EngineName_Textbox" text box is displayed in the corresponding color in runtime.

1.4.5.6 Example: Link faceplate type to plant object type (RT Uni)

Introduction

If you link the faceplate type to a plant object type, connect the tags of the faceplate type directly to the interface tags of the plant object type. Each time a faceplate instance is used, the tags of the linked plant object are accessed directly. This saves you configuration work when creating faceplate instances.

Task

You link the faceplate type with the plant object type.

Requirement

- A plant object type has been created.

Create interface tags at the plant object type

1. Open the editor of the plant object type.
2. Switch to the "Interface" tab.
3. Create the following project tags.

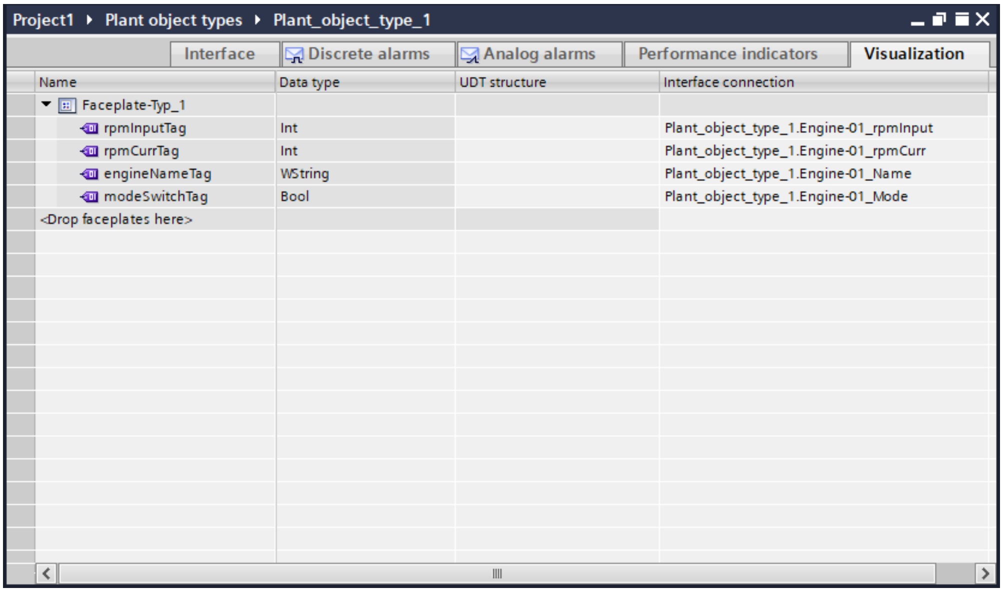
Faceplate tag	Project tag	Type	Meaning and function
engineNameTag	Engine-01_Mode	Bool	Set mode (manual/automatic) of the motor Transfer the set mode from the faceplate instance to the project
modeSwitchTag	Engine-01_Name	WString	Name of the motor (e.g. motor number or similar) Transfer the motor name from the project to the text box of the faceplate instance.
rpmCurrTag	Engine-01_rpmCurr	Int	Current speed of the motor Transfer the measured speed from the project to the gauge in the faceplate instance.
rpmInputTag	Engine-01_rpmInput	Int	Default value for speed Transfer the preset speed from the faceplate instance to the project.

Note

If you follow the example with PLC data types (UDT), note the deviating tag names.

Link faceplate type tags to project tags

1. Switch to the "Visualization" tab of the plant object type.
2. Drag-and-drop the faceplate type from the project tree into the work area to "<Drop faceplates here>".
3. Expand the faceplate type.
4. Under "Interface connection", connect the interface tags of the plant object type to the tags of the faceplate type.



Name	Data type	UDT structure	Interface connection
Faceplate-Typ_1			
rpmInputTag	Int		Plant_object_type_1.Engine-01_rpmInput
rpmCurrTag	Int		Plant_object_type_1.Engine-01_rpmCurr
engineNameTag	WString		Plant_object_type_1.Engine-01_Name
modeSwitchTag	Bool		Plant_object_type_1.Engine-01_Mode
<Drop faceplates here>			

Result

You have linked the faceplate type to the plant object type and connected all tags.

1.4.5.7 Example: Creating a local script in the faceplate type (RT Uni)

Task

You create a script that is triggered when a button is pressed and transfers values to tags.

Procedure

1. In the "Unified Faceplate Types" editor, go to the "Visualization" view.
2. Select the button.
3. Open the shortcut menu with a right-click and select "Properties".
   The properties of the button are displayed in the Inspector window.
4. Click "Events".
5. Select the "Click left mouse button" entry.
6. Click on the "Convert function to script" button.
   The script is created and displayed on the right-hand side of the Inspector window.
7. Write the code with which the values of the input field and of the switch are assigned to the corresponding tags.

   Note

   If you follow the example with user data types (UDTs), note the deviating tag names.

Sample code

```
export function ApplyValue_Button_OnTapped(item, x, y, modifiers, trigger)
{
   let rpm = Tags('rpmInputTag');
   rpm1 = Faceplate.Items('Revs_InputField');
   rpm.Write(rpm1.ProcessValue');
   //write value from Input Field to tag 'rpm'

   let mode = Tags('modeSwitchTag');
   let mode1 = Faceplate.Items('InputMode_Switch');
   mode.Write(mode1.IsAlternateState);
   //write value from Switch to tag 'mode'
}
```

Result

The script has been created in the faceplate type.

If you use the faceplate type in a screen, you assign a tag from the project to the tags. The values of these tags are given as new value to the assigned tags of the faceplate type. In this way, you supply values to the tags of the faceplate type.

See also

Configuring an event in the faceplate type (Page 115)

Configuring faceplate scripts (Page 119)

1.4.5.8 Example: Creating a faceplate instance and integrating it in the project (RT Uni)

Task

You insert an instance of the faceplate type in a screen and link the tags of the faceplate instance with tags in the project.

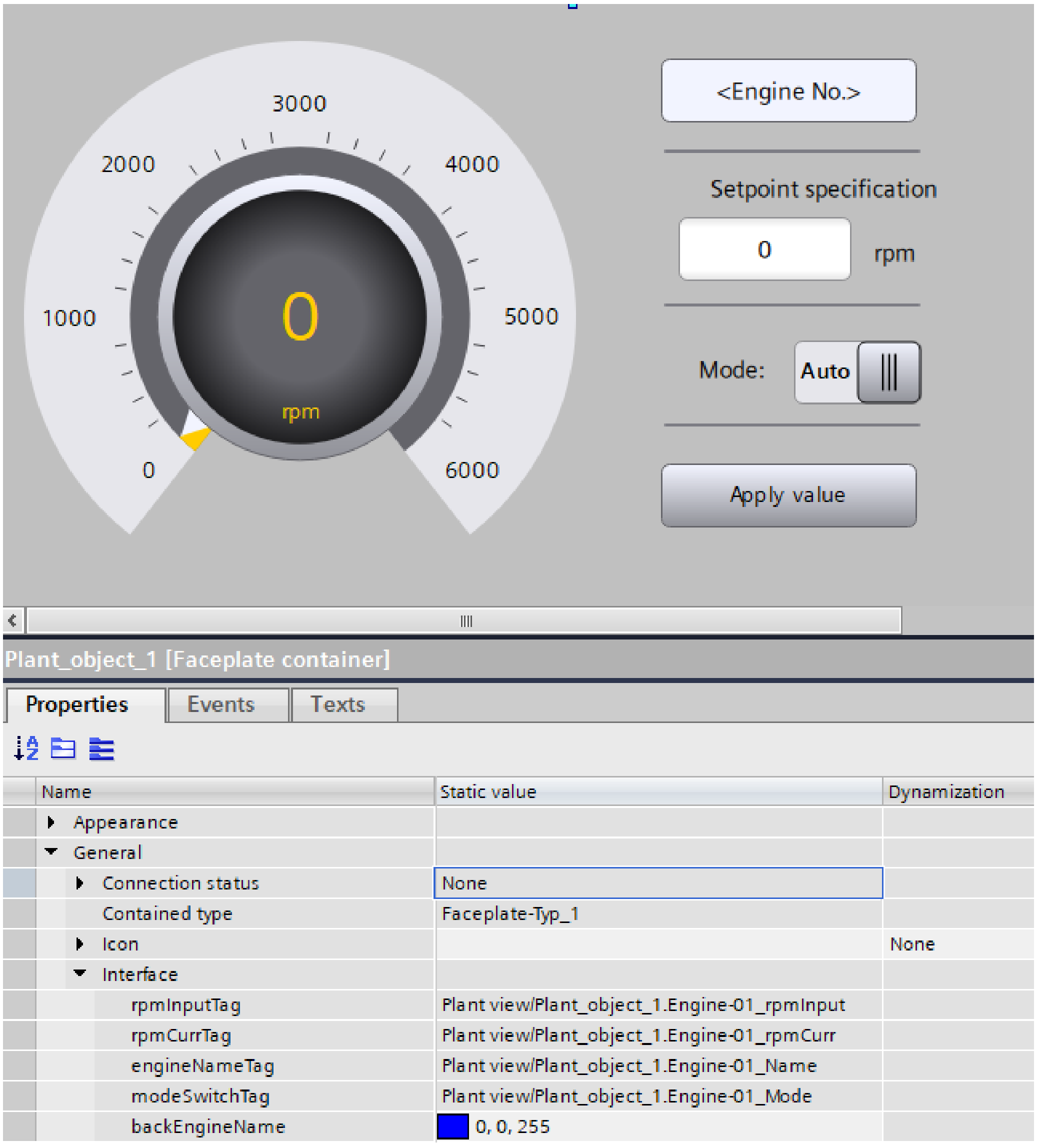
Requirement

- SIMATIC PC station - WinCC Unified SCADA RT has been configured.
- The HMI device is assigned in the plant view.
- The plant object type is used in the plant view.
- A new screen has been created.
- The "Screens" editor is open.

Creating a faceplate instance

1. Switch to the "Plant objects" tab in the project tree.
2. Drag-and-drop the faceplate type to the "Screens" editor.
   A faceplate container is created with an instance of the faceplate type.
3. Select the faceplate container and open its properties via the shortcut menu.

4. Define the properties of the faceplate container.
5. Assign a value for the interface property "backEngineName" (see below).



Name	Static value	Dynamization
Appearance		
General		
Connection status	None	
Contained type	Faceplate-Typ_1	
Icon		None
Interface		
rpmInputTag	Plant view/Plant_object_1.Engine-01_rpmInput	
rpmCurrTag	Plant view/Plant_object_1.Engine-01_rpmCurr	
engineNameTag	Plant view/Plant_object_1.Engine-01_Name	
modeSwitchTag	Plant view/Plant_object_1.Engine-01_Mode	
backEngineName	🟦 0, 0, 255	

Result

The faceplate type is instantiated on a screen in a faceplate container.

You have assigned a value to the background color property of the "EngineName_Textbox" text box.

This completes integration of the faceplate type in the project.

See also

Creating a faceplate instance (Page 109)

Configuring dynamization (RT Uni) 2

2.1 Basics of dynamizing screens (RT Uni)

Dynamizing objects

Dynamics are used to change the properties of screen objects and screens in runtime depending on another value. The source for this value changes is referred to as "Dynamization type".

Dynamization types

The following table shows the dynamization types available in WinCC:

Dynamization type	Description	Supported property classes	Examples
Tag	Defines the property value depending on the tag value.	All	"Process value" or "Left" properties
Script	Defines the property value depending on the return value.	All	"Process value" or "Left" properties
Resource list	Defines the property value depending on an entry from a text list or graphic list.	Text / Graphic	Properties "Text", "Tooltip" or "Graphic".
Flashing	Defines that the property flashes in configurable colors.	Colors	Properties "Foreground color" or "Border color".

Examples of dynamizations

The table below shows typical application examples for each type of dynamization:

Dynamization type	Application example
Tag	Visualize the level. The "Process value" property of a bar graph is dynamized with a tag that contains the level from the PLC.
Script	Simulate the filling process. To simulate a movement of bottles on a conveyor belt, the properties "Left", "Top" and "Visible" are dynamized with scripts.
Resource list	Display the plant status. The meaning of a quality code is saved in a text list. Depending on the transferred numerical quality code, its meaning is displayed on the HMI device.
Flashing	Visualize limit violations. When the level of a tank drops below a limit, the visualized tank is to flash in two signal colors.

See also

Dynamizing an object property with a "Script" (Page 132)

Dynamizing an object property with a "Tag" (Page 133)

Dynamizing an object property with "Flashing" (Page 134)

Dynamizing an object property with a "Resource list" (Page 134)

2.2 Dynamizing an object property with a "Script" (RT Uni)

Requirement

- A screen is open.
- An object is configured.
- The object property supports the dynamization type "Script".

Procedure

To dynamize an object property using a "Script", follow these steps:

1. Select the object.
2. Under "Properties > Properties > Dynamization" select the object property in the Inspector window.
   - Select the option "Script".
   - Write the code.
   - Select the trigger that triggers the dynamization in runtime.
3. If necessary, create a "Global definition".
   - Select a dynamized property.
   - Click "Global definition".
   - Write the code.
   - Select the trigger that triggers the dynamization in runtime.

Result

The object property is dynamized with a script. The return value of the script specifies the property value in runtime.

Note

The dynamization of an event is only monitored regarding an operator authorization if the triggering event, e.g. "Press button", is triggered by a user.

See also

Basics of dynamizing screens (Page 131)

Dynamizing an object property with a "Tag" (Page 133)

Dynamizing an object property with "Flashing" (Page 134)

Dynamizing an object property with a "Resource list" (Page 134)

2.3 Dynamizing an object property with a "Tag" (RT Uni)

Requirement

- A screen is open.
- An object is configured.
- You have configured a tag.
- The object property supports the dynamization type "Tag".

Procedure

To dynamize an object property using a tag, follow these steps:

1. Select the object.
2. Under "Properties > Properties > Dynamization" select the object property in the Inspector window.
3. Select the "Tag" option.
4. Select the tag.
5. If required, enable the options "Indirect addressing" or "Read only" in the "Settings" area.
6. Define the type of tag:
   - None
   - Area
     Define the conditions for the respective property.

Result

The object property tag is dynamized with a tag. The tag value specifies the property value in runtime.

See also

Basics of dynamizing screens (Page 131)

Dynamizing an object property with a "Script" (Page 132)

Dynamizing an object property with "Flashing" (Page 134)

Dynamizing an object property with a "Resource list" (Page 134)

2.4 Dynamizing an object property with "Flashing" (RT Uni)

Requirement

- A screen is open.
- An object is configured.
- The object property supports the dynamization type "Flashing".

Procedure

To dynamize an object property using "Flashing", follow these steps:

1. Select the object.
2. Under "Properties > Properties > Dynamization" select the object property in the Inspector window.
   - Select the "Flashing" option.
   - Select the flash colors.
   - Select the condition that triggers flashing.
   - Select the flashing frequency.

Result

The object property is dynamized. When the configured condition occurs in runtime, the object property flashes in the configured colors.

See also

Basics of dynamizing screens (Page 131)

Dynamizing an object property with a "Script" (Page 132)

Dynamizing an object property with a "Tag" (Page 133)

Dynamizing an object property with a "Resource list" (Page 134)

2.5 Dynamizing an object property with a "Resource list" (RT Uni)

Requirement

- A screen is open.
- An object is configured.
- A tag is configured.

- A text list or graphic list is configured.
- The object property supports the dynamization type "Resource list".

Procedure

To dynamize an object property using a "Resource list", follow these steps:

1. Select the object.
2. Under "Properties > Properties > Dynamization" select the object property in the Inspector window.
   – Select the "Resource list" option.
   – Select the tag.
   – Select the text list or graphic list.

Result

The object property tag is dynamized with a resource list. The tag value specifies the entry from the configured text list or graphic list that is displayed in runtime.

See also

Basics of dynamizing screens (Page 131)

Dynamizing an object property with a "Script" (Page 132)

Dynamizing an object property with a "Tag" (Page 133)

Dynamizing an object property with "Flashing" (Page 134)

Configuring tags (RT Uni)

3

3.1 Basics (RT Uni)

3.1.1 Basics of tags (RT Uni)

Introduction

Process values are forwarded in runtime using tags. Process values are data which is stored in the memory of one of the connected automation systems. They represent the status of a plant in the form of temperatures, fill levels or switching states, for example. Define external tags for processing the process values in WinCC.

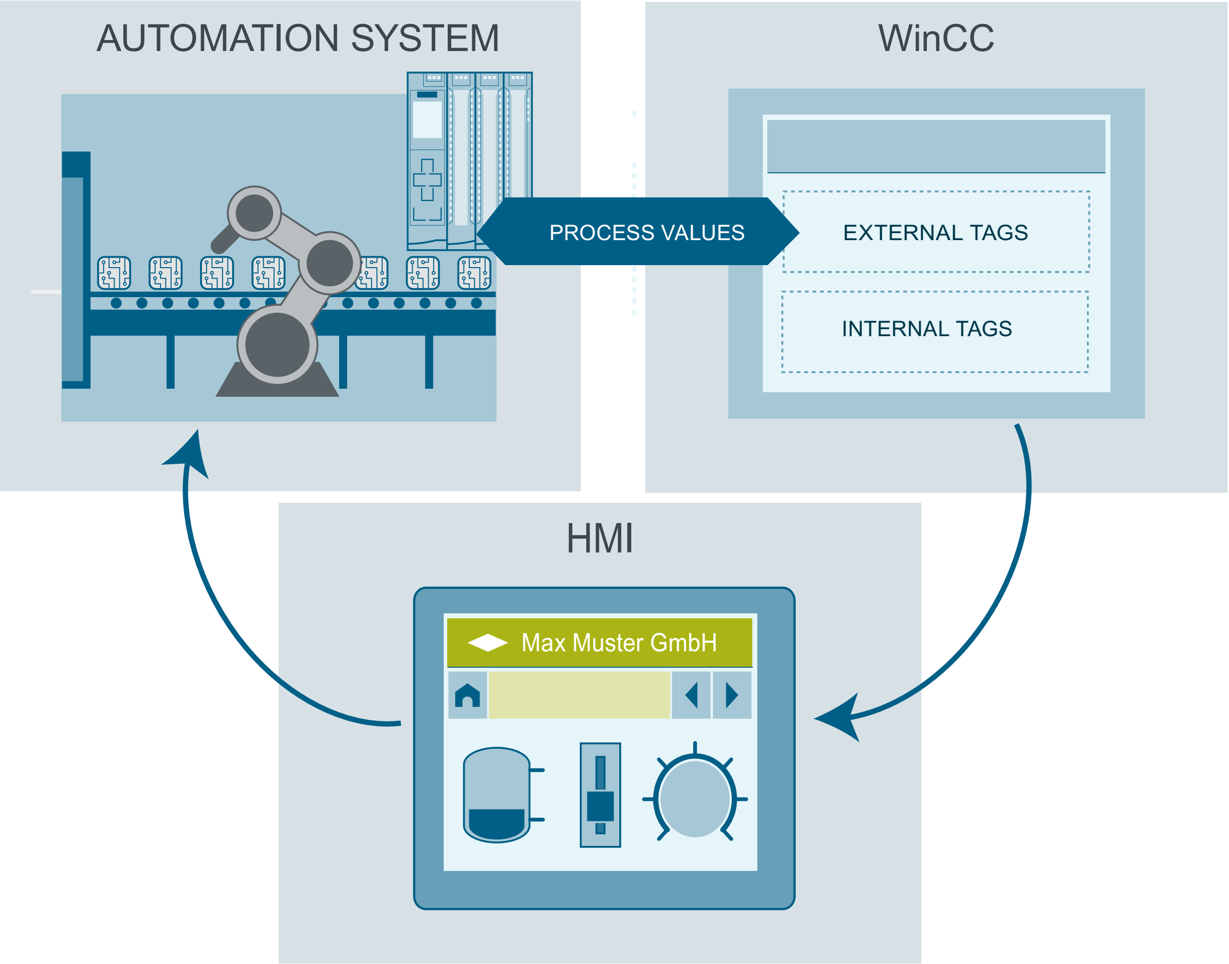
WinCC works with two types of tag:

- External tags
- Internal tags

The external tags form the link between WinCC and the automation systems. The values of external tags correspond to the process values from the memory of an automation system. The value of an external tag is determined by reading the process value from the memory of the automation system. It is also possible to rewrite a process value in the memory of the automation system.

Internal tags do not have a process link and only convey values within the WinCC. The tag values are only available as long as runtime is running.

In WinCC, you can visualize and change process values, which are transferred using tags, on your HMI device.



Tags in WinCC

For external tags, the properties of the tag are used to define the connection that the WinCC uses to communicate with the automation system and form of data exchange.

Tags that are not supplied with values by the process - the internal tags - are not connected to the automation system. In the tag's "Connection" property, this is identified by the "Internal tag" entry.

You can create tags in different tag tables for greater clarity. You then directly access the individual tag tables in the "HMI tags" node in the project tree. The tags from all tag tables can be displayed with the help of the table "Show all tags".

With structures you bundle a number of different tags that form one logical unit. Structures are project-associated data and are available for all HMI devices of the project. You use the "Types" editor in the project library to create and edit a structure.

See also

Overview of HMI tag tables (Page 139)

External tags (Page 140)

Internal tags (Page 144)

3.1.2 Overview of HMI tag tables (RT Uni)

Introduction

HMI tag tables contain the definitions of the HMI tags that apply across all devices. A tag table is created automatically for each HMI device created in the project.

In the project tree there is an "HMI tags" folder for each HMI device. The following tables can be contained in this folder:

- Default tag table
- User-defined tag tables
- Table of all tags

In the project tree you can create additional tag tables in the "HMI tags" folder and use these to sort and group tags and constants. You can move tags to a different tag table using a drag-and-drop operation or with the help of the "Tag table" field. Activate the "Tags table" field using the shortcut menu of the column headings.

In WinCC you can display the locations of use for all tags. Use the "Cross-references" command in the shortcut menu or the F11 key to call the "Cross-references" editor for a selected tag table. In the editor you can see all objects that the respective tag uses and you can jump directly to the location of use of the tag.

Default tag table

There is one default tag table for each HMI device of the project. It cannot be deleted or moved. The default tag table contains HMI tags and, depending on the HMI device, also system tags. You can declare all HMI tags in the standard tag table or, as necessary, in additional user-defined tag tables.

User-defined tag tables

You can create multiple user-defined tag tables for each HMI device in order to group tags according to your requirements. You can rename, gather into groups, or delete user-defined tag tables. To group tag tables, create additional subfolders in the HMI tags folder.

Show all tags

The "HMI tags" tab in the "All tags" table shows an overview of all HMI tags and system tags of the HMI device in question. This table cannot be deleted, renamed or moved. The "Tags table" column shows you in which tag table a tag is included. Using the "Tags table" field, the assignment of a tag to a tags table can be changed.

The "All tags" table contains an additional tab "System tags". The system tags are created by the system and used for internal management of the project. The names of the system tags begin with the "@" character. System tags cannot be deleted or renamed. You can evaluate the value of a system tag, but cannot modify it.

Discrete alarms, analog alarms and logging tags

The following tables are also available in an HMI tag table:

- Discrete alarms
  In the "Discrete alarms" table, you configure discrete alarms to the HMI tag selected in the HMI tag table. When you configure a discrete alarm, multiple selection in the HMI tag table is not possible. You configure the discrete alarms for each HMI tag separately.
- Analog alarms
  In the "Analog alarms" table, you configure analog alarms to the HMI tag selected in the HMI tag table. When you configure an analog alarm, multiple selection in the HMI tag table is not possible. You configure the analog alarms for each HMI tag separately.
- Logging tags
  In the "Logging tags" table, you configure logging tags to the HMI tag selected in the HMI tag table. When you configure a logging tag, multiple selection in the HMI tag table is not possible. You configure the logging tags for each HMI tag separately.

With the help of these tables you configure alarms and logging tags for the currently selected HMI tag.

3.1.3 External tags (RT Uni)

Introduction

External tags allow the data exchange between the components of an automation system, for example, between an HMI device and a PLC.

An external tag is the image of a defined memory location in the PLC. You have read and write access to this storage location from both the HMI device and from the PLC.

As external tags map a storage location in the PLC, the applicable data types depend on the PLC that is connected to the HMI device.

If you write a PLC control program in STEP 7, the PLC tags created in the control program will be added to the PLC tag table. If you want to connect an external tag to a PLC tag, access the PLC tags directly via the PLC tag table and connect them to the external tag.

Data types

All simple data types available on the connected PLC are available at an external tag in WinCC. If the data type of the PLC tag is not available in WinCC, a compatible data type is automatically used at the HMI tag. Interconnected PLC data types and arrays are not supported.

If you use PLC data types, the data type is adopted by WinCC. You can change the data type at the HMI tag, if necessary.

Central tag management in STEP 7

You can connect also connect DB instances of user-defined PLC data types (UDT) to the HMI tags.

The PLC data type and the corresponding DB instances are created and updated centrally in STEP 7. In WinCC, you can use the following sources as the PLC tag (DB instances):

- Data block elements that use a UDT as data type
- Data block instances of a UDT

The data type is taken from STEP 7 and is not converted to an HMI data type. The access type is always "Symbolic access". Depending on the release for WinCC in STEP 7, elements and structured elements of the PLC data type are applied to WinCC.

Note

Accessing PLC data types

Access to PLC data types is only available in conjunction with SIMATIC S7-1500.

Synchronization with PLC tags

A variety of options for synchronizing external tags with the PLC tags are available in the runtime settings under "Settings for tags".

When you perform synchronization, you have the option of automatically applying the tag names of the PLC to external tags and reconnecting the existing tags.

The generated tag name is derived from the position of the data value in the hierarchical structure of the data block.

Update of tag values

For external tags, the current tag values are transmitted in runtime via the communication connection between WinCC and the connected automation systems and then saved in the runtime memory. Next, the tag value will be updated to the set cycle time. For use in the runtime project, WinCC accesses tag values in the runtime memory that were read from the PLC at the previous cycle time. As a result, the value in the PLC can already change while the value from the runtime memory is being processed.

See also

Creating external tags (Page 152)

3.1.4 Addressing external tags (RT Uni)

Introduction

The options for addressing external tags depend on the type of connection between WinCC and the PLC in question. A distinction must be made between the following connection types:

- Integrated connection
  Connections of devices which are within a project and were created with the "Devices & Networks" editor are referred to as integrated connections.
- Non-integrated connection
  Connections of devices which were created with the "Connections" editor are referred to as non-integrated connections. It is not necessary that all of the devices be within a single project.

The connection type can also be recognized by its icon.

🖧	Integrated connection
⇄	Non-integrated connection

Addressing with integrated connections

An integrated connection offers the advantage that you can address a tag both symbolically and absolutely.

For symbolic addressing, you select the PLC tag via its name and connect it to the HMI tag. The valid data type for the HMI tag is automatically selected by the system.

During the symbolic addressing of a data block with optimized access and standard access, the address of an element in the data block is dynamically assigned and is automatically adopted in the HMI tag in the event of a change. You do not need to compile the connected data block or the WinCC project for this step.
For data blocks with optimized access, only symbolic addressing is available.

For symbolic addressing of elements in a data block, you only need to recompile and reload the WinCC project in case of the following changes:

- If the name or the data type of the linked data block element or global PLC tag has changed.
- If the name or the data type of the higher level structure node of a linked element in the data block element or global PLC tag has changed.
- If the name of the connected data block has changed.

Symbolic addressing is currently available with the following PLCs:

- SIMATIC S7-1500
- SIMATIC ET 200 CPU
- SIMATIC S7-1500 software controller

Symbolic addressing is also available if you have an integrated link.

You can also use absolute addressing with an integrated connection. You have to use absolute addressing for PLC tags from a SIMATIC S7-300/400 PLC. If you have connected an HMI tag with a PLC tag and the address of the PLC tag changes, you only have to recompile the control program to update the new address in WinCC. Then you recompile the WinCC project and load it onto the HMI device.

In WinCC, symbolic addressing is the default method. To change the default setting, select the menu command "Options > Settings > Visualization > HMI tags".

The availability of an integrated connection depends on the PLC used. The following table shows the availability:

Controller	Integrated connection	Comments
S7-300/400	Yes	The linking of tags is not checked in runtime. If the tag address changes in the PLC and the HMI device is not compiled again and loaded, the change is not registered in runtime.
S7-1500	Yes	A validity check of the tag connection is performed in runtime during symbolic addressing. If an address is changed in the PLC, the change is registered and an error message is issued. In the case of absolute addressing, the behavior described for the S7-300/400 applies.
SIMATIC ET 200 CPU	Yes	A validity check of the tag connection is performed in runtime during symbolic addressing. If an address is changed in the PLC, the change is registered and an error message is issued. In the case of absolute addressing, the behavior described for the S7-300/400 applies.
SIMATIC S7-1500 software controller	Yes	A validity check of the tag connection is performed in runtime during symbolic addressing. If an address is changed in the PLC, the change is registered and an error message is issued. In the case of absolute addressing, the behavior described for the S7-300/400 applies.

Create an integrated connection in the "Devices & Networks" editor. If the PLC is contained in the project and integrated connections are supported, you can then also have the connection created automatically. To do this, when configuring the HMI tag, simply select an existing PLC tag to which you want to connect the HMI tag. The integrated connection is then automatically created by the system.

Addressing with non-integrated connections

In the case of a project with a non-integrated connection, you always configure a tag connection with absolute addressing. Select the valid data type yourself. If the address of a PLC tag changes in a project with a non-integrated connection during the course of the project, you also have to make the change in WinCC. The tag connection cannot be checked for validity in runtime, an error message is not issued.

A non-integrated connection is available for all supported PLCs.

Symbolic addressing is not available in a non-integrated connection.

With a non-integrated connection, the control program does not need to be part of the WinCC project. You can perform the configuration of the PLC and the WinCC project independently of each other. For configuration in WinCC, only the addresses used in the PLC and their function have to be known.

3.1.5 Internal tags (RT Uni)

Introduction

Internal tags do not have any connection to the PLC.

Principle

Internal tags are stored in the memory of the HMI device. Therefore, only this HMI device has read and write access to the internal tags. You can create internal tags to perform local calculations, for example.

You can use the HMI data types for internal tags. Availability depends on the HMI device being used.

The following HMI data types are available:

HMI data type	Data format
Bool	Binary tag
Byte	Unsigned 8-bit value
DateTime	Date/time format
DInt	Signed 32-bit value
DWord	Unsigned 32-bit value
Int	Signed 16-bit value
LInt	Signed 32-bit value
LReal	Floating-point number 64-bit IEEE 754
LTime	Signed duration
LWord	Unsigned 64-bit value
Real	Floating-point number 32-bit IEEE 754
SInt	Signed 8-bit value
UDInt	Unsigned 32-bit value
UInt	Unsigned 16-bit value
ULInt	Unsigned 64-bit value
USInt	Unsigned 8-bit value
WChar	Text tag, 16-bit character set
Word	Unsigned 16-bit value
WString	Text tag, 16-bit character set
Array	Data structure

System tags

System tags are required for internal management of the project. The names of these tags always start with the "@" character. You may not delete or rename these tags. You cannot change the value of a tag.

Note

You must not create any tags whose name starts with a @.

3.1.6 Updating the tag value in runtime (RT Uni)

Introduction

Tags contain process values which change while runtime is running. Value changes are handled differently at internal and external tags.

Principle

When runtime starts, the value of a tag is equal to its start value. Tag values change in runtime.

In runtime, you have the following options for changing the value of a tag:

- A value change in an external tag in the PLC.
- By input, for example, in an I/O field.
- A value assignment in a script.

Updating the value of external tags

The value of an external tag is updated as follows:

- Cyclic in operation
  If you select the "Cyclic in operation" acquisition mode, the tag is updated in runtime while it is displayed in a screen or is logged. The acquisition cycle determines the update cycle for tag value updates on the HMI device. You can either choose a default acquisition cycle or define a user-specific cycle.
- Cyclic continuous
  If you select the "Cyclic continuous" acquisition mode, the tag will be updated continuously in runtime, even if it is not in the currently-open screen. This setting is activated for tags that are configured to trigger a function list when their value changes, for example.
  Only use the "Cyclic continuous" setting for tags that must truly be updated. Frequent read operations increase communication load.
- On demand
  If you select the "On demand" acquisition mode, the tag is not updated cyclically. It will only be updated on demand using the "UpdateTag" system function, for example, or by a script.

See also

Defining the acquisition cycle for a tag (Page 159)

3.1.7 Limits and start values of a tag (RT Uni)

Introduction

You can configure start values and restrict the value ranges with limits for numerical tags.

Use the limits to warn the operator when the value of a tag enters a critical range, for example.

Use the start values to assign a default value to an I/O field that is specified as start value in the linked tag.

Tags limits

You can specify a value range defined by a high limit and a low limit for numerical tags.

You configure four limit values that limit the value range. Using the limits Upper 2 and Lower 2 , you specify the maximum and minimum value for the value range. The limits Upper 1 and Lower 1 specify the threshold values at which the normal range is exceeded or undershot.

Limit	Application
Upper 2	Specifies the maximum value.
Lower 2	Specifies the minimum value.

If the operator enters a value for the tag that is outside the configured value range, the input is rejected. When the tag value leaves the value range, the function list is processed.

Note

If you want to output an analog alarm when a limit is violated, configure the respective tag in the "Analog alarms" tab. You can also configure the analog alarm in the "HMI alarms" editor. The values for output of an analog alarm depend on the configured tag limits.

Start value of a tag

You can configure a start value for numeric tags and tags for date/time values. The tag will be preset to this value when runtime starts. In this way, you can ensure that the tag has a defined status when runtime starts.

The start value cannot have the data type Raw or TextRef.

For external tags, the start value will be displayed on the HMI device until it is overwritten by the PLC or by input.

If no start value was configured, the tag contains the value "0" when starting runtime.

Use the "Persistence" setting to specify whether the value of the tag is to be retained when runtime is closed. The value saved will be used as the start value when you restart runtime.

See also

Defining limits for a tag (Page 160)

3.1.8 Data logging (RT Uni)

Introduction

Data logging is used to collect, process and log process data from an industrial system. When you analyze the logged process data, you can extract important business and technical information regarding the operational state of the system.

The process values to be logged are compiled, processed and saved in the log database in runtime. Current or previously logged process values can be output in runtime as a table or trend. In addition, it is possible to print out logged process values as a report.

Use

You can use data logging for the following tasks:

- Early detection of danger / fault states
- Increase of productivity
- Enhancement of product quality
- Optimization of maintenance cycles
- Documentation of process value trends

Configuration

You configure the logging of process values in the "Logs" editor. You create a data log and an alarm log. The data log stores process values in logging tags. When you configure the data log, select the storage location, the logging period and the size of the log. You also specify the settings for the logging segments.

You configure trend views and process controls for displaying process data in runtime in the "Screens" editor. These views allow you to output the process data in the form of trends and tables.

Logging tags

You can log the values of internal and external tags. Use the logging tags for logging tag values. In logging tags, you specify how the values of the corresponding tags are written to the log.

You can create a logging tag for each HMI tag in the tag tables. You define the logging tags in the "HMI Tags" editor under "Logging tags". You specify for each logging tag to which log the tag is written.

With the default logging type "On change", the process value is compared to the saved value and the new value is only written to the log if the process value has changed.

To preserve memory, you can activate smoothing for the logging tags. By doing so, insignificantly small changes are filtered out prior to writing and the number of logged values is reduced.

You create logging tags for internal and external tags. When logging the PLC tag values, the time stamp contains the time at which the value occurred in the PLC.

Note

Supported data type for logging external tags

When logging external PLC tags, all data types are supported except "Raw", "TextRef", "Struct" and "Array". The data type "TextRef" must not be used within the "Faceplate container" object.

3.1.9 Basics of tag management (RT Uni)

Basics of tag management

You can always rename, copy or delete tags.

When a tag is renamed, the new name must be unique for the whole device.

Note

The connection to the tags can be interrupted in runtime under the following conditions during renaming:

- HMI tag is used in a type, for example, to dynamize an object property via a script.
- One or more instances of the type are used in the project.
- Project is in runtime.
- After the renaming, execute the command "Compile > Software (only changes)".

Solution: Exit runtime and rename the tag. Execute the "Compile > Software (rebuild all)" command.

If you use the "Copy" command to copy a tag to the clipboard, the references are copied along with the tag.

If you use the "Insert" command to add a tag to another device, the tag will be added without the connected references. Only the object name of the reference will be inserted. If a reference of the same name and valid properties exists in the target system, the existing reference will then be connected to the copied tag.

If you copy a tag, some of the objects linked to the tag are copied as well. The following objects are copied:

- Logging tags
- Cycles
- Alarms

If you add the copied tag to another device, the tag is added together with the linked objects.

Before you delete a tag, check in the "Cross-references" editor where the tag is used and what impact the deletion of the tag will have on your project.

3.1.10 Basics of user data types (RT Uni)

Introduction

With user data types you bundle a number of different tags that form one logical unit. You create a user data type as a type and use instances of this type in the project. User data types are project-associated data and are available for all HMI devices of the project.

WinCC also supports the connection of PLC data types (UDTs) as user data types.

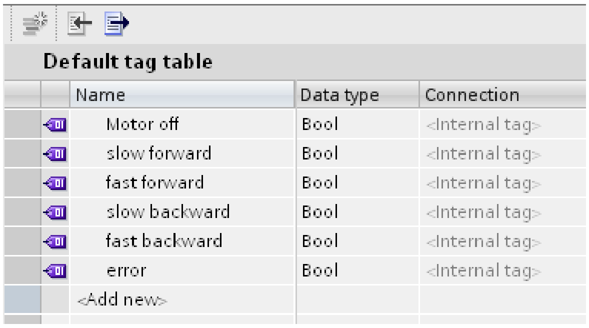
User data types also differ in their applicability with a specific communication driver. User data types are available for the following communication drivers:

- SIMATIC S7-300/400
- SIMATIC S7-1500

Create user data types and user data type elements in the project library.

Principle

For example, the different conditions of a motor can be described using 6 unique Boolean tags.



Default tag table

Name	Data type	Connection
Motor off	Bool	<Internal tag>
slow forward	Bool	<Internal tag>
fast forward	Bool	<Internal tag>
slow backward	Bool	<Internal tag>
fast backward	Bool	<Internal tag>
error	Bool	<Internal tag>
<Add new>		

If the overall condition should be described with a single tag, this tag can be created based on a user data type. For each of the individual Boolean tags you create a user data type element in the user data type.

This user data type can then be assigned complete to a faceplate for the motor. The created and released version of user data type is displayed at the tag in the "Data type" selection field.

The configured properties of a user data type are used in the instances of the user data type. If required, you change individual properties directly at the point of use, e.g. at a tag. Changing a property at the tag does not affect the user data type created.

3.1.11 Export and import of tags (RT Uni)

Introduction

With Export and Import, you have the option to export tags from one project and import them into another project. There is also the option to create larger numbers of tags outside of WinCC, edit them and subsequently import into any WinCC project.

You export and import tags with the "Import" and "Export" buttons in the "HMI Tags" editor. When importing the tag data to WinCC, pay attention to the structure required in the import file.

Tag data structure

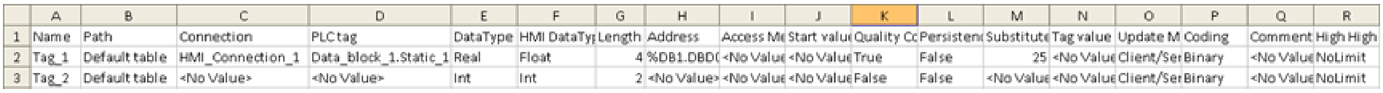
The tag data file must be in "*.xlsx" data format for the tag import and must be structured according to specific rules.

The import file in Microsoft Excel consists of a number of worksheets:

- HMI Tags (tags)
- SubstituteValueUsage (substitute value)

Each tag is on a separate row in the import file. The import file with the tag data must have the following format:

Example of the worksheet "HMI Tags"



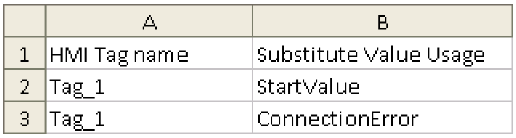
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Name	Path	Connection	PLC tag	DataType	HMI DataTyp	Length	Address	Access Me	Start valu	Quality Co	Persistenc	Substitute	Tag value	Update M	Coding	Comment	High High
2	Tag_1	Default table	HMI_Connection_1	Data_block_1.Static_1	Real	Float	4	%DB1.DBD(	<No Value	<No Value	True	False	25	<No Value	Client/Ser	Binary	<No Value	NoLimit
3	Tag_2	Default table	<No Value>	<No Value>	Int	Int	2	<No Value>	<No Value	<No Value	False	False	<No Value	<No Value	Client/Ser	Binary	<No Value	NoLimit

Table 3-1 Meaning of the entries

List entry	Meaning
Name	Indicates the configured name of an HMI tag.
Path	Specifies which folders in the project tree contain the tag. The folder structure is represented by ",": "FolderName1,FolderName2,TagName".
PLC Tag	Specifies which PLC tag is linked to the HMI tag.
Connection	Indicates the name of the connection.
Date type	Specifies the data type of the PLC tag. The data types allowed depend on the communication driver being used. See the "Communication" section of the documentation for additional information on the data types permitted for the various communication drivers.
HMI Data type	Specifies the data type of the HMI tag. The data types allowed depend on the communication driver being used. See the "Communication" section of the documentation for additional information on the data types permitted for the various communication drivers.
Length	Specifies the length of the tag in bytes. This entry is only useful for data types with a dynamic length, for example, strings. This entry remains empty for other data types.

List entry	Meaning
Access Method	Specifies the type of access.
Address	Specifies the tag address in the PLC. The tag address must exactly match the one used in WinCC, for example, "%DB1.DBW0". The tag address is empty for internal tags.
Start Value	Specifies the start value of a tag.
Quality code	Provides information on the quality of the connection.
Persistency	Indicates whether the value is to be retained after the end of runtime.
Substitute Value	Indicates the substitute value. The substitute value is used when a process value with errors is read.
ID tag	The update ID updates the value of a tag with the aid of a function or a PLC job. The update ID must be unique within an HMI device.
Update Mode	Indicates whether the tag is to be updated locally or for the entire project.
Acquisition mode	Indicates the acquisition mode.
Acquisition cycle	Indicates the acquisition cycle used.
Limit Upper 2 type	Indicates whether the limit value "Upper 2" is monitored by a constant or not at all.
Limit Upper 2	Displays the limit value "Upper 2".
Limit Lower 2 type	Indicates whether the limit value "Lower 2" is monitored by a constant or not at all.
Limit Lower 2	Displays the limit value "Lower 2".
End value PLC	Specifies the end value of the PLC tag.
Start value PLC	Specifies the start value of the PLC tag.
End value HMI	Specifies the end value of the HMI tag.
Start value HMI	Specifies the start value of the HMI tag.

Example of the worksheet "SubstituteValueUsage"



	A	B
1	HMI Tag name	Substitute Value Usage
2	Tag_1	StartValue
3	Tag_1	ConnectionError

Table 3-2 Meaning of the entries

List entry	Meaning
HMI Tag name	Specifies the tag for which a substitute value has been defined. The tag must be available in the "HMI-Tags" worksheet.
Substitute Value Usage	Indicates the substitute value. The substitute value can be used in the following situations: ● As start value ● After communication error ● Upper 2 limit value ● Lower 2 limit value

Note

"No value" in the table

Entries in the table which have the value "No value" delete the corresponding values in an existing tag of the same name.

See also

Importing and exporting tags (Page 162)

3.2 Configuring tags (RT Uni)

3.2.1 Creating external tags (RT Uni)

Introduction

You can access an address in the PLC via a PLC tag using an external tag. The following options are available for addressing:

- Symbolic addressing
- Absolute addressing

If possible, use symbolic addressing when configuring a tag. Symbolic addressing enables high-performance data access and is therefore less prone to errors. The system monitors the assignment of the storage address and the locations of use are automatically updated when changes occur.

You create tags either in the standard tag table or in a tag table you defined yourself.

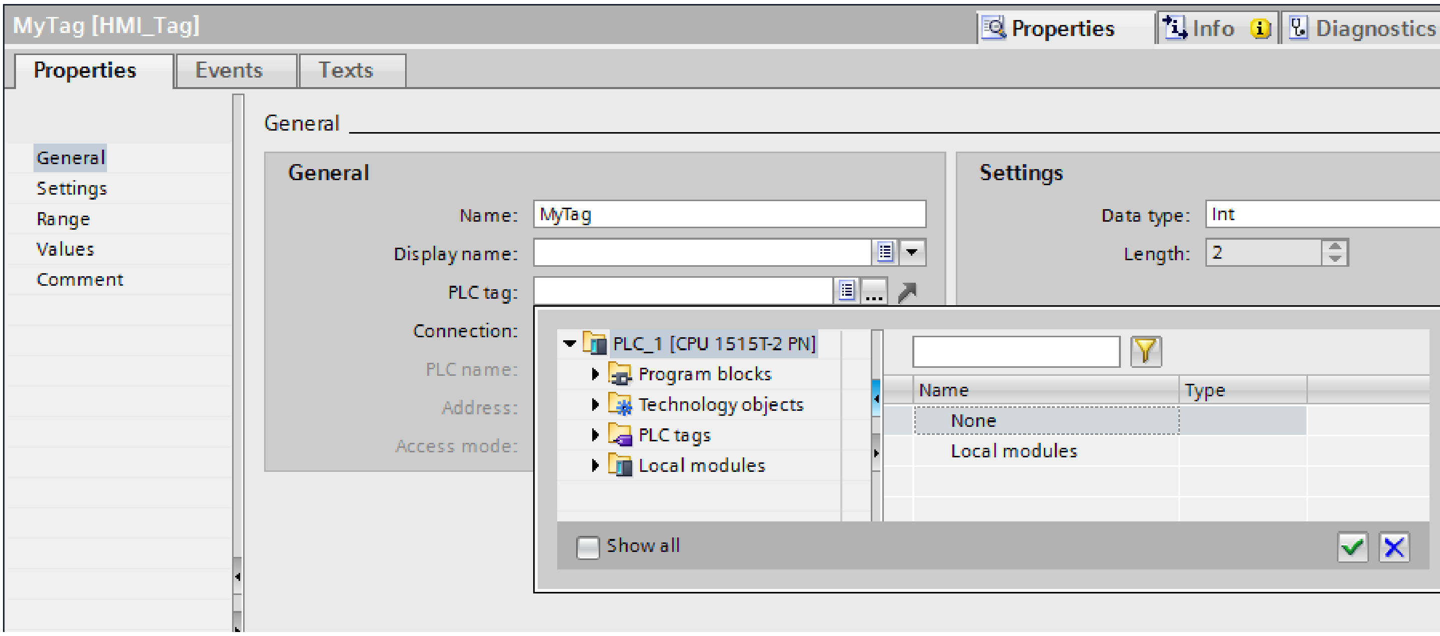
Requirement

- The project is open.
- A connection to the PLC is configured.
- The Inspector window is open.

Procedure

To create an external tag, proceed as follows:

1. Open the "HMI tags" folder in the project tree and double-click the standard tag table. The tag table opens.
   Alternatively, create and then open a new tag table.
2. In the "Name" column, double-click "Add" in the tag table.
   A new tag is created.
3. Select the "Properties > Properties >General" category in the Inspector window and, if required, enter a unique tag name in the "Name" field.
   The tag name must be unique throughout the device.
4. If required, select the "Display name" field to enter a name to be displayed in runtime.
5. Select the connection to the required PLC in the "Connection" box.
   If the connection you require is not displayed, you must first create the connection to the PLC. You create the connection to a SIMATIC S7 PLC in the "Devices & Networks" editor. You create the connection to external PLCs in the "Connections" editor.
   If the project contains the PLC and supports integrated connections, you can also have the connection created automatically. To do this, when configuring the HMI tag, simply select an existing PLC tag to which you want to connect the HMI tag. The integrated connection is then automatically created by the system.
6. If you are working with an integrated connection, click the … button in the "PLC tag" field and select an already created PLC tag in the object list. Click ✓ to confirm the selection.



Name: MyTag

Data type: Int

Length: 2

Alternatively, use the autocomplete to select a PLC tag for an integrated connection.
If you select a PLC tag from the autocomplete list, it is entered in the input path. The elements of the PLC tags are displayed in the autocomplete list. If you have selected a PLC tag that can be connected to the HMI tags, the PLC tag is connected to the HMI tags.

7. If you are working with a non-integrated connection, enter the address from the PLC in the "Address" field. To enter the address, make sure that the access mode "Absolute access" is configured.
   The "PLC tag" field remains empty.
8. Configure the other properties of the tag in the inspector window.
   You can also configure all tag properties directly in the tag table. To view hidden columns, activate the column titles using the shortcut menu.

Tips for effective procedure

- You also create new tags directly at the location of use, for example, on an I/O field. To do this, click "Add new" in the object list. You then configure the new tag in the Inspector window.
- You can also create external HMI tags by dragging and dropping data block elements or global PLC tags to an HMI tag table.

Result

An external tag has been created and linked to a PLC tag or an address in the PLC.

See also

Basics of tags (Page 137)

External tags (Page 140)

Addressing external tags (Page 142)

3.2.2 Creating internal tags (RT Uni)

Introduction

You must at least set the name and data type for internal tags. Select the "Internal tag" item, rather than a connection to a PLC.

For documentation purposes, it is a good idea to enter a comment for every tag.

Procedure

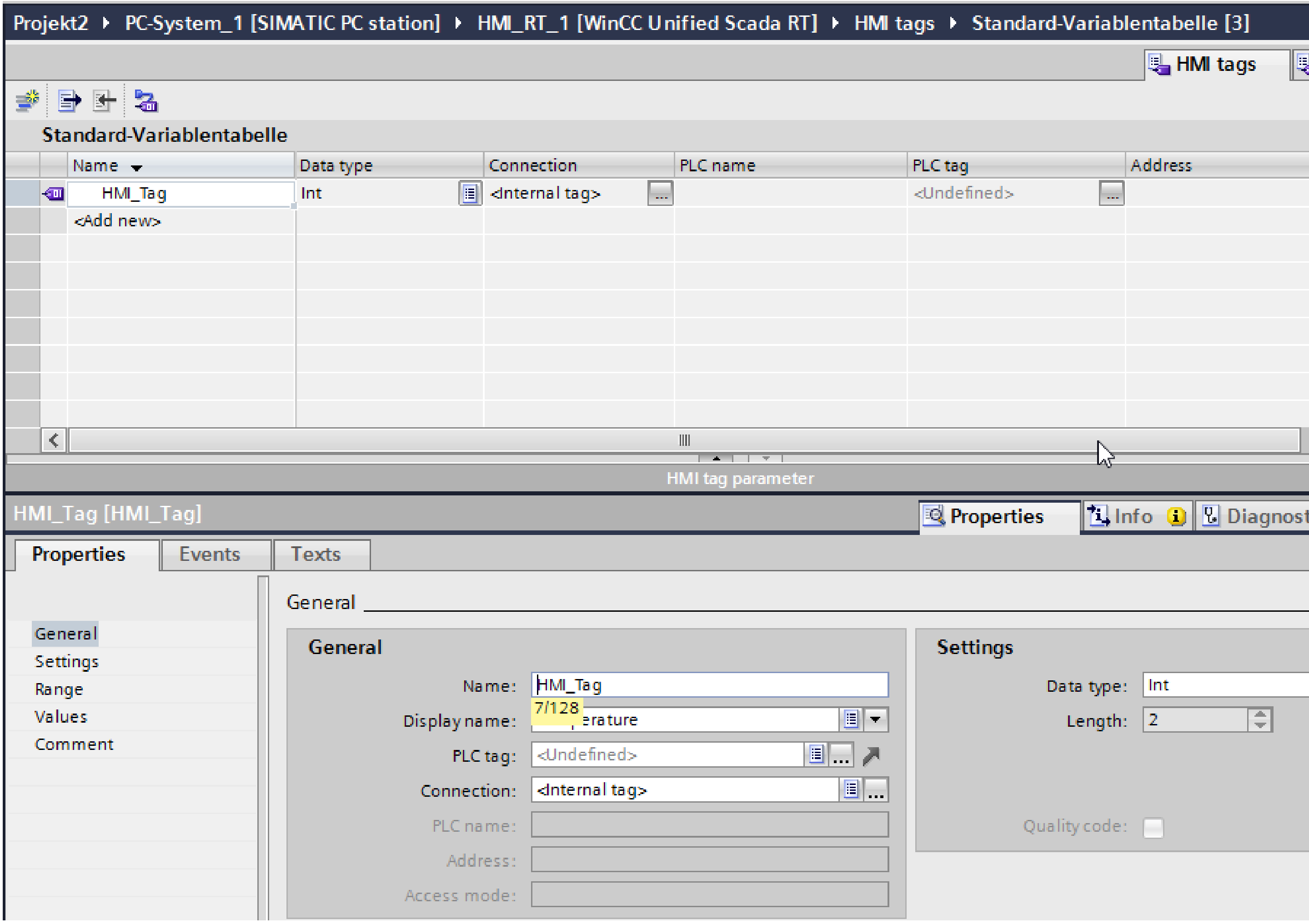
1. Open the "HMI tags" folder in the project tree and double-click the entry "Standard tag table".
   The tag table opens.
   Alternatively, create and then open a new tag table.
2. In the "Name" column, double-click "Add" in the tag table. A new tag is created.
3. If the Inspector window is not open, select the "Inspector window" option in the "View" menu.

4. Select the "Properties > Properties >General" category in the Inspector window and, if required, enter a unique tag name in the "Name" field.

   Note

   This tag name must be unique throughout the project. The tag name must not contain the special characters line feed, carriage return or quotation marks.

5. If required, select the "Display name" field to enter a name to be displayed in runtime. The name to be displayed is language-specific and can be translated for the required runtime languages. The display name is available for Basic Panels, Panels and Runtime Advanced.



Projekt2 ▸ PC-System_1 [SIMATIC PC station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ HMI tags ▸ Standard-Variablentabelle [3]

Standard-Variablentabelle

Name	Data type	Connection	PLC name	PLC tag	Address
HMI_Tag	Int	<Internal tag>		<Undefined>	
<Add new>					

HMI tag parameter

HMI_Tag [HMI_Tag]

General

Name: HMI_Tag
Display name: erature
PLC tag: <Undefined>
Connection: <Internal tag>
PLC name:
Address:
Access mode:

Settings

Data type: Int
Length: 2
Quality code: □

6. Select "Internal tag" as the connection in the "Connection" field.

7. Select the required data type in the "Data type" field.

8. In the "Length" field, you can specify the maximum number of characters to be stored in the tag in accordance with the selected data type.
   The length is automatically defined by the data type for numerical tags.

9. As an option, you can enter a comment regarding the use of the tag. To do so, click "Properties > Properties > Comment" in the Inspector window and enter a text.

Tips for effective configuring

You also configure the tag properties directly in the tag table. To view hidden columns, activate the column titles using the shortcut menu.

You also create new tags directly at the location of use, for example, on an I/O field. To do this, click "Add new" in the object list. You then configure the new tag in the Inspector window.

Result

An internal tag is created. You can now use this in your project.

In additional steps you can configure the tag, for example, by setting the start value and limits.

See also

Basics of tags (Page 137)

Internal tags (Page 144)

Configuring discrete alarms (Page 238)

Configuring analog alarms (Page 241)

3.2.3 Configuring multiple tags (RT Uni)

Introduction

In a tag table, you create a large number of identical tags by automatically filling the rows of the table below a tag. The tag names are incremented automatically when filling in automatically.

You can also use the auto fill function to fill table cells below a tag with a single tag property and thus modify the corresponding tags.

If you apply the automatic filling in to already filled cells of a tab table, you will be asked whether you want to overwrite the cells or insert new tags.

If you do not want to overwrite already configured tags, activate insert mode. Activate insert mode by keeping the <Ctrl> key pressed during insertion. Already existing entries in the tag table are moved down if insert mode is activated.

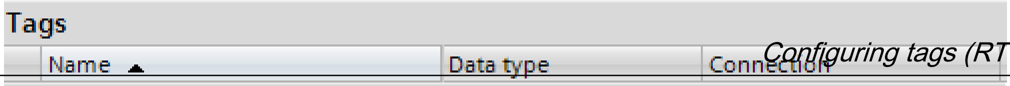
Tips for effective configuring

You can configure multiple tags simultaneously and use them in the screen. If you use drag and drop to drag multiple tags from the detail window to the screen, for example, for each tag an I/O field is created that is connected to the tag.

Requirement

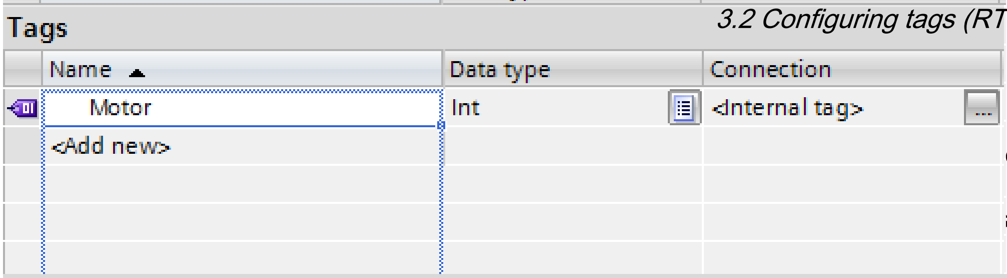
- The project is open.
- A tag table is open.
- The tag which is to serve as a template for other tags is configured.

Procedure



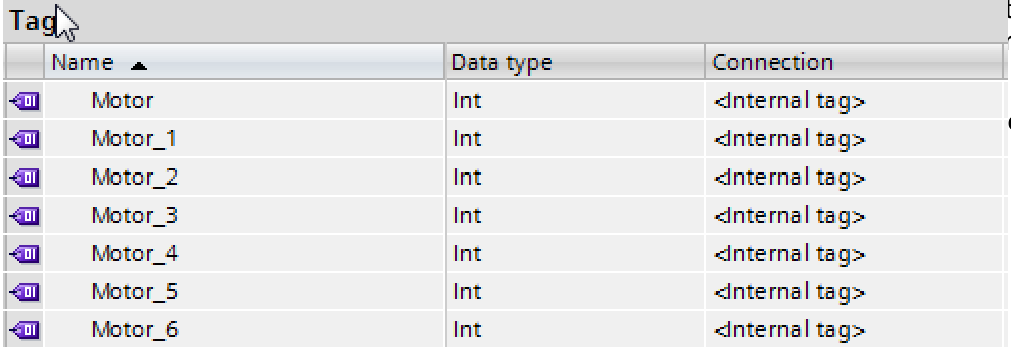
Tags

Name ▲	Data type	Connection



Tags

	Name ▲	Data type	Connection
	Motor	Int	<Internal tag>
	<Add new>		



Tags

	Name ▲	Data type	Connection
	Motor	Int	<Internal tag>
	Motor_1	Int	<Internal tag>
	Motor_2	Int	<Internal tag>
	Motor_3	Int	<Internal tag>
	Motor_4	Int	<Internal tag>
	Motor_5	Int	<Internal tag>
	Motor_6	Int	<Internal tag>

1. d as
 this
 ttom
 ross.
2. o fill
3.
4. In the tag table, select all the tags that you want to configure at the same time.
   If the selected property is identical for all the tags, the setting for this property will appear in the Inspector window.
   The associated field will remain blank otherwise.
5. You can define the shared properties in the Inspector window or directly in the tag table.

Result

Depending on which cells were selected, the function may automatically fill individual properties or create and configure new tags.

3.2.4 Adapting the data type of a tag (RT Uni)

Introduction

When you create a tag, you assign one of the possible data types to the tag. This data type depends on the type of data for which you would like to use the tag.

The data types available depend on the connected communication partner, such as a PLC.

Note

If you modify the data type of an existing, external tag, the previously defined tag address is identified as invalid. This reason for this is that the PLC address changes when the data type is modified.

Format adjustment

WinCC sets the data type of an external tag according to the data type of the connected PLC tag. If the data type of the PLC tag is not available in WinCC, a compatible data type is automatically used at the HMI tag. As required, you can specify that WinCC uses a different data type and converts the format of the data type of the PLC tag and the data type of the HMI tag.

In WinCC, you have access to the following data types:

HMI data type	Description	Value range
Bool	Binary tag	0 to 1
SInt	Signed 8-bit value	-128 ... +127
USInt	Unsigned 8-bit value	0 ... 255
Int	Signed 16-bit value	-32768 ... +32767
UInt	Unsigned 16-bit value	0 ... 65535
DInt	Signed 32-bit value	-2147483648 ... +2147483647
UDInt	Unsigned 32-bit value	0 ... 4294967295
LInt	Signed 64-bit value	-9223372036854775808 to +9223372036854775807
ULInt	Unsigned 64-bit value	0 to 18446744073709551615
Real	Floating-point number 32-bit IEEE 754	+-3.402823e+38
LReal	Floating-point number 64-bit IEEE 754	+-1.79769313486231e+308
Byte	Bit array of 8 bits	8-bit
Word	Bit array of 16 bits	16-bit
DWord	Bit array of 32 bits	32-bit
LWord	Bit array of 64 bits	64-bit
String	Text tag, 8-bit character set	-
WString	Text tag, 16-bit character set	-
WChar	Text tag	-
DateTime	Date/time format	01.01.1601 00:00 to 31.12.9999 23:59:59
LTime	Signed 64-bit integer value	-106751d23h47m16s854ms775 us808ns to +106751d23h47m16s854ms775 us807ns
Raw	Raw data type	-

For format adaptation, select the desired PLC data type at the respective external tag. The suitable standard data type is then selected automatically in the "HMI data type" field for use in WinCC. Change the format adaptation as required.

Data types without format adaptation

The data types are shown 1:1 without format adaptation.

SIMATIC S7-300/400 data types without format adjustment

PLC data type	Description
Bool	No format adaptation
String	No format adaptation

3.2.5 Defining the acquisition cycle for a tag (RT Uni)

Introduction

The value of an external tag can be changed in runtime by the PLC to which the tag is linked. To ensure that the HMI device is informed of any changes in tag values by the PLC, the values must be updated on the HMI. The value is updated at regular intervals while the tag is displayed in the process screen or is logged. The interval for regular updates is set with the acquisition cycle. The update can also be made continuous.

Requirement

- You have created the tag for which you want to define an acquisition cycle.
- The Inspector window with the tag properties is open.

Procedure

To configure an acquisition cycle for a tag, follow these steps:

1. In the Inspector window, select "Properties > Properties > Settings".
2. Select the acquisition mode for the tag:
   - "Cyclic in operation": If you want to update the tag at regular intervals while it is displayed on the screen or is being logged.
3. Select the required cycle time in the "Acquisition cycle" field or define a new acquisition cycle using the object list.

Alternatively, you can configure the acquisition cycle directly in the work area of the tag table. To view hidden columns, activate the column titles using the shortcut menu.

Note

For structured HMI tags, the acquisition mode can be selected for the respective structured HMI tag as well as their individual subordinate elements.

When the acquisition mode of structured HMI tags is changed, it is applied to all subordinate elements. This means changing the acquisition mode may overwrite subordinate elements.

Result

The configured tag is updated in runtime with the selected acquisition cycle.

See also

Updating the tag value in runtime (Page 145)

3.2.6 Defining limits for a tag (RT Uni)

Introduction

For numerical tags, you can specify a value range by defining a low and high limit as well as the threshold values.

Requirement

- The tag for which you want to set the limits is created.
- The Inspector window with the properties for this tag is open.

Procedure

To define limits for a tag, proceed as follows:

1. In the Inspector window, select "Properties > Properties > Range". If you want to define one of the limits as a constant value, select "Constant" using the ∅▾ button. Enter a number in the relevant field.
   If you want to define one of the limits as a tag value, select "HMI tag" using the ∅▾ button. Use the object list to define the tag for the limit.
2. To set additional limits for the tag, repeat step 1 with the appropriate settings.

Tips for effective configuring

You also configure the limits directly in the tag table. To view hidden columns, activate the column titles using the shortcut menu.

Result

You have set a value range defined by the limits for the selected tag.

See also

Limits and start values of a tag (Page 146)

3.2.7 Synchronizing tags (RT Uni)

Introduction

To synchronize the PLC and HMI tags, WinCC offers the following options:

- Synchronizing tags with or without name matching between PLC and WinCC
  The following options are available for this:
- Link tags with addresses in the PLC
  This procedure is suitable, for example, if changes were made to the connection between the HMI device and the PLC and the tag connections were lost. The function can also be used if you have configured the control program and HMI project separately.

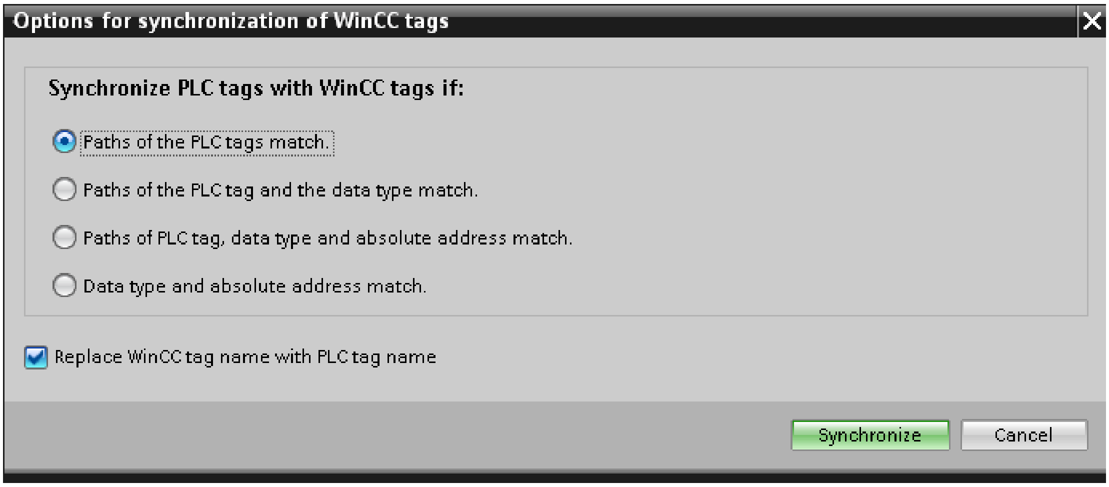
Requirement

- External HMI tags have been created.
- PLC tags have been created.
- An HMI connection to a PLC in the project has been established.

Procedure

To synchronize HMI tags with PLC tags, follow these steps:

1. In the project tree, select the directory that contains the tags in question.
2. Select "Synchronize with the PLC tag" from the shortcut menu.
   A dialog opens.



Options for synchronization of WinCC tags

Synchronize PLC tags with WinCC tags if:

● Paths of the PLC tags match.

○ Paths of the PLC tag and the data type match.

○ Paths of PLC tag, data type and absolute address match.

○ Data type and absolute address match.

☑ Replace WinCC tag name with PLC tag name

3. Select the option you want to use.
   If you want to synchronize the tags without name matching, disable "Replace WinCC tag name with PLC tag name".
   If you want to reconnect HMI tags with absolute access, select "Data type and absolute address match".
4. Confirm with "Synchronize".
   The system searches for a suitable PLC tag according to the selected option.

Result

The external HMI tags are synchronized with the PLC tags.

If you have selected the option "Data type and absolute address match", the tag connection is established as soon as a suitable PLC tag is found.

If you have selected a different option, the WinCC tags are updated accordingly and the PLC tag names are applied in WinCC.

3.2.8 Importing and exporting tags (RT Uni)

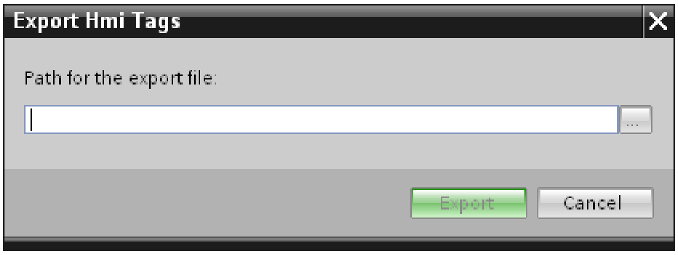
Introduction

WinCC gives you the option of exporting tags to an .xlsx file and reimporting them into the project once you are done editing them. You export and import tags in the "HMI Tags" editor.

For importing the tags, the xlsx import file must be structured according to the requirements. You will find more detailed information on the import file under "Export and import of tags (Page 150)".

Exporting tags

1. Click in the "HMI Tags" tab.
   The "Export" dialog box opens.



2. Click "..." and specify in which file the data are saved.
3. Click "Export".
   The export will start.

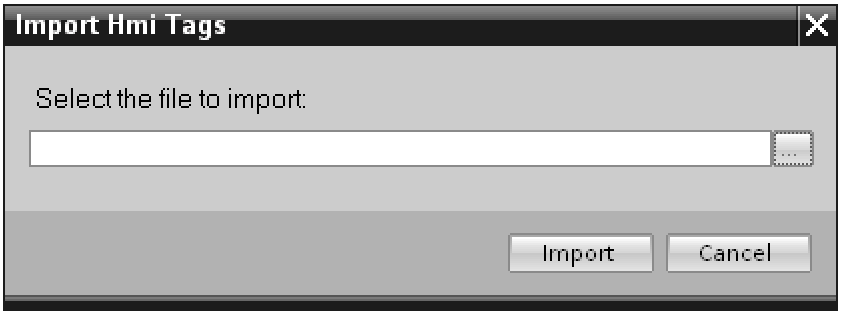
Note

It is not possible to export HMI tags of the data type "UDT" which contain structured elements via Excel for subsequent editing.

After export, only the higher-level HMI tag appears in Excel. Its lower-level elements cannot be edited.

Importing tags

1. Click in the "HMI Tags" tab.
   The "Import" dialog box opens.



2. Click "..." and select the file that you want to import.
3. Click "Import".
   The import will start.

Result

The relevant tags have been created in WinCC. Alarms relating to the import operation are displayed in the output window. A log file is saved in the source directory of the import files. The log file has the same name as the respective import file but with the "*.xml" extension.

Check when importing the data whether there are any links to objects, for example, dynamic parameters such as tags.

- If an object with the same name already exists, the existing object is used.
- If no object of the same name yet exists, create an object with the relevant name or create a new link.

Note

The syntax of the import file is checked during xlsx file import. The meaning of the properties or dependencies between the properties is not checked.

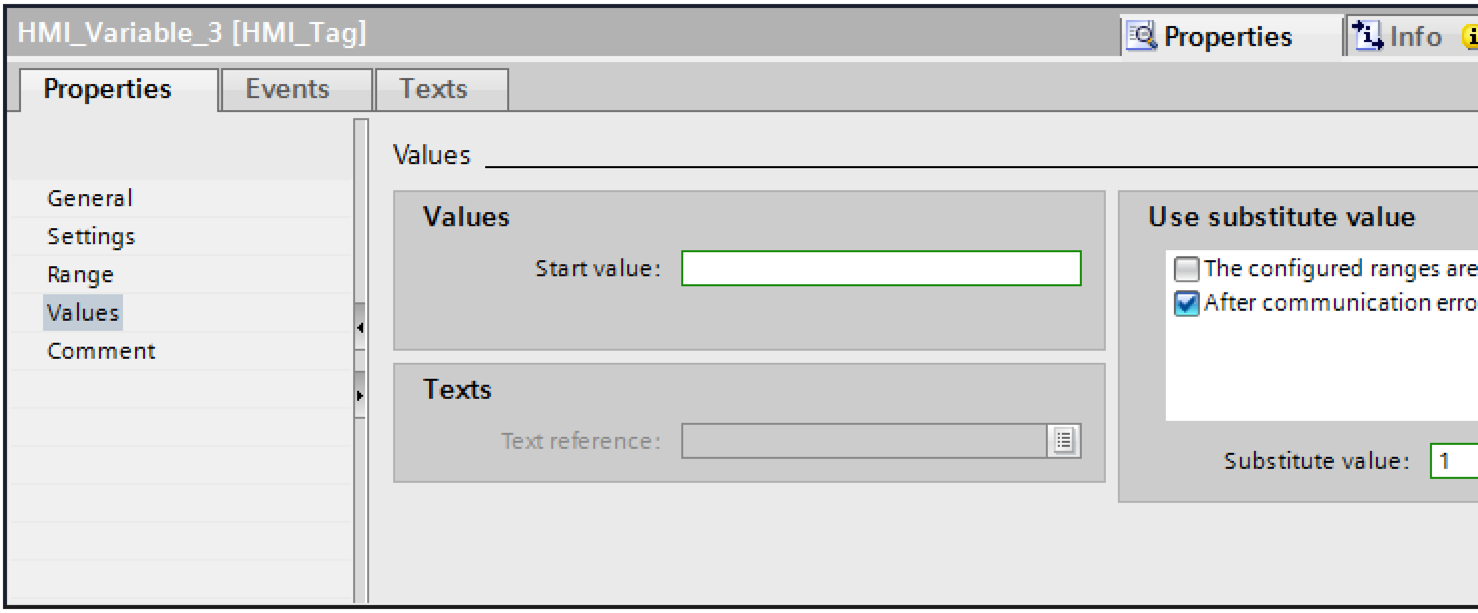
3.2.9 Defining a substitute value (RT Uni)

Introduction

You can define a specific value as a substitute value for a tag.

In the "Use substitute value" area you define when WinCC should use this substitute value. The current process value is then not accepted from the automation system.

You can define a substitute value for the following cases:



Start value:

Text reference:

☐ The configured ranges are

☑ After communication erro

Substitute value: 1

- The configured ranges have been violated
  If you have set limit values for the tag, WinCC sets the substitute value as soon as the process value violates a limit.
- In the event of a communication error
  WinCC sets the substitute value when the connection to the automation system is disturbed and there is no valid process value.

Requirement

- The tag table is open.
- The Inspector window with the tag properties is open.
- The HMI tag is linked to a PLC tag

Procedure

To configure a substitute value, follow these steps:

1. Select the desired tag in the tag table, and select "Properties > Properties > Values" in the Inspector window.
2. In the "Use substitute value" segment, select when you want WinCC to use this substitute value in the tag.
3. Enter the required substitute value in the "Substitute value" field.

Result

The configured tag is populated with the substitute value in runtime once the selected condition is fulfilled.

Note

If you have set a high or low limit in an I/O field, you cannot enter any value outside these limits.

WinCC ignores incorrect entries and therefore does not set a substitute value. The substitute value is only set by WinCC when an incorrect process value is read.

3.2.10 Connecting a tag to another PLC (RT Uni)

Introduction

In WinCC, you can change the PLC connection of a tag at any time. This is needed when you change the configuration of your plant, for example.

Depending on the PLC selected, you may need to modify the configuration of the tag. The tag properties which must be changed will be highlighted in color.

Requirement

- The external tag, whose connection you wish to change, must already exist.
- The connection to the PLC must already exist.
- The Properties window for this tag is open.

Procedure

To change the PLC connection of a tag, proceed as follows:

1. In the Inspector window select "Properties > Properties > General."
2. Select the new connection in the "Connection" field.
   The tag properties that you must change will be highlighted in color in the tag table and in the Inspector window.
3. Change all highlighted properties of the tag to suit the requirements of the new PLC.

Result

The external tag is connected to the new PLC.

3.3 Configuring user data types (RT Uni)

3.3.1 Creating a user data type (RT Uni)

Introduction

You create a user data type in the project library.

Requirement

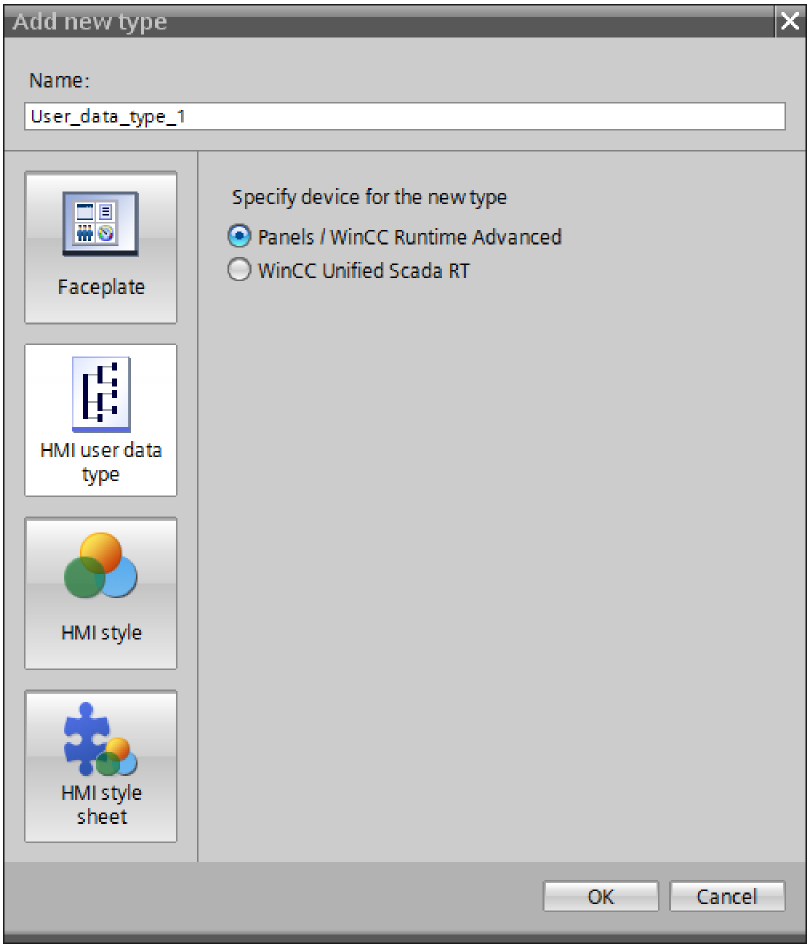
- A project is open.
- The "Libraries" task card is displayed or the library view is open.

Procedure

To create a user data type, follow these steps:

1. Click the "Libraries" task card.
2. Double-click the "Project library" item. The folder structure of the project library is open.

3. Click "Add new type".
   A dialog opens.



Add new type

Name:

User_data_type_1

Specify device for the new type

● Panels / WinCC Runtime Advanced

○ WinCC Unified Scada RT

4. Click the "HMI user data type" button in the dialog box.
5. In the "Specify device for the new type" area select the HMI device on which the user data type is used.
6. Enter a descriptive name in the "Name" field.
7. Click "OK" to apply your settings. The user data type is created. The library view opens.

Result

A user data type with the configured properties is created. Version 0.0.1 of the user data type is created and receives the status "in work".

Create the required user data type elements in the next step.

Before you use the version of user data types, for example at a tag, the version must be released.

3.3.2 Creating user data type elements (RT Uni)

Introduction

You define user data type elements in the library view. You add or delete elements in the "HMI user data types" table in the work area.

Requirement

- The library view is open.
- A user data type is created and opened in the editor.

Procedure

1. Select a communication driver for the user data type.

- If you select the <Internal communication> entry, you can only assign the user data type to the internal tags as the data type.
- If a connection to a PLC is selected as the communication driver, the user data type can only be assigned to tags with a connection to this PLC as the data type.
- The communication type set applies to all user data type elements of a user data type. In a user data type for WinCC Runtime Unified, you can define for each user data type element whether the configured driver for control or internal communication is used.

1. Double-click "Add" in the "Name" column of the table. A new user data type element is added to the user data type.
2. Assign a name.
3. Select the required data type in the "Data Type" field.
4. Create as many user data type elements as you need.
5. You configure the user data type elements in the "Properties" group in the Inspector window.

Alternatively, you can configure the properties of the user data type elements directly in the table. To view hidden columns, activate the column titles using the shortcut menu.

Result

You have added elements to version 0.0.1 of the user data type. The version 0.0.1 has the status "in work".

To use the version in the project, release the version.

3.3.3 Managing versions of user data types (RT Uni)

Introduction

All user data types have at least one version. When a user data type is created, a version is created at the same time and this version has the status "in work". You can edit the user data type in this status as required. When the editing is complete, you release the version of the user data type.

Requirement

- You have created a user data type.
- The user data type has the version 0.0.1. and the status "in work".
- The "Libraries" task card or the library view is open.

Releasing a version

1. Select the version 0.0.1 of the user data type in the project library.
2. Select "Release version" in the shortcut menu.
   A dialog opens.
3. If necessary, change the properties of the version:
   – Enter a name for the type in the "Name" field.
   – In the "Version" field, define a main and an intermediate version number for the version to be released.
   – To clean up version management of the type, enable "Delete unused type versions from the library".

You have released version 0.0.1 of the user data type.

Editing a version

1. Select, for example, the released version 0.0.1 of a user data type in the project library.
2. Select "Edit type" in the shortcut menu.

The library view opens. The new version 0.0.2 of the user data type is created.

The version has the status "in work".

Restoring the last version of a user data type

The last released version of the user data type is version 0.0.2.

You edit the user data type. A new version of the user data type, 0.0.3, is generated and receives the status "in work".

1. Select the version of the user data type in the project library.
2. Select "Discard changes and delete version" in the shortcut menu.

Alternatively

1. If you have opened a version for editing, click "Discard changes and delete version" in the toolbar.
   The version is deleted.

All changes to the user data type since the last release operation are discarded. The user data type is released again and has version 0.0.2.

Deleting user data type

If you delete a user data type, all instances and master copies of this user data type are deleted as well. The same is true for HMI tags that use an HMI user data type.

To delete a user data type, follow these steps:

1. In the project library, under "Types", select the user data type you want to delete.
2. Select "Delete" from the shortcut menu.

3.3.4 Creating tags with a user data type data type (RT Uni)

Introduction

When a tag is created, you assign the version of user data type as a data type. In the "Tag" editor you can create internal tags or tags with a link to a PLC. A tag provides all user data type versions that use the same communication driver as the tag itself.
A user data type cannot be used in combination with a PLC unless you have selected absolute addressing.

Requirement

- A user data type with a user data type element is created.
- The user data type is enabled.
- The tag table is open.
- The Inspector window with the tag properties is open.

Procedure

To create a tag of the "User data type" data type, follow these steps:

1. In the "Name" column, double-click "Add" in the tag table. A new tag is created.
2. In the Inspector window select "Properties > Properties > General".
3. Enter a unique tag name in the "Name" field.
4. Select the connection to the required PLC in the "Connection" box.

5. Select the desired version of the user data type in the "Data type" field.
   The selected connection determines which user data types will be displayed.
   For internal tags, only user data type versions of the <Internal user data type> type are available.

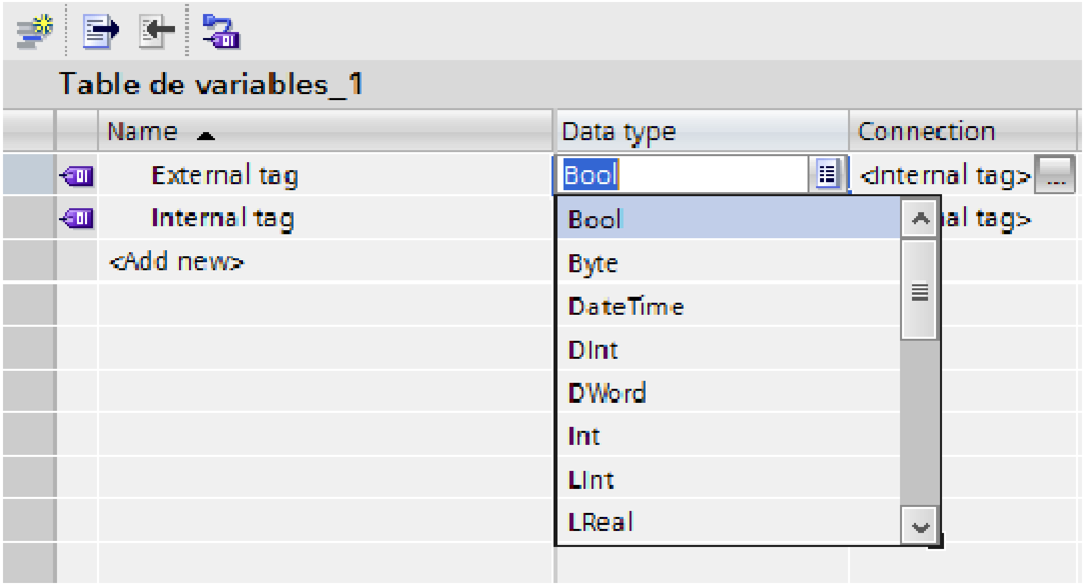


Table de variables_1

Name	Data type	Connection
External tag	Bool	<Internal tag>
Internal tag		al tag>
<Add new>		

> **Note**
>
> For tags with a connection to a PLC, only those user data types that have a link to a PLC can be accessed.

6. Enter the address of the PLC that you want to access with the external tags in the "Address" field of the "Settings" area. The specified address must always point to the start data bit, for example, <DB1.DBX0.0>.

Result

You have created a tag of the "User data type" data type. In additional steps you can configure the tag, for example, by setting the start value and limits.

If you wish to change the properties of a user data type tag, you must change the properties of the user data type element.

Properties such as "Start value" and "Linear scaling" can also be changed in the user data type instances used.

3.4 Logging tags (RT Uni)

3.4.1 Basics of data logging (RT Uni)

Introduction

Data logging is used to collect and log process data from an industrial system. You can use data logging to analyze error states and to document the process.

When you analyze the logged process data, you can extract important business and technical information regarding the operational state of the plant.

How it works

The process values to be logged are compiled, processed and saved in the log database in runtime. Current or previously logged process values can be output in runtime as a table or trend. The process values are logged if the tag value changes.

Configuration

You configure the logging of process values in the "Historical Logs" editor. You define the acquisition cycles, logging cycles, log size and, if necessary, the storage path.

You configure the logging tags in the "HMI tag" for each tag and specify whether the number of logged values is to be reduced with smoothing.

You configure trend views and process controls for displaying process data in runtime in the "Screens" editor. These views allow you to output the process data in the form of trends and tables.

Application

You can use data logging for the following tasks:

- Early detection of danger and fault conditions
- Increase of productivity
- Increase of product quality
- Optimization of maintenance cycles
- Documentation of process value trends

3.4.2 Defining log size, segmentation and backup (RT Uni)

Introduction

For the log, you define the time period in which the data is written to a given log, and the maximum size of the log file.

Each log consists of a configurable number of segments. You define a size in MB, a starting time and a period (for example one day) for the segments.

Note

Make sure that the log size does not exceed the free memory space available. The system does not validate the selected settings. A high number of linked log segments can lead to prolonged waiting periods in the system when starting and ending runtime.

Note

You define the maximum log size as a multiple of the maximum segment size. Multiple log segments must not exceed the maximum log size.

Segments

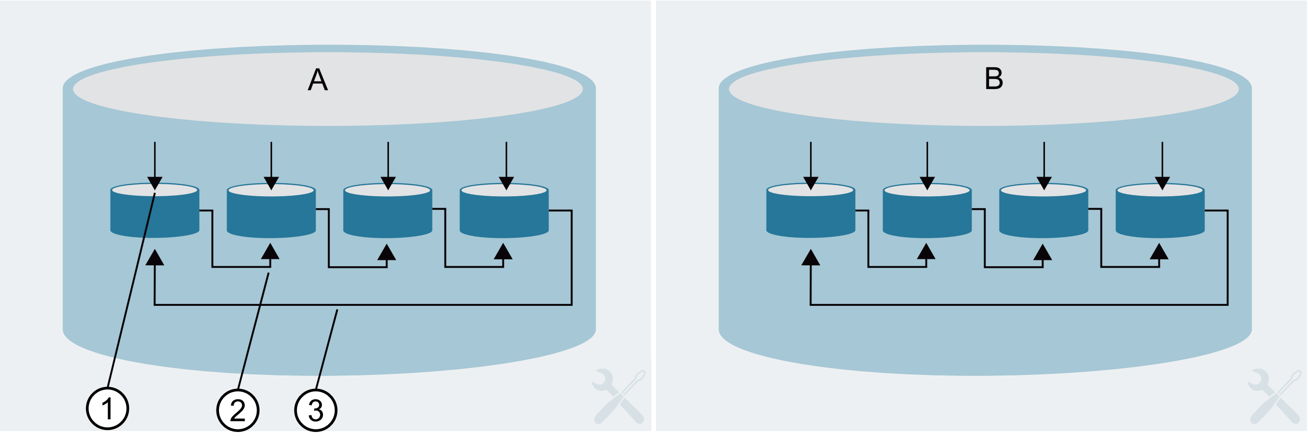
In a segmented log, multiple log segments of the same size are created, and filled in succession. When all segments are completely full, the oldest segment is overwritten.

You can configure the following properties for the log segments:

- The segment time period defines the maximum period for one log segment, for example one day.
- Maximum segment size defines the maximum size of a log segment in MB. When the log segment reaches the defined size, the current segment is closed and a new segment is created and filled with data.
- The start time and the time period define when to switch to the next log segment. The log segments are written to the log from the start time. The segment changes at the end of the configured time period, for example 8 hours. A new segment is created if the configured segment size is exceeded. The next log segment change then takes place at the end of the configured time period.

Response to segment change

The individual segments are filled one after the other in runtime. Once a segment is totally full, the next segment is created and filled. You can also configure the segment change at specific times. If you define a time for the segment change, the next log segment is filled when the time is reached.



1. The process values are written continuously to the first segment.
2. When the configured size of the segment is reached or the time period is exceeded, the system switches to the next segment.
3. When all segments are full, the oldest segment is deleted and a new segment is created.

To avoid losing process data as a result of overwriting, you can export it to a backup.

Example

The following input has been configured:

Property	Value
Time period of all segments	1 week
Maximum size of all segments	700 MB
Time period covered by a single segment	1 day
Maximum size of a segment	100 MB
Segment start time	Friday, November 23, 2017, 18:00

With the configuration shown in the table, the started segment will be changed for the first time at 18:00 on November 23, 2017. The next time-controlled segment change will take place cyclically after periods of 1 day from the configured time.

Note

If you change the segmentation settings and run "Download to device", a new segment will be created.

The segment will also change if the configured size of 100 MB is exceeded in the course of one day. The oldest single segment will be deleted if the maximum log size of 700 MB is exceeded.

Backup

You can export process values from the log database as a backup. All process values contained in a log segment are exported. A log segment is always exported upon segment change, when it is full and a new segment is started. A log segment is also exported when the time set for a segment change is reached and a new segment is started.

Note

Ensure that the memory for the swapped out process values is sufficient.

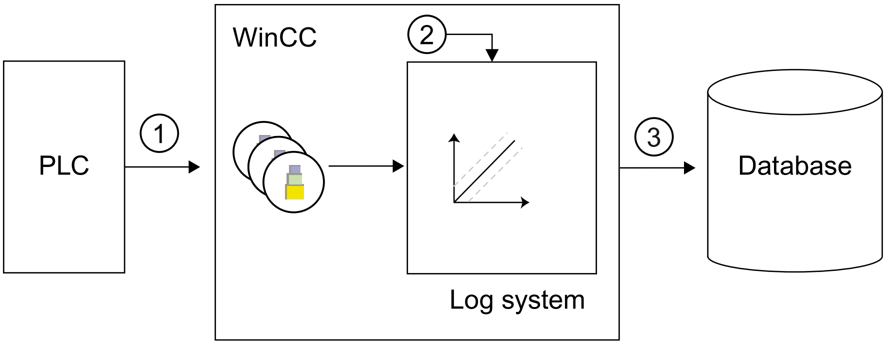
3.4.3 Data logging on change (RT Uni)

Introduction

Data logging on change saves the current process value to the log database in runtime when the process value changes. You can also specify limits. The process value is then only logged if it is within the defined limit range. A comparison of the process value with the limit values takes place after acquisition of the process value.

Data logging on change ends when runtime closes.

How it works



The external tags in WinCC correspond to a certain process value in the memory of one of the connected PLCs.

1. The process value is read cyclically from the memory of the connected PLC and monitored for change.
2. The process value is logged when a change occurs in the tag.
   The runtime component of the logging system processes the process value.
3. The current process value is then written to the logging database.
   If you have configured smoothing for logging, the values calculated are logged.

3.4.4 Creating a data log (RT Uni)

Introduction

The configuration of a data log consists of the following steps:

- Create a data log.
- Configure the data log, for example, by selecting the storage location.

Note

The SQL database under the folder configured for the Data log is only created after runtime has been started.

Requirement

- A project is open.
- An HMI device has been created.
- The Inspector window is open.

Procedure

To create a data log, proceed as follows:

1. Double-click on the "Logs" entry in the project tree.
   The editor for data logs and alarm logs opens.
2. Open the "Data logs" tab and double-click "<Add>" in the "Name" column of the "Data logs" editor.
   A new data log is created.
3. Select the storage path for the log in the "Storage path" field.

   Note

   The storage path of the log cannot be changed after the first transfer.

   You have the option of leaving the memory path empty. In this case the log is stored under the default path (in the current runtime project: ...\TLGDB\<TagLoggingDatabase>\<Segmentname>.mdf).
4. Define the maximum time period for logging in the "Log time period" field, for example 7 days.
   If you specify a value of "0" for the logging period, the log will be written continuously. As soon as the maximum size is reached, the oldest segment is deleted from the log and a new segment is written.
5. Define the maximum size of the log file in MB in the "Maximum log size (MB)" field.

6. Define the time period, the start time and the maximum size of log segments in the "Segment" area.

   Note

   If you change the log size or the time period in runtime, the previous segment will be closed and a new segment created with the new settings.

7. Set whether data is to be backed up and specify the path for the backup under "Backup > Backup mode".

   Note

   The oldest single segment will be deleted if the maximum log size of 700 MB is exceeded. Configure backup to avoid losing logged data.

   If you subsequently change the primary path, the new backup file will only be written to the new storage path after loading. The previous backup file will remain in the previous storage path.

Tips for effective procedure

You can configure log properties directly in the "Data logs" editor. To view hidden columns, activate the column titles using the shortcut menu.

Result

The data log is created.

3.4.5 Configuring logging tags (RT Uni)

Introduction

Logging tags are configured and edited in the "HMI tags" editor.

Note

If you delete, move or copy in the "HMI tags" editor, the changes also take effect in the "Logs" editor.

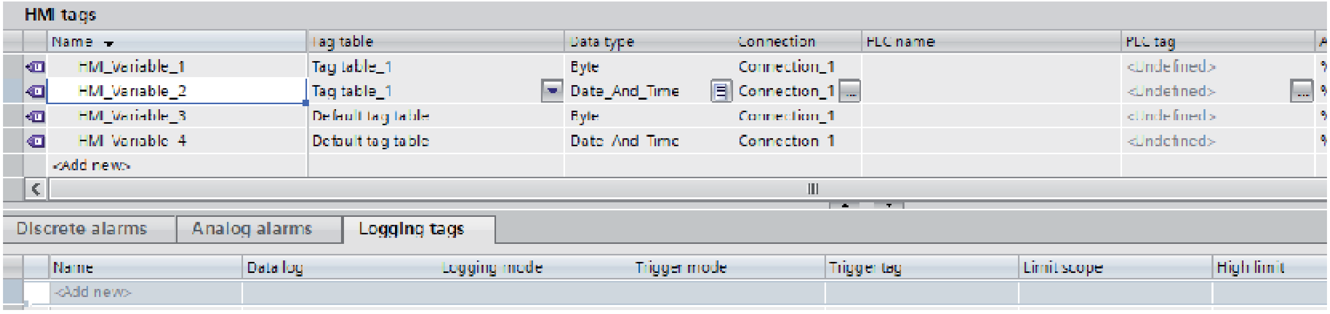
Requirement

- The "Logging tags" tab is open in the "HMI tags" editor.
- You have created a tag.

Procedure

Proceed as follows to configure a logging tag in the "HMI tags" editor:

1. Select an existing tag in the tag table.
   Alternatively, double-click "<Add>" in the "Name" column to create a new tag.
2. Click "Logging tags" in the lower editor. The editor for logging tags is brought to the foreground.

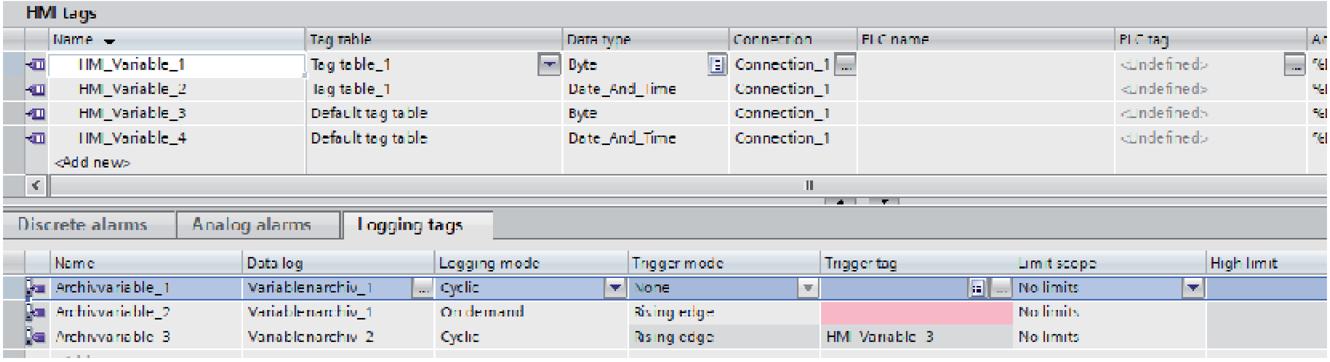


HMI tags

Name	Tag table	Data type	Connection	PLC name	PLC tag	A
HMI_Variable_1	Tag table_1	Byte	Connection_1		<Undefined>	%
HMI_Variable_2	Tag table_1	Date_And_Time	Connection_1		<Undefined>	%
HMI_Variable_3	Default tag table	Byte	Connection_1		<Undefined>	%
HMI_Variable_4	Default tag table	Date_And_Time	Connection_1		<Undefined>	%
<Add new>						

Name	Data log	Logging mode	Trigger mode	Trigger tag	Limit scope	High limit
<Add new>						

3. In the table of the "Logging tag" editor table, double-click "<Add>" in the "Name" column. A new logging tag is created. The logging tag is linked to the tag selected in the first step. The data type of the logging tag is taken automatically from the linked tag.



HMI tags

Name	Tag table	Data type	Connection	PLC name	PLC tag	Ar
HMI_Variable_1	Tag table_1	Byte	Connection_1		<Undefined>	%I
HMI_Variable_2	Tag table_1	Date_And_Time	Connection_1		<Undefined>	%I
HMI_Variable_3	Default tag table	Byte	Connection_1		<Undefined>	%I
HMI_Variable_4	Default tag table	Date_And_Time	Connection_1		<Undefined>	%I
<Add new>						

Name	Data log	Logging mode	Trigger mode	Trigger tag	Limit scope	High limit
Archivvariable_1	Variablenarchiv_1	Cyclic	None		No limits	
Archivvariable_2	Variablenarchiv_1	On demand	Rising edge		No limits	
Archivvariable_3	Variablenarchiv_2	Cyclic	Rising edge	HMI_Variable_3	No limits	

4. Assign a data log to the new logging tag. It is possible to assign a different log to each logging tag.
5. Select "Properties > Properties > Smoothing" in the Inspector window and select the required smoothing mode for compressing the logged values.
6. If "Cyclic" was selected as logging mode, select the desired compression mode for compressing the logged values under "Properties > Properties > Compression".
7. Select "Properties > Properties > Limits" in the Inspector window and enter the limit range and the required limit values. Now enter the required limit values. Process values that are outside the limits will not be logged.

Note

Using logging tag in the Engineering System

To visualize the value of a logging tag in a screen object, select the "Source type" as "Log" and start by entering the name of the HMI tag under "Tag" and then enter the name of the associated logging tag separated by a colon, e.g. "Hmi_Tag_1:Archive_Tag_1".

Result

The configured logging tag is created in the "Logging tags" editor.

3.4.6 Configuring smoothing (RT Uni)

Introduction

You can compress the data volume of the logged data using smoothing to reduce the memory space required. The process values are then only logged in accordance with certain predefined criteria.

Two types of smoothing are supported:

- Analog value smoothing
  Analog value smoothing allows you to set the maximum interval between the last value archived and the current values.
- Time smoothing
  Time smoothing allows you to define the minimum and maximum time interval after which the next value is to be logged.

Smoothing mode

Select one of five smoothing modes for the logging tag:

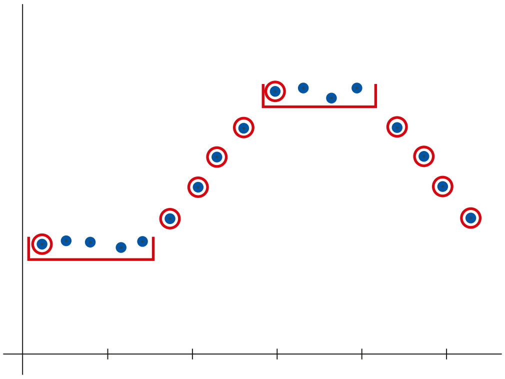
- No smoothing
  The values are logged without compression.
- Comparing values
  You specify a time interval. No value changes are logged within this defined interval.
- Value
  You specify a limit value that defines the maximum permitted distance between the values. All value changes occurring within the defined interval from the last logged value are not logged.

- Relative value
  You specify a percentage deviation that defines the maximum interval between the values. All value changes occurring within the defined interval relative to the last logged value are not logged.
  Example:
  – You define a deviation of 10% and the last logged value is 100.
  – The value 105 is not logged as the value change is less than 10%. The value 130 is, on the other hand, logged as the value change is greater than 10% and therefore relevant for logging.
- Swinging door
  The swinging door algorithm evaluates values on the basis of the defined rate of change and only logs them if the following value is outside the calculated range. The compression rate depends on the maximum deviation tolerated. The starting point for calculating the next logging time is the last value logged.
  Using the set deviation, you can influence the precision with which the values are saved. The greater the deviation, the fewer values are logged. With the maximum time, you specify the time after which a new value will definitely be logged. This specifies additional reference values for the logged data even if no significant changes occur during this time. With the minimum time, you specify the time interval after which the next value for logging is calculated. All measured values within the minimum time are not logged.

Example – smoothing with the value

You specify a constant value for the deviation. All values that are within the defined deviation and have not changed significantly are not logged.

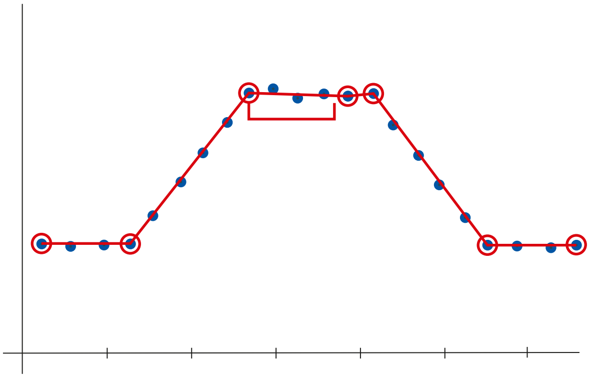
Only the values outside the deviation are written to the log.



Example – swinging door with deviation and maximum time

You define the deviation and the maximum time after which the next value is to be written to the log.

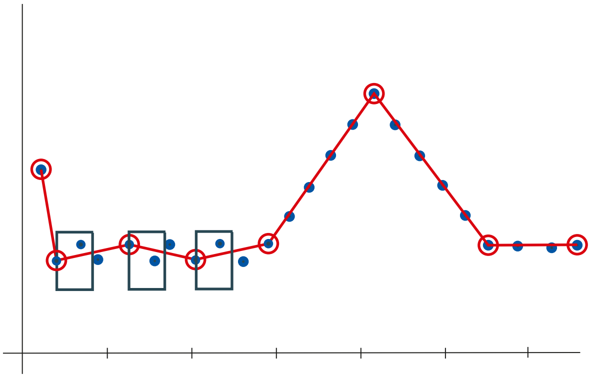
Once the first value has been saved, the following values are evaluated at the predefined rate of change. If the value is within the rate of change, it is not logged. If the value is outside the deviation, it is logged. Values that have not changed significantly compared to the previous log value are also logged at regular, defined intervals.



Example – swinging door with deviation and minimum time

You define the deviation and the minimum time after which the next value is to be evaluated in runtime.

Once the first value has been saved, the next value for logging in runtime is calculated after the preset time. If the value is within the rate of change, it is not logged. If the value is outside the deviation, it is logged.



3.4.7 Configuring compression (RT Uni)

Introduction

You can compress the data volume of the logged data using compression to reduce the memory space required.

If you have selected "Cyclic" as logging mode, HMI Runtime logs the tag values according to the defined logging cycle exactly when the current value deviates from the one already stored. Since only the changed value is saved, there is always an intentional loss of data.

Note

Please note that the LoggingCycle is independent of the data collection cycle of the logged tag. Although it is possible to set the logging cycle shorter than the data collection cycle, note that this does not make sense.

HMI Runtime provides different compression modes.

Compression can only be used in connection with the "Cyclic" log mode. Please note that when you use "Cyclic" logging mode you have to select a compression mode in each case. If you do not want a compressed calculation, select "End" as the mode to save only the most current value of the log interval.

Compression mode

1. Select one of eight compression modes for the logging tag:

Compression mode	Description
Minimum	The minimum of the values determined within the logging interval, including the start value, is logged. The logging receives a time stamp of the interval start.
Maximum	The maximum of the values determined within the logging interval, including the start value, is logged. The logging receives a time stamp of the interval start.
Minimum with time stamp	The minimum of the values determined within the logging interval, including the start value, is logged. Unlike in the "Minimum" mode, in this mode the logged minimum value receives the time stamp of when it occurred.
Maximum with time stamp	The maximum of the values determined within the logging interval, including the start value, is logged. Unlike in the "Maximum" mode, in this mode the logged maximum value receives the time stamp of its occurrence.
Total	The total of all values determined within the specified logging interval is logged without the start and end values.
Mean	The average value of all values determined within the specified logging interval is logged without the start and end values.
Time average stepped	The time-weighted average value of all values determined within the specified logging interval without start and end value is logged.
End	The last value determined within the specified logging interval is logged with a time stamp of occurrence.

2. Specify a time interval for compression mode:
   - Quarter-hourly
   - Hourly
   - Daily
   - Monthly
   - Yearly
   - plus: self-configurable
3. You have the option of defining a delay. The deceleration value determines the latest possible point in time up to which the compression value is to be logged after the end of a logging cycle and from which no new values are to be taken into account. Values with a time stamp that is after the deceleration value are not logged.

Result

In runtime, the determined tag values are calculated and stored according to the configuration.

The following values are not stored:

- Values that lie outside the interval
- Values that were not included in the determination due to the selected compression mode, are not saved.

3.4.8 Configuring limits (RT Uni)

Introduction

You have the option of logging the logging tag values outside or within a defined value range. Define a valid range and limits for the process values of the logging tag. In runtime, the process values are evaluated after the configured limit range and only process value within the defined range are logged.

Limit ranges

You can set the following ranges for the process values:

Range	Description	Example
No limits	No limit values are defined for logging.	No limit values are taken into account.
Upper	The process values that are greater than the configured low limit are logged.	Low limit = 3; Logged values = 4, 5, 6.
Lower	Only the process values that are less than the configured high limit are logged.	High limit = 6; Logged values = 3, 4, 5.
Upper or equal	Only the process values that are greater than or equal to the configured low limit are logged.	Low limit = 3; Logged values = 3, 4, 5, 6.
Lower or equal	Only the process values that are less than or equal to the configured high limit are logged.	High limit = 6; Logged values = 3, 4, 5, 6.
Within limits	Only the process values that are within the two configured limits are logged.	Low limit = 3, high limit = 6; Logged values = 4, 5.
Within or equal	Only the process values that are within the two configured limits or equal to one of the configured limits are logged.	Low limit = 3, high limit = 6; Logged values = 3, 4, 5, 6.
Outside limits	Only the process values that are beyond the two configured limits are logged.	Low limit = 3, high limit = 6; Logged values = 1, 2, 7, 8.
Outside or equal	Only the process values that are beyond the two configured limits or equal to one of the configured limits are logged.	Low limit = 3, high limit = 6; Logged values = 1, 2, 3, 6, 7, 8.

Configuring limits

You define the limit range and limit values for each logging tag in the "HMI tags" editor.

1. Select an existing logging tag from the "Logging tags" tab of the tag table.
2. Specify the limit range under "Limits" and enter the corresponding limit values.
   Only numerical values are supported.

Result

The tag values are logged in accordance with the configured value range in runtime. Process values outside the limit range are not logged.

3.5 Displaying tags (RT Uni)

3.5.1 Basics (RT Uni)

3.5.1.1 Outputting the tag values (RT Uni)

Overview

With WinCC you can output tag values in the HMI screen with different screen objects and change them.

- The I/O field is used for the input and output of process values.
- Bars are used for graphic display of the process values in form of a scale.
- Sliders are used for the input and output of process values within a defined range.
- The gauge is used to display the process values in form of an analog gauge.

In runtime you can also output tag values as table or as trend. You can use either process values or logged values as source for the tag values.

- Use a trend for the graphic display of tag values. Trends allows you to display the change in motor temperature, for example.
- Use a table to compare tag values. In the table you can, for example, compare fill levels of supply tanks.

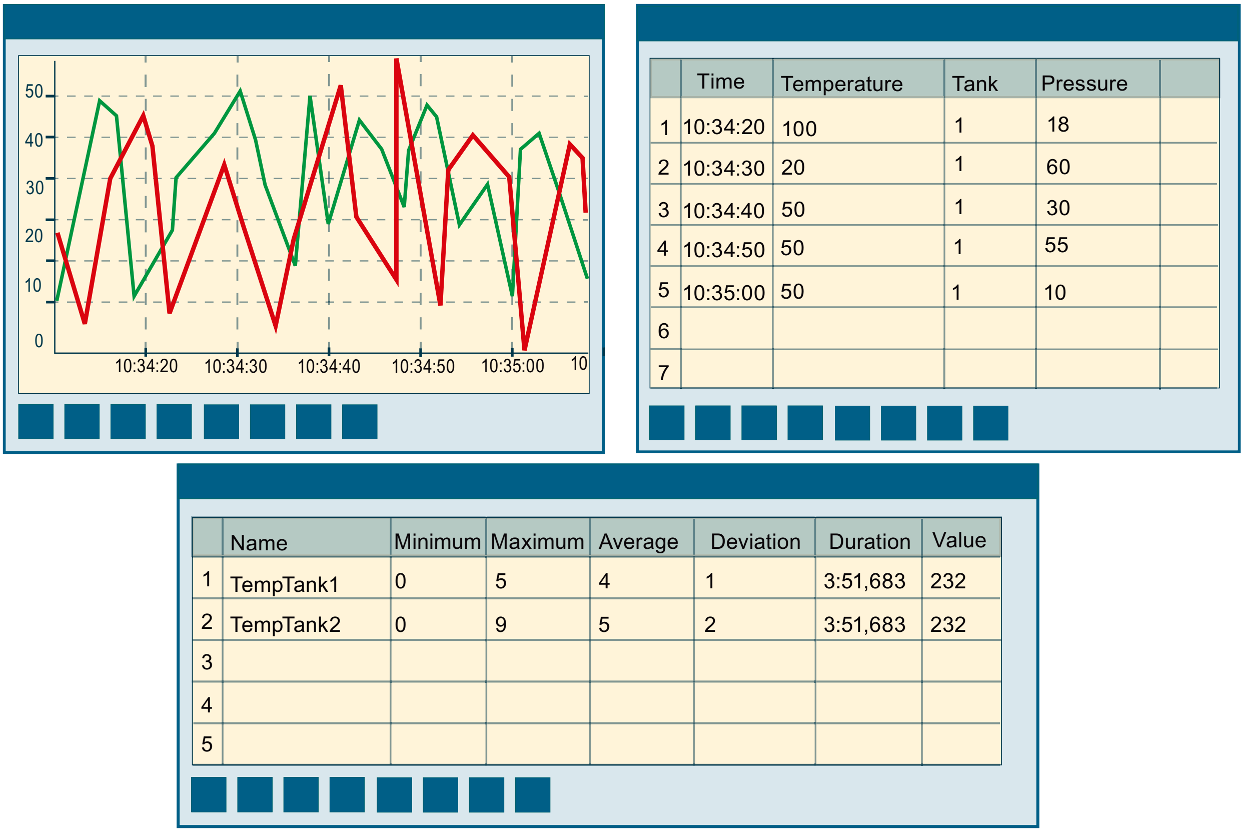
Controls for displaying tag values

To display tag values as a trend, use the trend control. The versions of trend views are available:

- "Trend control": You display a tag value over time, for example, the change in temperature. You can compare the current values and logged values or monitor the change in current values on the HMI device.
- "Function trend control": You display a tag value against a second tag value, for example, the engine speed against the heat produced.

You can use the "Trend companion" to create statistics, for example, from the displayed values. Furthermore, you can use the trend companion as reading assistance for the trend control.

To display tag values in a table, use the process control.



	Time	Temperature	Tank	Pressure	
1	10:34:20	100	1	18	
2	10:34:30	20	1	60	
3	10:34:40	50	1	30	
4	10:34:50	50	1	55	
5	10:35:00	50	1	10	
6					
7					

	Name	Minimum	Maximum	Average	Deviation	Duration	Value
1	TempTank1	0	5	4	1	3:51,683	232
2	TempTank2	0	9	5	2	3:51,683	232
3							
4							
5							

Displayed values

When configuring the trend control, specify which tag values are to be displayed.

- "Online": The trend is continued with current individual values from the PLC.
- "Log": In runtime, the trend control displays the values of a tag from a data log. The trend shows the logged values in a particular window in time. The operator can move the time window in runtime to view the desired information from the log.

3.5.1.2 Outputting tag values as trends (RT Uni)

Introduction

You have the option of displaying the values of tags graphically in runtime with the help of the following controls:

- You can visualize the trend control to display currently pending process values or logged values in runtime as trends over time.
- You use the function trend control to visualize currently pending process values or logged values in runtime as trends in relation to other tags.

The axis designations are different for the two trend views:

- The trend control has a "Time axis" and a "Value axis"
- The function trend control has an "X axis" and a "Y axis"

You can display up to nine trends in both the function trend control and the trend control.

Structure of a trend control

Configure the trend control appearance in the Inspector window: You define the number of trend areas and configure the trends it contains.

You can configure multiple trends, value axes and time axes for each trend area. You can alter the appearance, labeling and assignment in the Inspector window for each individual trend, value axis and time axis created.

Configure trend areas

You can divide the display area into multiple trend areas, if necessary. Each area functions like a standalone trend control. This allows you to show temperature changes or values from different days, for example, as trends and compare them. The "Range proportion" specifies how much space is provided for a given area in the trend control.

Each range proportion is calculated on the basis of the total number of range components. If you have configured a total of three trend areas, for example, a range proportion of "1" will result in three trend areas of equal size. To increase the range proportions in relation to each other, increase the range proportion of one or more trend areas.

Configuring axes

You configure the axes of the trend control for each trend area in the properties of the trend areas.

The following properties are set by default with a value axis:

- The value range is based on the current values of the assigned trend
- The value axis scale is linear and based on the value range
  Alternatively, you can configure a logarithmic scale:
  - In logarithmic scaling, only positive values are displayed.
  - In negative logarithmic scaling, only negative values are displayed.

If you configure a value axis for the trend control, you can also set up axis segments. You assign a value range and a display name to each axis segment.

In the function trend control, the value axis corresponds to both the "X axis" and the "Y axis".

Configuring trends

You configure the axes for each trend area:

- The time and value axes in the trend control
- The different value axes in the function trend control

By default, the data area is based on the current values of the associated trend.

You can also configure the visualization of limits and values with "Uncertain status" for a trend. If a trend exceeds or falls below a configured limit, the trend is colored.

Configuring the time axis and time range (trend control)

The time range for trend display is configured with time axes. In a trend control, you can create multiple time axes that you can assign to one or more trend areas.

If you configure several time axes to a trend control, the sequence of the time axes in the Inspector window corresponds to the sequence in the trend control. If multiple time axes run along the same side of a trend control, the first time axis in the list is at the bottom left. The last time axis is at the top right.

3.5.1.3 Representing multiple trends (RT Uni)

Introduction

If you display several trends simultaneously in the trend control, assign each trend its own value and time axis. Alternatively, you can assign a shared time and/or value axis to several trends.

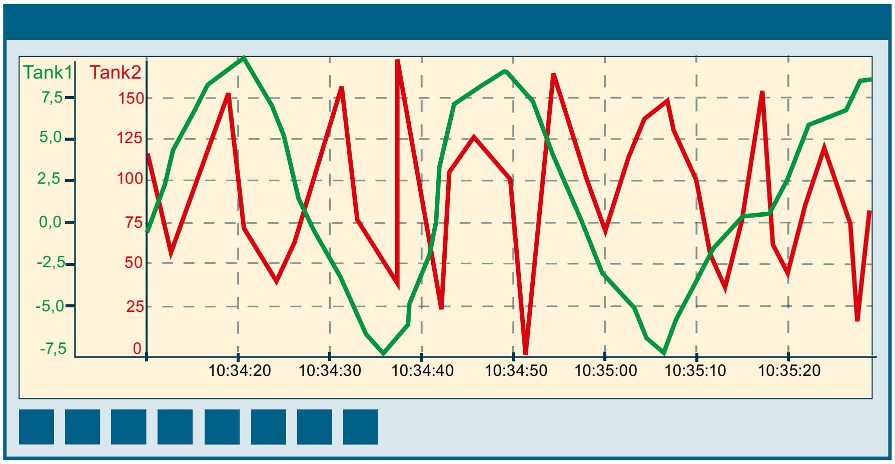
You configure the axes of a trend control for each individual trend in the Inspector window under "Properties > Trend areas".

The axes are assigned to the configured trends in the Inspector window under "Properties > Trend areas > Trends".

Representation using different axes

If the values to be displayed in a trend control differ greatly, a common value axis makes no sense. If you assign each trend its own value axis, they should also display different scales. Individual axes can be hidden if required.

The figure below shows two trends with different value axes using a trend control as an example:

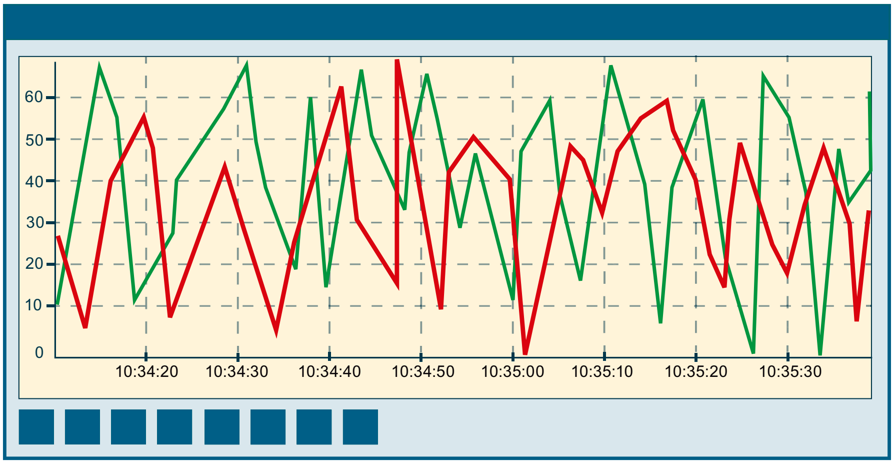


Representation using common axes

If the comparability of the trend directions is important, common axes in a trend control is sensible. Connected trend views can have a common time axis.

If you configure trends with a shared time axis, use tags with the same update cycle for the data supply.

In the case of different updating cycles, the length of the time axis is not identical for all tags. Since the trends are updated at different times due to the different updating cycles, a minimal different in the end time for the time axis occurs on each change. As a result, the trends displayed skip slightly to and fro on each change.



3.5.1.4 Basics of time range (RT Uni)

Time range

The time range is the range from which the values at the HMI device are shown. The time range is determined by the start time and the end time. The time range is always in the past. If the end time is later than the current system time, the current system time is used as a temporary end time.

A distinction is made between the following time ranges:

- Static time range
- Dynamic time range

Static time range

The static time range is determined by fixed start and end times. The values are displayed within this time range.

Dynamic time range

The dynamic time range is determined by a period of time beginning with a fixed start time. The end time thus corresponds to the conclusion of the time period.

You set the time period as follows:

- Duration, e.g. 30 minutes
- The number of measurement points multiplied by the update cycle also produces a duration.

Configuring time range

Configure the time range for all controls. Configure the time range in the time column or in the time axis for the process control and the trend control. For the function trend control, configure the time range directly at the trend.

You select one of three options for the time range:

- "Time span": You define the time range using a starting time and a following time span. You set the time span with the settings "Time range basis" and "Time range factor", for example 30 minutes.
- "Start time and end time": You define the time range using a starting time and an end time.
- "Measuring points": You define the time range using a starting time and a number of measuring points.

3.5.1.5 Representing trend directions (RT Uni)

Introduction

In a trend control, you display a trend direction with one of the following modes:

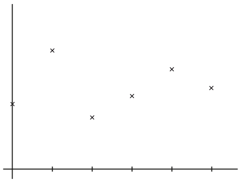
- Dots
- Interpolated
- Stepped
- Values

Select "Properties > Trend areas > Trends > Trend mode" to configure the trend display in the Inspector window.

Dots

Values are shown as dots. The display of the points can be configured as you wish.

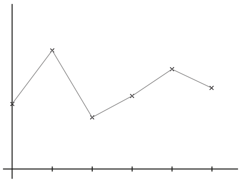
The following image shows the trend direction with the format pattern "Dots":



Interpolated

The trend curve is interpolated on a linear basis from the values. The display of the lines and points can be configured as you wish.

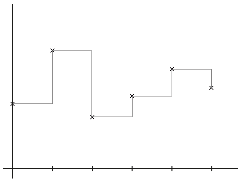
The following image shows the trend direction with the format pattern "Interpolated":



Stepped

The trend curve is interpolated as a stepped curve from the values. The display of the lines and points can be configured as you wish.

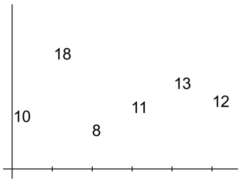
The following image shows the trend direction with the format pattern "Stepped":



Values

The trend curve is displayed as values. The display of the lines and points can be configured as you wish.

The figure below shows the trend direction with the format pattern "Values":



3.5.1.6 Outputting tag values in tabular format (RT Uni)

Introduction

To display tag values in tables in runtime, add a process control to a screen. A time stamp is displayed for each value. The values are displayed in value columns, and the time stamps in time columns. Assign the time column to one or several value columns. You have the option of configuring a time column and nine value columns in the process control.

If you assign multiple value and/or time columns to a process control, the sequence of columns in the Inspector window corresponds to the sequence in the process control. If you assign the same time column to multiple value columns, the value columns in the list are automatically grouped according to the assigned time column.

- The time range for the table display is configured using the time column. A table has a common time column for multiple value columns. The first column [0] in the process view of the process control is the time column.
- You configure the values of the process control using value columns. You can display several value columns in a table, for example to compare the fill levels of several containers. Each value column is connected to the time column.

Configuration options in the process control

You can configure the following properties in the process control in line with your requirements:

- Configure the appearance of the process control:
  - Colors
  - Time base
  - Window settings of the control
- Configure the columns of the process control in the Inspector window.
  - Configure the time range using the time columns. A table can have a common time column for multiple value columns as well as separate time columns.
  - Configure the display of the tag values using the value columns. Each value column is connected to a time column. The value columns can have a common time column.
- Configure the appearance of the table
- Configure the toolbar and status bar of the process control.
- If required, configure data export from the process control.

3.5.1.7 Configuring tag evaluation (RT Uni)

Introduction

Also configure a trend companion if you want to evaluate data from the trend control in runtime. You can also configure the trend companion as "Ruler".

You connect the trend companion to one of the following controls:

- Trend control
- Function trend control

Set a "Display mode" in the trend companion. The display mode determines which data is shown in the trend companion.

The contents of the trend companion are shown in columns. The available columns depend on the connected control. A block is assigned to each column. You define the alignment and appearance of the column headers using the blocks. By default, the format of the connected control, for example the time display, is used for the display format.

Configuration options in the trend companion

You can configure the following properties in the trend companion in line with your requirements:

- Configure the view of the trend companion in the Inspector window:
- Select the "Mode" of the trend companion under "Properties > General".
- Configure the display, labeling and sequence of the columns.

Display modes

Three different display modes are available in the trend companion:

- Ruler mode
  The ruler window shows the coordinate values of the trends on the ruler or values of a selected row in the table.
- Statistics area mode
  The statistics area window shows the values of the lower limit and upper limit of the trends between two rulers or the selected area in the table. You can only connect the statistics area window to the trend control or the process control.
- Statistics mode
  The statistics window displays the statistical evaluation of the trends. The statistics include:
  - Minimum
  - Maximum
  - Average
  - Standard deviation
  - Integral

All windows can also display additional information on the connected trends or columns, such as time stamps.

3.5.2 Configuring a trend control (RT Uni)

Introduction

For the graphic display of tag values in runtime, add a trend control to a screen. The trend control allows you to display current and logged values for a specific time window, for example. For the display of data logs in runtime, you can move the time window to view the logged values.

The list of elements in a group always starts with 0, for example trend [0] is the first trend that has already been created in the object. For a clearer display of multiple trends, you can configure different names, line types and colors.

Requirement

- Data log with backup has been configured.
- The HMI tag for temperature measurement has been configured, for example "MotorTemperature".
- The HMI tag for velocity measurement has been configured, for example "MotorSpeed".
- A screen has been configured.

Configuring the trend area and axes

1. Add the "trend control" object to the screen from the "Toolbox" task card.
2. Go to "Properties" and set the required height, width and position of the object.
3. Open the "Trend areas" group under "Properties".
   The index numbers of the trend areas created for the object are displayed.
4. Expand the index number of the first trend area.
   The properties of the first trend area are displayed.

   Note

   To add another trend area, go to "Properties > Trend areas > [0] trend areas > Trends" and click the selection button in the "Static value" column. In the dialog, click "Add".

5. Define the colors for displaying the trend area and the reference lines.
6. Configure the time axis and value axis settings under "Bottom time axis" and "Left value axis".

Configuring trends

1. Go to "Trend areas > [0] trend areas > Trends" and click on the selection button in the "Static value" column.
   A dialog opens.
2. Click "Add" in the "Index" column.
   This adds another trend. Close the dialog.
3. Expand the index number of the first trend [0]. The trend settings are displayed.
4. Specify the name of the first trend under "Display name", for example "Speed".
5. Select the entry "Online" under "Data source Y > Source".
6. Under "Tag" enter the tag "MotorSpeed".
7. Configure the line color for the trend, for example, blue.
8. Expand the index number of the second trend [1]. The trend settings are displayed.
9. Specify the name of the second trend under "Display name", for example "Temperature".
10. Specify "Online" as the source type under "Data source" and enter the name of the tag "MotorTemperature".
11. Configure the line color for the trend, for example, red.

Result

The trend control is now configured. In runtime, you monitor value changes over time on the basis of two trends. One trend shows the temperatures measured and the other trend the velocity.

Configure an additional value display if you want to evaluate the data of the trend control in runtime. You can also configure the value display as a "Ruler".

See also

Trend control (Page 82)

Configuring trend control for plant objects (Page 1536)

3.5.3 Configuring the function trend control (RT Uni)

Introduction

You use the function trend control to represent the values of a tag as a function of another tag. This means that you can present temperature trends as a function of the velocity, for example.

You can also compare the trend to a setpoint trend.

Requirement

- Data log with backup has been configured.
- The HMI tag for temperature measurement has been configured, for example "MotorTemperature".
- The HMI tag for velocity measurement has been configured, for example "MotorSpeed".
- A screen has been configured.

Configuring function trend areas and axes

1. Add the "trend control" object to the screen from the "Toolbox" task card.
2. Go to "Properties" and set the required height, width and position of the object.
3. Open the "Function trend area" group under "Properties".
   The index numbers of the function trend areas created for the object are displayed.
4. Expand the index number of the first function trend area.
   The properties of the first function trend area are displayed.

   Note

   To add another function trend area, go to "Properties > Function trend area > [0] function trend area > Function trends" and click on the selection button in the "Static value" column. In the dialog, click "Add".
5. Enter a meaningful name for the function trend area, for example, "SpeedToTemperature".

6. Open the temperature value axis settings under "Left value axis".
7. Define the value range for temperature, for example, by entering a maximum scale value of 350 degrees and a minimum scale value of 0 degrees.
8. Open the velocity value axis settings under "Bottom value axis".
9. Define the value range for speed, for example, by entering a maximum scale value of 1400 rpm and a minimum scale value of 0 rpm.

Note

Available scaling types

The f(x) trend view supports the "Linear" scaling type.

Configuring trends

1. Go to "Function trend area > [0] function trend area > Function trends > [0] function trend".
2. Specify "Online" as the source type under "Data source X", and enter the name of the process tag "MotorTemperature" under "Tag".
3. Specify "Online" as the source type under "Data source Y", and enter the name of the process tag "MotorSpeed" under "Tag".
4. Specify the time range of 1 second under "Properties > Function trend area > Function trends".

Result

The function trend control is now configured. In runtime, you monitor value changes on the basis of two trends. One trend shows the temperatures and the other trend the speed. In the function trend control, you can, for example, monitor how the temperature of the motor increases as the velocity increases.

Configure an additional value display if you want to evaluate the data of the trend control in runtime. You can also configure the value display as a "Ruler".

See also

Function trend control (Page 97)

3.5.4 Configuring the process control (RT Uni)

Introduction

To display tag values in tables in runtime, add a process control to a screen. The time column shows the time at which the value was reached. The value columns show the values at a given time stamp.

You can use the process control to display the incoming temperature values of a motor in a table in runtime, for example.

Requirement

- HMI tag for temperature measurement has been configured, for example "MotorTemperature"
- Cycle for regular display updates has been configured
- The screen is open
- The Inspector window is open

Configuring the process control

1. Add the required process control to the screen from the "Tools" task card.
2. Enter the label for the process control, for example "MotorTemperatureView", under "Properties" in the Inspector window.
3. Go to "Properties > Process view > Columns > [0]" and configure the time column with the time ranges for the table.
4. Configure the "Time range" and "Format" of the time display in the time column, for example "Time span".
5. Set the start time, the basis and the factor for the time range, for example 10 minutes.
6. If the values in the time column are to be updated automatically, enable "Update".
7. Go to "Properties > Process view > Columns > [1]" and configure the properties for the value column.
8. Enter the name of the column, for example "Temperature".
9. Configure the type "Online" for current values under "Data source" and enter the tag "MotorTemperature" under "Tag".
10. Configure the display of content and the headers for the given value column.
11. Configure the toolbar and status bar of the process control.
12. If required, configure the security settings of the process control.

Result

The process control is now configured and displays the temperature of the motor at the measured time in runtime.

See also

Process control (Page 100)

3.5.5 Configuring the trend companion (RT Uni)

Introduction

The value table allows you to display statistical data, for example mean values for the trend view with temperature trends. The calculation of statistical data gives the user access to trends and value changes over time. As well as calculating statistical data, you can use the value table as a viewing aid for trend values at a ruler position.

Requirement

- Trend view or f(x) trend view has been configured in the screen
- Cycle for regular display updates has been configured
- The screen is open
- The Inspector window is open

Procedure

1. Add the required value table to the screen from the "Toolbox" task card.
2. Select the relevant control under "Properties > Data source" in the Inspector window to connect the value table to the selected control.
3. To display the value table below the selected control, select the option "Dock to data source".
4. Select the "Value table mode" of the value table under "Properties", for example "Statistic result".
5. Configure the appearance of the selected mode under "Properties > Statistic mode appearance":
   - Change the colors, row height and fonts in the value table if required.
   - Configure the headers under "Properties > Statistic mode appearance > Header settings" if required.
6. Configure the value table columns under "Properties > Statistic mode appearance > Columns":
   - Change the display, labels or order of columns if required.
7. Configure the view of the value table in the Inspector window:
   - Change the display, labeling and colors of the value table if required, or use the colors of the control to which the value table is docked.
   - Configure the status bar and toolbar of the value table.
8. Configure the toolbar and status bar of the process control.
9. If required, configure the security settings of the process control.

Configuring selection

If, for example, a user wants to export values from a row, they must select the row. You specify the selection range and colors for selection during configuration. You define the settings for selection for each display mode.

1. In the Inspector window, go to "Properties > Trend ruler appearance" and select the "Selection mode" for the selection range, for example "Multiple elements".
2. Select the color mode for selection, for example rows.
3. If required, select the "Border color" and "Border width" to be displayed around the selection area.
4. Choose the colors for selection as required.

Result

The value table is configured. The statistical values calculated are displayed in the value table in runtime.

See also

Trend companion (Page 93)

3.5.6 Configuring toolbar and status bar (RT Uni)

Introduction

You operate the controls in runtime using the buttons in the toolbar. The status bar displays status messages from the control. During configuration, set the content of the toolbar and status bar.

Requirement

- The control is selected in the screen
- The Inspector window is open

Configuring the toolbar

1. In the Inspector window, configure the general properties of the toolbar, such as orientation and background color or displayed buttons, under "Properties > Toolbar".
2. In the Inspector window, enable the buttons you need in runtime under "Properties > Toolbar > Elements".
3. If required, configure the button display, for example background color, border and maximum and minimum size.

4. If necessary, select the authorization needed to operate the buttons in runtime.
5. If a button is not to be operated in Runtime, deselect "Allow operator control".
   You can reactivate a deactivated a button using a script in runtime, for example.

Configuring the status bar

1. In the Inspector window, configure the general properties of the status bar such as orientation and background color under "Properties > Status bar".
2. In the Inspector window, select the elements you need in runtime such as date and time under "Properties > Status bar > Elements".
3. You can adjust the display of an element in the status bar under "Properties > Status Bar > Elements" for the respective element.

3.5.7 Defining the data source (RT Uni)

Introduction

Using the data source, you define the sources from which the values are displayed on the HMI device in runtime. The following sources are available:

- Current process values from tags
- Archived values from logging tags

To set up data supply for the controls over a tag, enter the name of the tag in the "Static value" column under "Data source > Tag".

Requirement

- An online tag or logging tag is configured
- Value column or trend has been created
- The Inspector window is open

Displaying current process values

Proceed as follows to display current process values:

1. Click "Properties > Process view > Columns" in the Inspector window to define the data source for a process control.
   The first column is always reserved for the time column. You enter the data source for value columns [1] to [N].
2. Click "Properties > Trend areas > Trends" in the Inspector window to define the data source for a trend control.
   For the function trend control, click on "Properties > Function trend area > Function trends".
3. Configure the "data source":
   – Select the entry "Online" as "Source type".
   – When you configure the trend of an function trend control, enter one tag each for "Data source X" and "Data source Y".
   – When you configure the trend of a trend control or a value column, enter the corresponding tag under "Tag".
   – Select the update cycle.

Note

Using UDTs

In the "Static value" column under "Tag" first enter the name of the data type and then the name of the element separated by a period, for example, "PLCDatatypeName.ElementName".

Displaying values from a log

Proceed as follows to display values from a log:

1. Click "Properties > Process view > Columns" in the Inspector window to define the data source for a process control.
   The first column is always reserved for the time column. You enter the data source for value columns [1] to [N].
2. Click "Properties > Trend areas > Trends" in the Inspector window to define the data source for a trend control.
   For the function trend control, click on "Properties > Function trend area > Function trends".
3. Configure the "data source":
   – Select the entry "Logs" as "Source".
   – When you configure the trend of an function trend control, enter one tag each for "Data source X" and "Data source Y".
   – When you configure the trend of a trend control or a value column, enter the corresponding tag under "Tag".

Note

Using logging tags

In the "Static value" column under "Tag" first enter the name of the HMI tag and then the name of the associated logging tag separated by a period, for example, "HMITag_1:LoggingTag_1".

3.6 Reference (RT Uni)

3.6.1 Quality codes of HMI tags (RT Uni)

Introduction

The "Quality Code" is required to evaluate the status and quality of a tag. The quality of the entire value transfer and value processing of the respective HMI tag is summarized in the indicated Quality Code. For example, it is possible to determine from the Quality Code whether the current value is a start value or substitute value.

The quality codes are prioritized. If several codes occur at the same time, the Quality Code reflecting the lowest quality is displayed.

Evaluation of Quality Codes

You can evaluate the Quality Code in a number of different ways:

- Evaluation with JScript functions
- Evaluation using the "Quality Code changed" event of an I/O field.

Structure

The Quality Code has the following binary structure:

High byte: Specific information for WinCC Unified								Low byte: Quality code according to PROFIBUS PA or OPC DA							
Flags				Enhanced substatus				Quality		Substatus				Limits	
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1

Quality (bit 7 and bit 8)

Quality represents the basic values of the quality levels. Making use of the substatus and limits gives rise to intermediate values over and above the quality stage concerned.

Bit 8	Bit 7	
0	0	Bad - The value is not useful. The reasons are indicated by the sub-status.
0	1	Uncertain - The quality of the value is less than normal, but the value may still be useful. The reasons are indicated by the sub-status.
1	0	Good (Non-Cascade) - The quality of the value is good. Possible alarm conditions may be indicated by the sub-status.
1	1	Good (Cascade) - The quality of the value is good and may be used in control.

Flags (bit 12 to bit 15)

Flags contain information on the interpretation of the Quality Code.

Bit 12	Source quality	0: The data quality has been determined and assigned by external data source.
Bit 13	Source time	1: The data timestamp has been produced and assigned by external data source.
Bit 14	Time corrected	1: The data timestamp applied by external data source has been corrected by the system. Thus, Bit 13 "Source time" is not set. Time correction happens if the external timestamp is older than the timestamp of the last known value.
Bit 15	reserved	

Sub-status and extended sub-status

The quality alone is not enough. Substatuses divide the individual qualities. The Quality Code is binary-coded. The value must be converted to hexadecimal format for the analysis of the Quality Code.

Externally generated quality code of tags

If bit 12 is not set, the Quality Code was generated from an external source in accordance with PROFIBUS PA. The table begins with the worst Quality Code and ends with the best Quality Code. The best Quality Code has the lowest priority, while the worst Quality Code has the highest priority. If several statuses occur for one tag in the process, the poorest code is passed on.

Code (hex)	**Quality**		**Q**	**Q**	**S**	**S**	**S**	**S**	**L**	**L**
0x23	Bad	Device passivated - Diagnostic alerts inhibited	0	0	1	0	0	0	1	1
0x3C	Bad	Function check - Local override	0	0	1	1	1	1	1	1
0x04	Bad	Configuration Error - Set if the value is not useful because there is some inconsistency regarding the parameterization or configuration, depending on what a specific manufacturer can detect.	0	0	0	0	0	1	-	-

Code (hex)	Quality		Q	Q	S	S	S	S	L	L
0x1C	Bad	Out of Service - The value is not reliable because the block is not being evaluated, and may be under construction by a configurer. Set if the block mode is O/S.	0	0	0	1	1	1	-	-
0x73	Uncertain	Simulated value - Start	0	1	1	1	0	0	1	1
0x74	Uncertain	Simulated value - End	0	1	1	1	0	1	-	-
0x84	Good (Non-Cascade)	Active Update event - Set if the value is good and the block has an active Update event.	1	0	0	0	0	1	-	-
0x24	Bad	Maintenance alarm - More diagnostics available.	0	0	1	0	0	1	-	-
0x18	Bad	No Communication, with no usable value - Set if there has never been any communication with this value since it was last "Out of Service".	0	0	0	1	1	0	-	-
0x14	Bad	No Communication, with last usable value - Set if this value had been set by communication, which has now failed.	0	0	0	1	0	1	-	-
0x0C	Bad	Device Failure - Set if the source of the value is affected by a device failure.	0	0	0	0	1	1	-	-
0x10	Bad	Sensor failure	0	0	0	1	0	0	-	-
0x08	Bad	Not Connected - Set if this input is required to be connected and is not connected.	0	0	0	0	1	0	-	-
0x00	Bad	non-specific - There is no specific reason why the value is bad. Used for propagation.	0	0	0	0	0	0	-	-
0x28	Bad	Process related - Substitute value	0	0	1	0	1	0	-	-
0x2B	Bad	Process related - No maintenance	0	0	1	0	1	0	1	1
0x68	Uncertain	Maintenance demanded	0	1	1	0	1	0	-	-
0x60	Uncertain	Simulated value - Set when the process value is written by the operator while the block is in manual mode.	0	1	1	0	0	0	-	-
0x64	Uncertain	Sensor calibration	0	1	1	0	0	1	-	-
0x5C	Uncertain	Configuration error	0	1	0	1	1	1	-	-
0x58	Uncertain	Sub-normal	0	1	0	1	1	0	-	-
0x54	Uncertain	Engineering Unit Range Violation - Set if the value lies outside of the set of values defined for this parameter. The Limits define which direction has been exceeded.	0	1	0	1	0	1	-	-
0x50	Uncertain	Sensor conversion not accurate	0	1	0	1	0	0	-	-
0x4B	Uncertain	Substitute (constant)	0	1	0	0	1	0	1	1
0x78	Uncertain	Process related - No maintenance	0	1	1	1	1	0	-	-
0x4C	Uncertain	Initial Value - Value of volatile parameters during and after reset of the device or of a parameter.	0	1	0	0	1	1	-	-
0x48	Uncertain	Substitute value - Predefined value is used instead of the calculated one. This is used for fail safe handling.	0	1	0	0	1	0	-	-
0x44	Uncertain	Last Usable Value - Whatever was writing this value has stopped doing so. This is used for fail safe handling.	0	1	0	0	0	1	-	-
0x40	Uncertain	Non-specific - There is no specific reason why the value is uncertain. Used for propagation.	0	1	0	0	0	0	-	-
0xE0	Good(Cascade)	Initiate Fail Safe (IFS) - The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe.	1	1	0	1	1	0	-	-

Code (hex)	Quality		Q	Q	S	S	S	S	L	L
0xD8	Good (Cascade)	Local Override (LO) - The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control must be propagated to a function running in a host system for alarm and display purposes. This also implies "Not Invited".	1	1	0	1	1	0	-	-
0xD4	Good (Cascade)	Do Not Select (DNS) - The value is from a block which should not be selected, due to conditions in or above the block.	1	1	0	1	0	1	-	-
0xCC	Good (Cascade)	Not Invited (NI) - The value is from a block which does not have a target mode that would use this input.	1	1	0	0	1	1	-	-
0xC8	Good (Cascade)	Initialization Request (IR) - The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.	1	1	0	0	1	0	-	-
0xC4	Good (Cascade)	Initialization Acknowledge (IA) - The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).	1	1	0	0	0	1	-	-
0xA0	Good (Non-Cascade)	Initiate fail safe	1	0	1	0	0	0	-	-
0x98	Good (Non-Cascade)	Unacknowledged Critical Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.	1	0	0	1	1	0	-	-
0x94	Good (Non-Cascade)	Unacknowledged Advisory Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.	1	0	0	1	0	1	-	-
0x90	Good (Non-Cascade)	Unacknowledged Update event - Set if the value is good and the block has an unacknowledged Update event.	1	0	0	1	0	0	-	-
0x8C	Good (Non-Cascade)	Active Critical Alarm - Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.	1	0	0	0	1	1	-	-
0x88	Good (Non-Cascade)	Active Advisory Alarm - Set if the value is good and the block has an active Alarm with a priority less than 8.	1	0	0	0	1	0	-	-
0xA8	Good (Non-Cascade)	Maintenance demanded	1	0	1	0	1	0	-	-
0xA4	Good (Non-Cascade)	Maintenance required	1	0	1	0	0	1	-	-
0xBC	Good (Non-Cascade)	Function check - Local override	1	0	1	1	1	1	-	-
0xC0	Good(Cascade)	OK - No error or special condition is associated with this value.	1	1	0	0	0	0	-	-
0x80	Good (Non-Cascade)	OK - No error or special condition is associated with this value.	1	0	0	0	0	0	-	-

Internally generated quality code of tags

If bit 12 is set, the Quality Code was generated from the HMI system. The table begins with the worst Quality Code and ends with the best Quality Code. The best Quality Code has the lowest priority, while the worst Quality Code has the highest priority. If several statuses occur for one tag in the process, the poorest code is passed on.

Code (Hex)	Quality		Q	Q	S	S	S	S	L	L
0x70n	Bad	Disabled	0	0	0	0	-	-	-	-
0x300	Bad	Unusable value - A logged value has been identified to be incorrect, but a respective correction value is not available. The corresponding sub-status is set to ‘non-specific’.	0	0	0	0	0	0	0	0
0x23	Bad	Device passivated - Diagnostic alerts inhibited	0	0	1	0	0	0	1	1
0x3F	Bad	Function check - Local override	0	0	1	1	1	1	1	1
0x04	Bad	Configuration Error - Set if the value is not useful because there is some inconsistency regarding the parameterization or configuration, depending on what a specific manufacturer can detect.	0	0	0	0	0	1	-	-
0x1C	Bad	Out of Service - The value is not reliable because the block is not being evaluated, and may be under construction by a configurer. Set if the block mode is O/S.	0	0	0	1	1	1	-	-
0x73	Uncertain	Simulated value - Start	0	1	1	1	0	0	1	1
0x74	Uncertain	Simulated value - End	0	1	1	1	0	1	-	-
0x84	Good (Non-Cascade)	Active Update event - Set if the value is good and the block has an active Update event.	1	0	0	0	0	1	-	-
0x24	Bad	Maintenance alarm - More diagnostics available.	0	0	1	0	0	1	-	-
0x18	Bad	No Communication, with no usable value - Set if there has never been any communication with this value since it was last "Out of Service".	0	0	0	1	1	0	-	-
0x14	Bad	No Communication, with last usable value - Set if this value had been set by communication, which has now failed.	0	0	0	1	0	1	-	-
0x0C	Bad	Device Failure - Set if the source of the value is affected by a device failure.	0	0	0	0	1	1	-	-
0x10	Bad	Sensor failure	0	0	0	1	0	0	-	-
0x08	Bad	Not Connected - Set if this input is required to be connected and is not connected.	0	0	0	0	1	0	-	-
0x100	Bad	Aggregated value - The value has been calculated out of multiple values with less than the re-quired number of good sources. This includes data aggregation by means of data compression algorithms. The corresponding sub-status is set to ‘non-specific’.	0	0	0	0	0	0	0	0
0x00	Bad	non-specific - There is no specific reason why the value is bad. Used for propagation.	0	0	0	0	0	0	-	-
0x28	Bad	Process related - Substitute value	0	0	1	0	1	0	-	-
0x74n	Uncertain	Disabled - The provider of the value, e.g. logging tag for logged value, has been disabled and the previous value was GOOD or UNCERTAIN. In case of GOOD the corresponding sub- status is set to ‘last usable value’. In case of UNCERTAIN the corresponding sub-status is taken from the last sub-status.	0	1	0	0	-	-	-	-

Code (Hex)	Quality		Q	Q	S	S	S	S	L	L
0x158	Uncertain	Aggregated value - The value has been calculated out of multiple values with less than the required number of good sources to be GOOD as well as less than required number of bad sources to be BAD. This includes data aggregation by means of data compression algorithms. The corresponding sub-status is set to ‘sub-normal’.	0	1	0	1	1	0	0	0
0x2B	Bad	Process related - No maintenance	0	0	1	0	1	0	1	1
0x68	Uncertain	Maintenance demanded	0	1	1	0	1	0	-	-
0x60	Uncertain	Simulated value - Set when the process value is written by the operator while the block is in manual mode.	0	1	1	0	0	0	-	-
0x64	Uncertain	Sensor calibration	0	1	1	0	0	1	-	-
0x5C	Uncertain	Configuration error	0	1	0	1	1	1	-	-
0x58	Uncertain	Sub-normal	0	1	0	1	1	0	-	-
0x54	Uncertain	Engineering Unit Range Violation - Set if the value lies outside of the set of values defined for this parameter. The Limits define which direction has been exceeded.	0	1	0	1	0	1	-	-
0x50	Uncertain	Sensor conversion not accurate	0	1	0	1	0	0	-	-
0x4B	Uncertain	Substitute (constant)	0	1	0	0	1	0	1	1
0x78	Uncertain	Process related - No maintenance	0	1	1	1	1	0	-	-
0x4C	Uncertain	Initial Value - Value of volatile parameters during and after reset of the device or of a parameter.	0	1	0	0	1	1	-	-
0x48	Uncertain	Substitute value - Predefined value is used instead of the calculated one. This is used for fail safe handling.	0	1	0	0	1	0	-	-
0x44	Uncertain	Last Usable Value - Whatever was writing this value has stopped doing so. This is used for fail safe handling.	0	1	0	0	0	1	-	-
0x40	Uncertain	Non-specific - There is no specific reason why the value is uncertain. Used for propagation.	0	1	0	0	0	0	-	-
0x3C0	Good(Cascade)	Corrected value - A logged value has been corrected. The corresponding sub-status is set to ‘non-specific’.	1	1	0	0	0	0	0	0
0x2C0	Good(Cascade)	Manual input - A logged value has been created manually. The corresponding sub-status is set to ‘non-specific’.	1	1	0	0	0	0	0	0
0x1C0	Good(Cascade)	Aggregated value -The value has been calculated out of multiple (GOOD) values. This includes data aggregation by means of data compression algorithms. The corresponding sub-status is set to ‘non-specific’.	1	1	0	0	0	0	0	0
0xE0	Good (Cascade)	Initiate Fail Safe (IFS) - The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe.	1	1	1	0	0	0	-	-
0xD8	Good (Cascade)	Local Override (LO) - The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control must be propagated to a function running in a host system for alarm and display purposes. This also implies "Not Invited".	1	1	0	1	1	0	-	-
0xD4	Good (Cascade)	Do Not Select (DNS) - The value is from a block which should not be selected, due to conditions in or above the block.	1	1	0	1	0	1	-	-

Code (Hex)	Quality		Q	Q	S	S	S	S	L	L
0xCC	Good (Cascade)	Not Invited (NI) - The value is from a block which does not have a target mode that would use this input.	1	1	0	0	1	1	-	-
0xC8	Good (Cascade)	Initialization Request (IR) - The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.	1	1	0	0	1	0	-	-
0xC4	Good (Cascade)	Initialization Acknowledge (IA) - The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).	1	1	0	0	0	1	-	-
0x6C0	Good(Cas cade)	Initial value - The local data source has been initialized with the configured initial value. The corresponding sub-status is set to ‘non-specific’.	1	1	0	0	0	0	0	0
0c4C0	Good(Cas cade)	Last usable value - The local data source has been initialized with the last usable value, if pre-sent inside a local persistency. The corresponding sub-status is set to ‘non-specific’.	1	1	0	0	0	0	0	0
0xA0	Good (Non-Cascade)	Initiate fail safe	1	0	1	0	0	0	-	-
0x98	Good (Non-Cascade)	Unacknowledged Critical Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.	1	0	0	1	1	0	-	-
0x94	Good (Non-Cascade)	Unacknowledged Advisory Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.	1	0	0	1	0	1	-	-
0x90	Good (Non-Cascade)	Unacknowledged Update event - Set if the value is good and the block has an unacknowledged Update event.	1	0	0	1	0	0	-	-
0x8C	Good (Non-Cascade)	Active Critical Alarm - Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.	1	0	0	0	1	1	-	-
0x88	Good (Non-Cascade)	Active Advisory Alarm - Set if the value is good and the block has an active Alarm with a priority less than 8.	1	0	0	0	1	0	-	-
0xA8	Good (Non-Cascade)	Maintenance demanded	1	0	1	0	1	0	-	-
0xA4	Good (Non-Cascade)	Maintenance required	1	0	1	0	0	1	-	-
0xBC	Good (Non-Cascade)	Function check - Local override	1	0	1	1	1	1	-	-
0xC0	Good(Cas cade)	OK - No error or special condition is associated with this value.	1	1	0	0	0	0	-	-
0x80	Good (Non-Cascade)	OK - No error or special condition is associated with this value.	1	0	0	0	0	0	-	-

Limit

The Quality Codes can be further subdivided by limits. Limits are optional.

	Q	Q	S	S	S	S	L	L
O.K. - The value is free to move.	-	-	-	-	-	-	0	0
Low limited - The value has acceded its low limits.	-	-	-	-	-	-	0	1
High limited - The value has acceded its high limits.	-	-	-	-	-	-	1	0
Constant (high and low limited) - The value cannot move, no matter what the process does.	-	-	-	-	-	-	1	1

Quality Codes in Communication with OPC

When connecting to a OPC UA server, the OPC UA status code is shown in a quality code.

Quality Code in WinCC	Status code according to OPC
0x48	0x40
0x4C	0x40
0x5C	0x40
0x60	0x40
0x80...0xD4	0xC0
0xD8	0xC0

3.6.2 Data types (RT Uni)

3.6.2.1 Data types for SIMATIC S7-300/400 (RT Uni)

Overview

The following table shows the data types for SIMATIC S7-300/400 with the corresponding HMI data types and value ranges in WinCC.

Data type	Value range
Bool	0 (FALSE), 1 (TRUE)
Byte	0 ... 255
Char	0 ... 255 (ASCII)
Counter	0 ... 999
Date	1990-01-01 ... 2168-12-31
Date_And_Time	1990-1-1-0:0:0.0 ... 2089-12-31-23:59:59.999
DInt	−2147483648 … +2147483647
DWord	0 .. 4294967295
Int	−32768 … 32767
Real	±1.17549E-38 to ±3.40282E+38 and 0.0
S5Time	0 … 2h46m30s0ms

Data type	Value range
String	ASCII
Time[1]	-24d20h31m23s648ms ... +24d20h31m23s647ms
Time_of_Day, TOD	00:00:00 ... 23:59:59.999
Timer	-0ms ... 2h46m30s0ms
Word	0 ... 65535

1: If the value is set via the HMI, then the granularity is in 100 nanosecond intervals. In contrast, the granularity of WinCC Advanced, WinCC Comfort and WinCC Professional is milliseconds.

3.6.2.2 Data types for SIMATIC S7-1500 (RT Uni)

Overview

The following table shows the data types for SIMATIC S7-1500 with the corresponding HMI data types and value ranges in WinCC.

Data type	Value range
Bool	0 (FALSE), 1 (TRUE)
Byte	0 ... 255
Char	0 … 255 (ASCII)
Counter	0 … 65535
Date	1990-01-01 ... 2168-12-31
Date_And_Time, DT	1990-1-1-0:0:0.0 ... 2089-12-31-23:59:59.999
DInt	−2147483648 … +2147483647
DTL	1970-01-01-00:00:00.0 ... 2262-04-11-23:47:16.854775807
DWord	0 ... 4294967295
Int	-32768 ... +32767
LDT	1970-01-01-00:00:00.000000000 ... 2263-04-11-23:47:16.854775808
LInt	-9223372036854775808 ... +9223372036854775807
LReal	±1.79769313486231E+308 ... ±2.22507385850720E-308 and 0.0
LTime	106751d23h47m16s854ms775us808ns ... +106751d23h47m16s854ms775us807ns
LTime_of_Day, LTOD	00:00:00.000000000 ... 23:59:59.999999999
LWord	0 ... 18446744073709551615
Real	±1.17549E-38 ... ±-3.40282E+38 and 0.0
S5Time	0ms ... 2h46m30s0ms
SInt	-128 ... +127
String	ASCII
Time[1]	-24d20h31m23s648ms ... +24d20h31m23s647ms
Time_of_Day, TOD	00:00:00 ... 23:59:59.999
Timer	-24d20h31m23s648ms ... +24d20h31m23s647ms

Data type	Value range
UDInt	0 ... 4294967295
UInt	0 … 65535
ULInt	0 ... 18446744073709551615
USInt	0 ... 255
WChar	UNICODE
Word	0 … 65535
WString	UNICODE
PLCUDT	-

1: If the value is set via the HMI, then the granularity is in 100 nanosecond intervals. In contrast, the granularity of WinCC Advanced, WinCC Comfort and WinCC Professional is milliseconds.

3.6.2.3 User-defined PLC data types (UDT) (RT Uni)

Overview

You can connect with the HMI tags and DB instances of user-defined PLC data types (UDT).

The PLC data type and the corresponding DB instances are created and updated centrally in STEP 7. In WinCC, you can use the following sources as PLC tag (DB instances):

- Data block elements that use a UDT as data type
- Instance data blocks of a UDT

The data type is taken from STEP 7 and is not converted into an HMI data type. The access type is always "Symbolic access".

Elements of a PLC data type

You have access to the following elements in WinCC with a structured PLC data type:

- Elements that have been released for WinCC in STEP 7.
- Elements whose data types are supported in WinCC.

Note

Invalid elements of a PLC data type in WinCC

Invalid elements generate an error in WinCC.

If you disable the "Accessible from HMI" option for the corresponding elements of the associated PLC data type in STEP 7, these elements are excluded in WinCC.

Naming conventions

The following characters are invalid in the name of the PLC data type and generate an error in WinCC:

- Period: "."
- Brackets: "(" and ")"

Properties

The properties of the PLC data type and its elements are adopted in WinCC. Depending on the data type used, the properties are read-only or can be written to in WinCC.

If you change the connection of the PLC data type in WinCC, all elements of the PLC data type are deleted and the properties of the newly connected PLC tag are used.

In WinCC, you have access to STEP 7 comments on elements of the PLC data type.

You have limited access to properties in WinCC for the following elements of PLC data types:

- Elements of the data type "Struct"
- PLC data type
- Multidimensional arrays
- Array of complex data types except "DTL"

Mapping of the data type "DTL"

If a PLC data type contains elements of the data type "DTL", these elements are mapped in WinCC without lower-level elements. The data type "DTL" turns into "DateTime" in WinCC.

Tags with elements of the "DTL" data type

Tags that use the "DTL" data type element by element can only be used as read-only with symbolic addressing, e.g. with SIMATIC S7 1500. With absolute addressing, write access is also possible.

Configuring alarms (RT Uni) 4

4.1 Basics (RT Uni)

4.1.1 Alarm system (RT Uni)

Introduction

Alarms display events, operating states or faults that occur or predominate in your plant. You can use alarms for diagnostic purposes for fault rectification, for example, and they help you rapidly to identify the cause of a fault. You can adjust your processes through targeted intervention so that compliant products continue to be produced despite the fault, or the process is stabilized and the fault only causes a minimal loss of production.

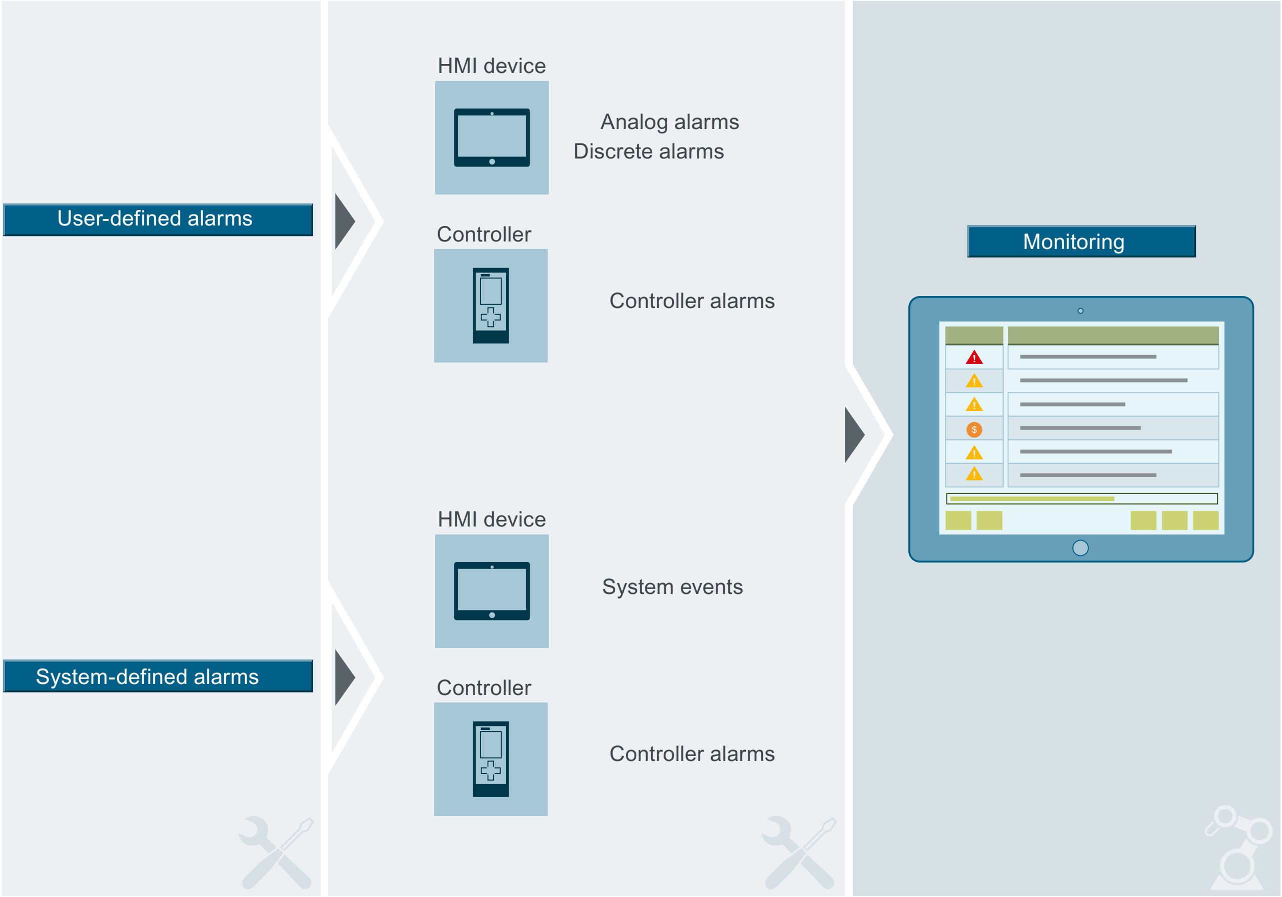
WinCC has a whole range of technical tools for implementing an alarm system. You use these tools to set up an alarm system that meets all requirements under currently applicable national and international standards and guidelines.

By means of the alarm system, events from the monitoring function in WinCC are displayed in form of alarms, acknowledged by the operator and logged, if necessary. To do this, alarms must be configured that are separated into alarm classes.

Alarm system

The alarm system distinguishes between the following alarms:

- User-defined alarms:
  - Analog alarms: Show limit violations (value changes), are used for monitoring the plant.
  - Discrete alarms: Show status changes, are used for monitoring the plant.
  - User-defined controller alarms: are configured in STEP 7, show status values of the controller, are used to monitor the plant.
- System-defined alarms:
  - System events: belong to the HMI device and are used to monitor it.
  - System-defined controller alarms: consist of system diagnostic alarms and system errors (RSE) and are used to monitor the controller.



The detected alarm events are displayed on the HMI device.

Note

Note the following restrictions for controller alarms.

- WinCC only supports controller alarms of a SIMATIC S7-1500 controller.
- WinCC only supports controller alarms that are automatically updated by the central alarm management in the controller.

Tips for effective procedure

- When you configure an alarm system, you need to take account of the abilities of future users.
- Alarm systems must be designed to use and allow for characteristic aspects of human perception.
- Important alarms must be highlighted so that they are noticed rapidly. The display of important information should be redundant to make it easier to see.
- Supplementary information about individual alarms ensures that faults are localized and cleared quickly.
- Information should if possible be directed at more than one sense (for example, visible and sound signals). Only alarm systems that meet these criteria will help the user to monitor and control the plant.

See also

Alarms (Page 215)

Alarm states (Page 220)

Alarm classes (Page 221)

Acknowledging alarms (Page 225)

Acknowledgment model (Page 226)

Alarm components and properties (Page 227)

Configuring analog alarms (Page 241)

4.1.2 Alarms (RT Uni)

4.1.2.1 User-defined alarms (RT Uni)

Analog Alarms (RT Uni)

Description

Analog alarms indicate limit violations. You have defined in advance a limit value for the trigger tag and the trigger mode. An analog alarm is triggered depending on which mode you have defined, for example, when the value is higher than, lower than or the same as the defined value.

Example

The speed of a motor must not be too high or too low. You can configure analog alarms to monitor the speed of the motor. If the high or low limit for the speed of the motor is violated, an alarm is output on the HMI device containing the following alarm text, for example: "Motor speed is too low".

See also

Configuring analog alarms (Page 241)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

Discrete alarms (Page 216)

User-defined controller alarms (Page 216)

System Alarms (Page 218)

System-defined controller alarms (Page 218)

Alarm system (Page 213)

Discrete alarms (RT Uni)

Description

Discrete alarms triggered by the PLC indicate status changes in a plant. A discrete alarm is triggered by a specific value (bit) of a tag.

Example

Imagine that the state of a valve is to be monitored during operation. The two possible valve states are "opened" and "closed". In this case, a discrete alarm is configured for each valve state. A discrete alarm containing the following alarm text is output, for example, when the state of this valve changes: "Valve closed".

See also

Configuring discrete alarms (Page 238)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

Analog Alarms (Page 215)

User-defined controller alarms (Page 216)

System Alarms (Page 218)

System-defined controller alarms (Page 218)

Alarm system (Page 213)

User-defined controller alarms (RT Uni)

Example of an alarm

"The temperature in Tank 2 is too high."

Description

A user-defined controller alarm, e.g. a program alarm, created by the control project engineer in STEP 7. The PLC status values, such as time stamp and process values, are mapped in the controller alarm. If controller alarms are configured in STEP 7, accept them into the integrated WinCC operation as soon as a connection is established to the PLC. In STEP 7, the controller alarm is assigned to an alarm class. You import this alarm class with the controller alarm as a common alarm class.

Note

Automatic update of new or modified controller alarms on the HMI device

If controller alarms are configured in STEP 7 and an HMI connection to a SIMATIC S7-1500 controller (firmware version 2.0 or higher) is established, controller alarms are sent to the HMI device. After changes of the controller alarms, the HMI device configuration must no longer be transferred. The prerequisite is that the option "Central alarm management in the PLC" is enabled in the properties of the controller. In addition, the option "Automatic update" must be enabled in the runtime settings of the HMI device under "Alarms > Controller alarms".

Note

WinCC only supports controller alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports controller alarms that are automatically updated by the central alarm management in the controller.

Controller alarms for multiple HMI devices

If a PLC is connected to multiple HMI devices, the project engineer assigns display classes to the controller alarms in STEP7. The display classes determine the allocation to the HMI device. You can activate the display classes for your HMI device that are to be displayed on it. In this case, only the controller alarms from this display class will be displayed on the HMI device. Up to 17 display classes are possible.

See also

Filtering controller alarms via display classes (Page 249)

Sending and automatically updating complete alarm from the controller to the HMI device (Page 287)

Configuring automatic update of controller alarms on the HMI device (Page 288)

Analog Alarms (Page 215)

Discrete alarms (Page 216)

System Alarms (Page 218)

System-defined controller alarms (Page 218)

Alarm system (Page 213)

4.1.2.2 System-defined alarms (RT Uni)

System Alarms (RT Uni)

Example of an alarm

"Memory is full!"

Description

A system event indicates the system status and communication errors between the HMI device and system. System events are output in runtime in the configured alarm view. System events are output in the language currently set on your HMI device.

The time format (AM/PM or 24-hour format) is based on the selected language. If no translation of the alarm texts exists in this language, English is used as replacement and the corresponding time format is displayed.

See also

Editing system events (Page 248)

Analog Alarms (Page 215)

Discrete alarms (Page 216)

User-defined controller alarms (Page 216)

System-defined controller alarms (Page 218)

Alarm system (Page 213)

System-defined controller alarms (RT Uni)

Example of an alarm

"CPU maintenance required"

Description

System-defined controller alarms are installed with STEP 7 and are only available if WinCC is operated in the STEP 7 environment.

System-defined controller alarms are used to monitor states and events of a controller. System-defined controller alarms consist of system diagnostic alarms and system errors (RSE)

Note

Automatic update of system diagnostic alarms on the HMI device

If an HMI connection to a SIMATIC S7-1500 controller (firmware version 2.0 or higher) is established, system diagnostic alarms are sent to the HMI device and automatically updated. The prerequisite is that the option "Central alarm management in the PLC" is enabled in the properties of the controller. In addition, the options "Automatic update" and "System diagnostics" must be enabled in the runtime settings of the HMI device under "Alarms > Controller alarms".

Note

Note the following restrictions:

- WinCC only supports system diagnostic alarms of a SIMATIC S7-1500 controller.
- WinCC only supports system diagnostic alarms that are automatically updated by the central alarm management in the controller.

See also

Configuring the display of system diagnostic alarms (Page 262)

Sending and automatically updating complete alarm from the controller to the HMI device (Page 287)

Configuring automatic update of controller alarms on the HMI device (Page 288)

Analog Alarms (Page 215)

Discrete alarms (Page 216)

User-defined controller alarms (Page 216)

System Alarms (Page 218)

Alarm system (Page 213)

4.1.3 Alarm states (RT Uni)

Description

Every alarm has an alarm state. The alarm states are made up of the following events:

- **Incoming**
  The condition for triggering an alarm is fulfilled. The alarm is displayed, such as "Boiler pressure too high".
- **Outgoing**
  The condition for triggering an alarm is no longer fulfilled. The alarm is no longer displayed as the boiler was vented.
- **Acknowledged**
  The operator has acknowledged the alarm.

Alarms without acknowledgment

The following table shows the alarm states for alarms that do not have to be acknowledged:

Status	Description
Incoming	The condition for an alarm is fulfilled. This alarm state is only visible for logged alarms.
Incoming/outgoing	The condition for an alarm is fulfilled; the alarm must not be acknowledged. The condition for an alarm is no longer fulfilled. The alarm is no longer pending.

Alarms with acknowledgment

The following table shows the alarm states for alarms that have to be acknowledged:

Status	Description
Incoming	The condition for an alarm is fulfilled.
Incoming/acknowledged	The condition for an alarm is fulfilled. The operator has acknowledged the alarm.
Incoming/acknowledged/outgoing	The condition for an alarm is fulfilled. The operator has acknowledged the alarm. The operator removes the cause which has triggered the event. The alarm is no longer pending.
Incoming/outgoing/acknowledged	The condition for an alarm is fulfilled. The operator removes the cause which has triggered the event. The condition for an alarm is no longer fulfilled. The operator has acknowledged the alarm after this time.

Note

The display text for the states of an alarm is language-specific and configuration-specific.

Shelving alarms

You shelve an alarm, for example, to prevent an error alarm from impairing the effectivity of your system.

- Shelved: The alarm was shelved.
- Unshelved: The alarm was unshelved. The alarm is visible again in its last state.

Locked alarms

You suppress the display of specific alarms, for example, to avoid an excessive burden of information for the plant operator.

- Suppressed manually: The alarm was suppressed manually.
- Suppressed by design: The alarm was suppressed automatically.

See also

Alarm system (Page 213)

Alarm classes (Page 221)

Acknowledging alarms (Page 225)

Acknowledgment model (Page 226)

Alarm components and properties (Page 227)

4.1.4 Alarm classes (RT Uni)

Introduction

Many alarms occur in a plant. These are all of different importance. You can assign the alarms of your project to alarm classes to clearly show the operator which of the alarms are most important.

Description

Every alarm must be assigned to an alarm class when you create new alarms. The alarm class hereby defines the appearance and the acknowledgment model of the alarm (single-mode acknowledgment, acknowledgment and confirmation, no acknowledgment).

A new alarm class with mandatory acknowledgment is generated in WinCC. Predefined alarm classes are available for each device.

Examples of how to use alarm classes

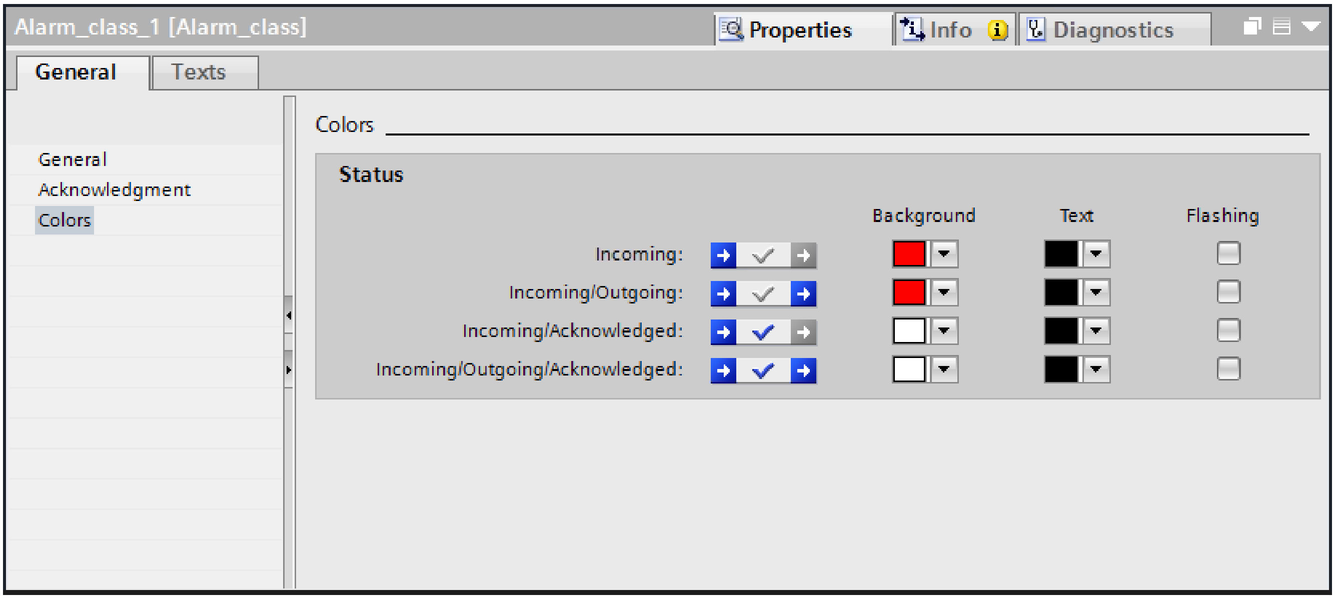
- The alarm class of the alarm "Fan 1 speed in upper tolerance range" is "Warning". The alarm is displayed with a yellow background. The alarm requires acknowledgment.
- The alarm "Speed of fan 2 has exceeded upper warning range" is assigned to the "Alarm" alarm class. The alarm is displayed with a red background and flashes at high frequency in runtime. The alarm is displayed until the alarm is gone and the operator has acknowledged it.

Using alarm classes

Use the following alarm classes to define the state machines and appearance of alarms for your project:

- Predefined alarm classes
  You cannot delete predefined alarm classes and edit them only to a limited extent. Predefined alarm classes are available under "HMI alarms > Alarm classes".
- Custom alarm classes
  You can create new alarm classes under "HMI alarms > Alarm classes", configure how you want the alarms to be displayed, and define an acknowledgment model for alarms of this alarm class. The possible number of custom alarm classes depends on which runtime is used in your project.
- Common alarm classes
  Common alarm classes are displayed under "Common data > Alarm classes" in the project tree and can be used for the alarms of an HMI device. Common alarm classes are used in STEP 7 for controller alarms. If required, create additional common alarm classes in WinCC. Common alarm classes are divided into predefined and user-defined common alarm classes. The predefined common alarm classes are "Acknowledgement" (for alarms with acknowledgment) and "No Acknowledgement" (for alarms without acknowledgment).

For each alarm class (including predefined alarm classes), you can configure the font color, background color and flashing for the alarm states "Incoming", "Incoming/outgoing", "Incoming/acknowledged", "Incoming/outgoing/acknowledged":



Alarm_class_1 [Alarm_class]

Colors

Status

		Background	Text	Flashing
Incoming:	→ ✓ →	🟥	⬛	☐
Incoming/Outgoing:	→ ✓ →	🟥	⬛	☐
Incoming/Acknowledged:	→ ✓ →	⬜	⬛	☐
Incoming/Outgoing/Acknowledged:	→ ✓ →	⬜	⬛	☐

Predefined alarm classes

The following predefined alarm classes are available under "Alarm classes" in the "HMI alarms":

- "Critical"
  Alarms in this class must always be acknowledged. The alarm class "Critical" is designed to show critical faults in the plant, for example, "Motor temperature too high".
- "System notification"
  The operator does not acknowledge alarms from this alarm class.
- "System alarm without clear event"
  Alarms of this class have no "Outgoing" state and must be acknowledged.
- "System warning without clear event"
  Alarms of this class have no "Outgoing" state and must be acknowledged.
- "System alarm"
  Alarms in this class must be acknowledged.
- "System warning"
  Alarms in this class must be acknowledged.
- "Information"
  Alarms in this class contain general information on your system, have no state and cannot be acknowledged.
- "Notification"
  The alarm class "Notification" alarm class is designed to show irregular states and routines in the process. The operator does not acknowledge alarms from this alarm class.
- "System information"
  Alarms in this class have no state and cannot be acknowledged.
- "Warning with reset"
  Alarms in this class usually indicate states of a plant such as "Motor switched on". Alarms in this alarm class must be acknowledged and confirmed.

- "Warning"
  Alarms in this class usually indicate states of a plant such as "Motor switched on". Alarms in this class must be acknowledged.
- "Alarm with reset"
  Alarms in this class must be acknowledged and confirmed. A reset is also required for alarms of this class. The locked alarm is unlocked during the reset.
- "Critical with reset"
  Alarms in this class must be acknowledged and confirmed. A reset is also required for alarms of this class. The locked alarm is unlocked during the reset. The alarm class "Critical with reset" is designed to show critical faults in the plant, for example, "Motor temperature too high".
- "Operator input information"
  Alarms in this class cannot be acknowledged. The alarm class "Operator input information" is designed to show the reports that are relevant for an audit.
- "Operator input request"
  Alarms in this class must be acknowledged.
- "Alarm"
  Alarms in this class must always be acknowledged. The alarm class "Alarm" is designed to show critical or dangerous states or limit violations in the process.
- "Acknowledgement"
  Alarms in this class must be acknowledged. The alarm class "Acknowledgement" is linked to the predefined common alarm class "Acknowledgement".
- "No Acknowledgement"
  Alarms in this class do not require acknowledgment. The alarm class "No Acknowledgement" is linked to the predefined common alarm class "No Acknowledgement".

Note

The alarm classes whose names contain "System" are designed to show the states of the device and the controllers, for example, to provide information on operating errors or faults in communication.

Note

The predefined alarm classes are write-protected and cannot be deleted. You can, however, change the preset background and foreground colors and font colors if necessary. If required, you can also change the name of the predefined alarm classes "Acknowledgement" and "No Acknowledgement". The name of the linked predefined common alarm classes "Acknowledgement" and "No Acknowledgement" is not changed. You cannot change the name of the linked predefined common alarm classes, not even under "Common data > Alarm classes".

Custom alarm classes

The properties of this alarm class are defined in the configuration.

For alarms with priority "0", the priority of the alarm class applies. The priority of the alarm when displayed in Runtime takes precedence over the priority of the alarm class.

See also

Creating alarm classes (Page 231)

Using common alarm classes (Page 233)

Alarm system (Page 213)

Acknowledging alarms (Page 225)

Alarm states (Page 220)

Acknowledgment model (Page 226)

Alarm components and properties (Page 227)

4.1.5 Acknowledging alarms (RT Uni)

Introduction

To make sure that an alarm was noticed by the plant operator, configure this alarm so that it is displayed until acknowledged by the operator. Alarms that indicate critical or hazardous states in the process have to be acknowledged.

The acknowledgment of an alarm is an event that is logged and reported. Acknowledging an alarm in the "Incoming" state changes the alarm state from "Incoming" to "Acknowledged". When the operator acknowledges an alarm, they confirm that they have processed the state that triggered the alarm.

The acknowledgment is not logged for the following state transitions: "Incoming", "Outgoing", "Acknowledged".

Acknowledging an alarm

The operator acknowledges in runtime an alarm via the alarm view buttons.

See also

Alarm system (Page 213)

Configuring alarm acknowledgment (Page 250)

Acknowledgment model (Page 226)

Alarm classes (Page 221)

Alarm states (Page 220)

Alarm components and properties (Page 227)

4.1.6 Acknowledgment model (RT Uni)

Overview

The acknowledgment model and the state machine for predefined alarm classes have already been set. You can only set the acknowledgment model and the state machine for user-defined alarm classes. All alarms included in this alarm class are then acknowledged according to this acknowledgment model and the state machine.

The following state machines are available:

- Alarm with single-mode acknowledgment
  This alarm must be acknowledged as soon as the event that triggers the alarm occurs. The alarm remains pending until it is acknowledged.
- Alarm with optional single-mode acknowledgment
  This alarm must not necessarily be acknowledged as soon as the event that triggers the alarm occurs. The alarm disappears when the event that triggered the alarm is no longer present.
- Alarm with acknowledgment and confirmation
  The alarm must be acknowledged as soon as the event that triggers the alarm has occurred or the alarm is reset. The alarm also requires a confirmation when the event that triggered the alarm is no longer present. The alarm remains pending until it was acknowledged and confirmed.
- Alarm without acknowledgment
  This alarm comes and goes without having to be acknowledged. There is no visible response from the system.
- Alarm without "outgoing" status with acknowledgment
  This alarm is displayed in the alarm view until it is acknowledged. It then disappears from the alarm view.
- Alarm without "outgoing" status without acknowledgment
  This alarm is displayed, and goes out when the event that triggered the alarm is no longer present. The alarm is not added to the alarm view.
- Alarm without status
  This alarm only has the temporary status "Incoming" and can be seen in the log.

Acknowledging and confirming alarms

- Group acknowledgment of alarms in the alarm view
  The alarm view has a "Group acknowledgment" button. This button triggers the acknowledgment of all visible alarms that require acknowledgment and are pending in the alarm view.
- Single acknowledgment of alarms in the alarm view
  The alarm view has a "Single acknowledgment" button. This button triggers the acknowledgment of individual alarms selected in the alarm view.
- Single confirmation of alarms with acknowledgment and confirmation in the alarm view.
  The alarm view has a "Single confirm" button. The alarm with the state machine "Alarm with acknowledgment and confirmation" is individually confirmed with this button after it has been acknowledged with group acknowledgment or single acknowledgment beforehand and is outgoing.

Note

If the "Show recent" button is pressed, the most recent alarm is always shown first. Group acknowledgment is only executed for the visible alarms.

See also

Alarm system (Page 213)

Configuring alarm acknowledgment (Page 250)

Alarm components and properties (Page 227)

Acknowledging alarms (Page 225)

Alarm states (Page 220)

Alarm classes (Page 221)

Alarm control (Page 76)

4.1.7 Alarm components and properties (RT Uni)

Overview

The following table shows the basic components of alarms that you can configure in WinCC:

Alarm class	Alarm number	Time of day	Date	State machine	Alarm text	Info text	Trigger tag	Limit
Warning	1	11:09:14	06.08.2017	Alarm with single-mode acknowledgment	Maximum speed reached	This alarm is ...	speed_1	27

Alarm class

The alarm class of an alarm determines whether the alarm has to be acknowledged.

The alarm class defines the following for an alarm:

- State machine/acknowledgment model
- Appearance in runtime (e.g. color)
- Priority

Alarm number

An alarm is identified by an alarm number (ID). The alarm number is assigned by the system for internally managing an alarm. You can change the alarm number to a sequential alarm number, if necessary, to identify alarms associated in your project.

An alarm number must only be used once on a device.

Note

Discrete alarms and analog alarms can receive an identical alarm number from the system. The alarm number can be customized on request.

Note

When adapting alarm numbers, observe the inter-project uniqueness of the alarm number.

The system event number overrides a custom alarm number. If using the system event number for a custom alarm, change the custom alarm number accordingly.

Time and date

Every alarm has a time stamp that shows the time and date at which the alarm was triggered.

State machine

An alarm has the state machine or the acknowledgment model of the alarm class.

The state machine is how an alarm is displayed in various states and processed from by the system.

Alarm states

An alarm always has a specific alarm state in runtime. The operator analyzes the process execution based on the alarm states.

Alarm text

The alarm text describes the cause of the alarm.

The alarm text can contain output fields for current values. The values you can insert depend on the runtime in use. The value is retained at the time at which the alarm status changes.

Info text

You can configure a separate infotext for each alarm; the operator can display this infotext in runtime.

Trigger tag

A tag is assigned to each alarm as trigger. The alarm is raised when this trigger tag meets the defined condition, e.g. when its state changes or it exceeds a limit.

Limit

Analog alarms indicate limit violations. Depending on the configuration, WinCC outputs the analog alarm as soon as the trigger tag exceeds or undershoots the limit value.

Computer

Operator input alarms have the "Computer" column in the alarm lists. The computer name is displayed for local alarms and the IP address for alarms from the web client.

Users

The user acknowledges the alarm. If an empty user name is transferred to an alarm, the alarm displays no user name.

See also

Alarm system (Page 213)

Acknowledgment model (Page 226)

Alarm states (Page 220)

Alarm classes (Page 221)

Acknowledging alarms (Page 225)

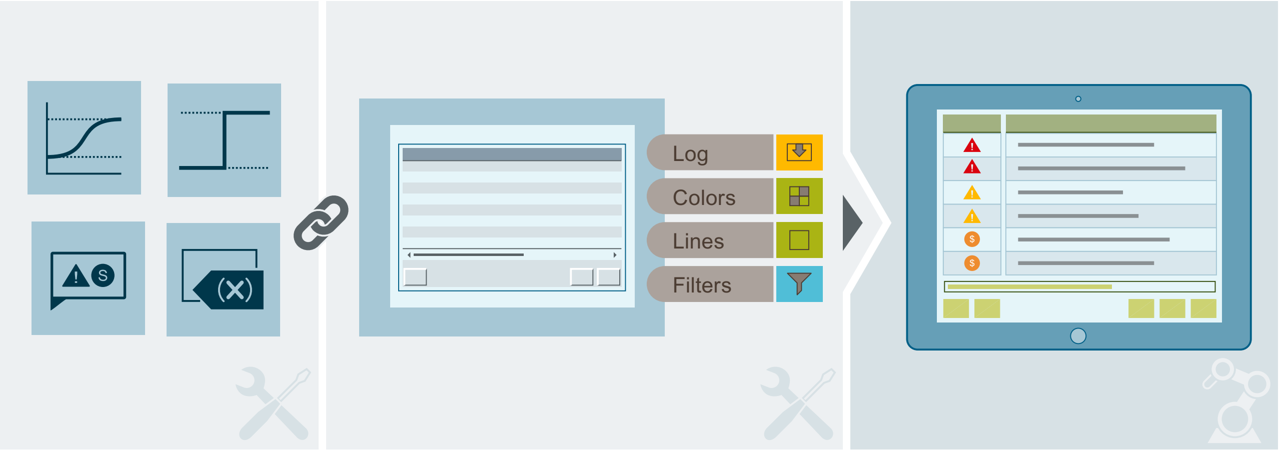
4.2 Configuring alarms (RT Uni)

4.2.1 Workflow for configuring alarms (RT Uni)

Steps to configure alarms

You configure alarms in the following stages:

1. Set alarm classes or configure your own alarm classes and assign them to alarms
   You use the alarm class to define how an alarm is to be displayed in runtime and to define the state machines for it.
2. Create trigger tags in the "HMI tags" editor
   - Configure the tags for your project (bit string for discrete alarms and trigger tags for analog alarms)
   - You create range values for the tags.
3. Creating tags in the "HMI alarms " editor
   - Create custom alarms and assign the tag to be monitored, alarm classes, and other properties to them.
4. Output of configured alarms
   To output configured alarms, configure an alarm view in the "Screens" editor.
5. Creating an alarm log
   To log alarms, create an alarm log in the "Logs" editor.



Additional configuration tasks

Additional tasks could be necessary for configuring alarms, depending on the requirements of your project:

- Editing system events
  If necessary, you edit system events under "Languages & Resources > Project texts". In the "Category" column you can recognize a system event by the name "HMI system event".
- Activating controller alarms
  For integrated operation of a project in STEP 7, specify the controller alarms to be displayed on your HMI device in the alarm settings.

Note

WinCC only supports controller alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports controller alarms that are automatically updated by the central alarm management in the controller.

See also

Creating alarm classes (Page 231)

Using common alarm classes (Page 233)

Configuring discrete alarms (Page 238)

Configuring analog alarms (Page 241)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

Configuring multilingual alarm texts (Page 247)

Editing system events (Page 248)

Configuring alarm acknowledgment (Page 250)

Filtering controller alarms via display classes (Page 249)

Sending and automatically updating complete alarm from the controller to the HMI device (Page 287)

Configuring automatic update of controller alarms on the HMI device (Page 288)

Configuring the display of system diagnostic alarms (Page 262)

4.2.2 Creating alarm classes (RT Uni)

Introduction

Create alarm classes to define the type of acknowledgment, the display of the alarm in runtime for an alarm. You assign the individual alarms to the alarm classes.

Create alarm classes in the "Alarm classes" tab of the "HMI alarms" editor. Some default alarm classes are already created for every project. You can create additional custom alarm classes.

System alarm classes are write-protected and cannot be deleted. You can, however, change the preset background and foreground colors and font colors if necessary. All system alarm classes have the word "System" in their name and are to be used for system-defined alarms.

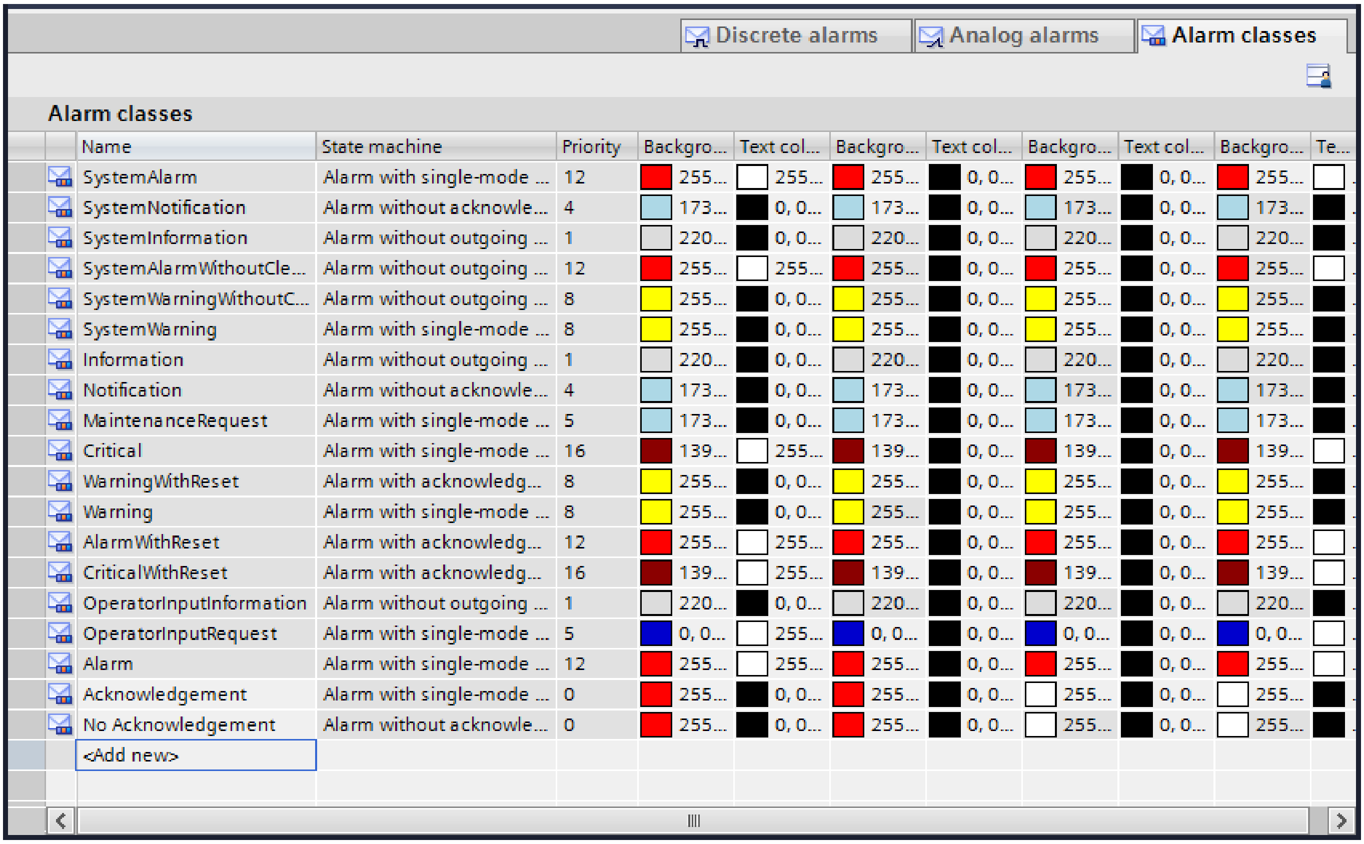
Requirement

- The "HMI alarms" editor is open.
- The Inspector window is open.

Procedure

To create an alarm class, proceed as follows:

1. Click the "Alarm classes" tab.
   A table of the pre-defined alarm classes is shown below:



Alarm classes

Name	State machine	Priority	Backgro...	Text col...	Backgro...	Text col...	Backgro...	Text col...	Backgro...	Te...
SystemAlarm	Alarm with single-mode ...	12	🟥 255...	⬜ 255...	🟥 255...	⬛ 0, 0...	🟥 255...	⬛ 0, 0...	🟥 255...	⬜
SystemNotification	Alarm without acknowle...	4	🟦 173...	⬛ 0, 0...	🟦 173...	⬛ 0, 0...	🟦 173...	⬛ 0, 0...	🟦 173...	⬛
SystemInformation	Alarm without outgoing ...	1	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛
SystemAlarmWithoutCle...	Alarm without outgoing ...	12	🟥 255...	⬜ 255...	🟥 255...	⬛ 0, 0...	🟥 255...	⬛ 0, 0...	🟥 255...	⬜
SystemWarningWithoutC...	Alarm without outgoing ...	8	🟨 255...	⬛ 0, 0...	🟨 255...	⬛ 0, 0...	🟨 255...	⬛ 0, 0...	🟨 255...	⬛
SystemWarning	Alarm with single-mode ...	8	🟨 255...	⬛ 0, 0...	🟨 255...	⬛ 0, 0...	🟨 255...	⬛ 0, 0...	🟨 255...	⬛
Information	Alarm without outgoing ...	1	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛
Notification	Alarm without acknowle...	4	🟦 173...	⬛ 0, 0...	🟦 173...	⬛ 0, 0...	🟦 173...	⬛ 0, 0...	🟦 173...	⬛
MaintenanceRequest	Alarm with single-mode ...	5	🟦 173...	⬛ 0, 0...	🟦 173...	⬛ 0, 0...	🟦 173...	⬛ 0, 0...	🟦 173...	⬛
Critical	Alarm with single-mode ...	16	🟥 139...	⬜ 255...	🟥 139...	⬛ 0, 0...	🟥 139...	⬛ 0, 0...	🟥 139...	⬜
WarningWithReset	Alarm with acknowledg...	8	🟨 255...	⬛ 0, 0...	🟨 255...	⬛ 0, 0...	🟨 255...	⬛ 0, 0...	🟨 255...	⬛
Warning	Alarm with single-mode ...	8	🟨 255...	⬛ 0, 0...	🟨 255...	⬛ 0, 0...	🟨 255...	⬛ 0, 0...	🟨 255...	⬛
AlarmWithReset	Alarm with acknowledg...	12	🟥 255...	⬜ 255...	🟥 255...	⬛ 0, 0...	🟥 255...	⬛ 0, 0...	🟥 255...	⬜
CriticalWithReset	Alarm with acknowledg...	16	🟥 139...	⬜ 255...	🟥 139...	⬛ 0, 0...	🟥 139...	⬛ 0, 0...	🟥 139...	⬜
OperatorInputInformation	Alarm without outgoing ...	1	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛ 0, 0...	⬜ 220...	⬛
OperatorInputRequest	Alarm with single-mode ...	5	🟦 0, 0...	⬜ 255...	🟦 0, 0...	⬛ 0, 0...	🟦 0, 0...	⬛ 0, 0...	🟦 0, 0...	⬜
Alarm	Alarm with single-mode ...	12	🟥 255...	⬜ 255...	🟥 255...	⬛ 0, 0...	🟥 255...	⬛ 0, 0...	🟥 255...	⬜
Acknowledgement	Alarm with single-mode ...	0	🟥 255...	⬛ 0, 0...	🟥 255...	⬛ 0, 0...	⬜ 255...	⬛ 0, 0...	⬜ 255...	⬛
No Acknowledgement	Alarm without acknowle...	0	🟥 255...	⬛ 0, 0...	🟥 255...	⬛ 0, 0...	⬜ 255...	⬛ 0, 0...	⬜ 255...	⬛
<Add new>										

2. Double-click "<Add>" in the table.
   A new alarm class is created. Each new alarm class is automatically assigned a static ID. The properties of the new alarm class are shown in the Inspector window.
3. Configure the alarm class under "Properties > General" in the Inspector window:
   – Enter a name for the alarm class.
   – Set the priority of the alarm class.

4. Define the state machine for the alarm class under "Properties > Acknowledgment" in the Inspector window.
5. You can also change the default background color as well as the text color and the settings for flashing under "Properties > Colors" in the Inspector window.
   These settings define how alarms from this alarm class are displayed in runtime.

   Note

   For alarm colors to be displayed in Runtime, the "Use alarm colors" option must be activated in the properties of the alarm view in the Inspector window. This option is enabled by default.

See also

Alarm classes (Page 221)

Using common alarm classes (Page 233)

Workflow for configuring alarms (Page 230)

Configuring alarm acknowledgment (Page 250)

Configuring discrete alarms (Page 238)

Configuring analog alarms (Page 241)

4.2.3 Using common alarm classes (RT Uni)

Introduction

Common alarm classes are displayed under "Common data > Alarm classes" in the project tree and can be used for the alarms of an HMI device. Common alarm classes are used in STEP 7 for controller alarms. If required, create additional common alarm classes in WinCC. Common alarm classes are divided into predefined and user-defined common alarm classes. The predefined common alarm classes are "Acknowledgement" (for alarms with acknowledgment) and "No Acknowledgement" (for alarms without acknowledgment).

When you create an HMI device, the system creates an alarm class for each existing common alarm class under "HMI alarms > Alarm classes" which is linked to the common alarm class. If you have created an HMI device and create a common alarm class, the system creates an alarm class for the created common alarm class under "HMI alarms > Alarm classes" which is linked to the common alarm class. If you change all properties for a created common alarm class, the system changes the properties "State machine" and "Priority" of the linked alarm class according to your changes to the "Acknowledgment" and "Priority" properties of the common alarm class. However, your changes to the "Name" and "Display name" properties of the common alarm class have no effect on the properties of the linked alarm class. When you delete a common alarm class, the linked alarm class is also deleted.

Note

By the "Common alarm class" property of an alarm class, you can see whether the alarm class is linked with a common alarm class and, if so, with which common alarm class. To see the property in the "HMI alarms" editor in the "Alarm classes" tab, activate there the "Common alarm class" column using the shortcut menu of the column headers. To see the property in the Inspector window, select an alarm class under "HMI alarms > Alarm classes" and click "General" in the Inspector window.

Requirements

- You have created a project.

Creating common alarm class

To create a common alarm class, proceed as follows:

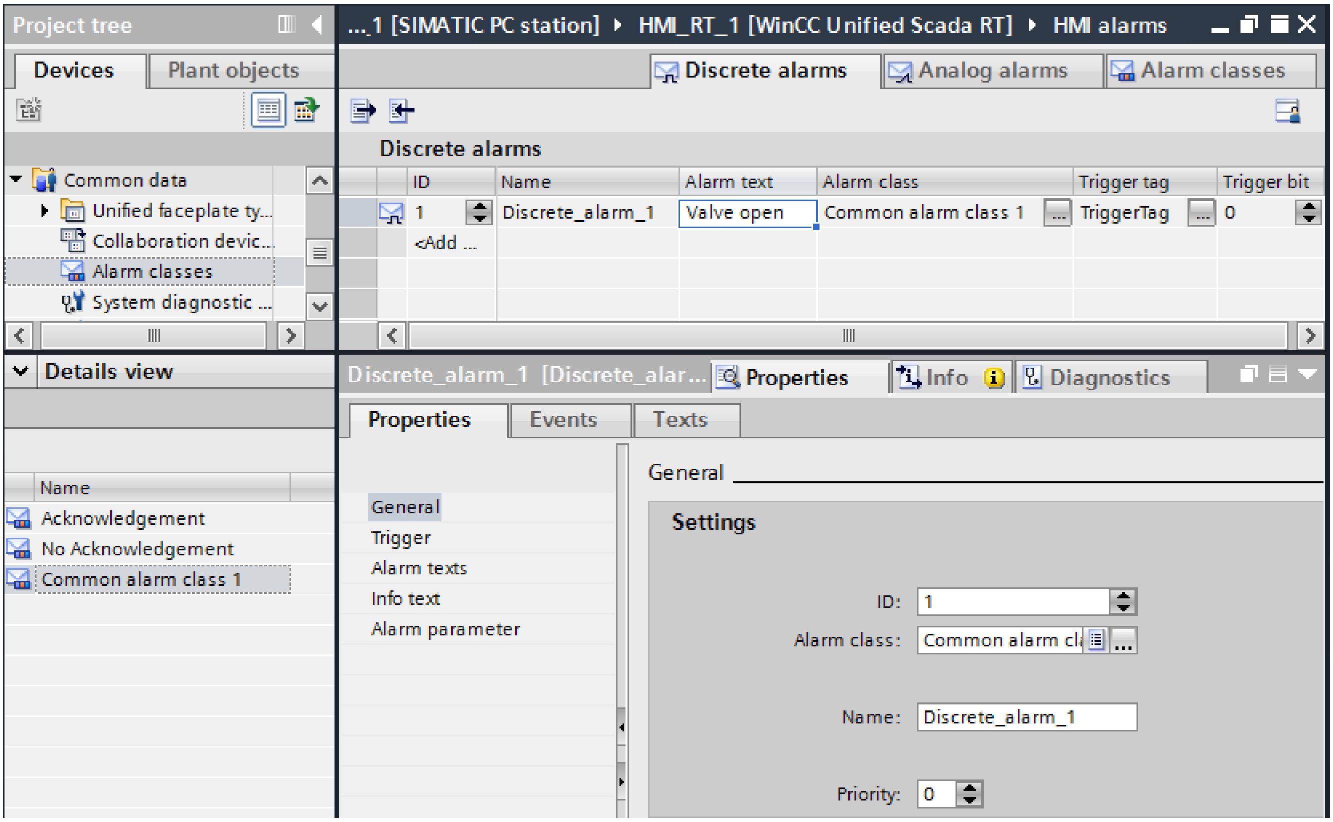
1. Double-click "Common data > Alarm classes" in the project tree.
   The "Alarm classes" editor opens in the working area.
2. To create a common alarm class, double-click in the first empty line of the table editor.
3. Specify the name, display name and priority of the common alarm class and activate the mandatory acknowledgment, if required.
   A common alarm class is created. The system also creates an alarm class under "HMI alarms > Alarm classes", which is linked to the common alarm class. The linked alarm class gets from the system the same name and priority as the common alarm class. If you have enabled the mandatory acknowledgment for the common alarm class, the linked alarm class gets the state machine "Alarm with single-mode acknowledgment" from the system, otherwise the state machine "Alarm without acknowledgment". The defined display name of the common alarm class has no effect on the properties of the linked alarm class.
4. If required, change the name of the alarm class that is linked to the common alarm class under "HMI alarms > Alarm classes".
   If you change the name of the linked alarm class, the common alarm class name is not changed by the system.

Assign alarms to a common alarm class

Proceed as follows to assign an analog or discrete alarm to a common alarm class:

1. In the "HMI alarms" editor, select the alarm that you want to assign to the common alarm class.
2. Click "General" in the Inspector window.
3. Click "Common data > Alarm classes" in the project tree. Alternatively, click "HMI Alarms" in the project tree.
   In the first case, the common alarm class is selected in the detail view. In the second case, the detail view shows the alarm class, which is linked to the common alarm class.

4. Select the common alarm class or alternatively the linked alarm class in the detail view.
5. Drag the common alarm class or alternatively the linked alarm class to the "Alarm class" field or "Alarm class" column in the working area of the Inspector window of the alarm.
   In both cases, the alarm is assigned to the alarm class which is linked to the common alarm class.



Discrete alarms

ID	Name	Alarm text	Alarm class	Trigger tag	Trigger bit
1	Discrete_alarm_1	Valve open	Common alarm class 1	TriggerTag	0
<Add ...					

ID: 1
Alarm class: Common alarm cl
Name: Discrete_alarm_1
Priority: 0

Changing a common alarm class

To change a common alarm class, proceed as follows:

1. Double-click "Common data > Alarm classes" in the project tree.
   The "Alarm classes" editor opens in the working area.
2. If necessary, change the name of the created common alarm class.
   The changed name of the common alarm class has no effect on the name of the alarm class which is linked to the common alarm class.
3. If necessary, change the display name of the common alarm class.
   The changed display name of the common alarm class has no effect on the properties of the linked alarm class.

4. If required, activate or deactivate the mandatory acknowledgment of the common alarm class.
   If you enable the mandatory acknowledgment, the system changes the state machine of the linked alarm class to "Alarm with single-mode acknowledgment". If you disable the mandatory acknowledgment, the system changes the state machine of the linked alarm class to "Alarm without acknowledgment".
5. If necessary, change the priority of the common alarm class.
   The system changes the priority of the linked alarm class according to your change to the priority of the common alarm class.

Note

You can only change the display names for a predefined common alarm class. The changed display name of a predefined common alarm class has no effect on the properties of the alarm class which is linked to the predefined common alarm class.

Deleting a common alarm class

To delete a common alarm class, proceed as follows:

1. Double-click "Common data > Alarm classes" in the project tree.
   The "Alarm classes" editor opens in the working area.
2. Select the created common alarm class that you want to delete.
3. Select the "Delete" entry from the shortcut menu.
   The system deletes the common alarm class and the alarm class linked with the common alarm class.
4. If an analog or discrete alarm has been assigned to the deleted linked alarm class, assign another alarm class to the alarm. Otherwise a compile error will be generated.

Note

You cannot delete predefined common alarm classes and the alarm classes that are linked with them.

See also

Alarm classes (Page 221)

Creating alarm classes (Page 231)

Workflow for configuring alarms (Page 230)

Configuring discrete alarms (Page 238)

Configuring analog alarms (Page 241)

4.2.4 Configuring state texts of alarms (RT Uni)

Introduction

The texts for the states of alarms in runtime are displayed in the alarm view in the "Status Text" column. You specify the state texts of alarms in the runtime settings.

Note

If no alarm state texts are defined, an error is generated during compiling.

Requirement

- The alarm view has been configured.

Procedure

To configure the state texts of alarms, follow these steps:

1. Open the "Runtime settings" of the HMI device.
2. Specify the state texts of alarms in runtime under "Alarms > State texts":

Field	Description
Normal	Text for alarms in "Normal" state. This state can only have alarms that are for information purposes only.
Incoming	Text for incoming alarms when changing to the operating state to be reported
Incoming/outgoing	Text for incoming and outgoing alarm
Incoming/acknowledged	Text for incoming and acknowledged alarm
Incoming/acknowledged/outgoing	Text for incoming, acknowledged and outgoing alarm
Incoming/outgoing/acknowledged	Text for incoming, outgoing and acknowledged alarm
Remote	Text for alarms in "Remote" state. Only controller alarms can have this state. The state text is only displayed in the alarm log. If the HMI connection between HMI device and controller is disconnected and then re-established, the status text is displayed.

3. To define the state texts in several languages, activate at least one additional language under "Languages & Resources > Project languages".

4. Specify the state texts in the other languages under "Languages & Resources > Project texts".

Note

The state texts have the entry "Alarm text" in the "Category" column. For the state texts, the texts displayed in the "Reference" column also end as follows:

- "…\NormalText"
- "…\ComingText"
- "…\ComingGoingText"
- "…\ComingAcknowledgedText"
- "…\ComingAcknowledgedGoingText"
- "…\ComingGoingAcknowledgedText"
- "…\RemovedText"

5. Select the alarm view in the "Screens" editor.
6. To display the column "Status text" in the Alarm view, select the property "Visibility" in the Inspector window under "Properties > Alarm view > Columns > [46] Status text alarm column".

See also

Configuring an alarm control (Page 253)

4.2.5 Configuring discrete alarms (RT Uni)

Introduction

Discrete alarms triggered by the PLC indicate status changes in a plant. A discrete alarm is triggered by a specific value (bit) of a tag.

Imagine, for example, that the state of a valve is to be monitored during operation. The two possible valve states are "opened" and "closed". In this case, a discrete alarm is configured for each valve state. A discrete alarm containing the following alarm text, for example, is output when the state of this valve changes: "Valve closed".

Note

By default, each new discrete alarm is assigned the alarm class "Alarm". You can then alter the alarm class as required.

Requirement

- The "HMI alarms" editor is open.
- The Inspector window is open.

Procedure

To configure a discrete alarm, proceed as follows:

1. Click the "Discrete alarms" tab.
2. To create a new discrete alarm, double-click on "<Add>" in the table.
   A new discrete alarm is created.
3. Assign a name for the discrete alarm.

   Note

   The name of a discrete alarm can contain up to 128 characters.

4. To configure the alarm, select "Properties > General" in the Inspector window:
   - Edit the name of the alarm as required.
   - Select the alarm class.
   - Configure the priority of the alarm (a value of between "0" and "16").

   Note

   You can use the priority to sort or filter the alarms in the alarm view. With sorting by priority, you can ensure that the most important alarm (high priority) is shown in the display range in a single-line alarm view.

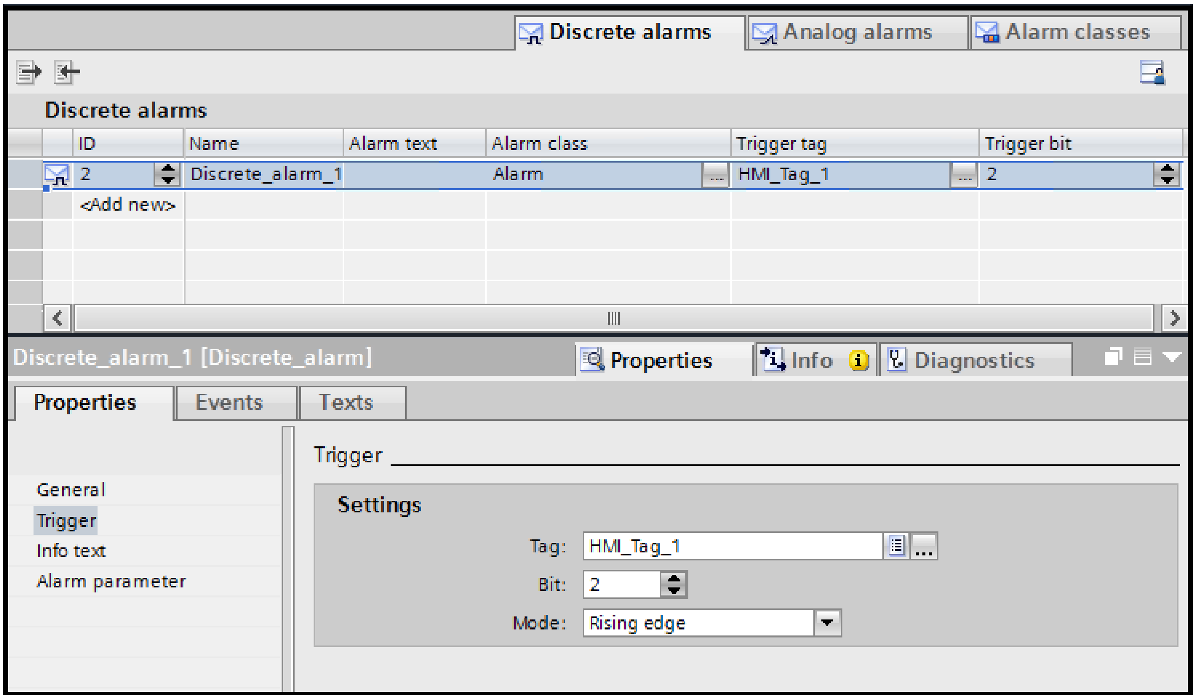
   If you filter the alarm view by priority "16", only the alarms with priority "16" will appear.

   For alarms with priority "0", the priority of the alarm class applies.

   The priority of the alarm when displayed in Runtime takes precedence over the priority of the alarm class.

5. Optional: Configure an additional info text in the Inspector window under "Properties > Info text", which operators can display in Runtime via "Info text configuration".
   To insert a line break in the info text, press Shift+ENTER.

6. Select "Properties > Trigger" in the Inspector window to select the tag and the bit that triggers the alarm.
   Note the following information:
   - A tag is monitored using only one alarm type. You should therefore create either analog alarms **or** discrete alarms for a tag.
   - Only use a trigger tag bit for one alarm.
   - Use one of the following data types: "Bool", "Byte", "Word", "LWord" or "DWord".
   - Do not use trigger tags for anything else.
   - The available area for the bit of the trigger tag depends on the trigger tag data type.
   - If the object does not yet exist in the selection list, create it directly in the object list and change its properties later.



Discrete alarms

ID	Name	Alarm text	Alarm class	Trigger tag	Trigger bit
2	Discrete_alarm_1		Alarm	HMI_Tag_1	2
<Add new>					

Trigger

Settings

Tag: HMI_Tag_1

Bit: 2

Mode: Rising edge

Note

The Engineering System ensure that you use the trigger bit only once.

---

7. In the Inspector window under "Properties > Trigger > Settings > Mode", specify whether the alarm is triggered at a rising or falling edge.
8. Specify alarm texts under "Properties > Alarm texts" in the Inspector window:
   - Specify an alarm text under "Alarm text".
   - Specify additional alarm texts in the fields for additional texts.
   - If necessary, insert parameter output fields in the alarm texts using the "Insert parameter field" shortcut menu command.

Tips for effective procedure

- Large machines and plants have a large number of alarm sources that can trigger various different types of alarms. It makes sense to structure the alarm system so that the user can keep track of this wide range. One suitable method available here is alarm prioritization. The criteria for assigning the priority value and/or the alarm class are importance and urgency. The priority of the alarm can also be based on the potential impact (system downtime, loss of production, production delay, etc.). If multiple alarms are output, the system can suggest the order in which they should be handled on the basis of priorities.
- You create discrete alarms together with the trigger tags and edit them in the "HMI tags" editor. You create tags in the usual way. Then click <Add> in the table on the "Discrete alarms" tab at the bottom of the work area. A new discrete alarm is created for the tag. If you have selected the wrong data type, the tag will be highlighted in the discrete alarm. If you delete, move or copy objects in the "HMI tags" editor, these changes also take effect in the "HMI alarms" editor. The configured discrete alarms are created in the "HMI tags" editor and displayed in the "HMI alarms" and "HMI tags" editors.
- Supplementary information about individual alarms ensures that faults are localized and cleared quickly.

See also

Discrete alarms (Page 216)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

Workflow for configuring alarms (Page 230)

Configuring alarm acknowledgment (Page 250)

Creating alarm classes (Page 231)

Using common alarm classes (Page 233)

Configuring multilingual alarm texts (Page 247)

Editing system events (Page 248)

Configuring analog alarms (Page 241)

Creating internal tags (Page 154)

4.2.6 Configuring analog alarms (RT Uni)

Introduction

You configure analog alarms to display limit violations. You have defined in advance a limit value for the trigger tag and the trigger mode. An analog alarm is triggered depending on which mode you have defined, for example, when the value is higher than, lower than or the same as the defined value.

If the speed of a motor drops below a certain value, for example, an analog alarm is triggered. This alarm could contain the following text: "Motor speed is too low".

Note

By default, each new analog alarm is assigned the alarm class "Alarm". You can then alter the alarm class as required.

Requirement

- The "HMI alarms" editor is open.
- The Inspector window is open.

Procedure

To configure an analog alarm, proceed as follows:

1. Click the "Analog Alarms" tab.
2. To create a new analog alarm, double-click in the table on "<Add>".
   A new analog alarm is created.
3. To configure the alarm, select "Properties > General" in the Inspector window:
   – Edit the name of the alarm as required.
   – Select the alarm class.
   – Configure the priority of the alarm (a value of between "0" and "16").

   ### Note

   You can use the priority to sort or filter the alarms in the alarm view. With sorting by priority, you can ensure that the most important alarm (high priority) is shown in the display range in a single-line alarm view.

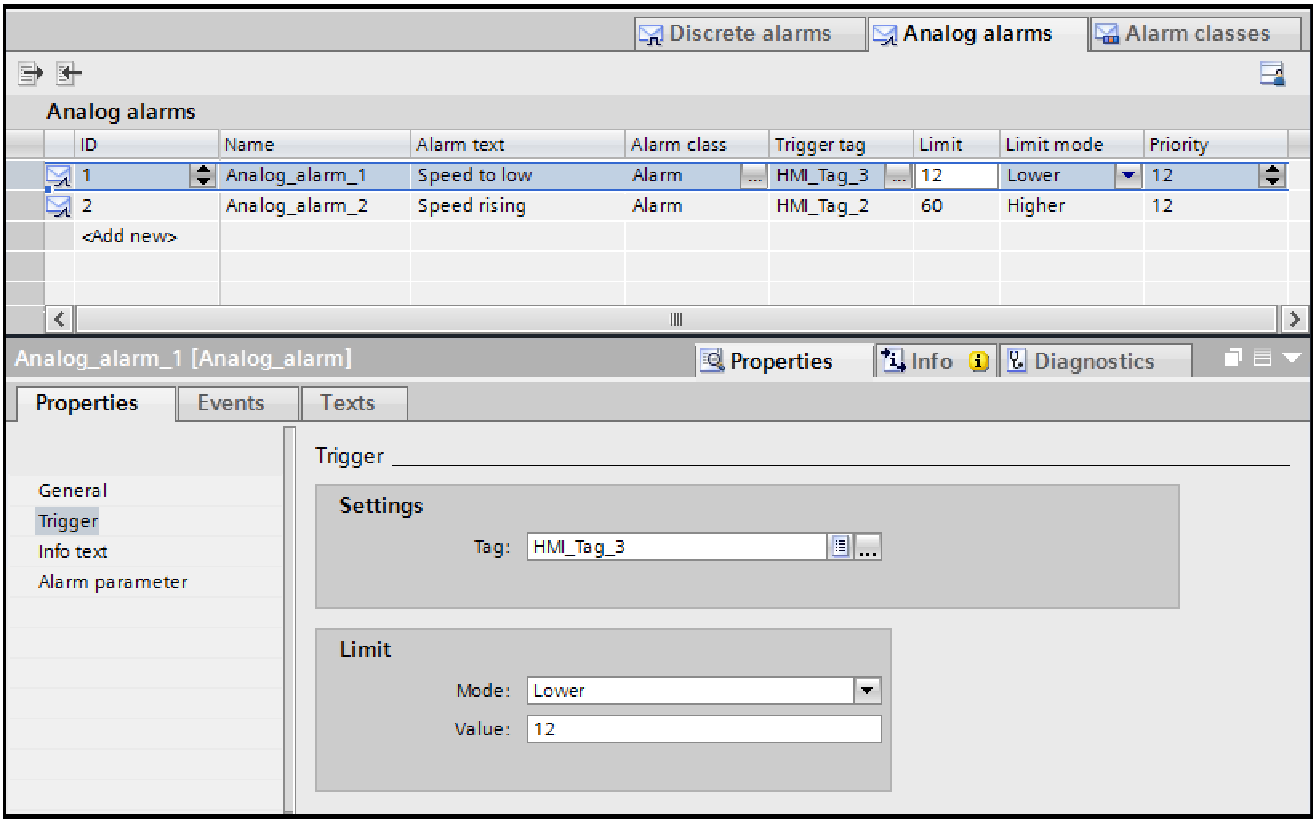
   If you filter the alarm view by priority "16", only the alarms with priority "16" will appear.

   For alarms with priority "0", the priority of the alarm class applies.

   The priority of the alarm when displayed in Runtime takes precedence over the priority of the alarm class.

4. Optional: Configure an additional info text in the Inspector window under "Properties > Info text", which operators can display in Runtime via "Info text configuration".
   To insert a line break in the info text, press Shift+ENTER.
5. In the Inspector window, select the tag that triggers the alarm under "Properties > Trigger".
   Do not use trigger tags for anything else.
   Use one of the following data types: "Int", "Real", "LReal", "SInt", "USInt", "UInt", "UDInt" and "ULInt".
6. Specify alarm texts under "Properties > Alarm texts" in the Inspector window:
   – Specify an alarm text under "Alarm text".
   – Specify additional alarm texts in the fields for additional texts.
   – If necessary, insert parameter output fields in the alarm texts using the "Insert parameter field" shortcut menu command.

7. Select the analog alarm to which you want to assign the limits.



Analog alarms

ID	Name	Alarm text	Alarm class	Trigger tag	Limit	Limit mode	Priority
1	Analog_alarm_1	Speed to low	Alarm	HMI_Tag_3	12	Lower	12
2	Analog_alarm_2	Speed rising	Alarm	HMI_Tag_2	60	Higher	12
<Add new>							

Analog_alarm_1 [Analog_alarm]

Trigger

Settings

Tag: HMI_Tag_3

Limit

Mode: Lower

Value: 12

8. Enter a limit value in the "Value" field under "Properties > Trigger" in the Inspector window.

9. Select the trigger mode in the "Mode" field:
   - "Less": The alarm is triggered if the limit is undershot.
   - "Greater": The alarm is triggered if the limit is exceeded.
   - "Equal": The alarm is triggered when the limit is reached.
   - "Not equal": The alarm is triggered if the limit is not reached.
   - "Less or equal": The alarm is triggered if the limit is undershot or reached.
   - "Greater or equal": The alarm is triggered if the limit is exceeded or reached.

10. You can create additional limits for the alarm, if necessary. Note the following information:
    - A tag is monitored using only one alarm type. You should therefore create either analog alarms **or** discrete alarms for a tag.
    - If the object included in the selection does not yet exist, create it in the object list and change its properties later.

Tips for effective procedure

- Large machines and plants have a large number of alarm sources that can trigger various different types of alarms. It makes sense to structure the alarm system so that the user can keep track of this wide range. One suitable method available here is alarm prioritization. The criteria for assigning the priority value and/or the alarm class are importance and urgency. The priority of the alarm can also be based on the potential impact (system downtime, loss of production, production delay, etc.). If multiple alarms are output, the system can suggest the order in which they should be handled on the basis of priorities.
- You create analog alarms together with the trigger tags and edit them in the "HMI tags" editor. You create tags in the usual way and configure the range values of the tags. Then click <Add> in the table on the "Analog alarms" tab at the bottom of the work area. A new analog alarm is created for the tag. If you have selected the wrong data type, the tag will be highlighted in the analog alarm. If you delete, move or copy objects in the "HMI tags" editor, these changes also take effect in the "HMI alarms" editor. The configured analog alarms are created in the "HMI tags" editor and displayed in the "HMI alarms" and "HMI tags" editors.
- Supplementary information about individual alarms ensures that faults are localized and cleared quickly.

See also

Analog Alarms (Page 215)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

Workflow for configuring alarms (Page 230)

Configuring alarm acknowledgment (Page 250)

Creating alarm classes (Page 231)

Using common alarm classes (Page 233)

Configuring discrete alarms (Page 238)

Configuring multilingual alarm texts (Page 247)

Editing system events (Page 248)

Creating internal tags (Page 154)

4.2.7 Configuring optional parameters for discrete alarms and analog alarms (RT Uni)

Setting the alarm context

In a large plant system, it makes sense to save information about the alarm origin such as the physical, geographical or logical grouping of plant units that is defined by the site. This helps users in identifying the causes of the alarm and the source of the fault.

You configure the information on alarm sources in the "Origin" field of the "Alarm context" area.

The "Area" field is a static field and contains information about the device.

Creating info texts for alarms

To configure an infotext for the alarm and thus support users, follow these steps:

1. Select a discrete alarm or an analog alarm.
2. Select "Properties > Infotext" in the Inspector window and enter the required text.

3. To insert a line break in the info text, press "Shift+Enter" at the corresponding text location.
4. To create multi-lingual info texts, enter the respective texts in the predefined project languages in the Inspector window and, if necessary, in the reference language.

Enabling parameters for a discrete or analog alarm

To output process values in an output field in the alarm text, assign tags to the parameter blocks. Proceed as follows:

1. Select the alarm.
2. In the Inspector window, click "Properties > Alarm parameters".
3. Select a tag for the alarm parameter.
4. You can enter multiple alarm parameters if required.
   Insert the activated process values as a selection box in an alarm text.

Note

You can configure up to 10 tags as alarm parameters for discrete alarms and analog alarms.

All available data types are supported.

See also

Configuring analog alarms (Page 241)

Configuring discrete alarms (Page 238)

Workflow for configuring alarms (Page 230)

Configuring multilingual alarm texts (Page 247)

Editing system events (Page 248)

Configuring alarm acknowledgment (Page 250)

Translating texts directly (Page 763)

Exporting project texts (Page 765)

Analog Alarms (Page 215)

Discrete alarms (Page 216)

4.2.8 Parameter output in a discrete or analog alarm (RT Uni)

Introduction

To display alarm parameters, insert an appropriate output field in a discrete or analog alarm. You can select the parameters configured in "Properties > Properties > Alarm parameters" for use as alarm parameters.

Requirement

- The "HMI alarms" editor is open.
- The discrete alarm or analog alarm is selected.

Procedure

To output a parameter in the alarm text, follow these steps:

1. Place the cursor at the required position in the alarm text.
2. To output an alarm parameter, select "Insert parameter field" from the shortcut menu.
   A dialog box opens.
3. Select the desired parameter.
4. Moreover, you can specify the following data for alarm parameters:
   - The tag that provides the parameter values.
     The tag configured for the parameter under "Properties > Properties > Alarm parameters" is entered by default. If you select a different tag, WinCC updates the parameter configuration in "Properties > Properties > Alarm parameters" accordingly.
   - Display type, text list, length, number of decimal places and alignment of the output field
   - To display leading zeros in the output field, enable "Leading zeros".
5. Confirm the dialog to save your entries.

4.2.9 Configuring alarm texts (RT Uni)

Introduction

For an alarm, you can configure up to ten alarm texts: one alarm text and up to nine additional texts. If required, you can insert output fields for displaying alarm parameters in each alarm text. Each alarm text contains up to 512 characters.

Requirement

- An alarm has been created.
- The alarm control has been configured.

Procedure

To configure alarm texts, follow these steps:

1. In the "HMI alarms" editor, select the alarm.
2. Enter an alarm text under "Properties > Properties > Alarm texts > Settings > Alarm text" in the Inspector window.

   Note

   Use the scroll buttons ⇕ to view text that is not fully visible in the text box. The additional text lines are displayed.

3. If necessary, insert parameter output fields in the alarm text via the "Insert parameter field" shortcut menu command.
4. Enter additional alarm texts in the fields for additional texts under "Properties > Properties > Alarm texts" in the Inspector window.
5. If necessary, insert parameter output fields in the other alarm texts using the "Insert parameter field" shortcut menu command.
6. Select the alarm view in the "Screens" editor.
7. To display the alarm texts in Runtime, enable the required columns of the columns numbered 10 to 19 in the Inspector window under "Properties > Properties > Alarm view > Columns".

4.2.10 Configuring multilingual alarm texts (RT Uni)

Requirements

- The "HMI alarms" editor is open.
- An alarm has been created.

Procedure

1. Select one or more alarms for which you want to configure multilingual alarm texts.
2. You can view the alarm texts already configured in the set project languages under "Properties > Texts".
3. If available, enter the alarm texts in the required project languages.
   The alarm texts will then be displayed in the set runtime language in runtime.

Note

All alarm texts are managed together with other project texts under "Languages & Resources > Project texts".

If you cannot configure the project texts in multiple languages yourself, export them to an Excel file and have them translated. You can then import the texts to your project.

See also

Basics of project texts (Page 762)

Workflow for configuring alarms (Page 230)

Configuring discrete alarms (Page 238)

Configuring analog alarms (Page 241)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

Working with multiple languages (Page 761)

Importing project texts (Page 767)

Translating texts directly (Page 763)

Selecting the reference language and editing language (Page 759)

4.2.11 Editing system events (RT Uni)

Basics

A system event indicates the system status and communication errors between the HMI device and system. System events are output in runtime in the configured alarm view. System events are output in the language currently set on your HMI device.

The time format (AM/PM or 24-hour format) is based on the selected language. If no translation of the alarm texts exists in this language, English is used as replacement and the corresponding time format is displayed.

Example of an alarm:

"Memory is full!"

Editing system events

If necessary, you edit system events under "Languages & Resources > Project texts". In the "Category" column you can recognize a system event by the name "HMI system event". You can export the system events together with the other texts under "Project texts" and have them translated.

System event parameters

System events may contain encrypted parameters. The parameters are of relevance when troubleshooting because they provide a reference to the source code of the runtime software. These parameters are output after the "Error code: text"

See also

Workflow for configuring alarms (Page 230)

Configuring discrete alarms (Page 238)

Configuring analog alarms (Page 241)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

System Alarms (Page 218)

4.2.12 Filtering controller alarms via display classes (RT Uni)

Introduction

Controller alarms are configured in STEP 7. Controller alarms are available in WinCC running in a STEP 7 environment.

If a PLC is connected to multiple HMI devices, the project engineer assigns display classes to the controller alarms in STEP7. The display classes determine the allocation to the HMI device. You can activate the display classes for your HMI device that are to be displayed on it. In this case, only the controller alarms from this display class will be displayed on the HMI device. Up to 17 display classes are possible.

Note

WinCC only supports controller alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports controller alarms that are automatically updated by the central alarm management in the controller.

Requirement

- The connection was established to the PLC.
- Alarms were configured in STEP 7.

Filtering controller alarms via display classes

To filter controller alarms by display classes, proceed as follows:

1. Click "Runtime settings > Alarms" in the project tree under your HMI device.
   One or several connections to a PLC are shown in "Contoller alarms".
2. Select the display classes whose controller alarms you want to display for the connection.

See also

Sending and automatically updating complete alarm from the controller to the HMI device (Page 287)

Configuring automatic update of controller alarms on the HMI device (Page 288)

Workflow for configuring alarms (Page 230)

User-defined controller alarms (Page 216)

Alarm system (Page 213)

4.2.13 Configuring alarm acknowledgment (RT Uni)

Introduction

The alarm classes define how the alarms from an alarm class are to be acknowledged. When you assign an alarm to an alarm class, you define the state machine and the acknowledgment model for that alarm.

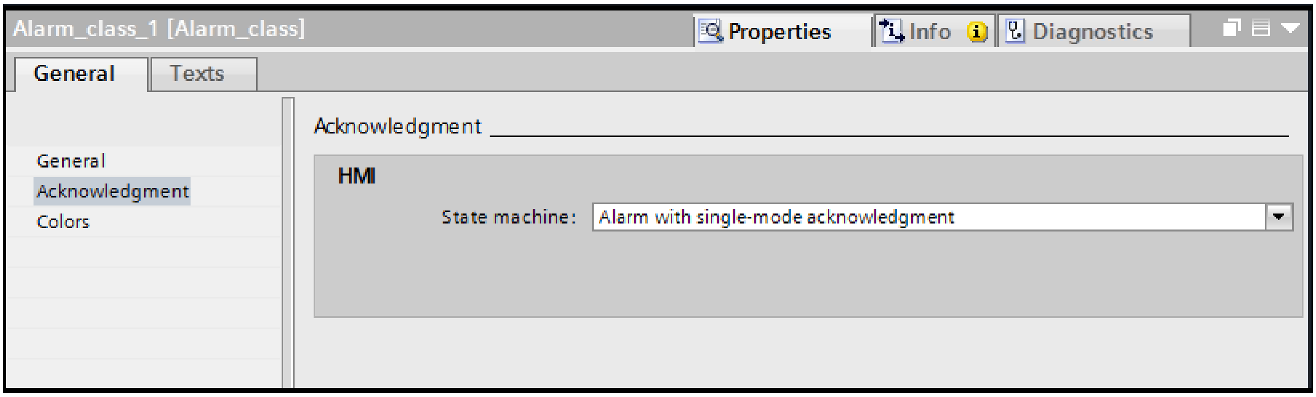
Requirements

- The "HMI alarms" editor is open.
- The required alarm class has been created.
- The required alarm has been created.

Procedure

To configure the acknowledgement of an alarm, follow these steps:

1. In the "HMI alarms" editor, click the "Alarm class" tab and select the alarm class.
2. You select the desired state machine under "Properties > General > Acknowledgment" in the Inspector window.



State machine: Alarm with single-mode acknowledgment

Note

The buttons relevant for acknowledgment, "Group acknowledgment", "Single acknowledgment" and "Single confirm", are activated in the alarm view by default and can be operated in runtime.

See also

Acknowledgment model (Page 226)

Acknowledging alarms (Page 225)

Acknowledging alarms (Page 277)

Workflow for configuring alarms (Page 230)

Creating alarm classes (Page 231)

Configuring discrete alarms (Page 238)

Configuring analog alarms (Page 241)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

4.3 Exporting and importing alarms (RT Uni)

4.3.1 Exporting alarms (RT Uni)

Introduction

WinCC makes an export function available for alarms.

Requirements

- The WinCC project for export is open.
- Alarms have been created in the project.
- The "HMI alarms" editor is open.

Exporting alarms

To export alarms from a WinCC project, follow the steps below:

1. Click the button in the "Discrete alarms" or "Analog alarms" tab.
   The "Export HMI alarms" dialog box opens.
2. Click "..." and specify the file in which data is saved.
3. Specify whether you want to export "Discrete alarms" and/or "Analog alarms".
4. Click "Export".
   The export starts. When the export is complete, a message on completion of the export is displayed.
5. Confirm the message on completion of the export with "OK".

Result

The exported data has been written to an xlsx file. The xlsx file has been stored in the specified folder.

If you have only exported discrete alarms, the xlsx file has the worksheet "DiscreteAlarms". If you have only exported analog alarms, the xlsx file has the worksheets "AnalogAlarms" and "Limits". If you have exported discrete alarms and analog alarms, the xlsx file has the worksheets 'DiscreteAlarms", "AnalogAlarms" and "Limits".

Each alarm is in a separate row in the xlsx file.

Note

The list entries with the "FieldInfo" designation specify whether the alarm text contains dynamic parameters. The settings are separated by a semicolon ";".

4.3.2 Importing alarms (RT Uni)

Introduction

WinCC makes an import function available for alarms. Alarms are identified by their alarm ID. An existing alarm is overwritten by the data from the import file if the alarm ID already exists in the project on import. A new alarm is created in the project if the alarm does not yet exist in the project on import.

Requirements

- An xlsx file with alarms has been created.
- The xlsx file has the same structure as an xlsx file that is created when alarms are exported.
- The IDs and names that were assigned for messages in the xlsx file are unique throughout the project.
- The WinCC project for import is open.
- The "HMI alarms" editor is open.

Importing alarms

To import alarms into a WinCC project, follow the steps below:

1. Click the button in the "Discrete alarms" or "Analog alarms" tab.
   The "Import HMI alarms" dialog box opens.
2. Click "..." and select the file that you want to import.
3. Click "Import".
   The import starts. An xml log file is created on import. When the import is complete, a message on completion of the import is displayed.

   Note

   To open the created xml log file, click the link "Click here to view the log file". It is advisable to open the xml log file, especially if the import was completed with warnings.

4. Confirm the message on completion of the import with "OK".

4.4 Configuring an alarm control (RT Uni)

4.4.1 Configuring an alarm control (RT Uni)

Introduction

The alarm view is configured for a screen. Current or logged alarms are displayed in the alarm view in runtime. More than one alarm can be displayed simultaneously, depending on the configured size. Configure the criteria for alarm filtering.

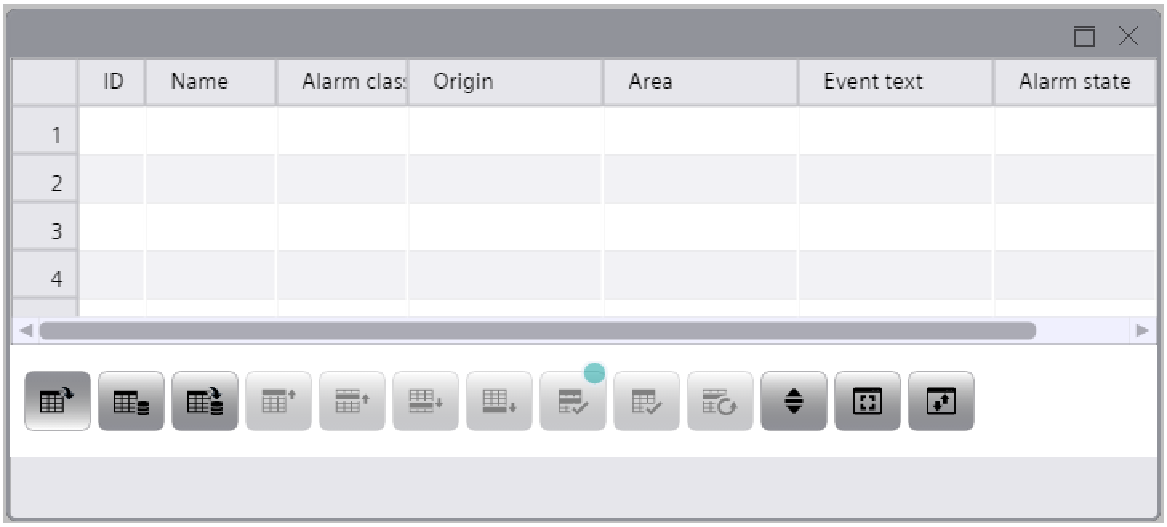
You can also configure multiple alarm views with different contents and in different screens.

Requirement

- A screen is open.
- The "Toolbox" task card is open.

Procedure

1. Insert an "Alarm view" object from the "Tools" task card into the screen.



	ID	Name	Alarm clas	Origin	Area	Event text	Alarm state
1							
2							
3							
4							

2. Go to "Properties" and set the required height, width and position of the alarm view.
3. Under "Properties > Alarm view" you define the layout and color composition of the alarm view as well as the design of the header and the contents of the table grid.
4. Select the property "Show recent" to display the latest alarm that is selected in the alarm view.
   The visible area of the alarm view moves, if necessary.

5. Under "Alarm source" you specify which alarms the alarm view displays in runtime by default.
   Depending on your task or the requirements in your company, you can select from the following display options:
   - "Not configured": The alarm view does not show any alarms.
   - "Pending alarms": The alarm view shows the currently pending alarms.
   - "Logged alarms": The alarm view shows the logged alarms.
   - "Logged alarms updated": The alarm view shows the logged alarms that are updated at specified intervals.
   - "Alarm definition": The alarm view shows all alarms configured in the engineering system, regardless of whether or not they have occurred.

   Depending on your selection, the view in the alarm view already changes in the engineering system. The buttons relevant for the settings are shown as being active, while buttons that are not relevant are grayed out. These settings are applied for runtime.
6. Define which alarms are displayed in runtime by default in the alarm lists for pending alarms and for defined alarms.
   - In the selection list under "General > Active alarms", select which alarms are displayed as pending alarms.
   - In the selection list under "General > Displayed alarms", select which alarms are displayed as defined alarms.

   Depending on your task or the requirements in your company, you select one or more display options depending on the status of the alarms:
   - "None": The alarm view shows all alarms.
   - "Not suppressed": The alarm view only shows the non-suppressed alarms.
   - "Locked": The alarm view only shows the locked alarms.
   - "Suppressed by design": The alarm view only shows the alarms suppressed by design.
   - "Shelved": The alarm view only shows the shelved alarms.

   You selection is displayed by default in the alarm view when you start runtime.

   Note

   If you do not make a selection, the alarm view shows all alarms.

   Note

   You can change the display at any time in runtime even if you have selected a different display option in the engineering system under "Alarm source" or "Active alarms".

7. If necessary, select the authorization needed to operate the alarm view in runtime.
8. Under time zone you set the desired time zone by entering a decimal value for the time zone.
   - "0" and positive numerical values: The values correspond to the index values of the Microsoft time zones.
   - "-1": The local time zone of the device

Note

In runtime you also have the option of setting the time zone via a selection list.

Result

Alarms of various alarm classes are output in the alarm view during runtime. To change the view in runtime, click the configured buttons on the alarm view toolbar.

See also

Configuring toolbar and status bar (Page 255)

Configuring filters in the alarm view (Page 258)

Configuring alarm export (Page 260)

Configuring the printing of alarms (Page 261)

Show logged alarms (Page 261)

Defining the output format (Page 41)

Lists of the alarm view (Page 273)

Configuring columns and sorting (Page 257)

Alarm control (Page 76)

Configuring an alarm control for plant objects (Page 1534)

Configuring state texts of alarms (Page 237)

4.4.2 Configuring toolbar and status bar (RT Uni)

Introduction

You operate the alarm view in runtime using the buttons in the toolbar. The status bar displays status messages from the alarm view. During configuration, set the content of the toolbar and status bar.

The following buttons are visible in the alarm view by default:

- Show active alarms
- Show logged alarms
- Show and update logged alarms

- First line
- Previous line
- Next line
- Last line
- Group acknowledgment
- Single acknowledgment
- Single confirm
- Selection display
- Sorting setup

You must select the "Visibility" property for all other buttons.

Requirement

- The alarm view is selected in the screen.
- The Inspector window is open.

Configuring the toolbar

1. In the Inspector window, configure the general properties of the toolbar, such as alignment and background color, under "Properties > Toolbar".
2. In the Inspector window, enable the buttons you need in runtime, e.g. "Export" or "Print", under "Properties > Toolbar > Elements".
3. If required, configure the button display, for example background color, border and maximum and minimum size.
4. If needed, you can define a tooltip for the buttons.
5. If needed, you can define a "hotkey" for the buttons.
   To use the hotkeys, also select "Properties > Toolbar > Use hotkeys".
6. If a button is not to be operated in Runtime, deselect "Allow operator control".
   You can reactivate a deactivated a button using a script in runtime, for example.

Note

The order and functionality of the buttons are defined in the system and cannot be changed.

Configuring the status bar

1. In the Inspector window, configure the general properties of the status bar such as the font or the background color under "Properties > Status bar".
2. In the Inspector window, select the elements you need in runtime such as date, time, connection status, etc. under "Properties > Status bar > Elements".
3. To adjust the size of an element in the status bar, select "User-defined".

4. Enter the width and height in pixels.
5. To set the order of the elements, select the element in the list and move it to the desired position.

See also

Configuring an alarm control (Page 253)

Configuring columns and sorting (Page 257)

Configuring filters in the alarm view (Page 258)

Configuring alarm export (Page 260)

Configuring the printing of alarms (Page 261)

Show logged alarms (Page 261)

4.4.3 Configuring columns and sorting (RT Uni)

Introduction

You configure the order in which the columns of the alarm view are displayed in runtime.

Requirement

- The alarm control is selected in the screen.
- The Inspector window is open.

Configuring columns

1. Click "Properties > Alarm view > Columns" in the Inspector window.
2. Enable the "Visibility" property for the relevant columns.
3. Under "Alarm text block" select the content that is to be displayed in the column, e.g. "Alarm class".
4. Under "Alarm column [n] > Header > Text", enter the desired column name that is to be displayed in the alarm view.

Note

For the column names to be configured as multilingual, you must enter the name of the column under "Alarm column [n] > Header > Text".

You will then see the configured text in the Inspector window under "Texts" and can store additional languages.

If you only enter the name under "Alarm column [n] > Name", multilingual configuration is not possible.

Configuring the sorting

To sort alarms in the alarm view by column, follow these steps:

1. Select "Properties > Alarm view" > Allow sorting" so that sorting is generally possible in the alarm view in runtime.
2. Under "Properties > Alarm view > Columns" open the alarm column by which you want to initially sort the alarms, e.g. the "Priority" column.
3. Select the sorting order "1".
4. Select the desired sorting direction, e.g. "Ascending".
   - The number "1" with the arrow pointing upwards for ascending sort order is displayed in runtime in the column with sorting order "1".
   - If the sorting order "Ascending" is enabled in the alarm view, each click in the column header toggles the sorting between the ascending and descending mode.
5. To allow sorting for this column, enable "Alarm column [n] > Allow sorting".

Note

You can configure any sorting order.

If the "Show recent" property was selected under "Properties", the latest alarm is always shown first.

See also

Configuring an alarm control (Page 253)

Configuring toolbar and status bar (Page 255)

Configuring filters in the alarm view (Page 258)

Configuring alarm export (Page 260)

Configuring the printing of alarms (Page 261)

Show logged alarms (Page 261)

4.4.4 Configuring filters in the alarm view (RT Uni)

Introduction

You can filter the display of alarms in the alarm control. You configure a static value, a tag or a script for the filter. You can configure this function in the alarm control in the "Screens" editor. To filter the alarms in Runtime, click "Selection display" in Runtime.

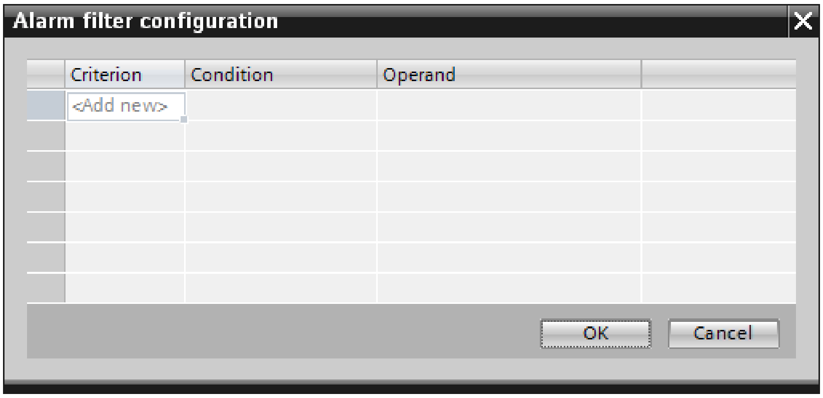
You can filter by all parameters, such as ID, name, alarm class, priority, etc.

Requirement

- The alarm control is selected in the screen.
- The Inspector window is open.

Procedure

1. In the Inspector window under "Properties > Filter", click on the "..." button in the "Static value" column.
   The "Alarm filter configuration" dialog box opens.



2. Create a filter. To create a filter that filters for alarms with the alarm class "Alarm" and with a priority of less than or equal to 5, for example, execute the following steps:
   - In the "Criterion" column, double-click "<Add>".
   - In the "Criterion" column, open the selection list and select the entry "Alarm class".
   - In the "Condition" column, open the selection list and select the entry "Equal to".
   - Enter the value "Alarm" in the field of the "Operand" column.
   - In the next line, double-click "<Add>" in the "Criterion" column.
   - In the "Criterion" column, open the selection list and select the entry "Priority".
   - In the "Condition" column, open the selection list and select the entry "Less than or equal to".
   - Enter the value "5" in the field of the "Operand" column.
   - Click the "OK" button.
3. To operate the filter in runtime, enable the "Visibility" property under "Properties > Toolbar > Elements > [26] Control bar button Selection display".

Tips for effective procedure

You can also create filter criteria directly in runtime and use them as filters.

Filter by time

When filtering by time, the start and stop values are not adjusted automatically when the time base of the alarm control is changed.

Example

At a PC location with time zone "UTC + 1h", the alarm control has the "Local time zone" time base. If you filter for the time 10:00 to 11:00 and then change the time base to "UTC", you need to change the start value and stop value of the filter to 9:00 and 10:00 to display the same alarms as before.

See also

Filtering alarms in runtime (Page 275)

Configuring an alarm control (Page 253)

Configuring toolbar and status bar (Page 255)

Configuring columns and sorting (Page 257)

Configuring alarm export (Page 260)

Configuring the printing of alarms (Page 261)

Show logged alarms (Page 261)

Alarm control (Page 76)

4.4.5 Configuring alarm export (RT Uni)

Introduction

To export alarms to a "*.csv" file in Runtime, click on the "Export" button in the alarm view. You configure the "Export" button in the alarm view in the "Screens" editor.

Requirement

- The screen with the configured alarm view is open.
- The Inspector window is open.

Procedure

To configure the export of alarms, proceed as follows:

1. Select the alarm view and enable the "Visibility" property in the Inspector window under "Properties > Toolbar > Elements > Export button [29]".

You define the export settings such as the file name, the scope of the export and the format in runtime in the "Export data" dialog.

See also

Configuring an alarm control (Page 253)

Configuring toolbar and status bar (Page 255)

Configuring columns and sorting (Page 257)

Configuring filters in the alarm view (Page 258)

Configuring the printing of alarms (Page 261)

Alarm control (Page 76)

4.4.6 Configuring the printing of alarms (RT Uni)

Introduction

Click "Print" in the alarm view to print alarms in Runtime. You configure the "Print" button in the alarm view in the "Screens" editor.

Requirement

- The screen with the configured alarm view is open.
- The Inspector window is open.

Procedure

To configure the printing of alarms, follow these steps:

1. Select the alarm view and enable the "Visibility" property in the Inspector window under "Properties > Toolbar > Elements > Print button [28]".

See also

Configuring an alarm control (Page 253)

Configuring toolbar and status bar (Page 255)

Configuring columns and sorting (Page 257)

Configuring filters in the alarm view (Page 258)

Configuring alarm export (Page 260)

Show logged alarms (Page 261)

Alarm control (Page 76)

4.4.7 Show logged alarms (RT Uni)

Overview

When an alarm log is created, an alarm view also shows logged alarms in runtime.

The buttons relevant for logging, "Show logged alarms" and "Show and update logged alarms", are activated in the alarm view by default and can be operated in Runtime.

You show logged alarms in runtime using these buttons.

See also

Creating an alarm log (Page 267)

Basics of alarm logging (Page 264)

Displaying logged alarms in runtime (Page 276)

Configuring an alarm control (Page 253)

Configuring toolbar and status bar (Page 255)

Configuring columns and sorting (Page 257)

Configuring filters in the alarm view (Page 258)

Configuring the printing of alarms (Page 261)

Configuring the display of security events (Page 286)

Alarm control (Page 76)

4.4.8 Configuring the display of system diagnostic alarms (RT Uni)

Introduction

System diagnostic alarms are installed with STEP 7 and are used for monitoring states and events of a controller. To display system diagnostic alarms in an alarm view in runtime, configure first in STEP 7 and then in WinCC.

Note

WinCC only supports system diagnostic alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports system diagnostic alarms that are updated by the central alarm management in the controller.

Requirement

- There is an HMI connection between the HMI Device and a SIMATIC S7-1500 controller (as of firmware version 2.0).

Configuring the display of system diagnostic alarms in STEP 7

To configure the display of system diagnostic alarms in runtime in STEP 7, proceed as follows:

1. Open the "Device configuration" of the controller in the project tree.
2. In the "Device view" tab, select the CPU on the rack.
3. Select "Properties > General > System diagnostics" in the Inspector window.
   You will see that the option "Select system diagnostics for this device" is selected and cannot be cleared. Because the system diagnostics of the controller is always enabled.
4. Activate the option "Central alarm management in the PLC" in the Inspector window under "Properties > General > PLC alarms".
   The automatic update of system diagnostic alarms on the HMI device is enabled in the controller.
5. Open the "Common data" folder in the project tree and double-click "System diagnostic settings".
   The system diagnostic settings are opened. You can see the predefined categories of the system diagnostic alarms in the table under "Category":
   - "Error"
   - "Maintenance demanded"
   - "Maintenance required"
   - "About"
6. In the table under "Category", select the alarm categories that are to be displayed in the alarm view in runtime.
7. In the table under "Alarm class", assign common alarm classes to the alarm classes.
8. Right-click the controller in the project tree and select "Compile > Hardware (rebuild all)" in the shortcut menu.

Configuring the display of system diagnostic alarms in WinCC

To configure the display of system diagnostic alarms in WinCC in runtime, proceed as follows:

1. Open the "Runtime settings" of the HMI device in the project tree.
2. Select the option "Automatic update" under "Alarms > Controller alarms".
   The automatic update of system diagnostic alarms on the HMI device is enabled in the HMI device.
3. Select the option "System diagnostics" under "Alarms > Controller alarms".
   The display of system diagnostic alarms is enabled in runtime.
4. Configure an alarm view.

Result

The alarm view displays of the system diagnostic alarms of the controller in runtime.

See also

Sending and automatically updating complete alarm from the controller to the HMI device (Page 287)

Configuring automatic update of controller alarms on the HMI device (Page 288)

System-defined controller alarms (Page 218)

Workflow for configuring alarms (Page 230)

4.5 Logging alarms (RT Uni)

4.5.1 Basics of alarm logging (RT Uni)

Introduction

An alarm log is used to log alarms that occur in the monitored process. You can use alarm logging to analyze error states and to document the process. When you analyze the logged alarms, you can extract important business and technical information regarding the operational state of the plant.

Alarms of connected and appropriately configured PLCs are also logged and made available in all configured languages.

Operating principle

Alarm logs are created by the system in runtime. For example, when a fault or a limit violation occurs, the corresponding alarm you configured in the "HMI alarms" editor is output in runtime. Each alarm event is logged, e.g., the transition of the alarm from "incoming" to "acknowledged" status.

The logged alarms are stored in a circular log that consists of multiple single segments. The size of all segments and of an individual segment is defined under "Alarm log" in the log settings.

Each logged alarm is assigned to an alarm class. Alarm classes can be prioritized and configured in different ways to ensure clarity even with large amounts of data.

Configuration

You configure alarm logging in the "Logs" editor. You define the logging cycles. The logs are assigned to the alarm classes under "HMI alarms".

Content of the alarm log

The alarm logs are used to store all alarm data, including configuration data. You can read all properties of an alarm from the logs, e.g. alarm class, time stamp and alarm texts.

The possible number of logged alarms depends on the server used.

Note

The time stamp of a logged alarm is always specified in standard UTC format (Universal Time Coordinated).

As alarm configuration is language-specific, the logs contain a configuration data table for each language configured.

Storage media and location

Log data are stored in a database. You can further process the saved data in other programs for analysis purposes, for example.

Hard disk drives or USB sticks, for example, are suitable storage media.

Displaying logged data

You display the logged data on the device. To do so, you configure a corresponding alarm view that displays the log data. You show the logged data in runtime by using the "Show logged alarms" button.

See also

Show logged alarms (Page 261)

Defining log size, segmentation and backup (Page 265)

Creating an alarm log (Page 267)

4.5.2 Defining log size, segmentation and backup (RT Uni)

Introduction

For the log, you define the time period in which the data is written to a given log, and the maximum size of the log file.

Each log consists of a configurable number of segments. You define a size in megabytes, the start time and a time period (for example one day) for the segments.

Note

Make sure that the log size does not exceed the free memory space available. The system does not validate the selected settings. A high number of linked log segments can lead to prolonged waiting periods in the system when starting and ending runtime.

Segments

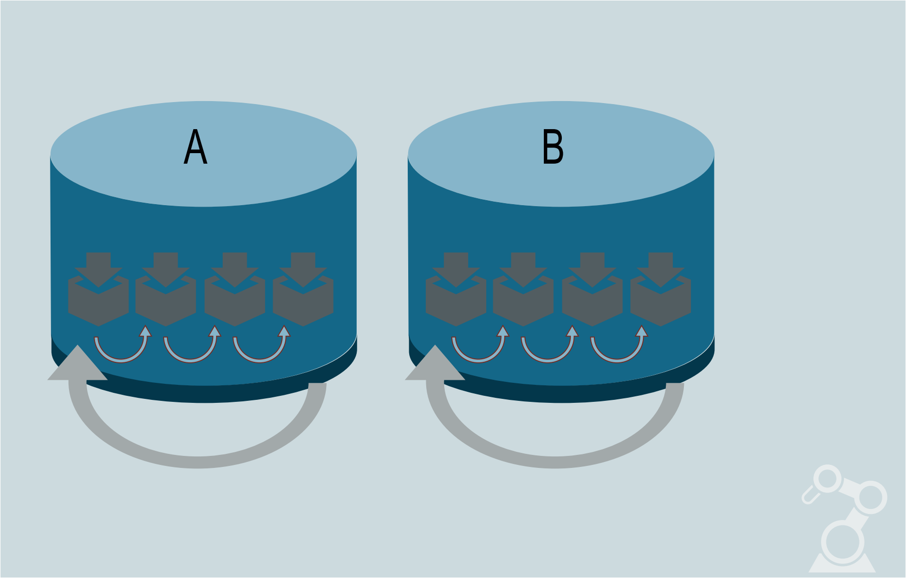
In a segmented log, multiple log segments of the same size are created, and filled in succession. When all logs are completely full, the oldest log is overwritten.

You can configure the following properties for the log segments:

- "Segment time period" defines the maximum period for one log segment, for example one day. When the log segment reaches the specified time, the current segment is closed and a new segment is created and filled with data.
- "Maximum segment size (MB)" defines the maximum size of a log segment in megabytes. When the log segment reaches the specified size, the current segment is closed and a new segment is created and filled with data.
- "Segment start time " and "Segment time period" define when to switch to the next log segment. The log segments are written to the log from the start time. The segment changes at the end of the configured time period, for example 8 hours. A new segment is created if the configured segment size is exceeded. The next log segment change then takes place at the end of the configured time period.

Response to segment change

The individual segments are filled one after the other in runtime. Once a segment is totally full, the next segment is created and filled. You can also configure the segment change at specific times. If you define a time for the segment change, the next log segment is filled when the time is reached.



The process values are written continuously to the first segment.

When the configured size of the segment is reached or the time period is exceeded, the system switches to the next segment.

When all segments are full, the oldest segment is deleted and a new segment is created.

To avoid losing process data as a result of overwriting, you can export the data to a backup.

Example

The following information has been configured:

Property	Value
Log time period	1 week
Maximum log size (MB)	700 MB
Segment time period	1 day
Maximum segment size (MB)	100 MB
Segment start time	Friday, November 23, 2017, 18:00

With the configuration suggested in the table, the started segment will be changed for the first time at 18:00 on November 23, 2017. The next time-controlled segment change will take place cyclically after periods of one day from the configured time.

Note

If you change the segmentation settings and run "Download to device", a new segment will be created.

The segment will also change if the configured size of 100 MB is exceeded in the course of one day. The oldest single segment will be deleted if the maximum log size of 700 MB is exceeded.

Backup

You can export process values from the log database as a backup. All process values contained in a log segment are exported. A log segment is always exported upon segment change, when it is full and a new segment is started. A log segment is also exported when the time set for a segment change is reached and a new segment is started.

See also

Basics of alarm logging (Page 264)

Creating an alarm log (Page 267)

4.5.3 Creating an alarm log (RT Uni)

Introduction

The configuration of an alarm log consists of the following steps:

- Create an alarm log
- Configure an alarm log, for example, select the storage location
- Select alarms for logging

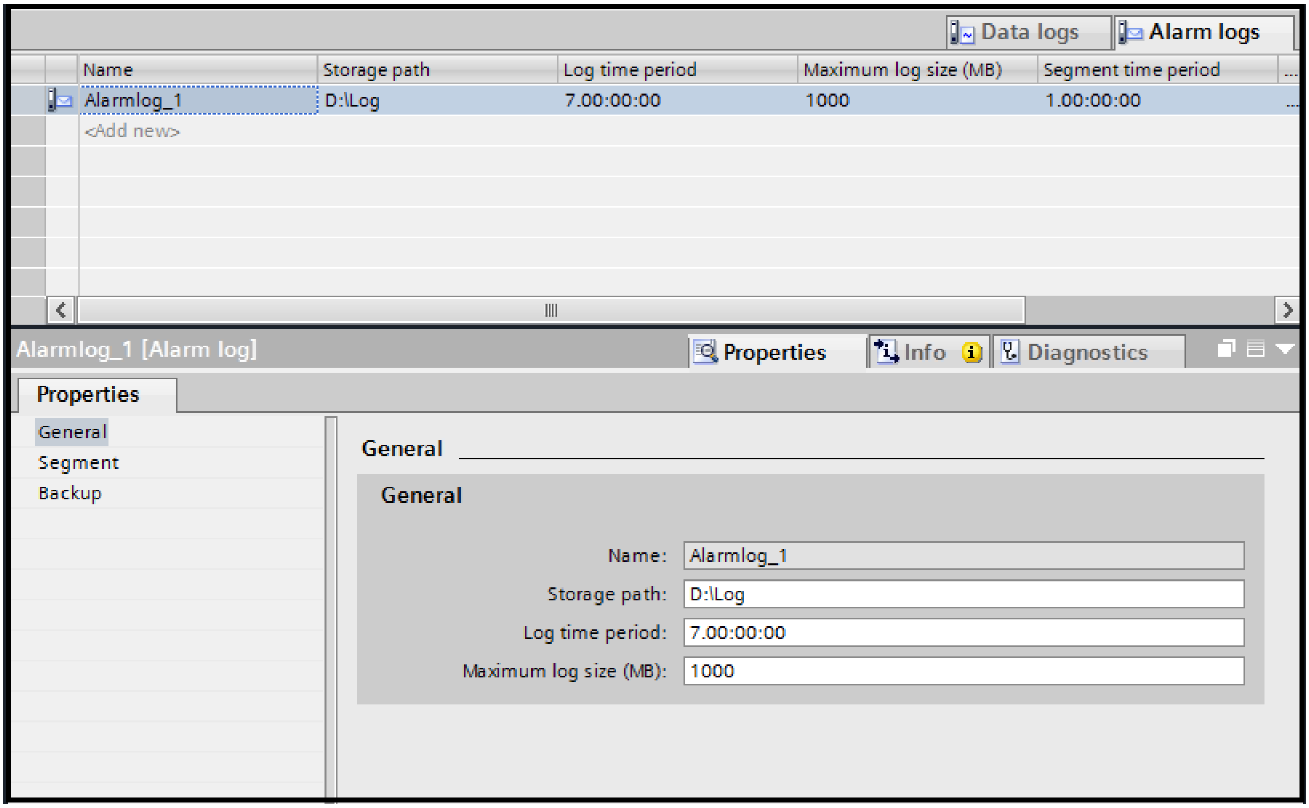
Requirement

- A project is open.
- The Inspector window is open.

Procedure

To create an alarm log, follow these steps:

1. Double-click on the "Logs" entry in the project tree.
   The "Logs" editor opens.
2. Open the "Alarm logs" tab and double-click "Add" in the "Name" column of the "Alarm logs" editor.
   A new alarm log is created.
3. In the "Storage path" field specify the storage path for the alarm log.



Name	Storage path	Log time period	Maximum log size (MB)	Segment time period
Alarmlog_1	D:\Log	7.00:00:00	1000	1.00:00:00
<Add new>				

Alarmlog_1 [Alarm log]

Properties

General
Segment
Backup

General

General

Name: Alarmlog_1
Storage path: D:\Log
Log time period: 7.00:00:00
Maximum log size (MB): 1000

Note

Do not change the storage path for the log after the first transfer to the device. Subsequent changes will result in errors during logging.

4. Define the maximum time period for logging in the "Log time period" field, for example 7 days.
   If you specify a value of "0" for the logging time period, the log will be written continuously. As soon as the maximum size is reached, the oldest segment is deleted from the log and a new segment is written.
5. Define the maximum size of the log file in megabytes in the "Maximum log size (MB)" field.
6. Define the time period and the start time as well as the maximum size of for the single segment in the "Segment" area.

   Note

   If you change the log size or the time period in runtime, the previous segment is closed and a new segment with the new settings will be created.

7. Set whether data is to be backed up and specify the path for the backup file under "Backup > Backup mode".

   Note

   We recommend creating backups of your log segments to ensure complete documentation of your process.

   The oldest single segment will be deleted if the maximum log size of 700 MB is exceeded. To prevent the loss of logged data, enable backup mode.

   If you subsequently change the primary path, the new backup file will only be written to the new storage path after loading. The previous backup file will remain in the previous storage path.

Tips for effective procedure

You configure the log properties directly in the table of the "Alarm logs" editor. To view hidden columns, activate the column titles using the shortcut menu.

Result

The alarm log is created.

See also

Show logged alarms (Page 261)

Basics of alarm logging (Page 264)

Defining log size, segmentation and backup (Page 265)

Configuring the display of security events (Page 286)

4.5.4 Assign alarm class (RT Uni)

Introduction

Requirement

Procedure

Result

4.6 Displaying and using alarms (RT Uni)

4.6.1 Displaying alarms in runtime (RT Uni)

Alarms

Alarms indicate events and states on the HMI device which have occurred in the system, in the process or on the HMI device itself. A status is reported when it is received.

An alarm could trigger one of the following alarm events:

- Incoming
- Outgoing
- Acknowledge

The configuration engineer defines which alarms must be acknowledged by the operator.

An alarm may, for example, contain the following information:

- Date
- Time
- Alarm text
- Area (fault location)
- Status
- Alarm class
- Alarm number

Alarm classes

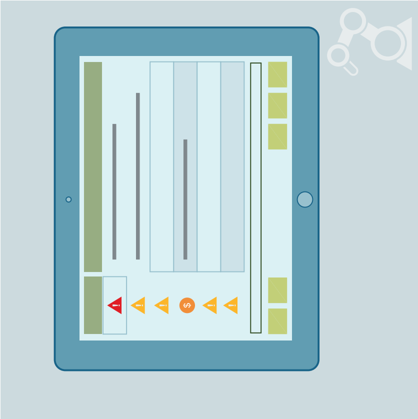
Alarms are assigned to various alarm classes. The alarm class defines how an alarm is displayed. The alarm class specifies if and how the operator has to acknowledge alarms of this alarm class. For more information on alarm classes, go to "Alarm classes".

Alarm log

Alarm events are stored in an alarm log, provided this log file is configured. The capacity of the log file is limited by the storage medium and system limits.

Alarm control

The alarm view shows selected alarms or events from the alarm buffer or alarm log. Whether alarm events have to be acknowledged or not is specified in your configuration. You can configure the order in which the alarms are displayed. At the first position, the current, or the oldest alarm will be displayed.



See also

Printing alarms in runtime (Page 284)

Operating an alarm view (Page 272)

Lists of the alarm view (Page 273)

Sorting alarms in runtime (Page 274)

Filtering alarms in runtime (Page 275)

Displaying logged alarms in runtime (Page 276)

Acknowledging alarms (Page 277)

Group acknowledgement of alarms (Page 278)

Exporting alarms (Page 279)

Shelving alarms (Page 280)

Lock alarms (Page 282)

4.6.2 Operating an alarm view (RT Uni)

Introduction

The "Alarm view" object displays alarms that occur during the process in a plant. You also use the alarm view to visualize alarms in list format. WinCC offers various views, such as "Current alarms" or "Logged alarms" views.

A shelved alarm is no longer displayed in the alarm view. The alarm is still available in the system and is logged.

If the shelving has been canceled, the alarm is again visible in its last state.

An alarm whose display has been suppressed is suppressed at the source. This alarm is not logged.

If the suppression has been canceled again, it is checked by the system and, if the cause still exits, displayed again.

Requirement

- The objects are enabled for operation.
- The operator authorization is assigned.

Operation using the mouse

1. Click on the alarm to be edited.
2. Click on the operator control whose function you wish to use.

See also

Printing alarms in runtime (Page 284)

Displaying alarms in runtime (Page 270)

Lists of the alarm view (Page 273)

Sorting alarms in runtime (Page 274)

Filtering alarms in runtime (Page 275)

Displaying logged alarms in runtime (Page 276)

Acknowledging alarms (Page 277)

Group acknowledgement of alarms (Page 278)

Exporting alarms (Page 279)

Shelving alarms (Page 280)

Lock alarms (Page 282)

4.6.3 Lists of the alarm view (RT Uni)

Introduction

The alarm view displays specific lists to provide a better overview of the active alarms. These lists filter, and sort alarms by certain properties.

Lists in the alarm view

You can display different lists in the alarm control. To display the alarm lists in the alarm view and switch the alarm view in runtime, click the associated button in the alarm view toolbar.

	List	Description
	Show active alarms	Shows the pending alarms.
	Show logged alarms	Shows the logged alarms. The display is not updated immediately when new incoming alarms occur.
	Show and update logged alarms	Shows the logged alarms. The display is updated immediately when new incoming alarms occur.
	Show defined alarms	Shows the alarms configured in the engineering system.

See also

Configuring an alarm control (Page 253)

Displaying alarms in runtime (Page 270)

Operating an alarm view (Page 272)

Sorting alarms in runtime (Page 274)

Printing alarms in runtime (Page 284)

Filtering alarms in runtime (Page 275)

Displaying logged alarms in runtime (Page 276)

Acknowledging alarms (Page 277)

Group acknowledgement of alarms (Page 278)

Exporting alarms (Page 279)

Shelving alarms (Page 280)

Lock alarms (Page 282)

4.6.4 Sorting alarms in runtime (RT Uni)

Introduction

In runtime, you can sort the alarms in the alarm view by column header.

Examples for sorting alarms:

- In descending order by date, time, and alarm number. The most recent alarm is displayed at the top.
- By priority
  As a result, in a single-line alarm view, only the top-priority alarm appears in the alarm window. A lower-priority alarm will not be displayed, even if it is more recent. The alarms are displayed in chronological order.
- By their "state"
  For an ascending sort order, the following order is used:
  - Incoming
  - Incoming/acknowledged
  - Incoming/acknowledged/outgoing
  - Incoming/outgoing/acknowledged
  - Shelved
  - Suppressed

When the alarm view is sorted by columns, an arrow and a number are shown on the right in the column header. The arrow indicates the sort order (ascending or descending). The number beside the arrow indicates the sort order of the column headers.

Requirement

- "Allow sorting" is enabled in the alarm view for the respective columns.

Procedure

To sort alarms in the alarm view by column, follow these steps:

1. In the alarm view, click the column header in the respective column.
   The alarms are sorted accordingly.

See also

Displaying alarms in runtime (Page 270)

Lists of the alarm view (Page 273)

Filtering alarms in runtime (Page 275)

Printing alarms in runtime (Page 284)

Operating an alarm view (Page 272)

Displaying logged alarms in runtime (Page 276)

Acknowledging alarms (Page 277)

Group acknowledgement of alarms (Page 278)

Exporting alarms (Page 279)

Shelving alarms (Page 280)

Lock alarms (Page 282)

4.6.5 Filtering alarms in runtime (RT Uni)

Introduction

In runtime, you can set specific criteria that define the alarms to be displayed in the alarm view. In the example below, only alarms that contain the alarm text "Motor on" are displayed.

Requirement

The "Selection display" button is configured in the alarm view.

Procedure

To filter alarms in the alarm view, proceed as follows:

1. Click "Selection display" in Runtime.
   The "Selection" dialog opens.
2. Under "Criterion" select the criterion "Alarm text".
3. Enter the alarm text "Motor on" in the "Settings" column.
   The alarm view only shows those alarms that contain the words "Motor on" in their alarm text.

Note

If necessary, define additional filter criteria by selecting the required condition in the "AND/OR" column and the respective criterion in the "Criterion" column.

See also

Configuring filters in the alarm view (Page 258)

Displaying alarms in runtime (Page 270)

Sorting alarms in runtime (Page 274)

Displaying logged alarms in runtime (Page 276)

Printing alarms in runtime (Page 284)

Operating an alarm view (Page 272)

Lists of the alarm view (Page 273)

Acknowledging alarms (Page 277)

Group acknowledgement of alarms (Page 278)

Exporting alarms (Page 279)

Shelving alarms (Page 280)

Lock alarms (Page 282)

Alarm control (Page 76)

4.6.6 Displaying logged alarms in runtime (RT Uni)

Introduction

In runtime, the alarm view displays alarms from the log, in addition to the current alarms.

Requirements

- All the archived data that you intend to display in runtime must be stored locally on the archive server. The alarm log does not allow access to backup files held elsewhere, such as on another computer in the network.
- The buttons "Show logged alarms" and "Show and update logged alarms" are configured in the alarm view.

Procedure

To show logged alarms in runtime, follow these steps:

1. In the alarm view, click "Show logged alarms" to display logged alarms. Only logged alarms are displayed.
2. In the alarm view, click "Show and update logged alarms" to display logged and recent alarms. Any new incoming alarms will be updated immediately in the view.

See also

Show logged alarms (Page 261)

Displaying alarms in runtime (Page 270)

Filtering alarms in runtime (Page 275)

Acknowledging alarms (Page 277)

Printing alarms in runtime (Page 284)

Operating an alarm view (Page 272)

Lists of the alarm view (Page 273)

Sorting alarms in runtime (Page 274)

Group acknowledgement of alarms (Page 278)

Exporting alarms (Page 279)

Shelving alarms (Page 280)

Lock alarms (Page 282)

Configuring the display of security events (Page 286)

Alarm control (Page 76)

4.6.7 Acknowledging alarms (RT Uni)

Introduction

You can acknowledge alarms in runtime according to your project configuration settings. You can acknowledge alarms as follows:

- In the alarm view with the buttons "Single acknowledgment", "Group acknowledgment" and for alarms with acknowledgment and confirmation also with the "Single confirm" button.

If an operator authorization is configured for the operator controls, the alarms can only be acknowledged by authorized users.

The number of alarms to be acknowledged is indicated by a counter at the "Single acknowledgment" button or, if the alarm view was configured accordingly in engineering, by the status bar.

Acknowledgment variants

You acknowledge individual alarms or multiple alarms together in runtime. The following options are possible:

- Single acknowledgment
  Acknowledgment of an alarm using the "Single acknowledgment" button
- Group acknowledgment
  Acknowledgment of all pending, visible alarms that require acknowledgment in the alarm view using the "Group acknowledgment" button in the alarm view.
- Acknowledgment and confirmation
  When an alarm requires acknowledgment and confirmation, you acknowledge that the alarm is incoming or outgoing. Once the alarm has gone out, you reset the alarm with the "Single confirm" button of the alarm view.

Requirement

An alarm is displayed on the HMI device.

Procedure

To acknowledge an alarm, follow these steps:

1. Click the "Show current" button in the alarm view.
2. Select the alarm.
3. Click "Single acknowledgment" in the alarm view.

Result

The alarm status is set to "Acknowledged". If the trigger condition for an alarm no longer applies, the alarm state is also set to "outgoing" and no longer displayed on the device.

See also

Displaying alarms in runtime (Page 270)

Printing alarms in runtime (Page 284)

Configuring alarm acknowledgment (Page 250)

Displaying logged alarms in runtime (Page 276)

Group acknowledgement of alarms (Page 278)

Operating an alarm view (Page 272)

Lists of the alarm view (Page 273)

Sorting alarms in runtime (Page 274)

Filtering alarms in runtime (Page 275)

Lock alarms (Page 282)

Shelving alarms (Page 280)

Exporting alarms (Page 279)

Alarm control (Page 76)

4.6.8 Group acknowledgement of alarms (RT Uni)

Introduction

The acknowledgement of all pending, visible alarms in the alarm window that need to be acknowledged is known as a group acknowledgement.

Requirement

There are several alarms that require acknowledgement in the alarm view.

Procedure

For group acknowledgement of alarms, follow these steps:

1. Read the alarm texts of the pending alarms and perform corrective actions, if necessary.
2. Click the "Group acknowledgment" button in the alarm view.

Result

All pending alarms with the following properties have been acknowledged:

- Requires acknowledgement
- Visible

See also

Printing alarms in runtime (Page 284)

Displaying alarms in runtime (Page 270)

Acknowledging alarms (Page 277)

Exporting alarms (Page 279)

Operating an alarm view (Page 272)

Lists of the alarm view (Page 273)

Sorting alarms in runtime (Page 274)

Filtering alarms in runtime (Page 275)

Displaying logged alarms in runtime (Page 276)

Shelving alarms (Page 280)

Lock alarms (Page 282)

Alarm control (Page 76)

4.6.9 Exporting alarms (RT Uni)

Introduction

In runtime you export the data directly from the alarm view, for example, for further processing or analysis.

Requirement

An alarm view with the "Export" button is displayed on the device.

Procedure

To export data from the alarm view, follow these steps:

1. Click the "Export" button in the alarm view.
2. Under "File name" specify the file name of the export file.
3. Under "Scope of data export" specify which data is to be exported from the alarm view.
4. Under "Format" select the format of the export file.
5. Confirm with "OK".
   The export file appears in the browser download and can be downloaded.

See also

Displaying alarms in runtime (Page 270)

Group acknowledgement of alarms (Page 278)

Shelving alarms (Page 280)

Printing alarms in runtime (Page 284)

Operating an alarm view (Page 272)

Lists of the alarm view (Page 273)

Sorting alarms in runtime (Page 274)

Filtering alarms in runtime (Page 275)

Displaying logged alarms in runtime (Page 276)

Acknowledging alarms (Page 277)

Lock alarms (Page 282)

Alarm control (Page 76)

4.6.10 Shelving alarms (RT Uni)

Introduction

You shelve an alarm, for example, to prevent an error alarm from impairing the effectivity of your system.

Shelving can be canceled at any time. To do so, you use the buttons "Shelve alarm" and "Unshelve alarm" in runtime.

If an operator authorization is configured for these control elements, the alarms can only be shelved by authorized users.

Requirement

- The "Visibility" and "Allow operator control" settings have been activated for the following buttons in the engineering system:
  - "Shelve alarm"
  - "Unshelve alarm"
  - "Show defined alarms"
- To unshelve:
  The "Show defined alarms" alarm list is configured in such a way that shelved alarms are displayed.
  Alternatively: If the "Visibility" and "Allow operator control" settings are enabled for the "Disabled alarms setup" button, you can change the alarm list configuration in Runtime with this button.
- The "Show defined alarms" alarm list is configured in such a way that shelved alarms are displayed.
- An alarm is displayed on the HMI device.

Procedure

To shelve an alarm, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm view.
2. Select the alarm.
3. Click the "Shelve alarm" button.

The alarm is shelved. It depends on the alarm list settings whether the alarm is visible in the alarm lists for active alarms and for defined alarms.

Shelved alarms are still available and logged in the system.

Displaying shelved alarms

To display the currently shelved alarms, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm view.
2. Click the "Disabled alarms setup" button.
3. Activate the option for shelved alarms.

Unshelving an alarm

To unshelve an alarm, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm view.
2. Select the alarm.
3. Click "Unshelve alarm".

If the shelving has been canceled, the alarm is again visible in its last state.

See also

Displaying alarms in runtime (Page 270)

Exporting alarms (Page 279)

Lock alarms (Page 282)

Printing alarms in runtime (Page 284)

Operating an alarm view (Page 272)

Lists of the alarm view (Page 273)

Sorting alarms in runtime (Page 274)

Filtering alarms in runtime (Page 275)

Displaying logged alarms in runtime (Page 276)

Acknowledging alarms (Page 277)

Group acknowledgement of alarms (Page 278)

Alarm control (Page 76)

4.6.11 Lock alarms (RT Uni)

Introduction

You can lock alarms to avoid an excessive burden of information for the plant operator. The operator will find it easier to concentrate on the important alarms if only selected alarms are shown.

You can unlock the locked alarms at any time. To do so, you use the buttons "Lock alarm" and "Unlock alarm" in runtime.

Locked alarms are automatically visible again when Runtime restarts.

Requirement

- The "Visibility" and "Allow operator control" settings have been activated for the following buttons in the engineering system:
  - "Lock alarm"
  - "Unlock alarm"
  - "Show defined alarms"
- To unlock:
  The "Show defined alarms" alarm list is configured in such a way that locked alarms are displayed.
  Alternatively: If the "Visibility" and "Allow operator control" settings are enabled for the "Disabled alarms setup" button, you can change the alarm list configuration in Runtime with this button.

- The user is authorized to lock and unlock alarms.

Note

The "Lock alarms" and "Unlock alarms" authorizations must be configured directly one under the other. This is necessary because the authorization level used automatically for the "Unlock alarms" authorization is directly below the "Lock alarms" authorization.

- An alarm is displayed on the HMI device.

Procedure

To lock an alarm, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm view.
2. Select the alarm.
3. Click the "Lock alarm" button.
   The alarm is removed from the alarm list.

The alarm is locked. It depends on the alarm list settings whether the alarm is visible in the alarm lists for active alarms and for defined alarms.

Displaying locked alarms

1. Select the alarm list "Show defined alarms" in the alarm view.
2. Click the "Disabled alarms setup" button.
3. Activate the option for locked alarms.

Unlocking messages

1. Select the alarm list "Show defined alarms" in the alarm view.
2. Select the alarm.
3. Click the "Unlock alarm" button.

Properties of locked alarms

The following applies to locked alarms:

- A locked alarm is not logged.
- If a locked alarm is unlocked again, it is checked by the system and, if the cause still exists, displayed again.

See also

Displaying alarms in runtime (Page 270)

Shelving alarms (Page 280)

Printing alarms in runtime (Page 284)

Operating an alarm view (Page 272)

Lists of the alarm view (Page 273)

Sorting alarms in runtime (Page 274)

Filtering alarms in runtime (Page 275)

Displaying logged alarms in runtime (Page 276)

Acknowledging alarms (Page 277)

Group acknowledgement of alarms (Page 278)

Exporting alarms (Page 279)

Alarm control (Page 76)

4.6.12 Printing alarms in runtime (RT Uni)

Introduction

In runtime you print the data directly from the alarm view, for example, for further logging or analysis.

Requirement

- Several alarms are displayed on the device.
- A printer is configured.

Procedure

1. Filter the alarm view using the alarm view controls, if necessary.
2. Click the "Print" button in the alarm view.
   Depending on the browser settings, the print preview appears in a new browser tab.
3. Click "Print".

Result

The alarms displayed in the alarm window are output on the printer.

See also

Displaying alarms in runtime (Page 270)

Acknowledging alarms (Page 277)

Operating an alarm view (Page 272)

Group acknowledgement of alarms (Page 278)

Lock alarms (Page 282)

Shelving alarms (Page 280)

Exporting alarms (Page 279)

Displaying logged alarms in runtime (Page 276)

Filtering alarms in runtime (Page 275)

Sorting alarms in runtime (Page 274)

Lists of the alarm view (Page 273)

Alarm control (Page 76)

4.7 Display security events (RT Uni)

4.7.1 Display security events on the HMI device (RT Uni)

Introduction

In addition to the existing alarms in WinCC, you can also view security events on the HMI device.

Security events are, for example, an attack on a device over the network or a change of the protection level for communication between the controller and the HMI device.

Security events are detected by the controller and passed on to the HMI device. Security events are displayed in the alarm log on the HMI device.

It is not necessary to configure or activate the security event functionality within the controller. Security events are automatically detected by the controller.

Configuring the display of security events

The following steps are necessary to display security events on the HMI Device:

- Selection of controller alarms
- Creation of an alarm log for controller alarms

You can find more detailed information on configuration here: Configuring the display of security events (Page 286)

Notes

- WinCC only supports security events of a SIMATIC S7-1500 controller.
- WinCC only supports security events that are automatically updated by the central alarm management in the controller.
- Security events always use the "System information" alarm class.

See also

Configuring the display of security events (Page 286)

Sending and automatically updating complete alarm from the controller to the HMI device (Page 287)

Logging alarms (Page 264)

Alarm system (Page 213)

Alarms (Page 215)

4.7.2 Configuring the display of security events (RT Uni)

Requirement

- There is an HMI connection between the HMI Device and a SIMATIC S7-1500 controller (as of firmware version 2.0).
- The option "Central alarm management in the PLC" is selected (the automatic update of security events on the HMI device is enabled in the controller).

Procedure

1. Open the "Runtime settings" of the HMI device.
2. Select the option "Automatic update" under "Alarms > Controller alarms".
   The automatic update of security events on the HMI device is enabled in the HMI device.
3. Select the option "Security events" under "Alarms > Controller alarms".
   The display by security events in runtime is enabled.

   Note

   The "Security events" option is cleared by default and must be selected for each HMI connection.
4. Create a new alarm log in the "Log" editor under "Alarm logs".
5. Open an HMI screen.
6. Create an alarm view.

Result

The security events are displayed in the alarm log in runtime.

See also

Display security events on the HMI device (Page 285)

Configuring automatic update of controller alarms on the HMI device (Page 288)

Creating an alarm log (Page 267)

Show logged alarms (Page 261)

Displaying logged alarms in runtime (Page 276)

4.8 Sending complete alarm from the controller to the HMI device (RT Uni)

4.8.1 Sending and automatically updating complete alarm from the controller to the HMI device (RT Uni)

Basics

In addition to alarms in WinCC, you can configure controller alarms in STEP 7 and display them on your HMI device.

When controller alarms are configured in STEP 7, an HMI connection to a SIMATIC S7-1500 controller is established and controller alarms are currently pending, the controller alarms are automatically sent to the HMI devices and updated in case of alarm changes (e.g. change to alarm text). This will save you time because you do not have to load configuration changes of the alarms to the HMI device separately. The HMI device does not need to exit Runtime operation when the alarms are changed.

The following controller alarms can be sent to the HMI device:

- Program alarms
- ProDiag alarms
- GRAPH alarms
- System diagnostic alarms

The controller alarms can be sent completely to the HMI device if corresponding settings are configured in the controller and on the HMI device. On the HMI Device, the option "Automatic update" under "Runtime settings > Alarms > Controller alarms" must be selected for the respective connection. You can find additional information on the settings at Configuring automatic update of controller alarms on the HMI device (Page 288).

Device dependency

If the controller and the HMI device are configured accordingly, the controller alarms from the following controller are sent automatically and completely to the HMI device when they occur:

- SIMATIC S7-1500 (firmware version 2.0 and higher)

Language settings

For alarms to be displayed in the correct language on the HMI device, the same three languages or fewer must be configured for the alarms in the controller and on the HMI device. You may have to coordinate the language selection with the configuration engineer.

If different languages are configured on the HMI device and in the controller, the HMI device in operation shows the text "###Text missing###" instead of the controller alarms.

Notes

- If the "Only information" option was activated for a program alarm in STEP 7, the program alarm uses the "Information" alarm class.
- Controller alarms that are automatically updated by the central alarm management in the controller cannot be shelved or manually suppressed.

See also

Filtering controller alarms via display classes (Page 249)

Configuring automatic update of controller alarms on the HMI device (Page 288)

Configuring the display of system diagnostic alarms (Page 262)

Display security events on the HMI device (Page 285)

User-defined controller alarms (Page 216)

System-defined controller alarms (Page 218)

Workflow for configuring alarms (Page 230)

4.8.2 Configuring automatic update of controller alarms on the HMI device (RT Uni)

Introduction

The "Automatic update" option is selected by default for a connection between a SIMATIC S7-1500 controller (firmware version 2.0 or higher) and an HMI device.

Requirement

- There is an HMI connection between the HMI Device and a SIMATIC S7-1500 controller (as of firmware version 2.0).
- The option "Central alarm management in the PLC" is selected in the properties of the controller (the automatic update of controller alarms on the HMI device is enabled in the controller).
- Controller alarms were configured in STEP 7.
- An alarm view is configured on the HMI device.
- The same three languages (or fewer) are configured in the controller and on the HMI device for alarms.

Procedure

1. Open the "Runtime settings" of the HMI device.
   One or more connections to controllers are displayed under "Alarms > Controller alarms".
2. Activate the "Automatic update" option for the respective connection for which you want to display the controller alarms.
   The "Automatic update" option must be selected separately for each connection.

Note

If the languages configured for the controller differ from those for the HMI device, the alarms cannot be displayed in runtime. Instead, the alarm "##Text missing##" is displayed.

Result

In runtime, the controller alarms are displayed in the alarm view.

See also

Sending and automatically updating complete alarm from the controller to the HMI device (Page 287)

Filtering controller alarms via display classes (Page 249)

Configuring the display of system diagnostic alarms (Page 262)

Configuring the display of security events (Page 286)

User-defined controller alarms (Page 216)

System-defined controller alarms (Page 218)

Workflow for configuring alarms (Page 230)

Archiving data (RT Uni) 5

5.1 Log basics (RT Uni)

Introduction

WinCC provides the following types of log for logging process data for HMI Runtime:

- Data logs
- Alarm logs

A data log is used to log process data from an industrial plant.

An alarm log is used to log alarms that occur in the monitored process.

Principle

The two types of log both have roughly the same structure and largely work in the same way. They are very clear and easy to configure. With both types of log you define the same properties for the log. The same logging methods are also available for both types of log.

The following logging methods are available:

- Circular log
  If a circular log is totally full, the oldest entries are overwritten.
- Segmented circular log
  In a segmented circular log, multiple single logs of the same size are filled in succession. When all log segments are filled, the oldest log segment is overwritten.
- Log with level-dependent system alarm
  A system alarm is triggered when a defined level is reached.
- Log with level-dependent triggering of an event
  The "Overflow" event is triggered when the log is completely full. The "Overflow" event triggers a system function.

Database types

The following table shows which database types are supported by the various HMI devices:

HMI device	Supported database type	Supported database language
Unified RT system	File-based logging	SQLite (default)
	Database logging	Microsoft SQL (option)
Unified Comfort Panel	File-based logging	SQLite
PC systems	File-based logging	SQLite
	Database logging	Microsoft SQL

Note

Microsoft SQL on Unified RT systems

- Unified RT systems use SQLite as the default database type. To use Microsoft SQL, the system provides an installation option with a setup package. Please note that after installing Microsoft SQL, SQLite logging will no longer be possible.
- Existing SQLite files remain, but are not available via HMI RT.

Note

Database for simulation

If you are using Microsoft SQL on the RT system, Microsoft SQL is also automatically used for simulation. The configuration of the storage location is ignored during simulation; a relative path is always used in the project folder instead of the set storage location.

5.2 Properties of logs (RT Uni)

Introduction

You define the properties of a data log in the "Data logs" editor.

You define the properties of an alarm log in the "Alarm logs" editor.

The properties of the data log and alarm log are configured in the same way. You configure the properties either directly in the table of the respective editor or in the log properties of the Inspector window.

General properties

Name

You can assign any name to the log. The name must contain at least one letter or one number. You can create multiple logs, but the names must be unique.

Storage location

The storage location determines where the log is stored. For the specification, "Storage medium" and "Storage directory" are specified. Which storage locations are available depends on the HMI device.

Size

The size of a log depends on the type of log and the selected settings.

- Size of a data log
  The size of a data log is calculated as follows:
  The number of items * the length of each tag value to be logged.
  In the Properties window, the maximum size that the log accepts for retention of the currently selected number of data records is displayed in the input field "Number of data records". The maximum log size is limited by the volume of the storage medium.
- Size of an alarm log
  The size of an alarm log is calculated from the number of data records and the approximate size of an entry. The size of an entry depends on whether the alarm text and the associated tag values are to be logged as well.

Restart characteristics

- Under Restart characteristics you can specify that the logging starts when Runtime starts. The activation takes place via the check box in "Enable logging at runtime start".
  You can also control the behavior when runtime starts:
  - "Reset log": If you want to overwrite existing logged data with the new data.
  - "Continue log": If you want to receive data that has already been logged. This setting adds the data to be logged to an existing log.

Note

You can use system functions to control the restarting of a log in Runtime.

Automatic log entries

In Runtime, the following log entries are created as standard:

Entry	File format	Log type	Meaning
RT_DIS	Any	Data log	Indicates that the connection to the log was interrupted at this point in time. (A bold line is shown in the trend view for this time period.)
RT_OFF	Any	Data log	Indicates that Runtime was shut down at this point in time. (No line is shown in the trend view for this time period.)
RT_ERR	Any	Data log Alarm log AuditTrail[1]	Indicates in the destination log that a copy operation was not successful or was interrupted. (The log copy was not fully created.)
$RT_COUNT	*.CSV *. TXT	Data log Alarm log AuditTrail[1]	This entry was created at the end of the log and serves to increase the system performance when Runtime starts.

[1] The "AuditTrail" logging method is not available for all HMI devices.

Security-related properties

Logs are protected in various ways:

- Protection against loss of data
- Protection against manipulation

Protection against loss of data

Even after an interruption of the power supply, all data is available unchanged.

In the event of an unintentional interruption of the power supply, the runtime is suddenly and unexpectedly switched off. In such a case, the data already logged is backed up and is available unchanged and completely after the power supply has been restored and the system has been rebooted. Logging can be continued with the same database.

Protection against manipulation

Logs can contain sensitive and confidential content such as performance parameters or product data that must be protected from unintentional or unauthorized modification.

Both the tag logs and the alarm logs can be protected using standard tools in the Windows security settings. On Microsoft SQL servers, you protect the log databases by using the Windows group "Simatic HMI" (read/write) and "Simatic HMI Viewer" (read). Only members of these groups have direct access to the databases.

Note

The manipulation protection is currently only valid for WinCC Unified PC systems. For Unified Comfort Panel, this feature will be made available shortly.

5.3 Working with logs (RT Uni)

Delete log contents

The contents of an alarm log or tag log can be deleted using a system function. The log itself remains; only the alarm or log data saved in it is deleted. This may be useful after a test phase is over, for example, if existing logs are to be emptied. You can find detailed information on using system functions in the sectionSystem functions (Page 308).

Editing log contents

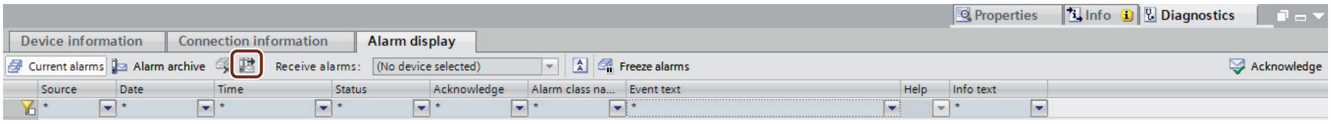
Existing contents of a log can be commented, corrected or expanded in the alarm control in Runtime. You can find more information on alarm views in the section Alarm control (Page 76).

Exporting logs

To ensure data processing by the most widely used data processing programs, logs can be exported to the following formats:

- CSV
- TXT
- XML

Export logs by clicking the "Export log" icon in the alarm display. A filter allows the exact specification of the content to be exported.



Logging mode

You have the option to select one of three "logging modes" for logging tags. This can happen in two ways:

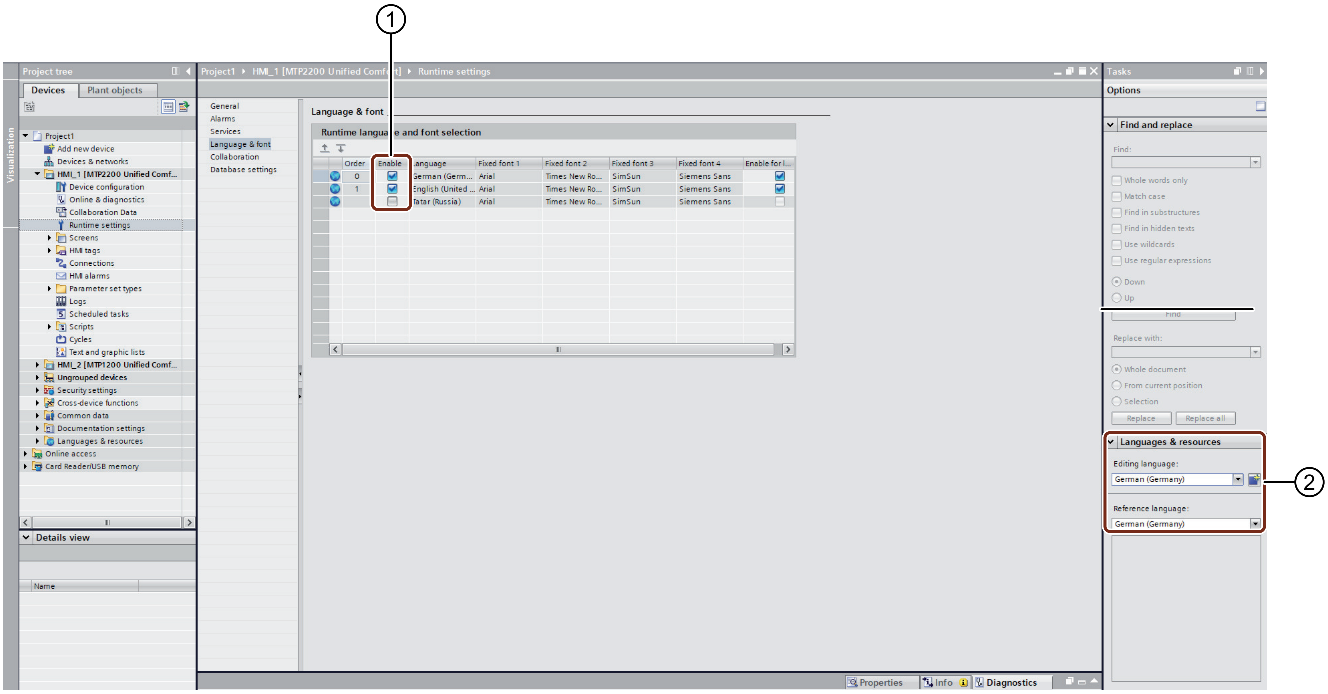
- In the "Logging tags" tab, select a mode by opening the drop-down menu in the "Logging mode" column.
- In the Inspector window, go to "Properties > General" and open the drop-down menu under "Logging mode" to select a mode.

The following options are available to you as logging modes:

- Cyclic: Logging takes place according to the set cycle.
- On demand: Logging is triggered by a state change of a preconfigured trigger tag.
- On change: Logging only takes place if there is a value change in the tags to be logged.

Logging in different languages

You can create logs in different languages. In the "Runtime settings", select the desired languages in the "Activate" column under "Language and Font". You can add more languages by selecting the desired languages in "Languages and Resources". Alarm texts are then created, saved and displayed in the selected languages.



Language & font

Runtime language and font selection

Order	Enable	Language	Fixed font 1	Fixed font 2	Fixed font 3	Fixed font 4	Enable for l...
0	☑	German (Germ...	Arial	Times New Ro...	SimSun	Siemens Sans	☑
1	☑	English (United ...	Arial	Times New Ro...	SimSun	Siemens Sans	☑
	□	Tatar (Russia)	Arial	Times New Ro...	SimSun	Siemens Sans	□

Editing language: German (Germany)

Reference language: German (Germany)

① Activation of the available languages

② Selection of the languages which are to be available

5.4 Storage locations of logs (RT Uni)

Storage location of a log

When you configure a log in WinCC, the available storage locations depend on the HMI device you are using.

HMI devices	Supported logs			Supported storage locations	Supported storage locations
	Alarms	Tags	Audit Trail		
Basic Panels[1]	No	No	No	-	-
Basic Panels 2nd Generation[2]	Yes	Yes	No	a TXT file (Unicode)	USB memory (at USB port)
Comfort Panels[3]	Yes	Yes	Yes	a CSV file (ASCII) RDB file a TXT file (Unicode)	Storage card (SD) Storage card (USB) Network drive

HMI devices	Supported logs			Supported storage locations	Supported storage locations
	Alarms	Tags	Audit Trail		
Mobile Panels[4]	Yes	Yes	Yes	a CSV file (ASCII) RDB file a TXT file (Unicode)	Storage card (MMC) Storage card (USB) Network drive
Mobile Panels 2nd Generation[5]	Yes	Yes	Yes	a CSV file (ASCII) RDB file a TXT file (Unicode)	Storage card (SD) USB memory (at USB port) Network drive
PC systems with Runtime Advanced	Yes	Yes	Yes	a CSV file (ASCII) Database RDB file a TXT file (Unicode)	Local file system Network drive

[1] KP 300, KP 400, KTP 1000, TP 1500

[2] KTP 400, KTP 700, KTP 900, KTP 1200

[3] All HMI devices from the device list

[4] Mobile Panel 277 only

[5] KTP 400F Mobile, KTP 700 Mobile, KTP 900 Mobile - including fail-safe versions

Note

Logging on network drives

Do not log alarms, tags and Audit Trail directly on a network drive. Power supply can be interrupted at any time. This means there is no guarantee for a reliable operation of logs and audit trails.

Save the logs on your local hard drive or a local storage card. Use the system function "ArchiveLogFile" to save the logs long-term on a network drive. This step ensures reliable operation.

Syntax examples for storage locations

Storage card storage location:

- <\Storage Card MMC\My_Archives\TagLogs>: Saves the archive on the MMC storage card to the subdirectory "My_Archives\TagLogs".

Local file system storage location:

- <C:\My_File_Folder\My_Archives\Machine_1>: Saves the log on the local hard disk drive C: in the subdirectory "My_File_Folder\My_Archives\Machine_1"

Network drive storage location:

- <\\ArchiveServer\My_File_Folder\My_Archives\Machine_1>: Saves the archive on the "ArchiveServer" server to the subdirectory "My_File_Folder\My_Archives\Machine_1".

Naming conventions

The log names must be unambiguous in a project. The name of a log must always be unique, regardless of whether different storage locations are selected for the log.

Note

The characters which can be used in the name of the data source depend on the storage location.

The \ / * ? : " < > | characters are not allowed at the following locations:

- File - RDB
- File - CSV (ASCII)
- File - TXT (Unicode)

If the "Database" storage location is used, the following characters may be used: a-z A-Z 0-9 _ @ # $

The characters _ @ # $ cannot be used as the first character of a name.

Note

Only applies to Basic Panels 2nd Generation

Unicode characters are not supported for the log names and the log paths.

File - CSV (ASCII)

Data is saved to a CSV file in standard ASCII format.

If you want to read or evaluate logged data without using WinCC Runtime, use the "CSV file" storage location.

Note

Double quotation marks or multiple characters are not permitted as list separators for the "CSV file" storage location. You can find the settings for list separators under "Start > Settings > Control Panel > Regional and Language Options".

Note

Logging tags of the "BOOL" data type

Boolean values are logged as digits:

- 0 (False) corresponds to the log entry 0
- 1 (True) corresponds to the log entry -1

File - TXT (Unicode)

Data is stored in Unicode.

This file format supports all characters that can be used in WinCC and WinCC Runtime. For editing, you will need software that can save files in Unicode, such as Notepad.

Note

Use "File - TXT (Unicode)" as the storage location to log Asian languages.

File - RDB

Data is saved with quick access in a proprietary database.

If you require maximum read performance in runtime, use the "RDB file" storage location.

Database

Data is saved to a database which is set up for ODBC access by the PC administrator.

Log with checksum (Audit Trail)

The following files are generated under special circumstances:

***.keep**

1. If a log is started without checksum and will be continued with a checksum.
2. If you update WinCC with a service pack or a new version and the Audit Trail or the log is continued with the checksum.

The content of the keep file will remain the same when compared with the original csv file or txt file.

***.bak**

If WinCC Runtime has determined a serious, irregular problem in the file.

Using system functions (RT Uni) 6

6.1 Working with function lists (RT Uni)

6.1.1 Basics of the function list (RT Uni)

Introduction

A function list performs one or more functions when the configured event occurs.

The following are available:

- System functions
- Functions which you configure in global modules

Principle

The function list is configured for an event of the following objects:

- Screen
- Screen object
- Task

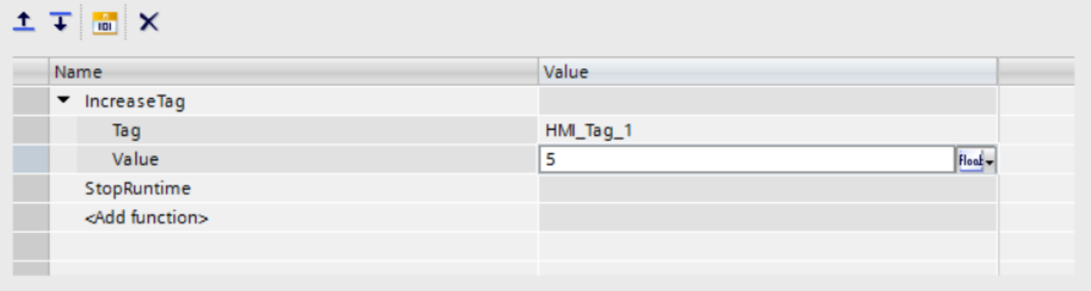
You can configure exactly one function list for each event. Which events are available depends on the selected object. Events occur only when the project is in Runtime.

Events include:

- Execution of a task
- Pressing of a button

The function list is opened in the Inspector window under "Properties > Events" as soon as an object has been selected.

Layout



Name	Value
IncreaseTag	
Tag	HMI_Tag_1
Value	5 Float
StopRuntime	
<Add function>	

The function list is divided into two columns. In the "Name" column you see the names of the functions and the corresponding parameters.

In the Values column you assign values to the parameters. You use different parameter types depending on the parameter.

The following buttons are located above the function list:

- "Move function up"
- "Move function down"
- "Convert function list to script"
- "Delete function list"

If the function list cannot be edited, the buttons are locked. This is the case, for example, with reference projects.

See also

Editing a function list (Page 305)

System functions (Page 308)

"Scripts" editor (Page 361)

6.1.2 Input support (RT Uni)

The function list supports you in:

- Function input
- Input of parameter values
- Troubleshooting

Function input

You have several options for entering a function in the function list:

- Enter the complete or partial name of the function.
  Autocomplete is supported.
- Select the "<Add function>" field and open the selection menu.
  The available functions are sorted by category.
- Select the "<Add function>" field and select the list icon.
  The available functions are displayed in alphabetical order.

The system functions provided in the function list differ depending on the object selected. For example, system functions of the "Screen" category are only available for screens and screen items.

Input of parameter values

The parameters required in functions are displayed in the function list and differ depending on the function selected.

You assign a value to each parameter.

Parameter type

The parameter type determines which values the parameter can accept.

The parameter type is either fixed by default or you can choose between several types.

Which parameter types are available depends on the respective parameter.

The following parameter types are implemented in the function list:

- Integer
- Double
- Bool
- String
- Color
- HMI tag: Specify a configured HMI tag. Autocomplete is supported.
- Screen item: Specify a configured screen item in the current screen. Autocomplete is supported.
- Selection: Sets the current screen as value. If this type is selected, the value cannot be edited.

Note

HMI tags and screen items can be renamed without updating the function list. Objects of the WinCC Unified object model are referenced in the function list.

Optional parameters

Optional parameters are marked with "(optional)".

Parameters of functions that you have configured in global modules are always optional.

Troubleshooting

Error while configuring

The project data is tested in the background during the configuration.

To inform you of errors, missing or incorrect entries in the function list are highlighted in red:

- At least one function is not completely supplied with parameters.
- At least one function is contained which is not supported by the selected HMI device, for example, by changing the device type.

Note

Missing screen items and HMI tags can be created later. Objects of the WinCC Unified object model are referenced in the function list.

Errors during compiling and loading

Alarms during the compiling and loading of a project are displayed in the Inspector window in the "Info > Compile" tab.

The function list supports you by displaying missing or incorrect entries directly for editing:

- To go directly to the function list, select the green arrow ↗.

See also

"Scripts" editor (Page 361)

System functions (Page 308)

6.1.3 Configuring a function list (RT Uni)

Note

Restriction of "activated" and "deactivated" events

If the focus is on the affected screen item, scripts are executed at the "activated" and "deactivated" events.

Requirement

One of the following objects is configured:

- Screen
- Screen item
- Task

Procedure

1. Select the object.
2. Go to "Properties > Events" in the Inspector window.
   The function list opens.
3. Select an event.
4. Select "<Add function>" in the function list.
5. Enter the function.
   If the desired function has parameters, the parameters are displayed.

6. Define the parameter values.
7. To add more functions, repeat steps 4.) to 6.).
8. Save the project.

Result

- The function list is configured.
- In addition, to the configured event, the status of the function list is displayed in the Inspector window.
- The function list is executed from top to bottom when the configured event occurs in runtime.

6.1.4 Editing a function list (RT Uni)

Requirement

- The function list is open.
- At least one function is configured at an event.

Changing the order of functions

The function list is executed from top to bottom.

You move a function within the function list as follows:

1. Select the name of the function or an associated parameter.
2. Select the ↑ "Move function up" or ↓ "Move function down" button.
   If the function is already at the first or last position in the list, pressing the button has no effect.

Replacing a function

- To replace a function, enter the name of another function in the input field.
  All parameter settings of the replaced function are deleted.

Converting a function list to a local script

- To convert the function list to a local script, use the "Convert function list to script" button.
  The "Scripts" editor opens.

The function list can be converted to a local script even if the parameter specifications are incorrect or incomplete. The parameter specifications must be adjusted accordingly in the script.

Note

This action can only be revoked using the ↶ "Undo" button.

Deleting functions

- To delete the entire function list of an event, select the ✕ "Delete function list" button.

To delete a single function, follow these steps:

1. Select the name of the function.
2. Press <Del>.

See also

"Scripts" editor (Page 361)

6.1.5 Using a screen item to specify the value of a parameter (RT Uni)

Introduction

For some parameters, you can assign "Screen item" as parameter type.

For example, enter the "Value" parameter of the "IncreaseTag" function via the I/O field in Runtime.

Note

The assigned screen item must have the property "Process value".

Possible screen items are, for example:

- I/O field
- Bar
- Slider
- Radio button

The value entered in the screen item must correspond to the data type expected by the system function.

If the data types do not match, convert the value using an additional tag.

Requirement

- A screen is configured.
- A suitable screen item (e.g. I/O field) is configured.
- A suitable system function (e.g. "IncreaseTag") is created in the function list.

Procedure

To assign a parameter to a screen item and convert the value entered in the screen item, proceed as follows:

1. Assign the desired screen item to the parameter.
2. Select the screen item.
3. Go to "Properties > General > Process value" in the Inspector window.
4. Select "Tag" in the "Dynamization" column.
   The tag selection range is displayed.
5. Select the selection button ....
6. Select the "Add" button.
7. Assign a meaningful name.
8. Specify the required tag data type.
   The "Value" parameter, for example, requires a numeric data type.

6.1.6 Adapt the function list to changed scripts (RT Uni)

You use functions that you have defined for global modules in the function list. These functions and associated parameters are referenced.

If you edit used functions after creating the function list, you must manually transfer some changes to the function list:

- Adding of a parameter
- Deleting of a parameter
- Deleting of a function

Note

If you rename functions or parameters, these changes are automatically transferred to the function list.

Requirement

- A function is defined in a global module.
- The function is used in a function list.

Procedure

1. Make at least one of the following changes to the function:
   - Add a parameter
   - Delete a parameter
   - Delete a function
2. Go to the relevant function in the function list.
3. Double-click to apply the change marked in red.

See also

"Scripts" editor (Page 361)

6.2 System functions (RT Uni)

6.2.1 ChangeConnection (RT Uni)

Description

Changes the connection parameters of an HMI connection. The following parameters can be changed: the IP address, slot and rack. Since the function is executed synchronously, the return value returns an error code that provides immediate information about the cause of the error. The error code can only be read if the function is called via a script.

Note

Change of function parameters after a function call

With the execution of the function you change the function parameters. The new connection may not be active yet at this point.

Note

Usage on devices of the S7 Plus PLC family

For devices of the S7 Plus PLC family (PLCs 15xx and 12xx) it is not possible to change the slot or the rack. The system function cannot be executed if parameters for slot or rack are not set.

Use in the function list

ChangeConnection (Connection name, IP V4 address, Slot (optional), Rack (optional))

The "SwapConnection" system function has the following parameters:

Parameter	Description
Connection name	Indicates the name of the connection.
IP V4 address	Specifies the IPv4 address. Example: 192.169.153.45
Slot (optional)	Specifies the slot number. Permitted values from 1 to 32.
Rack (optional)	Specifies the rack number. Permitted values from 0 to 7.

Use in scripts

```
HMIRuntime.Tags.SysFct.changeConnection (connectionName, address,
slot (optional), rack (optional))
```

The system function "changeConnection" has the following parameters:

Parameter	Type	Description
connectionName	HMIConnection (String)	Indicates the name of the connection.
address	STRING	Specifies the IPv4 address. Example: 192.169.153.45
slot (optional)	USINT	Specifies the slot number. Permitted values from 1 to 32.
rack (optional)	USINT	Specifies the rack number. Permitted values from 0 to 7.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.2 ChangeScreen (RT Uni)

Description

Performs a screen change to the specified screen windows or base screen.

Note

The function list is updated when the screen changes. The functions after "ChangeScreen" are therefore not executed.

Always execute "ChangeScreen" as the last function.

Use in the function list

ChangeScreen (Screen name, Screen window path)

The system function "ChangeScreen" has the following parameters:

Parameter	Description
Screen name	Name of the screen to which you change.
Screen window path	Path of the screen window or base screen that is displayed after the change has been completed. You can find additional information at "Addressing screen windows".

Use in scripts

```
HMIRuntime.UI.SysFct.ChangeScreen (screenName, screenWindowPath)
```

The system function "ChangeScreen" has the following parameters:

Parameters	Type	Description
`screenName`	HMIScreen (String)	Name of the screen to which you change.
`screenWindowPath`	HMIScreenWindow (String)	Path of the screen window or base screen that is displayed after the change has been completed. You can find additional information at "Addressing screen windows".

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.3 ClearAlarmLog (RT Uni)

Description

Deletes all recordings from the specified alarm log.

Note

No backup!

Note that no automatic backup is performed before execution of the function!

Use in the function list

ClearAlarmLog (Log name)

The system function "ClearAlarmLog" has the following parameters:

Parameter	Description
Log name	Name of the alarm log from which the entries are deleted.

Use in scripts

```
HMIRuntime.UI.SysFct.ClearAlarmLog (logName)
```

The system function "ClearAlarmLog" has the following parameters:

Parameter	Type	Description
`logName`	STRING	Name of the alarm log from which the entries are deleted.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.4 ClearTagLog (RT Uni)

Description

Deletes all data records in the specified data log.

Note

No backup!

Note that no automatic backup is performed before execution of the function!

Use in the function list

ClearLog (Log name)

The system function "ClearTagLog" has the following parameters:

Parameter	Description
Log name	Name of the data log from which all entries are deleted.

Use in scripts

```
HMIRuntime.Logging.SysFct.ClearTagLog(logName)
```

The system function "ClearTagLog" has the following parameters:

Parameter	Type	Description
`logName`	STRING	Name of the data log from which all entries are deleted.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.5 ClosePopup (RT Uni)

Description

Closes a popup window dynamically during runtime.

Use in the function list

ClosePopup (popup window path)

The system function "ClosePopup" has the following parameters:

Parameter	Description
Popup window path	Specifies the path to the popup window to be closed.

Use in scripts

```
HMIRuntime.Tags.SysFct.ClosePopup (popupWindowpath)
```

The system function "ClosePopup" has the following parameters:

Parameter	Type	Description
popupWindowPath	HMIPopupScreenWindow (string)	Specifies the path to the popup window to be closed.

Can be used if the configured device supports user functions. For additional information, refer to "Device dependency".

6.2.6 CreateScreenshot (RT Uni)

Description

Creates and saves a screenshot. The .jpg and .jpeg image formats are supported. If images already exist in the specified file path, they will be overwritten. If the specified file path cannot be accessed, an error message is displayed.

Note

The system function "CreateScreenshot" is only available for WinCC Unified Comfort Panel. If you configure it in the engineering system for WinCC Unified SCADA, this does not have an effect in Runtime. For this reason, a warning is displayed when the software is compiled for WinCC Unified SCADA.

Use in the function list

CreateScreenshot (path of storage medium)

The system function "CreateScreenshot" has the following parameters:

Parameter	Description
Storage medium path	Path name of the screenshot.

Use in user functions

```
HMIRuntime.UI.SysFct.CreateScreenshot(storagePath)
```

The system function "CreateScreenshot" has the following parameters:

Parameter	Type	Description
`storagePath`	STRING (constant, string tag)	Path name of the screenshot.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.7 CreateSystemInformation (RT Uni)

Description

Creates an alarm of the class "SystemInformation" in the alarm log.

The alarm class "SystemInformation" has the status "Raise only" and is therefore only displayed in the archive list and not in the online list of alarm monitoring.

Use in the function list

GenerateSystemInformation (alarm text, area (optional), Parameter value 1 (optional), Parameter value 2 (optional), Parameter value 3 (optional), Parameter value 4 (optional), Parameter value 5 (optional), Parameter value 6 (optional), Parameter value 7 (optional), Parameter value 8 (optional))

The system function "CreateSystemInformation" has the following parameters:

Parameters	Description
Alarm text	Specifies the alarm text.
Area (optional)	Specifies the scope of the alarm.
Parameter value 1 (optional)	Value of the first alarm parameter.
Parameter value 2 (optional)	Value of the second alarm parameter.
Parameter value 3 (optional)	Value of the third alarm parameter.
Parameter value 4 (optional)	Value of the fourth alarm parameter.
Parameter value 5 (optional)	Value of the fifth alarm parameter.
Parameter value 6 (optional)	Value of the sixth alarm parameter.
Parameter value 7 (optional)	Value of the seventh alarm parameter.
Parameter value 8 (optional)	Value of the eighth alarm parameter.

Use in scripts

```
HMIRuntime.Resources.SysFct.CreateSystemInformation(alarmText,
area, alarmParameterValue1, alarmParameterValue2,
alarmParameterValue3, alarmParameterValue4, alarmParameterValue5,
alarmParameterValue6, alarmParameterValue7, alarmParameterValue8)
```

The system function "CreateSystemInformation" has the following parameters:

Parameter	Type	Description
`alarmText`	STRING	Specifies the alarm text.
`area (optional)`	STRING	Specifies the scope of the alarm.

Parameter	Type	Description
`alarmParameterValue1 (optional)`	SCALAR (Variant)	Value of the first alarm parameter.
`alarmParameterValue2 (optional)`	SCALAR (Variant)	Value of the second alarm parameter.
`alarmParameterValue3 (optional)`	SCALAR (Variant)	Value of the third alarm parameter.
`alarmParameterValue4 (optional)`	SCALAR (Variant)	Value of the fourth alarm parameter.
`alarmParameterValue5 (optional)`	SCALAR (Variant)	Value of the fifth alarm parameter.
`alarmParameterValue6 (optional)`	SCALAR (Variant)	Value of the sixth alarm parameter.
`alarmParameterValue7 (optional)`	SCALAR (Variant)	Value of the seventh alarm parameter.
`alarmParameterValue8 (optional)`	SCALAR (Variant)	Value of the eighth alarm parameter.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.8 DecreaseTag (RT Uni)

Description

Subtracts the given value from the tag value: X = X - a

If you configure the system function for events of an alarm without the tag being used in the current screen, it is not ensured that the actual tag value is being used in the PLC.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Use in the function list

DecreaseTag (Tag, Value)

The system function "DecreaseTag" has the following parameters:

Parameter	Description
Tag	Tag from which the specified value is subtracted.
Value	Value to be subtracted.

Use in scripts

```
HMIRuntime.Tags.SysFct.DecreaseTag (tag, value)
```

The system function "DecreaseTag" has the following parameters:

Parameter	Type	Description
`tag`	HMITag (String)	Tag from which the specified value is subtracted.
`value`	VARIANT (Float)	Value to be subtracted.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

Converting a value

The system function uses the same tag as input and output values. If you are using this system function to convert a value, follow these steps:

Note

1. Create an auxiliary tag.
2. Assign the tag value to the auxiliary tag with the "SetTagValue" system function.

6.2.9 EjectStorageMedium (RT Uni)

Description

Ejects an inserted storage medium. The function checks whether the storage medium is currently being accessed. If no current read or write process is taking place, the storage medium can be removed without data loss.

Use in the function list

EjectStorageMedium (storage device)

The system function "EjectStorageMedium" has the following parameters:

Parameter	Description
Memory device	System variable name of the storage medium with path specification (e.g. / USB storage card)

Use in scripts

```
HMIRuntime.Device.SysFct.EjectStorageMedium(storageTag)
```

The system function "EjectStorageMedium" has the following parameters:

Parameter	Type	Description
`storageTag`	STRING (constant, string tag)	System variable name of the storage medium with path specification (e.g. /USB storage card)

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.10 ExecuteReport (RT Uni)

Description

Executes a report automatically and independently of the general report cycle. The execution can be triggered through a specific event, for example, change of a tag value, occurrence of a specific message or exceeding a tag limiting value.

Use in the function list

ExecuteReport (name report order)

The system function "ExecuteReport" has the following parameters:

Parameter	Description
Name report order	Specifies the name of the report to be executed.

Use in scripts

```
HMIRuntime.Tags.SysFct.ExecuteReport (reportTaskName)
```

The system function "ExecuteReport" has the following parameters:

Parameter	Type	Description
`reportTaskName`	STRING	Specifies the name of the report to be executed.

Can be used if the configured device supports user functions. For additional information, refer to "Device dependency".

6.2.11 ExportParameterSets (RT Uni)

Description

Allows the export of a parameter set to a file.

Use in the function list

ExportParameterSets (parameter set type ID, parameter set ID, file name, overwrite, output status, processing status (optional), generate checksum)

The system function "ExportParameterSets" has the following parameters:

Parameters	Description
Parameter set type ID	Specifies the name or the ID of the parameter set type. If the name or ID of the type does not exist, execution is terminated.
Parameter set ID	Specifies the name or the ID of the parameter set. The following cases are differentiated: ● If the parameter set ID is set to 0, all parameter sets available in the memory are exported. ● If the specified name or the ID does not exist in the imported file, the execution is cancelled. ● If the specified name or the ID does not exist in the imported file, this specific parameter set is imported. In the following cases the import is aborted and an alarm appears: ● No parameter set available ● Name or ID does not exist in the import file
File name	Specifies the file path of the file to be imported. In the following cases the execution is canceled and an alarm is generated: ● Invalid file path ● Error during file access
Overwrite	Specifies whether the existing file is overwritten: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. An alarm is generated and displayed if an error occurs during file access. This can occur, for example, when the existing file cannot be overwritten even though overwriting is allowed.
Output status	Specifies the output status: True = Alarms are output. False = Alarms are not output.
Processing status (optional)	Indicates the execution status of a function: 2 = Function just executed 4 = Function successfully executed 12 = Function was cancelled
Generate checksum	Specifies whether a checksum is generated for the parameter set to be exported: True = Checksum is generated. False = Checksum is not generated.

Use in scripts

```
HMIRuntime.Tags.SysFct.exportParameterSets (parameterSetTypeID,
parameterSetID, fileName, overwrite, outputStatus, processingStatus
(optional), generateChecksum)
```

The system function "ExportParameterSets" has the following parameters:

Parameter	Type	Description
`parameterSetTypeID`	STRING (constant, string tag) UDINT (constant, Int tag)	Specifies the name or the ID of the parameter set type. If the name or ID of the type does not exist, execution is terminated.
`parameterSetID`	STRING (constant, string tag) UDINT (constant, Int tag)	Specifies the name or the ID of the parameter set. The following cases are differentiated: • If the parameter set ID is set to 0, all parameter sets available in the memory are exported. • If the specified name or the ID does not exist in the imported file, the execution is cancelled. • If the specified name or the ID does not exist in the imported file, this specific parameter set is imported. In the following cases the import is aborted and an alarm appears: • No parameter set available • Name or ID does not exist in the import file
`fileName`	STRING (constant, string tag)	Specifies the file path of the file to be imported. In the following cases the execution is canceled and an alarm is generated: • Invalid file path • Error during file access
`overwrite`	USINT (constant, Int tag)	Specifies whether the existing file is overwritten: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. An alarm is generated and displayed if an error occurs during file access. This can occur, for example, when the existing file cannot be overwritten even though overwriting is allowed.
`outputStatus`	BOOL (constant, Bool tag)	Specifies the output status: True = Alarms are output. False = Alarms are not output.
`processingStatus (optional)`	Day	Indicates the execution status of a function: 2 = Function just executed 4 = Function successfully executed 12 = Function was cancelled
`generateChecksum`	BOOL	Specifies whether a checksum is generated for the parameter set to be exported: True = Checksum is generated. False = Checksum is not generated.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.12 GetBrightness (RT Uni)

Description

Reads out the brightness value.

Note

The system function "GetBrightness" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Use in the function list

GetBrightness (Value)

The system function "GetBrightness" has the following parameters:

Parameter	Description
Value	The tag to which the brightness value is written.

Use in scripts

```
HMIRuntime.Device.SysFct.GetBrightness (value)
```

The system function "GetBrightness" has the following parameters:

Parameter	Type	Description
`value`	SetCommand (scalar tag)	The tag to which the brightness value is written.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.13 GetDHCPState (RT Uni)

Description

Reads out the DHCP setting of the network adapter.

Note

The system function "GetDHCPState" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Use in the function list

GetDHCPState (Adapter name, Status, IPV6 (optional))

The system function "GetDHCPState" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: ● X1 = Static network adapter name 1 ● X2 = Static network adapter name 2 ● Manual input
State	Tag to which the status is written: 0 = DHCP is disabled. 1 = DHCP is enabled.
IP V6 (optional)	Tag to which the IPv6 address is written.

Use in scripts

```
HMIRuntime.Device.SysFct.GetDHCPState(adapterName, mode, IP V6
(optional))
```

The system function "GetDHCPState" has the following parameters:

Parameter	Type	Description
`adapterName`	STRING (constant, string tag)	Specifies the name of the network adapter. The following entries are possible: ● X1 = Static network adapter name 1 ● X2 = Static network adapter name 2 ● Manual input
`mode`	SetCommand (scalar tag)	Tag to which the status is written: 0 = DHCP is disabled. 1 = DHCP is enabled.
`IP V6 (optional)`	STRING (string tag)	Tag to which the IPv6 address is written.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.14 GetIPV4Address (RT Uni)

Description

Reads out the IPv4 settings of the network adapter.

Note

The system function "GetIPV4Address" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Use in the function list

GetIPV4Address (Adapter name, IP address, Subnet mask, Default gateway (optional), DNS server 1 (optional), DNS server 2 (optional))

The system function "GetIPV4Address" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
IP address	Tag to which the IP address is written.
Subnet mask	Tag to which the subnet mask of the IPv4 address is written.
Default gateway (optional)	Tag to which the IP address of the default gateway is written.
DNS server 1 (optional)	Tag to which the IP address of DNS server 1 is written.
DNS server 2 (optional)	Tag to which the IP address of DNS server 2 is written.

Use in scripts

```
HMIRuntime.Device.SysFct.GetIPV4Address(adapterName, ipAddress,
subnetMask, defaultGatewayIP, dnsIP1, dnsIP2)
```

The system function "GetIPV4Address" has the following parameters:

Parameter	Type	Description
`adapterName`	STRING (constant, string tag)	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
`ipAddress`	STRING (string tag)	Tag to which the IP address is written.
`subnetMask`	STRING (string tag)	Tag to which the subnet mask of the IPv4 address is written.

Parameter	Type	Description
`defaultGatewayIP` (optional)	STRING (string tag)	Tag to which the IP address of the default gateway is written.
`dnsIP1` (optional)	STRING (string tag)	Tag to which the IP address of DNS server 1 is written.
`dnsIP2` (optional)	STRING (string tag)	Tag to which the IP address of DNS server 2 is written.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.15 GetNetworkInterfaceState (RT Uni)

Description

Reads out the status of the network adapter.

Note

The system function "GetNetworkInterfaceStatus" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Use in the function list

GetNetworkInterfaceStatus (Adapter name, Status)

The system function "GetNetworkInterfaceState" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
State	Tag to which the state of the network adapter is written: 0 = Network adapter is disabled. 1 = Network adapter is enabled.

Use in scripts

```
HMIRuntime.Device.SysFct.GetNetworkInterfaceState(adapterName,
state, systemName)
```

The system function "GetNetworkInterfaceState" has the following parameters:

Parameter	Type	Description
`adapterName`	STRING (constant, string tag)	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
`state`	SetCommand (scalar tag)	Tag to which the state of the network adapter is written: 0 = Network adapter is disabled. 1 = Network adapter is enabled.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.16 GetSmartServerState (RT Uni)

Description

Returns the activation state of the Smart server.

Use in the function list

GetSmartServerState (Status)

The system function "GetSmartServerState" has the following parameters:

Parameter	Description

Status	Tag to which the activation status of the Smart Server is written:
	True = Smart Server is activated.
	False = Smart Server is deactivated.

Use in scripts

```
HMIRuntime.Device.SysFct.GetSmartServerState(state)
```

The system function "GetSmartServerState" has the following parameters:

Parameter	Type	Description

state	SetCommand (scalar tag)	Tag to which the activation status of the Smart Server is written:
		True = Smart Server is activated.
		False = Smart Server is deactivated.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.17 ImportParameterSets (RT Uni)

Description

Imports existing and exported parameter set data.

Use in the function list

ImportParameterSets (file name, parameter set ID, overwrite, output status, processing status (optional), check checksum)

The system function "ImportParameterSets" has the following parameters:

Parameters	Description
File name	Specifies the file path of the file to be imported. In the following cases the execution is canceled and an alarm is generated: ● Invalid file path ● Error during file access
Parameter set ID	Specifies the name or the ID of the parameter set. The following cases are differentiated: ● If the parameter set ID is set to 0, all parameter sets are imported from the file. ● If the name or the ID does not exist in the imported file, the execution is cancelled. ● If the specified name or the ID does not exist in the imported file, only this specific parameter set is imported. In the following cases the import is aborted and an alarm appears: ● Invalid file head. ● No parameter set available. ● Parameter set name or parameter set ID does not exist in the file.
Overwrite	Specifies whether the values in the memory are overwritten with the values from the import file: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. If the name / ID of the specified parameter set exists, the values in the memory are overwritten with the parameter set values from the import file if overwriting is allowed. If it may not be overwritten, the data in the memory is not renewed.
Output status	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).

Parameters	Description
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 = Function was cancelled.
Check checksum	Specifies whether the checksum of the import file is verified: True = Checksum is generated. False = Checksum is not generated.

Use in scripts

```
ImportParameterSets (fileName, parameterSetID, overwrite,
outputStatus, processingStatus (optional), verifyChecksum)
```

The system function "ImportParameterSets" has the following parameters:

Parameter	Type	Description
`fileName`	STRING (constant, string tag)	Specifies the file path of the file to be imported. In the following cases the execution is canceled and an alarm is generated: • Invalid file path • Error during file access
`parameterSetID`	STRING (constant, string tag) UDINT (constant, Int tag)	Specifies the name or the ID of the parameter set. The following cases are differentiated: • If the parameter set ID is set to 0, all parameter sets are imported from the file. • If the name or the ID does not exist in the imported file, the execution is cancelled. • If the specified name or the ID does not exist in the imported file, only this specific parameter set is imported. In the following cases the import is aborted and an alarm appears: • Invalid file head. • No parameter set available. • Parameter set name or parameter set ID does not exist in the file.
`overwrite`	USINT (constant, Int tag)	Specifies whether the values in the memory are overwritten with the values from the import file: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. If the name / ID of the specified parameter set exists, the values in the memory are overwritten with the parameter set values from the import file if overwriting is allowed. If it may not be overwritten, the data in the memory is not renewed.

Parameter	Type	Description
`outputStatus`	BOOL (constant, Bool tag)	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
`processingStatus (optional)`	Day	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 = Function was cancelled.
`verifyChecksum`	BOOL (constant, Bool tag)	Specifies whether the checksum of the import file is verified: True = Checksum is generated. False = Checksum is not generated.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.18 IncreaseTag (RT Uni)

Description

Adds the specified value to the tag value: X = X + a

If you configure the system function for events of an alarm without the tag being used in the current screen, it is not ensured that the actual tag value is being used in the PLC.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tag is "Cyclic continuous" and
- the tag is used by an object, e.g. an I/O field.

Use in the function list

IncreaseTag (Tag, Value)

The system function "IncreaseTag" has the following parameters:

Parameter	Description
Tag	Tag to which the specified value is added.
Value	Value to be added.

Use in scripts

```
HMIRuntime.Tags.SysFct.IncreaseTag (tag, value)
```

The system function "IncreaseTag" has the following parameters:

Parameter	Type	Description
`tag`	HMITag (String)	Tag to which the specified value is added.
`value`	Variant (Float)	Value to be added.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

Converting a value

The system function uses the same tag as input and output values. If you are using this system function to convert a value, follow these steps:

Note

1. Create an auxiliary tag.
2. Assign the tag value to the auxiliary tag with the "SetTagValue" system function.

6.2.19 InvertBitInTag (RT Uni)

Description

Inverts the bit with the specified number in the tag:

- If the bit in the tag has the value of 1 (TRUE), it will be set to 0 (FALSE).
- If the bit in the tag has the value of 0 (FALSE), it will be set to 1 (TRUE).

After changing the given bit, the system function transfers the entire tag back to the PLC. It is not checked whether other bits in the tags have changed in the meantime.

While the function is running, the operator and controller have only read access to the specified tag until it is transferred to the controller again.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Use in the function list

InvertBitInTag (Tag, Value)

The system function "InvertBitInTag" has the following parameters:

Parameter	Description
Tag	The tag in which the specified bit is inverted.
Value	The number of the bit that is inverted. When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. Numbering starts with 0.

Use in scripts

```
HMIRuntime.Tags.SysFct.InvertBitInTag (tag, value)
```

The system function "InvertBitInTag" has the following parameters:

Parameters	Type	Description
`tag`	HMITag (String)	The tag in which the specified bit is inverted.
`value`	USINT	The number of the bit that is inverted. When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. Numbering starts with 0.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.20 LoadAndWriteParameterSet (RT Uni)

Description

Loads the parameter set values from the memory and writes them to the PLC.

Use in the function list

LoadAndWriteParameterSet (parameter set type ID, parameter set ID, output status, processing status (optional))

The system function "LoadAndWriteParameterSet" has the following parameters:

Parameter	Description
Parameter set type ID	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set ID	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, execution is terminated.
Output status	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 - Function was cancelled.

Use in scripts

```
LoadAndWriteParameterSet (parameterSetTypeID, parameterSetID,
outputStatus, processingStatus (optional))
```

The system function "LoadAndWriteParameterSet" has the following parameters:

Parameter	Type	Description
`parameterSetTypeID`	STRING (constant, string tag) UDINT (constant, Int tag)	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
`parameterSetID`	STRING (constant, string tag) UDINT (constant, Int tag)	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, execution is terminated.
`outputStatus`	BOOL (constant, Bool tag)	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
`processingStatus (optional)`	Day	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 - Function was cancelled.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.21 Logoff (RT Uni)

Description

Logs off the current user on the HMI device.

Use in the function list

Logoff()

The system function "Logoff" has no parameters.

Use in scripts

```
HMIRuntime.Device.SysFct.Logoff()
```

The system function "Logoff" has no parameters.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.22 LookupText (RT Uni)

Description

Identifies a list entry from a text list. The result depends on the value of the list entry and on the selected runtime language. The result is saved to a tag of data type "String".

Use in the function list

CallText (output text, index, LCID, text list name)

The system function "LookUpText" has the following parameters:

Parameter	Description
Output text	The tag to which the result is written.
Index	The tag that defines the value of the list entry.
LCID	LCID of the language set on the HMI device. Specify the language ID, e.g. 0x0007 for German - Standard, 0x0409 for English - USA. You can find an overview of all languages under: "https://msdn.microsoft.com/en-us/library/cc233982.aspx".
Text list name	Defines the text list. The list entry is read from the text list.

Use in scripts

```
HMIRuntime.Resources.SysFct.LookUpText(outputText, index, lcid,
textListName)
```

The system function "LookUpText" has the following parameters:

Parameters	Type	Description
`outputText`	HMISetValueCommandBase (Object)	The tag to which the result is written.
`index`	Variant (UInt32)	The tag that defines the value of the list entry.
`lcid`	HMILCID (UInt32)	LCID of the language set on the HMI device. Specify the language ID, e.g. 0x0007 for German - Standard, 0x0409 for English - USA. You can find an overview of all languages under: "https://msdn.microsoft.com/en-us/library/cc233982.aspx".
`textListName`	HMIResourcedList (String)	Defines the text list. The list entry is read from the text list.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.23 OpenFileBrowser (RT Uni)

Description

Opens the file explorer of the device.

Use in the function list

OpenFileBrowser

The system function "OpenFileBrowser" has the following parameters:

Parameter	Description
Tag	Tag to which the specified value is assigned.
Value	Value assigned to the specified tag.
Parameter (optional)	That is an example.

Use in scripts

```
HMIRuntime.UI.SysFct.OpenFileBrowser()
```

The system function "OpenFileBrowser" has the following parameters:

Parameter	Type	Description
`tag`	HMITag (String)	Tag to which the specified value is assigned.
`value`	Variant	Value assigned to the specified tag.
Parameter (optional)	XX	That is an example.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.24 OpenScreenInPopup (RT Uni)

Description

Opens a popup window dynamically during runtime.

Use in the function list

OpenScreenInPopup (Popup window name, Screen name, Toggle Open, Header, Left, Top, Hide close button, Parent screen path (optional))

The system function "OpenScreenInPopup" has the following parameters:

Parameter	Description
Popup window name	Name of the popup window. The name must be unique within the ParentScreen area and is required for closing the popup window.
Screen name	Name of the screen that is to be loaded into the popup window.
Toggle open	True = If the window is opened during the system call, it is closed. False = If the window is opened during the system call, it remains open.
Header	Specifies the window title of the popup window.
Left	Specifies the window position as offset from the left-hand margin.
Top	Specifies the window position as offset from the top margin.
Hide close button	True = The close button is not displayed. False = The close button is displayed.
Parent screen path (optional)	Path of the parent HMI screen. If this value remains empty, the popup window is global.

Use in scripts

```
HMIRuntime.Tags.SysFct.OpenScreenInPopup (popupWindowName,
screenName, toggleOpen, Caption, Left, Top, hideCloseButton,
parentScreenPath (optional))
```

The system function "OpenScreenInPopup" has the following parameters:

Parameter	Type	Description
popupWindowName	HMIPopupScreenWindow (string)	Name of the popup window. The name must be unique within the ParentScreen area and is required for closing the popup window.
screenName	HMIScreen (string)	Name of the screen that is to be loaded into the popup window.
toggleOpen	Bool	True = If the window is opened during the system call, it is closed. False = If the window is opened during the system call, it remains open.
Caption	String	Specifies the window title of the popup window.
Left	Int32	Specifies the window position as offset from the left-hand margin.
Top	Int32	Specifies the window position as offset from the top margin.

Parameter	Type	Description
hideCloseButton	Bool	True = The close button is not displayed. False = The close button is displayed.
parentScreenPath (optional)	HMIScreen (string)	Path of the parent HMI screen. If this value remains empty, the popup window is global.

Can be used if the configured device supports user functions. For additional information, refer to "Device dependency".

6.2.25 ReadAndSaveParameterSet (RT Uni)

Description

Reads a parameter set from the PLC and writes the parameter set to the local memory.

Use in the function list

ReadAndWriteParameterSet (parameter set type ID, parameter set ID, overwrite, output status, processing status (optional))

The system function "ReadAndWriteParameterSet" has the following parameters:

Parameter	Description
Parameter set type ID	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set ID	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, a new parameter set is created. If the name or the ID of the specified parameter set exists, the values in the PLC are overwritten with the parameter set values in the memory if hmiOverwrite.Enabled is set. If hmiOverwrite.Disabled is set, the data in the memory are not replaced.
Overwrite	Specifies whether the values in the memory are overwritten with the values from the import file: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. The following cases are differentiated: • If the name / ID of the specified parameter set exists, the values in the PLC are overwritten with the parameter set values in the memory if overwriting is allowed. • If overwriting is not allowed, the data in the memory is not replaced. The process tag is updated to the state of the system function, if configured accordingly.

Parameter	Description
Output status	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function was successfully executed. 12 = Function cancelled.

Use in scripts

ReadAndSaveParameterSet (parameterSetTypeID, parameterSetID, overwrite, outputStatus, processingStatus (optional))

The system function "ReadAndSaveParameterSet" has the following parameters:

Parameters	Type	Description
`parameterSetTypeID`	STRING (constant, string tag) UDINT (constant, Int tag)	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
`parameterSetID`	STRING (constant, string tag) UDINT (constant, Int tag)	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, a new parameter set is created. If the name or the ID of the specified parameter set exists, the values in the PLC are overwritten with the parameter set values in the memory if hmiOverwrite.Enabled is set. If hmiOverwrite.Disabled is set, the data in the memory are not replaced.
`overwrite`	USINT (constant, Int tag)	Specifies whether the values in the memory are overwritten with the values from the import file: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. The following cases are differentiated: ● If the name / ID of the specified parameter set exists, the values in the PLC are overwritten with the parameter set values in the memory if overwriting is allowed. ● If overwriting is not allowed, the data in the memory is not replaced. The process tag is updated to the state of the system function, if configured accordingly.

Parameters	Type	Description
`outputStatus`	BOOL (constant, Bool tag)	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
`processingStatus (optional)`	Day	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function was successfully executed. 12 = Function cancelled.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.26 ResetBitInTag (RT Uni)

Description

Sets the bit with the specified number to 0 (FALSE) in the tag.

After changing the given bit, the system function transfers the entire tag back to the PLC. It is not checked whether other bits in the tags have changed in the meantime. Operator and PLC have read-only access to the indicated tag until it is transferred back to the PLC.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Use in the function list

ResetBitInTag (Tag, Value)

The system function "ResetBitInTag" has the following parameters:

Parameter	Description
Tag	Tag in which a bit is set to 0 (FALSE).
Value	The number of the bit that is set to 0 (FALSE). When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. Numbering starts with 0.

Use in scripts

```
HMIRuntime.Tags.SysFct.ResetBitInTag (tag, value)
```

The system function "ResetBitInTag" has the following parameters:

Parameters	Type	Description
`tag`	HMITag (String)	Tag in which a bit is set to 0 (FALSE).
`value`	Variant (USINT)	The number of the bit that is set to 0 (FALSE). When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. Numbering starts with 0.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.27 SetBitInTag (RT Uni)

Description

Sets the bit with the specified number to 1 (TRUE) in the tag.

After changing the given bit, the system function transfers the entire tag back to the PLC. It is not checked whether other bits in the tags have changed in the meantime. Operator and PLC have read-only access to the indicated tag until it is transferred back to the PLC.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Use in the function list

SetBitInTag (Tag, Value)

The "SetBitInTag" system function has the following parameters:

Parameter	Description
Tag	The tag in which a bit is set to 1 (TRUE).
Value	The number of the bit that is set to 1 (TRUE). When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. Numbering starts with 0.

Use in scripts

```
HMIRuntime.Tags.SysFct.SetBitInTag (tag, value)
```

The system function "SetBitInTag" has the following parameters:

Parameters	Type	Description
`tag`	HMITag (String)	The tag in which a bit is set to 1 (TRUE).
`value`	Variant (USINT)	The number of the bit that is set to 1 (TRUE). When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. The numbering starts with 0.d.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.28 SetBrightness (RT Uni)

Description

Assigns a new value to the brightness of the display.

The value for the system function "SetBrightness" can be set between 0% and 100%. The set value is transferred to the HMI device. The brightness settings on the HMI device can be viewed and edited in "Start Center > Settings > Display". The HMI devices support a brightness setting between 10% and 100%.

If the system function "SetBrightness" is assigned a value of 0%, the display of the HMI device is switched off by default in Runtime. If the operator touches the display, the display switches to the previous brightness setting.

If the system function "SetBrightness" is assigned a value between 1% and 10% and the operator opens the display settings in the Start Center, brightness is reset to 10%.

Note

The system function "SetBrightness" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Note

The configuration that is set in the Control Panel / Start Center will be reestablished when you restart the HMI device.

Use in the function list

SetBrightness (value)

The system function "SetBrightness" has the following parameters:

Parameter	Description
Value	The new value for the brightness of the display.

Use in scripts

```
HMIRuntime.Device.SysFct.SetBrightness (value)
```

The system function "SetBrightness" has the following parameters:

Parameter	Type	Description
`value`	USINT (constant, scalar tag)	The new value for the brightness of the display.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.29 SetConnectionMode (RT Uni)

Description

The specified connection is established or disconnected.

Use in the function list

SetConnectionMode (Connection name, Activated)

The system function "SetConnectionMode" has the following parameters:

Parameter	Description
Connection name	The PLC to which the HMI device is connected. You specify the name of the PLC in the connection editor.
Enabled	0 = Offline: Connection is terminated. 1 = Online: Connection is established.

Use in scripts

```
HMIRuntime.Connections.SysFct.SetConnectionMode (connectionName,
enableState)
```

The system function "SetConnectionMode" has the following parameters:

Parameter	Type	Description
`connectionName`	STRING	The PLC to which the HMI device is connected. You specify the name of the PLC in the connection editor.
`enableState`	BOOL	1 = Online: Connection is terminated. 0 = Offline: Connection is established.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.30 SetDHCPState (RT Uni)

Description

Changes the DHCP setting of the network adapter.

Note

The system function "SetDHCPStatus" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Use in the function list

SetDHCPState (Adapter name, Enabled, IP V6 (optional))

The system function "SetDHCPState" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: ● X1 = Static network adapter name 1 ● X2 = Static network adapter name 2 ● Manual input
Enabled	Defines the DHCP setting of the network adapter: 0 = DHCP is disabled. 1 = DHCP is enabled.
IP V6 (optional)	Tag to which the IPv6 address is written.

Use in scripts

```
HMIRuntime.Device.SysFct.SetDHCPState(adapterName, mode, IP V6
(optional))
```

The system function "SetDHCPState" has the following parameters:

Parameter	Type	Description
`adapterName`	STRING (constant, string tag)	Specifies the name of the network adapter. The following entries are possible: ● X1 = Static network adapter name 1 ● X2 = Static network adapter name 2 ● Manual input
`mode`	BOOL (constant, scalar tag)	Defines the DHCP setting of the network adapter: 0 = DHCP is disabled. 1 = DHCP is enabled.
`IP V6 (optional)`	STRING (string tag)	Tag to which the IPv6 address is written.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.31 SetIPV4Address (RT Uni)

Description

Changes the IPv4 settings of the network adapter.

Note

The system function "SetIPV4Address" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Use in the function list

SetIPV4Address (Name adapter, IP address, Subnet mask, Standard gateway (optional), DNS Server 1 (optional), DNS Server 2 (optional))

The system function "SetIPV4Address" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
IP address	Specifies the IP address.
Subnet mask	Subnet mask of the IPv4 network.
Default gateway (optional)	IP address of the default gateway.
DNS server 1 (optional)	IP address of DNS server 1.
DNS server 2 (optional)	IP address of DNS server 2.

Use in scripts

```
HMIRuntime.Device.SysFct.SetIPV4Address(adapterName, ipAddress,
subnetMask, defaultGatewayIP, dnsIP1, dnsIP2)
```

The system function "SetIPV4Address" has the following parameters:

Parameter	Type	Description
`adapterName`	STRING (constant, string tag)	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
`ipAddress`	STRING (constant, string tag)	Specifies the IP address.
`subnetMask`	STRING (constant, string tag)	Subnet mask of the IPv4 network.
`defaultGatewayIP` (optional)	STRING (constant, string tag)	IP address of the default gateway.
`dnsIP1` (optional)	STRING (constant, string tag or empty)	IP address of DNS server 1.
`dnsIP2` (optional)	STRING (constant, string tag or empty)	IP address of DNS server 2.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.32 SetLanguage (RT Uni)

Description

Toggles the language on the HMI device. All configured texts and system events are displayed on the HMI device in the newly set language.

Use in the function list

SetLanguage (LCID)

The system function "SetLanguage" has the following parameters:

Parameter	Description
LCID	LCID of the language set on the HMI device. Specify the language ID, e.g. 0x0007 for German - Standard, 0x0409 for English - USA. You can find an overview of all languages under: "https://msdn.microsoft.com/en-us/library/cc233982.aspx".

Use in scripts

```
HMIRuntime.UI.SysFct.SetLanguage(lcid)
```

The system function "SetLanguage" has the following parameters:

Parameters	Type	Description
`lcid`	HMILCID (UInt32)	LCID of the language set on the HMI device. Specify the language ID, e.g. 0x0007 for German - Standard, 0x0409 for English - USA. You can find an overview of all languages under: "https://msdn.microsoft.com/en-us/library/cc233982.aspx".

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.33 SetNetworkInterfaceState (RT Uni)

Description

Changes the state of the network adapter.

Note

The system function "SetNetworkInterfaceStatus" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Use in the function list

SetNetworkInterfaceState (Adapter name, Enabled)

The system function "SetNetworkInterfaceState" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
Enabled	Specifies the state of the network adapter: 0 = Network adapter is disabled. 1 = Network adapter is enabled.

Use in scripts

```
HMIRuntime.Device.SysFct.SetNetworkInterfaceState(adapterName,
state)
```

The system function "SetNetworkInterfaceState" has the following parameters:

Parameter	Type	Description
`adapterName`	STRING (constant, string tag)	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
`state`	BOOL (constant, scalar tag)	Specifies the status of the network adapter: 0 = Network adapter is disabled. 1 = Network adapter is enabled.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.34 SetPropertyValue (RT Uni)

Description

Assigns a new value to the specified property of the screen item.

Note

Depending on the type of the object property, you can use this system function to assign strings and numbers.

Use in the function list

SetPropertyValue (Screen item path, Screen item property name, Value)

The system function "SetPropertyValue" has the following parameters:

Parameter	Description
Screen item path	Path of the screen item whose property is changed.
Screen item property name	Property name that will be changed. The property name can be copied with selected screen item to the Inspector window under "Properties > Properties": 1. Right-click the name of the property. 2. Select "Copy property name".
Value	The value assigned to the property.

Use in scripts

```
HMIRuntime.UI.SysFct.SetPropertyValue (screenItemPath,
screenItemPropertyName, value)
```

The system function "SetPropertyValue" has the following parameters:

Parameters	Type	Description
`screenItemPath`	HMIScreenItemBase (String)	Path of the screen item whose property is changed.
`screenItemPropertyName`	STRING	Name of the property that will be changed. The property name can be copied with selected screen item to the Inspector window under "Properties > Properties": 1. Right-click the name of the property. 2. Select "Copy property name".
`value`	VARIANT	The value assigned to the property.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.35 SetSmartServerState (RT Uni)

Description

Allows you to enable or disable the smart server.

Use in the function list

SetSmartServerState (smartServerState)

The system function "SetSmartServerState" has the following parameters:

Parameter	Description

Status	Status to which the Smart Server is to be set:
	True = Smart Server is activated.
	False = Smart Server is deactivated.

Use in scripts

```
HMIRuntime.SysFct.SetSmartServerStart(smartServerState)
```

The system function "SetSmartServerStart" has the following parameters:

Parameter	Type	Description

`smartServerState`	BOOL (constant, scalar tag)	Status to which the Smart Server is to be set:
		True = Smart Server is activated.
		False = Smart Server is deactivated.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.36 SetTagValue (RT Uni)

Description

Assigns the specified tag a new value.

Depending on the tag type, you use this system function to assign strings and numbers.

Note

The "SetTagValue" system function is only executed after a connection has been established.

Use in the function list

SetTagValue (Tag, Value)

The system function "SetTagValue" has the following parameters:

Parameters	Description
Tag	Tag to which the specified value is assigned.
Value	The value that is assigned to the specified variable.

Use in scripts

```
HMIRuntime.Tags.SysFct.SetTagValue (tag, value)
```

The system function "SetTagValue" has the following parameters:

Parameters	Type	Description
`tag`	HMITag (String)	Tag to which the specified value is assigned.
`value`	VARIANT	The value that is assigned to the specified variable.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

See also

"Tag" object (Page 668)

6.2.37 ShiftAndMask (RT Uni)

Description

The system function converts the input bit pattern of the source into an output bit pattern of the target. This involves bit shifting and masking. An integer number serves is used as bit mask, with whose bit pattern the shifted input bit pattern is multiplied. You can enter the bit mask in three different ways:

- Hexadecimal: First enter the prefix "0h" or "0H", followed by an optional space for better readability. Then group the bit pattern in blocks of four, for example (0000)(1001)(1010)(1110) and set each block in hexadecimal code: (0)(9)(A)(E). Only the characters 0-9, A-F, a-f are allowed: "0h 09AE".
- Binary: First enter the prefix "0b" or "0B", followed by an optional space for better readability. Then group the binary bit pattern into blocks of four, for example 0000 1001 1010 1110 with spaces in between as a check. Only the characters "0" or "1" are allowed: "0b 0000 1001 1010 1110".
- Decimal: Enter the value directly without prefix, for example "2478".

The shifted input bit pattern is multiplied by the bit mask, with bit-by-bit logical AND operation. The result has a decimal value and is stored in the target.

Note the following:

- Source and target have the same number of bits.
- The number of the bits to shift is smaller than the number of bits of source and target.
- Bit pattern has no more bits than source and target.

Note

When the source and target have a different number of bits, using the system function in the target can result in a violation of the value range.

Note

The left bit is "1" in the source of a signed data type integer with the sign "-". This sign bit is padded with "0" when the bits are shifted right. The sign changes to "+".

Note

If you change the device version of the target HMI device after configuration (e.g."13.1.0" to "13.0.0" or vice versa), you must check and test the parameters of this system function.

The data types "Char" and "Word" can be used for the parameters "Source" and "Target" as of device version 13.1.0. In the device versions before 13.1.0, these parameters must be assigned other data types:

- Use "SInt" instead of "Char"
- Use "Int" instead of "Word"

Otherwise, there might be unwanted effects, for example incorrect or unexpected behavior of the configured system functions.

Use in the function list

ShiftAndMask (Source, Target, Bits to shift, Bit pattern)

The system function "ShiftAndMask" has the following parameters:

Parameter	Description
Source	The Source parameter includes the input bit pattern. Integer-type tags, e.g. "Byte", "Char", "Int", "UInt", "Long" and "ULong" are permitted. Example: The source of the 16-bit integer type has the current value 72: 0000000001001000.
Target	The output-bit pattern is stored in the Target. Integer type tags, e.g. "Byte", "Char", "Int", "UInt", "Long" and "ULong" are permitted.
Bits to shift	Number of bits by which the input bit pattern is shifted right. A negative value shifts the input bit pattern to the left. Example: "Bits to shift" has the value "+3". The input bit pattern is shifted right by three bits when the system function is called: 0000000000001001. Bits to the left are padded with "0". Three bits are truncated on the right. The new decimal value is "9".
Bit pattern	An integer serves as bit mask. The bit pattern is used to multiply the shifted input bit pattern.

Use in scripts

```
HMIRuntime.Tags.SysFct.ShiftAndMask (source, target, bitsToShift,
bitPattern)
```

The system function "ShiftAndMask" has the following parameters:

Parameter	Type	Description
source	HMITag (String)	The Source parameter includes the input bit pattern. Integer-type tags, e.g. "Byte", "Char", "Int", "UInt", "Long" and "ULong" are permitted. Example: The source of the 16-bit integer type has the current value 72: 0000000001001000.
target	HMITag (string)	The output-bit pattern is stored in the Target. Integer type tags, e.g. "Byte", "Char", "Int", "UInt", "Long" and "ULong" are permitted.
bitsToShift	Variant (SINT)	Number of bits by which the input bit pattern is shifted right. A negative value shifts the input bit pattern to the left. Example: "Bits to shift" has the value "+3". The input bit pattern is shifted right by three bits when the system function is called: 0000000000001001. Bits to the left are padded with "0". Three bits are truncated on the right. The new decimal value is "9".
bitPattern	Variant (UDINT)	An integer serves as bit mask. The bit pattern is used to multiply the shifted input bit pattern.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.38 ShowControlPanel (RT Uni)

Description

- Hides or shows the Control Panel.
- Opens an applet in Runtime.

Note

The system function "ShowControlPanel" is only available for WinCC Unified Comfort Panel and is not displayed on WinCC Unified SCADA. The system outputs a compiler warning if the function is nevertheless used through manual entry or through a device replacement in SCADA.

Use in the function list

ShowControlPanel (Home)

The system function "ShowControlPanel" has the following parameters:

Parameter	Description
Home	Specifies the applet to be displayed: "-p AppletName".

Use in scripts

```
HMIRuntime.UI.SysFct.ShowControlPanel(TargetPage)
```

The system function "ShowControlPanel" has the following parameters:

Parameter	Type	Description
`TargetPage`	STRING (HMITag)	Specifies the applet to be displayed: "-p AppletName".

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

List of available applets

List of applet names that are available to the system function:

Display page	Applet name
Panel information	SystemProperties.PanelInformation
Display	SystemProperties.Display
Screensaver	SystemProperties.Screensaver
Reboot	SystemProperties.Reboot
Network settings	NetworkandInternet.NetworkSettings

6.2.39 ShowSoftwareVersion (RT Uni)

Description

Hides or shows the version number of the Runtime software.

Use this system function if during servicing, for example, you required the version of the Runtime software used.

Use in the function list

ShowSoftwareVersion ()

The system function "ShowSoftwareVersion" has no parameters.

Use in scripts

```
HMIRuntime.UI.SysFct.ShowSoftwareVersion()
```

The system function "ShowSoftwareVersion" has no parameters:

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.40 StartProgram (RT Uni)

Description

Starts the specified program on the HMI device.

The Runtime software continues running in the background.

Alarms continue to be output and process values continue to be updated.

When the given application is exited, the screen which was active during the performance of the system function is displayed on the HMI device.

This system function is used, for example, to edit recipe data records in MS Excel on the HMI device.

The function is supported by both the Windows and Linux systems.

Note

On a SIMATIC Unified PC, this system function can only be used to start applications that do not have a user interface.

Use in the function list

StartProgram (Program name, Program parameters, Display mode, Wait for end of program, Result (optional))

The system function "StartProgram" has the following parameters:

Parameter	Description
Program name	Name and path of the program to start. Upper and lower case are taken into account in this parameter.
Program parameters	The parameters you transfer at the start of the program, for example a file that is opened after the start of the program. You can find additional information on the necessary parameters in the documentation of the program to be started.
Display mode	Determines how the program window is displayed on the HMI device. See AUTOHOTSPOT documentation for further information on the "OpenWindow" function. This function has no effect on Linux systems.
Waiting for end of program	Determines whether there is a change back to the project after the called up program has ended: 0 = No change to project. 1 = Change to project.
Result (optional)	Contains data that can be written to the standard output from an external application.

Note

If the path for the program name contains spaces, the program can only be started correctly if the path is specified in quotation marks, e.g. "C:\Program Files\START\start.exe".

Use in scripts

```
HMIRuntime.UI.SysFct.StartProgram(programName, programParameters,
displayMode, waitingForProgramToEnd, result)
```

The system function "StartProgram" has the following parameters:

Parameters	Type	Description
`programName`	STRING (constant, string tag)	Name and path of the program which is started. Upper and lower case are taken into account in this parameter.
`programParameters`	STRING (constant, string tag)	The parameters you transfer at the start of the program, for example a file that is opened after the start of the program. You can find additional information on the necessary parameters in the documentation of the program to be started.
`displayMode`	USINT (constant, scalar tag)	Determines how the program window is displayed on the HMI device. See Win32 Microsoft (https://docs.microsoft.com/en-us/locale/?target=https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-showwindow) documentation for further information on the "OpenWindow" function. This function has no effect on Linux systems.
`waitingForProgramToEnd`	BOOL (constant, scalar tag)	Determines whether there is a change back to the project after the called up program has ended: 0 = No change to project. 1 = Change to project.
`result (optional)`	SetCommand (scalar tag)	Contains data that can be written to the standard output from an external application.

Note

If the path for the program name contains spaces, the program can only be started correctly if the path is specified in quotation marks, e.g. "C:\Program Files\START\start.exe".

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.41 StopRuntime (RT Uni)

Description

Ends the Runtime software and the project running on the HMI device.

Note

All functions after "StopRuntime" are not executed.

Use in the function list

StopRuntime ()

The system function "StopRuntime" has no parameters.

Use in scripts

```
HMIRuntime.SysFct.StopRuntime()
```

The system function "StopRuntime" has no parameters.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.42 ToggleLanguage (RT Uni)

Description

This function allows you to change the Runtime language in order to display language-dependent texts correctly on the user interface. The conversion takes place according to the Runtime language configuration.

Use in the function list

ToggleLanguage()

The system function "ToggleLanguage" has no parameters.

Use in scripts

```
HMIRuntime.Resources.SysFct.ToggleLanguage()
```

The system function "ToggleLanguage" has no parameters.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

6.2.43 WriteManualValue (RT Uni)

Description

Assigns a new value to the specified logging tag. The associated time stamp is transferred in this process.

Use in the function list

WriteManualValue (Logging tag name, Value, Time stamp)

The system function "WriteManualValue" has the following parameters:

Parameter	Description
Logging tag name	Logging tag to which the specified value is assigned.
Value	The value assigned to the specified logging tag.
Time stamp	The time stamp assigned to the specified value.

Use in scripts

```
HMIRuntime.TagLogging.SysFct.WriteManualValue (loggingTagName,
value, timestamp)
```

The system function "WriteManualValue" has the following parameters:

Parameters	Type	Description
`loggedTagName`	HMILoggedTag (String)	Logging tag to which the specified value is assigned.
`value`	Variant	Value assigned to the specified logging tag.
`timestamp`	DateTime	The time stamp assigned to the specified value.

Can be used, provided the configured device supports scripts. For additional information, refer to "Device dependency".

Programming scripts (RT Uni) 7

7.1 Runtime scripting (RT Uni)

Area of application

You use Runtime Scripting in WinCC for the following tasks in runtime:

- Dynamization of properties
- Triggering functions for an event

Runtime Scripting uses JavaScript as the programming language. Runtime Scripting is supported at the following objects:

- Screen
- Screen item
- Task

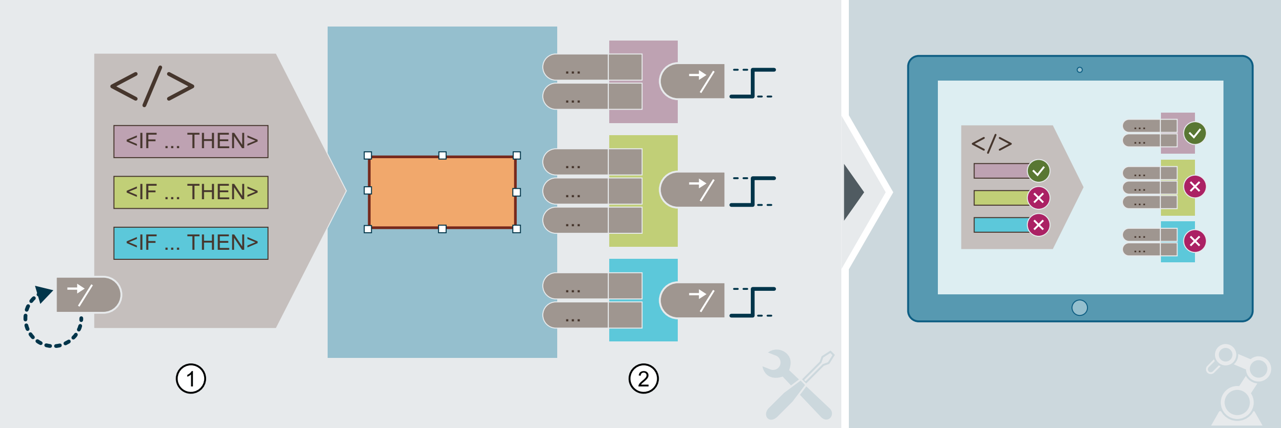
Global modules for frequently required functions

Global modules contain scripts which are available in the entire project. Global modules are therefore suitable for configuring frequently required functions.

Using scripts for dynamization

The properties of screens and screen items can be dynamized via local scripts. In addition, a screen or screen item has its own script area to create a "Global definition".

The "Global definition" is used to define modules, local tags and functions and to import other modules with "Import".



① Global definition for screen or screen item

② Local scripts per property

Input support

The "Scripts" editor assists you in entering code through:

- Syntax highlighting
- Snippets
- System functions
- Referencing HMI objects
- Tooltips
- Autocomplete
- Error marking and correction

See also

Global modules (Page 360)

Local scripts (Page 361)

Configuring a script to an event (Page 368)

Dynamizing object properties by script (Page 369)

Input support (Page 363)

7.2 Basics (RT Uni)

Scripting environment

WinCC provides you with a modern scripting environment that you can use to automate a variety of system components, such as the graphical Runtime system.

In the process, the scripting environment forms individual elements of the system components, such as the screens of the graphical runtime system, through an object model. You use this object model in your scripts to solve a variety of tasks or to control processes.

The script environment in WinCC offers:

- Efficiency and current technologies
  The scripting environment supports Unicode and uses JavaScript (JS) as the scripting language. The scripting environment is object-oriented and offers numerous asynchronous operations for high-performance and secure script execution.
- Support of mass data
  The script environment is optimized for the processing of mass data, for example the writing of 1000 tags in one pass. Special script objects are available to this purpose that record numerous HMI objects of the same type. These script objects execute operations on all the HMI objects simultaneously instead of processing each HMI object individually.

JavaScript

The script environment supports JavaScript in accordance with the ECMAScript Language Specification. Google V8 that implements ECMAScript 2015, 6th Edition of June 2015 (ECMAScript 6) to a very large extent is used as the script engine.

Scripting in WinCC

As a rule the script environment executes all the scripts on the server side. Referenced external resources such as files used in the script therefore have to be available in the server environment.

7.3 Notes on creating scripts (RT Uni)

7.3.1 Data types (RT Uni)

Data types in the object model

Unlike the basic JavaScript data types String, Number and Boolean, the data types in the object model of WinCC are more typified.

The following table lists the utilized data types of the object model:

Data type	Description
Bool	Logical values (True/False)
UInt8	Unsigned 8-bit integer
Int8	Signed 8-bit integer
UInt16	Unsigned 16-bit integer
Int16	Signed 16-bit integer
UInt32	Unsigned 32-bit integer
Int32	Signed 32-bit integer
Float	Signed 32-bit floating-point number
String	Sequence of characters
Variant	Object that can have any data type.
DateTime	Date/time information
StringStringMap	Map: Value pairs from strings
Promise	Object
Object	Object
Function	Method
ErrorCode	Error code

7.3.2 Object instances (RT Uni)

Create objects

In the object model, all object instances are returned via access methods. There are no constructors that create objects.

Example

```
var t1 = Tags("Tag_1"); // returns HMITag object
var screenItem1 = Screen.FindItem("Button_1"); // returns HMIScreenItem
object
```

Error example

```
var t1 = new Tag("Tag_1"); // Error!
```

7.3.3 Asynchronous operations (RT Uni)

Promises in JavaScript

Promises are used in the script environment to handle asynchronous operations. A `Promise` object contains placeholders for results of an operation that are not yet known.

A `Promise` has the following status:

- Pending: Operation of the `Promise` object is still being executed.
- Settled: Operation of the `Promise` object has been completed.
  - Fulfilled: Operation was successful. Result is a value.
  - Rejected: Operation was unsuccessful. Result is a `reason`.
    `reason` may contain an error code for an object with text, links or any other conceivable contents of an object. For this reason, for targeted error processing, it is advisable for `reason` to use an instance of an `Error` object.

As soon as the operation has been completed, a corresponding Handler with the available result is called.

Promises minimize latencies (delay times caused by signal processing), because the script continues to be executed during evaluation. In contrast, a script stops with asynchronous calls. Promises allow clean error handling with the "`catch`" method.

Promises can be cascaded in order to transform results or to sequence asynchronous operations.

Using promises

In the simplest form, Promises return a value or error which is processed with the "then" and "catch" methods of the Promise object:

```
getPromise()
.then(function(Value) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Cascaded promises each return a Promise and allow a sequence of asynchronous calls:

```
getPromise()
.then(return p1)
.then(return p2)
.then(return p3)
.catch();
```

The functions of the Promise objects are executed in the order p1 > p2 > p3 and the "catch" method is called in case of error. All tags of the calling function are also available in the internal functions.

7.3.4 Support for errors (RT Uni)

The user has various options of diagnosing errors and then rectifying them.

Trace Log

WinCC writes a log file for every subsystem. This file contains helpful information of the respective subsystem for narrowing down possible error causes.

Note

The log files are located in the directory "%PROGRAMDATA%\Siemens\Automation\Logfiles \WinCC_Unified_SCADA_V*".

Trace Viewer

The Trace Viewer is an external application for the display and targeted filtering of Trace alarms.

See also

RTIL Trace Viewer (Page 397)

7.3.5 Global modules (RT Uni)

Global modules can only be linked with the global definition (script). All scripts, except the Global definition, are simple functions.

The "import" statement must be a top-level statement.

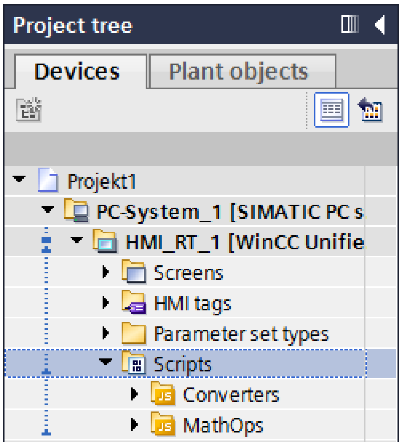
Core statement

A global module is a container for one or more global functions with a shared definition area.

Several global modules can be created for each device.

Global modules are suitable for categorizing or grouping global functions.

Call and view



Global modules are shown in the project tree in the respective device under "Scripts" as folder symbols with the letters "JS"JS.

You can create new global modules via the shortcut menu of "Scripts".

You can create new global functions via the shortcut menu of the desired global module.

You can also add new global modules and functions via "Add ... new" in the project tree.

Examples

- General mathematical operations
- General logic operations
- Unit of measure conversions

See also

Local scripts (Page 361)

7.3.6 Local scripts (RT Uni)

A local script is a function written in JavaScript which is assigned to the object in question.

Starting local scripts

A local script is always started by a trigger:

- an event at a screen object
- a trigger on dynamization of object properties (cycle or change of a tag value)
- the event "Update" of a task in the Scheduler (cycle, change to a tag value or alarm)

Access to global modules

Local scripts can call functions which are contained in the scripts of global modules.

Applications

For example, local scripts can be used to

- dynamize object properties,
- process user entries and
- automatize complex operations

See also

Creating a customized script (Page 366)

Configuring a script to an event (Page 368)

Dynamizing object properties by script (Page 369)

Creating a global definition in a local script (Page 369)

Basics of cycles (Page 927)

7.4 "Scripts" editor (RT Uni)

7.4.1 Structure of the "Scripts" editor (RT Uni)

In the "Scripts" editor, you create and edit customized JavaScript functions.

The "Scripts" editor can be opened in the following execution contexts:

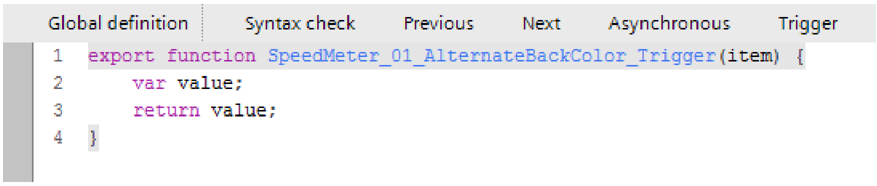
- Global functions in global modules
  You open the "Scripts" editor via the project tree by double-clicking a script.
- Local scripts which are triggered by events ("Images" editor and Scheduler)
  The "Scripts" editor is opened in the Inspector window under "Properties > Events" as soon as you have selected an event and selected the "Convert function list to script" button. Additional local scripts which are triggered by events can be triggered for each property under "Properties > Properties" for the events "Change" and "Quality code change".
- Local scripts for dynamizing object properties
  The "Scripts" editor is opened in the Inspector window under "Properties > Properties" as soon as you select the "Scripts" entry in the "Dynamization" column.

Different representation formats

Depending on the application, the "Scripts" editor is shown in different areas of TIA Portal and contains different control elements.

Depending on the execution context, the code area opens either as a new editor window in the working area (global modules) or embedded in the Inspector window (local script).

Overview



```
export function SpeedMeter_01_AlternateBackColor_Trigger(item) {
    var value;
    return value;
}
```

The code area represents the actual JavaScript code.

Buttons are located above the code area. Depending on the application context, buttons are available with specific functions:

Button	Global modules	Event-related functions	Dynamization of object properties
Global definition[1]	×	✓	✓
Function[2]	×	✓	✓
Deleting a script	×	✓	×
Syntax check	✓	✓	✓
Previous	✓	✓	✓
Continue	✓	✓	✓
Asynchronous	✓	✓	✓
Trigger	×	×	✓

[1] Only visible if the editing mode is active for functions.

[2] Only visible if the editing mode is active for global definitions.

Note

Asynchronous functions cannot be used if the function returns a value.

Alternatively, the value can be specified via the respective property.

Shortcut menu

The shortcut menu contains so-called "Snippets". Snippets provide frequently required code templates.

See also

Global modules (Page 360)

Configuring a script to an event (Page 368)

Dynamizing object properties by script (Page 369)

Input support (Page 363)

7.4.2 Input support (RT Uni)

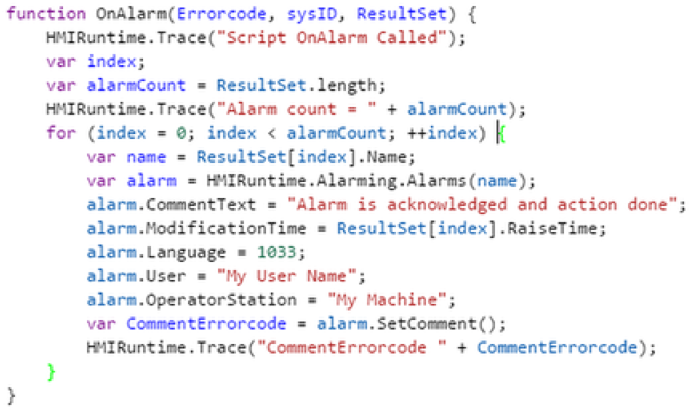
Introduction

You create the JavaScript code of your scripts in the code area of the "Scripts" editor.

The editor supports you with the following functions:

- Syntax highlighting
- Snippets (code templates)
- System functions
- Referencing HMI objects
- Tooltips
- Autocomplete
- Error marking and correction

Syntax highlighting



```
function OnAlarm(Errorcode, sysID, ResultSet) {
    HMIRuntime.Trace("Script OnAlarm Called");
    var index;
    var alarmCount = ResultSet.length;
    HMIRuntime.Trace("Alarm count = " + alarmCount);
    for (index = 0; index < alarmCount; ++index) {
        var name = ResultSet[index].Name;
        var alarm = HMIRuntime.Alarming.Alarms(name);
        alarm.CommentText = "Alarm is acknowledged and action done";
        alarm.ModificationTime = ResultSet[index].RaiseTime;
        alarm.Language = 1033;
        alarm.User = "My User Name";
        alarm.OperatorStation = "My Machine";
        var CommentErrorcode = alarm.SetComment();
        HMIRuntime.Trace("CommentErrorcode " + CommentErrorcode);
    }
}
```

The JavaScript code in the code area of the editor is highlighted in different colors to make it easier to read.

Snippets for programming support

Snippets are code templates for frequently required instructions: Snippets are divided into the following groups:

- HMIRuntime
  Contains Snippets for accessing the object model, e.g. "`Change base screen`" or "`Set Connection Mode`".
- Logic
  Contains Snippets such as "`If...Else`" or "`For loop`".

You insert a Snippet in the local script via the shortcut menu.

System functions

The system functions are provided in the "Scripts" editor. For additional information, refer to "System functions".

Referencing HMI objects

HMI objects (e.g. tags and screens) are referenced in scripts.

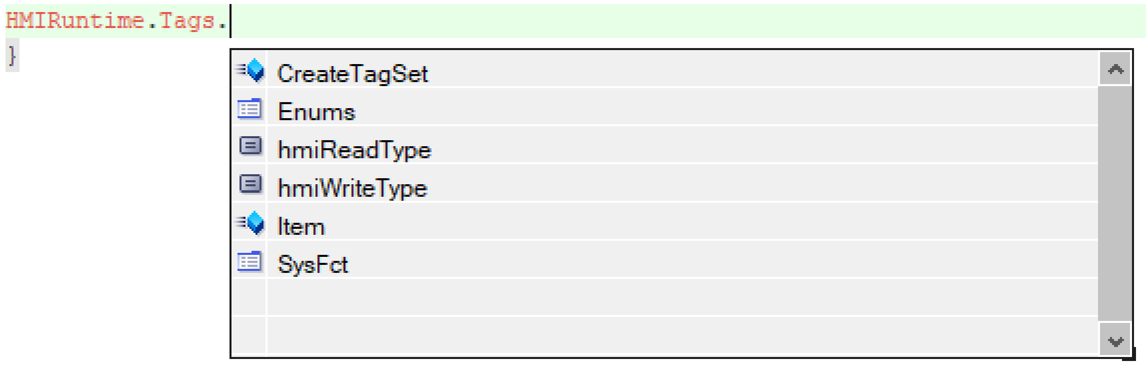
Therefore, you perform the following actions without editing the script:

- Rename HMI object
- Re-create a previously deleted HMI object
- Create HMI object used as text reference in the script

Info tips

While you compose the code, additional information about all objects of the WinCC Unified object model is displayed. For example, you receive information on the required parameters of the system functions.

Autocomplete



```
HMIRuntime.Tags.
}
```

Autocomplete supports you in entering your code. Suitable objects of the WinCC Unified object model are displayed.

Error marking and correction

Error while configuring

Your JavaScript code is checked immediately during input for a variety of criteria and highlighted in color in case of errors.

- To get a tooltip, move the cursor over the marking.

Errors during compiling and loading

Alarms during the compiling and loading of a project are displayed in the Inspector window in the "Info > Compile" tab.

The "Scripts" editor supports you by displaying faulty scripts directly for editing:

- To go directly to the "Scripts" editor, select the green arrow ↗.

See also

System functions (Page 308)

7.4.3 Creating a customized script (RT Uni)

Introduction

Customized scripts are functions in the form of JavaScript code which are developed and entered by the user.

You solve individual tasks with customized scripts, e.g.

- Automating processes
- Dynamizing objects
- Evaluating events, such as user input.

Note

When creating scripts, also consider the object model.

Preparation

In advance of creating the script, consider where you will enter the script and how it will be executed.

Depending on which objects and system components you address, you use a specific editor for creating your JavaScript code.

Depending on the execution context of the editor, the script is executed in a script context with differing ranges of validity.

Note

Each module has its own namespace, which means it is possible to use the same function name and global tag name in two modules.

The functions are distinguished by the symbol name of the imported module. Example:

```
import * as modA from 'ModuleA';

import * as modB from 'ModuleB';

modA.function1();

modB.function1();
```

Execution context of the editor	Script context and referencing
"Scripts" editor in the "Screens" editor	Each process screen has two independent script contexts: ● Context for dynamization of properties ● Context for evaluation of events The script of a property cannot access global tags of an event even in the same screen. Each script context references the modules that it has imported using the 'import' statement. However, each script context receives its own copy of the tags defined there.
"Scripts" editor in the Scheduler	All jobs share a script context. Different tasks can access common global tags. Each required module must be referenced by means of the 'import' statement.

Create 'import' statement

A module can be referenced in a script or another module:

- Drag and drop the module to be referenced into the "Global Definition" area of the script or into the definition area of the module.

Requirement

- "Scripts" editor is opened in the desired execution context.

Procedure

1. Enter your JavaScript code in the code area.
2. Use Snippets and system functions to create error-free code faster.
3. When you dynamize the property of a screen item, add a trigger.
4. Perform a syntax check.
5. Save the project.

See also

Local scripts (Page 361)

Configuring a script to an event (Page 368)

Dynamizing object properties by script (Page 369)

Creating a global definition in a local script (Page 369)

WinCC Unified object model (Page 411)

7.4.4 Configuring a script to an event (RT Uni)

Note

Restriction of "activated" and "deactivated" events

If the focus is on the affected screen item, scripts are executed at the "activated" and "deactivated" events.

Requirement

- One of the following objects is configured:
  - Task
  - Screen
  - Screen item

Procedure

To configure a script to an event, follow these steps:

1. Open the relevant editor.
2. Select the object.
3. Select the event under "Properties > Events" in the Inspector window.
4. Generate the local script.
5. Write the code.
6. Perform a syntax check.
7. Save the project.

See also

Runtime scripting (Page 355)

Local scripts (Page 361)

Creating a customized script (Page 366)

Creating a global definition in a local script (Page 369)

7.4.5 Dynamizing object properties by script (RT Uni)

Requirement

- One of the following objects is configured:
  - Screen
  - Screen object
- The selected object property supports the dynamization type "Script".

Procedure

1. Open the editor of the object in question.
2. Select the object.
3. Select the desired object property under "Properties > Properties" in the Inspector window.
4. Dynamize the object property:
   - Select the "Script" entry in the "Dynamization" column.
     The editor creates a script and is displayed in the Inspector window.
   - Write the code.
5. Select the trigger that triggers the dynamization in runtime.

Result

The script changes the selected property dynamically in line with the written code.

See also

Local scripts (Page 361)

Creating a global definition in a local script (Page 369)

7.4.6 Creating a global definition in a local script (RT Uni)

Procedure

1. Open the local script.
2. Click "Global definition".
3. Write the code.
4. Perform a syntax check.
5. Click "Function".

See also

Local scripts (Page 361)

Creating a customized script (Page 366)

Configuring a script to an event (Page 368)

Dynamizing object properties by script (Page 369)

7.5 Examples (RT Uni)

7.5.1 Notes on the code examples (RT Uni)

General

The comments at the beginning of each code example are required for technical reasons.

At the same time these comments show the relationships between various code examples within a chapter.

Further comments in the code examples explain individual program code lines.

Executing examples

1. Set the language of the develop environment to "English", so that the object names used in the examples are automatically assigned correctly.
2. Create a project with corresponding screens in which you can configure buttons, I/O fields, etc. These elements are required to carry out the code examples.
3. Apply the code examples to the associated script areas.
4. Compile the project.
5. Start the simulation of the project.
6. Start the tracer to diagnose potential errors.

See also

RTIL Trace Viewer (Page 397)

7.5.2 Dynamizing the position of an object (RT Uni)

Introduction

The example shows how to dynamically change an object position using JavaScript.

A classic application would be the adjustment of the object position to the size and/or position of adjacent objects on the screen page.

Execution of example

1. Configure 4 tags with the names "HMI_Tag1" to "HMI_Tag_4".
2. Configure a screen page with the following objects:
   - 4 objects of the type I/O field with the process values "HMI_Tag_1" to "HMI_Tag_4"
   - 1 object of the type "Circle" with the name "Circle_1"
   - 1 object of the type "Button" with the name "Button_1"
3. Dynamize the parameters "Position X" and ""Position" Y" of the objects "Circle_1" and "Button_1" using scripts.
4. Set the triggers for dynamization with the tags
   - HMI_Tag_1 for Button_1/Position X,
   - HMI_Tag_2 for Button_1/Position Y,
   - HMI_Tag_3 for Circle_1/Position X,
   - HMI_Tag_4 for Circle_1/Position Y.

   The scripts are started accordingly by changing the tag.
5. Copy the sample code below to your project.

Reading out and returning the tag `HMI_Tag_1`

The script with the function "`Circle_1_CenterX_Trigger(item)`" is created when the parameter "Position X" is dynamized by a script in the properties of the object "Circle_1".

```
//JEx: "Position X dynamization of a Circle with tags"
//TagsRequired: "HMI_Tag_1"

export function Circle_1_CenterX_Trigger(item) {
    var value = Tags("HMI_Tag_1").Read();
    return value;
}
```

Reading out and returning the tag `HMI_Tag_2`

The script with the function "`Circle_1_CenterY_Trigger(item)`" is generated when the parameter "Position Y" is dynamized by a script in the properties of the object "Circle_1".

```
//JEx: "Position Y dynamization of a Circle with tag"
//TagsRequired: "HMI_Tag_2"

export function Circle_1_CenterY_Trigger(item) {
    var value = Tags("HMI_Tag_2").Read();
    return value;
}
```

Reading out and returning the tag `HMI_Tag_3`

The script with the function "`Button_1_Left_Trigger(item)`" is generated when the parameter "Position X" is dynamized by a script in the properties of the object "Button_1".

```
//JEx: "Position X dynamization of a Button with tag"
//TagsRequired: "HMI_Tag_3"

export function Button_1_Left_Trigger(item) {
    var value = Tags("HMI_Tag_3").Read();
    return value;
}
```

Reading out and returning the tag `HMI_Tag_4`

The script with the function "`Button_1_Top_Trigger(item)`" is generated when the parameter "Position Y" is dynamized by a script in the properties of the object "Button_1".

```
//JEx: "Position Y dynamization of a Button with tag"
//TagsRequired: "HMI_Tag_4"

export function Button_1_Top_Trigger(item) {
    var value = Tags("HMI_Tag_4").Read();
    return value;
}
```

Result (in runtime)

The position of the object in question changes in accordance with the values entered in the I/O fields.

See also

Notes on the code examples (Page 370)

7.5.3 Reading and writing tag values (RT Uni)

Introduction

The example shows how to read, multiply and write the values from two tags (HMI_Tag_2, HMI_Tag_3) to another tag (HMI_Tag_1).

In practical use, the tags could represent the following parameters:

- HMI_Tag_1: Apparent power S
- HMI_Tag_2: Electrical voltage U
- HMI_Tag_3: Electrical current I

Execution of example

1. Configure 3 tags "HMI_Tag1" to "HMI_Tag_3" of the "Int" type.
2. You configure the following objects on a screen:
   – 1 button (in the example "Button_1")
   – 3 I/O fields with the process values "HMI_Tag_1" to "HMI_Tag_3"
3. Create a script for the event "Click left mouse button".
   The JavaScript editor creates the function "`Button_1_OnTapped(item, x, y, modifiers, trigger)`".
4. Insert the example code.

Sample code

```
//JEx: "Reading and writing tag values"
//Tags_Required: "HMI_Tag_1"; "HMI_Tag_2"; "HMI_Tag_3"

export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
    Tags("HMI_Tag_1").Write(Tags("HMI_Tag_2").Read() * Tags("HMI_Tag_3").Read());
}
```

See also

Notes on the code examples (Page 370)

7.5.4 Change language (RT Uni)

Introduction

The example shows how to change the language of the interface using JavaScript.

Execution of example

1. Configure a button "Button_1".
2. Activate the project languages required for the example:
   - English
   - German
   - Hungarian
   - French
3. Create a script for the event "Left mouse button clicked" of the button "Button_1".
4. Define the tag "step" under "Global definition" and assign the value "2" to it.

Global definition

```
//JEx: "LanguageChangeGlobalDef"
var step = 2;
```

Sample code

Clicking the button increments the tag "step". The stored code for the language is assigned to `HMIRuntime.Language` according to its value.

```
//JEx: "LanguageChange"
//JExRequired: "LanguageChangeGlobalDef"

export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
    //change to English
    if (step == 1) {HMIRuntime.Language = "1033";}

    //change to Hungarian
    if (step == 2) {HMIRuntime.Language = "1038";}

    //change to German
    if (step == 3) {HMIRuntime.Language = "1031";}

    //change to French
    if (step == 4) {HMIRuntime.Language = "1036";}

    step ++; //step
    if (step == 5) {step = 1;} //reset
}
```

See also

Notes on the code examples (Page 370)

7.5.5 Dynamically changing the output format of an object (RT Uni)

Introduction

This example shows how to dynamically change the output format of an object of the type "I/O field" using JavaScript.

The output in the example is switched between the following formats:

- Hexadecimal
- Decimal
- Binary

Execution of example

1. Configure a screen page with 3 buttons with the names "Button_1" to "Button_3".
2. Configure max. 9 objects of the type I/O field with the names "HmiIOField_1" to "HmiIOField_9".
3. Create a script for the event "Button pressed" for each button.
4. Define the constants as described in the section "Global definition of constants".
5. Create 2 global scripts:
   - toggleOutputFormat()
   - setOutputFormat(format)
6. Transfer the source code from the following sections.

Global script "toggleOutputFormat()"

```
//JEx: "Toggle Output Format"
//SOM_OM_"HmiIOField"
//JExRequired: "Set Output Format";"GlobalConstants for OutputFormat"

function toggleOutputFormat() {
    let index = outputFormats.indexOf(Screen.FindItem(screenItemNameBase +
minScreenItemIndex).OutputFormat);
    index = (index + 1) % outputFormats.length;
    setOutputFormat(outputFormats[index]);
}
```

Global script "setOutputFormat(format)"

```
//JEx: "Set Output Format"
//SOM_OM_"HmiIOField"
//JExRequired: "GlobalConstants for OutputFormat"

function setOutputFormat(format) {
    for(let i = minScreenItemIndex; i <= maxScreenItemIndex; i++) {
        let name = screenItemNameBase + i;
        Screen.FindItem(name).OutputFormat = format;
    }
}
```

Global definition of constants

```
//JEx: "GlobalConstants for SetOutputFormat"

const outputFormats = ['{I}', '{H2,2}', '{B8,4}'];
// Erstellen der dynamischen Objektnamen, die sich ausschließlich durch die
Nummer am Ende unterscheiden
const screenItemNameBase = 'HmiIOField_';
// the screen item names begin with the prefix 'HmiIOField_'
const minScreenItemIndex = 1; // range of the screen items: min
const maxScreenItemIndex = 9; // range of the screen items: max
```

Switch to hexadecimal output format

```
//JEx: "Switch Output Format Hex"
//SOM_OM_"HmiIOField"
//JExRequired: "Set Output Format"

export function Button_1_OnDown(item, x, y, modifiers, trigger) {
    setOutputFormat('{H2,2}');
}
```

Switch to decimal output format

```
//JEx: "Switch Output Format Dec"
//SOM_OM_"HmiIOField"
//JExRequired: "Set Output Format"

export function Button_2_OnDown(item, x, y, modifiers, trigger) {
    setOutputFormat('{I}');
}
```

Switch to the next output format

```
//JEx: "Switch Output Format Bin"
//SOM_OM_"HmiIOField"
//JExRequired: "Toggle Output Format"

export function Button_3_OnDown(item, x, y, modifiers, trigger) {
    toggleOutputFormat();
}
```

See also

Notes on the code examples (Page 370)

7.5.6 Setting the alarm filter (RT Uni)

Introduction

The example shows how to set the alarm filter using a JavaScript function.

To make the examples easy, the filters are set by pressing a button.

Classic application example: Setting alarm filters using check boxes on a screen page.

Execution of example

1. Configure a screen page with the following elements:
   – 1 alarm view (Alarm_control_1)
   – 2 buttons (in the example "Button_7" and "Button_8").
2. Configure the generation of alarms of the alarm class "Warning" and "Alarm".
3. Ensure that the alarms in the alarm view "Alarm control_1" are displayed.

Filtering sample code for "Warning"

In the sample code, messages of the alarm class "Warning" are filtered.

```
//JEx: "Alarm Filter Control with Warnings"
//SOM_OM_"Alarm control" (Alarm control_1);SOM_OM_
//JExRequired: "Alarm Subscription"

export function Button_7_OnDown(item, x, y, modifiers, trigger) {
    let alarmControl = Screen.FindItem('Alarm control_1');
    alarmControl.Filter = "AlarmClassName='Warning'";
}
```

Filtering sample code for "Warning" and "Alarm"

In the sample code, messages of the alarm class "Warning" and "Alarm" are filtered.

```
//JEx: "Alarm Filter Control with Warnings and Alarms"
//SON_ON_"Alarm control" (Alarm control_1);
//JExRequired: "Alarm Subscription"

export function Button_8_OnDown(item, x, y, modifiers, trigger) {
    let alarmControl = Screen.FindItem('Alarm control_1');
    alarmControl.Filter = "AlarmClassName IN ('Warning','Alarm')";
}
```

See also

Notes on the code examples (Page 370)

7.5.7 Creating an alarm subscription (RT Uni)

Introduction

The example shows how an "Alarm Subscription" is programmed in JavaScript.

Execution of example

1. Configure a task in the Scheduler with the time interval for the update.
2. Create a script for the event "Update" at the configured task.
3. Create the tag "subs" under "Global definition" (compare section "Global definition").
4. Copy the code under "Example code" to the script for the "Update" event.

Global definition

```
//JEx: "AlarmSubscriptionDef"
// A global tag is required for the Alarm Subscription,
// to be able to end the subscription
var subs;
```

Sample code

The JavaScript function in the example is started via the event "Update" of a task in the Scheduler.

```
//JEx: "AlarmSubsciption"
//JExRequired: "AlarmSubscriptionDef"
//TagsReqired: "HMI_Tag_1_Int"

// Function will be generated when you create a script on event "Update" of a task
export function Task_Task_1_Update() {
    let tag = Tags("HMI_Tag_1_Int");
    tag.Read();
    if(tag.Value == 1) {
        // start subscription
        if(subs) {
            // stop already existing subscription
            subs.Stop();
            subs =  undefined;
        }
        subs = HMIRuntime.Alarming.CreateSubscription();
        subs.Filter = 'AlarmClassName=\'Warning\'';
        //subs.Language = 1033; // For explicit setting of a specific language
        subs.OnAlarm = function(Errorcode, SystemNames, ResultSet) {
            // Callback function is performed as soon as the alarm attribute changes
            for (let index in ResultSet) {
              HMIRuntime.Trace('Alarm[' + index + '] Name="' + ResultSet[index].Name + '"');
                HMIRuntime.Trace('Alarm[' + index + '] State="' + ResultSet[index].State +
'"');
                HMIRuntime.Trace('Alarm[' + index + '] EventText="' +
ResultSet[index].EventText + '"');
            }
        }
        subs.Start();
    } else if(tag.Value == 2) {
        // stop subscription
        if(subs) {
            // Stop already existing Alarm Subscription
            subs.Stop();
            subs = undefined;
        }
    }
}
```

See also

Notes on the code examples (Page 370)

7.5.8 Reading and writing binary files (RT Uni)

Introduction

The example shows how to write a binary file to the file system with JavaScript.

Note

- For access to the data, use the class "DataView" or one of the "*Array" classes together with the class "arrayBuffer".
- If the Endianess of the computer systems used for development (Compiler) and execution (Runtime) are different, use the class "DataView".
- The method HMIRuntime.Trace(text) outputs whether the function was successful via the Tracer.

Execution of example

1. Configure 3 buttons with the names "Button_10", "Button_11" and "Button_14".
2. Create a script for the event "Print" for each button.
3. Copy the sample code below to your project.

Writing a binary file

```
//JEx: Write a binary file with class "Int32Array" into file system

export function Button_10_OnDown(item, x, y, modifiers, trigger) {
    var arrayBuffer = new Int32Array([1,2,3]);
    HMIRuntime.FileSystem.WriteFileBinary('C:\\Users\\Public\\binaryfile.bin',
arrayBuffer).then(
    function() {
        HMIRuntime.Trace('Write file finished successfully');
    }).catch(
    function(e) {
        HMIRuntime.Trace(`Write file failed with error code ${e}`);
    });
}
```

Reading a binary file with the class **`Int32Array`**

```
//JEx: Read a binary file with class "Int32Array" from file system

export function Button_11_OnDown(item, x, y, modifiers, trigger) {
    var arrayBuffer = new Int32Array([1,2,3]);
    HMIRuntime.FileSystem.ReadFileBinary('C:\\Users\\Public\\binaryfile.bin',
arrayBuffer).then(
    function(arrayBuffer) {
        let intView = new Int32Array(arrayBuffer);
        for(let i in intView) {
            HMIRuntime.Trace('intView[' + i  + '] = ' + intView[i]);
        }
    }).catch(
    function(e) {
        HMIRuntime.Trace(`Read file failed with error code ${e}`);
    });
}
```

Reading a binary file with the class **`DataView`**

```
//JEx: Read a binary file with class "DataView" from file system


export function Button_14_OnDown(item, x, y, modifiers, trigger) {
    HMIRuntime.FileSystem.ReadFileBinary('C:\\Users\\Public\\binaryfile.bin').then(
    function(arrayBuffer) {
        let dv = new DataView(arrayBuffer, 0, arrayBuffer.length);
        HMIRuntime.Trace('Int32[0] LE:' + dv.getInt32(0, true).toString(16));
        HMIRuntime.Trace('Int32[0] BE:' + dv.getInt32(0      ).toString(16));
        HMIRuntime.Trace('Int16[2] LE:' + dv.getInt16(2, true).toString(16));
    }).catch(
    function(e) {
        HMIRuntime.Trace(`Read file failed with error code ${e}`);
    });
}
```

See also

Notes on the code examples (Page 370)

7.5.9 Reading and writing text files (RT Uni)

Introduction

This example shows how text files can be read and written with JavaScript.

Execution of example

Note

The method HMIRuntime.Trace(text) outputs whether the function was successful via the Tracer.

1. Configure 2 buttons "Button_12" and "Button_13".
2. Create a script for the event "Print" for each button.
3. Copy the sample code below to your project.

Writing sample code for text file

```
//JEx: "Write Text File"

export function Button_12_OnDown(item, x, y, modifiers, trigger) {
    HMIRuntime.FileSystem.WriteFile('C:\\Users\\Public\\textfile.txt', 'my utf8 string',
'utf8').then(
    function() {
        HMIRuntime.Trace('Write file finished successfully');
    }).catch(function(errorCode) {
        HMIRuntime.Trace('Write failed errorcode=' + errorCode);
    });
}
```

Reading sample code for text file

```
//JEx: "Read Text File"

export function Button_13_OnDown(item, x, y, modifiers, trigger) {
    HMIRuntime.FileSystem.ReadFile('C:\\Users\\Public\\textfile.txt',
'utf8').then(
    function(text) {
        HMIRuntime.Trace('Text=' + text);
    }).catch(function(errorCode) {
        HMIRuntime.Trace('Read failed errorcode=' + errorCode);
    });
}
```

See also

Notes on the code examples (Page 370)

7.5.10 Converting values (RT Uni)

Introduction

The example shows how temperature values can be converted into a different unit with JavaScript.

Executing an example

1. Create a global module "TemperatureConversions" with 2 global scripts:
   - "celsiusToFahrenheit(t_celsius)"
   - "fahrenheitToCelsius(t_fahrenheit)"
2. Copy the corresponding sample code to the global scripts.
3. Copy the sample code from the "Global definition range" to the global definition range of the global module "TemperatureConversion".
4. Create a screen with 2 elements of the type "I/O field".
5. Dynamize the "Process value" property of the two "I/O field" elements through scripts.
6. Copy the sample code of both scripts.

Celsius to Fahrenheit (global script)

```
//JEx: "CelsiusToFahrenheit"
//JExRequired: "TempConv_GlobalDefRange"

export function celsiusToFahrenheit(t_celsius) {
    return t_celsius * 1.8 + 32;
}
```

Fahrenheit to Celsius (global script)

```
//JEx: "FahrenheitToCelsius"
//JExRequired: "TempConv_GlobalDefRange"

export function fahrenheitToCelsius(t_fahrenheit) {
    return (t_fahrenheit - 32) / 1.8;
}
```

Global definition range

```
//JEx: "TempConv_GlobalDefRange"
import * as tempConv from 'TemperatureConversion';
```

Celsius to Fahrenheit (dynamization of process value)

The JavaScript function is triggered by the tag 'celsius1'.

```
//JEx: "DynCelsiusToFahrenheit"
//SOM_OM_"HmiIOField"
//TagsRequired: "celsius1"
//JExRequired: "celsiusToFahrenheit"
//JExRequired: "TempConv_GlobalDef"

export function I_O_field_1_ProcessValue_Trigger(item) {
    const tagCelsius = Tags('celsius1');
    tagCelsius.Read();
    return tempConv.celsiusToFahrenheit(tagCelsius.Value);
}
```

Fahrenheit to Celsius (dynamization of process value)

The JavaScript function is triggered by the tag 'fahrenheit1'.

```
//JEx: "DynFahrenheitToCelsius"
//SOM_OM_"HmiIOField"
//TagsRequired: "fahrenheit1"
//JExRequired: "fahrenheitToCelsius"
//JExRequired: "TempConv_GlobalDef"

export function I_O_field_2_ProcessValue_Trigger(item) {
    const tagFahrenheit = Tags('fahrenheit1');
    tagFahrenheit.Read();
    return tempConv.fahrenheitToCelsius(tagFahrenheit.Value);
}
```

See also

Notes on the code examples (Page 370)

7.5.11 Setting bits (RT Uni)

Introduction

The example shows how single and multiple bits are masked and set with JavaScript.

Note

Using different methods for integer data types

The methods of the "Math.Uint64" object and of the standard model exist for setting and resetting multiple bits.

The methods of the "Math.Uint64" object work with all integer data types.

- For values < 2^{31} use the standard methods of JavaScript.
- For values ≥ 2^{31} use the methods from the "Math.Uint64" object model.

Executing an example

1. Create 6 buttons in a project.
2. Create a tag of the "Int" type.
3. For all 6 buttons create local scripts for the event "Left mouse button pressed".
4. Transfer the source code for the examples to the respective script areas.
5. To retain a clear overview assign descriptive texts to the buttons.

Setting a single bit (not error handling)

The methods "SetBit()" and "ResetBit()" of the "HMITag" class exist for setting and resetting single bits.

```
//JEx: "SetSingleBit"
//SOM_OM_"
//TagsRequired: "HMI_Tag_1"

export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
    Tags('HMI_Tag_1').SetBit(37);
}
```

Setting a single bit

During error handling a corresponding message is output with the "`HMIRuntimeTrace`" method.

```
//JEx: "SetSingleBitWithErrorHandling"
//SOM_OM_"
//TagsRequired: "HMI_Tag_1"

export function Button_5_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    const bitNum = 37;
    tag1.SetBit(bitNum).then(() => {HMIRuntime.Trace('SetBit succeeded');})
    .catch((e) => {HMIRuntime.Trace(`SetBit failed, error=${e}`);});
}
```

Changing bits with "Xor" without error handling

Changing multiple bits with 64 bits without error handling.

The example is created with the method of the "Math.Uint64" object model.

```
//JEx: "SetMultipleBits"
//SOM_OM_"
//TagsRequired: "HMI_Tag_1"

export function Button_6_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    // Define a 64-bit mask using a binary constant
    const mask =
HMIRuntime.Math.Uint64('0b0110011000000000000000000000000000001');
    tag1.Read();
    // Check whether the value is of the type "Uint64".
    // The 'And', 'Or' and 'Xor' methods of the "Math.Uint64" object model
only work if this is the case.
    const newValue = HMIRuntime.Math.Uint64(tag1.Value);
    newValue.Xor(mask); // use '.And()' / '.Or' for clearing / setting bits
    tag1.Write(newValue);
}
```

Setting bits with "Or"

The function sets every bit in the mask that has the value "1".

The example is created with the method of the "Math.Uint64" object model, uses asynchronous writing and contains an extended error handling.

```
//JEx: "SetMultipleBitsWithErrorHandlingOr"
//TagsRequired: "HMI_Tag_1"

// set bits with 'Or': sets every bit which is '1' in mask
export function Button_16_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    const mask = 
HMIRuntime.Math.Uint64('0b011001100000000000000000000000000001');
    tag1.Read();
    if(tag1.LastError != 0) {
        HMIRuntime.Trace(`Read failed, error=${tag1.LastError}`);
    } else if((tag1.QualityCode & 0x80) == 0) {
        // Check whether QC is 'good' or 'good cascade'
        HMIRuntime.Trace(
            `Read succeeded, but Quality is not 'good', QC=$
{tag1.QualityCode}`
        );
    } else {
        const newValue = HMIRuntime.Math.Uint64(tag1.Value);
        newValue.Or(mask); // Set bits
        const ts = Tags.CreateTagSet([[tag1.Name, newValue]]);
        ts.WriteAsync().then(()=>{HMIRuntime.Trace('Write succeeded')})
        .catch((e)=>{HMIRuntime.Trace(`Write failed, error=${e}`)});
    }
}
```

Deleting bits with "AND"

The script deletes each masked bit with the value "1".

The example is created with the method of the "Math.Uint64" object model.

```
//JEx: "ClearMultipleBitsWithErrorHandlingAnd"
//TagsRequired: "HMI_Tag_1"

export function Button_17_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    const mask =
HMIRuntime.Math.Uint64('0b0110011000000000000000000000000000001');
    // invert all bit of mask for 'And' operation
     mask.Xor(HMIRuntime.Math.Uint64('0xffffffffffffffff'));
    tag1.Read();
    if(tag1.LastError != 0) {
        HMIRuntime.Trace(`Read failed, error=${tag1.LastError}`);
    } else if((tag1.QualityCode & 0x80) == 0) {
        // Check whether QC is 'good' or 'good cascade'
        HMIRuntime.Trace(
            `Read succeeded, but Quality is not 'good', QC=$
{tag1.QualityCode}`
        );
    } else {
        const newValue = HMIRuntime.Math.Uint64(tag1.Value);
        newValue.And(mask); // Delete bits
        const ts = Tags.CreateTagSet([[tag1.Name, newValue]]);
        ts.WriteAsync().then(()=>{HMIRuntime.Trace('Write succeeded')})
        .catch((e)=>{HMIRuntime.Trace(`Write failed, error=${e}`)});
    }
}
```

Replacing bits with "Xor"

The script replaces each bit in the mask that has the value "1".

The example is created with the method of the "Math.Uint64" object model.

```
//JEx: "FlipMultipleBitsWithErrorHandlingXor"
//TagsRequired: "HMI_Tag_1"

export function Button_12_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    const mask =
HMIRuntime.Math.Uint64('0b0110011000000000000000000000000000001');
    tag1.Read();
    if(tag1.LastError != 0) {
        HMIRuntime.Trace(`Read failed, error=${tag1.LastError}`);
    } else if((tag1.QualityCode & 0x80) == 0) {
        // Check whether QC is 'good' or 'good cascade'
        HMIRuntime.Trace(
            `Read succeeded, but Quality is not 'good', QC=$
{tag1.QualityCode}`
        );
    } else {
        const newValue = HMIRuntime.Math.Uint64(tag1.Value);
        newValue.Xor(mask);
        const ts = Tags.CreateTagSet([[tag1.Name, newValue]]);
        ts.WriteAsync().then(()=>{HMIRuntime.Trace('Write succeeded')})
        .catch((e)=>{HMIRuntime.Trace(`Write failed, error=${e}`)});
    }
}
```

See also

Notes on the code examples (Page 370)

7.5.12 Changing the date format (RT Uni)

Introduction

The example shows how the date format is changed using JavaScript.

Executing an example

1. Create 2 I/O fields und 1 button.
2. Configure 2 global tags:
   – HMI_Tag_1 of the type "Int"
   – HMI_Tag_2 of the type "WString"
3. Create a script for the event "Press" of the button "Button_1".

Sample code

```
//JEx: "ChangeDateformat"
//SOM_OM_"HmiIOField_1", SOM_OM"HmiIOField_2"

//TagsRequired: "HMI_Tag_1:Int","HMI_Tag_2:WString"

export function Button_1_OnDown(item, x, y, modifiers, trigger) {
    //Create array with all month names
    var monthNames = [
        "January", "February", "March",
        "April", "May", "June", "July",
        "August", "September", "October",
        "November", "December"
    ];
    var date = new Date(); //Create tag with current date
    var day = date.getDate(); //Separation of the individual date components
    var monthIndex = date.getMonth();
    var year = date.getFullYear();
    var month = monthIndex + 1;
    //The "getMonth()" object contains 12 values from "0" to "11".

    //Set the date format
    switch (Tags("HMI_Tag_1").Read()) {
        case 1: Tags("HMI_Tag_2").Write(day +'/' +month +'/' +year); break;
        case 2: Tags("HMI_Tag_2").Write(year +'-'  +month+ '-' +day); break;
        case 3:
           Tags("HMI_Tag_2").Write(year +'-'  +monthNames[monthIndex] + '-'
+day);
            break;
        default:
            Tags("HMI_Tag_2").Write(day + ' ' + monthNames[monthIndex] + '
' + year);
            break;
    }
}
```

See also

Notes on the code examples (Page 370)

7.5.13 Simulating value changes in tags (RT Uni)

Introduction

This example shows how tags are supplied with values in defined time intervals by a simulation.

With the simulation, demo projects can be created or tested without process integration.

Note

Connection of existing process tags leads to influencing of processes

The simulation writes the calculated values to the tags without further test steps.

- Do not link any external tags; these remain linked to any existing process.
  The simulation thus influences the process in which the external tag is integrated.
- If you purposely want to influence a process with simulation, note that the external tag of the process can only be reached for simulation if the following requirements are met:
  - The connection to the controller (PLC) is established.
  - The controller (PLC) is in "RUN" mode.

The global functions for generating sine and sawtooth waves use 5 parameters:

- `counter`: Counter that uses the current date in milliseconds.
- `phase`: Phase offset as a factor between 0.0 and 1.0
  The factor 0.0 to 1.0 corresponds to a phase offset of 0° to 360°.
- `period`: Duration of a full vibration cycle in milliseconds
- `amp`: Strength of the amplitude
- `offset`: Shift of the amplitude on the y-axis

Executing an example

1. Create a global "Simulator" module.
2. Configure the two functions "sinWave" and "sawTooth" in this global module.
   Use the source codes from the following sections for the functions:
   - "Example code > Simulate sine wave (global script)"
   - "Example code > Simulate sawtooth wave (global script)"
3. Create a script for the "Loaded" event for the screen.
4. Go to the "Global definition" view of the event.
5. Insert the sample code under "Sample code > Event - Global definition area" in the "Global definition" view of the event.
6. If necessary, adapt the copied sample code to your project. For example, if you use more than 2 tags, you must add more lines with the function `ts.Add(...)`.
7. Go back to the "Function" view of the event.
8. Insert the source code from "Example code > Event".

The tags HMI_Tag_1 and HMI_Tag_2 can now be connected to any objects in the screen that can display the values, such as:

- f(x) trend view
- Gauges

- Bar graphs
- Text fields

Result

1. When the application is loaded in Runtime, the `import` function initializes the scripts from the global module "Simulator" for the "Loaded" event of the screen.
   The example code can be found under "Event - Global definition area".
2. If the event is triggered when the screen is loaded, the function "simulateTags()" starts at the specified intervals.
   In the example, the interval is 500 ms.
   The call of the function "simulateTags()" is stopped as soon as the screen is deselected.
   The example code can be found under "Event" and "Event - Global definition area".
3. The function "simulateTags()" starts with each call of the functions "sinWave" and "sawTooth" and transfers the new values to the tags.
   The example code can be found under "Simulate sine wave (global script)" and "Simulate sawtooth wave (global script)".

Sample code

Simulate sine wave (global script)

```
//JEx: "Simulate Sine Wave"

export function sinWave(counter, phase, period, amp, offset) {
 return offset + amp * Math.sin((phase + ((counter % period) / period)) * (2*Math.PI));
}
```

Simulate sawtooth wave (global script)

```
//JEx: "Simulate Saw Tooth Wave"

export function sawTooth(counter, phase, period, amp, offset) {
 return offset + amp * (((counter + phase * period) % period) / period);
}
```

Global event definition area

```
//JEx: "Generate signals"
//SOM_OM_"HmiTrendControl"
//JExRequired: "Simulate Sine Wave", "Simulate Saw Tooth Wave"

import * as sim from "Simulator";

function simulateTags() {
 let counter = Date.now();
 let ts = Tags.CreateTagSet();
 ts.Add([['HMI_Tag_1', sim.sinWave(counter, 0.00, 10000, 25, 25)]]);
 ts.Add([['HMI_Tag_2', sim.sawTooth(counter, 0.25, 37000, 30, 15)]]);
 ts.WriteAsync();
}
```

Event

```
//JEx: "Event Screen_1 OnLoad"
//SOM_OM_"HmiTrendControl"
//JExRequired: "Generate signals"

export function Screen_1_OnLoad(item) {
 HMIRuntime.Timers.SetInterval(simulateTags, 500);
}
```

See also

Notes on the code examples (Page 370)

7.5.14 Monitoring alarms (RT Uni)

Introduction

This example shows how to create and monitor active alarms.

The reason (NotificationReason) for which the alarms were sent can have the following values:

- `Unknown` (1)
- `Add` (1)
  The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring, for example "State = 1".

- `Modify` (2)
  Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- `Remove` (3)
  The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

Note

Changes to the alarm will not result in notifications until the alarm meets the filter criteria again, such as "State = 1". In this case, "NotificationReason" is set to `Add`.

State-based and event-based monitoring

The respective client application determines whether or not notifications of alarms with the "NotificationReason" `Modify` or `Remove` are ignored.

For example:

- State-based monitoring: A customer wants to create a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
- Event-based monitoring: If an alarm is received, an email should be sent. Only the notification reason `Add` is relevant.

Execution of example

1. Configure a button (in the example "Button_1") on a screen.
2. Create a script for the event "Click left mouse button".
   The JavaScript editor creates the function "`Button_1_OnTapped(item, x, y, modifiers, trigger)`".
3. Insert the example code.
4. Compile and load it in Runtime.
5. Trigger the event "Click left mouse button" on the button.

Result

1. A customer begins monitoring with the filter criterion "State = 1".
2. An alarm is triggered. Runtime notifies the customer of the "NotificationReason" as follows:

NotificationReason	Description
Add	"State" is 1 The alarm has arrived.
Modify	"State" property has not changed. Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	"State" has changed. The alarm is removed.

Sample code

```
//JEx: "A client starts an alarm subscription with filter criterion “State = 1” (Raised
alarms)."
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
var subs = HMIRuntime.Alarming.CreateSubscription();
subs.Filter = 'State=1';
subs.Language = 1033;
subs.OnAlarm = function(Errorcode, SystemNames, ResultSet) {
  for (let index in ResultSet)
  {
    HMIRuntime.Trace('Alarm Name_' + (index+1) + ' = ' + ResultSet[index].Name);
    HMIRuntime.Trace(' Alarm State_' + (index+1) + '= ' + ResultSet[index].State);
    HMIRuntime.Trace(' Notification Reason_' + (index+1) + '= ' +
ResultSet[index].NotificationReason);
  }
}
subs.Start();
}
```

7.5.15 Using tag values globally (RT Uni)

Introduction

The example shows how to use tag values globally. A tag value can therefore be shared between screens and tasks, for example.

The basic procedure is as follows:

1. The tag value is written into an internal tag via a script.
2. The tag value is read by another script at the desired place.

You can save and read the values of all data types supported by the object model in internal tags.

Execution of example

The following example writes a structured value of a JavaScript tag to an internal tag. Another member is added to the structured value in the sample code on button 2.

1. Configure an HMI tag "Tag".
2. Configure two buttons (in the example "Button_1" and "Button_2") on a screen.
3. Create a script for each "Click left mouse button" event of the buttons.
   The JavaScript editor creates the functions "`Button_1_OnTapped(item, x, y, modifiers, trigger)`" und `Button_2_OnTapped(item, x, y, modifiers, trigger).`
4. Insert the sample code "Write structured value into internal tag" into the script of the first button.

5. Insert the sample code "Enhance structure by member "c"" into the script of the second button.
6. Compile and load it in runtime.
7. Trigger the event "Click left mouse button" on both buttons.

Result

Button 1

1. The JavaScript tag "tag" is created and initialized with the HMI tag "Tag".
2. The JavaScript tag "myObj" is created and initialized.
3. The JavaScript tag "myObj" is converted into a JSON string and assigned to the JavaScript tag "json".
4. The value of the JavaScript tag "json" is written into the HMI tag "Tag".

Button 2

1. The JavaScript tag "tag" is created and initialized with the HMI tag "Tag".
2. The JavaScript tag "myObj" is created and initialized.
3. The JavaScript tag "myObj" is extended by the member "c".
4. The JavaScript tag "myObj" is converted into a JSON string and assigned to the JavaScript tag "json".
5. The value of the JavaScript tag "json" is written into the HMI tag "Tag".

Sample code

Write structured value into internal tag

```
//JEx: "set initial values without 'c'"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
let tag = Tags('Tag');
let myObj = {a:10, b:20, pos: {x:100, y:200}, layers: [1, 8, 18, 24,
33]};
let json = JSON.stringify(myObj);
HMIRuntime.Trace('Jason ' + json);
tag.Write(json, 1);
}
```

Enhance structure by member "c"

```
//JEx: "add member 'c'"
export function Button_2_OnTapped(item, x, y, modifiers, trigger) {
const tag = Tags('Tag');
let myObj = JSON.parse(tag.Read(1));
myObj.c = (myObj.c || 0) + 1; // increment 'c'
let json = JSON.stringify(myObj);
HMIRuntime.Trace("New JSON: " + json);
tag.Write(json, 1);
}
```

7.6 Troubleshooting (RT Uni)

7.6.1 RTIL Trace Viewer (RT Uni)

Core statement

The RTIL Trace Viewer is a separate application which runs independently of the TIA Portal, but which can be integrated into the TIA Portal as an "external application".

Principle

During runtime, the RTIL Trace Viewer displays all alarms which are listed in the configurable TraceCatalog.

Layout

The traces are displayed in tabular form and can be sorted in ascending and descending order by the columns displayed.

Filters

The required alarms can be filtered using filters. Alarms in non-selected categories are hidden.

File functions

You export alarms as trace logs in the following formats:

- Text file (.txt, .log): Loading and evaluation in the RTIL Trace Viewer
- CSV file: Evaluation in conventional spreadsheet programs or other CSV-compatible applications

See also

Support for errors (Page 359)

7.6.2 Integrate RTIL Trace Viewer as an external application (RT Uni)

Procedure

1. Open the settings via "Tools > Settings".
2. Open the category "External applications".
3. Double-click in the first empty line.
   An input mask for the external application opens.
4. Assign a descriptive name for the application, e.g. "RTIL Trace Viewer" in the field "Name".
5. Insert the following path under "Command": `%ProgramFiles%\Siemens\Automation\WinCCUnified\bin\RTILtraceViewer.exe`
6. Leave the fields "Arguments" and "Start in" empty.
7. Click "Add" and then close the "Settings" dialog.

The application is now available via the menu "Tools > External applications".

7.6.3 Tracing with the RTIL Trace Viewer (RT Uni)

There is a "Trace Viewer" for finding errors in the JavaScript code.

Preparation

The "Trace Viewer" are located in the following path:

- %ProgramFiles%\Siemens\Automation\WinCCUnified\bin\RTILtraceViewer.exe

Requirement

- Simulation or runtime are started.
- RTILtraceViewer.exe has been started.

Procedure

1. Carry out an action in the simulation which starts a JavaScript function.
2. Filter by the trace messages "Subsystem > ScriptFW".

Note

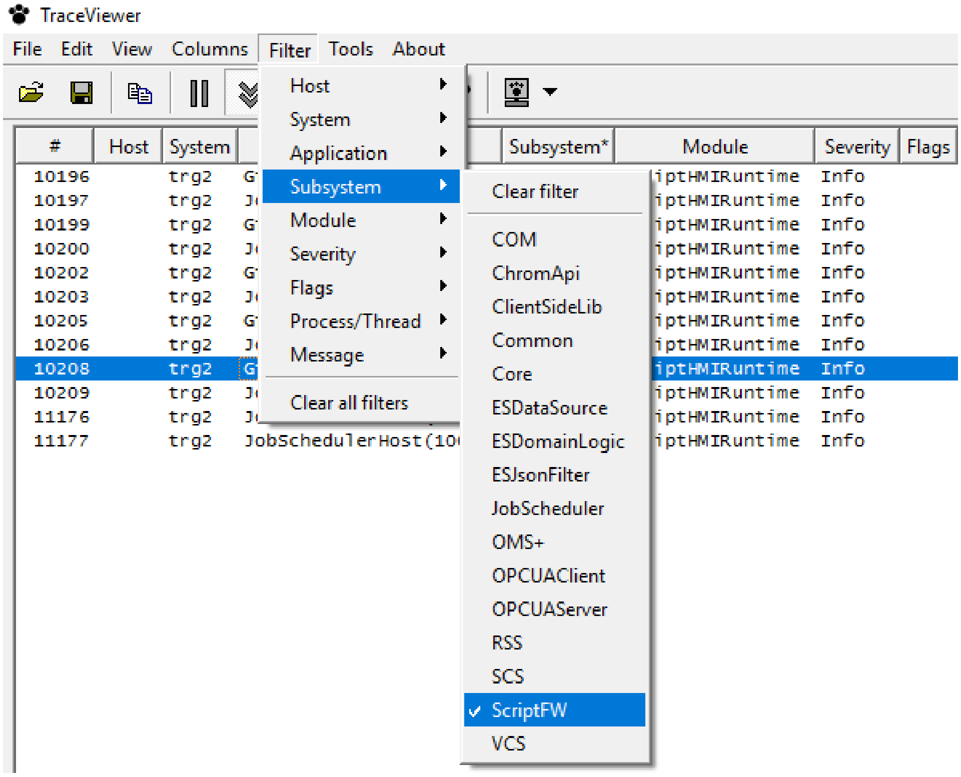
As long as no JavaScript function was executed in the simulation during the runtime of the Trace Viewer, the entry "ScriptFW" is missing in the "Filter > Subsystem" menu.

3. Define additional filter criteria if required.
4. Evaluate trace messages if actions in the simulation lead to errors.

Note

If no messages are displayed in the Trace Viewer despite errors in the simulation, reset the filters:

- Only in the submenu, e.g. "Subsystem": "Filter > Subsystem > Clear filter" menu
- All filters: "Filter > Clear all filters" menu



#	Host	System		Subsystem*	Module	Severity	Flags
10196		trg2	G		iptHMIRuntime	Info	
10197		trg2	J		iptHMIRuntime	Info	
10199		trg2	G		iptHMIRuntime	Info	
10200		trg2	J		iptHMIRuntime	Info	
10202		trg2	G		iptHMIRuntime	Info	
10203		trg2	J		iptHMIRuntime	Info	
10205		trg2	G		iptHMIRuntime	Info	
10206		trg2	J		iptHMIRuntime	Info	
10208		trg2	G		iptHMIRuntime	Info	
10209		trg2	J		iptHMIRuntime	Info	
11176		trg2	J		iptHMIRuntime	Info	
11177		trg2	JobSchedulerHost(10		iptHMIRuntime	Info	

See also

Integrate RTIL Trace Viewer as an external application (Page 398)

7.7 Debugging scripts (RT Uni)

7.7.1 Basics of debugging (RT Uni)

Introduction

For example, you can use a debugger to test whether correct values are being transferred to tags and whether abort conditions are being correctly implemented. Check the following in the debugger:

- Source code of functions
- Function sequence
- Values

Note

Your code is displayed in the debugger, but is write-protected.

Basic procedure

To find an error, check the script with the debugger.

The following options are available for your support:

- Setting breakpoints
- Step-by-step execution
- Viewing values parallel to execution of the script

You do not edit the code of your scripts directly in the debugger. When you find an error, follow these steps:

1. Correct the error in the engineering system.
2. Compile the changed code.
3. Load the runtime.
4. Update the debugger.

See also

Starting the debugger (Page 404)

7.7.2 Design and function of the debugger (RT Uni)

Google Chrome provides the user interface of the debugger. Not all functions of the user interface of the debugger are relevant for debugging WinCC Unified scripts. Only the functions that are needed to debug scripts in WinCC Unified are explained below.

You can find more information on Chrome DevTools under: https://developers.google.com/web/tools/chrome-devtools/ (https://developers.google.com/web/tools/chrome-devtools/).

The debugger is divided into two areas:

- Debugger for screens
- Debugger for jobs

With the debugger for screens you view scripts at screens and screen objects. With the debugger for jobs, you view scripts that you have configured in the Scheduler.

Start page of the debugger

After the debugger has been started, its start page is displayed.

The available contents differ depending on the selected area.

On the start page of the debugger for screens you can see two different contexts:

- Dynamizations (e.g. "UMCadmin@192.168.116.144 VCS_1 Dynamics")
- Events (e.g. "UMCadmin@192.168.116.144 VCS_1 Events")

The name of the contexts is composed as follows:

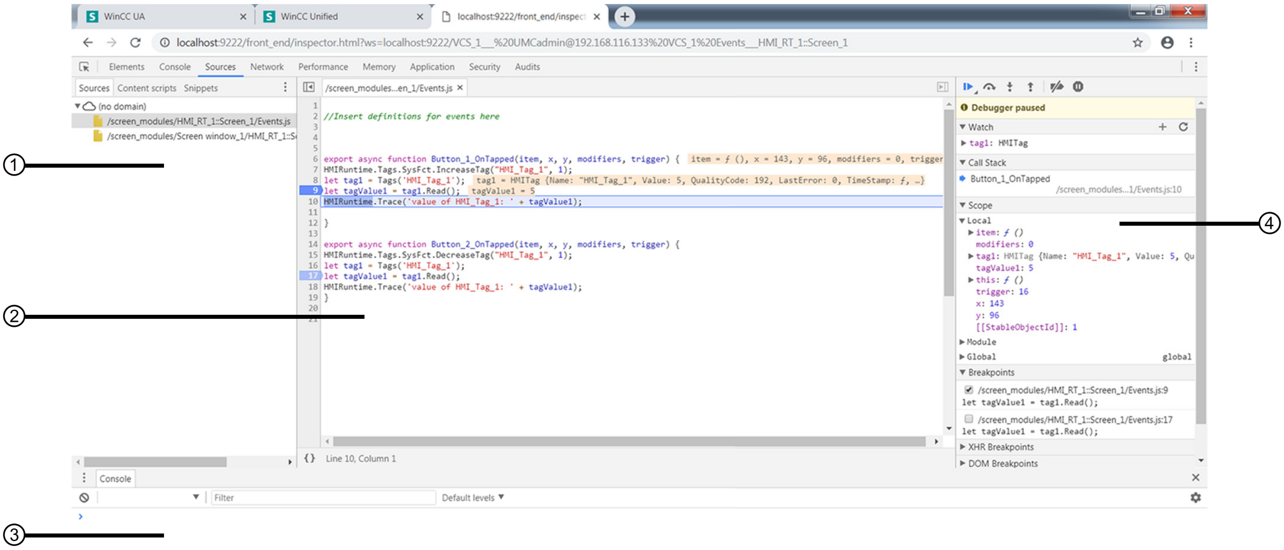
- UMCadmin: User name
- 192.168.116.144: IP address of the computer
- VCS: Name of the graphic component
- _1: Number of the open client
- Events/Dynamics: Scripts at events or dynamizations

Note

A client corresponds to a tab in Google Chrome in which the runtime is open. When you have opened runtime in multiple tabs, multiple clients are used. The client opened first is given the number 1. Numbering is reset when the runtime is restarted.

On the start page of the debugger for jobs you can see the context "JobsExecution".

User interface of the debugger


WinCC UA
WinCC Unified
localhost:9222/front_end/inspect
localhost:9222/front_end/inspector.html?ws=localhost:9222/VCS_1__%20UMCadmin@192.168.116.133%20VCS_1%20Events__HMI_RT_1::Screen_1
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
Sources
Content scripts
Snippets
(no domain)
/screen_modules/HMI_RT_1::Screen_1/Events.js
/screen_modules/Screen window_1/HMI_RT_1::S
/screen_modules...en_1/Events.js
Line 10, Column 1
Console
Filter
Default levels
Debugger paused
Watch
Call Stack
Scope
Local
Module
Global
Breakpoints
XHR Breakpoints
DOM Breakpoints
1
2
3
4

```
//Insert definitions for events here


export async function Button_1_OnTapped(item, x, y, modifiers, trigger) {  item = ƒ (), x = 143, y = 96, modifiers = 0, trigger
HMIRuntime.Tags.SysFct.IncreaseTag("HMI_Tag_1", 1);
let tag1 = Tags('HMI_Tag_1');  tag1 = HMITag {Name: "HMI_Tag_1", Value: 5, QualityCode: 192, LastError: 0, TimeStamp: ƒ, …}
let tagValue1 = tag1.Read();  tagValue1 = 5
HMIRuntime.Trace('value of HMI_Tag_1: ' + tagValue1);

}

export async function Button_2_OnTapped(item, x, y, modifiers, trigger) {
HMIRuntime.Tags.SysFct.DecreaseTag("HMI_Tag_1", 1);
let tag1 = Tags('HMI_Tag_1');
let tagValue1 = tag1.Read();
HMIRuntime.Trace('value of HMI_Tag_1: ' + tagValue1);
}
```

```
tag1: HMITag
Button_1_OnTapped
    /screen_modules...1/Events.js:10
item: ƒ ()
modifiers: 0
tag1: HMITag {Name: "HMI_Tag_1", Value: 5, Qu
tagValue1: 5
this: ƒ ()
trigger: 16
x: 143
y: 96
[[StableObjectId]]: 1
Global                    global
☑ /screen_modules/HMI_RT_1::Screen_1/Events.js:9
let tagValue1 = tag1.Read();
□ /screen_modules/HMI_RT_1::Screen_1/Events.js:17
let tagValue1 = tag1.Read();
```

①	Navigation area
②	Code display area
③	Console
④	Debugging area

Navigation area

In the navigation area, the available contents for the screen shown in runtime are displayed in groups. The available groups vary depending on the use of scripts and functions.

Groups in the debugger for screens

The debugger for screens can contain the following groups in the dynamizations context:

- A group for scripts that were configured for dynamizations.
- One group per screen window in which scripts were configured for dynamizations.

The debugger for screens can contain the following groups in the events context:

- A group for scripts that were configured for events.
- One group for functions that were configured for events using the function list.
- One group per screen window in which scripts were configured for events.
- One group per screen window in which functions were configured for events using the function list.

Groups in the debugger for jobs

The debugger for jobs can contain the following groups:

- A group for scripts that were configured for tasks.
- One group for functions that were configured for tasks using the function list.

Code display area

Your code is displayed in the code display area. The rows are numbered.

Debugging area

The debugging area offers the following relevant options for WinCC Unified:

- Toolbar: Control for executing the script
- "Watch": Display of values
- "Callstack": Display of the current call stack
- "Scope": Available local values ("Local"), functions ("Module") and global values ("Global"),
- "Breakpoints": List of set breakpoints

7.7.3 Enabling the debugger (RT Uni)

Requirement

- SIMATIC Runtime Manager is installed.
- The logged-on user has administrator rights.

Note

The debugger is only available locally.

Remote access from the debugger to other devices is not possible.

Procedure

The debugger is disabled by default.

Note

The debugger should be deactivated in production operation, as using the debugger can endanger system stability and security. Actions can accumulate if the debugger is, for example, at a breakpoint for a long time or the screen is not refreshed.

To activate the debugger, follow these steps:

1. Run the SIMATIC Runtime Manager application with administrator rights.
2. Click ⚙ "Settings of SIMATIC Runtime Manager".
3. Open the "Script Debugger Settings" tab.
4. To enable the debugger for screens, select the "Enable" check box in the "Screen Debugger" area.
5. To enable the debugger for jobs, select the "Enable" check box in the "Scheduler Debugger" area.
6. Assign an available port number to the debugger for screens (default port number: 9222).
7. Assign an available port number to the debugger for jobs (default port number: 9224).
8. Click "Save".

Note

Start the runtime after enabling the debugger.

7.7.4 Starting the debugger (RT Uni)

Requirement

- Google Chrome (version 55 or higher) is installed.
- A project is opened in runtime.

Procedure

- To start the debugger, call the URL http://localhost:*port number* (e.g. http://localhost:9222) in Google Chrome. Instead of the term "localhost", you can use the computer name.

You access the debugger user interface as follows:

1. Select the desired context.
2. Open the "Sources" tab.
   The user interface of the debugger is displayed.

Updating the debugger

Updating the debugger is necessary when

- You reload the runtime.
- You execute a screen change in runtime.

The connection to the debugger is lost in both cases. Google Chrome therefore shows an error message and asks whether you want to restore the connection.

- To restore the connection, click "Reconnect DevTools".

Note

If the start page of the debugger is displayed while you are restarting Runtime or change the screen in Runtime, update the page display in Google Chrome.

Stopping the debugger

- You stop the debugger by closing the respective tab in Google Chrome.

This does not stop runtime.

See also

Design and function of the debugger (Page 401)

7.7.5 Working with breakpoints (RT Uni)

Set breakpoints to stop the execution of the script at certain points and thus localize errors step-by-step. Previously set breakpoints are still available after updating the debugger.

Requirement

- Runtime has started.
- The debugger has been started.
- The group you want to debug is selected.

Pause script

To pause the execution of a script, you have 2 options:

- To pause the script immediately, click the ⏸ "Pause script execution" button while the script is being executed.
- Set a breakpoint in the desired line.
  The script pauses when a breakpoint is reached.

To pause a script at a breakpoint that is configured to an event, follow these steps:

1. Set a breakpoint in the script.
2. Trip the respective event in runtime.
   The script pauses at the breakpoint.

Setting breakpoints

You have several options to set a breakpoint in a line of the script:

- Click on the line number.
- Right-click the line number and select "Add Breakpoint".

All set breakpoints are displayed in the debugging area under "Breakpoints".

Linking breakpoints to conditions

To link a breakpoint to a condition, proceed as follows:

1. Open the shortcut menu of the relevant line.
2. Select the entry "Add conditional breakpoint".
   Execution of the script is stopped at the breakpoint when the condition is fulfilled.

Edit conditions as follows:

1. Open the shortcut menu of the relevant line.
2. Select the entry "Edit breakpoint...".

To prevent the script from pausing at a selected line, proceed as follows:

1. Open the shortcut menu of the respective line.
2. Select the entry "Never pause here".

Showing and hiding breakpoints

When you hide a breakpoint, its position is retained. The script then ignores the hidden breakpoint. When you need the breakpoint again, it can simply be shown.

In the debugging area, all breakpoints set in the selected group are displayed under "Breakpoints".

You have several options to show a breakpoint:

- Set the check mark in front of the relevant breakpoint in the debugging area under "Breakpoints".
- Alternatively, right-click the number of the respective line in the code display area and then select "Enable breakpoint".

You have several options to hide a breakpoint:

- Remove the check mark in front of the relevant breakpoint in debugging area under "Breakpoints".
- Alternatively, right-click the number of the respective line in the code display area and then select "Disable breakpoint".

To show or hide all breakpoints, follow these steps:

1. Open the shortcut menu in the debugging area under "Breakpoints".
2. Select "Enable all breakpoints" or "Disable all breakpoints"

Enabling and disabling breakpoints

You can enable or disable all breakpoints independent of showing or hiding individual breakpoints.

You have several options to enable or disable all breakpoints:

- Click on the "Deactivate breaktpoints" button in the debugging area.
- Open the shortcut menu of a breakpoint in the debugging area and select "Activate breakpoints" or "Deactivate breakpoints".
- Press <Ctrl + F8>.

Deleting breakpoints

You have several options to delete a breakpoint:

- Click on the breakpoint in the code display area.
- Open the shortcut menu of the breakpoint in the code display area and select "Remove breakpoint".
- Open the shortcut menu in the debugging area under "Breakpoints" and select "Remove breakpoint"..

To delete breakpoints, the shortcut menu offers the following additional options in the debugging area under "Breakpoints":

- Delete all breakpoints ("Remove all breakpoints")
- Delete all breakpoints except the selected breakpoint ("Remove other breakpoints")

7.7.6 Step-by-step execution of scripts (RT Uni)

Introduction

The following options are available to execute your script step-by-step:

- Execute script to the next breakpoint
- Force execution of a script
- Execute script to the next function call
- Jump into a function
- Jump out of a function
- Execute script up to a selected line
- Pause at Exceptions
- Use call stack

Requirement

- The group you want to debug is selected.
- The script pauses at a breakpoint.

Execute script to the next breakpoint

To pause the continuation of a script, you have several options:

- Click on the ▶ "Resume script execution" button in the debugging area.
- Press the <F8> key.
  The script is executed to the next breakpoint. If there is no other breakpoint, the script is executed completely.

Force execution of a script

To ignore the following breakpoints when resuming execution of a paused script, proceed as follows:

1. Click and hold down the ▶ "Resume script execution" button.
   The ▶ "Force script execution" button appears.
2. Move the mouse pointer to the ▶ "Force script execution" button while keeping the mouse button pressed.
3. Now release the mouse button.
   The script is executed to the end.

Execute script to the next function call

If a line with a breakpoint contains a function that you are not interested in, you can suppress the debugging of this function:

- Click on the ↷ "Step over next function call" button in the debugging area.
- Press the <F10> key.
  The function is executed without the script pausing within the function.

Jumping into a function

If the script pauses in a line containing a function that interests you, you can pause the script in that function:

- Click on the ⇣ "Step into next function call" button in the debugging area.
- Press the <F11> key.
  The script pauses in the first line of the function.

Note

You can only jump into functions that you have defined yourself.

Jump out of a function

If the script pauses within a function that you are not interested in, you can suppress further debugging of this function:

- Click on the ↥ "Step out of current function" button in the debugging area.
- Press the key combination <Shift + F11>.

Note

You can only jump out of a function that you have defined yourself.

Execute script up to a selected line

To pause a paused script again at a selected line, proceeds as follows:

1. Right-click the number of the line in the code display area.
2. Select the entry "Continue to here".
   The script pauses at the selected line.

Pause at Exceptions

- To pause the script at Exceptions, click on the ⏸ "Pause on exceptions" button in the debugging area.

Use call stack

- To jump into a function of the call stack, click on the corresponding entry under "Call Stack".

Note

You can only jump into functions that you have defined yourself.

7.7.7 Show values (RT Uni)

Introduction

To identify errors in your script efficiently, have current values displayed while the script is being executed. This way you can view properties of objects or parameters of functions, for example. You can find additional information on objects and their properties under "WinCC Unified Object Model".

Requirements

- The group you want to debug is selected.
- The script pauses at a breakpoint.

Procedure

You view values by moving the mouse over the label in the code display area.

You also have the following options to view values:

- In the debugging area under "Scope"
- In the debugging area under "Watch"
- In the console

"Scope" area

All local values ("Local"), functions ("Module") and global values ("Global") that are defined at this time are displayed in the "Scope" area.

The values cannot be edited.

"Watch" area

In the "Watch" area, you view how values change in the course of a script.

The following buttons are available to you:

- + "Add expression": Add a value
- ↻ "Refresh": Refresh the "Watch" area
- ▬ "Delete watch expression": Delete a value from the "Watch" area. Available when the mouse pointer is located above the respective value.

Console

The values available at the current time can be called in the console.

- You show or hide the console with <Esc>.

Call the current values in the console as follows:

1. Enter the name of a local or global value in the console.
2. Press <Enter>.

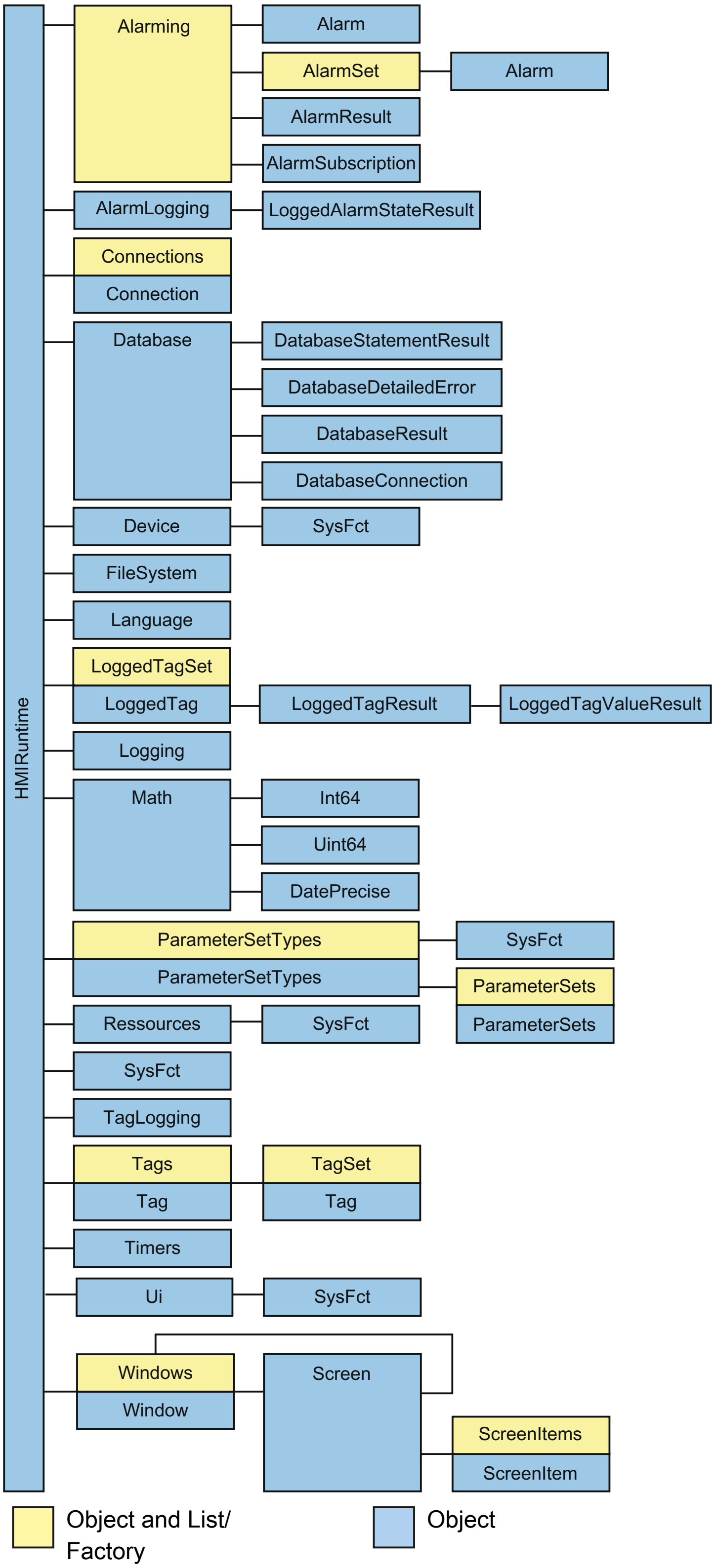
See also

WinCC Unified object model (Page 411)

7.8 WinCC Unified object model (RT Uni)

Object model

The figure below shows an overview of the object model of the graphical Runtime system of WinCC Unified:



You access the objects of the graphical runtime system through the object model.

Use

You use the object model as follows:

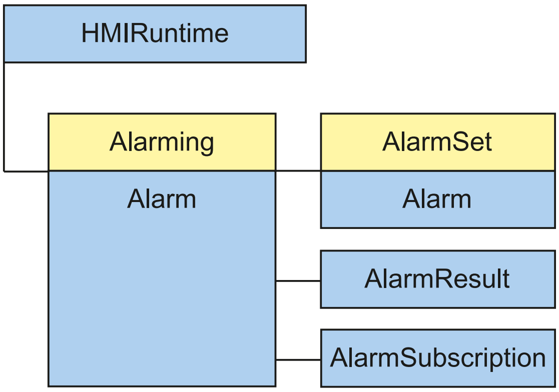
- Objects
  Objects and lists give you access to basic elements of the runtime system, for example, tags, screens and levels.
- Properties
  You use the properties to read the current status of individual objects, for example, the name of a tag. You can also change many properties of the objects directly, for example, enable a button.
- Methods
  You apply methods to individual objects and write, for example, tag values back to the AS or output alarms in runtime.

7.8.1 Objects (RT Uni)

7.8.1.1 "Alarming" area (RT Uni)

"Alarming" object (RT Uni)

Description



The "Alarming" object ("HMIAlarming" type) gives you access to the WinCC Unified alarm system. You can create a new "AlarmSet" list, compile active alarms ("AlarmSubscription" objects) and reference configured alarms ("Alarm" objects) for read and write access.

Type identifier in JavaScript

HMIAlarming

See also

"AlarmSet" description (Page 417)

"CreateAlarmSet" method (Alarming.CreateAlarmSet) (Page 415)

"CreateSubscription" method (Alarming.CreateSubscription) (Page 416)

"Alarms" method (Alarming.Alarms) (Page 414)

Properties (RT Uni)

Properties

The "Alarming" object has the following properties:

Properties	Type	Access	Description
SysFct	Object	read only	Returns the "SysFct" object.

Methods of "Alarming" (RT Uni)

Overview (RT Uni)

Methods

The "Alarming" object has the following methods:

Methods	Description
Alarms	Returns an "Alarm" object (type "HMIAlarm").
CreateAlarmSet	Creates a new "AlarmSet" object (type "HMIAlarmSet").
CreateSubscription	Creates an "AlarmSubscription" object (type "HMIAlarmSubscription").
GetActiveAlarms	Supplies all active alarms at the time of the call.

"Alarms" method (Alarming.Alarms) (RT Uni)

Description

Returns an "Alarm" object (type "HMIAlarm"). With the "Alarm" object, you can execute operations with an alarm, e.g. acknowledgment or commenting.

Member

Method of the "Alarming" object

Syntax

```
HMIRuntime.Alarming.Alarms(AlarmName);
```

Parameters

`AlarmName`

Type: String

Alarm name

Return value

Object of the type "HMIAlarm"

See also

"Alarming" object (Page 413)

"CreateAlarmSet" method (Alarming.CreateAlarmSet) (RT Uni)

Description

Creates a new "AlarmSet" object (type "HMIAlarmSet"). With the returned "AlarmSet" object, you can execute operations with multiple alarms in one call, e.g. acknowledgment or commenting.

Member

Method of the "Alarming" object

Syntax

```
HMIRuntime.Alarming.CreateAlarmSet([AlarmNames]);
```

Parameters

`AlarmNames`

Optional, type: String, String[]

Name of one or multiple active alarms that are added to the "AlarmSet" object. Without parameters, an empty "AlarmSet" object is created.

Return value

Object of the type "HMIAlarmSet"

See also

"Alarming" object (Page 413)

"AlarmSet" description (Page 417)

"CreateSubscription" method (Alarming.CreateSubscription) (RT Uni)

Description

Creates an "AlarmSubscription" object (type "HMIAlarmSubscription"). With the returned "AlarmSubscription" object, you specify the grouping of active alarms.

Member

Method of the "Alarming" object

Syntax

```
HMIRuntime.Alarming.CreateSubscription();
```

Parameters

--

Return value

Object of the type "HMIAlarmSubscription"

See also

"Alarming" object (Page 413)

"GetActiveAlarms" method (Alarming.GetActiveAlarms) (RT Uni)

Description

Supplies all active alarms at the time of the call. Unlike with an AlarmSubscription, no status changes or new alarms are signaled that occur after the function call.

Users can filter the alarms or specify a SystemName if they only want to receive the active alarms of a specific system.

Member

Method of the "Alarming" object

Syntax

```
PromiseParameterTagSet
HMIRuntime.Alarming.GetActiveAlarms(Language, Filter, SystemNames);
```

Parameters

`Language`

Type: UInt32

Language for all texts of an alarm and the filter

`Filter`

Optional, type: String

SQL-type string for filtering

`SystemNames`

Optional, type: String[]

A string array with the SystemNames of the systems by which its alarms are to be filtered

Return value

`Promise`

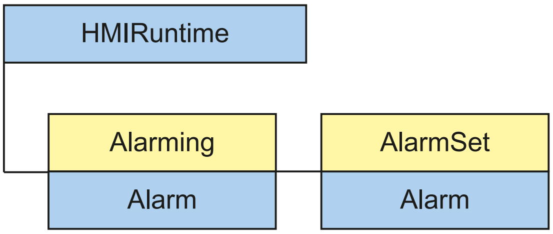
Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMIAlarmResult[]"
- Promise rejected
  Error code as parameter of the "catch()" handler.

"AlarmSet" object (RT Uni)

"AlarmSet" description (RT Uni)

Description



The "AlarmSet" object ("HMIAlarmSet" type) is a list of "Alarm" objects that give you optimized access to active alarms in runtime. After initialization of the "AlarmSet" object, you can execute operations with multiple alarms in one call, e.g. acknowledgment or commenting. Simultaneous access demonstrates better performance and lower communication load than single access to multiple alarms.

You create a new "AlarmSet" object with the "Alarming.CreateAlarmSet" method.

Type identifier in JavaScript

HMIAlarmSet

See also

"Alarming" object (Page 413)

"CreateAlarmSet" method (Alarming.CreateAlarmSet) (Page 415)

"Add" method (AlarmSet.Add) (Page 420)

"Remove" method (AlarmSet.Remove) (Page 424)

"Disable" method (Alarm.Disable, AlarmSet.Disable) (Page 421)

"Enable" method (Alarm.Enable, AlarmSet.Enable) (Page 422)

"Shelve" method (Alarm.Shelve, AlarmSet.Shelve) (Page 426)

"Unshelve" method (Alarm.Unshelve, AlarmSet.Unshelve) (Page 427)

"Acknowledge" method ((Alarm.Acknowledge, AlarmSet.Acknowledge) (Page 419)

"Reset" method (Alarm.Reset, AlarmSet.Reset) (Page 425)

"Item" method (AlarmSet.Item) (Page 423)

Properties (RT Uni)

Properties

The "AlarmSet" object has the following properties:

Properties	Type	Access	Description
Count	UInt32	read only	Returns the number of elements in the specified list.

Methods of "AlarmSet" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmSet" object has the following methods:

Methods	Description
Acknowledge	Acknowledges active alarms.
Add	Adds alarms to the "AlarmSet" list.
Disable	Temporarily deactivates the generation of alarms in the alarm source.
Enable	Re-enables deactivated alarms for display.
Item	Returns an "Alarm" object of the "AlarmSet" list.
Remove	Removes an alarm by its name from the "AlarmSet" list.
Reset	Acknowledges the outgoing state of an active alarm.
Shelve	Hides active alarms.
Unshelve	Makes hidden active alarms visible again.

"Acknowledge" method ((Alarm.Acknowledge, AlarmSet.Acknowledge) (RT Uni)

Description

Acknowledges active alarms. The method uses the entire "Alarm" object as reference. If you want to acknowledge individual instances of an active alarm using the InstanceID, use the "AcknowledgeInstance" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
```
HMIRuntime.Alarming.Alarm.Acknowledge();
```
- For objects of the type "HMIAlarmSet" (asynchronous):
```
HMIRuntime.Alarming.AlarmSet.Acknowledge()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

"AlarmSet" description (Page 417)

"Add" method (AlarmSet.Add) (RT Uni)

Description

Adds alarms to the "AlarmSet" list. The alarms are referenced by the name.

Member

Method of the "AlarmSet" object

Syntax

```
HMIRuntime.Alarming.AlarmSet.Add(AlarmName);
```

Parameters

`AlarmName`

Type: String or String[]

Names of "Alarm" objects that are added to the list.

The following data types are supported:

- Alarm name
- Array with tag names

Note

No "Alarm" object can be transferred as a parameter. An "Alarm" object is referenced using the name.

Return value

Array of objects of the "HMIAlarm" type

See also

"AlarmSet" description (Page 417)

"Disable" method (Alarm.Disable, AlarmSet.Disable) (RT Uni)

Description

Temporarily deactivates the generation of alarms in the alarm source. You can reactivate the generation of the alarms again with the "Enable" method.

You can use the "Disable" method to prevent the display of the alarms, for example, for maintenance work.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
```
HMIRuntime.Alarming.Alarm.Disable();
```
- For objects of the type "HMIAlarmSet" (asynchronous):
```
HMIRuntime.Alarming.AlarmSet.Disable()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

"AlarmSet" description (Page 417)

"Enable" method (Alarm.Enable, AlarmSet.Enable) (RT Uni)

Description

Re-enables deactivated alarms for display. You can temporarily deactivate the generation of the messages in the alarm source with the "Disable" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  ```
  HMIRuntime.Alarming.Alarm.Enable();
  ```
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.Enable()
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

"AlarmSet" description (Page 417)

"Item" method (AlarmSet.Item) (RT Uni)

Description

Returns an "Alarm" object of the "AlarmSet" list.

Member

Method of the "AlarmSet" object

Syntax

```
HMIRuntime.AlarmSet[.Item](name);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "AlarmSet" object.

Parameters

`name`

Type: String

Name or index number (1...n) of an "Alarm" object in the list

Note

The index number of an "Alarm" object does not represent the order in which the "Alarm" objects were added to the "AlarmSet" list.

Return value

Object of the type "HMIAlarm"

See also

"AlarmSet" description (Page 417)

"Remove" method (AlarmSet.Remove) (RT Uni)

Description

Removes an alarm by its name from the "AlarmSet" list.

Member

Method of the "AlarmSet" object

Syntax

```
HMIRuntime.Alarming.AlarmSet.Remove(AlarmName);
```

Parameters

`AlarmName`

Type: String or String[]

Name of "Alarm" objects that are removed from the "AlarmSet" list.

The following data types are supported:

- Alarm name
- Array with tag names

Note

No "Alarm" object can be transferred as a parameter. An "Alarm" object is referenced using the name.

Return value

ErrorCode

See also

"AlarmSet" description (Page 417)

"Reset" method (Alarm.Reset, AlarmSet.Reset) (RT Uni)

Description

Acknowledges the outgoing state of an active alarm. The alarms are removed from the alarm system. The method uses the entire "Alarm" object as reference. If you want to acknowledge individual instances of an active alarm using the InstanceID, use the "ResetInstance" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
```
HMIRuntime.Alarming.Alarm.Reset();
```
- For objects of the type "HMIAlarmSet" (asynchronous):
```
HMIRuntime.Alarming.AlarmSet.Reset()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

"AlarmSet" description (Page 417)

"Shelve" method (Alarm.Shelve, AlarmSet.Shelve) (RT Uni)

Description

Hides active alarms. These are no longer displayed by Alarm Control in runtime. You can show the alarms again with the "Unshelve" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
```
HMIRuntime.Alarming.Alarm.Shelve();
```
- For objects of the type "HMIAlarmSet" (asynchronous):
```
HMIRuntime.Alarming.AlarmSet.Shelve()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

"AlarmSet" description (Page 417)

"Unshelve" method (Alarm.Unshelve, AlarmSet.Unshelve) (RT Uni)

Description

Makes hidden active alarms visible again. These are once again displayed by the Alarm Control in runtime. You can hide the alarms again with the "Shelve" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  ```
  HMIRuntime.Alarming.Alarm.Unshelve();
  ```
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.Unshelve()
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

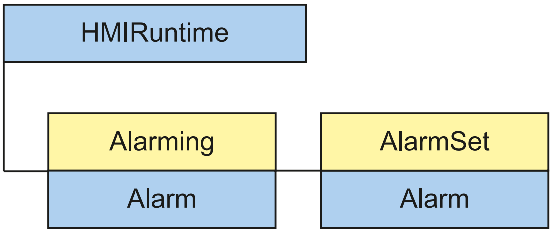
See also

"AlarmSet" description (Page 417)

"Alarm" object (RT Uni)

"Alarm" description (RT Uni)

Description



The "Alarm" object ("HMIAlarm" type) gives you access to the properties and methods of the active alarms. An "Alarm" object is returned by the "Alarming" or "AlarmSet" lists.

Type identifier in JavaScript

HMIAlarm

Properties (RT Uni)

Properties

The "Alarms" object has the following properties:

Properties	Type	Access	Description
ErrorCode	Error-Code	read/ write	Specifies the error code of the last method call of the object.
Name	String	read/ write	Returns the object name or specifies it
InstanceID	UInt32	read	Returns the instance ID of the alarm.

Methods of "Alarm" (RT Uni)

Overview (RT Uni)

Methods

The "Alarms" object has the following methods:

Methods	Description
Acknowledge	Acknowledges an alarm.
AcknowledgeInstance	Acknowledges an alarm instance.
Disable	Temporarily deactivates the generation of alarms in the alarm source.
Enable	Re-enables deactivated alarms for display.
Reset	Acknowledges the outgoing state of an alarm.
ResetInstance	Acknowledges the outgoing state of an alarm instance.
Shelve	Hides active alarms.
Unshelve	Makes hidden active alarms visible again.

"Acknowledge" method ((Alarm.Acknowledge, AlarmSet.Acknowledge) (RT Uni)

Description

. The method uses the entire "Alarm" object as reference.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  `HMIRuntime.Alarming.Alarm.Acknowledge();`
- For objects of the type "HMIAlarmSet" (asynchronous):

```
HMIRuntime.Alarming.AlarmSet.Acknowledge()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"AcknowledgeInstance" method (Alarm.AcknowledgeInstance, AlarmSet.Acknowledge.Instance) (RT Uni)

Description

Acknowledges an alarm instance. The method uses the entire "Alarm" object as reference.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  ```
  HMIRuntime.Alarming.Alarm.AcknowledgeInstance(InstanceID);
  ```
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.AcknowledgeInstance(InstanceID)
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

InstanceID

Type: UInt32

Number of the alarm instance.

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"Disable" method (Alarm.Disable, AlarmSet.Disable) (RT Uni)

Description

Temporarily deactivates the generation of alarms in the alarm source. You can reactivate the generation of the alarms again with the "Enable" method.

You can use the "Disable" method to prevent the display of the alarms, for example, for maintenance work.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  `HMIRuntime.Alarming.Alarm.Disable();`
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.Disable()
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"Enable" method (Alarm.Enable, AlarmSet.Enable) (RT Uni)

Description

Re-enables deactivated alarms for display. You can temporarily deactivate the generation of the messages in the alarm source with the "Disable" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  ```
  HMIRuntime.Alarming.Alarm.Enable();
  ```
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.Enable()
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"Reset" method (Alarm.Reset, AlarmSet.Reset) (RT Uni)

Description

Acknowledges the outgoing state of an active alarm. The alarm is removed from the alarm system. The method uses the entire "Alarm" object as reference. If you want to acknowledge individual instances of an active alarm using the InstanceID, use the "ResetInstance" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  ```
  HMIRuntime.Alarming.Alarm.Reset();
  ```
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.Reset()
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"ResetInstance" method (Alarm.ResetInstance, AlarmSet.ResetInstance) (RT Uni)

Description

Acknowledges the outgoing state of an alarm instance. The alarm instance is removed from the alarm system. The method uses the entire "Alarm" object as reference. If you want to acknowledge individual instances of an active alarm using the InstanceID, use the "ResetInstance" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  ```
  HMIRuntime.Alarming.Alarm.Reset();
  ```
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.Reset()
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"Shelve" method (Alarm.Shelve, AlarmSet.Shelve) (RT Uni)

Description

Hides active alarms. These are no longer displayed by Alarm Control in runtime. You can show the alarms again with the "Unshelve" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  ```
  HMIRuntime.Alarming.Alarm.Shelve();
  ```
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.Shelve()
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

--

Return value

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"Unshelve" method (Alarm.Unshelve, AlarmSet.Unshelve) (RT Uni)

Description

Makes hidden active alarms visible again. These are once again displayed by the Alarm Control in runtime. You can hide the alarms again with the "Shelve" method.

As member of the "Alarm" object, the method is executed as a synchronous operation.

As member of the "AlarmSet" object, the method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Member

Method of the following objects:

- Alarm
- AlarmSet

Syntax

The syntax of the method depends on the object:

- For objects of the type "HMIAlarm" (synchronous):
  ```
  HMIRuntime.Alarming.Alarm.Unshelve();
  ```
- For objects of the type "HMIAlarmSet" (asynchronous):
  ```
  HMIRuntime.Alarming.AlarmSet.Unshelve()
  .then(function(ErrorCode) {
      ...
  })
  .catch(function(ErrorCode) {
      ...
  });
  ```

Parameters

--

Return value

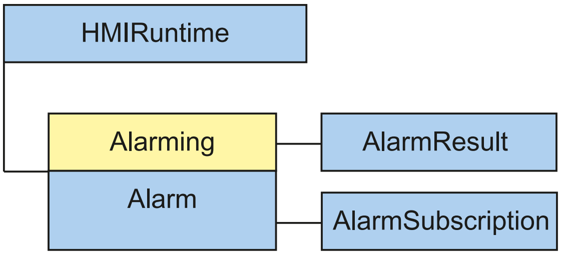
ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"AlarmSubscription" object (RT Uni)

"AlarmSubscription" description (RT Uni)

Description



The "AlarmSubscription" object (type "HMIAlarmSubscription") gives you access to active alarms.

Use

The "AlarmSubscription" object represents a selection of active alarms. An "AlarmSubscription" object is initialized through the "CreateSubscription" method of the "Alarming" object. Afterwards, the active alarms are grouped and called according to the properties of the "AlarmSubscription" object. Finally, notification is given of the changes to the alarm mapping

Type identifier in JavaScript

HMIAlarmSubscription

Properties (RT Uni)

Properties

The "AlarmSubscription" object has the following properties:

Properties	Type	Access	Description
Filter	String	read/ write	Specifies a string for filtering active alarms. The syntax of the filter string is equivalent to the WHERE clause of an SQL command.
Language	UInt32	read/ write	Specifies the current Runtime language.

Properties	Type	Access	Description
OnAlarm	Function	write only	Specifies the name of the "OnAlarm" callback function for monitoring active alarms. The properties of the active alarms are transferred as "AlarmResultArray" object to the "OnAlarm" callback function. Required prototype of the callback function: OnAlarm(errorCode, systemName, alarmResultArray)
SystemNames	String	read/write	Specifies the name of the Runtime system of type "HMI-System" for the grouping of active alarms.

Methods of "AlarmSubscription" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmSubscription" object has the following methods:

Methods	Description
Start	Activates the monitoring of defined alarms of the "AlarmSubscription" object.
Stop	Cancels the monitoring of defined alarms of the "AlarmSubscription" object.

"Start" method (AlarmSubscription.Start) (RT Uni)

Description

Activates the monitoring of defined alarms of the "AlarmSubscription" object.

Member

Method of the "AlarmSubscription" object

Syntax

```
HMIRuntime.AlarmSubscription.Start();
```

Parameters

--

Return value

ErrorCode

"Stop" method (AlarmSubscription.Stop) (RT Uni)

Description

Cancels the monitoring of defined alarms of the "AlarmSubscription" object.

Member

Method of the "AlarmSubscription" object

Syntax

```
HMIRuntime.AlarmSubscription.Stop();
```

Parameters

--

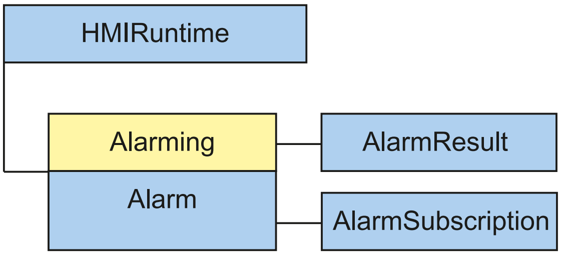
Return value

ErrorCode

"AlarmResult" object (RT Uni)

"AlarmResult" description (RT Uni)

Description



The "AlarmResult" object (type "HMIAlarmResult") gives you access to the properties of an active alarm. The "AlarmResult" object is a pure data object which maps all properties of an active alarm.

Use

Once an active alarm has been output, the "OnAlarm" callback function is called. The "AlarmResult" object together with the SystemName and ErrorCode are transferred as parameters to this function.

In the case of multiple active alarms, the parameters are transferred as lists to the "OnAlarm" callback function. Alarms from different servers are processed in a single call of the OnAlarm callback function.

The "AlarmResult" object only contains properties and no methods.

All texts of the "AlarmResult" object are monolingual strings. The language is specified with the "AlarmSubscription.Language" property.

Type identifier in JavaScript

HMIAlarmResult

Properties (RT Uni)

Properties

The "AlarmResult" object has the following properties:

Properties	Type	Access	Description
AcknowledgementTime	DateTime	read only	Returns the time of alarm acknowledgment.
AlarmClassName	String	read only	Returns the name of the alarm class of an alarm.
AlarmClassSymbol	String	read only	Returns the abbreviation for the display of the alarm class of the alarm, for example, "W" for the alarm class "Warning".
AlarmParameterValues	Variant	read only	Returns an array with parameter values of an alarm. The property is mapped in the "AlarmResult" object. The parameter values are added to an alarm from the alarm source in the alarm state "Incoming" and "Reset". They can also include diagnostic or raw data from the PLC in addition to simple tag values from the configured AlarmParamterTags.
AlarmText	String[]	read only	Returns the localized additional texts 1 to 9 of an alarm as an array. The text can contain triggered placeholders and reference all the "AlarmParameterValues" properties of the respective alarm state "Incoming" or "Reset".
Area	String	read only	Specifies the origin area of an alarm. The "Area" property can be configured and, together with the "Origin" property, defines the source of an alarm. You can also use placeholders for context-sensitive format. The "Area" property, for example, includes subsystem, application name or PLC ID. You can sort and filter alarms through the "Area" context.

Properties	Type	Access	Description
BackColor	UInt32	read only	Specifies the background color.
ChangeReason (Page 445)	UInt16	read only	Returns the trigger event for the modification of the alarm state.
ClearTime	DateTime	read only	Returns the time of alarm reset.
Connection	String	read only	Returns the name of the connection by which an alarm was triggered.
EventText	String	read only	Returns a localized text that describes an alarm event for the alarm. The text can contain triggered placeholders and reference all the "AlarmParameterValues" properties of the respective alarm state "Incoming" or "Reset".
Flashing (Page 445)	Bool	read only	Returns whether the specified object flashes in runtime.
InstanceID	UInt32	read only	Returns the ID of alarms with multiple instances.
InvalidFlags (Page 446)	UInt8	read only	Returns the cause of invalid data of an alarm.
LoopInAlarm	String	read only	Navigates from the display of an active alarm to its origin. The "LoopInAlarm" property includes the name of a function that leads to the origin of an alarm. The required parameters of the function are returned with the associated "LoopInAlarmParameterValues" property.
LoopInAlarmParameterValues	Variant	read only	Returns the parameters of the function that navigates from the display of an active alarm to its origin. The associated "LoopInAlarm" property contains the name of the function that is used for the call, for example the "OpenScreen" function when the origin of the alarm is a screen.
ModificationTime	DateTime	read only	Returns the time stamp of the last modification of the alarm state. The reason for the change is included in the "ChangeReason" property.
Name	String	read only	Returns the name of the object or specifies it.
NotificationReason (Page 446)	UInt8	read only	Returns the reason for an alarm.
Origin (Page 447)	String	read only	Returns the origin of an alarm.
Priority	UInt8	read only	Specifies the relevance of an alarm or a machine status.
RaiseTime	DateTime	read only	Returns the trigger time of an alarm.
ResetTime	DateTime	read only	Returns the time of alarm reset. After the reset, the alarm is deleted from the alarm system.

Properties	Type	Access	Description
SourceID	String	read only	Returns the source at which an alarm was triggered. The value depends on the origin of an alarm and is assigned by the data source, for example, Range-ID, controller/connection name or computer name.
State (Page 448)	UInt32	read only	Returns the state of an alarm.
StateText	String	read only	Returns the alarm state as text, e.g. "Incoming" or "Outgoing". The texts can be assigned system-wide for each alarm status.
SuppressionState (Page 449)	UInt8	read only	Returns the status of visibility of an active alarm.
SystemSeverity (Page 450)	UInt16	read only	Returns the severity level of a system fault as property of a system alarm.
TextColor	UInt32	read only	Returns the text color of the alarm state. Each alarm state has its own visual representation.
UserName	String	read only	Returns the name of the user who triggered the alarm object.
UserResponse (Page 449)	UInt16	read only	Returns the expected or required user response to an alarm.
Value	Variant	read only	Specifies a value for the object being used or returns it.
ValueLimit	Variant	read only	Returns the limit of a process value of an alarm.
ValueQuality	UInt16	read only	Returns the quality level of a process value of an alarm.

See also

"Flashing" property (Page 445)

"ChangeReason" property (Page 445)

"NotificationReason" property (Page 446)

"InvalidFlags" property (Page 446)

"Origin" property (Page 447)

"State" property (Page 448)

"SuppressionState" property (Page 449)

"UserResponse" property (Page 449)

"SystemSeverity" property (Page 450)

Special properties (RT Uni)

"Flashing" property (RT Uni)

Description

Returns whether the specified object flashes in runtime.

Value	Status
1	Alarm flashes
0	Alarm does not flash

A background color for flashing is specified with the "BackColor" property. The second background color and the frequency of the flashing is configured in the Alarm Control.

Syntax

```
Object.Flashing
```

`Object`

Required. An object from the "Availability" section.

"ChangeReason" property (RT Uni)

Description

Returns the trigger event for the modification of the alarm state. The time of last modification is saved in the "ModificationTime" property.

The alarm state can change for the following reasons:

Values	ChangeReason	Description
0x0001	AlarmStateChanged	"State" property has changed
0x0003	RaiseEvent	Status change "Incoming"
0x0005	ClearEvent	Status change "Reset"
0x0007	AcknowledgeEvent	Status change "Acknowledged"
0x0009	ResetEvent	Status change "Deleted"
0x000F	RemoveEvent	Status change "Removed"
0x0010	AlarmQualityChanged	"Quality" property has changed
0x0020	AlarmParameter-ValuesChanged	A value of the "AlarmParameterValues" property has changed
0x0040	AlarmPriorityChanged	"Priority" property has changed
0x0100	AlarmSuppression-StateChanged	"SuppressionState" property has changed
0x1000	ConfigurationChanged	Alarm configuration has changed

Syntax

```
Object.ChangeReason
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 442)

"NotificationReason" property (RT Uni)

Description

Returns the reason for an alarm.

The property can have the following values:

- 0: Unknown (e.g. when the alarm was read out from a log)
- 1: Add
- 2: Change
- 3: Remove

Syntax

```
Object.NotificationReason
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 442)

"InvalidFlags" property (RT Uni)

Description

Returns the cause of invalid data of an alarm.

An invalid alarm is marked with the following bits:

Bit number	InvalidFlags	Description
Bit 0	Invalid configuration flag	Alarm configuration is invalid. HMI device does not match data source.
Bit 1	Invalid timestamp flag	Data source transfers invalid time stamps.

Bit number	InvalidFlags	Description
Bit 2	Invalid alarm parameter flag	Data source transfers invalid parameter values.
Bit 3	Invalid event text flag	Runtime system cannot format text due to missing parameter values.

A valid alarm has the following properties:

- InvalidFlags = 0
- Quality = "good"

Syntax

```
Object.InvalidFlags
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 442)

"Origin" property (RT Uni)

Description

Returns the origin of an alarm.

The "Origin" property, for example, includes system names, data source or CPU ID. You can sort and filter alarms through the "Origin" context.

The "Origin" property can be configured and, together with the "Area" property, defines the source of an alarm. You can also use placeholders for context-sensitive format.

Type

String

Access

Access depends on the object.

Availability

The property is available for the following objects:

- AlarmResult
- LoggedAlarmStateResult

Syntax

```
Object.Origin
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 442)

"State" property (RT Uni)

Description

Returns the state of an alarm.

The table below shows the possible states of an alarm.

Value	State	Description
0x00	Normal (Idle)	Not an active alarm
0x01	Raised	Incoming
0x02	RaisedCleared	Incoming and reset
0x05	RaisedAcknowledged	Incoming and acknowledged
0x06	RaisedAcknowledgedCleared	Incoming, acknowledged and reset
0x07	RaisedClearedAcknowledged	Incoming, reset and acknowledged
0x80	Removed	Alarm was removed and is no longer available

Syntax

```
Object.State
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 442)

"SuppressionState" property (RT Uni)

Description

Returns the status of visibility of an active alarm.

Value	SuppressionState	Description
0x0	Unsuppressed	Alarm is visible.
0x1	Suppressed	Alarm is configured as invisible.
0x3	Shelved	Alarm was hidden manually. The methods "Unshelve" and "Shelve" can be applied.

Syntax

```
Object.SuppressionState
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 442)

"UserResponse" property (RT Uni)

Description

Returns the expected or required user response to an alarm:

Value	UserResponse	Description
0x0	No response	Active message expects no user response
0x1	Acknowledgment	Active message expects acknowledgment (also in group)
0x2	Reset	Active message expects reset (also in group)
0x5	Single acknowledgment	Active message explicitly expects individual acknowledgment
0x6	Single reset	Active message explicitly expects individual reset

Syntax

```
Object.UserResponse
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 442)

"SystemSeverity" property (RT Uni)

Description

Returns the severity level of a system fault as property of a system alarm. The value of the "SystemSeverity" property also has an impact on the runtime system monitoring (SystemHealthIndex).

The "SystemSeverity" property can display the following severity:

Value	SystemSeverity	Description
0	None	No effect on system monitoring.
1	Lowest severity	Fault with lowest impact on system monitoring.
2	Low severity	Fault with low impact on system monitoring.
3	Medium severity	Fault with medium impact on system monitoring.
4	High severity	Fault with great impact on system monitoring.
5	Highest severity	Fault with greatest impact on system monitoring.

Syntax

```
Object.SystemSeverity
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 442)

Methods of "AlarmResult" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmResult" object has the following methods:

Methods	Description
-	

"AlarmColumn" object (RT Uni)

"AlarmColumn" description (RT Uni)

Description

The "AlarmColumn" object enables access to the columns of the alarm view.

Type identifier in JavaScript

HMIAlarmColumn

Properties (RT Uni)

Properties

The "AlarmColumn" object has the following properties:

Properties	Type	Access	Description
AlarmBlock (Page 452)	HmiAlarmBlock	read/ write	Specified which component of the alarm is displayed.
AllowSort	Bool	read/ write	Specifies whether the sorting of columns is allowed.
BackColor	UInt32	read/ write	Specifies the background color.
Content	Object	read/ write	Returns the "Content" object.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
ForeColor	UInt32	read/ write	Specifies the font color.
Header	Object	read/ write	Specifies the "Header" object. Specifies the properties of a column header.
MaximumWidth	UInt32	read/ write	Specifies the maximum width.
MinimumWidth	UInt32	read/ write	Specifies the minimum width.
Name	String	read only	Returns the name of the object or specifies it.
OutputFormat	String	read/ write	Specifies the format for displaying values.
SortDirection	HmiSortDirection	read/ write	Specifies the direction of sorting: • None (0): None • Ascending (1): Oldest entries first • Descending (2): Newest entries first

Properties	Type	Access	Description
SortOrder	UInt8	read/ write	Specifies the sorting order.
UseAlarmColors	Bool	read/ write	Specifies whether the configured color of the alarm is used.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.

See also

"AlarmBlock" property (Page 452)

Special properties (RT Uni)

"AlarmBlock" property (RT Uni)

Description

Specifies which component of the alarm is displayed:

- Undefined (0): Not defined
- ID (1): Alarm number
- Name (2): Name
- Class (3): Alarm class
- Priority (4): Priority
- Group (5): Alarm group
- Origin (6): Origin
- Area (7): Area
- Comments (8): Alarm comment
- Information (9): Information text
- LoopInAlarm (10): Navigates to the screen in which the alarm was triggered.
- EventText (11): Alarm text
- AlarmText1..9 (12- 20): Custom alarm text
- AlarmState (21): Alarm state
- ModificationTime (22): Time of modification
- RaiseTime (23): Trigger time
- AcknowledgeTime (24): Time of acknowledgment
- ClearTime (25): Time of completion

- ResetTime (26): Reset time
- SuppressionState (27): Status of alarm suppression
- EscalationLevel (28): Escalation level
- Context (29): Context
- Duration (30): Duration
- AckknowledgementState (31): Acknowledgment state
- Value (32): Value
- ValueQuality (33): Quality code
- ValueLimit (34): Value limit
- TagName (35): Trigger tag
- Computer (36): PLC name
- User (37): Logged-on user
- ProcessValue1..10 (38-47): Process value
- ClassSymbol (48): Alarm class symbol
- StateSymbol (49): State symbol

Syntax

```
Object.AlarmBlock
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 451)

Methods of "AlarmColumn" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmColumn" object has the following methods:

Methods	Description
-	

"AlarmStatisticColumn" object (RT Uni)

"AlarmStatisticColumn" description (RT Uni)

Description

The "AlarmStatisticColumn" object enables access to the column for statistic calculations of the alarm view.

Type identifier in JavaScript

HMIAlarmStatisticColumn

Properties (RT Uni)

Properties

The "AlarmStatisticColumn" object has the following properties:

Properties	Type	Access	Description
AlarmStatistic-Block (Page 455)	HmiAlarmStatistic-Block	read/write	Specifies the property of the alarm that is displayed in the column.
AllowSort	Bool	read/write	Specifies whether the sorting of columns is allowed.
BackColor	UInt32	read/write	Specifies the background color.
Content	Object	read/write	Specifies the "Content" object.
Enabled	Bool	read/write	Specifies whether the specified object can be operated in runtime.
ForeColor	UInt32	read/write	Specifies the font color.
Header	Object	read/write	Specifies the "Header" object. Specifies the properties of a column header.
MaximumWidth	UInt32	read/write	Specifies the maximum width.
MinimumWidth	UInt32	read/write	Specifies the minimum width.
Name	String	read only	Returns the name of the object or specifies it.
OutputFormat	String	read/write	Specifies the format for displaying values.
SortDirection	HmiSortDirection	read/write	Specifies the direction of sorting: ● None (0): None ● Ascending (1): Oldest entries first ● Descending (2): Newest entries first

Properties	Type	Access	Description
SortOrder	UInt8	read/ write	Specifies the sorting order.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.

See also

"AlarmStatisticBlock" property (Page 455)

Special properties (RT Uni)

"AlarmStatisticBlock" property (RT Uni)

Description

Specifies the property of the alarm that is displayed in the column:

- Undefined (0): Not defined
- AverageRaisedRaised (4097): Average time between the alarm and the resulting alarms
- AverageRaisedCleared (4098): Average time between the alarm and its clearance
- AverageRaisedAcknowledged (4099): Average time between the alarm and its acknowledgment
- AverageRaisedReset (4100): Average time between the alarm and its reset
- Frequency (4101): Number of alarms per unit of time
- SumRaisedRaised (4102): Sum of all alarms and resulting alarms

Syntax

```
Object.AlarmStatisticBlock
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 454)

Methods of "AlarmStatisticColumn" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmStatisticColumn" object has the following methods:

Methods	Description
-	

"AlarmStatisticsView" object (RT Uni)

"AlarmStatisticsView" description (RT Uni)

Description

Represents the "Report statistics" object.

Type identifier in JavaScript

HMIAlarmStatisticsView

Properties (RT Uni)

Properties

The "AlarmStatisticsView" object has the following properties:

Properties	Type	Access	Description
AllowSort	Bool	read/ write	Specifies whether the sorting of columns is allowed.
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateForeColor	UInt32	read/ write	Specifies the flashing color for the text.
BackColor	UInt32	read/ write	Specifies the background color.
CellPadding	Object	read/ write	Specifies the inner distance of the contents from the cell frame.
CellTextTrimming	HmiTextTrimming	read/ write	Specifies the type of trimming of cell contents: ● None (0): None ● Ellipsis (1): Abbreviation at the end of the text

Properties	Type	Access	Description
ColoringMode	HmiGridColoring-Mode	read/ write	Specifies whether the alternate coloring of every other row or column is activated: • None (0): None • Rows (1): Alternately color the rows • Columns (2): Alternately color the columns
Columns	Object	read/ write	Specifies the "Columns" object.
ElementID	UInt32	read/ write	Specifies the ID of an element within the active screen.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
GridLineColor	UInt32	read/ write	Specifies the color of the grid lines.
GridLineVisibility	HmiSimpleGrid-Line	read/ write	Specifies the visibility of the grid lines. • None (0): None • Vertikal (1): Vertical • Horizontal (2): Horizontal
GridLineWidth	UInt8	read/ write	Specifies the width of the separator lines in pixels.
GridSelectionMode	HmiGridSelec-tionMode	read/ write	Specifies whether multiple selection is enabled in the table content. • None (0): None • Single (1): Only single selection • Multi (2): Multiple selection
HeaderSettings	Object	read/ write	Specifies the "HeaderSettings" object. Specifies the settings for all column headers of the table.
HorizontalScrollBar-Visibility	HmiScrollBarVisi-bility	read/ write	Specifies the setting for the horizontal scroll bar of the window. • Automatic (0): Only visible if required • Visible (1): Visible • Collapsed (2): Not visible
RowHeight	UInt8	read/ write	Specifies the height of all rows of the table in DIU (Device Independent Unit). "0" corresponds to an automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.
SelectFullRow	Bool	read/ write	Specifies whether only the cell or the whole row is included in a selection.
SelectionBackColor	UInt32	read/ write	Specifies the background color of the selected cells.
SelectionBorderCol-or	UInt32	read/ write	Specifies the border color of a selection.

Properties	Type	Access	Description
SelectionBorderWidth	UInt8	read/ write	Specifies the border thickness of a selection.
SelectionForeColor	UInt32	read/ write	Specifies the foreground color of the selected cells.
Type	UInt32	read/ write	Specifies the type ID.
VerticalScrollBarVisibility	HmiScrollBarVisibility	read/ write	Specifies the setting for the vertical scroll bar of the window: ● Automatic (0): Only visible if required ● Visible (1): Visible ● Collapsed (2): Not visible

Methods of "AlarmStatisticsView" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmStatisticsView" object has the following methods:

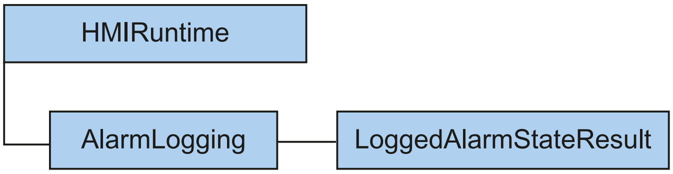
Methods	Description
-	

7.8.1.2 "AlarmLogging" area (RT Uni)

"AlarmLogging" object (RT Uni)

"AlarmLogging" description (RT Uni)

Description



The "AlarmLogging" object ("HMIAlarmLogging" type) gives you access to logged alarms of a logging system.

Type identifier in JavaScript

HMIAlarmLogging

Properties (RT Uni)

Properties

The "AlarmLogging" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "AlarmLogging" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmLogging" object has the following methods:

Methods	Description
AddComment	Adds comments for logged alarms ("LoggedAlarmStateResult" objects) asynchronously in the logging system.
Read	Reads out logged alarms ("LoggedAlarmStateResult" objects) of a time period asynchronously from a logging system.

"AddComment" method (AlarmLogging.AddComment) (RT Uni)

Description

Adds comments for logged alarms ("LoggedAlarmStateResult" objects) asynchronously in the logging system.

The method executes an asynchronous write operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the write operation, the corresponding handler of the Promise pattern is called.

Note

The "LoggedAlarmStateObjectID", "InstanceID" and "TimeStamp" parameters must correspond to the properties of the associated "LoggedAlarmStateResult" object.

Member

Method of the "AlarmLogging" object

Syntax

```
HMIRuntime.AlarmLogging.AddComment(LoggedAlarmStateObjectID,Instance
ID,TimeStamp,Language,Comment)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`LoggedAlarmStateObjectID`

Type: String

ID of the logged alarm

`InstanceID`

Type: UInt32

InstanceID of the logged alarm

`TimeStamp`

Type: DateTime

Time stamp of the comment

`Language`

Type: UInt32

Country identification of the language of the comment

`Comment`

Type: String

Comment for the logged alarms

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"Read" method (AlarmLogging.Read) (RT Uni)

Description

Reads out logged alarms ("LoggedAlarmStateResult" objects) of a time period asynchronously from a logging system.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, after the read operation is complete, the corresponding handler of the Promise pattern is called with an array with "LoggedAlarmStateResult" objects or an error code as parameter.

Member

Method of the "AlarmLogging" object

Syntax

```
HMIRuntime.AlarmLogging.Read(dateFrom,dateTo,filter,language,systemN
ames)
.then(function(LoggedAlarmStateResult[]) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`dateFrom`

Type: DateTime

Start date of the time period

`dateTo`

Type: DateTime

End date of the time period

`filter`

Type: String

SQL-like string for filtering the result set of the logged alarms.

`language`

Type: UInt32

Country identification of the language of the logged alarms and filter

`systemNames`

Type: String[]

System names of the Runtime systems of the logged alarms.

Return value

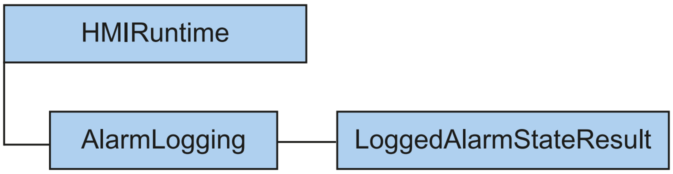
Depending on the status of the Promise object:

- Promise fulfilled
  Array with "LoggedAlarmStateResult" objects as parameters of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

"LoggedAlarmStateResult" object (RT Uni)

"LoggedAlarmStateResult" description (RT Uni)

Description



The "LoggedAlarmStateResult" object ("HMILoggedAlarmStateResult" type) gives you access to the properties of a logged alarm. The "LoggedAlarmStateResult" object is a pure data object that maps all properties of a logged alarm.

Type identifier in JavaScript

HMILoggedAlarmStateResult

Properties (RT Uni)

Properties

The "LoggedAlarmStateResult" object has the following properties:

Properties	Type	Access	Description
Acknowledgement-Time	Date-Time	read on-ly	Returns the time of alarm acknowledgment.
AlarmClassName	String	read on-ly	Returns the name of the alarm class of an alarm.

Properties	Type	Access	Description
AlarmClassSymbol	String	read only	Returns the abbreviation for the display of the alarm class of the alarm, for example, "W" for the alarm class "Warning".
AlarmParameterValues	Variant	read only	Returns an array with parameter values of an alarm. The property is mapped in the "AlarmResult" object. The parameter values are added to an alarm from the alarm source in the alarm state "Incoming" and "Reset". They can also include diagnostic or raw data from the PLC in addition to simple tag values from the configured AlarmParamterTags.
AlarmText	String[]	read only	Returns the localized additional texts 1 to 9 of an alarm as an array. The text can contain triggered placeholders and reference all the "AlarmParameterValues" properties of the respective alarm state "Incoming" or "Reset".
Area	String	read only	Specifies the origin area of an alarm. The "Area" property can be configured and, together with the "Origin" property, defines the source of an alarm. You can also use placeholders for context-sensitive format. The "Area" property, for example, includes subsystem, application name or PLC ID. You can sort and filter alarms through the "Area" context.
BackColor	UInt32	read only	Specifies the background color.
ChangeReason (Page 465)	UInt16	read only	Returns the trigger event for the modification of the alarm state.
ClearTime	DateTime	read only	Returns the time of alarm reset.
Connection	String	read only	Returns the name of the connection by which an alarm was triggered.
Deadband	Variant	read only	Returns the hysteresis value of the trigger tag of the alarm.
EventText	String	read only	Returns a localized text that describes an alarm event for the alarm. The text can contain triggered placeholders and reference all the "AlarmParameterValues" properties of the respective alarm state "Incoming" or "Reset".
HostName	String	read only	Returns the name of the PC on which the alarm was triggered.
ID	UInt32	read only	Returns the ID of a logged alarm.
InfoText	String	read only	Returns the information text of an alarm in all archived languages. Normally, the text is used for operator instructions.
InstanceID	UInt32	read only	Returns the ID of alarms with multiple instances.
InvalidFlags (Page 466)	UInt8	read only	Returns the cause of invalid data of an alarm.

Properties	Type	Access	Description
LoggedAlarmStateObjectID	String	read only	Returns the ID of a logged alarm. The ID is used for referencing a logged alarm, e.g. for commenting with the "AlarmLogging.AddComment" method.
ModificationTime	DateTime	read only	Returns the time stamp of the last modification of the alarm state. The reason for the change is included in the "ChangeReason" property.
Origin	String	read only	Returns the origin of an alarm. The "Origin" property, for example, includes system names, data source or CPU ID. You can sort and filter alarms through the "Origin" context. The "Origin" property can be configured and, together with the "Area" property, defines the source of an alarm. You can also use placeholders for context-sensitive format.
Priority	UInt8	read only	Specifies the relevance of an alarm or a machine status.
RaiseTime	DateTime	read only	Returns the trigger time of an alarm.
ResetTime	DateTime	read only	Returns the time of alarm reset.
SingleAcknowledgement	Boolean	read only	Returns whether an alarm must be acknowledged exclusively or can also be acknowledged in a group.
SourceType (Page 466)	UInt16	read only	Returns the type of the alarm source of an alarm. After the reset, the alarm is deleted from the alarm system.
State (Page 467)	UInt32	read only	Returns the state of an alarm.
StateMachine	UInt8	read only	Returns the response of the alarm for alarm states and alarm events.
StateText	String	read only	Returns the alarm state as text, e.g. "Incoming" or "Outgoing". The texts can be assigned system-wide for each alarm status.
SuppressionState (Page 468)	UInt8	read only	Returns the status of visibility of an active alarm.
TextColor	UInt32	read only	Returns the text color of the alarm state. Each alarm state has its own visual representation.
UserName	String	read only	Returns the name of the user who triggered the alarm object.
UserResponse (Page 468)	UInt16	read only	Returns the expected or required user response to an alarm.
Value	Variant	read only	Specifies a value for the object being used or returns it.
ValueLimit	Variant	read only	Returns the limit of a process value of an alarm.
ValueQuality	UInt16	read only	Returns the quality level of a process value of an alarm.

See also

"ChangeReason" property (Page 465)

"InvalidFlags" property (Page 466)

"SourceType" property (Page 466)

"State" property (Page 467)

"SuppressionState" property (Page 468)

"UserResponse" property (Page 468)

Special properties (RT Uni)

"ChangeReason" property (RT Uni)

Description

Returns the trigger event for the modification of the alarm state. The time of last modification is saved in the "ModificationTime" property.

The alarm state can change for the following reasons:

Values	ChangeReason	Description
0x0001	AlarmStateChanged	"State" property has changed
0x0003	RaiseEvent	Status change "Incoming"
0x0005	ClearEvent	Status change "Reset"
0x0007	AcknowledgeEvent	Status change "Acknowledged"
0x0009	ResetEvent	Status change "Deleted"
0x000F	RemoveEvent	Status change "Removed"
0x0010	AlarmQualityChanged	"Quality" property has changed
0x0020	AlarmParameter-ValuesChanged	A value of the "AlarmParameterValues" property has changed
0x0040	AlarmPriorityChanged	"Priority" property has changed
0x0100	AlarmSuppression-StateChanged	"SuppressionState" property has changed
0x1000	ConfigurationChanged	Alarm configuration has changed

Syntax

```
Object.ChangeReason
```

Object

Required. An object from the "Availability" section.

See also

Properties (Page 462)

"InvalidFlags" property (RT Uni)

Description

Returns the cause of invalid data of an alarm.

An invalid alarm is marked with the following bits:

Bit number	InvalidFlags	Description
Bit 0	Invalid configuration flag	Alarm configuration is invalid. HMI device does not match data source.
Bit 1	Invalid timestamp flag	Data source transfers invalid time stamps.
Bit 2	Invalid alarm parameter flag	Data source transfers invalid parameter values.
Bit 3	Invalid event text flag	Runtime system cannot format text due to missing parameter values.

A valid alarm has the following properties:

- InvalidFlags = 0
- Quality = "good"

Syntax

```
Object.InvalidFlags
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 462)

"SourceType" property (RT Uni)

Description

Returns the type of the alarm source of an alarm.

The following types of alarm sources are available:

- Tag system for tag-based alarms (Tag)
- PLC or external communication source for controller-based alarms (Controller)
- Subsystem of HMIRuntime for system-based alarms (System)
- Alarm system itself for alarms that are grouped (Alarm)

Syntax

```
Object.SourceType
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 462)

"State" property (RT Uni)

Description

Returns the state of an alarm.

The table below shows the possible states of an alarm.

Value	State	Description
0x00	Normal (Idle)	Not an active alarm
0x01	Raised	Incoming
0x02	RaisedCleared	Incoming and reset
0x05	RaisedAcknowledged	Incoming and acknowledged
0x06	RaisedAcknowledgedCleared	Incoming, acknowledged and reset
0x07	RaisedClearedAcknowledged	Incoming, reset and acknowledged
0x80	Removed	Alarm was removed and is no longer available

Syntax

```
Object.State
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 462)

"SuppressionState" property (RT Uni)

Description

Returns the status of visibility of an active alarm.

Value	SuppressionState	Description
0x0	Unsuppressed	Alarm is visible.
0x1	Suppressed	Alarm is configured as invisible.
0x3	Shelved	Alarm was hidden manually. The methods "Unshelve" and "Shelve" can be applied.

Syntax

```
Object.SuppressionState
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 462)

"UserResponse" property (RT Uni)

Description

Returns the expected or required user response to an alarm:

Value	UserResponse	Description
0x0	No response	Active message expects no user response
0x1	Acknowledgment	Active message expects acknowledgment (also in group)
0x2	Reset	Active message expects reset (also in group)
0x5	Single acknowledgment	Active message explicitly expects individual acknowledgment
0x6	Single reset	Active message explicitly expects individual reset

Syntax

```
Object.UserResponse
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 462)

Methods of "LoggedAlarmStateResult" (RT Uni)

Overview (RT Uni)

Methods

The "LoggedAlarmStateResult" object has the following methods:

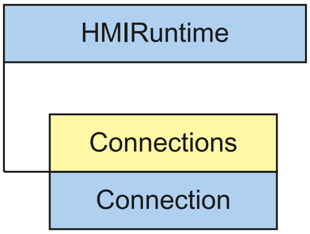
Methods	Description
-	

7.8.1.3 "Connections" area (RT Uni)

"Connection" object (RT Uni)

"Connection" description (RT Uni)

Description



The "Connection" object ("HMIConnection" type) enables access to individual connections of the Runtime system. A connection is a configured, logical assignment of two communication partners.

Type identifier in JavaScript

HMIVorlage

Properties (RT Uni)

Properties

The "Connection" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "Connection" (RT Uni)

Overview (RT Uni)

Methods

The "Connection" object has the following methods:

Methods	Description
SetConnectionMode	Changes the status of a connection ("Connection" object) in the Runtime system.

"SetConnectionMode" method (Connection.SetConnectionMode) (RT Uni)

Description

Changes the status of a connection ("Connection" object) in the Runtime system.

The method executes an asynchronous operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Member

Method of the "Connection" object

Syntax

```
[HMIRuntime.]Connections.Connection.SetConnectionMode(mode)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`mode`

Type: hmiConnectionMode

New connection status

The enumeration contains the following values:

- `Disabled` (0)
  Connection disconnected
- `Enabled` (1)
  Connection established

Return value

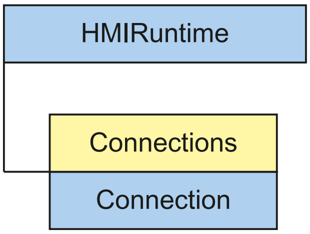
Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

"Connections" object (RT Uni)

"Connections" description (RT Uni)

Description



The "Connections" object ("HMIConnections" type) enables access to the connections of the Runtime system. A connection is a configured, logical assignment of two communication partners.

Use

Note

The "Connections" object is not a list, but rather a "Factory". You create an instance of the "Connection" object using the tag name.

The "Connection" objects cannot be counted and enumerated like conventional lists.

To reduce the use of the "Connections" object, you can also use the alias `Connections` for `HMIRuntime.Connections`.

Type identifier in JavaScript

HMIVorlage

Properties (RT Uni)

Properties

The "Connections" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "Connections" (RT Uni)

Overview (RT Uni)

Methods

The "Connections" object has the following methods:

Methods	Description
Item	Returns a connection ("Connection" object) of the runtime system.

"Item" method (Connections.Item) (RT Uni)

Description

Returns a connection ("Connection" object) of the runtime system.

Member

Method of the "Connections" object

Syntax

```
[HMIRuntime.]Connections[.Item](name);
```

Note

The `HMIRuntime.` part of the expression is not required. The alias `Connections` stands for `HMIRuntime.Connections`.

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Connections" object.

Parameter

`name`

Type: String

Name of a connection

Return value

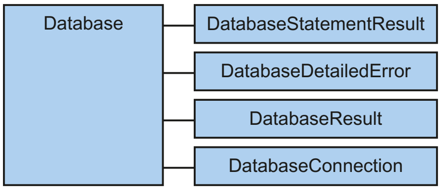
Object of the type "HMIConnection"

7.8.1.4 "Database" area (RT Uni)

"Database" object (RT Uni)

"Database" description (RT Uni)

Description



Displays the ODBC interface. You use this interface to access the data in a database using SQL commands.

Requirement is that an ODBC interface is installed on the HMI device.

Type identifier in JavaScript

HMIDatabase

Properties (RT Uni)

Properties

The "Database" object has the following properties:

Properties	Type	Access	Description
-	-	-	-

Methods of the "Database" object (RT Uni)

Overview (RT Uni)

Methods

The "Database" object has the following methods:

Methods	Description
CreateConnection	Creates a connection to the database.

See also

"CreateConnection" method (HMIDatabase.CreateConnection) (Page 474)

"CreateConnection" method (HMIDatabase.CreateConnection) (RT Uni)

Description

Establishes the connection to a database.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "HMIDatabase" object or the error code as parameter.

Member

Method of the "HMIDatabase" object

Syntax

```
HMIRuntime.HMIDatabase.CreateConnection(connectionString);
```

Parameters

`connectionString`

Type: String

Name of the database.

Return value

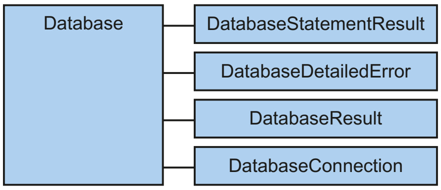
Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMIDatabase" as parameter of the "then()" handler.
- Promise failed (rejected)
  ErrorCode as parameter of the "catch()" handler

"DatabaseConnection" object (RT Uni)

"DatabaseConnection" description (RT Uni)

Description



Displays the connection to the database.

Type identifier in JavaScript

HMIDatabaseConnection

Properties (RT Uni)

Properties

The "DatabaseConnection" object has the following properties:

Properties	Type	Access	Description
-	-	-	-

Methods of the "DatabaseConnection" object (RT Uni)

Overview (RT Uni)

Methods

The "DatabaseConnection" object has the following methods:

Methods	Description
Close	Terminates the connection to the database.
Execute	Executes a query in the database.

See also

"Close" method (HMIDatabaseConnection.Close) (Page 476)

"Execute" method (HMIDatabaseConnection.Execute) (Page 477)

"Close" method (HMIDatabaseConnection.Close) (RT Uni)

Description

Terminates the connection to the database.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "HMIDatabaseConnection" object or the error code as parameter.

Member

Method of the "HMIDatabaseConnection" object

Syntax

```
HMIRuntime.HMIDatabaseConnection.Close();
```

Parameters

-

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMIDatabaseConnection" as parameter of the "then()" handler.
- Promise failed (rejected)
  ErrorCode as parameter of the "catch()" handler

"Execute" method (HMIDatabaseConnection.Execute) (RT Uni)

Description

Executes a query in the database.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "HMIDatabaseConnection" object or the error code as parameter.

Member

Method of the "HMIDatabaseConnection" object

Syntax

```
HMIRuntime.HMIDatabaseConnection.Execute(query, values);
```

Parameters

`query`

Type: String

Query

`values`

Type: Variant

Value array

Return value

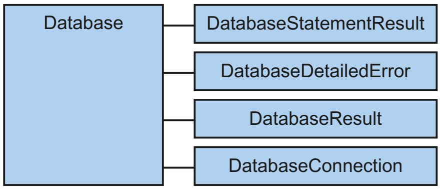
Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMIDatabaseConnection" as parameter of the "then()" handler.
- Promise failed (rejected)
  ErrorCode as parameter of the "catch()" handler

"DatabaseDetailedError" object (RT Uni)

"DatabaseDetailedError" description (RT Uni)

Description



Displays the error description of a failed database query.

Type identifier in JavaScript

HMIDatabaseDetailedError

Properties (RT Uni)

Properties

The "DatabaseDetailedError" object has the following properties:

Properties	Type	Access	Description
Message	String	read only	Returns the error description.
State	String	read only	Returns the ODBC error type.

Methods of the "DatabaseDetailedError" object (RT Uni)

Overview (RT Uni)

Methods

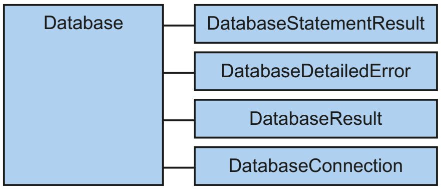
The "DatabaseDetailedError" object has the following methods:

Methods	Description
-	-

"DatabaseResult" object (RT Uni)

"DatabaseResult" description (RT Uni)

Description



Displays the result of a database query.

Type identifier in JavaScript

HMIDatabaseResult

Properties (RT Uni)

Properties

The "DatabaseResult" object has the following properties:

Properties	Type	Access	Description
GlobalError	ErrorCode	read only	Returns the error ID.
Results	Object[]	read only	Contains the result of the database query.

Methods of the "DatabaseResult" object (RT Uni)

Overview (RT Uni)

Methods

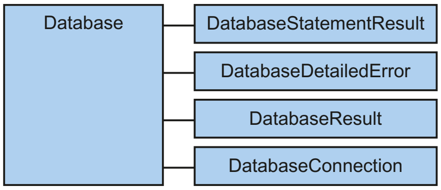
The "DatabaseResult" object has the following methods:

Methods	Description
-	-

"DatabaseStatementResult" object (RT Uni)

"DatabaseStatementResult" description (RT Uni)

Description



Displays table rows of a database query.

Type identifier in JavaScript

HMIDatabaseStatementResult

Properties (RT Uni)

Properties

The "DatabaseStatementResult" object has the following properties:

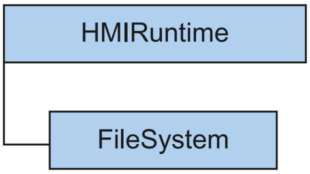
Properties	Type	Access	Description
Errors	Object[]	read only	Returns the error descriptions.
Rows	Object[]	read only	Returns the rows of the database table.

Methods of the "DatabaseStatementResult" object (RT Uni)

7.8.1.5 "FileSystem" object (RT Uni)

"FileSystem" description (RT Uni)

Description



The "FileSystem" object ("HMIFileSystem" type) enables access to the file system of the server on which WinCC Unified is installed.

Type identifier in JavaScript

HMIFileSystem

Properties (RT Uni)

Properties

The "FileSystem" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "FileSystem" (RT Uni)

Overview (RT Uni)

Methods

The "FileSystem" object has the following methods:

Methods	Description
AppendFile	Appends text to the end of a text file in the file system.
AppendFileBinary	Appends binary data to the end of a binary file in the file system.
CreateDirectory	Creates a new directory in the file system.

Methods	Description
DeleteDirectory	Deletes a directory with all subdirectories and files contained in the file system.
DeleteFile	Deletes a file from the file system.
ReadFile	Reads the content of a text file from the file system.
ReadFileBinary	Reads the content of a binary file from the file system.
WriteFile	Writes text to a new file in the file system.
WriteFileBinary	Writes binary data to a new file in the file system.

"AppendFile" method (FileSystem.AppendFile) (RT Uni)

Description

Appends text to the end of a text file in the file system.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Syntax

```
HMIRuntime.FileSystem.AppendFile(path,data,encoding)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Member

Method of the "FileSystem" object

Parameters

path

Type: String

Path of the text file in the file system

data

Type: String

Content that is written to the text file.

encoding

Type: String

Encoding of text file, e.g. UFT-8 or UCS-2.

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

See also

"AppendFileBinary" method (FileSystem.AppendFileBinary) (Page 483)

"AppendFileBinary" method (FileSystem.AppendFileBinary) (RT Uni)

Description

Appends binary data to the end of a binary file in the file system.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Syntax

```
HMIRuntime.FileSystem.AppendFileBinary(path,data)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Member

Method of the "FileSystem" object

Parameters

`path`

Type: String

Path of the binary file in the file system

`data`

Type: Blob

Content that is written to the binary file.

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

See also

"AppendFile" method (FileSystem.AppendFile) (Page 482)

"CreateDirectory" method (FileSystem.CreateDirectory) (RT Uni)

Description

Creates a new directory in the file system.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Syntax

```
HMIRuntime.FileSystem.CreateDirectory(path)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Member

Method of the "FileSystem" object

Parameters

path

Type: String

Path of the directory in the file system

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

See also

"DeleteDirectory" method (FileSystem.DeleteDirectory) (Page 485)

"DeleteDirectory" method (FileSystem.DeleteDirectory) (RT Uni)

Description

Deletes a directory with all subdirectories and files contained in the file system.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Syntax

```
HMIRuntime.FileSystem.DeleteDirectory(path)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Member

Method of the "FileSystem" object

Parameters

path

Type: String

Path of the directory in the file system

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

See also

"CreateDirectory" method (FileSystem.CreateDirectory) (Page 484)

"DeleteFile" method (FileSystem.DeleteFile) (RT Uni)

Description

Deletes a file from the file system.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Syntax

```
HMIRuntime.FileSystem.DeleteFile(path)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Member

Method of the "FileSystem" object

Parameters

path

Type: String

Path of the file in the file system

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

See also

"WriteFile" method (FileSystem.WriteFile) (Page 489)

"ReadFile" method (FileSystem.ReadFile) (RT Uni)

Description

Reads the content of a text file from the file system.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called with the content of the file or the error code as parameter.

Member

Method of the "FileSystem" object

Syntax

```
HMIRuntime.FileSystem.ReadFile(path,encoding)
.then(function(data) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`path`

Type: String

Path of the text file in the file system

`encoding`

Type: String

Encoding of text file, e.g. UFT-8 or UCS-2.

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  File content as string as parameter of the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

"ReadFileBinary" method (FileSystem.ReadFileBinary) (RT Uni)

Description

Reads the content of a binary file from the file system.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called with the content of the file or the error code as parameter.

Member

Method of the "FileSystem" object

Syntax

```
HMIRuntime.FileSystem.ReadFileBinary(path)
.then(function(data) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the binary file in the file system

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  File content as binary data object (Blob) as parameter of the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

"WriteFile" method (FileSystem.WriteFile) (RT Uni)

Description

Writes text to a new file in the file system.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Syntax

```
HMIRuntime.FileSystem.WriteFile(path,data,encoding)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Member

Method of the "FileSystem" object

Parameters

`path`

Type: String

Path of the new text file in the file system

`data`

Type: String

Content that is written to the new text file.

`encoding`

Type: String

Encoding of the new text file, e.g. UFT-8 or UCS-2.

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

See also

"WriteFileBinary" method (FileSystem.WriteFileBinary) (Page 490)

"DeleteFile" method (FileSystem.DeleteFile) (Page 486)

"WriteFileBinary" method (FileSystem.WriteFileBinary) (RT Uni)

Description

Writes binary data to a new file in the file system.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Syntax

```
HMIRuntime.FileSystem.WriteFileBinary(path,data)
.then(function() {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Member

Method of the "FileSystem" object

Parameters

`path`

Type: String

Path of the new binary file in the file system

`data`

Type: Blob

Content that is written to the new binary file.

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  No return for the "then()" handler
- Promise rejected
  Error code as parameter of the "catch()" handler.

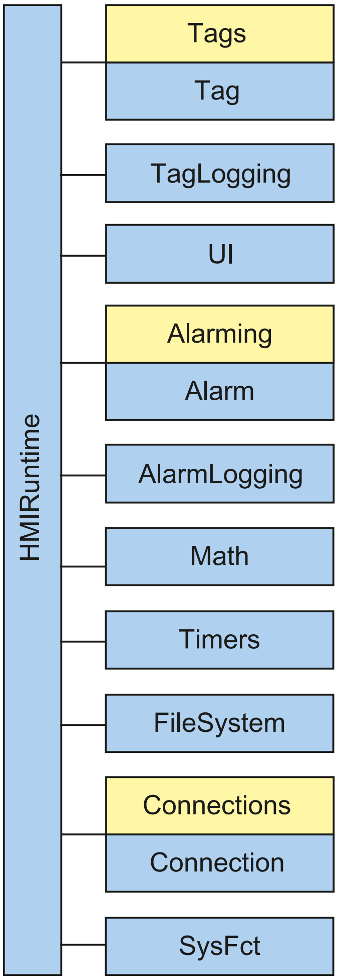
See also

"WriteFile" method (FileSystem.WriteFile) (Page 489)

7.8.1.6 "HMIRuntime" object (RT Uni)

"HMIRuntime" description (RT Uni)

Description



The "HMIRuntime" object represents the Runtime system of WinCC Unified. The "HMIRuntime" object contains properties, methods and all objects of the runtime system that can be scripted.

Use

Note

The following objects have an alias to allow briefer notation:

- Alias `Tags` corresponds to `HMIRuntime.Tags`
- Alias `UI` corresponds to `HMIRuntime.UI`

Type identifier in JavaScript

HMIRuntime

Properties (RT Uni)

Properties

The "HMIRuntime" object has the following properties:

Properties	Type	Access	Description
Alarming	Object	read only	Returns the "Alarming" object.
AlarmLogging	Object	read only	Returns the "AlarmLogging" object.
Connections	HMIConnections	read only	Returns the "Connections" list.
CPM	Object	read only	Returns the "CPM" object.
FileSystem	Object	read only	Returns the "FileSystem" object.
Language	UInt32	read/write	Specifies the current Runtime language.
Math	Object	read only	Returns the "Math" object.
OLEAutomation	Object	read only	Returns the "OLEAutomation" object.
ParameterSetTypes	Object	read only	Returns the "ParameterSetTypes" object.
SysFct	Object	read/write	Returns the "SysFct" object.
TagLogging	Object	read only	Returns the "TagLogging" object.
Tags	Object	read only	Returns the "Tags" object.
Timers	Object	read only	Returns the "Timers" object.
UI	Object	read only	Returns the "UI" object.

Methods of "HMIRuntime" (RT Uni)

Overview (RT Uni)

Methods

The "HMIRuntime" object has the following methods:

Methods	Description
Trace	Outputs a user-defined text through the debug output of the runtime system.

"Trace" method (HMIRuntime.Trace) (RT Uni)

Description

Outputs a user-defined text through the debug output of the runtime system.

Member

Method of the "HMIRuntime" object

Syntax

```
HMIRuntime.Trace(message);
```

Parameter

`message`

Type: String

The text which is output.

Return value

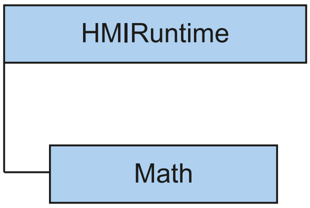
--

7.8.1.7 "Math" area (RT Uni)

"Math" object (RT Uni)

"Math" description (RT Uni)

Description



The "Math" object ("HMIMath" type) enables the use of 64-bit data types in the scripting environment. Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as object for further processing.

Use

Some methods return different data types; the "Tag.Read" method, for example, returns the data type "Variant". You can check the return values of these methods with the JavaScript operator "instanceof" for agreement with a 64-bit data type.

Note

64-bit values should only be processed with the methods of the "Int64" and "Uint64" objects. When you are using the native computing operations of JavaScript (e.g. "+" or "-"), the accuracy of the results is diminished.

Type identifier in JavaScript

HMIMath

Example

Checks if a "Tag" object returns a 64-bit data type (signed or unsigned):

Copy code

```
var tagVal = HMIRuntime.Tags('Tag1').Read();
if (tagVal instanceof HMIRuntime.Math.Int64Base)
{
    ...
}
```

Checks if a "Tag" object returns a "signed" 64-bit data type:

Copy code

```
var tagVal = HMIRuntime.Tags('Tag1').Read();
if (tagVal instanceof HMIRuntime.Math.Int64)
{
    ...
}
```

Properties (RT Uni)

Properties

The "Math" object has the following properties:

Properties	Type	Access	Description
DatePrecise (Page 495)	Object	read only	Returns the "DatePrecise" object.
Int64 (Page 496)	Object	read only	Represents a 64-bit integer value with sign.
Int64Base (Page 497)	Object	read only	Checks for a 64-bit data type in combination with the "`instanceof`" operator.
Uint64 (Page 497)	Object	read only	Represents an unsigned 64-bit integer value and contains methods for mathematical operations.

See also

"DatePrecise" property (Page 495)

"Int64" property (Page 496)

"Int64Base" property (Page 497)

"Uint64" property (Page 497)

Special properties (RT Uni)

"DatePrecise" property (RT Uni)

Description

Returns the "DatePrecise" object.

Represents high-resolution time information with a resolution of 100 ns as a 64-bit integer value.

Type

Object

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Math

Syntax

```
Object.DatePrecise
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 495)

"Int64" property (RT Uni)

Description

Represents a 64-bit integer value with sign.

Type

Object

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Math

Syntax

```
Object.Int64
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 495)

"Int64Base" property (RT Uni)

Description

Checks for a 64-bit data type in combination with the "`instanceof`" operator.

Type

Object

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Math

Syntax

```
Object.Int64Base
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 495)

"Uint64" property (RT Uni)

Description

Represents an unsigned 64-bit integer value and contains methods for mathematical operations. Returns "TRUE" for unsigned 64-bit integer values when checked with the "instanceof" operator against an object.

Type

Object

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Math

Syntax

```
Object.Uint64
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 495)

Methods of "Math" (RT Uni)

Overview (RT Uni)

Methods

The "Math" object has the following methods:

Methods	Description
RGB	Converts an RGB(A) specification into the corresponding hexadecimal value.
RGBWeb	Converts an RGB(A) specification into the corresponding hexadecimal value.

"RGB" method (HMIMath.RGB) (RT Uni)

Description

Converts an RGB(A) specification into the corresponding hexadecimal value.

Member

Method of the "HMIMath" object

Syntax

```
HMIRuntime.HMIMath.RGB(R, G, B, A);
```

Parameters

R

Type: UInt32

Red value

G

Type: UInt32

Green value

B

Type: UInt32

Blue value

A

Type: UInt32

Alpha value (density)

Return value

Type: UInt32

"RGBWeb" method (HMIMath.RGBWeb) (RT Uni)

Description

Converts an RGB(A) specification into the corresponding hexadecimal value.

Member

Method of the "HMIMath" object

Syntax

```
HMIRuntime.HMIMath.RGBWeb();
```

Parameters

RGB

Type: UInt32

Red-green-blue value in hexadecimal notation

A

Type: UInt32

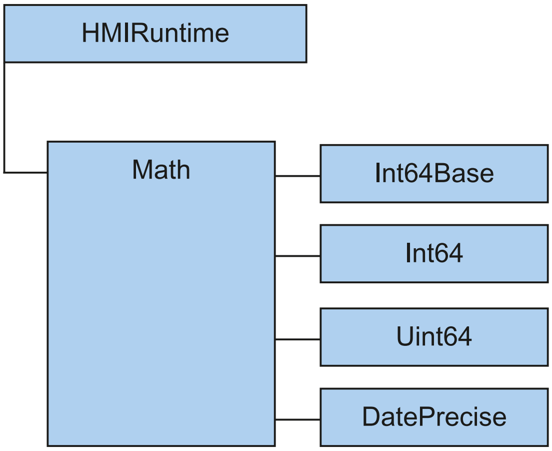
Alpha value (density) in hexadecimal format

Return value

Type: UInt32

"Int64Base" object (RT Uni)

Description



The "Int64Base" object (type "HMIInt64Base") is used exclusively when checking for a 64-bit integer data type. Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as "Int64" or "Uint64" objects for further use.

Object properties

--

Application

Note

64-bit values should only be processed with the methods of the "Int64" and "Uint64" objects. When you are using the native computing operations of JavaScript (e.g. "+" or "-"), the accuracy of the results is diminished.

The "Int64Base" object returns TRUE for all signed or unsigned 64-bit integer values when checked with the "instanceof" operator against an object.

Example

Checks whether a "Tag" object is a 64-bit data type:

Copy code

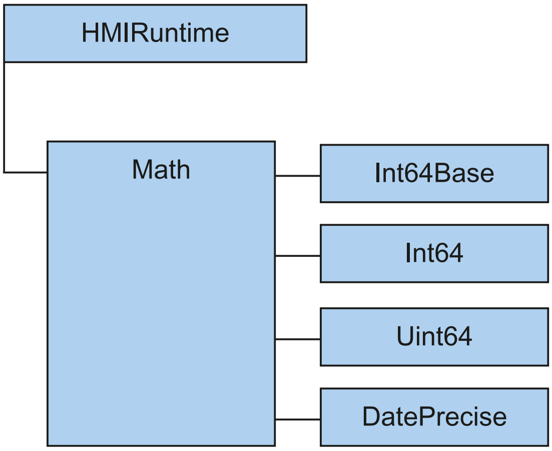
```
function Int64TagValue() {
    var tagVal = HMIRuntime.Tags('Tag1').Read();

    //check if it is 64-Bit type (signed or unsigned)
    if (tagVal instanceof HMIRuntime.Math.Int64Base) {
        ...
    }
}
```

"Int64" object (RT Uni)

"Int64" description (RT Uni)

Description



The "Int64" object (type "HMIInt64") represents a signed 64-bit integer value and includes basic arithmetic and bit operations. Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as "Int64" object.

Application

Note

64-bit values should only be processed with the methods of the "Int64" and "Uint64" objects. When you are using the native computing operations of JavaScript (e.g. "+" or "-"), the accuracy of the results is diminished.

Type identifier in JavaScript

HMIInt64

Example

Creates a new "signed" 64-bit object and writes the value to the tag:

Copy code

```
function Write_Int64TagValue() {
    //create new Int64-object
    var newTagVal = HMIRuntime.Math.Int64('-6000000000000000000');

    //write to tag
    HMIRuntime.Tags('Tag1').Write(newTagVal);
}
```

Checks if a "Tag" object is a "signed" 64-bit data type, multiplies the value with "-1" and returns the result as 64-bit object:

Copy code

```
function NegMul_Int64TagValue() {
    var tagVal = HMIRuntime.Tags('Tag1').Read();

    //check if it is *signed* 64-Bit type
    if (tagVal instanceof HMIRuntime.Math.Int64) {
        //if yes, use Mul method with negative number
        return tagVal.Mul(-1);
    }
}
```

Properties (RT Uni)

Properties

The "Int64" object has the following properties:

Properties	Type	Access	Description
Hi (Page 503)	UInt32	read only	Saves and returns the high part (32-bit) of a 64-bit integer value.
Lo (Page 504)	UInt32	read only	Saves and returns the low part (32-bit) of a 64-bit integer value.

See also

"Hi" property (Page 503)

"Lo" property (Page 504)

Special properties (RT Uni)

"Hi" property (RT Uni)

Description

Saves and returns the high part (32-bit) of a 64-bit integer value.

Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as object for further processing. The high and low 32-bit parts of a 64-bit value are mapped in the properties "Hi" and "Lo".

Type

UInt32

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Int64
- Uint64

Syntax

```
Object.Hi
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 503)

"Lo" property (RT Uni)

Description

Saves and returns the low part (32-bit) of a 64-bit integer value.

Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as object for further processing. The high and low 32-bit parts of a 64-bit value are mapped in the properties "Hi" and "Lo".

Type

UInt32

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Int64
- Uint64

Syntax

```
Object.Lo
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 503)

Methods of "Int64" (RT Uni)

Overview (RT Uni)

Methods

The "Int64" object has the following methods:

Methods	Description
Add	Provides the "Addition" arithmetic operation for 64-bit objects.
And	Provides the "AND" bit operation for 64-bit objects.
Div	Provides the "Division" arithmetic operation for 64-bit objects.
Item	Creates and returns 64-bit integer values ("Int64" and "Uint64" objects).
Mul	Provides the "Multiplication" arithmetic operation for 64-bit objects.
Or	Provides the "OR" bit operation for 64-bit objects.
ShiftLeft	Provides a bit shift "SHL" for 64-bit objects.
ShiftRight	Provides the bit shift "SHR" for 64-bit objects.
Sub	Provides the "Subtraction" arithmetic operation for 64-bit objects.
toString	Converts the value of a 64-bit object into a string.
Xor	Provides the "XOR" bit operation for 64-bit objects.

"Add" method (Int64.Add, Uint64.Add) (RT Uni)

Description

Provides the "Addition" arithmetic operation for 64-bit objects. The value of the "Int64" or "Uint64" objects is increased by the specified value.

This method corresponds to the JavaScript operator "+" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Add(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Value that is added to the current value of the object.

The following data types are supported:

- Numerical value
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"And" method ((Int64.AndUint64.And) (RT Uni)

Description

Provides the "AND" bit operation for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "&" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.And(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Bit sequence of the same length with which the binary value of the object is ANDed.

The following data types are supported:

- Bit string
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Div" method (Int64.DivUint64.Div) (RT Uni)

Description

Provides the "Division" arithmetic operation for 64-bit objects. The value of the "Int64" or "Uint64" objects is divided by the specified value.

This method corresponds to the JavaScript operator "/" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Div(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Value by which the current value of the object is divided.

The following data types are supported:

- Numerical value
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Mul" method ((Int64.MulUint64.Mul) (Page 509)

"Item" method (Int64.Item, Uint64.Item) (RT Uni)

Description

Creates 64-bit integer values ("Int64"and "Uint64" objects) and returns these.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object[.Item](value);
```

Object

Required. An object of the type "HMIInt64" or "HMIUint64".

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Int64" and "Uint64" objects.

Parameter

value

Type: Variant

New 64-bit integer value as integer string with base 10.

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Mul" method ((Int64.MulUint64.Mul) (RT Uni)

Description

Provides the "Multiplication" arithmetic operation for 64-bit objects. The value of the "Int64" or "Uint64" object is multiplied by the specified value.

This method corresponds to the JavaScript operator "*" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Mul(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Value by which the current value of the object is multiplied.

The following data types are supported:

- Numerical value
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Div" method (Int64.DivUint64.Div) (Page 507)

"Or" method (Int64.OrUint64.Or) (RT Uni)

Description

Provides the "OR" bit operation for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "|" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Or(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Bit sequence of the same length with which the binary value of the object is ORed.

The following data types are supported:

- Bit string
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Xor" method (Int64.XorUint64.Xor) (Page 515)

"ShiftLeft" method ((Int64.ShiftLeftUint64.ShiftLeft) (RT Uni)

Description

Provides a bit shift "SHL" for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is shifted by the specified number of places.

This method corresponds to the JavaScript operator "<<" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.ShiftLeft(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: UInt8

Number of digits by which the binary value of the object is shifted to the left.

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"ShiftRight" method ((Int64.ShiftRightUint64.ShiftRight) (Page 512)

"ShiftRight" method ((Int64.ShiftRightUint64.ShiftRight) (RT Uni)

Description

Provides the bit shift "SHR" for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is shifted by the specified number of places.

This method corresponds to the JavaScript operator ">>" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.ShiftRight(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: UInt8

Number of digits by which the binary value of the object is shifted to the right.

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"ShiftLeft" method ((Int64.ShiftLeftUint64.ShiftLeft) (Page 511)

"Sub" method (Int64.SubUint64.Sub) (RT Uni)

Description

Provides the "Subtraction" arithmetic operation for 64-bit objects. The value of the "Int64" or "Uint64" objects is increased by the specified value.

This method corresponds to the JavaScript operator "-" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Sub(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Value that is subtracted from the current value of the object. The following data types are supported:

- Numerical value
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"toString" method (Int64.toStringUint64.toString) (RT Uni)

Description

Converts the value of a 64-bit object into a string. For other data types, you can use the native JavaScript method "toString" with the same name.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.toString([base]);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`base`

Optional, type: UInt8

Basis on which the 64-bit value of the object is converted into a string. Without parameters, the basis is set to "10".

Return value

Representation of the numerical value of the type String.

See also

"Int64Base" object (Page 500)

"Xor" method (Int64.XorUint64.Xor) (RT Uni)

Description

Provides the "XOR" bit operation for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "^" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Xor(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Bit sequence of the same length with which the binary value of the object is XORed.

The following data types are supported:

- Bit string
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

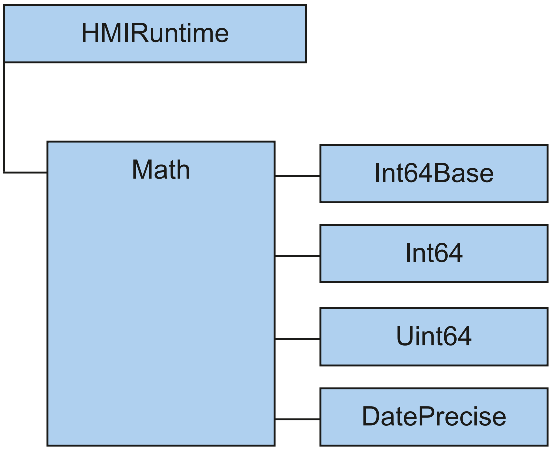
"Int64Base" object (Page 500)

"Or" method (Int64.OrUint64.Or) (Page 510)

"Uint64" object (RT Uni)

"Uint64" description (RT Uni)

Description



The "Uint64" object (type "HMIUint64") represents an unsigned 64-bit integer value and includes basic arithmetic and bit operations. Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as "Uint64" object.

Application

Note

64-bit values should only be processed with the methods of the "Int64" and "Uint64" objects. When you are using the native computing operations of JavaScript (e.g. "+" or "-"), the accuracy of the results is diminished.

Type identifier in JavaScript

HMIUint64

Example

Creates a new "unsigned" 64-bit object and writes the value to the tag:

Copy code

```
function Write_Uint64TagValue() {
    //create new Uint64-object
    var newTagVal = HMIRuntime.Math.Uint64('6000000000000000000');

    //write to tag
    HMIRuntime.Tags('Tag1').Write(newTagVal);
}
```

Checks if a "Tag" object is a signed 64-bit data type, adds the value "99" and returns the result as 64-bit object:

Copy code

```
function Add_Int64TagValue() {
    var tagVal = HMIRuntime.Tags('Tag1').Read();

    //check if it is 64-Bit type (unsigned)
    if (tagVal instanceof HMIRuntime.Math.Uint64) {
        //if yes, use Add method
        return tagVal.Add(99);
    }
}
```

Properties (RT Uni)

Properties

The "Uint64" object has the following properties:

Properties	Type	Access	Description
Hi (Page 518)	UInt32	reat only	Saves and returns the high part (32-bit) of a 64-bit integer value.
Lo (Page 518)	UInt32	read only	Saves and returns the low part (32-bit) of a 64-bit integer value.

See also

"Hi" property (Page 518)

"Lo" property (Page 518)

Special properties (RT Uni)

"Hi" property (RT Uni)

Description

Saves and returns the high part (32-bit) of a 64-bit integer value.

Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as object for further processing. The high and low 32-bit parts of a 64-bit value are mapped in the properties "Hi" and "Lo".

Type

UInt32

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Int64
- Uint64

Syntax

```
Object.Hi
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 503)

Properties (Page 517)

"Lo" property (RT Uni)

Description

Saves and returns the low part (32-bit) of a 64-bit integer value.

Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as object for further processing. The high and low 32-bit parts of a 64-bit value are mapped in the properties "Hi" and "Lo".

Type

UInt32

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Int64
- Uint64

Syntax

```
Object.Lo
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 503)

Properties (Page 517)

Methods of "Uint64" (RT Uni)

Overview (RT Uni)

Methods

The "Uint64" object has the following methods:

Methods	Description
Add	Provides the "Addition" arithmetic operation for 64-bit objects.
And	Provides the "AND" bit operation for 64-bit objects.
Div	Provides the "Division" arithmetic operation for 64-bit objects.
Item	Creates and returns 64-bit integer values ("Int64" and "Uint64" objects).
Mul	Provides the "Multiplication" arithmetic operation for 64-bit objects.

Methods	Description
Or	Provides the "OR" bit operation for 64-bit objects.
ShiftLeft	Provides a bit shift "SHL" for 64-bit objects.
ShiftRight	Provides the bit shift "SHR" for 64-bit objects.
Sub	Provides the "Subtraction" arithmetic operation for 64-bit objects.
toString	Converts the value of a 64-bit object into a string.
Xor	Provides the "XOR" bit operation for 64-bit objects.

"Add" method (Int64.Add, Uint64.Add) (RT Uni)

Description

Provides the "Addition" arithmetic operation for 64-bit objects. The value of the "Int64" or "Uint64" objects is increased by the specified value.

This method corresponds to the JavaScript operator "+" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Add(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Value that is added to the current value of the object.

The following data types are supported:

- Numerical value
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Sub" method (Int64.SubUint64.Sub) (Page 528)

"And" method ((Int64.AndUint64.And) (RT Uni)

Description

Provides the "AND" bit operation for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "&" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.And(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Bit sequence of the same length with which the binary value of the object is ANDed.

The following data types are supported:

- Bit string
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Div" method (Int64.DivUint64.Div) (RT Uni)

Description

Provides the "Division" arithmetic operation for 64-bit objects. The value of the "Int64" or "Uint64" objects is divided by the specified value.

This method corresponds to the JavaScript operator "/" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Div(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Value by which the current value of the object is divided.

The following data types are supported:

- Numerical value
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Mul" method ((Int64.MulUint64.Mul) (Page 524)

"Item" method (Int64.Item, Uint64.Item) (RT Uni)

Description

Creates 64-bit integer values ("Int64"and "Uint64" objects) and returns these.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object[.Item](value);
```

Object

Required. An object of the type "HMIInt64" or "HMIUint64".

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Int64" and "Uint64" objects.

Parameter

value

Type: Variant

New 64-bit integer value as integer string with base 10.

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Mul" method ((Int64.MulUint64.Mul) (RT Uni)

Description

Provides the "Multiplication" arithmetic operation for 64-bit objects. The value of the "Int64" or "Uint64" object is multiplied by the specified value.

This method corresponds to the JavaScript operator "*" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Mul(value);
```

Object

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

value

Type: Variant, object

Value by which the current value of the object is multiplied.

The following data types are supported:

- Numerical value
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Div" method (Int64.DivUint64.Div) (Page 522)

"Or" method (Int64.OrUint64.Or) (RT Uni)

Description

Provides the "OR" bit operation for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "|" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Or(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Bit sequence of the same length with which the binary value of the object is ORed.

The following data types are supported:

- Bit string
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Xor" method (Int64.XorUint64.Xor) (Page 529)

"ShiftLeft" method ((Int64.ShiftLeftUint64.ShiftLeft) (RT Uni)

Description

Provides a bit shift "SHL" for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is shifted by the specified number of places.

This method corresponds to the JavaScript operator "<<" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.ShiftLeft(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: UInt8

Number of digits by which the binary value of the object is shifted to the left.

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"ShiftRight" method ((Int64.ShiftRightUint64.ShiftRight) (Page 527)

"ShiftRight" method ((Int64.ShiftRightUint64.ShiftRight) (RT Uni)

Description

Provides the bit shift "SHR" for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is shifted by the specified number of places.

This method corresponds to the JavaScript operator ">>" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.ShiftRight(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: UInt8

Number of digits by which the binary value of the object is shifted to the right.

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"ShiftLeft" method ((Int64.ShiftLeftUint64.ShiftLeft) (Page 526)

"Sub" method (Int64.SubUint64.Sub) (RT Uni)

Description

Provides the "Subtraction" arithmetic operation for 64-bit objects. The value of the "Int64" or "Uint64" objects is increased by the specified value.

This method corresponds to the JavaScript operator "-" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Sub(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Value that is subtracted from the current value of the object. The following data types are supported:

- Numerical value
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

"Int64Base" object (Page 500)

"Add" method (Int64.Add, Uint64.Add) (Page 520)

"toString" method (Int64.toStringUint64.toString) (RT Uni)

Description

Converts the value of a 64-bit object into a string. For other data types, you can use the native JavaScript method "toString" with the same name.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.toString([base]);
```

Object

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

base

Optional, type: UInt8

Basis on which the 64-bit value of the object is converted into a string. Without parameters, the basis is set to "10".

Return value

Representation of the numerical value of the type String.

See also

"Int64Base" object (Page 500)

"Xor" method (Int64.XorUint64.Xor) (RT Uni)

Description

Provides the "XOR" bit operation for 64-bit objects. The binary value of the "Int64" or "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "^" for other data types.

Member

Method of the following objects:

- Int64
- Uint64

Syntax

```
Object.Xor(value);
```

`Object`

Required. An object of the type "HMIInt64" or "HMIUint64."

Parameters

`value`

Type: Variant, object

Bit sequence of the same length with which the binary value of the object is XORed.

The following data types are supported:

- Bit string
- Reference to object of the type "HMIInt64"
- Reference to object of the type "HMIUint64"

Return value

Object of the type:

- "HMIInt64" when using HMIRuntime.Math.Int64
- "HMIUint64" when using HMIRuntime.Math.Uint64

See also

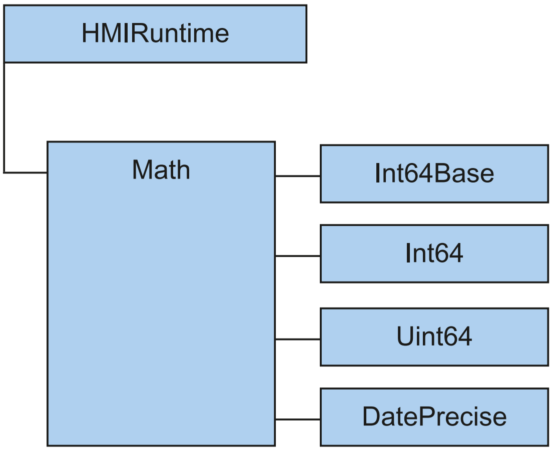
"Int64Base" object (Page 500)

"Or" method (Int64.OrUint64.Or) (Page 525)

Object "DatePrecise" (RT Uni)

"DatePrecise" description (RT Uni)

Description



The "DatePrecise" object ("HMIDatePrecise" type) represents high-resolution time information with a resolution of 100 ns as a 64-bit integer value.

It contains methods for converting between various time stamp formats. The following representations are supported:

- Number of milliseconds since 1970-01-01T00:00:00
- "DOMHighResTimeStamp":
  Number of milliseconds since 1970-01-01T00:00:00Z and the fraction for microseconds (resolution is approximately 5 µs).
- "hrtime"
  Array of two numbers [secs, nanosecs]
  - "secs": Number of seconds since 1970-01-01T00:00:00Z
  - "nanosecs": Number of nanoseconds (the resolution is limited to 100 nanoseconds). This data type is taken from "node.js" (https://nodejs.org/api/process.html#process_process_hrtime_time).
- "fileTime": Number of 100 nanosecond intervals since 1601-01-01T00:00:00Z (corresponds to "Win32 "FILETIME").
  This is the internal format and therefore the exact representation in terms of accuracy and value range.

The object is used, for example, to enable commenting of alarms via scripting.

Use

Note

All internal time information is available as "DatePrecise" objects. To use the native JavaScript functions for handling data objects, you must first convert the "DatePrecise" object to a JavaScript "Date" object.

Type identifier in JavaScript

HMIDatePrecise

Example

Checks whether the time information is a "DatePrecise" object:

Copy code

```
function AlarmTriggerFunction(errorCode, SystemName, alarmResultArray) {
    let nanoSeconds;
    //check first if RaiseTime is a DatePrecise object
    if (alarmResultArray[0].RaiseTime instanceof
HMIRuntime.Math.DatePrecise) {
        nanoSeconds = alarmResultArray[0].RaiseTime.GetNanoseconds();
    }
}
```

Converts a "DatePrecise" object to a JavaScript "Date" object so that the native JavaScript "getFullYear" function can be used:

Copy code

```
function AlarmTriggerFunction(errorCode, SystemName, alarmResultArray) {
    //convert to JavaScript-Date object first
    let fullYear = new Date(alarmResultArray[0].RaiseTime).getFullYear();
}
```

Properties (RT Uni)

Properties

The "DatePrecise" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "DatePrecise" (RT Uni)

Overview (RT Uni)

Methods

The "DatePrecise" object has the following methods:

Methods	Description
GetFiletime	Returns the time information as the type "FILETIME".
GetHrTime	Returns the time information as high-resolution time (hrtime) as Array in the format [seconds, nanoseconds].
GetMicroseconds	Returns the microseconds of time information in the value range of 0 ... 999.
GetNanoseconds	Returns the nanoseconds of time information in the value range of 0 ... 999.
GetTime	Returns the time information as the type "DOMHighResTimeStamp".
Item	Creates a precise time information as a 64-bit integer value ("DatePrecise" object) and returns it.
SetFiletime	Saves high-resolution time information as the type FILETIME.
SetHrTime	Saves high-resolution time information of the type "hrtime".
SetMicroseconds	Saves the microseconds of high-resolution time information.
SetNanoseconds	Saves the microseconds of high-resolution time information.
SetTime	Saves high-resolution time information of the type "DOMHighResTimeStamp".
toString	Converts the time information of a "DatePrecise" object into a string.
valueOf	Returns the time information saved in a "DatePrecise" object as "DOMHighResTimeStamp".

"GetFiletime" method (DatePrecise.GetFiletime) (RT Uni)

Description

Returns the time information as the type "FILETIME".

The type corresponds to the format of Win32 "FILETIME": Number of 100 nanosecond intervals since 1601-01-01T00:00:00Z

Member

Method of the "DatePrecise" object.

Syntax

```
Object.GetFiletime();
```

Required: An object of type "HMIDatePrecise".

Parameters

--

Return value

Object of type "HMIInt64"

See also

"SetFiletime" method (DatePrecise.SetFiletime) (Page 538)

"GetHrTime" method (DatePrecise.GetHrTime) (RT Uni)

Description

Returns the time information as high-resolution time (hrtime) as an array in the format [seconds, nanoseconds].

Example: 2020-10-04 11:30:22.5454118 is returned as [13238335822, 545411800].

Member

Method of the "DatePrecise" object

Syntax

```
Object.GetHrTime();
```

Required: An object of type "HMIDatePrecise".

Parameters

--

Return value

hrtime

See also

"SetHrTime" method (DatePrecise.SetHrTime) (Page 539)

"GetMicroseconds" method (DatePrecise.GetMicroseconds) (RT Uni)

Description

Returns the microseconds of time information in the value range of 0 ... 999.

Member

Method of the "DatePrecise" object.

Syntax

```
Object.GetMicroseconds();
```

Required: An object of type "HMIDatePrecise".

Parameters

--

Return value

UInt16

See also

"SetMicroseconds" method (DatePrecise.SetMicroseconds) (Page 540)

"GetNanoseconds" method (DatePrecise.GetNanoseconds) (RT Uni)

Description

Returns the nanoseconds of time information in the value range of 0 ... 999.

Member

Method of the "DatePrecise" object.

Syntax

```
Object.GetNanoseconds();
```

Required: An object of type "HMIDatePrecise".

Parameters

--

Return value

UInt16

See also

"SetNanoseconds" method (DatePrecise.SetNanoseconds) (Page 540)

"GetTime" method (DatePrecise.GetTime) (RT Uni)

Description

Returns the time information as the type "DOMHighResTimeStamp".

The type corresponds to the following format: Number of milliseconds since 1970-01-01T00:00:00Z and the fraction for microseconds (resolution is approximately 5 µs).

Member

Method of the "DatePrecise" object.

Syntax

```
Object.GetTime();
```

Required: An object of type "HMIDatePrecise".

Parameters

--

Return value

DOMHighResTimeStamp

See also

"SetTime" method (DatePrecise.SetTime) (Page 541)

"Item" method (DatePrecise.Item) (RT Uni)

Description

Creates a precise time information as a 64-bit integer value ("DatePrecise" object) and returns this.

Member

Method of the "DatePrecise" object

Syntax

```
Object[.Item]
([year,month,day,hours,seconds,milliseconds,microseconds,nanoseconds
]);
```

Object

Required: An object of type "HMIDatePrecise".

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "DatePrecise" object.

Parameters

`year`

Type: Variant, optional

Full year of the time information. The year can be specified as follows:

- Object of type "DatePrecise"
- JavaScript object of type "Date"
- Time stamp of type "DOMHighResTimeStamp"
- Time stamp of type "hrtime"
- Number of milliseconds

`month`

Optional, type: UInt8

Number that represents a month. (0 = January … 11 = December)

`day`

Optional, type: UInt8

Number that represents a day (1 ... 31)

`hours`

Optional, type: UInt8

Number that represents an hour (0 ... 23)

`minutes`

Optional, type: UInt8

Number that represents a minute (0 ... 59)

`seconds`

Optional, type: UInt8

Number that represents a second (0 ... 59)

`milliseconds`

Optional, type: UInt16

Number that represents a millisecond (0 ... 999)

`microseconds`

Optional, type: UInt16

Number that represents a microsecond (0 ... 999)

`nanoseconds`

Optional, type: UInt16

Number that represents a nanosecond (0 ... 999)

Return value

Object of type "HMIDatePrecise"

See also

"Math" area (Page 494)

"SetFiletime" method (DatePrecise.SetFiletime) (RT Uni)

Description

Saves high-resolution time information as the type FILETIME.

Member

Method of the "DatePrecise" object.

Syntax

```
Object.SetFiletime([fileTime]);
```

Required: An object of type "HMIUint64".

Parameters

fileTime

Optional, type: "HMIUint64" object

Time information in the format of Win32 "FILETIME": Number of 100 nanosecond intervals since 1601-01-01T00:00:00Z

Return value

--

See also

"GetFiletime" method (DatePrecise.GetFiletime) (Page 533)

"SetHrTime" method (DatePrecise.SetHrTime) (RT Uni)

Description

Saves high-resolution time information of the type "hrtime".

Member

Method of the "DatePrecise" object.

Syntax

```
Object.SetHrTime(hrtime);
```

Required: An object of type "HMIDatePrecise".

Parameters

hrtime

Type: Float[]

Time information of type "hrtime" as an array in format [seconds, nanoseconds]: for example, 2020-10-04 11:30:22.5454118 as [13238335822, 545411800]. The type is also returned by the "GetHrTime" method.

Return value

--

See also

"GetHrTime" method (DatePrecise.GetHrTime) (Page 534)

"SetMicroseconds" method (DatePrecise.SetMicroseconds) (RT Uni)

Description

Saves the microseconds of high-resolution time information.

Member

Method of the "DatePrecise" object.

Syntax

```
Object.SetMicroseconds(microSeconds);
```

Required: An object of type "HMIDatePrecise".

Parameters

`microSeconds`

Type: UInt16

Microseconds of time information in the value range of 0 ... 999

Return value

--

See also

"GetMicroseconds" method (DatePrecise.GetMicroseconds) (Page 535)

"SetNanoseconds" method (DatePrecise.SetNanoseconds) (RT Uni)

Description

Saves the microseconds of high-resolution time information.

Member

Method of the "DatePrecise" object.

Syntax

```
Object.SetNanoseconds(nanoSeconds);
```

Required: An object of type "HMIDatePrecise".

Parameters

nanoSeconds

Type: UInt16

Nanoseconds of time information in the value range of 0 ... 999

Return value

--

See also

"GetNanoseconds" method (DatePrecise.GetNanoseconds) (Page 535)

"SetTime" method (DatePrecise.SetTime) (RT Uni)

Description

Saves high-resolution time information of the type "DOMHighResTimeStamp".

The type corresponds to the following format: Number of milliseconds since 1970-01-01T00:00:00Z and the fraction for microseconds (resolution is approximately 5 µs).

Member

Method of the "DatePrecise" object.

Syntax

```
Object.SetTime(time);
```

Required: An object of type "HMIDatePrecise".

Parameters

time

Type: Float

Time information of type "DOMHighResTimeStamp": Number of milliseconds since 1970-01-01T00:00:00Z and the fraction for microseconds (resolution is approximately 5 µs). The type is also returned by the "GetTime" method.

Return value

--

See also

"GetTime" method (DatePrecise.GetTime) (Page 536)

""toString" method (DatePrecise.toString) (RT Uni)

Description

Converts the time information of a "DatePrecise" object into a string.

The fixed format of the character string is "yyyy-mm-dd hh:mm:ss.HundredNanoSeconds", for example, "2020-07-04 11:30:22.5454118".

Member

Method of the "DatePrecise" object

Syntax

```
Object.toString();
```

Required: An object of type "HMIDatePrecise".

Parameters

--

Return value

String

""valueOf" method (DatePrecise.valueOf) (RT Uni)

Description

Returns the time information saved in a "DatePrecise" object as "DOMHighResTimeStamp".

The value can represent either a specific point in time (in milliseconds since 01/01/1970) or the difference between two points in time.

The value is specified in the unit milliseconds. The accuracy is up to 5 µs.

Member

Method of the "DatePrecise" object

Syntax

```
Object.valueOf();
```

Required. An object of type "HMIDatePrecise".

Parameters

--

Return value

DOMHighResTimeStamp

7.8.1.8 "ParameterSetTypes" area (RT Uni)

"ParameterSetTypes" object (RT Uni)

"ParameterSetTypes" object (RT Uni)

Description

Represents a list of "Parameter data set types".

Type identifier in JavaScript

HMIParameterSetTypes

Properties (RT Uni)

Properties

The "Template" object has the following properties.

Properties	Type	Access	Description
-	-	-	-

Overview (RT Uni)

Methods

The "ParameterSetType" object has the following methods:

Methods	Description
Item(ParameterSetTypes.Item)	Returns an object of the type "ParameterSetType".

Methods of "ParameterSetTypes" (RT Uni)

"Item" method (ParameterSetTypes.Item) (RT Uni)

Description

Returns an object of the type "ParameterSetType".

Member

Method of the "ParameterSetTypes" object

Syntax

```
HMIRuntime.ParameterSetTypes.Item(parameterSetId);
```

Parameter

parameterSetTypeId

Type: String

Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.

Return value

Object of the type "HMIParameterSetType"

"ParameterSetType" object (RT Uni)

"ParameterSetType" object (RT Uni)

Description

Represents the "Parameter set type" object.

Type identifier in JavaScript

HMIParameterSetType

Properties (RT Uni)

Properties

The "ParameterSetType" object has the following properties:

Properties	Type	Access	Description
ParameterSets	HMIParameter-Sets	read only	Returns the "ParameterSets" list.

Overview (RT Uni)

Methods

The "ParameterSetType" object has the following methods:

Methods	Description
-	-

"ParameterSets" object (RT Uni)

"ParameterSets" object (RT Uni)

Description

Represents the list of "Parameter sets".

Type identifier in JavaScript

HMIParameterSets

Properties (RT Uni)

Properties

The "ParameterSet" object has the following properties:

Properties	Type	Access	Description
-	-	-	-

Overview (RT Uni)

Methods

The "ParameterSet" object has the following methods:

Methods	Description
Item	Returns an object of the type "ParameterSet".

Methods of "ParameterSets" (RT Uni)

"Item" method (ParameterSets.Item) (RT Uni)

Description

Returns an object of the type "ParameterSet".

Member

Method of the "ParameterSets" object

Syntax

```
HMIRuntime.ParameterSets.Item(parameterSetId);
```

Parameter

parameterSetId

Type: String

Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, execution is terminated.

Return value

Object of the type "HMIParameterSet"

"ParameterSet" object (RT Uni)

"ParameterSet" object (RT Uni)

Description

Represents the "Parameter set" object.

Type identifier in JavaScript

HMIParameterSet

Properties (RT Uni)

Properties

The "ParameterSet" object has the following properties:

Properties	Type	Access	Description
-	-	-	-

Overview (RT Uni)

Methods

The "ParameterSet" object has the following methods:

Methods	Description
LoadAndWrite	Loads the parameter set values from the memory and writes them to the PLC.
ReadAndSave	Reads a parameter set from the PLC and writes the parameter set to the local memory.

Methods of "ParameterSet" (RT Uni)

"LoadAndWrite" method (ParameterSet.LoadAndWrite) (RT Uni)

Description

Loads the parameter set values from the memory and writes them to the PLC.

Member

Method of the "ParameterSet" object

Syntax

```
HMIRuntime.ParameterSet.LoadAndWrite(parameterSetTypeID,
parameterSetID, outputStatus,processingStatus);
```

Parameter

`parameterSetTypeID`

Type: STRING

Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.

`parameterSetID`

Type: STRING

Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, execution is terminated.

`outputStatus`

Type: BOOL

TRUE, if completion of the write operation is confirmed with a message.

`processingStatus`

Type: Variant

Indicates the execution status of a function:

- 2 = Function is being executed.
- 4 = Function successfully executed.
- 12 - Function was cancelled.

Return value

ErrorCode

"ReadAndSave" method (ParameterSet.ReadAndSave) (RT Uni)

Description

Reads a parameter set from the PLC and writes the parameter set to the local memory.

Member

Method of the "ParameterSet" object

Syntax

```
HMIRuntime.ParameterSet.ReadAndSave(parameterSetTypeID,
parameterSetID, overWrite,outputStatus,processingStatus);
```

Parameter

`parameterSetTypeID`

Type: STRING

Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.

`parameterSetID`

Type: STRING

Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, execution is terminated.

`overWrite`

Type: hmiOverwrite

Specifies whether the values in the memory are overwritten with the values from the import file:

- 0 = Overwriting is not allowed.
- 1 = Overwriting is allowed.

The following cases are differentiated:

- If the name / ID of the specified parameter set exists, the values in the PLC are overwritten with the parameter set values in the memory if overwriting is allowed.
- If overwriting is not allowed, the data in the memory is not replaced. The process tag is updated to the state of the system function, if configured accordingly.

`outputStatus`

Type: BOOL

TRUE, if the conclusion of the read operation is confirmed with a message.

`processingStatus`

Type: Variant

Indicates the execution status of a function:

- 2 = Function is being executed.
- 4 = Function successfully executed.
- 12 - Function was cancelled.

Return value

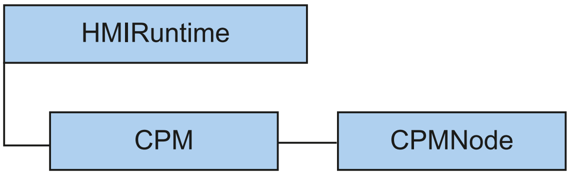
ErrorCode

7.8.1.9 "PlantModel" area (RT Uni)

"PlantModel" object (RT Uni)

"PlantModel" description (RT Uni)

Description



The "PlantModel" object ("HMIPlantModel" type) represents the common plant model of the graphical runtime system. You reference all object instances ("PlantObject" objects) and properties of the common plant model by means of the "PlantModel" object.

Note

The common plant model can map a plant hierarchy in runtime. Each hierarchy consists of object instances which represent a component or a function part of the plant.

Each object instance is assigned to a hierarchy node of a plant hierarchy. Through this assignment the object instance has a unique position and address in a hierarchy.

The address in the hierarchy is represented with a path. This hierarchy path is used to reference specific objects in the properties and methods of the common plant model.

Use

You reference the instanced objects of the common plant model by means of the "PlantModel" object. These object instances represent specific components or parts of the plant. This means that you have access to all the properties and methods of these objects.

Type identifier in JavaScript

HMIPlantModel

Properties (RT Uni)

Properties

The "PlantModel" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "PlantModel" (RT Uni)

Overview (RT Uni)

Methods

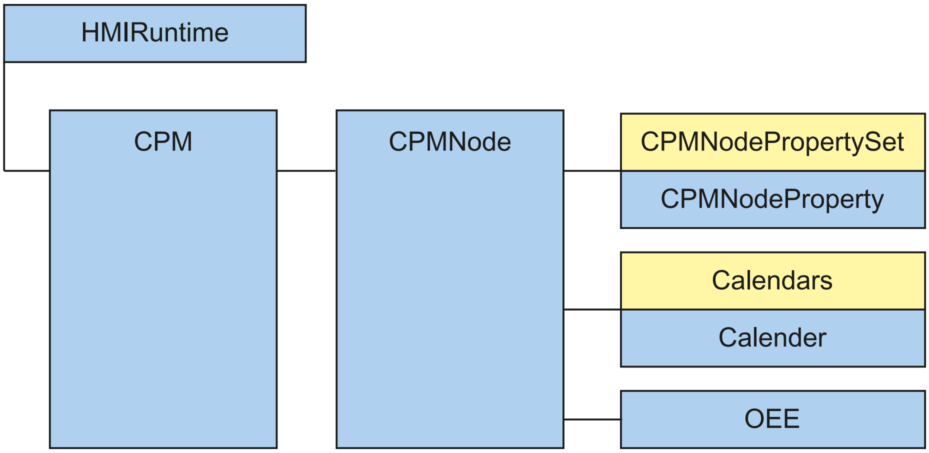
The "PlantModel" object has the following methods:

Methods	Description
GetPlantObject	Returns an object instance.
GetPlantObjectByPath	
GetPlantObjectsByExpression	Returns an array of object instances.
GetPlantObjectsByPropertyNames	
GetPlantObjectsByType	

"PlantObject" object (RT Uni)

"PlantObject" description (RT Uni)

Description



The "PlantObject" object ("HMIPlantObject" type) represents the object instances of the common plant model.

Note

The common plant model can map any number of plant hierarchies in runtime. Each hierarchy consists of object instances which represent a component or a function part of the plant.

Each object instance is assigned to a hierarchy node in a plant hierarchy. Through this assignment the object instance has a unique position and address in a hierarchy.

The address in the hierarchy is represented with a path. This hierarchy path is used to reference specific objects in the properties and methods of the common plant model.

Use

The "PlantObject" object gives you access to all object properties or adjacent object instances in the plant hierarchy.

Type identifier in JavaScript

HMIPlantObject

Properties (RT Uni)

Properties

The "PlantObject" object has the following properties:

Properties	Type	Access	Description
Children	Object	read only	Returns all child object instances of an object instance in the hierarchy.
CurrentPlantView	String	read/ write	Specifies the view that is displayed in the "PlantModel View".
Name	String	read only	Returns the name of the object or specifies it.
Parent	Object	read only	Returns the higher-level object instance (parent), which contains the current object instance as child.
PlantViewPaths	StringStringMap	read only	Returns the currently displayed path of a view.

Methods of "PlantObject" (RT Uni)

Overview (RT Uni)

Description

The "PlantObject" object has the following methods:

Methods	Description
GetChild	Returns the child object instance ("CPMNode" object) of an object instance of the common plant model.
GetProperties	Returns the properties of object instances ("CPMNode" objects) of the common plant model as "CPMNodePropertySet" list.

"GetChild" method (CPMNode.GetChild) (RT Uni)

Description

The "GetChild" method returns the child object instance ("CPMNode" object) of an object instance of the common plant model.

Member

Method of the "CPMNode" object

Syntax

```
Object.GetChild(ChildName);
```

Object

Required. An object of the "HMICPMNode" type

Parameters

`ChildNames`

Type: String

Name of the object instance in the hierarchy

Return

Object of the "HMICPMNode" type

Example

Return the child of the "cpmNode1" object instance with the name "TemperatureSensor":

```
var cpmChildNode = cpmNode1.GetChild('TemperatureSensor');
```

"GetProperties" method (CPMNode.GetProperties) (RT Uni)

Description

The "GetProperties" method returns the properties of object instances ("CPMNode" objects) of the common plant model as CPMNodePropertySet list.

Member

Method of the "CPMNode" object

Syntax

```
Object.GetProperties([PropertyNames]);
```

Object

Required. An object of type "HMICPMNode"

Parameters

`PropertyNames`

Optional, type: String, String[]

One or more properties of an object instance

Return

Array with objects of type "HMICPMNode"

Example

Return all properties of the "cpmNode" object instance as "cpmPropertySet" list:

```
var cpmPropertySet = cpmNode.GetProperties();
```

or only return the properties "Speed" and "Temperature" as list:

```
var cpmPropertySet = cpmNode.GetProperties(['Speed', 'Temperature']);
```

Output the name of all properties of the "cpmNode" object instance:

Copy code

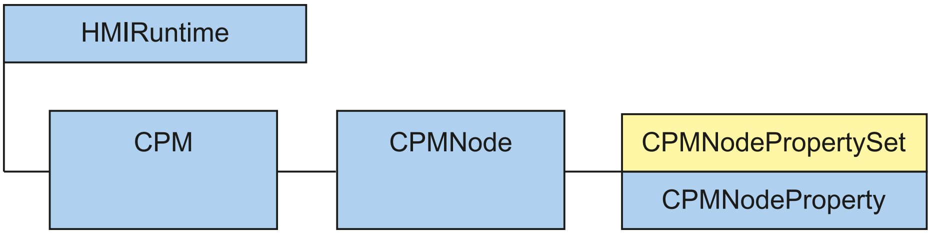
```
var cpmPropertySet = cpmNode.GetProperties();

for (let i in cpmPropertySet)
{
    var cpmProperty = cpmPropertySet[i];
    HMIRuntime.Trace(cpmProperty.Name);
}
```

"PlantObjectProperty" object (RT Uni)

"PlantObjectProperty" description (RT Uni)

Description



The "PlantObjectProperty" object ("PlantObjectProperty" type) represents the property of an object instance ("PlantObject" object) of the common plant model.

A "PlantObjectProperty" object is returned by the "PlantObjectPropertySet" list or the `PlantObject.GetProperties` method.

Use

The "PlantObjectProperty" object gives you read and write access to the properties of an object instance.

Type identifier in JavaScript

HMIPlantObjectProperty

Properties (RT Uni)

Properties

The "PlantObejctProperty" object has the following properties:

Properties	Type	Access	Description
LastError	Error-Code	read only	Returns an error code for the last faulty read or write operation.
Name	String	read only	Returns the name of the object or specifies it.
QualityCode (Page 556)	UInt32	read only	Returns the quality level of a tag value after reading a tag.
TimeStamp	Date-Time	read only	Returns the time stamp of the last read operation. The value 0 is returned after writing or failed reading.
Value	Variant	read/ write	Specifies a value for the object being used or returns it.

See also

"QualityCode" property (Page 556)

Special properties (RT Uni)

"QualityCode" property (RT Uni)

Description

Returns the quality level of a tag value after reading a tag.

The quality code has the binary 8-bit structure **QQSSSSLL.** The first two positions (QQ) of the quality code define the quality of the tag value:

Quality	Description	Q	Q	S	S	S	S	L	L
Bad	Tag value cannot be used.	0	0	-	-	-	-	-	-
Uncertain	Quality of the tag value is worse than usual. However, it might still be possible to use the tag value.	0	1	-	-	-	-	-	-

Quality	Description	Q	Q	S	S	S	S	L	L
Good (Non-Cascade)	Quality of the tag value is good. Attention should be paid to substatus.	1	0	-	-	-	-	-	-
Good (Cascade)	Quality of the tag value is good. Tag value could be used.	1	1	-	-	-	-	-	-

Positions 3 to 6 (SSSS) of the quality code specify the substatus of the quality. Positions 7 and 8 (LL) are optional and define possible limits.

Syntax

```
Object.QualityCode
```

`Object`

Required. An object from the "Availability" section.

Quality code of tags

The realized quality codes are listed in the following table. The table begins with the worst quality code and ends with the best quality code. The best quality code has the lowest priority, while the worst quality has the highest priority. If several statuses occur simultaneously for a tag in the processing chain, the poorest code is passed on.

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x23	Bad	Device passivated - Diagnostic alerts inhibited	0	0	1	0	0	0	1	1
0x3F	Bad	Function check - Local override	0	0	1	1	1	1	1	1
0x1C	Bad	Out of Service - The value is not reliable because the block is not being evaluated, and may be under construction by a configurer. Set if the block mode is O/S.	0	0	0	1	1	1	-	-
0x73	Uncertain	Simulated value - Start	0	1	1	1	0	0	1	1
0x74	Uncertain	Simulated value - End	0	1	1	1	0	1	-	-
0x84	Good (Non-Cascade)	Active Update event - Set if the value is good and the block has an active Update event.	1	0	0	0	0	1	-	-
0x24	Bad	Maintenance alarm - More diagnostics available.	0	0	1	0	0	1	-	-
0x18	Bad	No Communication, with no usable value - Set if there has never been any communication with this value since it was last "Out of Service".	0	0	0	1	1	0	-	-
0x14	Bad	No Communication, with last usable value - Set if this value had been set by communication, which has now failed.	0	0	0	1	0	1	-	-
0x0C	Bad	Device Failure - Set if the source of the value is affected by a device failure.	0	0	0	0	1	1	-	-
0x10	Bad	Sensor failure	0	0	0	1	0	0	-	-
0x08	Bad	Not Connected - Set if this input is required to be connected and is not connected.	0	0	0	0	1	0	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x04	Bad	Configuration Error - Set if the value is not useful because there is some inconsistency regarding the parameterization or configuration, depending on what a specific manufacturer can detect.	0	0	0	0	0	1	-	-
0x00	Bad	Non-specific - There is no specific reason why the value is bad. Used for propagation.	0	0	0	0	0	0	-	-
0x28	Bad	Process related - Substitute value	0	0	1	0	1	0	-	-
0x2B	Bad	Process related - No maintenance	0	0	1	0	1	0	1	1
0x68	Uncertain	Maintenance demanded	0	1	1	0	1	0	-	-
0x60	Uncertain	Simulated value - Set when the process value is written by the operator while the block is in manual mode.	0	1	1	0	0	0	-	-
0x64	Uncertain	Sensor calibration	0	1	1	0	0	1	-	-
0x5C	Uncertain	Configuration error	0	1	0	1	1	1	-	-
0x58	Uncertain	Sub-normal	0	1	0	1	1	0	-	-
0x54	Uncertain	Engineering Unit Range Violation - Set if the value lies outside of the set of values defined for this parameter. The Limits define which direction has been exceeded.	0	1	0	1	0	1	-	-
0x50	Uncertain	Sensor conversion not accurate	0	1	0	1	0	0	-	-
0x4B	Uncertain	Substitute (constant)	0	1	0	0	1	0	1	1
0x78	Uncertain	Process related - No maintenance	0	1	1	1	1	0	-	-
0x4C	Uncertain	Initial Value - Value of volatile parameters during and after reset of the device or of a parameter.	0	1	0	0	1	1	-	-
0x48	Uncertain	Substitute value - Predefined value is used instead of the calculated one. This is used for fail safe handling.	0	1	0	0	1	0	-	-
0x44	Uncertain	Last Usable Value - Whatever was writing this value has stopped doing so. This is used for fail safe handling.	0	1	0	0	0	1	-	-
0x40	Uncertain	Non-specific - There is no specific reason why the value is uncertain.	0	1	0	0	0	0	-	-
0xE0	Good (Cascade)	Initiate Fail Safe (IFS) - The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe.	1	1	1	0	0	0	-	-
0xD8	Good (Cascade)	Local Override (LO) - The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control must be propagated to a function running in a host system for alarm and display purposes. This also implies "Not Invited".	1	1	0	1	1	0	-	-
0xD4	Good (Cascade)	Do Not Select (DNS) - The value is from a block which should not be selected, due to conditions in or above the block.	1	1	0	1	0	1	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0xCC	Good (Cascade)	Not Invited (NI) - The value is from a block which does not have a target mode that would use this input.	1	1	0	0	1	1	-	-
0xC8	Good (Cascade)	Initialization Request (IR) - The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.	1	1	0	0	1	0	-	-
0xC4	Good (Cascade)	Initialization Acknowledge (IA) - The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).	1	1	0	0	0	1	-	-
0xC0	Good (Cascade)	OK - No error or special condition is associated with this value.	1	1	0	0	0	0	-	-
0xA0	Good (Non-Cascade)	Initiate fail safe	1	0	1	0	0	0	-	-
0x98	Good (Non-Cascade)	Unacknowledged Critical Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.	1	0	0	1	1	0	-	-
0x94	Good (Non-Cascade)	Unacknowledged Advisory Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.	1	0	0	1	0	1	-	-
0x90	Good (Non-Cascade)	Unacknowledged Update event - Set if the value is good and the block has an unacknowledged Update event.	1	0	0	1	0	0	-	-
0x8C	Good (Non-Cascade)	Active Critical Alarm - Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.	1	0	0	0	1	1	-	-
0x88	Good (Non-Cascade)	Active Advisory Alarm - Set if the value is good and the block has an active Alarm with a priority less than 8.	1	0	0	0	1	0	-	-
0xA8	Good (Non-Cascade)	Maintenance demanded	1	0	1	0	1	0	-	-
0xA4	Good (Non-Cascade)	Maintenance required	1	0	1	0	0	1	-	-
0xBC	Good (Non-Cascade)	Function check - Local override	1	0	1	1	1	1	-	-
0x80	Good (Non-Cascade)	OK - No error or special condition is associated with this value.	1	0	0	0	0	0	-	-

Limit

The quality codes can be further subdivided by limits. Limits are optional.

Description	Q	Q	S	S	S	S	L	L
O.K. - The value is free to move.	-	-	-	-	-	-	0	0
Low limited - The value has acceded its low limits.	-	-	-	-	-	-	0	1

Description	Q	Q	S	S	S	S	L	L
High limited - The value has acceded its high limits.	-	-	-	-	-	-	1	0
Constant (high and low limited) - The value cannot move, no matter what the process does.	-	-	-	-	-	-	1	1

See also

Properties (Page 556)

Overview (Page 673)

Methods of "PlantObjectProberty" (RT Uni)

Overview (RT Uni)

Methods

The "PlantObjectProperty" object has the following methods:

Methods	Description
Read	Reads out a property ("CPMNodeProperty" object) of an object instance of the common plant model.
Write	Writes the value of the property ("CPMNodeProperty" object) of an object instance of the common plant model.

"Read" method (CPMNodeProperty.Read) (RT Uni)

Description

The "Read" method reads a property ("CPMNodeProperty" object) of an object instance of the common plant model. The value, the Quality Code and the time stamp of the property are determined when the property is read.

The method executes a synchronous read operation. When completed, you can use the "LastError" property of the "CPMNodeProperty" object to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "ReadAsync" method of the "CPMNodePropertySet"object.

Member

Method of the "CPMNodeProperty" object

Syntax

```
Object.Read();
```

Object

Required. An object of the "HMICPMNodeProperty" type

Parameters

--

Return

Variant

Example

Reads the value of the "PPP" property of an object instance:

```
 var value = cpmNode.Properties.Item('PPP').Read();
```

"Write" method (CPMNodeProperty.Read) (RT Uni)

Description

The "Write" method writes the value of the property ("CPMNodeProperty" object) of an object instance of the common plant model. You must first set the values of the "CPMNodeProperty" objects with the "Value" property. The value of the "Value" property must not correspond to the actual value of the "CPMNodeProperty" object once the write operation is complete. If you want to update the "CPMNodeProperty" objects, perform a Read method.

The method executes an asynchronous write operation. The "LastError" property is written after the write operation has been completed. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method of the "CPMNodePropertySet" object.

The properties "QualityCode" and "TimeStamp" of the "CPMNodeProperty" object are not determined during writing.

Member

Method of the "CPMNodeProperty" object

Syntax

```
Object.Write([value]);
```

Object

Required. An object of the "HMICPMNodeProperty" type

Parameters

value

Optional, type: Variant

Writes the value of the property of an object instance:

- Specify value
  The specified value overwrites the current value of the "Value" property of the "CPMNodeProperty" object.
- Without value
  The current value of the "Value" property of the "CPMNodeProperty" object is written.

Return

--

Example

Writes a new value of the "PPP" property of an object instance:

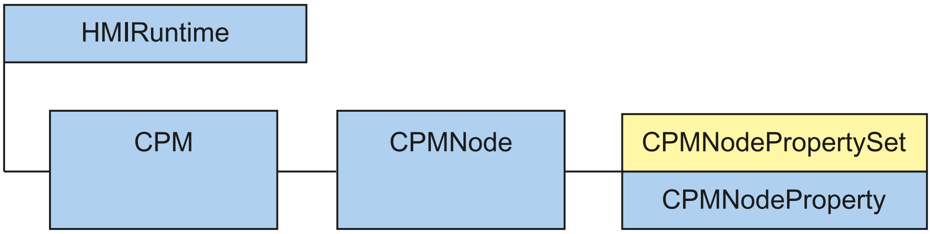
Copy code

```
cpmNode.Properties.Item('PPP').Write(3000);
```

"PlantObjectPropertySet" object (RT Uni)

"PlantObjectPropertySet" description (RT Uni)

Description



The "PlantObjectPropertySet" object ("PlantObjectPropertySet" type) is a list of "PlantObjectProperty" objects that give you optimized access to the properties of the object instances of the common plant model.

After initialization of the "PlantObjectPropertySet" object, you can read and write multiple properties in a single call. Simultaneous access has better performance and a lower communication load than single access to multiple properties.

You reference a "PlantObjectPropertySet" object via the "PlantObject" object.

Use

The "CPMNodePropertySet" object gives you access to all properties or a subset of properties of an object instance.

The "CPMNodePropertySet" object is a list and can be counted and enumerated. You can access the "CPMNodePropertySet" list through the index or the "item" method.

Type identifier in JavaScript

HMIPlantObjectPropertySet

Properties (RT Uni)

Properties

The "PlantObjectPropertySet" object has the following properties:

Properties	Type	Access	Description
Count	UInt32	read only	Returns the number of elements in the specified list.

Methods of "PlantObjectPropertySet" (RT Uni)

Overview (RT Uni)

Methods

The "PlantObjectPropertySet" object has the following methods:

Methods	Description
Add	Adds one or more properties ("CPMNodeProperty" objects) to an existing "CPMNodePropertySet" list. The properties are referenced by the name.
Item	Returns a property ("CPMNodeProperty" object) of an object instance from the "CPMNodePropertySet" list.
Read	Reads in all properties ("CPMNodeProperty" objects) of the "CPMNodePropertySet" list.
ReadAsync	Reads in all properties ("CPMNodeProperty" objects) of the "CPMNodePropertySet" list.
Remove	Removes one or more properties ("CPMNodeProperty" objects) from an existing "CPMNodePropertySet" list.
Write	Writes the values of all "CPMNodeProperty" objects of the "CPMNodePropertySet" list.
WriteAsync	Writes the values of all "CPMNodeProperty" objects of the "CPMNodePropertySet" list.

"Add" method (CPMNodePropertySet.Add) (RT Uni)

Description

The "Add" method adds one or more properties ("CPMNodeProperty" objects) to an existing "CPMNodePropertySet" list. The properties are referenced by the name.

Member

Method of the "CPMNodePropertySet" object

Syntax

```
Object.Add(name);
```

Object

Required. An object of type "HMICPMNodePropertySet"

Parameters

`name`

Type: String or String[]

Name of "CPMNodeProperty" objects that are added to the list.

The following data types are supported:

- Name of a CPM object property
- Array with names of CPM object properties

Return

Array with objects of type"CPMNodeProperty" that were newly added with the method.

Example

The properties "Temp3", "Temp4", "Temp5" are added to an existing "CPMNodePropertySet" object:

```
var nodePropertySet = cpmNode.GetProperties(['Temp','Temp2']);
var addedProperties_1 = nodePropertySet.Add(['Temp3','Temp4']);
var addedProperties_2 = nodePropertySet.Add('Temp5');
```

"Item" method (CPMNodePropertySet.Item) (RT Uni)

Description

The "Item" method returns a property ("CPMNodeProperty" object) of an object instance from the "CPMNodePropertySet" list.

Member

Method of the "CPMNodePropertySet" object

Syntax

```
Object.CPMNodePropertySet[.Item](name);
```

Object

Required. An object of the "HMICPMNodePropertySet" type

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "CPMNodePropertySet" object.

Parameters

`name`

Type: String, Int32

Name or index number (0…n) of a property of the "CPMNodePropertySet" list.

Note

The index number of a "CPMNodeProperty" object does not represent the order in which the "CPMNodeProperty" objects were added to the CPMNodePropertySet list.

Return

Object of the "HMICPMNodeProperty" type

Example

Read the value of the "PPP" property of the "cpmNode" object instance:

```
 var value = cpmNode.Properties.Item('PPP').Read();
```

or more simply as standard method:

```
 var value = cpmNode.Properties('PPP').Read();
```

"Read" method (CPMNodeProperty.Read) (RT Uni)

Description

The "Read" method reads in all properties ("CPMNodeProperty" objects) of the "CPMNodePropertySet" list. The value, the Quality Code and the time stamp of all properties are determined when the properties are read.

The method executes a synchronous read operation. When completed, you can use the "LastError" property of the "CPMNodeProperty" object to determine if the execution was successful. If you need the values of the write operation without blocking the script execution, use the "ReadAsync" method.

Member

Method of the "CPMNodePropertySet" object

Syntax

```
Object.Read();
```

Object

Required. An object of the "HMICPMNodePropertySet" type

Parameters

--

Return

--

Example

A "CPMNodePropertySet" object is created and the properties are read and output:

Copy code

```
var cpmPropertySet = cpmNode.GetProperties();

cpmPropertySet.Read();

for (let i in cpmPropertySet)
{
    var cpmProperty = cpmPropertySet[i];
    HMIRuntime.Trace(cpmProperty.Name + '=' + cpmProperty.Value);
}
```

"ReadAsync" method (CPMNodePropertySet.ReadAsync) (RT Uni)

Description

The "ReadAsync" method reads in all properties ("CPMNodeProperty" objects) of the "CPMNodePropertySet" list. The value, the Quality Code and the time stamp of all properties are determined when the properties are read.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "CPMNodePropertySet" object or the error code as parameter. Execution only fails (Promise rejected) when no "CPMNodeProperty" object of the "CPMNodePropertySet" object could be read.

Member

Method of the "CPMNodePropertySet" object

Syntax

```
HMIRuntime.CPM.CPMNode.CPMNodePropertySet.ReadAsync()
.then(function(HMICPMNodePropertySet) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

--

Return

Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMICPMNodePropertySet" as parameter of the "then()" handler.
- Promise rejected
  Error code as parameter of the "catch()" handler. This status only exists if no "HMICPMNodeProperty" object of the list could be read.

Example

A "CPMNodePropertySet" object is read in via an asynchronous call. Once all properties have been read out, they are output with names and values:

Copy code

```
var cpmPropertySet = cpmNode.GetProperties();

cpmPropertySet.ReadAsync();

.then(function (propertySet) { // async execution when values have been read
    for (let i in cpmPropertySet)
    {
        var cpmProperty = propertySet[i];
        HMIRuntime.Trace(cpmProperty.Name + '=' + cpmProperty.Value);
    }
})

.catch (function (err) {
    HMIRuntime.Trace("\nReading PropertySet failed\n");
});
```

"Remove" method (CPMNodePropertySet.Remove) (RT Uni)

Description

The "Remove" method removes one or more properties ("CPMNodeProperty" objects) from an existing "CPMNodePropertySet" list. The properties are referenced by the name.

Member

Method of the "CPMNodePropertySet" object

Syntax

```
Object.Remove(name);
```

Object

Required. An object of type "HMICPMNodePropertySet"

Parameters

name

Type: String or String[]

Name of "CPMNodeProperty" objects that are removed from the list.

The following data types are supported:

- Name of a CPM object property
- Array with names of CPM object properties

Return

--

Example

The property "Temp2" is removed in an existing "CPMNodePropertySet" object:

```
var nodePropertySet = cpmNode.GetProperties(['Temp','Temp2']);
nodePropertySet.Remove('Temp2');
```

"Write" method (CPMNodePropertySet.Write) (RT Uni)

Description

The "Write" method writes the values of all "CPMNodeProperty" objects of the "CPMNodePropertySet" list. You must first set the values of the "CPMNodeProperty" objects with the "Value" property. The value of the "Value" property must not correspond to the actual value of the "CPMNodeProperty" object once the write operation is complete. If you want to update the "CPMNodeProperty" objects, perform a Read method.

The method executes an asynchronous write operation. The "LastError" property is written after the write operation for each "CPMNodeProperty" property has been completed. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method.

The properties "QualityCode" and "TimeStamp" of the "CPMNodeProperty" objects are not determined during writing.

Member

Method of the "CPMNodePropertySet" object

Syntax

```
Object.Write();
```

Object

Required. An object of the "HMICPMNodePropertySet" type

Parameters

--

Return

--

Example

A "HMICPMNodePropertySet" object with two properties is generated. The values of properties are set and the "HMICPMNodePropertySet" object are written back:

Copy code

```
var cpmPropertySet = cpmNode.GetProperties(['Speed', 'Temperature']);

for (let i in cpmPropertySet)
{
    cpmPropertySet[i].Value = 4999;
}

cpmPropertySet.Write();
```

"WriteAsync" method (CPMNodePropertySet.WriteAsync) (RT Uni)

Description

The "WriteAsync" method writes the values of all "CPMNodeProperty" objects of the "CPMNodePropertySet" list. You must first set the values of the individual properties of the object instance with the "Value" property. The value of the "Value" property does not have to correspond to the actual current value of the "CPMNodeProperty" objects after completion of the write operation. If you want to update the "CPMNodeProperty" objects, execute a Read method.

The "LastError" property is written for each "CPMNodeProperty" object after completion of the write operation. This enables you to determine if the execution was successful.

The properties "QualityCode" and "TimeStamp" of the "CPMNodeProperty" objects are not determined during writing.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the

"CPMNodePropertySet" object or the error code as parameter. Execution only fails (Promise rejected) when no "CPMNodeProperty" object of the "CPMNodePropertySet" object could be written.

Member

Method of the "CPMNodePropertySet" object

Syntax

```
HMIRuntime.CPM.CPMNode.CPMNodePropertySet.WriteAsync()
.then(function(HMICPMNodePropertySet) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

--

Return

Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMICPMNodePropertySet" as parameter of the "then()" handler.
- Promise rejected
  Error code as parameter of the "catch()" handler. This status only exists if no "CPMNodeProperty" object of the list could be written.

Example

A "CPMNodePropertySet" object is generated. All properties are read, the values are changed and the "CPMNodePropertySet" object is written back asynchronously:

Copy code

```
var cpmPropertySet = cpmNode.GetProperties();

for (let i in cpmPropertySet)
{
    cpmPropertySet[i].Value = 4999;
}
cpmPropertySet.WriteAsync();

.then(function (propertySet) {
    HMIRuntime.Trace("\nPropertySet written\n");
})

.catch (function (err) {
    HMIRuntime.Trace("\nPropertySet not written\n");
});
```

7.8.1.10 "ScreenInterface" object (RT Uni)

"ScreenInterface" description (RT Uni)

Description

The "ScreenInterface" object enables access to screens and screen windows in runtime.

Type identifier in JavaScript

HMIScreenInterface

Properties (RT Uni)

Properties

The "ScreenInterface" object has the following properties:

Properties	Type	Access	Description
CurrentWindow	Object	read only	Returns the screen window which contains the current screen.
Items	Object	read only	Returns a list of all screen objects of the current screen.
Parent	Object	read only	Returns the higher-level object instance (parent), which contains the current object instance as child.

Properties	Type	Access	Description
ParentScreen	Object	read only	Returns the higher-level screen that contains the screen window of the current screen.
Windows	Object	read only	Returns the "Windows" object with the list of the screen windows.

Methods of "ScreenInterface" (RT Uni)

Overview (RT Uni)

Methods

The "ScreenInterface" object has the following methods:

Methods	Description
FindItem	Searches for and references screen windows or screen objects through their object path.

"FindItem" method (UI.FindItem, ScreenInterface.FindItem) (RT Uni)

Description

Searches for and references screen windows or screen objects through their object path.

Member

Method of the following objects:

- UI
- ScreenInterface

Syntax

```
Object.FindItem(ScreenItemPath);
```

Parameter

ScreenItemPath

Type: String

Object path of the searched screen window or screen object.

Note

The "UI.FindItem" method has a global search context and requires absolute object paths. The "Screen.FindItem" method has the current screen as the search context and can also use relative object paths.

Formulation of the object path

The syntax of the object path orients itself to the notation of tile system paths. The object path consists of the names of the screen windows (Screen Windows) and screen objects (Screen Items). The names are connected via a slash ("/") according to the hierarchical positioning. Screens (Screens) and their names are not used in the formulation.

Relative and absolute objects paths are distinguished by the prefix of the object path. The following prefixes can be used:

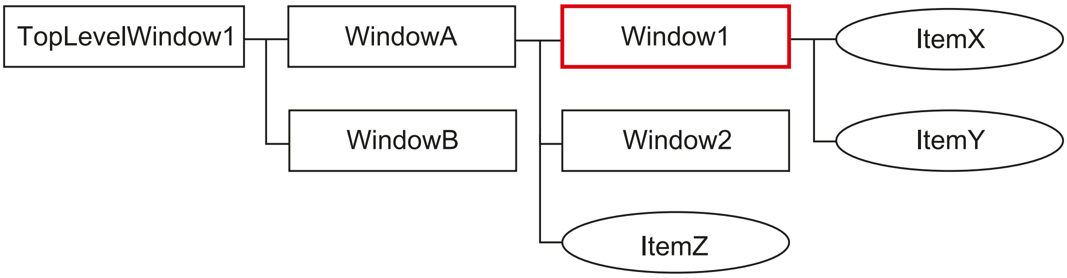
- Relative object path
  - "..": References the higher level screen window (parent) in the context of the current screen window.
  - ".": References the own screen window (self).
  - "": A screen object of the current screen window is referenced without prefix.
- Absolute object path
  - "/": References a screen window on the highest level, whose name must follow.
  - "~": References the screen window on the highest level in the own screen hierarchy.

Further rules for formulating an object path:

- The string ".." may be used several times in the object path, but only together at the beginning of the object path, for example, "../../Window5".
- If the object path does not end with a screen object name, a screen window is referenced.
- An automatic search is performed for screen objects of the object path in the screens of the referenced screen window. It is not permitted to specify a screen name.

Examples of object paths

The following window / screen object hierarchy is adopted for the following examples:



□ Current script context/screen window of the current screen

The following objects paths for addressing the object result from this:

Object path	Referenced object
ItemX	"ItemX" screen object
.	"Window1" screen window
./ItemX	"ItemX" screen object
..	"WindowA" screen window
../ItemZ	"ItemZ" screen object
../ItemZ/Axes/5	5th element of the axis list of the screen object "ItemZ"
../Window2	"Window2" screen window
../..	Screen windows "TopLevelWindow1" on the highest level
/TopLevelWindow1	Screen windows "TopLevelWindow1" on the highest level
~	Screen windows "TopLevelWindow1" on the highest level
/TopLevelWindow1/WindowB	"WindowB" screen window
~/WindowB	"WindowB" screen window
/	Invalid as the name of a screen window is missing on the highest level.

Return value

HmiScreenObjectBase

See also

"ScreenItem" object (Page 730)

7.8.1.11 "ScreenItems" area (RT Uni)

"AlarmControl" object (RT Uni)

"AlarmControl" description (RT Uni)

Description

Represents the "AlarmView" object.

Type identifier in JavaScript

HMIAlarmControl

Properties (RT Uni)

Properties

The "AlarmControl" object has the following properties:

Properties	Type	Access	Description
AcknowledgmentFlashingRate	HmiFlashingRate	read/ write	Specifies the flashing rate for alarms that have to be acknowledged.
ActiveAlarmsViewSetup (Page 578)	HmiAlarmPredefinedFilter	read/ write	Specifies which pre-defined alarm filter is applied for pending alarms.
AlarmDefinitionViewSetup (Page 578)	HmiAlarmPredefinedFilter	read/ write	Specifies which pre-defined alarm filter is applied.
AlarmSourceType (Page 579)	HmiAlarmSourceType	read/ write	Specifies the source of the alarms of the alarm window.
AlarmStatisticsView	Object	read/ write	Returns the "AlarmStatisticsView" object. Specifies an alarm window with columns and alarm lines for statistical calculations of alarms to be logged.
AlarmView	Object	read/ write	Specifies the "AlarmView" object.
AlwaysShowRecent	Bool	read/ write	Specifies whether the newest alarm is displayed at the beginning or end of the list based on the sorting.
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
Caption	String	read/ write	Specifies the text to be displayed in the header.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DefaultSortDirection (Page 579)	HmiSortDirection	read/ write	Specifies the sort sequence of the time column if no other sorting is active.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Filter	String	read/ write	Specifies a string for filtering active alarms. The syntax of the filter string is equivalent to the WHERE clause of an SQL command.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
Icon	String	read/ write	Specifies the icon.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.

Properties	Type	Access	Description
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
ResetFlashingRate	HmiFlashingRate	read/ write	Specifies the flashing rate for alarms that have to be reset: ● Slow (0): Slow ● Medium (1): Medium ● Fast (2): Fast
StatusBar	Object	read/ write	Specifies the "StatusBar" object.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
SuppressFlashing	Bool	read/ write	Specifies whether flashing is suppressed.
Systems	String[]Int32[]	read/ write	Specifies the name of the Runtime system for the grouping of active alarms. If no reference is specified, all known systems are used.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
TimeZone	HmiTimeZone	read/ write	Specifies the time zone.
ToolBar	Object	read/ write	Specifies the "ToolBar" object. Specifies the toolbar of the window.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
UseAlarmColors	Bool	read/ write	Specifies whether the configured color of the alarm is used.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.
WindowFlags (Page 580)	HmiWindowFlag	read/ write	Specifies the properties of the window.

See also

"ActiveAlarmsViewSetup" property (Page 578)

"AlarmDefinitionViewSetup" property (Page 578)

"AlarmSourceType" property (Page 579)

"DefaultSortDirection" property (Page 579)

"WindowFlags" property (Page 580)

Special properties (RT Uni)

"ActiveAlarmsViewSetup" property (RT Uni)

Description

Specifies which pre-defined alarm filter is applied for pending alarms:

- None (0): All alarms are displayed:
- OutOfService (1): Only disabled alarms.
- Suppressed (2): Only suppressed alarms
- SuppressedByDesign (4): Only alarms suppressed by design
- Shelved (8): Only shelved alarms

Syntax

```
Object.ActiveAlarmsViewSetup
```

Object

Required. An object from the "Availability" section.

See also

Properties (Page 576)

"AlarmDefinitionViewSetup" property (RT Uni)

Description

Specifies which pre-defined alarm filter is applied:

- None (0): All alarms are displayed:
- OutOfService (1): Only disabled alarms.
- Suppressed (2): Only suppressed alarms
- SuppressedByDesign (4): Only alarms suppressed by design
- Shelved (8): Only shelved alarms

Syntax

```
Object.AlarmDefinitionViewSetup
```

Object

Required. An object from the "Availability" section.

See also

Properties (Page 576)

"AlarmSourceType" property (RT Uni)

Description

Specifies the source of the alarms of the alarm window:

- NotConfigured (0): Not defined
- ActiveAlarms (1): Pending alarms
- LoggedAalarms (2): Logged alarms
- LoggedAlarmsUpdated (3): Logged alarms with updates.
- AlarmDefintion (4): Custom filter
- AlarmStatistics (5): Alarm statistics

Syntax

```
Object.AlarmSourceType
```

Object

Required. An object from the "Availability" section.

See also

Properties (Page 576)

"DefaultSortDirection" property (RT Uni)

Description

Specifies the sort sequence of the time column if no other sorting is active:

- None (0): None
- Ascending (1): Oldest entries first
- Descending (2): Newest entries first

Syntax

```
Object.DefaultSortDirection
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 576)

"WindowFlags" property (RT Uni)

Description

Specifies the properties of the window:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed

Syntax

```
Object.WindowFlags
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 576)

Methods of "AlarmControl" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmControl" object has the following methods:

Methods	Description
-	

"AlarmView" object (RT Uni)

"AlarmView" description (RT Uni)

Description

Represents the "AlarmView" object.

Type identifier in JavaScript

HMIAlarmView

Properties (RT Uni)

Properties

The "AlarmView" object has the following properties:

Properties	Type	Access	Description
AllowSort	Bool	read/ write	Specifies whether the sorting of columns is allowed.
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateForeColor	UInt32	read/ write	Specifies the flashing color for the text.
BackColor	UInt32	read/ write	Specifies the background color.
CellPadding	Object	read/ write	Specifies the inner distance of the contents from the cell frame.
CellTextTrimming	HmiTextTrimming	read/ write	Specifies the type of trimming of cell contents: ● None (0): None ● Ellipsis (1): Abbreviation at the end of the text
ColoringMode	HmiGridColoring-Mode	read/ write	Specifies whether the alternate coloring of every other row or column is activated. ● None (0): None ● Rows (1): Alternately color the rows ● Columns (2): Alternately color the columns
Columns	Object	read/ write	Specifies the "Columns" object.
ElementID	UInt32	read/ write	Specifies the ID of an element within the active screen.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.

Properties	Type	Access	Description
GridLineColor	UInt32	read/ write	Specifies the color of the grid lines.
GridLineVisibility		read/ write	Specifies the visibility of the grid lines.
GridLineWidth	UInt8	read/ write	Specifies the width of the separator lines in pixels.
GridSelectionMode	HmiGridSelectionMode	read/ write	Specifies whether multiple selection is enabled in the table content. ● None (0): None ● Single (1): Only single selection ● Multi (2): Multiple selection
HeaderSettings	Object	read/ write	Returns the "HeaderSettings" object. Specifies the settings for all column headers of the table.
HorizontalScrollBarVisibility	HmiScrollBarVisibility	read/ write	Specifies the setting for the horizontal scroll bar of the window. ● Automatic (0): Only visible if required ● Visible (1): Visible ● Collapsed (2): Not visible
RowHeight	UInt8	read/ write	Specifies the height of all rows of the table in DIU (Device Independent Unit). "0" corresponds to an automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.
SelectFullRow	Bool	read/ write	Specifies whether only the cell or the whole row is included in a selection.
SelectionBackColor	UInt32	read/ write	Specifies the background color of the selected cells.
SelectionBorderColor	UInt32	read/ write	Specifies the border color of a selection.
SelectionBorderWidth	UInt8	read/ write	Specifies the border thickness of a selection.
SelectionForeColor	UInt32	read/ write	Specifies the foreground color of the selected cells.
Type	UInt32	read/ write	Specifies the type ID.
VerticalScrollBarVisibility	HmiScrollBarVisibility	read/ write	Specifies the setting for the vertical scroll bar of the window: ● Automatic (0): Only visible if required ● Visible (1): Visible ● Collapsed (2): Not visible

Methods of "AlarmView" (RT Uni)

Overview (RT Uni)

Methods

The "AlarmView" object has the following methods:

Methods	Description
-	

"Bar" object (RT Uni)

"Bar" description (RT Uni)

Description

Represents the "Bar" object.

Type identifier in JavaScript

HMIBar

Properties (RT Uni)

Properties

The "Bar" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BarMode (Page 586)	HmiBarMode	read/ write	Specifies the color display of the bar graph.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.

Properties	Type	Access	Description
ComputedMaxPeakValue	Variant	read only	Returns the highest occurred process value.
ComputedMinPeakValue	Variant	read only	Returns the lowest occurred process value.
ComputedValueTendency	HmiValueTendency	read only	Returns the change of the process value. - Steady (0): None - Upwards (1): Change upwards - Downwards (2): Change downwards
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
NormalRangeColor	UInt32	read/ write	Specifies the color of the normal range.
Opacity	Float	read/ write	Specifies the opacity.
OriginValue	Float	read/ write	Specifies the start value that is visualized.
OutputFormat	String	read/ write	Specifies the format for displaying values.
PeakIndicators	HmiPeakIndicator	read/ write	Specifies whether the highest and lowest process value up to this time are displayed. - None (0): Not displayed - Low (1): Only the highest process value - High (2): Only the lowest process value
ProcessValue	Variant	read/ write	Specifies the process value.
ProcessValueIndicator-BackColor	UInt32	read/ write	Specifies the background color for the process value.
ProcessValueIndicator-ForeColor	UInt32	read/ write	Specifies the foreground color for the process value.

Properties	Type	Access	Description
ProcessValueIndicatorMode (Page 587)	HmiProcessIndicatorMode	read/ write	Specifies the type of display of the current process value.
RelativeToOrigin	Bool	read/ write	Specifies whether the start value is an absolute or a percentage value between minimum and maximum value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenterPlacement	HmiRotationCenterPlacement	read/ write	Specifies the reference point around which the specified object rotates. ● 0: Absolute distance from the object center. ● 1: Relative distance from the object center. ● 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ScaleBackColor	UInt32	read/ write	Specifies the background color of the scale.
ScaleForeColor	UInt32	read/ write	Specifies the foreground color of the scale.
ShowTrendIndicator	Bool	read/ write	Specifies whether the trend (rising or falling) of the measured value to be monitored is indicated by means of a small arrow.
StraightScale	Object	read/ write	Specifies the linear scale of the bar display.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
ThresholdIndicators	HmiThresholdIndicator	read/ write	Specifies how parameterized limits are visualized. ● None (0): None ● Lines (1): As line ● Markers (2): As selection
Thresholds	Object	read only	Returns the "Thresholds" object.
Title	Object	read/ write	Returns the "Title" object. Specifies the labeling that is displayed as the header.
ToolTipText	String	read/ write	Specifies the tooltip text.

Properties	Type	Access	Description
Top	Int32	read/ write	Specifies the value of the Y coordinate.
TrendIndicatorColor	UInt32	read/ write	Specifies the color of the trend view. The trend view uses a small arrow to represent the tendency (rising or falling) of the measurement value to be monitored. To activate the trend view, the "ShowTrendIndicator" property must have the value "TRUE".
Unit	Object	read/ write	Specifies the unit of measurement. Returns the "Unit" object.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

See also

"BarMode" property (Page 586)

"ProcessValueIndicatorMode" property (Page 587)

Special properties (RT Uni)

"BarMode" property (RT Uni)

Description

Specifies the color display of the bar graph:

- Segmented (0): Bar graph changes color according to the bar segments.
- Unicolor (1): Entire bar display has one color.
- SegmentedStatic (2): Bar segments in the background. Indicator for process value runs prior to the bar segments.
- UnicolorStatic (3): Background color changes according to the process value and the limit colors. Indicator for process value runs prior to the bar segments.

Type

HmiBarMode

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Bar
- Slider

Syntax

```
Object.BarMode
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 583)

"ProcessValueIndicatorMode" property (RT Uni)

Description

Specifies the type of display of the current process value:

- Bar (0): Bar graph
- Indicator (1): Hairline or needle. No numerical display of the process value.
- DetailedIndicator (2): Detailed display with numerical value
- BarWithDetailedIndicator (3): Bar graph with numerical value

Type

HmiProcessIndicatorMode

Access

Access depends on the object.

Availability

The property is available for the following objects:

- Bar
- Gauge
- Slider

Syntax

```
Object.ProcessValueIndicatorMode
```

Object

Required. An object from the "Availability" section.

See also

Properties (Page 583)

Methods of "Bar" (RT Uni)

Overview (RT Uni)

Methods

The "Bar" object has the following methods:

Methods	Description
-	-

"Button" object (RT Uni)

"Button" description (RT Uni)

Description

Represents the "Button" object.

Type identifier in JavaScript

HmiButton

Properties (RT Uni)

Properties

The "Button" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
AlternateGraphic	String	read/ write	Specifies the graphic for the "pressed" state.
AlternateGraphicHeight	UInt32	read/ write	Specifies the height of the graphic for the "pressed" state.
AlternateGraphicWidth	UInt32	read/ write	Specifies the width of the graphic for the "pressed" state.
AlternateText	String	read/ write	Specifies the text for the "pressed" state.
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
Content	Object	read/ write	Returns the "Content" object.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
Graphic	String	read/ write	Specifies the graphic.
GraphicHeight	UInt32	read/ write	Specifies the height of the graphic.
GraphicWidth	UInt32	read/ write	Specifies the width of the graphic.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.

Properties	Type	Access	Description
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Padding	Object	read/ write	Specifies the distance of the content from the border.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Specifies the position of the object in the tab order.
Text	String	read/ write	Specifies the labeling.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "Button" (RT Uni)

Overview (RT Uni)

Methods

The "Button" object has the following methods:

Methods	Description
-	

"CheckBoxGroup" object (RT Uni)

"CheckBoxGroup" description (RT Uni)

Description

Represents the "Checkbox" object.

Type identifier in JavaScript

HMICheckBoxGroup

Properties (RT Uni)

Properties

The "CheckBoxGroup" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorder-Color	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
Content	Object	read/ write	Specifies the "Content" object.

Properties	Type	Access	Description
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height. The value "0" corresponds to a default setting, not the actual value "0".
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Padding	Object	read/ write	Specifies the distance of the content from the border.
ProcessValue	Variant	read/ write	Specifies the process value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
SelectionItem-Height	UInt16	read/ write	Specifies the height of a selected entry. • 0 = Automatic

Properties	Type	Access	Description
SelectionItems	Object	read only	Returns the entries which can be selected.
SelectorPosition	HmiHorizontalAlignment	read/ write	Specifies the horizontal alignment of the entries. • Left (0): Left • Center (1) • Right (2): Right • Stretch (3)
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "CheckBoxGroup" (RT Uni)

Overview (RT Uni)

Methods

The "CheckBoxGroup" object has the following methods:

Methods	Description
-	

"Circle" object (RT Uni)

"Circle" description (RT Uni)

Description

Represents the "Circle" object.

Type identifier in JavaScript

HMICircle

Properties (RT Uni)

Description

The "Circle" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BackFillPattern	HmiFillPattern	read/ write	Specifies the pattern of the background or the fill.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
CenterX	Int32	read/ write	Specifies the X coordinate of the rotation point.
CenterY	Int32	read/ write	Specifies the Y coordinate of the rotation point.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. ● Solid (0): Solid ● Dash(1): Dashed ● Dot (2): Dotted ● Dash-Dot (3): Dash-dot ● Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
FillDirection	HmiFillDirection	read/ write	Specifies the direction from which the object is filled. ● BottomToTop (0): From bottom to top ● TopToBottom (1): From top to bottom ● LeftToRight (2): From left to right ● RightToLeft (3): From right to left

Properties	Type	Access	Description
FillLevel	UInt8	read/write	Specifies the object fill in percent.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Margin	Object	read/write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/write	Specifies the opacity.
Radius	UInt32	read/write	Specifies the radius.
RotationAngle	Int16	read/write	Specifies the angle of rotation in degrees.
RotationCenterPlacement	HmiRotationCenterPlacement	read/write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/write	Specifies the Y coordinate of the rotation point.
ShowFillLevel	Bool	read/write	Specifies whether the fill level is displayed.
StyleItemClass	String	read/write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/write	Specifies the tooltip text.
Visible	Bool	read/write	Specifies whether the selected object is visible.

Methods of "Circle" (RT Uni)

Overview (RT Uni)

Methods

The "Circle" object has the following methods:

Methods	Description
-	

"CircleSegment" object (RT Uni)

"CircleSegment" description (RT Uni)

Description

Represents the "CircleSegment" object.

Type identifier in JavaScript

HMICircleSegment

Properties (RT Uni)

Properties

The "CircleSegment" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
AngleRange	Int32	read/ write	Specifies the arc angle.
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BackFillPattern	HmiFillPattern	read/ write	Specifies the pattern of the background or the fill.
BorderColor	UInt32	read/ write	Specifies the line color.

Properties	Type	Access	Description
BorderWidth	UInt8	read/ write	Specifies the line thickness.
CenterX	Int32	read/ write	Specifies the X coordinate of the rotation point.
CenterY	Int32	read/ write	Specifies the Y coordinate of the rotation point.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. ● Solid (0): Solid Dash(1): Dashed ● Dot (2): Dotted ● Dash-Dot (3): Dash-dot ● Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
FillDirection	HmiFillDirection	read/ write	Specifies the direction from which the object is filled. ● BottomToTop (0): From bottom to top ● TopToBottom (1): From top to bottom ● LeftToRight (2): From left to right ● RightToLeft (3): From right to left
FillLevel	UInt8	read/ write	Specifies the object fill in percent.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Radius	UInt32	read/ write	Specifies the radius.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.

Properties	Type	Access	Description
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ShowFillLevel	Bool	read/ write	Specifies whether the fill level is displayed.
StartAngle	Int32	read/ write	Specifies the angle by which the start point deviates from the zero position (0°).
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Visible	Bool	read/ write	Specifies whether the selected object is visible.

Methods of "CircleSegment" (RT Uni)

Overview (RT Uni)

Methods

The "CircleSegment" object has the following methods:

Methods	Description
-	

"CircularArc" object (RT Uni)

"CircularArc" description (RT Uni)

Description

Represents the "CircularArc" object.

Type identifier in JavaScript

HMICircularArc

Properties (RT Uni)

Properties

The "CircularArc" object has the following properties:

Properties	Type	Access	Description
AlternateLineColor	UInt32	read/ write	Specifies the second line color which is displayed for line styles such as "Dash".
AngleRange	Int32	read/ write	Specifies the arc angle.
Authorization	Object	read only	Returns the operator authorization.
CapType	HmiCapType	read/ write	Specifies the shape of the line ends. • Round (0): Round (line extends "Line thickness/2" beyond the line end point) • Square (256): Square (line extends "Line thickness/2" beyond the line end point) • Flat (512): Flat (line ends at the line end point)
CenterX	Int32	read/ write	Specifies the X coordinate of the rotation point.
CenterY	Int32	read/ write	Specifies the Y coordinate of the rotation point.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. • Solid (0): Solid Dash(1): Dashed • Dot (2): Dotted • Dash-Dot (3): Dash-dot • Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
EndType	HmiLineEndType	read/ write	Specifies the type of line end. • Line (0): Line • EmptyArrow (1): Arrow • Arrow (2): Arrow (filled) • ReversedArrow (3): Arrow (reversed) • EmptyCircle (5): Circle • Circle (6): Circle (filled)

Properties	Type	Access	Description
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
LineColor	UInt32	read/ write	Specifies the line color.
LineWidth	UInt8	read/ write	Specifies the line thickness.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Radius	UInt32	read/ write	Specifies the radius.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. ● 0: Absolute distance from the object center. ● 1: Relative distance from the object center. ● 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
StartAngle	Int32	read/ write	Specifies the angle by which the start point deviates from the zero position (0°).
StartType	HmiLineEndType	read/ write	Specifies the type of line start. ● Line (0): Line ● EmptyArrow (1): Arrow ● Arrow (2): Arrow (filled) ● ReversedArrow (3): Arrow (reversed) ● EmptyCircle (5): Circle ● Circle (6): Circle (filled)
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.

Properties	Type	Access	Description
ToolTipText	String	read/ write	Specifies the tooltip text.
Visible	Bool	read/ write	Specifies whether the selected object is visible.

Methods of "CircularArc" (RT Uni)

Overview (RT Uni)

Methods

The "CircularArc" object has the following methods:

Methods	Description
-	

"Clock" object (RT Uni)

"Clock" description (RT Uni)

Description

Represents the "Clock" object.

Type identifier in JavaScript

HMIClock

Properties (RT Uni)

Properties

The "Clock" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.

Properties	Type	Access	Description
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
ComputedHours	UInt8	read/ write	Specifies the hours of the current time.
ComputedMinutes	UInt8	read/ write	Specifies the minutes of the current time.
ComputedSeconds	UInt8	read/ write	Specifies the seconds of the current time.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DialBackColor	UInt32	read/ write	Specifies the color of the dial background.
DialLabelColor	UInt32	read/ write	Specifies the color of the text of the dial.
DialLabelFont	Object	read/ write	Specifies the character set for the dial.
DialMode	HmiScaleMode	read/ write	Specifies the details of the dial that are displayed. ● None (0): None ● Labels (1): Labels
DialTickColor	UInt32	read/ write	Specifies the color of the graduation of the dial.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
ProcessValue	Variant	read/ write	Specifies the process value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.

Properties	Type	Access	Description
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ShowHours	Bool	read/ write	Specifies whether the hour hand is displayed.
ShowMinutes	Bool	read/ write	Specifies whether the minute hand is displayed.
ShowSeconds	Bool	read/ write	Specifies whether the second hand is displayed.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
Title	Object	read/ write	Specifies the labeling that is displayed as the header. Returns the "Title" object.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "Clock" (RT Uni)

Overview (RT Uni)

Methods

The "Clock" object has the following methods:

Methods	Description
-	

"Ellipse" object (RT Uni)

"Ellipse" description (RT Uni)

Description

Represents the "Ellipse" object.

Type identifier in JavaScript

HMIEllipse

Properties (RT Uni)

Properties

The "Ellipse" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BackFillPattern	HmiFillPattern	read/ write	Specifies the pattern of the background or the fill.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
CenterX	Int32	read/ write	Specifies the X coordinate of the rotation point.
CenterY	Int32	read/ write	Specifies the Y coordinate of the rotation point.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. • Solid (0): Solid Dash(1): Dashed • Dot (2): Dotted • Dash-Dot (3): Dash-dot • Dash-Dot-Dot (4): Dash-dot-dot

Properties	Type	Access	Description
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
FillDirection	HmiFillDirection	read/ write	Specifies the direction from which the object is filled. ● BottomToTop (0): From bottom to top ● TopToBottom (1): From top to bottom ● LeftToRight (2): From left to right ● RightToLeft (3): From right to left
FillLevel	UInt8	read/ write	Specifies the object fill in percent.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
RadiusX	UInt32	read/ write	Specifies the X radius.
RadiusY	UInt32	read/ write	Specifies the Y radius.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. ● 0: Absolute distance from the object center. ● 1: Relative distance from the object center. ● 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ShowFillLevel	Bool	read/ write	Specifies whether the fill level is displayed.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.

Properties	Type	Access	Description
ToolTipText	String	read/ write	Specifies the tooltip text.
Visible	Bool	read/ write	Specifies whether the selected object is visible.

Methods of "Ellipse" (RT Uni)

Overview (RT Uni)

Methods

The "Ellipse" object has the following methods:

Methods	Description
-	

"EllipseSegment" object (RT Uni)

"EllipseSegment" description (RT Uni)

Description

Represents the "EllipseSegment" object.

Type identifier in JavaScript

HMIEllipseSegment

Properties (RT Uni)

Properties

The "EllipseSegment" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorder-Color	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
AngleRange	Int32	read/ write	Specifies the arc angle.
Authorization	Object	read only	Returns the operator authorization.

Properties	Type	Access	Description
BackColor	UInt32	read/ write	Specifies the background color.
BackFillPattern	HmiFillPattern	read/ write	Specifies the pattern of the background or the fill.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
CenterX	Int32	read/ write	Specifies the X coordinate of the rotation point.
CenterY	Int32	read/ write	Specifies the Y coordinate of the rotation point.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. ● Solid (0): Solid Dash(1): Dashed ● Dot (2): Dotted ● Dash-Dot (3): Dash-dot ● Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
FillDirection	HmiFillDirection	read/ write	Specifies the direction from which the object is filled. ● BottomToTop (0): From bottom to top ● TopToBottom (1): From top to bottom ● LeftToRight (2): From left to right ● RightToLeft (3): From right to left
FillLevel	UInt8	read/ write	Specifies the object fill in percent.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
RadiusX	UInt32	read/ write	Specifies the X radius.
RadiusY	UInt32	read/ write	Specifies the Y radius.

Properties	Type	Access	Description
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ShowFillLevel	Bool	read/ write	Specifies whether the fill level is displayed.
StartAngle	Int32	read/ write	Specifies the angle by which the start point deviates from the zero position (0°).
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Visible	Bool	read/ write	Specifies whether the selected object is visible.

Methods of "EllipseSegment" (RT Uni)

Overview (RT Uni)

Methods

The "EllipseSegment" object has the following methods:

Methods	Description
-	

"EllipticalArc" object (RT Uni)

"EllipticalArc" description (RT Uni)

Description

Represents the "EllipticalArc" object.

Type identifier in JavaScript

HMIEllipticalArc

Properties (RT Uni)

Properties

The "Elliptical Arc" object has the following properties:

Properties	Type	Access	Description
AlternateLineColor	UInt32	read/ write	Specifies the second line color which is displayed for line styles such as "Dash".
AngleRange	Int32	read/ write	Specifies the arc angle.
Authorization	Object	read only	Returns the operator authorization.
CapType	HmiCapType	read/ write	Specifies the shape of the line ends. • Round (0): Round (line extends "Line thickness/2" beyond the line end point) • Square (256): Square (line extends "Line thickness/2" beyond the line end point) • Flat (512): Flat (line ends at the line end point)
CenterX	Int32	read/ write	Specifies the X coordinate of the rotation point.
CenterY	Int32	read/ write	Specifies the Y coordinate of the rotation point.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. • Solid (0): Solid Dash(1): Dashed • Dot (2): Dotted • Dash-Dot (3): Dash-dot • Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.

Properties	Type	Access	Description
EndType	HmiLineEndType	read/ write	Specifies the type of line end. • Line (0): Line • EmptyArrow (1): Arrow • Arrow (2): Arrow (filled) • ReversedArrow (3): Arrow (reversed) • EmptyCircle (5): Circle • Circle (6): Circle (filled)
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
LineColor	UInt32	read/ write	Specifies the line color.
LineWidth	UInt8	read/ write	Specifies the line thickness.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
RadiusX	UInt32	read/ write	Specifies the X radius.
RadiusY	UInt32	read/ write	Specifies the Y radius.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
StartAngle	Int32	read/ write	Specifies the angle by which the start point deviates from the zero position (0°).

Properties	Type	Access	Description
StartType	HmiLineEndType	read/ write	Specifies the type of line start. • Line (0): Line • EmptyArrow (1): Arrow • Arrow (2): Arrow (filled) • ReversedArrow (3): Arrow (reversed) • EmptyCircle (5): Circle • Circle (6): Circle (filled)
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Visible	Bool	read/ write	Specifies whether the selected object is visible.

Methods of "EllipticalArc" (RT Uni)

Overview (RT Uni)

Methods

The "EllipticalArc" object has the following methods:

Methods	Description
-	

"FunctionTrendControl" object (RT Uni)

"FunctionTrendControl" description (RT Uni)

Description

Represents the "FunctionTrendControl" object.

Type identifier in JavaScript

HmiFunctionTrendControl

Properties (RT Uni)

Properties

The "FunctionTrendControl" object has the following properties:

Properties	Type	Access	Description
AlwaysShowRecent	Bool	read/write	Specifies whether the newest alarm is displayed at the beginning or end of the list based on the sorting.
AreaSpacing	UInt16	read/write	Specifies the distance between trend windows.
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/write	Specifies the background color.
Caption	String	read/write	Specifies the text to be displayed in the header.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/write	Specifies whether the specified object can be operated in runtime.
ExtendRulerToAxis	Bool	read/write	Specifies whether the ruler is extended.
Font	Object	read/write	Specifies the font of the text.
FunctionTrendAreas	Object	read only	Returns the "FunctionTrendControls" object.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/write	Specifies the height.
Icon	String	read/write	Specifies the icon.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/write	Specifies the value of the X coordinate.
Legend	Object	read/write	Returns the "Legend" object.
Margin	Object	read/write	Specifies the margin.
Name	String	read only	Returns the object name.
Online	Bool	read/write	Specifies the start and stop of the updating.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.

Properties	Type	Access	Description
ShiftAxis	Bool	read/ write	Specifies whether the X axis and Y axis of the control are exchanged.
ShowRuler	Bool	read/ write	Specifies whether a scale division (help line) is displayed for an axis label of the object.
StatusBar	Object	read/ write	Returns the "StatusBar" object.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Specifies the position of the object in the tab order.
ToolBar	Object	read/ write	Returns the "ToolBar" object. Specifies the toolbar of the window.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.
WindowFlags	HmiWindowFlag	read/ write	Specifies the properties of the window. ● None (0): Use default setting of the object ● ShowCaption (1): Show title ● ShowBorder (2): Show border ● AlwaysOnTop (4): Always on top ● CanSize (8): Can be sized ● CanMove (16): Can be positioned ● CanMaximize (32): Can be maximized ● CanClose (64): Can be closed

Methods of "FunctionTrendControl" (RT Uni)

Overview (RT Uni)

Methods

The "FunctionTrendControl" object has the following methods:

Methods	Description
-	

"Gauge" object (RT Uni)

"Gauge" description (RT Uni)

Description

Represents the "Gauge" object.

Type identifier in JavaScript

HMIGauge

Properties (RT Uni)

Properties

The "Gauge" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
ComputedMaxPeakValue	Variant	read only	Returns the highest occurred process value.
ComputedMinPeakValue	Variant	read only	Returns the lowest occurred process value.
ComputedValueTendency	HmiValueTendency	read only	Returns the change of the process value. • Steady (0): None • Upwards (1): Change upwards • Downwards (2): Change downwards
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
CurvedScale	Object	read/ write	Specifies the curved scale of the display.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.

Properties	Type	Access	Description
Font	Object	read/ write	Specifies the font of the text.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
NormalRangeColor	UInt32	read/ write	Specifies the color of the normal range.
Opacity	Float	read/ write	Specifies the opacity.
OriginValue	Float	read/ write	Specifies the start value that is visualized.
OutputFormat	String	read/ write	Specifies the format for displaying values.
PeakIndicators	HmiPeakIndicator	read/ write	Specifies whether the highest and lowest process value up to this time are displayed. ● None (0): Not displayed ● Low (1): Only the highest process value ● High (2): Only the lowest process value
ProcessValue	Variant	read/ write	Specifies the process value.
ProcessValueIndicator-BackColor	UInt32	read/ write	Specifies the background color for the process value.
ProcessValueIndicator-ForeColor	UInt32	read/ write	Specifies the foreground color for the process value.
ProcessValueIndicator-Mode	HmiProcessIndicator-Mode	read/ write	Specifies the type of display of the current process value. ● Bar (0): Bar graph ● Indicator (1): Hairline or needle. No numerical display of the process value. ● DetailedIndicator (2): Detailed display with numerical value ● BarWithDetailedIndicator (3): Bar graph with numerical value

Properties	Type	Access	Description
RelativeToOrigin	Bool	read/ write	Specifies whether the start value is an absolute or a percentage value between minimum and maximum value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenterPlacement	HmiRotationCenterPlacement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ScaleBackColor	UInt32	read/ write	Specifies the background color of the scale.
ScaleForeColor	UInt32	read/ write	Specifies the foreground color of the scale.
ShowTrendIndicator	Bool	read/ write	Specifies whether the trend (rising or falling) of the measured value to be monitored is indicated by means of a small arrow.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
ThresholdIndicators	HmiThresholdIndicator	read/ write	Specifies how parameterized limits are visualized. • None (0): None • Lines (1): As line • Markers (2): As selection
Thresholds	Object	read only	Returns the "Thresholds" object.
Title	Object	read/ write	Specifies the labeling that is displayed as the header.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.

Properties	Type	Access	Description
TrendIndicatorColor	UInt32	read/ write	Specifies the color of the trend view. The trend view uses a small arrow to represent the tendency (rising or falling) of the measurement value to be monitored. To activate the trend view, the "ShowTrendIndicator" property must have the value "TRUE".
Unit	Object	read/ write	Specifies the unit of measurement. Returns the "Unit" object.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "Gauge" (RT Uni)

Overview (RT Uni)

Methods

The "Gauge" object has the following methods:

Methods	Description
-	-

"GraphicView" object (RT Uni)

"GraphicView" description (RT Uni)

Description

Represents the "GraphicView" object.

Type identifier in JavaScript

HMIGraphicView

Properties (RT Uni)

Properties

The "GraphicView" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BackFillPattern	HmiFillPattern	read/ write	Specifies the pattern of the background or the fill.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
FillDirection	HmiFillDirection	read/ write	Specifies the direction from which the object is filled. ● BottomToTop (0): From bottom to top ● TopToBottom (1): From top to bottom ● LeftToRight (2): From left to right ● RightToLeft (3): From right to left
FillLevel	UInt8	read/ write	Specifies the object fill in percent.
Graphic	String	read/ write	Specifies the graphic.
GraphicHeight	UInt32	read/ write	Specifies the height of the graphic.
GraphicStretchMode	HmiGraphicStretchMode	read/ write	Specifies the type of scaling of the graphic in the screen. ● None (0): The graphic is shown in the original size and centered. ● Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled. ● Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed. ● UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed. ● Tiled (4): The graphic is shown in the original size and repeated in tiles.
GraphicWidth	UInt32	read/ write	Specifies the width of the graphic.

Properties	Type	Access	Description
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Padding	Object	read/ write	Specifies the distance of the content from the border.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ShowFillLevel	Bool	read/ write	Specifies whether the fill level is displayed.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.

Methods of "GraphicView" (RT Uni)

Overview (RT Uni)

Methods

The "GraphicView" object has the following methods:

Methods	Description
-	

"IOField" object (RT Uni)

"IOField" description (RT Uni)

Description

Represents the "I/O field" object.

Type identifier in JavaScript

HmiIOField

Properties (RT Uni)

Properties

The "IOField" object has the following properties:

Property	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.

Property	Type	Access	Description
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
HorizontalTextAlignment	HmiHorizontalAlignment	read/ write	Specifies the horizontal alignment of a text.
InputBehavior	Object	read/ write	Returns the "InputBehavior" object.
IOFieldType	HmiIOFieldType	read/ write	Specifies the input/output type. • Output (0): Output • InputOutput (2): Input and output
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
MeasurementUnit	String	read only	Returns the displayed unit.
MeasurementUnitType	HmiMeasurementUnit	read/ write	Specifies the display format of the unit. • None (0): None • Name (1): Name, e.g. "kilogram" • Symbol (2): SI unit, e.g. "kg"
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
OutputFormat	String	read/ write	Specifies the format for displaying values.
Padding	Object	read/ write	Specifies the distance of the content from the border.
ProcessValue	Variant	read/ write	Specifies the process value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.

Property	Type	Access	Description
RotationCenterPlacement	HmiRotationCenterPlacement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Specifies the position of the object in the tab order.
TextTrimming	HmiTextTrimming	read/ write	Specifies the type of trimming of a text if the space is not sufficient. • None (0): None • Ellipsis (1): Abbreviation at the end of the text
Thresholds	Object	read only	Returns the "Thresholds" object.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
VerticalTextAlignment	HmiVerticalAlignment	read/ write	Specifies the vertical alignment of a text. • Top (0) • Center (1) • Bottom (2) • Stretch (3)
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "IOField" (RT Uni)

Overview (RT Uni)

Methods

The "IOField" object has the following methods:

Methods	Description
-	

"Label" object (RT Uni)

"Label" description (RT Uni)

Description

Represents the "Editable text box" object.

Type identifier in JavaScript

HMILabel

Properties (RT Uni)

Properties

The "Label" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.

Properties	Type	Access	Description
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
HorizontalTextAlignment	HmiHorizontalAlignment	read/ write	Specifies the horizontal alignment of a text.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Padding	Object	read/ write	Specifies the distance of the content from the border.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenterPlacement	HmiRotationCenterPlacement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Returns the position of the object in the tab order.
Text	String	read/ write	Specifies the labeling.

Properties	Type	Access	Description
TextTrimming	HmiTextTrimming	read/ write	Specifies the type of trimming of a text if the space is not sufficient. • None (0): None • Ellipsis (1): Abbreviation at the end of the text
TextWrapping	HmiTextWrapping	read/ write	Specifies how text is wrapped if there is insufficient space. • NoWrap (0): No wrap • WordWrap (1): Wrap after the last fully displayed value.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
VerticalTextAlignment	HmiVerticalAlignment	read/ write	Specifies the vertical alignment of a text. • Top (0) • Center (1) • Bottom (2) • Stretch (3)
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "Label" (RT Uni)

Overview (RT Uni)

Methods

The "Label" object has the following methods:

Methods	Description
-	

"Line" object (RT Uni)

"Line" description (RT Uni)

Description

Represents the "Line" object.

Type identifier in JavaScript

HMILine

Properties (RT Uni)

Properties

The "Line" object has the following properties:

Properties	Type	Access	Description
AlternateLineColor	UInt32	read/ write	Specifies the second line color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
CapType	HmiCapType	read/ write	Specifies the shape of the line ends. • Round (0): Round (line extends "Line thickness/2" beyond the line end point) • Square (256): Square (line extends "Line thickness/2" beyond the line end point) • Flat (512): Flat (line ends at the line end point)
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. • Solid (0): Solid Dash(1): Dashed • Dot (2): Dotted • Dash-Dot (3): Dash-dot • Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.

Properties	Type	Access	Description
EndType	HmiLineEndType	read/ write	Specifies the type of line end. • Line (0): Line • EmptyArrow (1): Arrow • Arrow (2): Arrow (filled) • ReversedArrow (3): Arrow (reversed) • EmptyCircle (5): Circle • Circle (6): Circle (filled)
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
LineColor	UInt32	read/ write	Specifies the line color.
LineWidth	UInt8	read/ write	Specifies the line thickness.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.

Properties	Type	Access	Description
StartType	HmiLineEndType	read/ write	Specifies the type of line start. • Line (0): Line • EmptyArrow (1): Arrow • Arrow (2): Arrow (filled) • ReversedArrow (3): Arrow (reversed) • EmptyCircle (5): Circle • Circle (6): Circle (filled)
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.
X1	Int32	read/ write	Specifies the x coordinate of the start point of the line.
X2	Int32	read/ write	Specifies the x coordinate of the end point of the line.
Y1	Int32	read/ write	Specifies the y coordinate of the start point of the line.
Y2	Int32	read/ write	Specifies the y coordinate of the end point of the line.

Methods of "Line" (RT Uni)

Overview (RT Uni)

Methods

The "Line" object has the following methods:

Methods	Description
-	

"ListBox" object (RT Uni)

"ListBox" description (RT Uni)

Description

Represents the "Listbox" object.

Type identifier in JavaScript

HMIListBox

Properties (RT Uni)

Properties

The "ListBox" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
Content	Object	read/ write	Specifies the "Content" object.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.

Properties	Type	Access	Description
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Padding	Object	read/ write	Specifies the distance of the content from the border.
ProcessValue	Variant	read/ write	Specifies the process value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
SelectionItemHeight	UInt16	read/ write	Specifies the height of a selected entry. • 0 = Automatic
SelectionItems	Object	read only	Returns the entries which can be selected.
SelectionMode	HmiSelectionMode	read/ write	Specifies whether one or more entries can be selected in the selection list. • NonExclusive (0): Multiple selection possible • Exclusive (1): Multiple selection not possible
SelectorPosition	HmiHorizontalAlignment	read/ write	Specifies the horizontal alignment of the entries. • Left (0): Left • Center (1) • Right (2): Right • Stretch (3)

Properties	Type	Access	Description
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "ListBox" (RT Uni)

Overview (RT Uni)

Methods

The "ListBox" object has the following methods:

Methods	Description
-	

"Polygon" object (RT Uni)

"Polygon" description (RT Uni)

Description

Represents the "Polygon" object.

Type identifier in JavaScript

HMIPolygon

Properties (RT Uni)

Properties

The "Polygon" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BackFillPattern	HmiFillPattern	read/ write	Specifies the pattern of the background or the fill.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. ● Solid (0): Solid Dash(1): Dashed ● Dot (2): Dotted ● Dash-Dot (3): Dash-dot ● Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
FillDirection	HmiFillDirection	read/ write	Specifies the direction from which the object is filled. ● BottomToTop (0): From bottom to top ● TopToBottom (1): From top to bottom ● LeftToRight (2): From left to right ● RightToLeft (3): From right to left
FillLevel	UInt8	read/ write	Specifies the object fill in percent.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.

Properties	Type	Access	Description
JoinType	HmiLineJoinType	read/ write	Specifies the corner style of the polyline. • Round (0): Round • Bevel (1): Flat • Miter (2): Miter
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Points	Variant	read/ write	Specifies the coordinates of the points of the polyline.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ShowFillLevel	Bool	read/ write	Specifies whether the fill level is displayed.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Returns the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.

Methods of "Polygon" (RT Uni)

Overview (RT Uni)

Methods

The "Polygon" object has the following methods:

Methods	Description
-	

"Polyline" object (RT Uni)

"Polyline" description (RT Uni)

Description

Represents the "Polyline" object.

Type identifier in JavaScript

HMIPolyline

Properties (RT Uni)

Properties

The "Polyline" object has the following properties:

Properties	Type	Access	Description
AlternateLineColor	UInt32	read/ write	Specifies the second line color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
CapType	HmiCapType	read/ write	Specifies the shape of the line ends. ● Round (0): Round (line extends "Line thickness/2" beyond the line end point) ● Square (256): Square (line extends "Line thickness/2" beyond the line end point) ● Flat (512): Flat (line ends at the line end point)
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.

Properties	Type	Access	Description
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. • Solid (0): Solid Dash(1): Dashed • Dot (2): Dotted • Dash-Dot (3): Dash-dot • Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
EndType	HmiLineEndType	read/ write	Specifies the type of line end. • Line (0): Line • EmptyArrow (1): Arrow • Arrow (2): Arrow (filled) • ReversedArrow (3): Arrow (reversed) • EmptyCircle (5): Circle • Circle (6): Circle (filled)
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
JoinType	HmiLineJoinType	read/ write	Specifies the corner style of the polyline. • Round (0): Round • Bevel (1): Flat • Miter (2): Miter
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
LineColor	UInt32	read/ write	Specifies the line color.
LineWidth	UInt8	read/ write	Specifies the line thickness.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Points	Variant	read/ write	Specifies the coordinates of the points of the polyline.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.

Properties	Type	Access	Description
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. - 0: Absolute distance from the object center. - 1: Relative distance from the object center. - 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
StartType	HmiLineEndType	read/ write	Specifies the type of line start. - Line (0): Line - EmptyArrow (1): Arrow - Arrow (2): Arrow (filled) - ReversedArrow (3): Arrow (reversed) - EmptyCircle (5): Circle - Circle (6): Circle (filled)
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Specifies the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.

Methods of "Polyline" (RT Uni)

Overview (RT Uni)

Methods

The "Polyline" object has the following methods:

Methods	Description
-	

"PopupScreenWindow" object (RT Uni)

"PopupScreenWindow" description (RT Uni)

Description

Represents the "Popup screen window" object.

Type identifier in JavaScript

HMIPopupScreenWindow

Properties (RT Uni)

Properties

The "PopupScreenWindow" object has the following properties:

Properties	Type	Access	Description
Adaption	HmiScreenWindowAdaption	read/ write	Specifies how the window size is specified. • None (0): No adaptation. • WindowToScreen (1): Window size corresponds to screen size. • ScreenToWindow(2): Screen is scaled.
Authorization	Object	read only	Returns the operator authorization.
Caption	String	read/ write	Specifies the text to be displayed in the header. Type
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
CurrentZoomFactor	Float	read/ write	Specifies the zoom factor which is applied to the displayed screen. The value 1.0 corresponds to a zoom factor of 100%.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
HardKeysEnabled	Bool	read/ write	Specifies whether the operation is possible via keyboard or function keys.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
HorizontalScrollBarPosition	Int32	read/ write	Specifies the horizontal alignment for the scroll bar.

Properties	Type	Access	Description
HorizontalScrollBarVisibility	HmiScrollBarVisibility	read/ write	Specifies the setting for the horizontal scroll bar of the window. • Automatic (0): Only visible if required • Visible (1): Visible • Collapsed (2): Not visible
Icon	String	read/ write	Specifies the icon.
InteractiveZooming	Bool	read/ write	Specifies whether zooming is supported.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Monitor	UInt8	read only	Specifies the monitor on which the window is displayed.
Name	String	read only	Returns the name of the object or specifies it.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
Screen	String	read/ write	Specifies the name of the screen ("HMIScreen" type) that is contained in the referenced screen window. Loads a new screen into the referenced screen window via its name. The "Screen" property returns a different value than the "CurrentScreen" property when the referenced screen is not yet loaded completely or does not exist.
ScreenName	String	read only	Returns the screen name.
ScreenNumber	UInt16	read only	Returns the screen number.
StartupPosition	HmiWindowStartupPosition	read/ write	Specifies the position of the screen window in case of a runtime start. • None (0): Relative placement on the configured monitor via "Left" and "Top". • CenteredMonitor (1): Centered on the configured monitor. • Maximized (2): Maximized on the configured monitor. • CenteredOwner (3): Centered on the displayed screen.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.

Properties	Type	Access	Description
System	String	read/ write	Specifies the server prefix.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
TabIntoWindow	Bool	read/ write	Specifies at activation via the tab order that the configured tab order of the displayed screen is resumed.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
VerticalScrollBarPosition	Int32	read/ write	Specifies the vertical alignment for the scroll bar.
VerticalScrollBarVisibility	HmiScrollBarVisibility	read/ write	Specifies the setting for the vertical scroll bar of the window. • Automatic (0): Only visible if required • Visible (1): Visible • Collapsed (2): Not visible
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.
WindowFlags	HmiWindowFlag	read/ write	Specifies the properties of the window. • None (0): Use default setting of the object • ShowCaption (1): Show title • ShowBorder (2): Show border • AlwaysOnTop (4): Always on top • CanSize (8): Can be sized • CanMove (16): Can be positioned • CanMaximize (32): Can be maximized • CanClose (64): Can be closed

Methods of "PopupScreenWindow" (RT Uni)

Overview (RT Uni)

Methods

The "PopupScreenWindow" object has the following methods:

Methods	Description
-	-

"RadioButtonGroup" object (RT Uni)

"RadioButtonGroup" description (RT Uni)

Description

Represents the "Option buttons" object.

Type identifier in JavaScript

HMIRadioButtonGroup

Properties (RT Uni)

Properties

The "RadioButtonGroup" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorder-Color	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
Content	Object	read/ write	Specifies the "Content" object.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height. The value "0" corresponds to a default setting, not the actual value "0".

Properties	Type	Access	Description
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Padding	Object	read/ write	Specifies the distance of the content from the border.
ProcessValue	Variant	read/ write	Specifies the process value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. ● 0: Absolute distance from the object center. ● 1: Relative distance from the object center. ● 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
SelectionItem-Height	UInt16	read/ write	Specifies the height of a selected entry. ● 0 = Automatic
SelectionItems	Object	read only	Returns the entries which can be selected.
SelectorPosition	HmiHorizontalAlign-ment	read/ write	Specifies the horizontal alignment of the entries. ● Left (0): Left ● Center (1) ● Right (2): Right ● Stretch (3)
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Returns the position of the object in the tab order.

Properties	Type	Access	Description
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "RadioButtonGroup" (RT Uni)

Overview (RT Uni)

Methods

The "RadioButtonGroup" object has the following methods:

Methods	Description
-	

"Rectangle" object (RT Uni)

"Rectangle" description (RT Uni)

Description

Represents the "Rectangle" object.

Type identifier in JavaScript

HMIRectangle

Properties (RT Uni)

Properties

The "Rectangle" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BackFillPattern	HmiFillPattern	read/ write	Specifies the pattern of the background or the fill.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
Corners	Object	read/ write	Specifies the rounding of corners of the rectangle.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
DashType	HmiDashType	read/ write	Specifies the dash type of the border or the line. ● Solid (0): Solid Dash(1): Dashed ● Dot (2): Dotted ● Dash-Dot (3): Dash-dot ● Dash-Dot-Dot (4): Dash-dot-dot
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
FillDirection	HmiFillDirection	read/ write	Specifies the direction from which the object is filled. ● BottomToTop (0): From bottom to top ● TopToBottom (1): From top to bottom ● LeftToRight (2): From left to right ● RightToLeft (3): From right to left
FillLevel	UInt8	read/ write	Specifies the object fill in percent.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.

Properties	Type	Access	Description
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. ● 0: Absolute distance from the object center. ● 1: Relative distance from the object center. ● 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ShowFillLevel	Bool	read/ write	Specifies whether the fill level is displayed.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Specifies the position of the object in the tab order.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.

Methods of "Rectangle" (RT Uni)

Overview (RT Uni)

Methods

The "Rectangle" object has the following methods:

Methods	Description
-	

"Slider" object (RT Uni)

"Slider" description (RT Uni)

Description

Represents the "Slider" object.

Type identifier in JavaScript

HMISlider

Properties (RT Uni)

Properties

The "Slider" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.

Properties	Type	Access	Description
BarMode	HmiBarMode	read/ write	Specifies the color display of the bar graph. • Segmented (0): Bar graph changes color according to the bar segments. • Unicolor (1): Entire bar display has one color. • SegmentedStatic (2): Bar segments in the background. Indicator for process value runs in front of the bar segments. • UnicolorStatic (3): Background color changes according to the process value and the limit colors. Indicator for process value runs in front of the bar segments.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
ComputedMaxPeakValue	Variant	read only	Returns the highest occurred process value.
ComputedMinPeakValue	Variant	read only	Returns the lowest occurred process value.
ComputedValueTendency	HmiValueTendency	read only	Returns the change of the process value. • Steady (0): None • Upwards (1): Change upwards • Downwards (2): Change downwards
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.

Properties	Type	Access	Description
NormalRangeColor	UInt32	read/ write	Specifies the color of the normal range.
Opacity	Float	read/ write	Specifies the opacity.
OriginValue	Float	read/ write	Specifies the start value that is visualized.
OutputFormat	String	read/ write	Specifies the format for displaying values.
PeakIndicators	HmiPeakIndicator	read/ write	Specifies whether the highest and lowest process value up to this time are displayed. • None (0): Not displayed • Low (1): Only the highest process value • High (2): Only the lowest process value
ProcessValue	Variant	read/ write	Specifies the process value.
ProcessValueIndicator-BackColor	UInt32	read/ write	Specifies the background color for the process value.
ProcessValueIndicator-ForeColor	UInt32	read/ write	Specifies the foreground color for the process value.
ProcessValueIndicator-Mode	HmiProcessIndicatorMode	read/ write	Specifies the type of display of the current process value. • Bar (0): Bar graph • Indicator (1): Hairline or needle. No numerical display of the process value. • DetailedIndicator (2): Detailed display with numerical value • BarWithDetailedIndicator (3): Bar graph with numerical value
RelativeToOrigin	Bool	read/ write	Specifies whether the start value is an absolute or a percentage value between minimum and maximum value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenterPlacement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. • 0: Absolute distance from the object center. • 1: Relative distance from the object center. • 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.

Properties	Type	Access	Description
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
ScaleBackColor	UInt32	read/ write	Specifies the background color of the scale.
ScaleForeColor	UInt32	read/ write	Specifies the foreground color of the scale.
ShowTrendIndicator	Bool	read/ write	Specifies whether the trend (rising or falling) of the measured value to be monitored is indicated by means of a small arrow.
ShowValue	Bool	read/ write	Specifies that the position is output additionally as a value.
StraightScale	Object	read/ write	Specifies the linear scale of the bar display.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Returns the position of the object in the tab order.
ThresholdIndicators	HmiThresholdIndicator	read/ write	Specifies how parameterized limits are visualized. • None (0): None • Lines (1): As line • Markers (2): As selection
Thresholds	Object	read only	Returns the "Thresholds" object.
ThumbBackColor	UInt32	read/ write	Specifies the background color of the slider.
ThumbForeColor	UInt32	read/ write	Specifies the foreground color of the slider.
Title	Object	read/ write	Specifies the "Title" object.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
TrendIndicatorColor	UInt32	read/ write	Specifies the color of the trend view. The trend view uses a small arrow to represent the tendency (rising or falling) of the measurement value to be monitored. To activate the trend view, the "ShowTrendIndicator" property must have the value "TRUE".
Unit	Object	read/ write	Specifies the "Unit" object.

Properties	Type	Access	Description
ValuePosition	HmiSimplePosition	read/ write	Specifies where the value of the current slider position is additionally displayed numerically. ● LeftOrTop (0): Left for vertical alignment, top for horizontal alignment ● RightOrBottom (1): Right for vertical alignment, bottom for horizontal alignment
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.
WriteDuringChange	Bool	read/ write	Specifies when changes are transferred to the PLC. ● 0: Only after the slider has been released. ● 1: At every change of the slider.

Methods of "Slider" (RT Uni)

Overview (RT Uni)

Methods

The "Slider" object has the following methods:

Methods	Description
-	-

"SymbolicIOField" object (RT Uni)

"SymbolicIOField" description (RT Uni)

Description

Represents the "SymbolicIOField" object.

Type identifier in JavaScript

HMISymbolicIOField

Properties (RT Uni)

Properties

The "SymbolicIOField" object has the following properties:

Properties	Type	Access	Description
AcceptExplicitely	Bool	read/ write	Specifies whether the process value is only written by explicit triggering of double-click, enter key or tab.
AcceptOnDeactivated	Bool	read/ write	Specifies whether the process value is written when the object loses the input focus.
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
AutoTabOnAccept	Bool	read/ write	Specifies whether a switch to the next object in the configured tab order occurs automatically after a value is entered.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
Content	Object	read/ write	Returns the "Content" object.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
ExpandOnActivate	Bool	read/ write	Specifies whether the list drops down when it receives the input focus.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
Graphic	String	read/ write	Specifies the graphic.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
IOFieldType	HmiIOFieldType	read/ write	Specifies the input/output type. ● Output (0): Output ● InputOutput (2): Input and output

Properties	Type	Access	Description
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Padding	Object	read/ write	Specifies the distance of the content from the border.
ProcessValue	Variant	read/ write	Specifies the process value.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. ● 0: Absolute distance from the object center. ● 1: Relative distance from the object center. ● 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
SelectionBackColor	UInt32	read/ write	Specifies the background color of the selected cells.
SelectionForeColor	UInt32	read/ write	Specifies the foreground color of the selected cells.
ShowDropDown-Button	Bool	read/ write	Specifies whether a button for the selection list is displayed.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	Object	read only	Specifies the position of the object in the tab order.
Text	String	read/ write	Specifies the labeling.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.

Properties	Type	Access	Description
Visible	Bool	read/ write	Specifies whether the selected object is visible.
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "SymbolicIOField" (RT Uni)

Overview (RT Uni)

Methods

The "SymbolicIOField" object has the following methods:

Methods	Description
-	

"Text" object (RT Uni)

"Text" description (RT Uni)

Description

Represents the "Text box" object.

Type identifier in JavaScript

HMIText

Properties (RT Uni)

Properties

The "Text" object has the following properties:

Properties	Text	Access	Description
Authorization	Object	read only	Returns the operator authorization.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.

Properties	Text	Access	Description
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
HorizontalTextAlignment	HmiHorizontalAlignment	read/ write	Specifies the horizontal alignment of a text.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenterPlacement	HmiRotationCenterPlacement	read/ write	Specifies the reference point around which the specified object rotates. ● 0: Absolute distance from the object center. ● 1: Relative distance from the object center. ● 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
Text	String	read/ write	Specifies the labeling.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.

Properties	Text	Access	Description
VerticalTextAlignment	HmiVerticalAlignment	read/ write	Specifies the vertical alignment of a text. • Top (0) • Center (1) • Bottom (2) • Stretch (3)
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.

Methods of "Text" (RT Uni)

Overview (RT Uni)

Methods

The "Text" object has the following methods:

Methods	Description
-	

"TextBox" object (RT Uni)

"TextBox" description (RT Uni)

Description

Represents the "Editable text box" object.

Type identifier in JavaScript

HMITextBox

Properties (RT Uni)

Properties

The "TextBox" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
AlternateBorderColor	UInt32	read/ write	Specifies the second border color which is displayed for line styles such as "Dash".
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BorderColor	UInt32	read/ write	Specifies the line color.
BorderWidth	UInt8	read/ write	Specifies the line thickness.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
Font	Object	read/ write	Specifies the font of the text.
ForeColor	UInt32	read/ write	Specifies the font color.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
HorizontalTextAlignment	HmiHorizontalAlignment	read/ write	Specifies the horizontal alignment of a text.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Opacity	Float	read/ write	Specifies the opacity.
Padding	Object	read/ write	Specifies the distance of the content from the border.
ReadOnly	Bool	read/ write	Specifies whether the text box is write-protected.

Properties	Type	Access	Description
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
RotationAngle	Int16	read/ write	Specifies the angle of rotation in degrees.
RotationCenter-Placement	HmiRotationCenter-Placement	read/ write	Specifies the reference point around which the specified object rotates. ● 0: Absolute distance from the object center. ● 1: Relative distance from the object center. ● 2: Absolute distance from the screen origin.
RotationCenterX	Float	read/ write	Specifies the X coordinate of the rotation point.
RotationCenterY	Float	read/ write	Specifies the Y coordinate of the rotation point.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
Text	String	read/ write	Specifies the labeling.
TextTrimming	HmiTextTrimming	read/ write	Specifies the type of trimming of a text if the space is not sufficient. ● None (0): None ● Ellipsis (1): Abbreviation at the end of the text
TextWrapping	HmiTextWrapping	read/ write	Specifies how text is wrapped if there is insufficient space. ● NoWrap (0): No wrap ● WordWrap (1): Wrap after the last fully displayed value.
ToolTipText	String	read/ write	Specifies the tooltip text.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
VerticalTextAlign-ment	HmiVerticalAlignment	read/ write	Specifies the vertical alignment of a text. ● Top (0) ● Center (1) ● Bottom (2) ● Stretch (3)
Visible	Bool	read/ write	Specifies whether the selected object is visible.

Properties	Type	Access	Description
VisualizeQuality	Bool	read/ write	Specifies whether the quality of the process value is displayed.
Width	UInt32	read/ write	Specifies the width.

Methods of "TextBox" (RT Uni)

Overview (RT Uni)

Methods

The "TextBox" object has the following methods:

Methods	Description
-	

"TouchArea" object (RT Uni)

"TouchArea" description (RT Uni)

Description

Displays the "TouchArea" object.

Type identifier in JavaScript

HMITouchArea

Properties (RT Uni)

Properties

The "TouchArea" object has the following properties:

Properties	Type	Access	Description
-	-	-	-

Methods of "TouchArea" (RT Uni)

Overview (RT Uni)

Methods

The "TouchArea" object has the following methods:

Methods	Description
-	-

"TrendCompanion" object (RT Uni)

"TrendCompanion" description (RT Uni)

Description

Represents the "TrendCompanion" object.

Type identifier in JavaScript

HmiTrendCompanion

Properties (RT Uni)

Properties

The "TrendCompanion" object has the following properties:

Properties	Type	Access	Description
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
Caption	String	read/ write	Specifies the text to be displayed in the header.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.

Properties	Type	Access	Description
Icon	String	read/ write	Specifies the icon.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
RenderingTemplate	String	read only	Returns the "RenderingTemplate" property.
ShowAlways	Bool	read/ write	Specifies whether the TrendCompanion can be closed.
SnapToSourceControl	Bool	read/ write	Specifies whether the TrendCompanion snaps to the window of the associated TrendControls.
SourceTrendControl	Object	read only	Returns the "SourceTrendControl" object.
StatusBar	Object	read only	Returns the "StatusBar" object.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Returns the position of the object in the tab order.
TimeZone	HmiTimeZone	read/ write	Specifies the time zone.
ToolBar	Object	read only	Returns the "ToolBar" object.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
TrendCompanionMode	HmiTrendCompanionMode	read/ write	Specifies the window display of the TrendCompanion. • Ruler (0): Reading aid • StatisticArea (1): Statistics area • StatisticResult (2): Statistics result
TrendRulerView	Object	read/ write	Specifies the ruler window of the TrendCompanion.
TrendStatisticAreaView	Object	read only	Returns the "TrendStatisticAreaView" object.
TrendStatisticResultView	Object	read only	Returns the "TrendStatisticResultView" object.
UseSourceControlBackColor	Bool	read/ write	Specifies whether the background color of the table is taken from the associated TrendControl.

Properties	Type	Access	Description
UseSourceControl-TrendColors	Bool	read/ write	Specifies whether the font color of the table is taken from the associated TrendControl.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.
WindowFlags	HmiWindowFlag	read/ write	Specifies the properties of the window. • None (0): Use default setting of the object • ShowCaption (1): Show title • ShowBorder (2): Show border • AlwaysOnTop (4): Always on top • CanSize (8): Can be sized CanMove (16): Can be positioned • CanMaximize (32): Can be maximized • CanClose (64): Can be closed

Methods of "TrendCompanion" (RT Uni)

Overview (RT Uni)

Methods

The "TrendCompanion" object has the following methods:

Methods	Description
-	

"TrendControl" object (RT Uni)

"TrendControl" description (RT Uni)

Description

Represents the "Trend control" object.

Type identifier in JavaScript

HmiTrendControl

Properties (RT Uni)

Properties

The "TrendControl" object has the following properties:

Properties	Type	Access	Description
AlwaysShow-Recent	Bool	read/ write	Specifies whether the newest alarm is displayed at the beginning or end of the list based on the sorting.
AreaSpacing	UInt16	read/ write	Specifies the distance between trend windows.
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
Caption	String	read/ write	Specifies the text to be displayed in the header.
CurrentQuality	HmiQuality	read only	Returns the poorest quality code of all tags which influence the object specified.
Enabled	Bool	read/ write	Specifies whether the specified object can be operated in runtime.
ExtendRuler-ToAxis	Bool	read/ write	Specifies whether the ruler is extended.
Font	Object	read/ write	Specifies the font of the text.
HasFocus	Bool	read only	Returns whether the object has the focus in runtime.
Height	UInt32	read/ write	Specifies the height.
Icon	String	read/ write	Specifies the icon.
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layer	Object	read only	Returns the layer of the screen that contains the object.
Left	Int32	read/ write	Specifies the value of the X coordinate.
Legend	Object	read/ write	Returns the "Legend" object.
Margin	Object	read/ write	Specifies the margin.
Name	String	read only	Returns the name of the object or specifies it.
Online	Bool	read/ write	Specifies the start and stop of the updating.
Rendering-Template	String	read only	Returns the "RenderingTemplate" property.
ShiftAxis	Bool	read/ write	Specifies whether the X axis and Y axis of the control are exchanged.

Properties	Type	Access	Description
ShowRuler	Bool	read/ write	Specifies whether a scale division (help line) is displayed for an axis label of the object.
ShowStatis-ticRulers	Bool	read/ write	Specifies whether the ruler is shown for the statistics area.
StatusBar	Object	read/ write	Returns the "StatusBar" object.
StyleItemClass	String	read/ write	Specifies the style which is applied to the object.
TabIndex	UInt16	read only	Specifies the position of the object in the tab order.
TimeZone	HmiTime-Zone	read/ write	Specifies the time zone.
ToolBar	Object	read/ write	Returns the "ToolBar" object.
Top	Int32	read/ write	Specifies the value of the Y coordinate.
TrendAreas	Object	read only	Returns the "TrendAreas" object.
Visible	Bool	read/ write	Specifies whether the selected object is visible.
Width	UInt32	read/ write	Specifies the width.
WindowFlags	HmiWin-dowFlag	read/ write	Specifies the properties of the window. • None (0): Use default setting of the object • ShowCaption (1): Show title • ShowBorder (2): Show border • AlwaysOnTop (4): Always on top • CanSize (8): Can be sized • CanMove (16): Can be positioned • CanMaximize (32): Can be maximized • CanClose (64): Can be closed

Methods of "TrendControl" (RT Uni)

Overview (RT Uni)

Methods

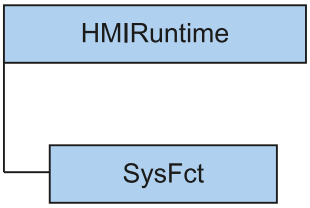
The "TrendControl" object has the following methods:

Methods	Description
-	

7.8.1.12 "SysFct" object (RT Uni)

"SysFct" description (RT Uni)

Description



The "SysFct" object ("HMISysFct" type) enables access to system functions.

Type identifier in JavaScript

HMISysFct

Properties (RT Uni)

Description

The "SysFct" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "SysFct" (RT Uni)

Overview (RT Uni)

Methods

The "SysFct" object has the following methods:

Methods	Description
StopRuntime	Terminates runtime

"StopRuntime" method (SysFct.StopRuntime) (RT Uni)

"StopRuntime" method

Terminates runtime.

Member

Method of the "SysFct" object

Syntax

```
HMIRuntime.SysFct.StopRuntime();
```

Parameters

-

Return value

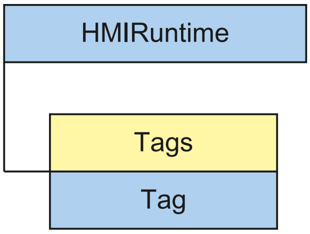
ErrorCode

7.8.1.13 "Tags" area (RT Uni)

"Tags" object (RT Uni)

"Tags" description (RT Uni)

Description



The "Tags" object ("HMITags" type) allows you access to tags in runtime. By default, you reference a "Tag" object ("HMITag" type) through the "Tags" object. The "Tag" object gives you access to all properties and methods of the tags.

Use

Note

The "Tags" object is not a listing like, for example the objects "TagSet" or "AlarmSet", but rather a Factory. You therefore generate an instance of the "Tag" object through the tag name.

The "Tag" objects cannot be counted and enumerated like conventional lists.

The Tags object declares tags ("Tag" objects) for read and write access. The appropriate HMI tags must exist for the read and write access to be executed without errors.

To reduce the use of the "Tags" object, you can also use the alias `Tags` for `HMIRuntime.Tags`.

Type identifier in JavaScript

HMITags

Properties (RT Uni)

Properties

The "Tags" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "Tags" (RT Uni)

Methods

The "Tags" object has the following methods:

Methods	Description
CreateTagSet	Creates a new "TagSet" object (type "HMITagSet").
Item	Returns a new instance of a "Tag" object.

See also

"Item" method (Tags.Item) (Page 666)

"CreateTagSet" method (Tags.CreateTagSet) (Page 666)

"TagSet" description (Page 683)

Overview (Page 673)

"CreateTagSet" method (Tags.CreateTagSet) (RT Uni)

Description

Creates a new "TagSet" object (type "HMITagSet"). The "TagSet" object can be filled with one or multiple tags.

You use the returned "TagSet" object for optimized read and write access to multiple tags.

Member

Method of the "Tag" object

Syntax

```
[HMIRuntime.]Tags.CreateTagSet([tagNameArray]);
```

Note

The `HMIRuntime.` part of the expression is not required. The alias `Tags` stands for `HMIRuntime.Tags`.

Parameter

`tagNameArray`

Optional, type: String or String[]

Name of a tag or array with names of multiple tags that are added to the "TagSet" object. Without parameters, an empty "TagSet" object is created.

Return value

Object of the type "HMITagSet"

See also

Methods of "Tags" (Page 665)

"TagSet" description (Page 683)

"Item" method (Tags.Item) (RT Uni)

Description

Returns a new instance of a "Tag" object.

Member

Method of the "Tags" object

Syntax

```
[HMIRuntime.]Tags[.Item](tagName);
```

Note

The `HMIRuntime.` part of the expression is not required. The alias `Tags` stands for `HMIRuntime.Tags`.

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Tags" object.

Parameter

`tagname`

Type: String

Tag name of a "Tag" object.

Note

The "Tags" object is a Factory. Tags are only referenced through their configured name.

Return value

Object of the type "HMITag"

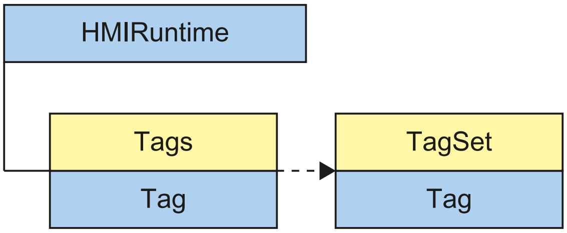
See also

Methods of "Tags" (Page 665)

"Tag" object (RT Uni)

"Tag" description (RT Uni)

Description



The "Tag" object (type "HMITag") represents an HMI tag in runtime. A "Tag" object is returned by the "Tags" object or the "TagSet" list. The "Tag" object gives you access to all properties and methods of a tag.

Type identifier in JavaScript

HMITag

Properties (RT Uni)

Properties

The "Tag" object has the following properties:

Properties	Type	Access	Description
ErrorDescription	String	read	Returns a description of the error code for the last faulty access.
LastError	Error-Code	read	Returns an error code for the last faulty read or write operation.
Name	String	read	Returns the name of the object or specifies it.
QualityCode (Page 670)	UInt32	read/ write	Returns the quality level of a tag value after reading a tag.
TimeStamp	Date-Time	read/ write	Returns the time stamp of the last read operation. The value 0 is returned after writing or failed reading.
Value	Variant	read/ write	Specifies a value for the object being used or returns it.

Value after initialization

The properties of the "Tag" object include the following values after initialization of the object:

Property	After successful initialization
Name	Tag name (unchanged)
Value	VT_EMPTY
QualityCode	BAD NON-SPECIFIC
TimeStamp	0
LastError	0
ErrorDescription	""

Values after a read operation

The properties of the "Tag" object include the following values after the last read operation:

Property	After successful read operation	After unsuccessful read operation
Name	Tag name (unchanged)	
Value	Current tag value	VT_EMPTY
QualityCode	Quality level	BAD OUT OF SERVICE
TimeStamp	Current time stamp of tag	0
LastError	0	Error code of read operation
ErrorDescription	""	Description of the error code

Values after a write operation

The properties of the "Tag" object include the following values after the last write operation:

Property	After successful write operation	After unsuccessful write operation
Name	Tag name (unchanged)	
Value	Current value of the "Tag" object (unchanged)	
QualityCode	BAD OUT OF SERVICE	
TimeStamp	0	
LastError	0	Error code of write operation
ErrorDescription	""	Description of the error code

See also

"QualityCode" property (Page 670)

Special properties (RT Uni)

"QualityCode" property (RT Uni)

Description

Returns the quality level of a tag value after reading a tag.

The quality code has the binary 8-bit structure **QQSSSLL.** The first two positions (QQ) of the quality code define the quality of the tag value:

Quality	Description	Q	Q	S	S	S	S	L	L
Bad	Tag value cannot be used.	0	0	-	-	-	-	-	-
Uncertain	Quality of the tag value is worse than usual. However, it might still be possible to use the tag value.	0	1	-	-	-	-	-	-
Good (Non-Cascade)	Quality of the tag value is good. Attention should be paid to substatus.	1	0	-	-	-	-	-	-
Good (Cascade)	Quality of the tag value is good. Tag value could be used.	1	1	-	-	-	-	-	-

Positions 3 to 6 (SSSS) of the quality code specify the substatus of the quality. Positions 7 and 8 (LL) are optional and define possible limits.

Syntax

```
Object.QualityCode
```

`Object`

Required. An object from the "Availability" section.

Quality code of tags

The realized quality codes are listed in the following table. The table begins with the worst quality code and ends with the best quality code. The best quality code has the lowest priority, while the worst quality has the highest priority. If several statuses occur simultaneously for a tag in the processing chain, the poorest code is passed on.

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x23	Bad	Device passivated - Diagnostic alerts inhibited	0	0	1	0	0	0	1	1
0x3F	Bad	Function check - Local override	0	0	1	1	1	1	1	1
0x1C	Bad	Out of Service - The value is not reliable because the block is not being evaluated, and may be under construction by a configurer. Set if the block mode is O/S.	0	0	0	1	1	1	-	-
0x73	Uncertain	Simulated value - Start	0	1	1	1	0	0	1	1
0x74	Uncertain	Simulated value - End	0	1	1	1	0	1	-	-
0x84	Good (Non-Cascade)	Active Update event - Set if the value is good and the block has an active Update event.	1	0	0	0	0	1	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x24	Bad	Maintenance alarm - More diagnostics available.	0	0	1	0	0	1	-	-
0x18	Bad	No Communication, with no usable value - Set if there has never been any communication with this value since it was last "Out of Service".	0	0	0	1	1	0	-	-
0x14	Bad	No Communication, with last usable value - Set if this value had been set by communication, which has now failed.	0	0	0	1	0	1	-	-
0x0C	Bad	Device Failure - Set if the source of the value is affected by a device failure.	0	0	0	0	1	1	-	-
0x10	Bad	Sensor failure	0	0	0	1	0	0	-	-
0x08	Bad	Not Connected - Set if this input is required to be connected and is not connected.	0	0	0	0	1	0	-	-
0x04	Bad	Configuration Error - Set if the value is not useful because there is some inconsistency regarding the parameterization or configuration, depending on what a specific manufacturer can detect.	0	0	0	0	0	1	-	-
0x00	Bad	Non-specific - There is no specific reason why the value is bad. Used for propagation.	0	0	0	0	0	0	-	-
0x28	Bad	Process related - Substitute value	0	0	1	0	1	0	-	-
0x2B	Bad	Process related - No maintenance	0	0	1	0	1	0	1	1
0x68	Uncertain	Maintenance demanded	0	1	1	0	1	0	-	-
0x60	Uncertain	Simulated value - Set when the process value is written by the operator while the block is in manual mode.	0	1	1	0	0	0	-	-
0x64	Uncertain	Sensor calibration	0	1	1	0	0	1	-	-
0x5C	Uncertain	Configuration error	0	1	0	1	1	1	-	-
0x58	Uncertain	Sub-normal	0	1	0	1	1	0	-	-
0x54	Uncertain	Engineering Unit Range Violation - Set if the value lies outside of the set of values defined for this parameter. The Limits define which direction has been exceeded.	0	1	0	1	0	1	-	-
0x50	Uncertain	Sensor conversion not accurate	0	1	0	1	0	0	-	-
0x4B	Uncertain	Substitute (constant)	0	1	0	0	1	0	1	1
0x78	Uncertain	Process related - No maintenance	0	1	1	1	1	0	-	-
0x4C	Uncertain	Initial Value - Value of volatile parameters during and after reset of the device or of a parameter.	0	1	0	0	1	1	-	-
0x48	Uncertain	Substitute value - Predefined value is used instead of the calculated one. This is used for fail safe handling.	0	1	0	0	1	0	-	-
0x44	Uncertain	Last Usable Value - Whatever was writing this value has stopped doing so. This is used for fail safe handling.	0	1	0	0	0	1	-	-
0x40	Uncertain	Non-specific - There is no specific reason why the value is uncertain.	0	1	0	0	0	0	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0xE0	Good (Cascade)	Initiate Fail Safe (IFS) - The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe.	1	1	1	0	0	0	-	-
0xD8	Good (Cascade)	Local Override (LO) - The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control must be propagated to a function running in a host system for alarm and display purposes. This also implies "Not Invited".	1	1	0	1	1	0	-	-
0xD4	Good (Cascade)	Do Not Select (DNS) - The value is from a block which should not be selected, due to conditions in or above the block.	1	1	0	1	0	1	-	-
0xCC	Good (Cascade)	Not Invited (NI) - The value is from a block which does not have a target mode that would use this input.	1	1	0	0	1	1	-	-
0xC8	Good (Cascade)	Initialization Request (IR) - The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.	1	1	0	0	1	0	-	-
0xC4	Good (Cascade)	Initialization Acknowledge (IA) - The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).	1	1	0	0	0	1	-	-
0xC0	Good (Cascade)	OK - No error or special condition is associated with this value.	1	1	0	0	0	0	-	-
0xA0	Good (Non-Cascade)	Initiate fail safe	1	0	1	0	0	0	-	-
0x98	Good (Non-Cascade)	Unacknowledged Critical Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.	1	0	0	1	1	0	-	-
0x94	Good (Non-Cascade)	Unacknowledged Advisory Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.	1	0	0	1	0	1	-	-
0x90	Good (Non-Cascade)	Unacknowledged Update event - Set if the value is good and the block has an unacknowledged Update event.	1	0	0	1	0	0	-	-
0x8C	Good (Non-Cascade)	Active Critical Alarm - Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.	1	0	0	0	1	1	-	-
0x88	Good (Non-Cascade)	Active Advisory Alarm - Set if the value is good and the block has an active Alarm with a priority less than 8.	1	0	0	0	1	0	-	-
0xA8	Good (Non-Cascade)	Maintenance demanded	1	0	1	0	1	0	-	-
0xA4	Good (Non-Cascade)	Maintenance required	1	0	1	0	0	1	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0xBC	Good (Non-Cascade)	Function check - Local override	1	0	1	1	1	1	-	-
0x80	Good (Non-Cascade)	OK - No error or special condition is associated with this value.	1	0	0	0	0	0	-	-

Limit

The quality codes can be further subdivided by limits. Limits are optional.

Description	Q	Q	S	S	S	S	L	L
O.K. - The value is free to move.	-	-	-	-	-	-	0	0
Low limited - The value has acceded its low limits.	-	-	-	-	-	-	0	1
High limited - The value has acceded its high limits.	-	-	-	-	-	-	1	0
Constant (high and low limited) - The value cannot move, no matter what the process does.	-	-	-	-	-	-	1	1

See also

Properties (Page 668)

Methods of "Tag" (RT Uni)

Overview (RT Uni)

Methods

The "Tag" object has the following methods:

Methods	Description
Decrease	Reduces the current tag value in the AS by the specified value.
Increase	Increases the current tag value in the AS by the specified value.
Read	Reads a tag ("Tag" object).
ResetBit	Deletes a bit of the tag in the AS.
SetBit	Sets a bit of the tag in the AS.
Write	Write tag to AS.
WriteQCD	Writes the values of the tag ("Tag" object).
WriteWithOperatorMessage	Writes the values of the tag ("Tag" object) and subsequently triggers an operator input alarm.

See also

"Decrease" method (Tag.Decrease) (Page 674)

"Increase" method (Tag.Increase) (Page 675)

"Read" method (Tag.Read) (Page 676)

"ResetBit" method (Tag.ResetBit) (Page 677)

"SetBit" method (Tag.SetBit) (Page 679)

"Write" method (Tag.Write) (Page 680)

"WriteQCD" method (Tag.WriteQCD) (Page 681)

"WriteWithOperatorMessage" method (Tag.WriteWithOperatorMessage) (Page 682)

"Decrease" method (Tag.Decrease) (RT Uni)

Description

Reduces the current tag value in the AS by the specified value. The value is written directly to the AS and not returned to the "Tag" object. If you need the changed tag value, execute a "Read" method.

The method prevents multiple transfer of the tag value for reading, calculating and writing.

The method executes an asynchronous write operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern with the error code as parameter is called after the execution. In case of a successful execution, the handler will always receive the error code "0".

For the method to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Member

Method of the "Tag" object

Syntax

```
HMIRuntime.Tag.Decrease(value)
.then(function(ErrorCode) {
    ...
})
```

```
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`value`

Type: Variant

Numerical value by which the current tag value is decreased in the AS.

Return value

Depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

Overview (Page 673)

"Increase" method (Tag.Increase) (RT Uni)

Description

Increases the current tag value in the AS by the specified value. The value is written directly to the AS and not returned to the "Tag" object. If you need the changed tag value, execute a "Read-" method.

The method prevents multiple transfer of the tag value for reading, calculating and writing.

The method executes an asynchronous write operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern with the error code as parameter is called after the execution. In case of a successful execution, the handler will always receive the error code "0".

For the method to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Member

Method of the "Tag" object

Syntax

```
HMIRuntime.Tag.Increase(value)
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`value`

Type: Variant

Numerical value by which the current tag value is increased in the AS.

Return value

Depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

Overview (Page 673)

"Read" method (Tag.Read) (RT Uni)

Description

Reads a tag ("Tag" object). The value, the Quality Code and the time stamp of the tag are determined when the tag is read.

The tags are either read from the tag image (cache) or directly from the AS. When the tag image is used, the method registers all tags that are not yet registered. You should use the tag image for cyclic readout of tags. Use the direct readout (hmiReadDirect) only if you do not need the tag value cyclically or if the update cycle of the tag is too large.

The method executes a synchronous read operation. When completed, you can use the "" properties "LastError" and "ErrorDescription" to determine if the execution was successful.

Member

Method of the "Tag" object

Syntax

```
HMIRuntime.Tag.Read([readType]);
```

Parameters

`readType`

Optional, type: hmiReadType

Origin of the tag values:

- 0 (hmiReadCache) or empty
  Reads the tag value from the tag image. If no registration exists, the tag is registered. For high-performance access you should define the utilized tags as triggers of the script.
- 1 (hmiReadDirect)
  Reads the tag value directly from the AS. The tag image is not used.

Return value

Variant

See also

Overview (Page 673)

"ResetBit" method (Tag.ResetBit) (RT Uni)

Description

Deletes a bit of the tag in the AS. The bit of the tag is written directly to the AS and not returned to the "Tag" object. If you need the changed tag value, execute a "Read" method.

The method prevents multiple transfer of the tag value for reading, calculating and writing.

The method executes an asynchronous write operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern with the error code as parameter is called after the execution. In case of a successful execution, the handler will always receive the error code "0".

For the method to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Member

Method of the "Tag" object

Syntax

```
HMIRuntime.Tag.ResetBit(BitNumber)
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

BitNumber

Type: UInt8, value range: 0-63, depending on the data type of the tag

Bit of the tag that is set to "0" (FALSE).

Return value

Depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

Overview (Page 673)

"SetBit" method (Tag.SetBit) (RT Uni)

Description

Sets a bit of the tag in the AS. The bit of the tag is written directly to the AS and not returned to the "Tag" object. If you need the changed tag value, execute a "Read" method.

The method prevents multiple transfer of the tag value for reading, calculating and writing.

The method executes an asynchronous write operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern with the error code as parameter is called after the execution. In case of a successful execution, the handler will always receive the error code "0".

For the method to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

For internal tags, either

- a start value must be configured, or
- the current value was written by an object, e.g. a script or an I/O field, or
- the current value was generated by tag retentivity.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Member

Method of the "Tag" object

Syntax

```
HMIRuntime.Tag.SetBit(BitNumber)
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`BitNumber`

Type: UInt8, value range: 0-63, depending on the data type of the tag

Bit of the tag that is set to "1" (TRUE).

Return value

Depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

Overview (Page 673)

"Write" method (Tag.Write) (RT Uni)

Description

Writes the values of the tag ("Tag" object). You must first set the values of the individual tags with the "Value" property. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

If the method waits for the write operation to be completed (hmiWriteWait), the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method of the "TagSet" object.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

Member

Method of the "Tag" object

Syntax

```
HMIRuntime.Tag.Write([value],[writeType]);
```

Parameters

`value`

Optional, type: Variant

Writes the tag value:

- Specify value
  The specified value overwrites the current value of the "Value" property of the tag.
- Without value
  The current value of the "Value" property of the tag is written.

`writeType`

Optional, type: hmiWriteType

Specifies if the method waits for the write operation to be completed:

- 0 (hmiWriteNoWait) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- 1 (hmiWriteWait)
  Waits until the tag value is written to the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

Return value

ErrorCode

See also

Overview (Page 673)

"WriteQCD" method (Tag.WriteQCD) (RT Uni)

Description

Writes the values of an internal tag (Tag object), including its quality code and time stamp, both synchronously and asynchronously.

When you call the method for an external tag, it writes the quality code and time stamp predefined by the system, not the one defined by the user.

Use this method to import already logged tag values from a third-party runtime system to WinCC Unified.

Member

Method of the "Tag" object

Syntax

```
HMIRuntime.Tag.WriteQCD([value],[writeType],[TimeStamp],
[QualityCode]);
```

Parameters

`value`

Optional, type: Variant

Writes the tag value.

`writeType`

Optional, type: hmiWriteType

Specifies whether the method waits for the write operation to be completed (`hmiWriteWait`) or not (`hmiWriteNoWait`, default).

`TimeStamp`

Optional, type: DateTime

Writes the time stamp.

`QualityCode`

Optional, type: UInt32

Writes this quality code.

Return value

ErrorCode

"WriteWithOperatorMessage" method (Tag.WriteWithOperatorMessage) (RT Uni)

Description

Writes the values of the tag ("Tag" object) and subsequently triggers an operator input alarm. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

In addition to the reason, the triggered operation message contains the old and new value, the user and host names and the unit.

Once the write operation is completed, the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

Member

Method of the "Tag" object

Syntax

```
HMIRuntime.Tag.WriteWithOperatorMessage(value,reason);
```

Parameters

`value`

Type: Variant

Tag value. The specified value overwrites the current value of the "Value" property of the tags.

`reason`

Type: String

Reason for the value change of the triggered message

Return value

ErrorCode

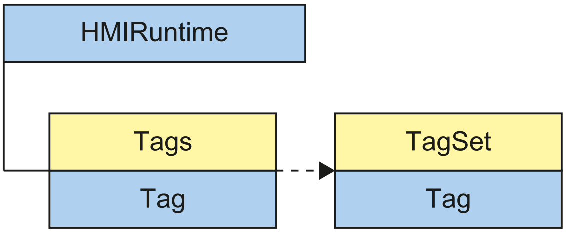
See also

Overview (Page 673)

"TagSet" object (RT Uni)

"TagSet" description (RT Uni)

Description



The "TagSet" object ("HMITagSet" type) is a list of "Tag" objects that gives you optimized access to tags in Runtime. After initialization of the "TagSet" object, you can execute read and write access to multiple tags in one call. Access demonstrates better performance and lower communication load than single access to multiple tags.

You reference a "TagSet" object through the "Tags" object or create a new "TagSet" object with the "Tags.CreateTagSet" method.

By default, you access a "Tag" object (type "HMITag") through the "TagSet" object. The "Tag" object gives you access to all properties and methods of the tags.

Use

The "TagSet" object is a list and can be counted and enumerated. You can access the "TagSet" list using the index or the tag name.

The appropriate HMI tags must exist for the read and write access to tags ("Tag" objects) of the list to be executed without errors. If a read or write access error has occurred, can be read out with the properties "LastError" and "ErrorDescription" once the methods have been executed.

Type identifier in JavaScript

HMITagSet

See also

"Add" method (TagSet.Add) (Page 685)

"Item" method (TagSet.Item) (Page 687)

"Remove" method (TagSet.Remove) (Page 691)

"Clear()" method (TagSet.Clear) (Page 686)

"Read" method (TagSet.Read) (Page 688)

"ReadMaxAge" method (TagSet.ReadMaxAge) (Page 689)

"ReadAsync" method (TagSet.ReadAsync) (Page 690)

"Write" method (TagSet.Write) (Page 692)

"WriteWithOperatorMessage" method (TagSet.WriteWithOperatorMessage) (Page 694)

"WriteAsync" method (TagSet.WriteAsync) (Page 693)

Methods of "Tags" (Page 665)

"CreateTagSet" method (Tags.CreateTagSet) (Page 666)

Overview (Page 673)

Properties (RT Uni)

Properties

The "TagSet" object has the following properties:

Properties	Type	Access	Description
Count	UInt32	read	Returns the number of elements in the specified list.
ErrorDescription	String	read	Returns a description of the error code for the last faulty access.
LastError	Error-Code	read	Returns an error code for the last faulty read or write operation.

Methods (RT Uni)

Overview (RT Uni)

Methods

The "TagSet" object has the following methods:

Methods	Description
Add	Adds a tag ("Tag" object) to the "TagSet" list.
Clear	Removes all tags ("Tag" objects) from the "TagSet" list.
Item	Returns a "Tag" object of the "TagSet" list.
Read	Reads in all tags ("Tag" objects) of the "TagSet" list.
ReadAsync	Reads in all tags ("Tag" objects) of the "TagSet" list.
ReadMaxAge	Reads in all tags ("Tag" objects) of the "TagSet" list and ensures that these are not older than the specified time period (maxAge).
Remove	Removes a tag ("Tag" object) from the "TagSet" list using its tag name.
Write	Writes the values of all tags ("Tag" objects) of the "TagSet" list.
WriteAsync	Writes the values of all tags ("Tag" objects) of the "TagSet" list.
WriteWithOperatorMessage	Writes the values of all tags ("Tag" objects) of the "TagSet" list and then triggers an operator input alarm for each tag.

"Add" method (TagSet.Add) (RT Uni)

Description

Adds a tag ("Tag" object) to the "TagSet" list. The tags are referenced by the name.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.Add(tag);
```

Parameters

tag

Type: String

Names of "Tag" objects that are added to the list.

The following data types are supported:

- Tag name
- Array with tag names
- Two-dimensional array with tag name/value pairs

Note

No "Tag" object can be transferred as a parameter. A "Tag" object is referenced using the name.

Return value

Array with objects of type "HMITag"

See also

"TagSet" description (Page 683)

"Clear()" method (TagSet.Clear) (RT Uni)

Description

Removes all tags ("Tag" objects) from the "TagSet" list.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.Clear();
```

Parameter

--

Return value

--

See also

"TagSet" description (Page 683)

"Item" method (TagSet.Item) (RT Uni)

Description

Returns a "Tag" object of the "TagSet" list.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet[.Item](name);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "TagSet" object.

Parameter

`name`

Type: String

Tag name or index number (1...n) of a "Tag" object in the list

Note

The index number of a "Tag" object does not represent the order in which the "Tag" objects were added to the TagSet list.

Return value

Object of the type "HMITag"

See also

"TagSet" description (Page 683)

"Read" method (TagSet.Read) (RT Uni)

Description

Reads in all tags ("Tag" objects) of the "TagSet" list. The value, the Quality Code and the time stamp of all tags are determined when the tag is read.

The tags are either read from the tag image (cache) or directly from the AS. When the tag image is used, the method registers all tags that are not yet registered. You should use the tag image for cyclic readout of tags. Use the direct readout (hmiReadDirect) only if you do not need the tag value cyclically or if the update cycle of the tag is too large.

The method executes a synchronous read operation. When completed, you can use the "TagSet" properties "LastError" and "ErrorDescription" to determine if the execution was successful.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.Read([readType]);
```

Parameter

`readType`

Optional, type: hmiReadType

Origin of the tag values:

- 0 (hmiReadCache) or empty
  Reads the tag value from the tag image. If no registration exists, the tag is registered. For high-performance access you should define the utilized tags as triggers of the script.
- 1 (hmiReadDirect)
  Reads the tag value directly from the AS. The tag image is not used.

Return value

--

See also

"TagSet" description (Page 683)

"ReadMaxAge" method (TagSet.ReadMaxAge) (RT Uni)

Description

Reads in all tags ("Tag" objects) of the "TagSet" list and ensures that these are not older than the specified time period (maxAge). The value, the Quality Code and the time stamp of all tags are determined when the tag is read.

The tags are read either from the process image (maxAge>0) or directly from the AS (maxAge=0). The method does not use the tag image and does not register tags. You should not use this method for cyclic readout of tags.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "TagSet" object or the error code as parameter.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.ReadMaxAge(maxAge)
.then(function(HMITagSet) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

maxAge

Type: UInt32

Time interval in milliseconds after which a tag value must be updated.

- maxAge = 0
  Read tag value immediately directly from the AS.
- maxAge > 0
  Read tag value from process image according to time stamp.

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMITagSet" as parameter of the "then()" handler.
- Promise rejected
  Error code as parameter of the "catch()" handler. This status only exists when all tags of the "TagSet" object could not be read.

See also

"TagSet" description (Page 683)

"ReadAsync" method (TagSet.ReadAsync) (RT Uni)

Description

Reads in all tags ("Tag" objects) of the "TagSet" list. The value, the Quality Code and the time stamp of all tags are determined when the tag is read.

The tags are either read from the tag image (cache) or directly from the AS. When the tag image is used, the method registers all tags that are not yet registered. You should use the tag image for cyclic readout of tags. Use the direct readout (hmiReadDirect) only if you do not need the tag value cyclically or if the update cycle of the tag is too large.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "TagSet" object or the error code as parameter.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.ReadAsync([readType])
.then(function(HMITagSet) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

readType

Optional, type: hmiReadType

Origin of the tag values:

- 0 (hmiReadCache) or empty
  Reads the tag value from the tag image. If no registration exists, the tag is registered. For high-performance access you should define the utilized tags as triggers of the script.
- 1 (hmiReadDirect)
  Reads the tag value directly from the AS. The tag image is not used.

Return

Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMITagSet" as parameter of the "then()" handler.
- Promise rejected
  Error code as parameter of the "catch()" handler. This status only exists when all tags of the "TagSet" object could not be read.

See also

"TagSet" description (Page 683)

"Remove" method (TagSet.Remove) (RT Uni)

Description

Removes a tag ("Tag" object) from the "TagSet" list using its tag name.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.Remove(tag);
```

Parameters

`tag`

Type: String

Removes a "Tag" object from the "TagSet" list.

The following data types are supported:

- Tag name
- Array with tag names
- Two-dimensional array with tag name/value pairs

Note

No "Tag" object can be transferred as a parameter. A "Tag" object is referenced using the name.

Return value

--

See also

"TagSet" description (Page 683)

"Write" method (TagSet.Write) (RT Uni)

Description

Writes the values of all tags ("Tag" objects) of the "TagSet" list. You must first set the values of the individual tags with the "Value" property. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

If the method waits for the write operation to be completed (hmiWriteWait), the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.Write([writeType]);
```

Parameter value

`writeType`

Optional, type: hmiWriteType

Specifies if the method waits for the write operation to be completed:

- 0 (hmiWriteNoWait) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- 1 (hmiWriteWait)
  Waits until the tag value is written to the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

Return

--

See also

"TagSet" description (Page 683)

"WriteAsync" method (TagSet.WriteAsync) (RT Uni)

Description

Writes the values of all tags ("Tag" objects) of the "TagSet" list. You must first set the values of the individual tags with the "Value" property. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

If the method waits for the write operation to be completed (hmiWriteWait), the properties "LastError" and "ErrorDescription" are written for each tag. This enables you to determine if the execution was successful.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

The method executes an asynchronous write operation without blocking further script execution. The method uses a Promise object to do this which has handlers for the successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "TagSet" object or the error code as parameter. An execution is only faulty ("Promise rejected") when none of the tags of the "TagSet" object could be written.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.WriteAsync([writeType])
.then(function(HMITagSet) {
     ...
})
```

```
.catch(function(ErrorCode) {
    ...
});
```

Parameters

writeType

Optional, type: hmiWriteType

Specifies if the method waits for the write operation to be completed:

- 0 (hmiWriteNoWait) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- 1 (hmiWriteWait)
  Waits until the tag value is written to the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMITagSet" as parameter of the "then()" handler.
- Promise rejected
  Error code as parameter of the "catch()" handler. This status only exists when none of the tags of the "TagSet" object could be written.

See also

"TagSet" description (Page 683)

"WriteWithOperatorMessage" method (TagSet.WriteWithOperatorMessage) (RT Uni)

Description

Writes the values of all tags ("Tag" objects) of the "TagSet" list and then triggers an operator input alarm for each tag. You must first set the values of the individual tags with the "Value" property. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

In addition to the reason, the triggered operation messages contain the old and new value, the user and host names and the unit.

Once the write operation is completed, the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

Member

Method of the "TagSet" object

Syntax

```
HMIRuntime.TagSet.WriteWithOperatorMessage(reason);
```

Parameters

`reason`

Type: String

Reason for the value change of the triggered messages

Return value

ErrorCode

See also

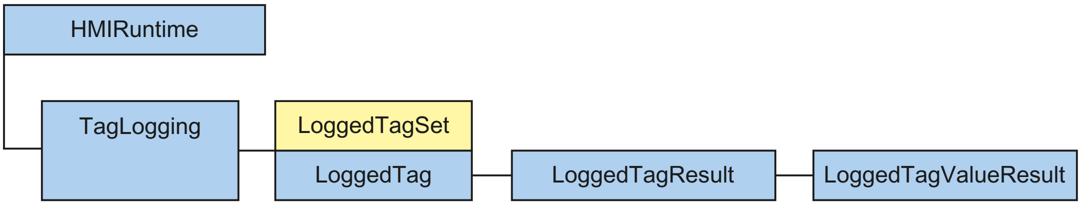
"TagSet" description (Page 683)

7.8.1.14 "TagLogging" area (RT Uni)

"TagLogging" object (RT Uni)

"TagLogging" description (RT Uni)

Description



The "TagLogging" object ("HMITagLogging" type) enables access to the logging tags of a logging system.

Type identifier in JavaScript

HMITagLogging

Properties (RT Uni)

Properties

The "TagLogging" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "TagLogging" (RT Uni)

Overview (RT Uni)

Methods

The "TagLogging" object has the following methods:

Methods	Description
CreateLoggedTagSet	Creates a new "LoggedTagSet" object (type "HMILoggedTagSet").
LoggedTags	References a logging tag ("LoggedTag" object) of a logging system.

"CreateLoggedTagSet" method (TagLogging.CreateLoggedTagSet) (RT Uni)

Description

Creates a new "LoggedTagSet" object (type "HMILoggedTagSet"). The "LoggedTagSet" object can be filled with one or more logging tags.

You use the returned "LoggedTagSet" object for optimized read and write access to multiple logging tags.

Member

Method of the "TagLogging" object

Syntax

```
[HMIRuntime.]TagLogging.CreateLoggedTagSet([loggedTagNameArray]);
```

Parameter

`loggedTagNameArray`

Optional, type: String or String[]

Logging tag name or array with names of multiple logging tags that are added to the "LoggedTagSet" object. Without parameters, an empty "LoggedTagSet" object is created.

Return value

Object of the type "HMILoggedTagSet"

"LoggedTags" method (TagLogging.LoggedTags) (RT Uni)

Description

References a logging tag ("LoggedTag" object) of a logging system.

Member

Method of the "TagLogging" object

Syntax

```
[HMIRuntime.]TagLogging.LoggedTags(loggedTagName);
```

Parameter

`loggedTagName`

Type: String

Logging tag name of a "LoggedTag" object.

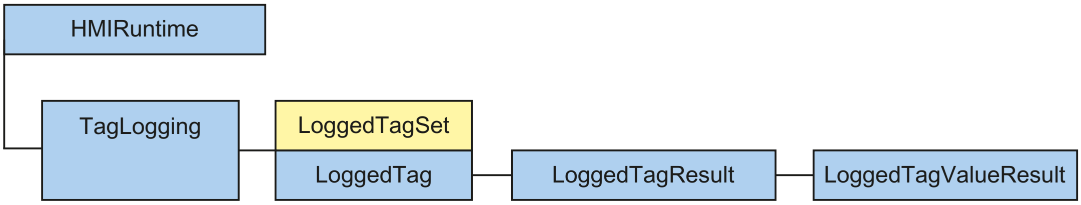
Return value

Object of the type "HMILoggedTag"

"LoggedTagSet" object (RT Uni)

"LoggedTagSet" description (RT Uni)

Description



The "LoggedTagSet" object ("HMILoggedTagSet" type) is a list of "LoggedTag" objects that gives you optimized access to logging tags. After initialization of the "LoggedTagSet" object, you have read access to multiple logging tags in one call. Simultaneous access takes place with better performance and lower communication load than single access to multiple logging tags.

You create a new "LoggedTagSet" object with the "TagLogging.CreateLoggedTagSet" method.

Use

The "LoggedTagSet" object is a list and can be counted and enumerated. You can access the "LoggedTagSet" list using the index or the tag name.

Type identifier in JavaScript

HMILoggedTagSet

Properties (RT Uni)

Properties

The "LoggedTagSet" object has the following properties:

Properties	Type	Access	Description
Count	UInt32	read only	Returns the number of elements in the specified list.
Error	Error-Code	read only	Returns an error code for the last faulty read or write operation. After a successful read or write operation, the value 0 is returned. The error code always relates to the last method call of the object.

Methods of "LoggedTagSet" (RT Uni)

Overview (RT Uni)

Methods

The "LoggedTagSet" object has the following methods:

Methods	Description
Add	Adds logging tags ("LoggedTag" objects) to the "LoggedTagSet" list.
Clear	Removes all tags ("LoggedTag" objects) from the "LoggedTagSet" list.
Item	Returns a "LoggedTag" object of the "LoggedTagSet" list.
Read	Reads out all logging tags ("LoggedTag" objects) of the "LoggedTagSet" list.
Remove	Removes logging tags ("LoggedTag" objects) using their names from the "LoggedTagSet" list.

"Add" method (LoggedTagSet.Add) (RT Uni)

Description

Adds logging tags ("LoggedTag" objects) to the "LoggedTagSet" list. The logging tags are referenced using the name.

Member

Method of the "LoggedTagSet" object

Syntax

```
[HMIRuntime.]TagLogging.LoggedTagSet.Add(loggedTags);
```

Parameters

`loggedTags`

Type: String or String[]

Names of "LoggedTag" objects that are added to the list.

The following data types are supported:

- Logging tag name
- Array with logging tag names

Note

No "LoggedTag" object can be transferred as a parameter. A "LoggedTag" object is referenced using the name.

Return value

Array with objects of type "HMILoggedTag"

"Clear" method (LoggedTagSet.Clear) (RT Uni)

Description

Removes all tags ("LoggedTag" objects) from the "LoggedTagSet" list.

Member

Method of the "LoggedTagSet" object

Syntax

```
[HMIRuntime.]TagLogging.LoggedTagSet.Clear();
```

Parameter

--

Return value

--

"Item" method (LoggedTagSet.Item) (RT Uni)

Description

Returns a "LoggedTag" object of the "LoggedTagSet" list.

Member

Method of the "LoggedTagSet" object

Syntax

```
[HMIRuntime.]TagLogging.LoggedTagSet[.Item](name);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "LoggedTagSet" object.

Parameter

`name`

Type: String or Int32

Logging tag name or index number (1...n) of a "LoggedTag" object in the list

Note

The index number of a "LoggedTag" object does not describe the order in which the "LoggedTag" objects were added to the "LoggedTagSet" list.

Return

Object of type "HMILoggedTag"

"Read" method (LoggedTagSet.Read) (RT Uni)

Description

Reads out all logging tags ("LoggedTag" objects) of the "LoggedTagSet" list. The value, the Quality Code, the time stamp and context information of all logging tags are determined when the logging tag is read.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, after execution is complete, the corresponding handler of the Promise pattern is called with an array with "LoggedTagResult" object or an error code as parameter.

Member

Method of the "TagSet" object

Syntax

```
[HMIRuntime.]TagLogging.LoggedTagSet.Read(dateFrom,dateTo,boundingVa
lue)
.then(function(HMILoggedTagResult) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

dateFrom

Type: DateTIme

Start date of the time period

dateTo

Type: DateTIme

End date of the time period

boundingValue

Type: Bool

Specifies whether the limit values of the time period are transferred.

Return value

Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMILoggedTagResult" as parameter of the "then()" handler.
- Promise rejected
  ErrorCode as parameter of the "catch()" handler. This status only exists when all logging tags of the "LoggedTagSet" object could not be read.

"Remove" method (LoggedTagSet.Remove) (RT Uni)

Description

Removes logging tags ("LoggedTag" objects) using their names from the "LoggedTagSet" list.

Member

Method of the "LoggedTagSet" object

Syntax

```
[HMIRuntime.]TagLogging.LoggedTagSet.Remove(loggedTags);
```

Parameters

`loggedTags`

Type: String or String[]

Removes a "LoggedTag" object from the "LoggedTagSet" list.

The following data types are supported:

- Logging tag name
- Array with logging tag names

Note

No "LoggedTag" object can be transferred as a parameter. A "LoggedTag" object is referenced using the name.

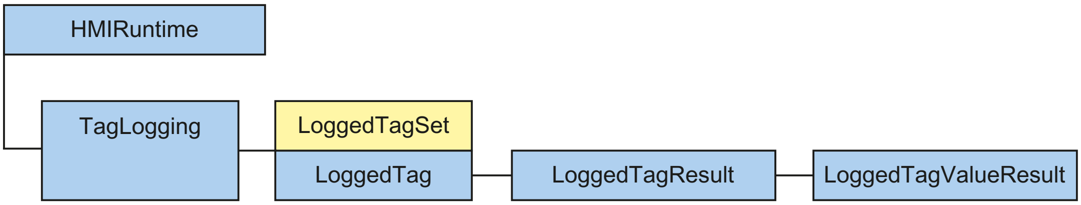
Return value

--

"LoggedTag" object (RT Uni)

"LoggedTag" description (RT Uni)

Description



The "LoggedTag" object ("HMILoggedTag" type) represents a logging tag of a logging system. A "LoggedTag" object is returned by the "TagLogging" object or the "LoggedTagSet" list.

Type identifier in JavaScript

HMILoggedTag

Properties (RT Uni)

Properties

The "LoggedTag" object has the following properties:

Table 7-1 Properties

Properties	Type	Access	Description
Name	String	read only	Returns the name of the object or specifies it.

Methods of "LoggedTag" (RT Uni)

Overview (RT Uni)

Methods

The "LoggedTag" object has the following methods:

Methods	Description
Read	Reads out a logging tag ("LoggedTag" object) of a time period from a logging system.

"Read" method (LoggedTag.Read) (RT Uni)

Description

Reads out a logging tag ("LoggedTag" object) of a time period from a logging system. The value, the Quality Code, the time stamp and context information of the logging tag are determined when the logging tag is read.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, after execution is complete, the corresponding handler of the Promise pattern is called with a "LoggedTagResult" object or an error code as parameter.

Member

Method of the "LoggedTag" object

Syntax

```
[HMIRuntime.]TagLogging.LoggedTag.Read(dateFrom,dateTo,boundingValue
)
.then(function(HMILoggedTagResult) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`dateFrom`

Type: DateTIme

Start date of the time period

`dateTo`

Type: DateTIme

End date of the time period

`boundingValue`

Type: Bool

Specifies whether the limit values of the time period are transferred.

Return value

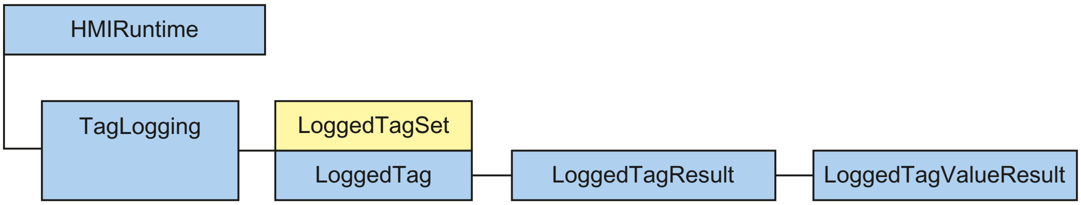
Depending on the status of the Promise object:

- Promise fulfilled
  Object of type "HMILoggedTagResult" as parameter of the "then()" handler.
- Promise rejected
  ErrorCode as parameter of the "catch()" handler.

"LoggedTagResult" object (RT Uni)

"LoggedTagResult" description (RT Uni)

Description



The "LoggedTagResult" object ("HMILoggedTagResult" type) enables access to the process values of a logging tag.

Use

The "LoggedTagResult" object is returned by a read operation of objects "LoggedTag" and "LoggedTagSet" in the logging system. You have access to the process values of logging tags and errors of read operations in the logging system.

Type identifier in JavaScript

HMILoggedTagResult

Properties (RT Uni)

Properties

The "LoggedTagResult" object has the following properties:

Properties	Type	Access	Description
Error	Error-Code	read/ write	Returns an error code for the last faulty read or write operation. After a successful read or write operation, the value 0 is returned. The error code always relates to the last method call of the object.
Name	String	read/ write	Returns the name of the object or specifies it.
Values	Object	read/ write	Returns an array of process values including the quality code.

Methods of "LoggedTagResult" (RT Uni)

Overview (RT Uni)

Methods

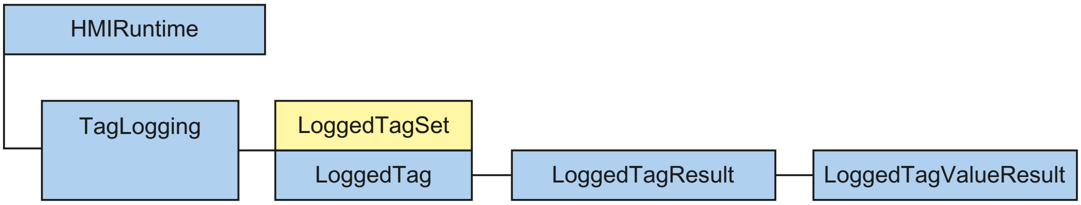
The "LoggedTagResult" object has the following methods:

Methods	Description
-	

"LoggedTagValueResult" object (RT Uni)

"LoggedTagValueResult" description (RT Uni)

Description



The "LoggedTagValueResult" object ("HMILoggedTagValueResult" type) represents the process values of a logging tag.

Use

The "LoggedTagValueResult" object is referenced using the "`LoggedTagResult.Values`" property. The object represents the process value with all associated context information.

Type identifier in JavaScript

HMILoggedTagValueResult

Properties (RT Uni)

Properties

The "LoggedTagValueResult" object has the following properties:

Properties	Type	Access	Description
Flags (Page 708)	HmiTagLoggingValueFlags	read/ write	Returns context information on the process value:
Quality (Page 709)	UInt16 UInt32	read/ write	Returns the quality level.
TimeStamp	DateTime	read/ write	Returns the time stamp of the last read operation. The value 0 is returned after writing or failed reading.
Value	Variant	read/ write	Specifies a value for the object being used or returns it.

See also

"Flags" property (Page 708)

"Quality" property (Page 709)

Special properties (RT Uni)

"Flags" property (RT Uni)

Description

Returns context information on the process value:

- extra (0): There are still additional values at the time of the process value.
- calculated (2): Process value is calculated.
- bounding (16): Process value is a limit value.
- noData (32): No additional information available
- firstStored (64): Process value is the first value stored in the logging system.
- lastStored (128): Process value is the last value stored in the logging system.

Type

HmiTagLoggingValueFlags

Access

Access depends on the object.

Availability

The property is available for the following objects:

- LoggedTagValueResult

Syntax

```
Object.Flags
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 707)

"Quality" property (RT Uni)

Description

Returns the quality level.

- For "AlarmResult" and "LoggedAlarmStateResult" objects:
  Returns the level for the data quality of the alarm state. The quality can have the values "good", "uncertain" or "bad".
  A valid alarm has the following properties:
  - Quality = "good"
  - InvalidFlags = 0
- For "LoggedTagValueResult" objects:
  Returns the level for the quality of a process value of a logging tag.
  For further information, see also property "QualityCode".

Type

- "AlarmResult" or "LoggedAlarmStateResult" objects: UInt16
- "LoggedTagValueResult" object: UInt32

Syntax

```
Object.Quality
```

`Object`

Required. An object from the "Availability" section.

See also

Properties (Page 707)

Methods of "LoggedTagValueResult" (RT Uni)

Overview (RT Uni)

Methods

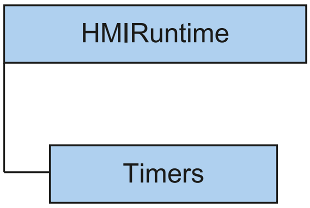
The "LoggedTagValueResult" object has the following methods:

Methods	Description
-	

7.8.1.15 "Timers" object (RT Uni)

"Timers" description (RT Uni)

Description



The "Timers" object (type "HMITimers") allows you to control the script execution time through one-time or cyclic timers. You can execute individual functions delayed or repeatedly.

Type identifier in JavaScript

HMITimers

Example

The execution of the "alertFunc" function is delayed:

Copy code

```
function setDelay() {
    var timerId = Timers.SetTimeout(alertFunc, 5000);
    function alertFunc() {
        alert("SetTimeout triggered");
    }
}
```

Properties (RT Uni)

Properties

The "Timers" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "Timers" (RT Uni)

Overview (RT Uni)

Methods

The "Timers" object has the following methods:

Methods	Description
ClearInterval	Deletes a timer object for cyclic execution of a function.
ClearTimeout	Deletes a timer object for delayed execution of a function.
SetInterval	Creates a timer object for cyclic execution of a function.
SetTimeout	Creates a timer object for one-time, delayed execution of a function.

"ClearInterval" method (Timers.ClearInterval) (RT Uni)

Description

Deletes a timer object for cyclic execution of a function. You can delete a timer object at any time prior to the next execution.

The method requires an ID for referencing the timer object. The ID is returned during the creation of the timer object by the "SetInterval" method.

Member

Method of the "Timers" object

Syntax

```
HMIRuntime.Timers.ClearInterval(TimerID);
```

Parameters

`TimerID`

Type: Int32

ID of the timer object that is deleted. ID is returned by the "SetInterval" method during creation of the timer object.

Return value

--

See also

"SetInterval" method (Timers.SetInterval) (Page 712)

"ClearTimeout" method (Timers.ClearTimeout) (RT Uni)

Description

Deletes a timer object for delayed execution of a function. You can delete a timer object at any time prior to the execution.

The method requires an ID for referencing the timer object. The ID is returned during the creation of the timer object by the "SetTimeout" method.

Member

Method of the "Timers" object

Syntax

```
HMIRuntime.Timers.ClearTimeout(TimerID);
```

Parameters

TimerID

Type: Int32

ID of the timer object that is deleted. ID is returned by the "SetTimeout" method during creation of the timer object.

Return value

--

See also

"SetTimeout" method (Timers.SetTimeout) (Page 713)

"SetInterval" method (Timers.SetInterval) (RT Uni)

Description

Creates a timer object for cyclic execution of a function.

When a time interval has expired, a function is started and scheduled for a new execution according to the time interval.

Member

Method of the "Timers" object

Syntax

```
HMIRuntime.Timers.SetInterval(CallbackFunction,DelayInMillisecs);
```

Parameters

`CallbackFunction`

Type: Function

Function that is executed cyclically.

`DelayInMillisecs`

Type: UInt32

Time interval in milliseconds after which the function is executed cyclically. When the interval is < 10 ms, the value is reset to 10 ms.

Return value

TimerID as Int32

See also

"ClearInterval" method (Timers.ClearInterval) (Page 711)

"SetTimeout" method (Timers.SetTimeout) (RT Uni)

Description

Creates a timer object for one-time, delayed execution of a function. A function is executed in this case when a specified time interval has expired.

Member

Method of the "Timers" object

Syntax

```
HMIRuntime.Timers.SetTimeout(CallbackFunction,DelayInMillisecs);
```

Parameters

`CallbackFunction`

Type: Function

Function that is executed delayed once.

`DelayInMillisecs`

Type: UInt32

Time interval in milliseconds after which the function is executed delayed.

Return value

TimerID as Int32

See also

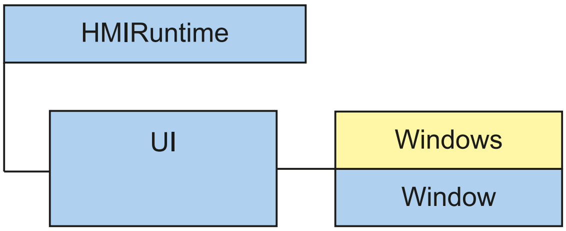
"ClearTimeout" method (Timers.ClearTimeout) (Page 712)

7.8.1.16 "UI" area (RT Uni)

"UI" object (RT Uni)

"UI" description (RT Uni)

Description



The "UI" object ("HMIUI" type) represents the user interface of the graphical runtime system. You use the "UI" object to directly reference the currently active screen or the listing of the screen windows on the highest level.

Application

The "UI" object is used to reference the configured elements of the graphical runtime system, such as screen windows, screens or screen objects. This means that you have access to all the properties and methods of these elements.

To simplify the use of the "UI" object, you can also use the alias `UI` for `HMIRuntime.UI`.

Note

Several screens can be opened simultaneously in runtime. These screens are displayed in screen windows. Each screen window contains exactly one screen (Screen) that can contain any number of further screen windows (Screen Window). The resulting hierarchy can be mapped by using an object path, which is used in the "FindItem" method for addressing.

You can specify one top level screen window (Top Level Screen Window) for each monitor in the window layout of the Runtime system (Screen Window Layout).

Type identifier in JavaScript

HMIUI

Example

Modify the screen of the own screen window on the highest level:

```
UI.RootWindow.Screen = 'NewScreen';
```

or with the "FindItem" method and relative addressing:

```
UI.FindItem('~').Screen = 'NewScreen';
```

Reference and modify the screen of a screen window in absolute terms on the highest level outside of the own screen hierarchy:

```
UI.FindItem('/TopLevelWindow2').Screen = 'NewScreen';
```

Properties (RT Uni)

Properties

The "UI" object has the following properties:

Properties	Type	Access	Description
ActiveScreen	Object	read only	Returns the screen that has the input focus.
RootWindow	Object	read only	Returns the screen window of the highest level (Top Level Screen Window) of the screen in which the script is being executed.

Properties	Type	Access	Description
Style	String	read/ write	Specifies the style of the display and operating objects.
Windows	Object	read only	Returns the "Windows" object with the list of the screen windows.

Methods of "UI" (RT Uni)

Overview (RT Uni)

Methods

The "UI" object has the following methods:

Table 7-2 Methods

Methods	Description
FindItem	Searches for and references screen windows or screen items through their object path.
OpenFaceplateInPopup	Opens a Faceplate in a popup window and sets the values of the faceplate interface.

See also

"OpenFaceplateInPopup" method (HMIUI.OpenFaceplateInPopup) (Page 718)

"FindItem" method (UI.FindItem, ScreenInterface.FindItem) (RT Uni)

Description

Searches for and references screen windows or screen objects through their object path.

Member

Method of the following objects:

- UI
- ScreenInterface

Syntax

```
Object.FindItem(ScreenItemPath);
```

Parameter

`ScreenItemPath`

Type: String

Object path of the searched screen window or screen object.

Note

The "UI.FindItem" method has a global search context and requires absolute object paths. The "Screen.FindItem" method has the current screen as the search context and can also use relative object paths.

Formulation of the object path

The syntax of the object path orients itself to the notation of tile system paths. The object path consists of the names of the screen windows (Screen Windows) and screen objects (Screen Items). The names are connected via a slash ("/") according to the hierarchical positioning. Screens (Screens) and their names are not used in the formulation.

Relative and absolute objects paths are distinguished by the prefix of the object path. The following prefixes can be used:

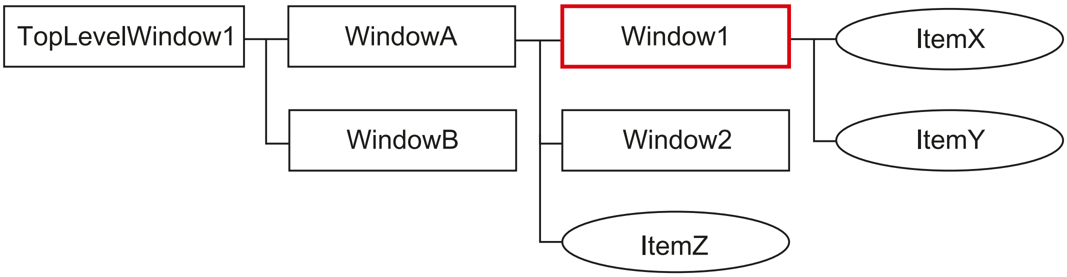
- Relative object path
  - "..": References the higher level screen window (parent) in the context of the current screen window.
  - ".": References the own screen window (self).
  - "": A screen object of the current screen window is referenced without prefix.
- Absolute object path
  - "/": References a screen window on the highest level, whose name must follow.
  - "~": References the screen window on the highest level in the own screen hierarchy.

Further rules for formulating an object path:

- The string ".." may be used several times in the object path, but only together at the beginning of the object path, for example, "../../Window5".
- If the object path does not end with a screen object name, a screen window is referenced.
- An automatic search is performed for screen objects of the object path in the screens of the referenced screen window. It is not permitted to specify a screen name.

Examples of object paths

The following window / screen object hierarchy is adopted for the following examples:



□ Current script context/screen window of the current screen

The following objects paths for addressing the object result from this:

Object path	Referenced object
ItemX	"ItemX" screen object
.	"Window1" screen window
./ItemX	"ItemX" screen object
..	"WindowA" screen window
../ItemZ	"ItemZ" screen object
../ItemZ/Axes/5	5th element of the axis list of the screen object "ItemZ"
../Window2	"Window2" screen window
../..	Screen windows "TopLevelWindow1" on the highest level
/TopLevelWindow1	Screen windows "TopLevelWindow1" on the highest level
~	Screen windows "TopLevelWindow1" on the highest level
/TopLevelWindow1/WindowB	"WindowB" screen window
~/WindowB	"WindowB" screen window
/	Invalid as the name of a screen window is missing on the highest level.

Return value

HmiScreenObjectBase

See also

"ScreenItem" object (Page 730)

"OpenFaceplateInPopup" method (HMIUI.OpenFaceplateInPopup) (RT Uni)

Description

Opens a Faceplate in a popup window and sets the values of the faceplate interface.

Member

Method of the "HMIUI" object

Syntax

```
HMIRuntime.HMIUI.OpenFaceplateInPopup(faceplateType, title,
parenScreen, invisible);
```

Parameters

`faceplateType`

Type: String, HmiFaceplateType

Faceplate type

`title`

Type: String

Faceplate title

`interface`

Type: Object

Faceplate interface

`parentScreen`

Type: Object, HmiScreen

Parent screen or screen item of the faceplate

`invisible`

Type: Bool

Causes the faceplate to be configured so that it is not visible to the operator.

To display the faceplate, set the property `visible=true`.

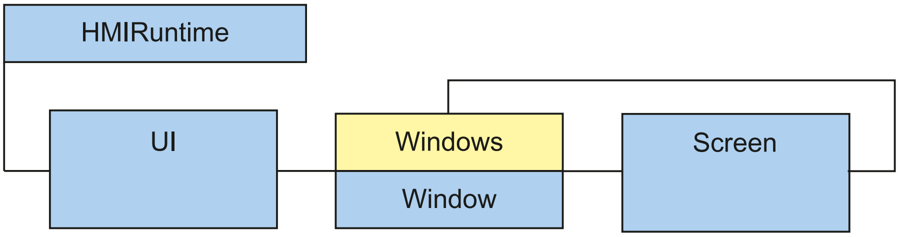
Return value

Object, HmiPopupScreenWindow

"Windows" object (RT Uni)

"Windows" description (RT Uni)

Description



The "Windows" object ("HMIWindows" type) is a list of "Window" objects that enables you to access all screen windows in runtime.

You reference the "Windows" object via the property of the "UI" or "Screen" object.

By default, you access a "Window" object ("HMIWindow" type) through the "Windows" object. The screen with all the contained elements is available through the "Window" object.

Use

The "Windows" object is a list and can be counted and enumerated. You can access the "Windows" list using the index or the screen window name.

Note

If `UI.Windows` is used, the list contains all the screen windows on the highest level.

If `Screen.Windows` is used, the list contains all the screen windows of the screen.

Note

Several screens can be opened simultaneously in runtime. These screens are displayed in screen windows. Each screen window contains exactly one screen (Screen) that can contain any number of further screen windows (Screen Window). The resulting hierarchy can be mapped by using an object path, which is used in the "FindItem" method for addressing.

You can specify one top level screen window (Top Level Screen Window) for each monitor in the window layout of the Runtime system (Screen Window Layout).

You can use the "this" and "Item" objects in scripts. The "this" object refers in this case to the screen ("Screen" object) and the "item" object to the screen object ("ScreenItem" object), in which the script is created. For better clarity, you can also use the `Screen` alias for the "this" object.

Type identifier in JavaScript

HMIWindows

Example

The screen of the top level screen window "TopLevelWindowName" is assigned to the "Screen" tag:

```
var Screen = UI.Windows('TopLevelWindowName').CurrentScreen;
```

The screen of the first top level screen window is assigned to the "Screen" tag:

```
var Screen = UI.Windows(0).CurrentScreen;
```

The screen of the top level screen window of the screen's own screen hierarchy is assigned to the "Screen" tag:

```
var Screen = UI.FindItem('~').CurrentScreen;
```

Change the screen in the adjacent screen window "Window2":

```
Screen.ParentScreen.Windows('Window2').Screen = 'NewScreen';
```

or with the "FindItem" method and relative addressing:

```
Screen.FindItem('../Window2').Screen = 'NewScreen';
```

Properties (RT Uni)

Properties

The "Windows" object has the following properties:

Properties	Type	Access	Description
Count	UInt32	read only	Returns the number of elements in the specified list.

Methods of "Windows" (RT Uni)

Overview (RT Uni)

Methods

The "Windows" object has the following methods:

Methods	Description
Item	Returns a "Window" object of the "Windows" list.

"Item" method (Windows.Item) (RT Uni)

Description

Returns a "Window" object of the "Windows" list.

Member

Method of the "Windows" object

Syntax

```
HMIRuntime.UI.Windows[.Item](WindowName);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Windows" object.

Note

If `UI.Windows` is used, the list contains all the screen windows on the highest level.

If `Screen.Windows` is used, the list contains all the screen windows of the screen.

Parameter

`WindowName`

Type: String

Name of a screen window

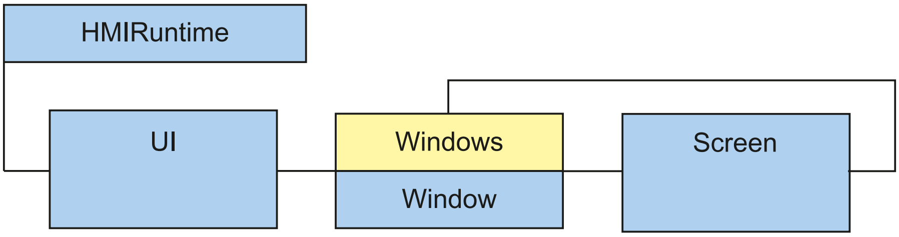
Return value

Object of the type "HmiWindow"

"Window" object (RT Uni)

"Window" description (RT Uni)

Description



The "Window" object ("HMIWindow" type) represents a screen window in runtime. A screen window contains exactly one screen ("Screen" object).

The "Window" object is returned by the "Windows" list, the "FindItem" method or the "Screen.CurrentWindow" property.

Use

You use the "Window" object to reference a "Screen" object and have access to all the objects and properties of a screen.

Note

Several screens can be opened simultaneously in runtime. These screens are displayed in screen windows. Each screen window contains exactly one screen (Screen) that can contain any number of further screen windows (Screen Window). The resulting hierarchy can be mapped by using an object path, which is used in the "FindItem" method for addressing.

You can specify one top level screen window (Top Level Screen Window) for each monitor in the window layout of the Runtime system (Screen Window Layout).

You can use the "this" and "Item" objects in scripts. The "this" object refers in this case to the screen ("Screen" object) and the "item" object to the screen object ("ScreenItem" object), in which the script is created. For better clarity, you can also use the `Screen` alias for the "this" object.

Type identifier in JavaScript

HMIScreenWindowInterface

Example

Change the screen in the adjacent screen window "Window2":

```
Screen.ParentScreen.Windows('Window2').Screen = 'NewScreen';
```

or with the "FindItem" method and relative addressing:

```
Screen.FindItem('../Window2').Screen = 'NewScreen';
```

Properties (RT Uni)

Properties

The "Window" object has the following properties:

Properties	Type	Access	Description
CurrentScreen	Object	read only	Returns the screen of the current screen window.
Parent	Object	read only	Returns the higher-level object instance (Parent) that contains the current object instance as child.
Path	String	read only	Returns the absolute object path of a screen window in runtime starting from the screen window on the highest level.
Screen	String	read/ write	Specifies the name of the screen ("HMIScreen" type) that is contained in the referenced screen window. Loads a new screen into the referenced screen window via its name. The "Screen" property returns a different value than the "CurrentScreen" property when the referenced screen is not yet loaded completely or does not exist.

Methods of "Window" (RT Uni)

Overview (RT Uni)

Methods

The "Window" object has the following methods:

Methods	Description
-	

"PopupScreenWindowAddProps" object (RT Uni)

"PopupScreenWindowAddProps" object (RT Uni)

Description

Assigns parameters for the screen window.

Type identifier in JavaScript

HMIPopupScreenWindowAddProps

Abbreviation	Access in runtime
R	Read
RW	Read and write
-	No access

Table 7-3 Properties

Properties	Access	Description
Monitor	R	Specifies the monitor on which the window is displayed.
StartupPosition	RW	Specifies the position of the screen window at runtime start.

Table 7-4 Methods

Methods	Description
-	

Properties (RT Uni)

Properties

The "PopupScreenWindowAddProps" object has the following properties

Properties	Type	Access	Description
Monitor	UInt8	read	Specifies the monitor on which the window is displayed.
StartupPosition	HmiWindowStartupPosition	read/ write	Specifies the position of the screen window at runtime start.

"PopupScreenWindowAddProps" methods (RT Uni)

Overview (RT Uni)

Methods

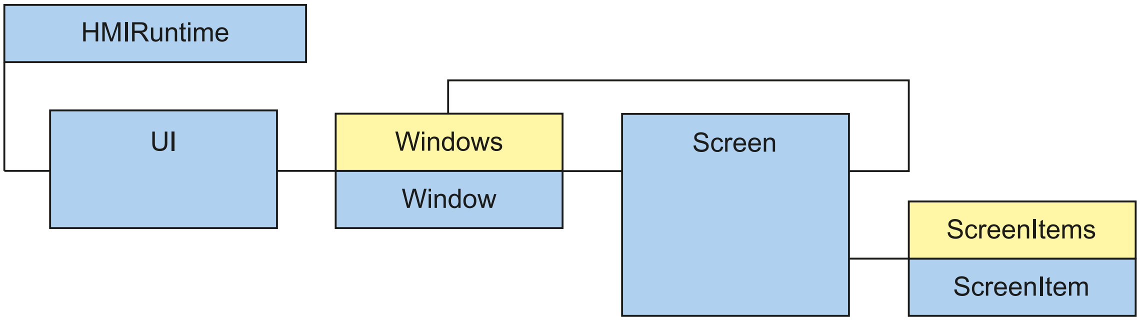
The "PopupScreenWindowAddProps" object has the following methods

Methods	Description
-	-

"Screen" object (RT Uni)

"Screen" description (RT Uni)

Description



Represents a screen in runtime.

Type identifier in JavaScript

HmiScreen

Properties (RT Uni)

Properties

The "Screen" object has the following properties:

Properties	Type	Access	Description
AlternateBackColor	UInt32	read/ write	Specifies the second color for a color gradient.
Authorization	Object	read only	Returns the operator authorization.
BackColor	UInt32	read/ write	Specifies the background color.
BackFillPattern	HmiFillPattern	read/ write	Specifies the pattern of the background or the fill.
BackGraphic	String	read/ write	Specifies the graphic for the image background.
BackGraphic-StretchMode	HmiGraphic-StretchMode	read/ write	Specifies the type of scaling of the background graphic in the screen.
BackgroundFill-Mode	HmiBackground-FillMode	read/ write	Specifies the fill area of the background fill.
DisplayName	String	read/ write	Specifies the display name.

Properties	Type	Access	Description
Enabled	Bool	read/write	Specifies whether the specified object can be operated in runtime.
Height	UInt32	read/write	Specifies the height.
HorizontalAlignment	HmiHorizontalAlignment	read/write	Specifies the horizontal alignment: • Left (0): • Center (1) • Right (2): • Stretch (3)
IsAuthorized	Bool	read only	Returns whether the current user has sufficient rights.
Layers	Object	read only	Returns the list of the "Layers" type.
Name	String	read only	Returns the name of the object or specifies it.
ScreenMaster	Object	read/write	Returns the "ScreenMaster" object.
ScreenNumber	UInt16	read only	Returns the screen number.
VerticalAlignment	HmiVerticalAlignment	read/write	Specifies the vertical alignment: • Top (0) • Center (1) • Bottom (2) • Stretch (3)
Width	UInt32	read/write	Specifies the width.

Methods of "Screen" (RT Uni)

Overview (RT Uni)

Methods

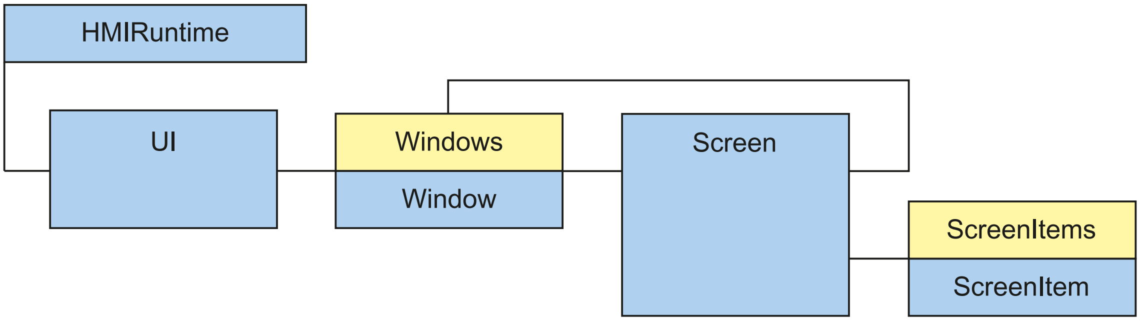
The "Screen" object has the following methods:

Methods	Description
-	

"Screenitems" object (RT Uni)

"ScreenItem" description (RT Uni)

Description



The "ScreenItems" object ("HMIScreenItems" type) is a list of "ScreenItem" objects of a screen. The "ScreenItem" objects enable you to access all configured screen objects in runtime, such as text boxes, buttons or graphical objects.

By default, you reference a "ScreenItem" object ("HMIScreenItem" type) through the "ScreenItems" object.

Use

The "ScreenItems" object is a list and can be enumerated. You can access the "ScreenItems" list using the index or the screen window name.

Note

The list cannot be counted.

You can use the "this" and "Item" objects in scripts. The "this" object refers in this case to the screen ("Screen" object) and the "item" object to the screen object ("ScreenItem" object), in which the script is created. For better clarity, you can also use the `Screen` alias for the "this" object.

Type identifier in JavaScript

HMIScreenItems

Properties (RT Uni)

Properties

The "Screenitems" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "Screenitems" (RT Uni)

Overview (RT Uni)

Methods

The "Screenitems" object has the following methods:

Methods	Description
Item	Returns a "ScreenItem" object of the "ScreenItems" list.

"Item" method (ScreenItems.Item) (RT Uni)

Description

Returns a "ScreenItem" object of the "ScreenItems" list.

Member

Method of the "ScreenItems" object

Syntax

```
Object.ScreenItems[.Item](ScreenItemName);
```

Object

Required. An object of the type "HMIScreen"

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Windows" object.

Parameter

`ScreenItemName`

Type: String

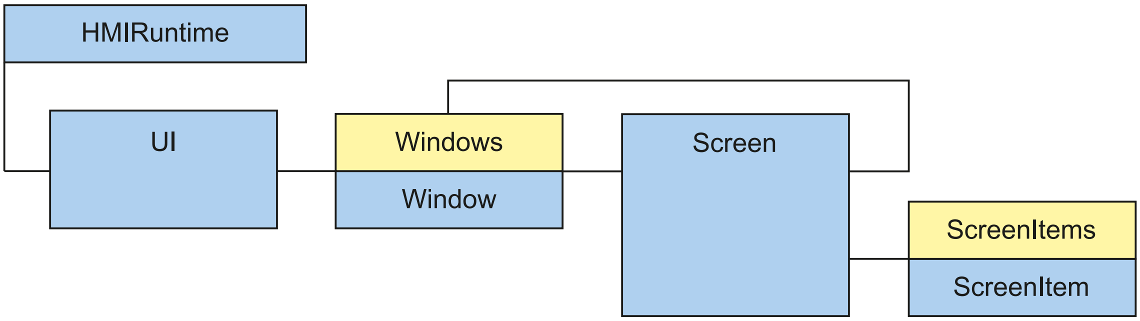
Name of a screen object of the "ScreenItems" list.

Return value

Object of the type "HmiScreenItemBase"

"ScreenItem" object (RT Uni)

Description



The "ScreenItem" object ("HMIScreenIterm" type) represents the configured screen objects in runtime, such as text fields, buttons or graphical objects. The screen objects have different object types depending on their characteristics; for example, a "GaugeControl" screen object is an object of type "HMIGauge". Depending on the characteristics of the "ScreenItem" object you can access different properties of a screen object.

Object properties

In addition to standard properties, each "ScreenItem" object has specific properties that depend on the respective object type.

Note

An overview of the "ScreenItem" objects and their properties is available in the screen object model.

Use

You reference a "ScreenItem" object and have access to all the properties of this screen object through the "ScreenItems" list or the "FindItem" method.

You can use the "this" and "Item" objects in scripts. The "this" object refers in this case to the screen ("Screen" object) and the "item" object to the screen object ("ScreenItem" object), in which the script is created. For better clarity, you can also use the `Screen` alias for the "this" object.

Example

Change the color of the screen object "ItemX" of screen item's own screen. If defined in the "ItemX" screen object:

```
item.BackColor = 0;
```

Or if defined in another screen object of the screen:

```
item.Parent.Items('ItemX').BackColor = 0;
```

or more simply with "Screen" as the current screen:

```
Screen.Items('ItemX').BackColor = 0;
```

or with the "FindItem" method from the current screen:

```
Screen.FindItem('ItemX').BackColor = 0;
```

"ScreenItemInterface" object (RT Uni)

"ScreenItemInterface" description (RT Uni)

Description

Internal use only.

Type identifier in JavaScript

HMIScreenItemInterface

Properties (RT Uni)

Properties

The "ScreenItemInterface" object has the following properties:

Properties	Type	Access	Description
Parent	Object	read only	Returns the higher-level object instance (parent), which contains the current object instance as child.

Methods of "ScreenItemInterface" (RT Uni)

Overview (RT Uni)

Methods

The "ScreenItemInterface" object has the following methods:

Methods	Description
-	-

"ScreenObjectBase" object (RT Uni)

"ScreenObjectBase" description (RT Uni)

Description

Internal use only.

Type identifier in JavaScript

HMIScreenObjectBase

Properties (RT Uni)

Properties

The "ScreenObjectBase" object has the following properties:

Properties	Type	Access	Description
-			

Methods of "ScreenObjectBase" (RT Uni)

Methods (RT Uni)

Methods

The "ScreenObjectBase" object has the following methods:

Methods	Description
-	

"ScreenObjectBaseInterface" object (RT Uni)

"ScreenObjectBaseInterface" description (RT Uni)

Description

Internal use only.

Type identifier in JavaScript

HMIScreenObjectBaseInterface

Properties (RT Uni)

Properties

The "ScreenObjectBaseInterface" object has the following properties:

Table 7-5 Properties

Properties	Type	Access	Description
Parent	Object	readonly	Returns the higher-level object instance (parent), which contains the current object instance as child.

Methods of "ScreenObjectBaseInterface" (RT Uni)

Overview (RT Uni)

Methods

The "ScreenObjectBaseInterface" object has the following methods:

Methods	Description
-	

Configuring text lists and graphic lists (RT Uni) 8

8.1 Configuring text lists (RT Uni)

8.1.1 Basics of text lists (RT Uni)

Introduction

Texts are assigned to the values of a tag in a text list. During configuration, you assign the text list to a text field, for example. This supplies the text to be displayed to the object.

You create and edit the text list in the "Text and graphic list" editor. You configure the interface between the text list and a tag at the object that uses the text list.

The selection of objects that can have a text list assigned depends on the runtime.

Application

You use the text list to output texts depending on the tag value, for example, or to display a selection list in a list box. The associated texts are displayed in the list box depending on the value of the configured tags.

Note

Display of tag values without text

The display of tag values to which no text has been assigned depends on the runtime:

- The display and operating element remains empty.
- Three asterisks *** are displayed.

Ranges for the text list

Three types are available for the text lists:

- Value/Range
  This setting assigns text entries from the text list to integer values or value ranges of a tag. You can select the number of text entries as needed. The maximum number of entries depends on the HMI device you are using.
  You specify a default value which is shown if the value of the tag lies outside the defined range.
- Bit (0, 1)
  This setting assigns text entries from the text list to two states of a binary tag. You can create a text entry for each state of the binary tag.
- Bit number (0 - 31)
  This setting assigns a text entry from the text list to each bit of a tag. The maximum number of text entries is 32. This form of text list can be used, for example, in a sequential control chart when processing a sequencer in which only one bit of the used tag may be set. You influence the behavior of the bit number (0 - 31) with the set bit of the least significance and a default value.

Multilingual texts

You can configure multiple languages for the texts in a text list. The texts will then be displayed in the set language in runtime. To this purpose you set the languages in the Project window under "Languages & Resources > Project languages."

Configuration steps

The following steps are necessary to display texts in a screen object:

1. Creating the text list
2. Assignment of the texts to values or value ranges of a text list
3. Assigning a text list in the display object
4. Assigning a tag

8.1.2 Creating a text list (RT Uni)

Introduction

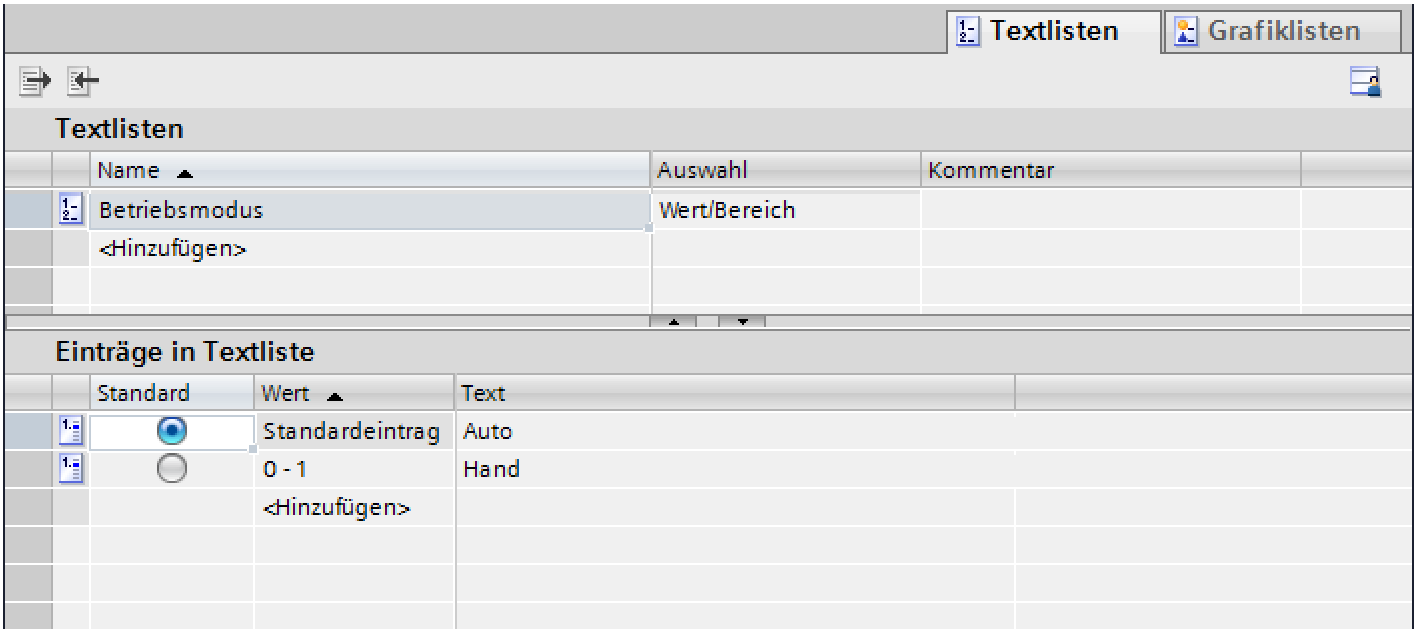
The text list allows you to assign specific texts to values and to output these in runtime, for example, in an I/O field. The type of I/O field can be specified, for example, as a pure input field.

The following types of list are available:

- Value/Range
- Bit
- Bit Number

Procedure

1. Double-click "Text and graphic lists" in the project window.
2. Open the "Text lists" tab.



Textlisten

Name	Auswahl	Kommentar
Betriebsmodus	Wert/Bereich	
<Hinzufügen>		

Einträge in Textliste

Standard	Wert	Text
●	Standardeintrag	Auto
○	0 - 1	Hand
	<Hinzufügen>	

3. Click "Add" in the "Text lists" table.
   The Inspector window of the text list is open.
4. Assign a name to the text list that indicates its function.
5. Select the text list type under "Selection":
   – Value/Range: Text from the text list is displayed when the tag has a value that lies within the specified range.
   – Bit (0,1): A text from the text list is displayed when the tag has the value 0. A different text from the text list is displayed when the tag has the value 1.
   – Bit number (0-31): Text from the text list is displayed when the tag has the value of the assigned bit number.
6. Enter a comment for the text list.

Result

A text list is created.

8.1.3 Assigning texts and values to an area text list (RT Uni)

Introduction

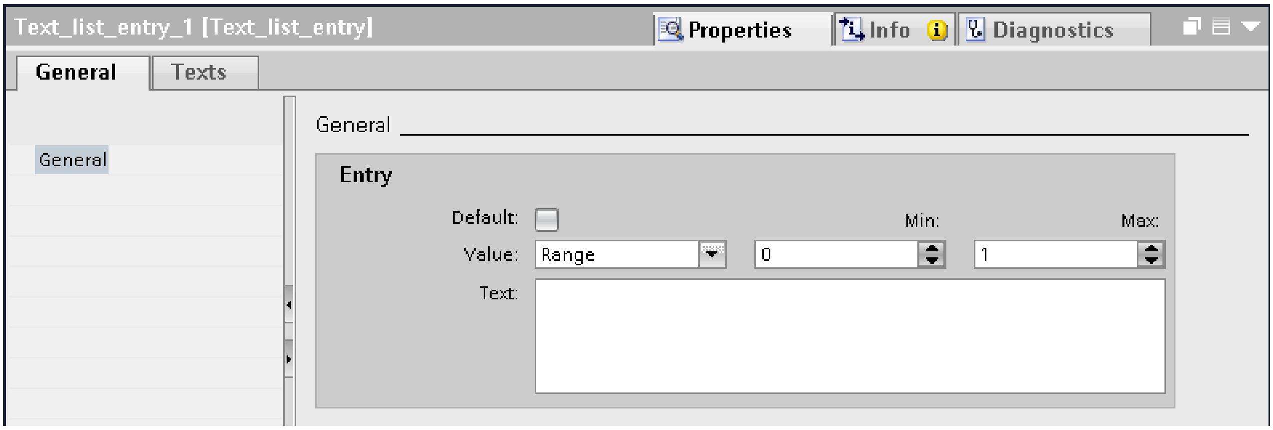
For each area text list you specify which texts are displayed at which value range.

Requirement

- The "Text and graphic list" editor is open.
- The "Text lists" tab is open.
- An area text list has been created and selected.

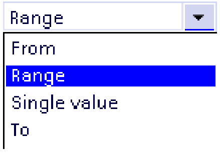
Procedure

1. Click "Add" in the "Text list entries" table.
   The Inspector window for this list entry opens.



Default: □
Value: Range
Min: 0
Max: 1
Text:

2. Select one of the following settings in the Inspector window under "Properties > Properties > General > Value".



   - "Range": Minimum to maximum tag value, for example 1 ≦ Value ≦ 21
   - "To": Maximum tag value, e.g. value ≦ 13
   - "Individual value": Exactly one tag value, for example Value = 21
   - "From": Minimum tag value, e.g. value ≧ 2

3. Enter the text that is displayed in runtime when the tag has the specified value or lies within the specified range of values under "Text."
4. If required, activate the "default entry".
   The entered text is always displayed when the tag has an undefined value. Only one default entry is possible per list.
5. Create further corresponding list entries for additional value ranges of the same text list.

Result

An area text list is created. Texts that appear in runtime are assigned to the possible values.

8.1.4 Assigning texts and values to a bit text list (RT Uni)

Introduction

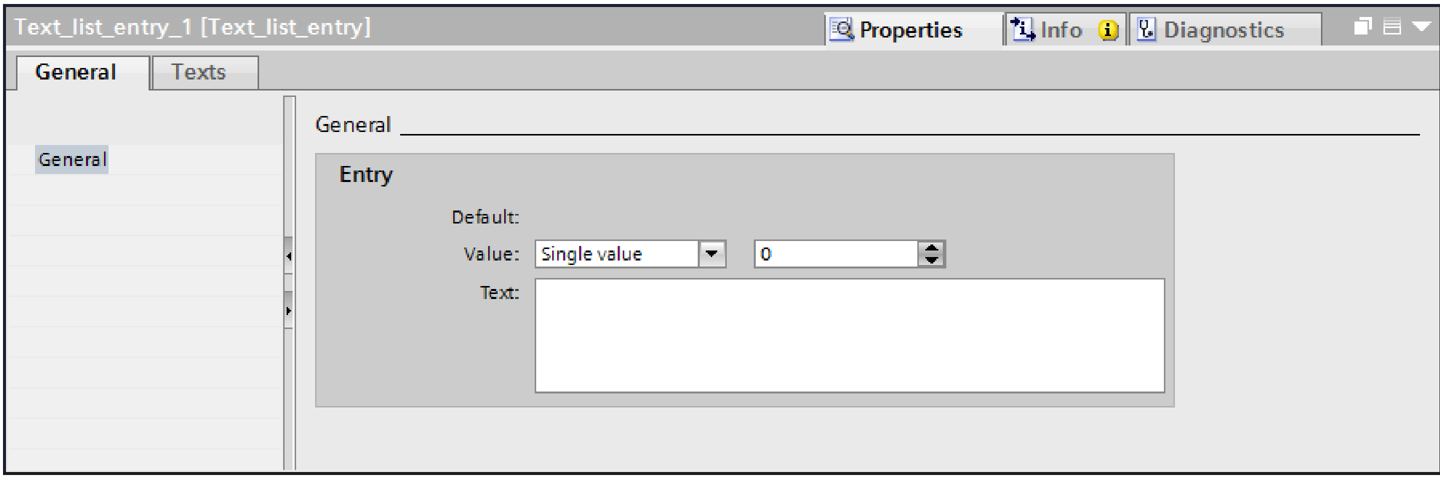
For each text list, you specify which text is displayed at which bit value.

Requirement

- The "Text and graphic list" editor is open.
- The "Text lists" tab is open.
- A bit text list has been created and selected.

Procedure

1. Click "Add" in the "Text list entries" table.
   The Inspector window for this list entry opens.



Default:
Value: Single value 0
Text:

2. Select the setting "Single value" in "Properties > Properties > General > Value" in the Inspector window.
   – Enter "0" for "Value."
   – Enter the text which is displayed in runtime under "Text" if the bit tag is set to "0".
3. Click "Add" in the "Text list entries" table. A second list entry is created.
4. Select the setting "Single value" in "Properties > Properties > General > Value" in the Inspector window.
   – Enter "1" under "Value."
   – Enter the text which is to be displayed in runtime under "Text" if the bit tag is set to "1".

Result

A bit text list is created. Texts that appear in runtime are assigned to the possible values "0" and "1".

8.1.5 Assigning texts and values to a bit number text list (RT Uni)

Introduction

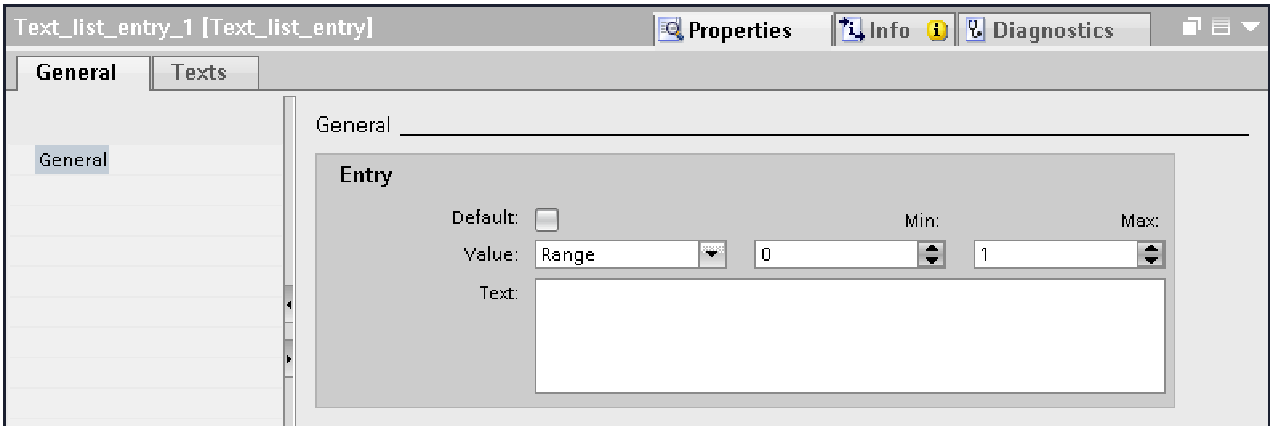
For each bit number text list you specify which texts are displayed at which bit number.

Requirement

- The "Text and graphic list" editor is open.
- The "Text lists" tab is open.
- A bit number text list has been created and selected.

Procedure

1. Click "Add" in the "Text list entries" table.
   The Inspector window for this list entry opens.



Default: □

Value: Range

Min: 0

Max: 1

Text:

2. Select the setting "Single value" in "Properties > Properties > General > Value" in the Inspector window.
   – Enter "10", for example, for "Value".
   – Under "Text", enter the text that is displayed in runtime when the tag has the value "10".
3. If required, activate the "default entry".
   The entered text is always displayed when the tag has an undefined value. Only one default entry is possible per list.
4. Create further list entries for additional bit numbers of the same text list.

Result

A bit number text list is created. Texts that appear in runtime are assigned to the specified bit numbers.

8.1.6 Notes for bit number text list (RT Uni)

Introduction

The bit number (0 - 31) range assigns a text entry from the list to each bit of a tag.

If only 1 bit is configured of all set bits, the stored text is displayed for the configured bit.
In the following example, only the set bit with significance "4" is configured. Text 2 is displayed.

Significance	7	6	5	4	3	2	1	0
Set bits	0	0	1	1	0	1	0	0
Configured	-	Text 3	-	Text 2	Text 1	-	-	-

If no bit is set or when several configured bits are set, no text is displayed.

Default value

Define a default value to prevent an empty display. A configured default value is displayed in the following cases:

- Unified Comfort:
  - The option "Bit selection for text and graphic lists" is disabled. No bit is set in the tag or several configured bits are set.
  - The option "Bit selection for text and graphic lists" is enabled and no bit is set or a text is not configured for the set bit with the least significance.
- Unified PC based: No bit is set in the tag or several configured bits are set.

Displaying the default value

1. Enable the text for the default entry in the "Default" column of the "Text list entries" table.
   The value "Default entry" appears in the "Value" column of the text entry.
2. You can also select the "Default" option under "Properties > General" in the inspector window.

Set bit with the least significance

For Unified Comfort, you have the option of displaying only the text of the set bit with the lowest value. To do this, activate the option "Bit selection for text and graphic lists" under "Screens" in the "Runtime settings" editor.

If no text is configured for the set bit with the least significance and if no default value is configured, nothing is displayed. If a default value is configured, the default value is displayed.

This setting is deselected by default to maintain downward compatibility. The setting is valid for all text lists of the HMI device.

Note

Setting for PC-based

This option is not available for Unified PC-based. If several configured bits are set, the configured default value is displayed. If no default value is configured, nothing is displayed.

Multiline text list entries

Use the <SHIFT>+<RETURN> shortcut to enter a line break in the text entry. Line breaks are represented by the "¶" paragraph mark.

Multiline text list entries are only output in symbolic output fields as well as on buttons with multiple lines. In all other cases, multiline texts are displayed with the paragraph mark.

8.1.7 Configuring object with a text list (RT Uni)

Introduction

The output value and value application for text lists are specified in the display and operating object that displays the texts of the text list in runtime. The properties of these objects are configured as required.

Requirement

- A text list is created.
- You have created a tag.
- The "Screens" editor is open.
- A screen with an text field is open. The object is edited.

Procedure

1. In the Inspector window under "Properties > Text" select the entry "Resource list" in the "Dynamization" column.
2. Select the tag whose values determine the display in the screen object under "Resource list > Tag".
3. Select the text list which you want to have displayed in runtime under "Resource list > Resource list".

Result

The defined texts of the text list are displayed in the text field in runtime when the tag has the specified value.

8.2 Configuring graphic lists (RT Uni)

8.2.1 Basics of graphic lists (RT Uni)

Introduction

The possible values of a tag are assigned to specific graphics in a graphic list. During configuration, assign the graphic list to a button or a graphic view. This supplies the graphics to be displayed to the object.

The graphic lists are created with the "Text and graphic list" editor. You configure the interface between the graphic list and a tag at the object that uses the graphic list. The availability of the graphic list is determined by the HMI device used.

Application

You can configure the graphic list for the following situations:

- Selection list with a graphic display
- State-specific graphic for a button

The graphics in a graphic list can be configured as multilingual. The graphics will then be displayed in the set runtime language.

Graphic sources

Graphics can be added to the graphic list from the following sources:

- By selecting from the project graphics
- Selection of an existing file
  You can use the following file types:
  *.bmp, *.ico, *.emf, *.wmf, *.gif, *.tiff, *.png, *.svg, *.jpeg and *.jpg.
- By creating a new file

Ranges for the graphic list

Three types are available for the graphic lists:

- Value/Range
  This setting assigns graphic entries from the graphic list to integer values or value ranges of a tag. You can select the number of graphic entries as needed. The maximum number of entries depends on the HMI device you are using.
  You specify a default value which is shown if the value of the tag lies outside the defined range.
- Bit (0, 1)
  This setting assigns graphic entries from the graphic list to two states of a binary tag. You can create a graphic entry for each state of the binary tag.
- Bit number (0 - 31)
  This setting assigns a graphic entry from the graphic list to each bit of a tag. The maximum number of graphic entries is 32. This form of graphic list can be used, for example, in a sequence control when processing a sequence in which only one bit of the used tag may be set. You influence the behavior of the bit number (0 - 31) with the set bit of the least significance and a default value.

Configuration steps

The following steps are required to display graphics, for example, in a graphic view:

1. Creating the graphic list
2. Assignment of the graphics to values or value ranges of a graphic list
3. Assigning a graphic list in the display object
4. Assigning a tag

8.2.2 Creating a graphic list (RT Uni)

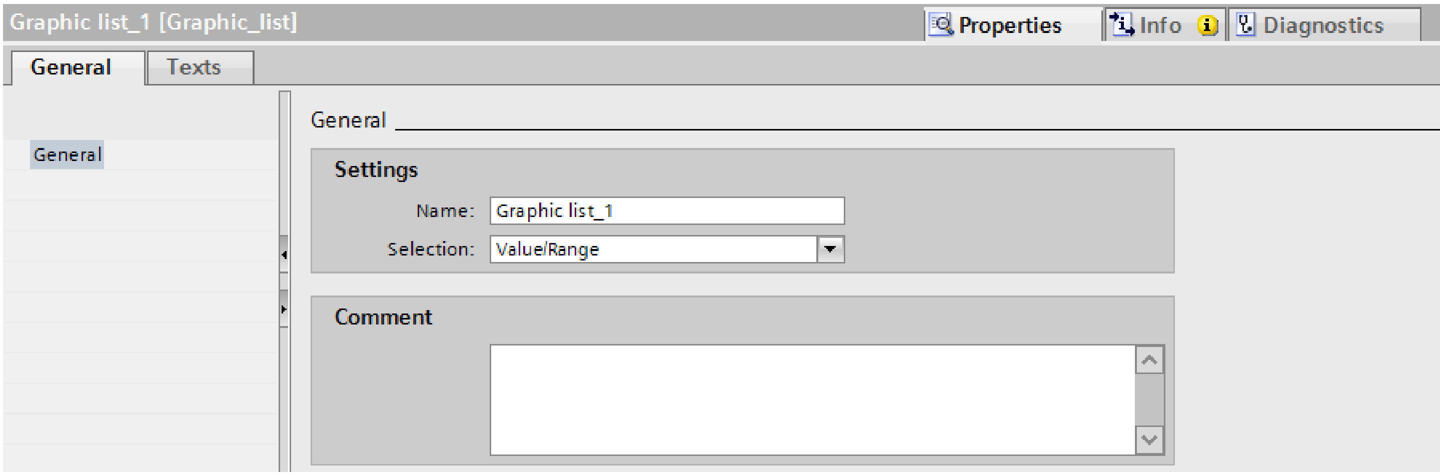
Introduction

The graphic list allows you to assign specific graphics to variable values and to output these in a graphic IO field in runtime. You can specify the type of graphic IO field, for example as a pure output field.

Procedure

1. Double-click "Text and graphic lists" in the project navigation.
2. Open the "Graphic lists" tab.

3. Click "Add" in the "Graphic lists" table.
   The Inspector window of the graphic list will open up.



Settings

Name: Graphic list_1

Selection: Value/Range

Comment

4. Assign a name to the graphic list that indicates its function.
5. Select the "Value/Range" graphic list type under "Selection."
6. Enter a comment for the graphic list.

Result

An area graphic list is created.

8.2.3 Assigning graphics and values to an area graphic list (RT Uni)

Introduction

For each area graphic list you specify which graphics are displayed at which value range. The selected graphic is only displayed when the value is within the permitted range.

The following options are available:

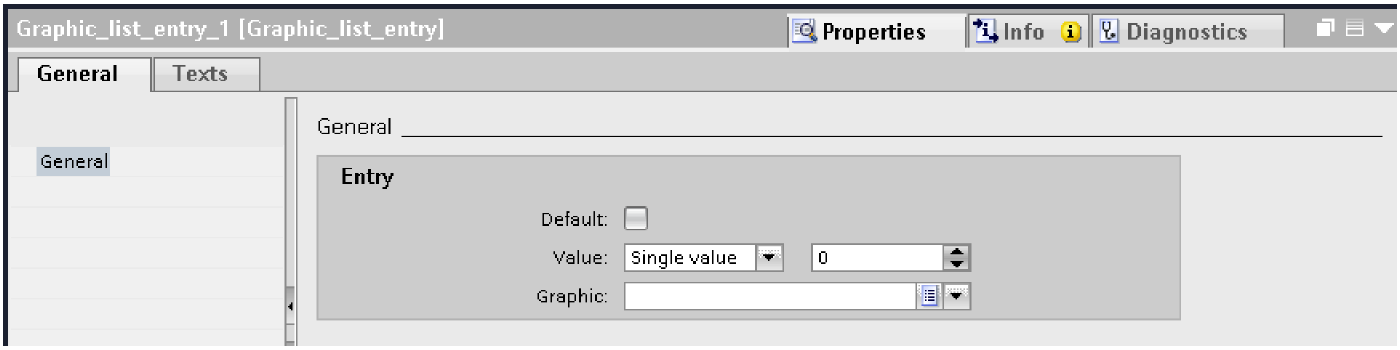
- "Individual value": When the specified bit is set, the selected graphic is displayed in runtime.
- "Range": You enter the minimum value and maximum value for the range.
- "From": You enter the minimum value for the permitted range.
- "To": You enter the maximum value for the permitted range.

Requirement

- The "Text and graphic list" editor is open.
- The "Graphic list" tab is open.
- An area graphic list has been created and selected.

Procedure

1. Click "Add" in the "Graphic list entries" table.
   The Inspector window for this list entry opens.
2. Select the settings "Single value" in "Properties > Properties > General > Value" in the Inspector window:
   – Enter the value "0" for example.
   – Select a graphic which is displayed in runtime when the bit "0" is set.



Graphic_list_entry_1 [Graphic_list_entry]

Entry

Default: □

Value: Single value 0

Graphic:

3. If required, activate the "default entry".
   The graphic is always displayed when the tag has an undefined value. Only one default entry is possible per list.
4. Create further list entries for additional bit numbers of the same graphic list.

Result

An area graphic list is created. Graphics that appear in runtime are assigned to the specified bit numbers.

8.2.4 Assigning graphics and values to a bit graphic list (RT Uni)

Introduction

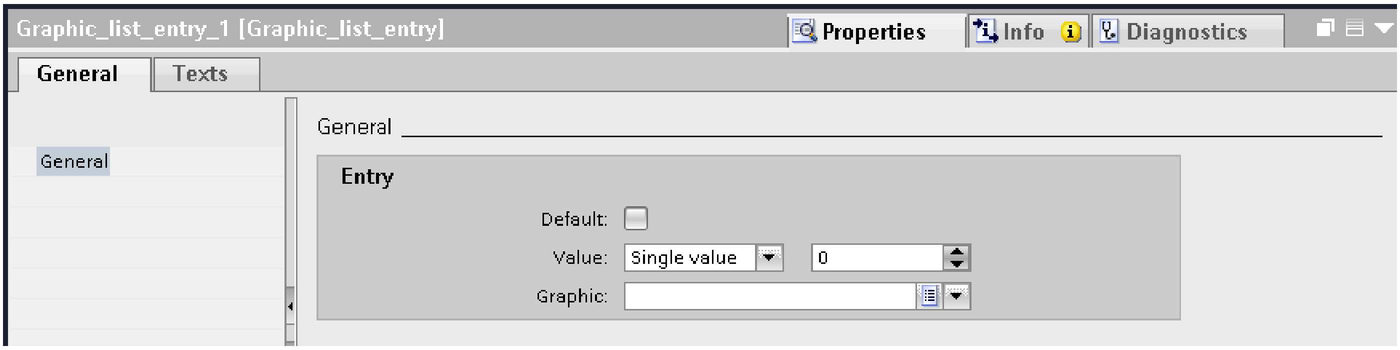
For each graphic list you specify which graphic is displayed at which bit value.

Requirement

- The "Text and graphic list" editor is open.
- The "Graphic list" tab is opened.
- A bit graphic list has been created and selected.

Procedure

1. Click "Add" in the "Graphic list entries" table.
   The Inspector window for this list entry opens.



Default: □

Value: Single value 0

Graphic:

2. Select the settings "Single value" in the inspector window "Properties > Properties > General > Value":
   - Enter "0" as the value.
   - Select a graphic which is displayed in runtime if the bit "0" is set in the tag.

   Note

   As an alternative to the drop-down menu, you can insert graphics from libraries or from your file system:
   1. Select a graphic in the library or in your file system.
   2. Drag-and-drop the graphic into the "Graphic list entries > Graphic" table.

3. Click "Add" in the "Graphic list entries" table. A new list entry is created.
4. Select "Properties > Properties > General > Value > Single value": in the Inspector window.
   - Enter "1" as the value.
   - Select a graphic which is displayed in runtime if the bit "1" is set in the tag.

Result

A bit graphic list is created. Graphics that appear in runtime are assigned to the values "0" and "1".

8.2.5 Assigning graphics and values to a bit number graphic list (RT Uni)

Introduction

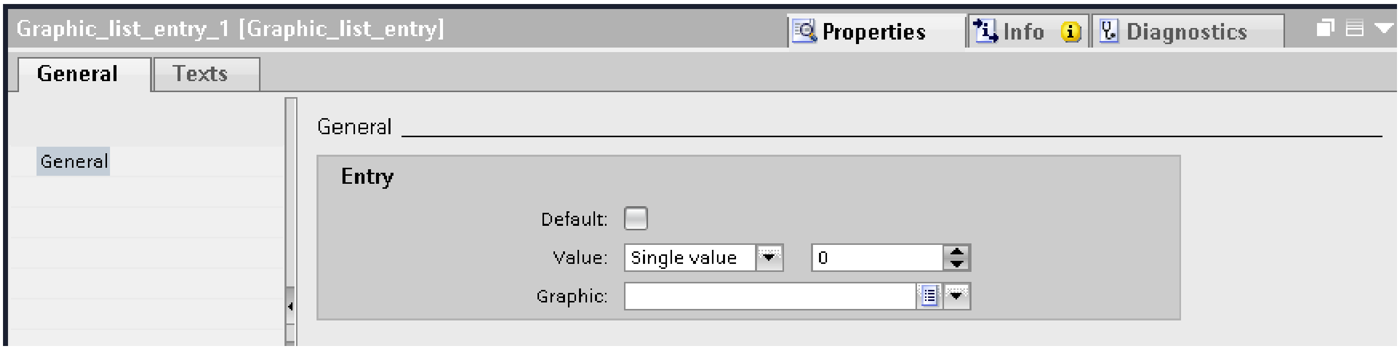
For each bit number graphic list you specify which graphics are displayed at which bit number.

Requirement

- The "Text and graphic list" editor is open.
- The "Graphic list" tab is open.
- A bit number graphic list has been created and selected.

Procedure

1. Click "Add" in the "Graphic list entries" table.
   The Inspector window for this list entry opens.



Graphic_list_entry_1 [Graphic_list_entry]

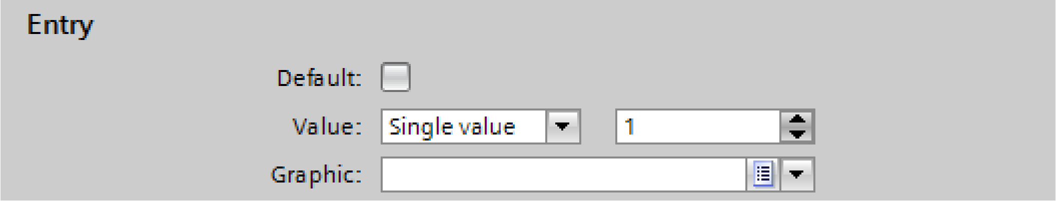
Entry

Default: □

Value: Single value 0

Graphic:

2. Select the settings "Single value" in the Inspector window "Properties > Properties > General > Value":
   - Enter the value "1" for example.
   - Select a graphic which is displayed in runtime if the bit "0" is set in the tag.



Entry

Default: □

Value: Single value 1

Graphic:

Note

As an alternative to the drop-down menu, you can insert graphics from libraries or from your file system:

1. Select a graphic in the library or in your file system.
2. Drag-and-drop the graphic into the "Graphic list entries > Graphic" table.

3. If required, activate the "default entry".
   The graphic is always displayed when the tag has an undefined value. Only one default entry is possible per list.
4. Create further list entries for additional bit numbers of the same graphic list.

Result

A bit number graphic list is created. Graphics that appear in runtime are assigned to the specified bit numbers.

8.2.6 Notes for bit number graphic list (RT Uni)

Introduction

The bit number (0 - 31) range assigns a graphic entry from the list to each bit of a tag.

If only 1 bit is configured of all set bits, the stored graphic is displayed for the configured bit. In the following example, only the set bit with significance "4" is configured. Graphic 2 is displayed.

Significance	7	6	5	4	3	2	1	0
Set bits	0	0	1	1	0	1	0	0
Configured	-	Graphic 3	-	Graphic 2	Graphic 1	-	-	-

If no bit is set or when several bits are set that are also configured, only the placeholder is displayed.

Default value

Define a default value to prevent an empty display. A configured default value is displayed in the following cases:

- The option "Bit selection for text and graphic lists" is disabled. No bit or several configured bits are set in the tag.
- The option "Bit selection for text and graphic lists" is enabled and no bit is set or a graphic is not configured for the set bit with the least significance.

Displaying the default value

1. Enable the graphic for the default entry in the "Default" column of the "Graphic list entries" table.
   The value "Default entry" appears in the "Value" column of the entry.
2. You can also select the "Default" option under "Properties > General" in the inspector window.

Set bit with the least significance

When "Bit selection for text and graphic lists" is enabled, the graphic displayed is the one configured for the set bit with the least significance.

If no graphic is configured for the set bit with the least significance and if no default value is set, the placeholder is displayed. If a default value is configured, the graphic configured for the default value is displayed.

To only display the graphic for the set bit with the least significance, enable the "Bit selection for text and graphic lists" option in the "Runtime settings" editor.

This setting is deselected by default to maintain downward compatibility. The setting is valid for all graphic lists of the HMI device.

8.2.7 Configuring objects with a graphic list (RT Uni)

Introduction

The output value and value application for graphic list are specified in the display and operating object that displays the graphics of the graphic list in runtime. The properties of these objects are configured as required.

Requirement

- A graphic list is created. The values have been defined. Graphics have been assigned to the values.
- You have created a tag.
- The "Screens" editor is open.
- A screen with a graphic view is displayed. The object is edited.

Procedure

1. In the Inspector window under "Properties > Graphic" select the entry "Resource list" in the "Dynamization" column.
2. Select the tag whose values determine the display in the screen object under "Resource list > Tag".
3. Select the graphic list which you want to have displayed in runtime under "Resource list > Resource list".

Result

The defined graphics are displayed in the graphic view in runtime when the tag has the specified value.

Planning tasks (RT Uni) 9

9.1 Basic of the scheduler (RT Uni)

Definition

In the Scheduler, you configure tasks which are only to be executed cyclically or at a specific condition. Each task has a trigger and an action.

Triggers

You use the triggers to define when and how often the task is to be processed during runtime. The following triggers are supported:

Triggers	Type	Description
Time	Cyclic	Executed cyclically at the set time from runtime start, for example, every 2 seconds with "T2s".
Daily, weekly, monthly, yearly	Cyclic	Is executed in cycles starting from runtime start, in each case at the configured time, for example "Daily, 12:00:00 h".
Once	Acyclic	Is executed exactly once at the configured time.
Tags	Acyclic	Executed when the value of one of the projected tags changes.
Alarms	Acyclic	Executed when the state of one of the following alarm properties changes: • Alarm class, for example "Warning" • Alarm state, for example "Incoming" • Priority, for example "4"

Trigger an action

If the configured trigger condition is fulfilled, the event "Update" is triggered. You configure a local script, which triggers one or more actions.

See also

Creating tasks with the "Time" trigger (Page 752)

Creating tasks with the "Tags" trigger (Page 752)

Creating tasks with the "Alarms" trigger (Page 753)

9.2 Creating tasks with the "Time" trigger (RT Uni)

Requirement

- The "Scheduler" editor is open.

Procedure

Follow these steps to create a task with the trigger "Time":

1. Create a new task with "Add".
2. Select the required cycle as the "Trigger", for example "T250ms" for 250 ms.

Result

The task with the "Time" trigger has been created.

See also

Basic of the scheduler (Page 751)

Creating tasks with the "Tags" trigger (Page 752)

Creating tasks with the "Alarms" trigger (Page 753)

9.3 Creating tasks with the "Tags" trigger (RT Uni)

Requirement

- The "Scheduler" editor is open.
- You have created a tag that is monitored for changes in value.

Procedure

Follow these steps to create a task with the trigger "Tags":

1. Create a new task with "Add".
2. Select the option "Tags" as the "Trigger."
3. Select "Properties > Properties > General" in the Inspector window to select the tag.

Result

The task with the "Tags" trigger has been created.

See also

Basic of the scheduler (Page 751)

Creating tasks with the "Time" trigger (Page 752)

Creating tasks with the "Alarms" trigger (Page 753)

9.4 Creating tasks with the "Alarms" trigger (RT Uni)

Requirement

- The "Scheduler" editor is open.

Procedure

Follow these steps to create a task with the trigger "Alarms":

1. Create a new task with "Add".
2. Select the option "Alarms" as the "Trigger".
3. Configure the trigger under "Properties > Properties > General" in the Inspector window.
   - Select the "Criterion", for example "Alarm class".
   - Select the "Condition", for example "Not equal".
   - Select the "Operand", for example "Alarm".

Result

The task with the "Alarms" trigger has been created.

See also

Basic of the scheduler (Page 751)

Creating tasks with the "Time" trigger (Page 752)

Creating tasks with the "Tags" trigger (Page 752)

Configuring in multiple languages (RT Uni) 10

10.1 Languages in WinCC (RT Uni)

User interface language and project languages

A distinction is drawn between two different language levels in WinCC:

- User interface language
  During configuration, the text in the WinCC menus and dialogs is displayed in the user interface language. The user interface language also affects the labeling of operating elements, the parameters of the system functions, the online help, etc.
- Project languages
  Project languages are all languages in which a project will later be used. Project languages are used to create a project in multiple languages.

The two language levels are completely independent of one another. For example, you can create English projects at any time using a German user interface and vice versa.

Project languages

The following languages are differentiated within the project languages:

- Reference language
  The reference language is the language that you use to configure the project initially. During configuration, you select one of the project languages as the reference language. You use the reference language as a template for translations. All of the texts for the project are first created in the reference language and then translated. While you are translating the texts, you can have them displayed simultaneously in the reference language.
- Editing language
  You produce translations of the texts in the editing language.
  Once you have created your project in the reference language, you can translate the texts into the remaining project languages. Select a project language respectively as an edit language and edit the texts for the appropriate language variant. You can change the editing language at any time.

  Note

  When switching the project languages, the assignment to the keys on the keyboard also changes. For some languages (for example, Spanish), the operating system does not allow you to switch to the corresponding keyboard assignment. In this case, the keyboard assignment is switched to English.

- Runtime languages
  Runtime languages are those project languages that are transferred to the HMI device. You decide which project languages to transfer to the HMI device depending on your project requirements.
  You must provide appropriate controls so that the operator can switch between languages in runtime.

10.2 Settings for languages in the operating system (RT Uni)

Introduction

The configuration PC operating system settings influence WinCC language management in the following areas:

- Selection of project languages
- Regional format of dates, times, currency, and numbers
- Displaying ASCII characters

Project language selection

A language is not available as a project language unless it is installed in the operating system.

Regional format of dates, times, currency, and numbers

WinCC specifies a fixed date and time format in the Date - Time field for the selected project language and runtime language.

In order for dates, times, and numbers to be presented correctly in the selected editing language, this language must be set in the Regional Options in the Control Panel.

Displaying ASCII characters

With text output fields, the display of ASCII characters as of 128 depends on the set language and the operating system being used.

If the same special characters are to be displayed on different PCs, the PCs must use the same operating system and regional settings.

10.3 Settings for Asian languages in the operating system (RT Uni)

Settings on Western operating systems

If you want to enter Asian characters, you must activate the support for this language in the operating system.

The Input Method Editor (IME) is available in Windows for configuring Asian texts. Without this editor, you can display Asian text but not edit it. For more information on the Input Method Editor, refer to the documentation for Windows. To enter Asian characters when configuring, switch to the Asian entry method in the "Input Method Editor".

Switch the operating system to the appropriate language to have language-specific project texts, such as alarm texts, displayed in the simulator in Asian characters.

Settings on Asian operating systems

If you are configuring on an Asian operating system, you must switch to the English default input language to enter ASCII characters, for example, for object names. As the English default input language is included in the basic installation of the operating system, you do not need to install an additional input locale.

Enabling language support

1. Open the system controller.
2. Select "Regional and Language Options".
3. On the "Languages" tab, activate the check box "Install files for East Asian languages".
4. Then click on "Details" under "Text Services and Input Languages". The dialog "Text Services and Input Languages" is opened.

5. On the "Settings" tab add the required default input language under the "Installed Services".
6. Select the language of the operating system in the "Language for non-Unicode programs" area in the "Advanced" tab.

10.4 Setting project languages (RT Uni)

10.4.1 Selecting the user interface language (RT Uni)

Introduction

The user interface language is used for displaying menu entries, title bars, infotexts, dialog texts and other designations in the WinCC user interface.

You can switch between the installed user interface languages during configuration. The labeling of the operating elements remains in the language you set when you added the object even if you change the user interface language.

Procedure

1. Select "Options > Settings" in the menu.
   The "Settings" dialog box is opened.
2. Select the desired user interface language under "General > General settings".

Result

WinCC will use the selected language as user interface language.

10.4.2 Enabling project languages (RT Uni)

Introduction

The project languages are set in the "Project languages" editor. You define which project language is to be the reference language and which the editing language.

Enabling project languages

1. Click on the arrow to the left of "Languages & resources" in the project tree.
   The lower-level elements will be displayed.
2. Double-click on "Project languages".
   The possible project languages will be displayed in the working area.
3. Enable the relevant project languages.

Note

Copying multilingual objects

The copies of multilingual objects to a different project only include text objects in the project languages which are activated in the target project. Activate all project languages in the target project to include the corresponding text objects when transferring the copy.

Disabling project languages

1. Disable the languages which are not relevant for the project.

NOTICE
If you disable a project language, all text and graphic objects you have already created in this language will be deleted from the current project.

10.4.3 Selecting the reference language and editing language (RT Uni)

Introduction

The project languages are set in the "Project languages" editor. You define which project language is to be the reference language and which the editing language. You can change the editing language at any time.

Requirements

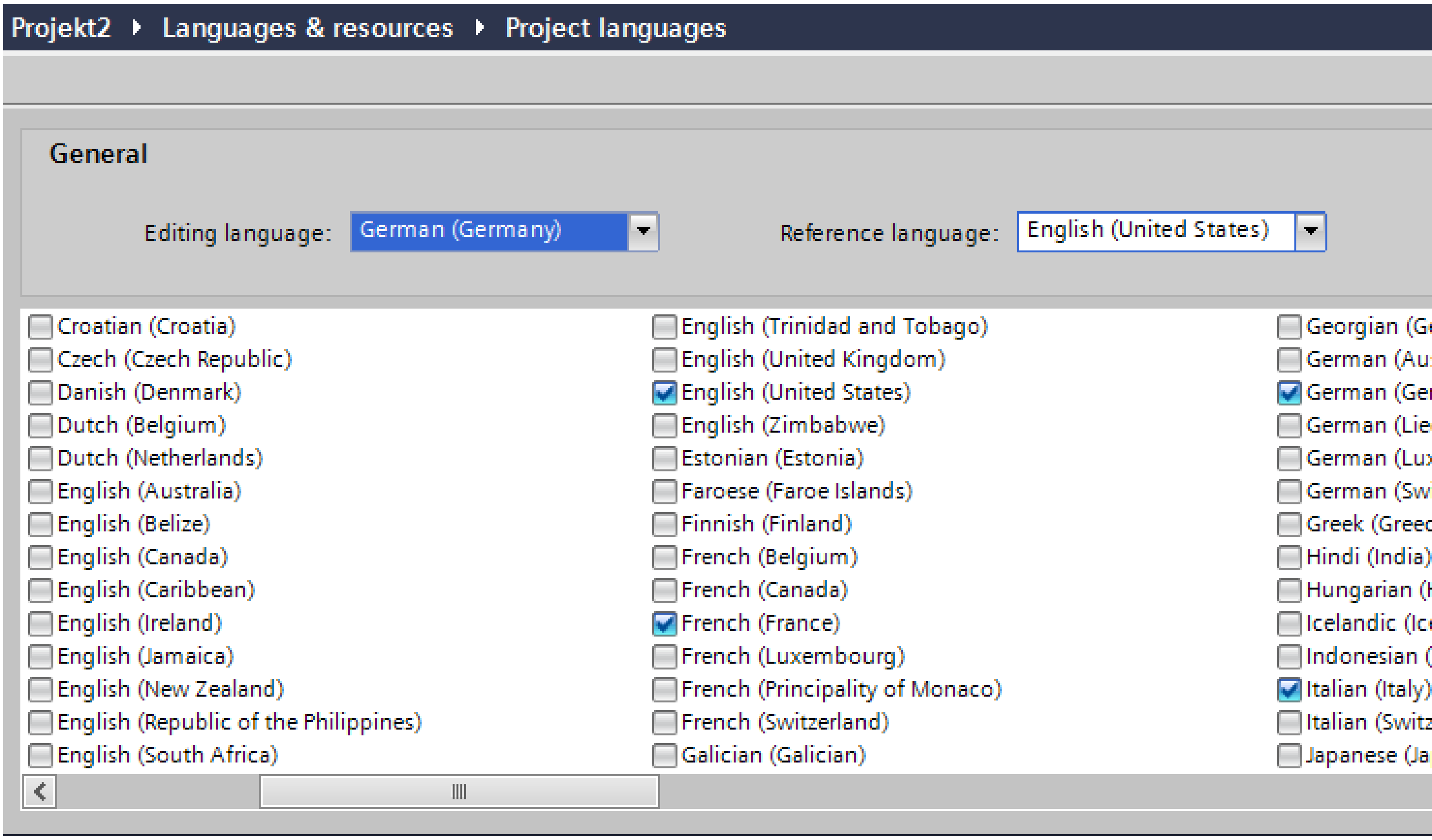
The "Project languages" editor is open.

Several project languages have been activated.

Selecting the reference language and editing language

1. Click the arrow in the drop-down list in the "General > Editing language" section.
2. Click the required language in the drop-down list, for example, German.
3. Click on the arrow in the drop-down list in the "General > Reference language" section.
4. Click the required language in the drop-down list, for example, English.

The language selection is displayed in the list box.



Projekt2 ▸ Languages & resources ▸ Project languages

General

Editing language: German (Germany)

Reference language: English (United States)

- □ Croatian (Croatia)
- □ Czech (Czech Republic)
- □ Danish (Denmark)
- □ Dutch (Belgium)
- □ Dutch (Netherlands)
- □ English (Australia)
- □ English (Belize)
- □ English (Canada)
- □ English (Caribbean)
- □ English (Ireland)
- □ English (Jamaica)
- □ English (New Zealand)
- □ English (Republic of the Philippines)
- □ English (South Africa)
- □ English (Trinidad and Tobago)
- □ English (United Kingdom)
- ☑ English (United States)
- □ English (Zimbabwe)
- □ Estonian (Estonia)
- □ Faroese (Faroe Islands)
- □ Finnish (Finland)
- □ French (Belgium)
- □ French (Canada)
- ☑ French (France)
- □ French (Luxembourg)
- □ French (Principality of Monaco)
- □ French (Switzerland)
- □ Galician (Galician)
- □ Georgian (Ge
- □ German (Au
- ☑ German (Ge
- □ German (Lie
- □ German (Lux
- □ German (Sw
- □ Greek (Gree
- □ Hindi (India)
- □ Hungarian (H
- □ Icelandic (Ic
- □ Indonesian (
- ☑ Italian (Italy)
- □ Italian (Switz
- □ Japanese (Ja

Result

You have now selected the editing and reference languages.

If you change the editing language, all future text input will be stored in the new editing language.

See also

Configuring multilingual alarm texts (Page 247)

10.5 Creating one project in multiple languages (RT Uni)

10.5.1 Working with multiple languages (RT Uni)

Multilingual configuration in WinCC

You can configure your projects in multiple languages using WinCC. There are various reasons for creating a project in multiple languages:

- You would like to use a project in more than one country.
  You create the project in multiple languages but when the HMI device is commissioned, only the language spoken by the operators at the respective site will be transferred to the HMI device.
- The operators of a system speak a range of different languages.
  Example: An HMI device is used in China, but the service personnel understand only English.

Translating project texts

With WinCC, you can enter project texts directly in several languages in various different editors, for example, in the "Project texts" editor. WinCC also allows you to export and import your configuration for translation purposes. This is particularly advantageous if you configure projects containing a large amount of text and want to have it translated.

Language management and translation in WinCC

The following editors are used to manage languages and translate texts in WinCC:

Editor	Short description
Project languages	Selection of project languages, editing language and reference language.
Languages and fonts	Management of runtime languages and fonts used on the HMI device.
Project texts	Central management of configured texts in all project languages.
Graphics	Project graphics for managing graphics and their language-specific versions.

See also

Configuring multilingual alarm texts (Page 247)

10.5.2 Basics of project texts (RT Uni)

Texts in different languages in the project

Texts that are output on display devices during processing are typically entered in the language in which the automation solution is programmed. Comments and the names of objects are also entered in this language.

If operators do not understand this language, they require a translation of all operator-relevant texts into a language they understand. You can therefore translate all the texts into any language. In this way, you can ensure that anyone who is subsequently confronted with the texts in the project sees the texts in his/her language of choice.

User texts and system texts

In the interests of clarity, a distinction is drawn between user texts and system texts:

- User texts are texts created by the user.
- System texts are texts created automatically and which are a product of configuration in the project.

The project texts are managed in the project text editor. This can be found in the project tree under "Languages & Resources > Project texts".

Examples of multilingual project texts

You can, for example, manage the following types of text in more than one language:

- Display texts
- Alarm texts
- Comments in tables
- Labels of screen objects
- Text lists

Translating texts

There are two ways of translating texts.

- Translating texts directly
  You can enter the translations for the individual project languages directly in the "Project texts" editor.
- Translating texts using reference texts
  You can change the editing language for shorter texts. You can enter the new texts in the editing language while the texts of the reference language are displayed.

Missing translation for texts of screen elements

Note

If a text of a screen element has no translation in the current user interface language, the text entered for the default language is displayed.

See also

Configuring multilingual alarm texts (Page 247)

10.5.3 Translating texts directly (RT Uni)

Translating texts

If you use several languages in your project, you can translate individual texts directly. As soon as you change the language of the software user interface, the translated texts are available in the selected language.

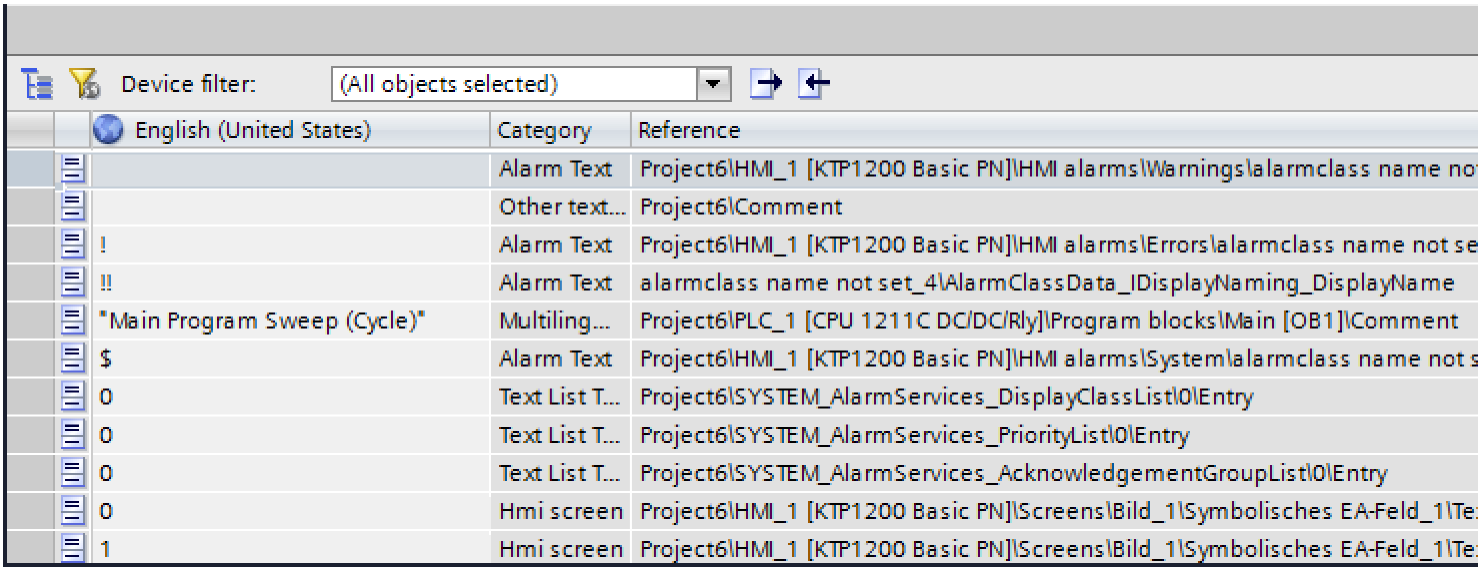
Requirements

- You are in the project view.
- A project is open.
- You have selected at least two further project languages.

Procedure

Proceed as follows to translate individual texts:

1. Click on the arrow to the left of "Languages & resources" in the project tree.
   The elements below this are displayed.
2. Double-click on "Project texts".
   A list with the texts in the project is displayed in the work area. There is a separate column for each project language.



Device filter: (All objects selected)

English (United States)	Category	Reference
	Alarm Text	Project6\HMI_1 [KTP1200 Basic PN]\HMI alarms\Warnings\alarmclass name no
	Other text...	Project6\Comment
!	Alarm Text	Project6\HMI_1 [KTP1200 Basic PN]\HMI alarms\Errors\alarmclass name not se
!!	Alarm Text	alarmclass name not set_4\AlarmClassData_IDisplayNaming_DisplayName
"Main Program Sweep (Cycle)"	Multiling...	Project6\PLC_1 [CPU 1211C DC/DC/Rly]\Program blocks\Main [OB1]\Comment
$	Alarm Text	Project6\HMI_1 [KTP1200 Basic PN]\HMI alarms\System\alarmclass name not s
0	Text List T...	Project6\SYSTEM_AlarmServices_DisplayClassList\0\Entry
0	Text List T...	Project6\SYSTEM_AlarmServices_PriorityList\0\Entry
0	Text List T...	Project6\SYSTEM_AlarmServices_AcknowledgementGroupList\0\Entry
0	Hmi screen	Project6\HMI_1 [KTP1200 Basic PN]\Screens\Bild_1\Symbolisches EA-Feld_1\Te
1	Hmi screen	Project6\HMI_1 [KTP1200 Basic PN]\Screens\Bild_1\Symbolisches EA-Feld_1\Te

3. To group identical texts and translate them simultaneously, click "☰" in the toolbar.
4. To hide texts that do not have a translation, click ⏷ in the toolbar.
5. Click on an empty column and enter the translation.

Result

You have translated individual texts in the "Project texts" editor. The texts will then be displayed in the runtime language.

See also

Configuring multilingual alarm texts (Page 247)

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

10.5.4 Translating texts using reference texts (RT Uni)

Introduction

After changing the editing language, all texts are shown in input boxes in the new editing language. If there is not yet a translation available for this language, the input boxes are empty or filled with default values.

If you enter text again in an input field, this is saved in the current editing language. Following this, the texts exist in two project languages for this input field, in the previous editing language and in the current editing language. This makes it possible to create texts in several project languages.

You can display existing translations for an input box in other project languages. These serve as a comparison for text input in the current editing language and they are known as the reference language.

Requirement

There is at least one translation into a different project language for an input field.

Procedure

To display the translation of an input cell in a reference language, follow these steps:

1. Select "Tasks > Languages & resources" in the task card.
2. Select a reference language from the "Reference language" drop-down list.

Result

The reference language is preset. If you click in a text block, translations that already exist in other project languages are shown in the "Tasks > Reference text" task card.

10.5.5 Exporting project texts (RT Uni)

Project texts are exported for translation. Texts are exported to Office Open XML files ending in ".xlsx". These files can be edited in Microsoft Excel, for example.

You can exchange the file with the translators and import it back to the project as soon as it has been translated.

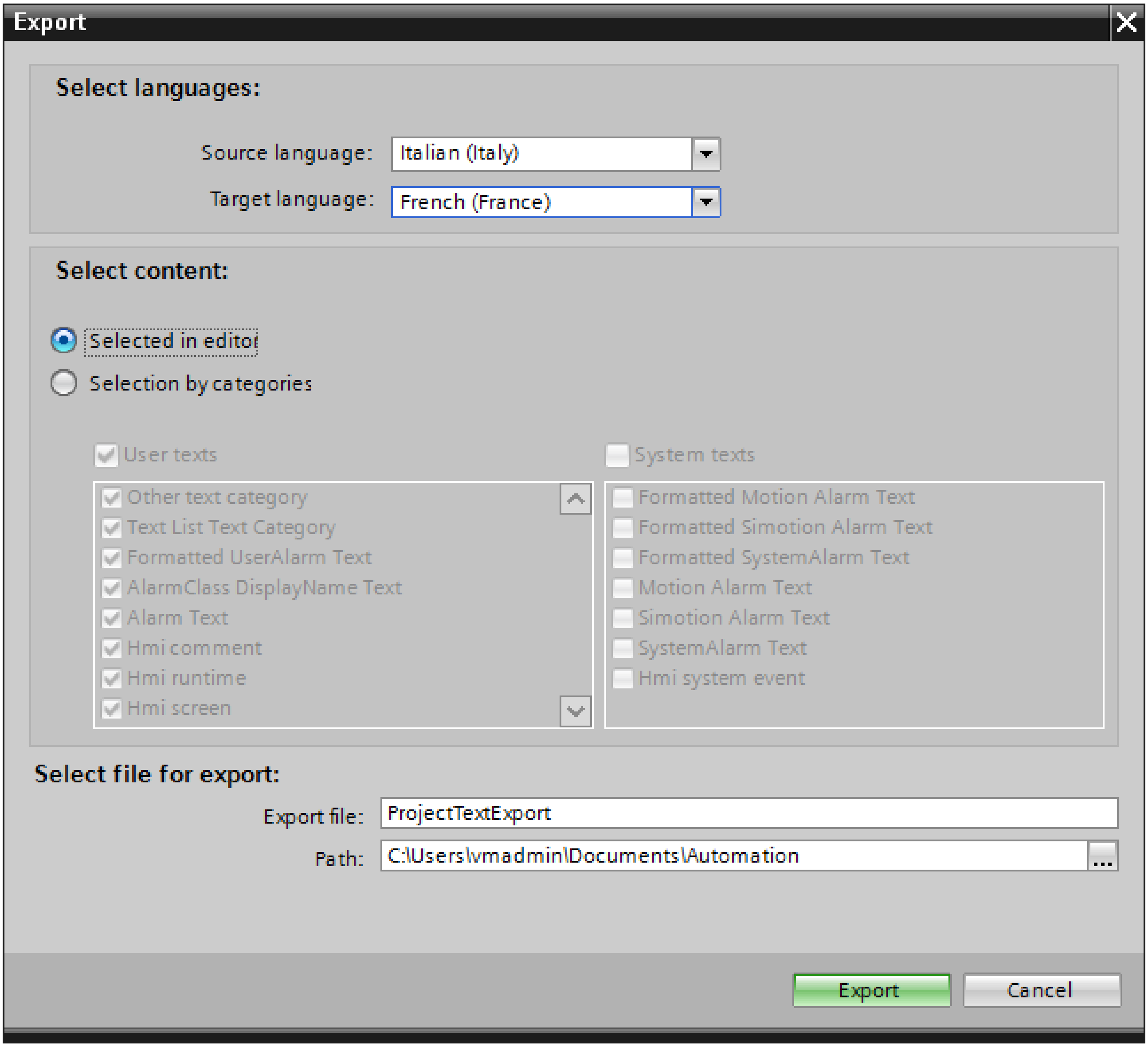
Requirements

- At least two languages have been enabled in the "Project languages" editor, for example, Italian and French.

Exporting project texts

To export individual project texts, proceed as follows:

1. Click on the arrow to the left of "Languages & resources" in the project tree.
   The child elements are displayed.
2. Double-click on "Project texts". The "Project texts" editor will open.
3. Select the texts you want to export.
4. Click ⇨. The "Export" dialog opens.



Export

Select languages:

Source language: Italian (Italy)

Target language: French (France)

Select content:

● Selected in editor

○ Selection by categories

☑ User texts

☑ Other text category
☑ Text List Text Category
☑ Formatted UserAlarm Text
☑ AlarmClass DisplayName Text
☑ Alarm Text
☑ Hmi comment
☑ Hmi runtime
☑ Hmi screen

☐ System texts

☐ Formatted Motion Alarm Text
☐ Formatted Simotion Alarm Text
☐ Formatted SystemAlarm Text
☐ Motion Alarm Text
☐ Simotion Alarm Text
☐ SystemAlarm Text
☐ Hmi system event

Select file for export:

Export file: ProjectTextExport

Path: C:\Users\vmadmin\Documents\Automation

5. From the "Source language" drop-down list, select the language from which you wish to translate, for example Italian.
6. From the "Target language" drop-down list, select the language into which the texts are to be translated, for example, French.

7. Enter a file path and a file name for the export file in the "Export file" input field.
8. Click "Export".

Result

The texts selected in the "Project texts" editor are written to an xlsx file. The xlsx file will be stored in the specified folder.

You can alternatively select and export all project texts from categories. Select "User texts" or "System texts" in the "Export" dialog in line with the type of texts you wish to export. In this case, export can additionally be limited by categories.

Note

Project texts in library objects cannot be exported.

See also

Configuring optional parameters for discrete alarms and analog alarms (Page 244)

10.5.6 Importing project texts (RT Uni)

Edit the xlsx file or send it to a translator. Import the texts once they have been translated. The foreign languages will be imported to the relevant object in the project.

Note

In WinCC, you only import the previously exported project texts into the same project. Importing into a different project is not supported.

Requirements

- At least two languages have been enabled in the "Project languages" editor, for example, Italian and French.

Importing project texts

To import a project text file, proceed as follows:

1. Click on the arrow to the left of "Languages & resources" in the project tree.
   The lower-level elements will be displayed.
2. Double-click on "Project texts". The "Project texts" editor will open.
3. Click ←. The "Import" dialog opens.
4. Select the path and file name of the import file from the "Import file" field.

5. Activate the "Import source language" check box if you have made changes to the source language in the export file and would like to overwrite the entries in the project with the changes.
6. Click on "Import".

Result

You have imported the project texts.

See also

Configuring multilingual alarm texts (Page 247)

10.6 Using language-specific graphics (RT Uni)

10.6.1 "Project graphics" editor (RT Uni)

Introduction

You use the "Project graphics" editor to manage the configured graphic objects in different language versions. Multilingual projects sometimes also require language-specific versions of the graphics, for example, if

- The graphics contain text;
- Cultural aspects play a role in the graphics.

Opening the "Project graphics" editor

Double-click in the project tree on "Languages and resources > Project graphics".

Work area

The work area displays all configured graphic objects in a table. There is a separate column in the table for each project language. Each column in the table contains the versions of the graphics for one particular language.

In addition, you can specify a default graphic for each graphic to be displayed whenever a language-specific graphic for a project language does not exist.

Preview

The preview shows you how the graphics will look on various devices.

10.6.2 Storing an image in the project graphics (RT Uni)

Introduction

You use the "Graphics" editor to import graphics you want to use in screens in the "Screens" editor. It also allows you to manage language-specific versions of graphics. A preview shows the graphic displays on various HMI devices.

Note

File names for language-dependent graphics

Case is relevant in the file names of language-dependent graphics. Make sure that the format is consistent for all languages.

Note

File format of language-dependent graphics

In the language dependent versions of a graphic, only use the graphic files with the same format. The graphic versions with different file formats are not supported.

Requirement

- The language-dependent versions of a graphic are available.
- Multiple languages have been enabled in the "Project languages" editor.
- The "Graphics" editor is open.

Inserting graphics

1. Click "Add" in the "Project graphics" table. A dialog opens.
2. Select the required graphic file.
3. Click "Open" in the dialog box.
   The graphic will be imported to the project and displayed in all cells in this row in the "Graphics" editor.
4. Click in the corresponding cell of a language for which a language-dependent version of this graphic exists.
5. Select "Add graphic" from the shortcut menu. A dialog box opens.
6. Select the desired graphic file and click "Open."
   The language-dependent version is inserted in the table in place of the reference language graphic.
7. Then, in the "Default graphic" column, import a graphic to be displayed in runtime for those languages for which there is no language-specific graphic.

You can also drag&drop a graphic from Windows Explorer to the relevant position in the "Project graphics" table.

Displaying graphics in the HMI device preview

1. Click on a graphic in the table.
2. Select the required HMI device under "Properties > Graphics settings > Device preview" in the Inspector window.
   The graphic will then be displayed as it will appear in runtime on the selected HMI device.

Result

The graphics added are available in the "Graphics" editor. The graphic assigned to the respective editing language will be displayed during editing. The default screen will be displayed in all editing languages for which no screen has been imported.

The screens assigned to the respective runtime language are displayed during runtime. The default screen is displayed in all runtime languages for which a screen has not been imported.

Note

If you disable a project language, all of the graphic objects you have already created in this language will be deleted from the current project.

10.6.3 Storing an external image in the project graphics (RT Uni)

Introduction

To display graphics that have been created in an external graphics program in your screens, you will first have to store these graphics in the project graphics of the WinCC project.

Requirement

- Multiple languages have been enabled in the "Project languages" editor.
- The "Graphics" editor is open.
- There is a graphic in the "Graphics" editor.

Creating and adding a new graphic as an OLE object

1. Click "Add" in the "Project graphics" table. A dialog box opens.
2. Navigate to the folder in which the graphic is stored.
3. Click "Open" in the dialog box.
   The graphic will be imported to the project and displayed in all cells in this row in the "Graphics" editor.

4. Click in the corresponding cell of a language for which a language-dependent version of this graphic exists.
5. Select "Insert object" from the shortcut menu. The "Insert object" dialog box opens.

   Note

   In addition, the dialog "External application running..." will open. The dialog will not close until you exit the external application.

6. Select "Insert object > Create new" and an object type in the dialog.
7. Click "OK." The associated graphic program is opened.
8. Close the graphics program once you have created the graphic.
   The graphic will be stored in the graphic programming software standard format and added to the project graphics.

Inserting created graphics in WinCC

1. Click in the corresponding cell of a language for which a language-dependent version of this graphic exists.
2. Select "Insert object" from the shortcut menu. The "Insert object" dialog box opens.

   Note

   In addition, the dialog "External application running..." will open. The dialog will not close until you exit the external application.

3. From the "Insert object" dialog box, select "Create from file."
4. Click "Browse".
5. Navigate to the created graphic and select it.

   Note

   To import graphics files, note the following size restrictions:

   *.bmp, *.tif, *.emf, *.wmf ≤4 MB

   *.jpg, *.jpeg, *.ico, *.gif "*≤1 MB

Result

The OLE objects added are available in the "Graphics" editor.

Versions of the graphics for the current editing language are displayed in the "Screens" editor. The default graphic is displayed in all editing languages for which no screen has been imported.

The graphic is displayed in runtime in the set runtime language. The default graphic is displayed in all runtime languages for which no graphic has been imported.

You can double-click OLE objects in your project graphics to open them for editing in the corresponding graphic editor.

10.7 Languages in runtime (RT Uni)

10.7.1 Languages and fonts in runtime (RT Uni)

Using multiple runtime languages

You can decide which project languages are to be used in runtime on a particular HMI device. The number of runtime languages that are available at one time on the HMI device depends on the device. To enable the operator to switch between languages in runtime, you need to configure a corresponding operator control.

When runtime starts, the project is displayed according to the most recent language setting. When runtime starts the first time, the language with the lowest number in the "Order for language setting" is displayed.

"Language & font" runtime setting

Configure the following under "Language & font":

- Project languages available as runtime languages for the relevant device
- The order in which the languages are switched.

Note

Post installing fonts

When a configured font on the configuration computer is missing, the project is not compiled. Re-installation is required.

After installing a font, restart the TIA Portal and completely re-compile your project. Only then can the HMI device be compiled without errors.

Note

Runtime language in controls

If you have configured a control in a specific runtime language that is not a user-interface language, it can happen that some texts, for example, system texts in the status bar, appear in the English language.

Note

Specify runtime languages as input languages

To enter and edit data in runtime, configure the specified runtime languages also as input languages for the keyboard of your PC.

10.7.2 Methods for language switching (RT Uni)

Introduction

You need to configure language switching if you want to have multiple runtime languages available on the HMI device. This is necessary to enable the operator to switch between the various runtime languages.

Methods for language switching

You can configure the following methods for language switching:

- Direct language selection
  Each language is set by means of a separate button. In this case, you create a button for each runtime language.
- Language switching
  The operator switches the languages using a button.

Regardless of the method used, the button names must be translated into each of the languages used. You can also configure an output field that displays the current language setting.

10.7.3 Enabling the runtime language (RT Uni)

Introduction

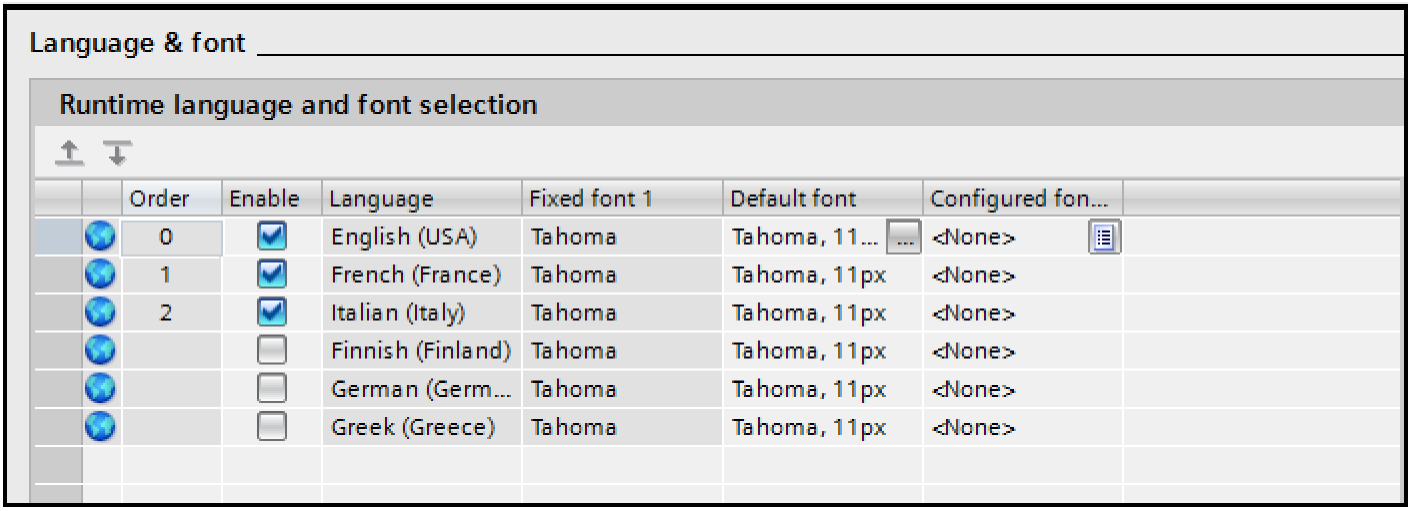
The "Language & Font" editor shows all project languages available in the project. Here you select which project languages are to be available as runtime languages on the HMI device.

Requirements

Multiple languages have been selected in the "Project languages" editor.

Procedure

1. Double-click on "Runtime settings" in the project tree.
2. Click on "Language & Font".
3. Select the following languages:
   - English
   - French
   - Italian



Language & font

Runtime language and font selection

		Order	Enable	Language	Fixed font 1	Default font	Configured fon...
	🌐	0	☑	English (USA)	Tahoma	Tahoma, 11... ...	<None>
	🌐	1	☑	French (France)	Tahoma	Tahoma, 11px	<None>
	🌐	2	☑	Italian (Italy)	Tahoma	Tahoma, 11px	<None>
	🌐		☐	Finnish (Finland)	Tahoma	Tahoma, 11px	<None>
	🌐		☐	German (Germ...	Tahoma	Tahoma, 11px	<None>
	🌐		☐	Greek (Greece)	Tahoma	Tahoma, 11px	<None>

Result

You have now set three runtime languages. A number is automatically assigned to each language in the "Order" column. The enabled runtime languages are transferred with the compiled project to the HMI device.

If the number of languages selected exceeds the number that can be transferred to the HMI device, the table background changes color.

10.7.4 Setting the runtime language order for language switching (RT Uni)

Introduction

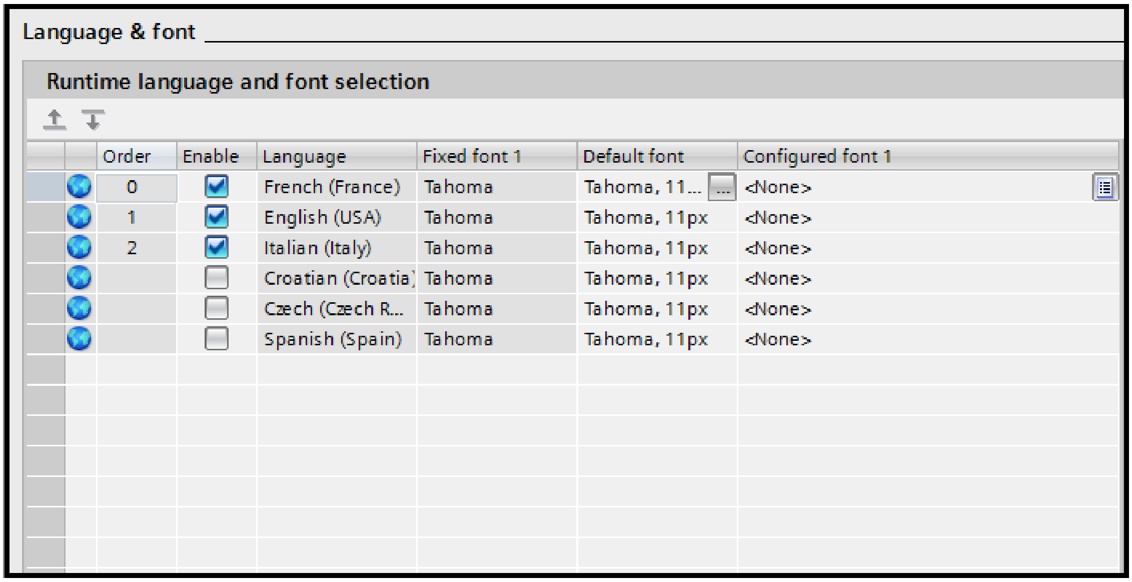
You specify the language order for runtime language switching. The first time runtime starts, the project is displayed in the language with the lowest number in the "Order" column.

Requirements

- Multiple languages have been enabled in the "Project languages" editor.
- The "Language & Font" editor is open and three runtime languages have been set in the following order:
  1. English
  2. Italian
  3. French

Procedure

1. Select the runtime language "English".
2. Click ↧. The runtime language "English" is moved down a place. The number will automatically be changed to "1" in the "Order" column.



Language & font

Runtime language and font selection

		Order	Enable	Language	Fixed font 1	Default font	Configured font 1
	🌐	0	☑	French (France)	Tahoma	Tahoma, 11...	<None>
	🌐	1	☑	English (USA)	Tahoma	Tahoma, 11px	<None>
	🌐	2	☑	Italian (Italy)	Tahoma	Tahoma, 11px	<None>
	🌐		☐	Croatian (Croatia)	Tahoma	Tahoma, 11px	<None>
	🌐		☐	Czech (Czech R...	Tahoma	Tahoma, 11px	<None>
	🌐		☐	Spanish (Spain)	Tahoma	Tahoma, 11px	<None>

Result

You have changed the order of runtime languages. The first time runtime starts, the project will be displayed in the language with the lowest number. If the language is switched, this will happen in numerical order.

10.7.5 Setting the default font for a runtime language (RT Uni)

Introduction

You can specify the font used to display the texts for each runtime language on the device in the "Language & Font" editor. The default font is used in all texts, such as dialog texts, for which you cannot define a specific font.

WinCC offers only fonts supported by the HMI device.

Requirements

- Multiple languages have been enabled in the "Project languages" editor.
- Three runtime languages have been enabled in the "Language & Font" editor.
  1. Chinese
  2. German
  3. French

Procedure

1. Double-click on "Runtime settings" in the project tree.
2. Click on "Language & Font". The table shows the runtime languages and fonts set.
3. Click in the "French" row in the "Default font" column.
4. Select the font to be used by default if a font cannot be selected for a given text.

Result

The project texts for the runtime language "French" are displayed on the device in the selected font.

The default font is also used for the display of dialogs in the operating system of the device. Select a smaller font as default if the full length of the dialog texts or headers is not displayed.

10.7.6 Standardizing font for all languages (RT Uni)

Introduction

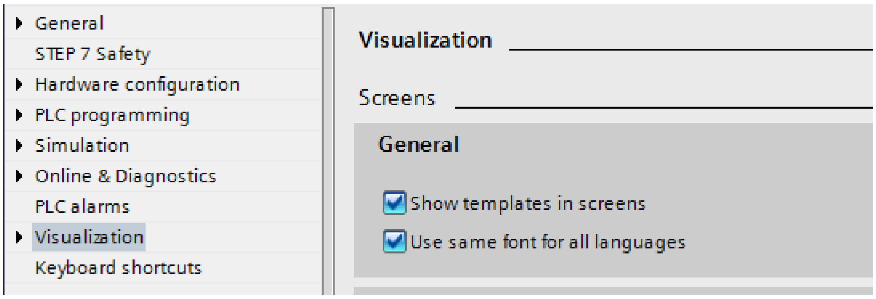
You can standardize the font for all project languages during configuration with the "Use same font for all languages" option.

Requirement

- Multiple languages have been selected in the "Project languages" editor.
- Multiple languages have been selected in the "Language & font" editor.
- The same font is defined for the selected runtime languages under "Configured font".

Procedure

1. In the "Options > Settings > Visualization > General" menu, select the "Use same font for all languages" option.



☑ Show templates in screens

☑ Use same font for all languages

Result

You have enabled the option "Use same font for all languages". If you change the font of an object in one language during configuration, this font will be applied to all active languages.

10.7.7 Specific features of Asian and Eastern languages in runtime (RT Uni)

Introduction

Note the following special considerations for the operation in runtime of projects for Asian languages.

Note

During configuration, only use the Asian fonts that your configuration computer supports.

Memory requirement for Asian character sets

The memory requirement is greater when using Asian languages. Therefore look out for corresponding error messages when compiling the project.

Font size for Asian character sets

Use at least a font size of 10 points to display the text of projects created for Asian languages in runtime. Asian characters will become illegible if smaller font sizes are used. This also applies to the default font in the runtime settings under "Language & font".

Text field length for Asian languages

Make allowances for an appropriate length of the text fields when working on multilingual projects with Asian languages. Field contents may be partially hidden, depending on the font and the font size.

1. Open the "Properties > Appearance" text box in the Inspector window.
2. Under "Fit to size", disable the "Auto-size" option.
3. Verify the proper display in runtime.

Configuring parameter sets (RT Uni) 11

11.1 Basics (RT Uni)

11.1.1 Basics of parameter control (RT Uni)

Introduction

The parameter control is a comprehensive function for the control of parameter sets for configuration engineers, operators and recipe creators. The parameter control brings you the following benefits:

- You can apply the structure of a user data type for one or more parameter set types.
- You can change the structure of one or more parameter set types automatically via a new user data type version.
- You can exchange a large number of parameters manually or automatically between HMI device and control system to set up a machine for a production.
- You can create parameter sets simply and uniformly for products to be manufactured in the works during engineering or during ongoing operation.
- You can transfer parameters by simple means through the structuring of the associated parameters/setpoint values.

Elements of the parameter control

- Parameter set type
  A parameter set type with parameter set type items determines the structure that is used for parameter sets on a machine. You create a parameter set type with parameter set type items on the basis of a released HMI or PLC user data type that has user data type elements.
- Parameter set type item
  Element of a parameter set type that is based on a user data type element. A parameter set type item has the same name and data type as the corresponding user data type element.
- Parameter record
  Set of parameters with concrete values that can be activated on a machine.
- Parameters
  Element of a parameter set that is based on a parameter set type item. A parameter has the same display name and data type as well as the same unit of measure as the corresponding parameter set type item. A parameter has a concrete value that can be activated on a machine.

Tools of the parameter control

- "Parameter set types" editor
  In the "Parameter set types" editor, you create parameter set type items on the basis of an HMI or PLC application data type. In addition, you configure the properties of parameter set types and parameter set type items in the editor.
- Parameter set control
  The parameter set type display is a control with which you can display and manage parameter sets in runtime and exchange them with the control system.

  Note

  The parameters set view is only supported for Unified PC. If you have configured a parameter set view on a Unified Comfort Panel, you must delete the control before compiling.

Parameter set memory

Part of the parameter control is the parameter set memory which is the internal memory of the HMI device for parameter sets. The parameter set memory is located locally on the computer in runtime.

Parameter control in the Engineering System

Perform the following tasks in Engineering System for the parameter control:

- You create a parameter set type with elements to specify the structure of parameter sets.
- You change the parameter set type with elements to change the structure of parameter sets.
- You assign a tag of the data type user data type to a parameter set type to transfer parameter sets between HMI device and control system in runtime.
- You create local scripts in image objects or tasks to transfer parameter sets in runtime between the HMI device and control system.
- You assign control tags to a parameter set type to automatically transfer or delete parameter sets between HMI device and control system in runtime.
- You configure a parameter set control to display parameter sets, manage them and exchange them with the control system through the Control in runtime.

Parameter control in runtime

The following options are available in runtime with the parameter control:

- You create, change and delete parameter sets in a parameter set control to manage parameter sets for different productions.
- You export parameter sets from the parameter set memory into a "*.tsv" file to edit them in a text editor.

  Note

  A "*.tsv" file is a text file that uses the tabulator as a list separator.

- You import parameter sets from a "*.tsv" file into the parameter set memory.
- You transfer parameter sets manually or automatically to the control system to set up machines with values for different productions.
- You read parameter sets manually or automatically from the control system to call up currently used values of production machines for later use.
- You automatically delete parameter sets from the parameter set memory.

See also

Configuring parameter sets (Page 788)

Using parameter sets in runtime (Page 807)

11.1.2 "Parameter set types" editor (RT Uni)

Introduction

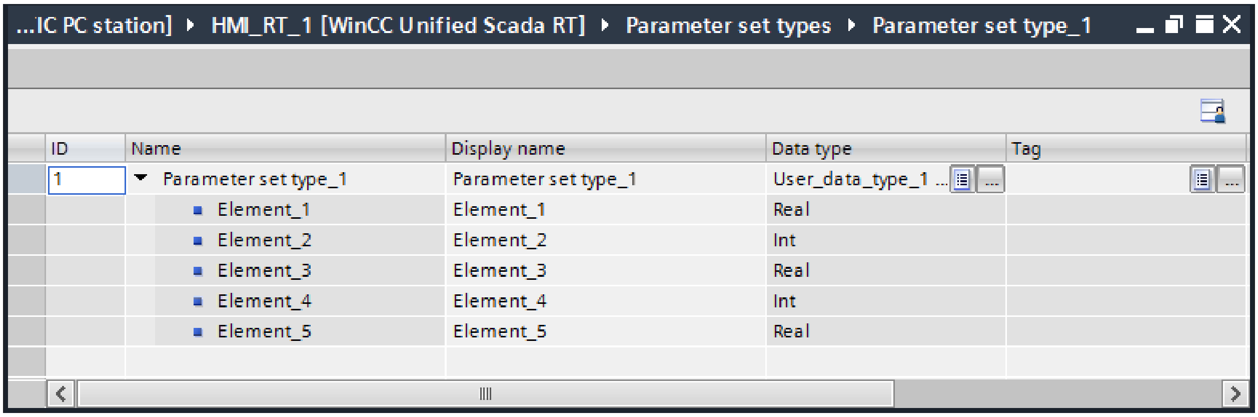
In the "Parameter set types" editor, you create a parameter set type with elements on the basis of an HMI or PLC application data type. In addition, you configure the properties of parameter set types and parameter set type items in the editor.

Note

Alternatively create the parameter set type items on the basis of a user data type in the Inspector window. In addition, you configure the properties of parameter set types and parameter set type items also alternatively in the Inspector window.

Structure of the "Parameter set types" editor

The "Parameter set types" editor is a tabular editor. The editor always contains only a single parameter set type and, if applicable, its elements. To view hidden columns, activate the column titles using the shortcut menu.


...IC PC station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Parameter set type_1

ID	Name	Display name	Data type	Tag
1	Parameter set type_1	Parameter set type_1	User_data_type_1 ...	
	Element_1	Element_1	Real	
	Element_2	Element_2	Int	
	Element_3	Element_3	Real	
	Element_4	Element_4	Int	
	Element_5	Element_5	Real	

Properties of parameter set types

In the "Parameter set types" editor, you can configure the following properties:

Property	Description
ID	Number of a parameter set type. The ID uniquely identifies a parameter set type within the HMI device. The ID appears in the parameter set control in runtime.
Name	Name of a parameter set type. The name uniquely identifies a parameter set type within the HMI device.
Display name	Display name of a parameter set type. The display name appears in the parameter set control in runtime. You can configure display names in multiple languages. The property is optional. If you do not set a display name, the value from the "Name" property appears in the parameter set control in runtime.
Data type	Enabled HMI or PLC user data type with which you define the structure of a parameter set type.
Tag	External HMI tag of the data type HMI or PLC user data type. In runtime you transfer parameter sets between HMI device and control system via the HMI tag.
Parameter ID	Control tag with numerical data type. The control tag is used to define an ID of a parameter set which is the target of one of the following control jobs in runtime: ● Control job with job ID "6": Reading a parameter set from the PLC and storing it in the parameter set memory. ● Control job with job ID "7": Loading a parameter set from the parameter set memory and writing it to the PLC. ● Control job with job ID "8": Deleting a parameter set from the parameter set memory. The property is optional.
Job ID	Control tag with numerical data type. The control tag is used to define a control job which is applied to a parameter set in runtime. In runtime, you can apply the following control jobs to a parameter set: ● Control job with job ID "6": Reading a parameter set from the PLC and storing it in the parameter set memory. ● Control job with job ID "7": Loading a parameter set from the parameter set memory and writing it to the PLC. ● Control job with job ID "8": Deleting a parameter set from the parameter set memory. The property is optional.
Author	Author of a parameter set type. The property is optional. By default, the property is pre-assigned with the logged-on Windows user. By default, the column is hidden.
Version	Version of a parameter set type. The property is optional. By default, the property is pre-assigned with the date and time of creation of the parameter set type. By default, the column is hidden.
Path	System standard path in runtime, in which the parameter set memory is located. The path refers to the path of the runtime project. The property is write-protected. By default, the column is hidden.

Properties of parameter set type items

In the "Parameter set types" editor, you can configure the following properties for parameter set type items:

Property	Description
Name	Name of a parameter set type item. The name uniquely identifies a parameter set type item. The name is write-protected and identical to the name of the corresponding user data type element.
Display name	Display name of a parameter set type item and a corresponding parameter in a parameter set. The display name appears in the table of the parameter set control in runtime. You can configure display names in multiple languages. The property is pre-assigned with the value from the "Name" property.
Data type	Data type of a parameter set type item and a corresponding parameter in a parameter set. The name is write-protected and identical to the data type of the corresponding user data type element.
Start value	Start value of a parameter set type item. The start value is used to pre-assign a corresponding parameter in a newly created parameter set in runtime. If you have set a start value in a user data type element with numerical data type, bit sequence data type or string data type, the corresponding parameter set type item receives this start value. If you have not set a start value in a user data type element with numerical data type or bit sequence data type, the corresponding parameter set type item receives "0" as the start value. If you have not set a start value in a user data type element with string data type, the corresponding parameter set type item does not receive a start value. A parameter set type item which is based on a user data type element with date/time data type receives the creation time as start value by default. The property is optional for parameter set type items with string data types.
Minimum value	Minimum permissible value of a parameter set type item and a corresponding parameter in a parameter set. If you have set a minimum value in a user data type element with numerical data type, the corresponding parameter set type item receives this minimum value. Otherwise, the corresponding parameter set type item does not receive a minimum value. The minimum value of a parameter set type item may not be below the minimum value of the data type of the parameter set type item. In addition, the minimum value of a parameter set type item may not be below the minimum value of the corresponding user type element. The property is optional and is disabled for parameter set type items with string data types and bit sequence data types.

Property	Description
Maximum value	Maximum permissible value of a parameter set type item and a corresponding parameter in a parameter set. If you have set a maximum value in a user data type element with numerical data type, the corresponding parameter set type item receives this maximum value. Otherwise, the corresponding parameter set type item does not receive a maximum value. The maximum value of a parameter set type item may not be above the maximum value of the data type of the parameter set type item. In addition, the maximum value of a parameter set type item may not be above the maximum value of the corresponding user data type element. The property is optional and is disabled for parameter set type items with string data types and bit sequence data types.
Value required	If the check box of the property "Value required" is selected in a parameter set type item, a corresponding parameter must have a value in a parameter set. Otherwise, a corresponding parameter must have no value in a parameter set. The check box of the property is selected by default for parameter set type items with numerical data types. In this case, you cannot clear the check box. The check box of the property is cleared by default for parameter set type items with string data types. In this case, however, you can select the check box if required.
Unit of measure	Unit of measure of a parameter set type item and a corresponding parameter in a parameter set. The unit of measure appears in the table of the parameter set control in runtime. The property is optional.

See also

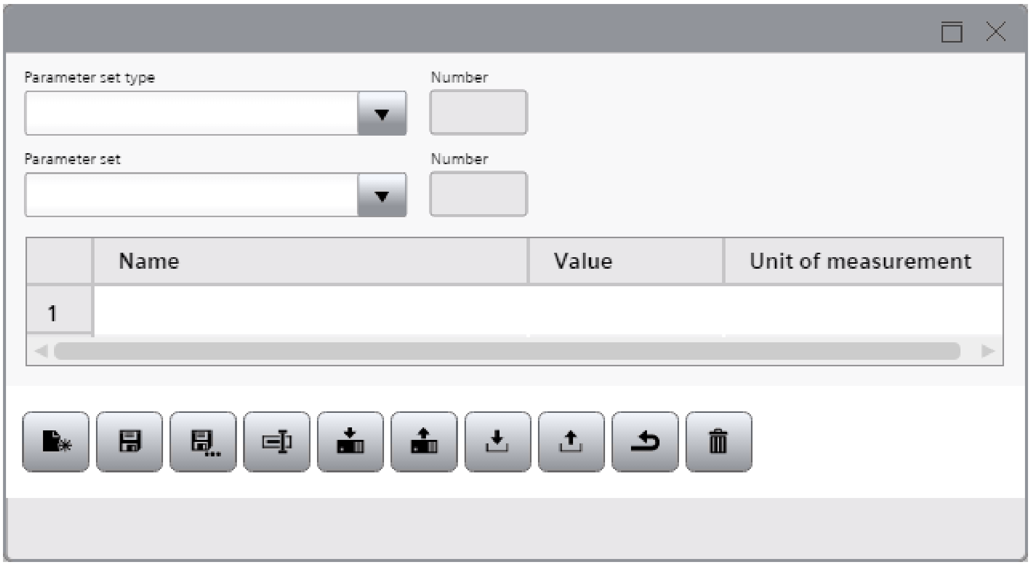
Creating a parameter set type with elements via an HMI user data type (Page 788)

Creating a parameter set type with elements via a PLC user data type (Page 791)

11.1.3 Parameter set control (RT Uni)

Use

The parameter set control is used to display parameter sets in runtime, to manage them and to exchange them with the control system.



Note

The "Parameter set control" object is supported with version V16 exclusively for Unified PC. If the user uses the object under Unified Comfort Panel, an error message of the compiler is returned. Existing projects under Unified Comfort Panel that have configured the object, must delete the object before compiling to version V16.

Layout

You change the settings for the position, geometry, style, color, and font of the object in the Inspector window. You can adapt the following properties in particular:

- Display selection list: If you clear the check box of the property, the Parameter set type field and the Parameter set type ID field with the associated labels are hidden.

  Note

  If you hide the two fields and do not select a parameter set type under "Properties > Fixed parameter set type", the Parameter set type field is disabled in runtime. In addition, no parameter set ID is displayed in the Parameter set ID field in runtime.

- "Parameter view": Defines the display of the parameter table in the control.

- "Toolbar": Defines the operator controls of the parameter set control.
- "Status bar": Specifies the display of the status line.

Note

The "Status Text" element is the only status line element of the parameter set display. Status messages are displayed in this element in runtime.

Using a parameter set type.

If you only want to use a particular parameter set type with its parameter sets in runtime, select the desired parameter set type under "Properties > Fixed parameter set type".

Operator controls

You define the operator controls for the parameter set control in runtime, and their operator authorizations, under "Properties > Toolbar" in the Inspector window. By default, all buttons are displayed in the toolbar. To hide specific buttons, deactivate the "Visibility" property in the settings of the corresponding button.

The following operator controls are available for the parameter set control:

	Button	Function
	Create	Creates a new parameter set.
	Save	Saves a parameter set.
	Save as	Saves an existing parameter set under a new name and new ID.
	Rename	Renames the selected parameter set.
	Write to PLC	Writes the values of the selected parameter set to the PLC.
	Read from PLC	Writes the values of the selected parameter set from the PLC.
	Import	Imports parameter sets from a "*.tsv" file.
	Export	Exports parameter sets to a "*.tsv" file.

	Button	Function
↩	Cancel	Cancels the process.
🗑	Delete	Deletes the selected parameter set.

Note

A "*.tsv" file is a text file that uses the tabulator as a list separator.

Enabling/disabling operator controls

In "Properties > Editing mode", configure the activation status of the toolbar buttons "Create", "Save", "Save as", "Rename" and "Delete". These toolbar buttons are used to edit parameter sets.

You can select between the following settings:

- "None": Deactivates all buttons.
- "Update": Activates the "Save" and "Rename" buttons.
- "Create": Activates the "Create" and "Save as" buttons
- "Delete": Activates the "Delete" button

Configuring a status bar

1. Configure the general properties of the status bar such as the font or the background color under "Properties > Status bar".
2. To adjust the size of the "Status Text" element, activate the "Customized" property under "Properties > Status bar > Elements > Control bar label [0]".
3. You can enter a pixel value for the width and height.

Note

Status messages are displayed in runtime in the "Status text" element.

Configuring a time zone

Under "Properties > Time zone", you set the desired time zone in which you enter a numerical value.

The numerical value stands for a time zone, for example:

- "-1" stands for UTC-1h (Central European Time, standard time)
- "1" stands for UTC-12h (International Date Line West)
- "2" stands for UTC-11h (Hawaii)

See also

Configuring the parameter set view (Page 805)

11.2 Configuring parameter sets (RT Uni)

11.2.1 Creating a parameter set type with elements via an HMI user data type (RT Uni)

Introduction

You have created an HMI user data type with elements. You can assign the user data type to one or more parameter set types and this way apply the elements of the user data type for the parameter set type or types. A parameter set type with elements determines the structure that is used for parameter sets on a machine. Since you do not have to create the elements of the user data type again separately for the parameter set type or types, engineering efficiency and reusability.

Note

Observe the following restrictions when creating the HMI user data type:

- The user data type may have a maximum of 1000 elements.
- No user data type element may have the data type TEXTREF or RAW.

Note

You can also create a parameter set type with elements via a PLC user data type that is created in a SIMATIC S7-1500 control system.

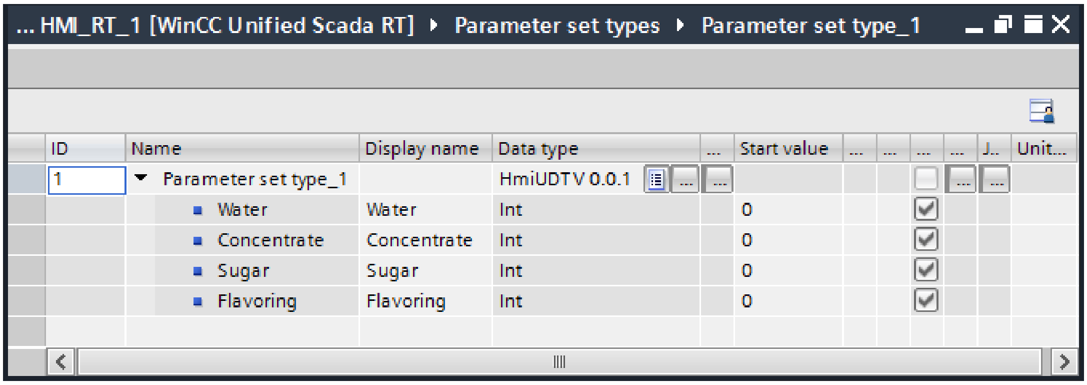
Requirement

- The HMI device WinCC Unified Scada RT has been created.
- An HMI user data type with user data type elements has been created for the HMI device.
- The HMI user data type is enabled.

Creating a parameter set type with elements via an HMI user data type

Proceed as follows to create a parameter set type with elements via an HMI user data type:

1. Open the "Parameter set types" folder in the project tree.
2. Double-click "Add new parameter set type".
   A parameter set type with unique standard name and unique ID is created. The "Parameter set types" editor opens.
3. Select the released HMI user data type in the "Data type" column in the "Parameter set types" editor.
   In the editor, parameter set elements which are based on the user data type elements are added to the parameter set type.



... HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Parameter set type_1

ID	Name	Display name	Data type	...	Start value	...	...	...	...	J..	Unit...
1	▼ Parameter set type_1		HmiUDT V 0.0.1	...				☐	...	...	
	■ Water	Water	Int		0			☑			
	■ Concentrate	Concentrate	Int		0			☑			
	■ Sugar	Sugar	Int		0			☑			
	■ Flavoring	Flavoring	Int		0			☑			

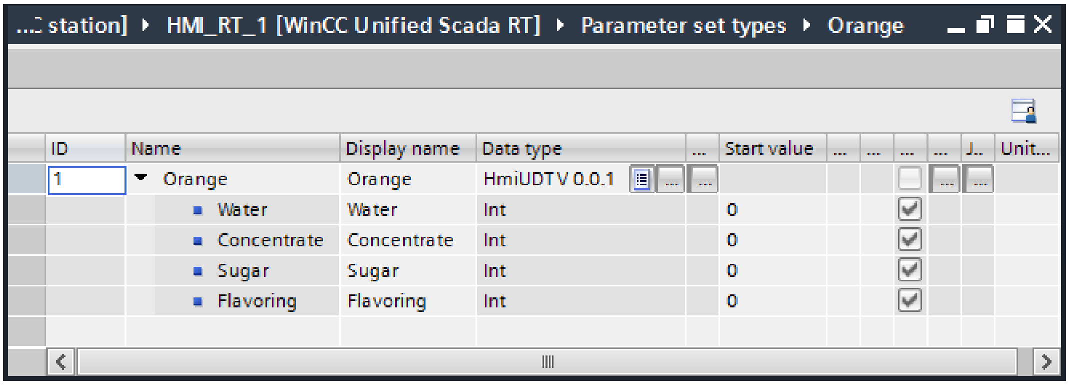
Note

The parameter set type items have the same name and data type as the user data type elements. The name and data type of the parameter set type items are write-protected in the editor.

Configuring a parameter set type

To configure the created parameter set type, proceed as follows:

1. Enter a meaningful name for the parameter set type in the "Name" column in the "Parameter set types" editor
   The name uniquely identifies the parameter set type within the HMI device.
2. Enter a language-dependent name for the parameter set type in the "Display name" column.
   The display name appears in the parameter set control in runtime.
3. If required, enter an own number for the parameter set type in the "ID" column.
   The ID uniquely identifies the parameter set type within the HMI device. The ID appears in the parameter set control in runtime.



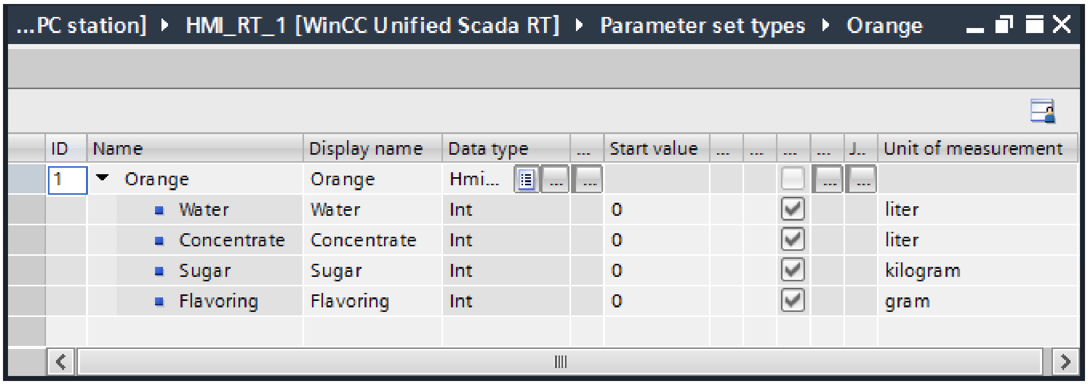
...] station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Orange

ID	Name	Display name	Data type	...	Start value	...	...	...	...	J..	Unit...
1	▾ Orange	Orange	HmiUDT V 0.0.1					□	...	...	
	▪ Water	Water	Int		0			☑			
	▪ Concentrate	Concentrate	Int		0			☑			
	▪ Sugar	Sugar	Int		0			☑			
	▪ Flavoring	Flavoring	Int		0			☑			

Configuring parameter set type items

To configure the created parameter set type items, proceed as follows:

1. In the "Display name" column of the "Parameter set types" editor configure a language-dependent name for the parameter set type items and corresponding parameters in a parameter set.
   The display name appears in the table of the parameter set control in runtime.
2. Configure a unit of measure In the "Unit of measure" column for parameter set type items and corresponding parameters in a parameter set.
   The unit of measure appears in the table of the parameter set control in runtime.
3. Configure a start value in the "Start value" column for parameter set type items.
   The start value is used to pre-assign the corresponding parameters in a newly created parameter set in runtime.



...PC station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Orange

ID	Name	Display name	Data type	...	Start value	...	...	...	...	J..	Unit of measurement
1	▾ Orange	Orange	Hmi...					□	...	...	
	▪ Water	Water	Int		0			☑			liter
	▪ Concentrate	Concentrate	Int		0			☑			liter
	▪ Sugar	Sugar	Int		0			☑			kilogram
	▪ Flavoring	Flavoring	Int		0			☑			gram

Result

You have created a parameter set type with elements via an HMI user data type.

See also

Configuring user data types (Page 166)

"Parameter set types" editor (Page 781)

Changing a parameter set type with elements (Page 795)

Configuring the parameter set view (Page 805)

Assigning a tag of the data type HMI user data type to a parameter set type (Page 798)

11.2.2 Creating a parameter set type with elements via a PLC user data type (RT Uni)

Introduction

You have created a PLC user data type with elements. You can assign the user data type to one or more parameter set types and this way apply the elements of the user data type for the parameter set type or types. A parameter set type with elements determines the structure that is used for parameter sets on a machine. Since you do not have to create the elements of the user data type again separately for the parameter set type or types, engineering efficiency and reusability.

Note

Observe the following restrictions when creating the PLC user data type:

- The user data type may have a maximum of 1000 elements.
- No user data type element may have the data type ARRAY or STRUCT.

Note

You can also create a parameter set type with elements via an HMI user data type for which the communications driver SIMATIC S7-300/400 or SIMATIC S7-1500 is set.

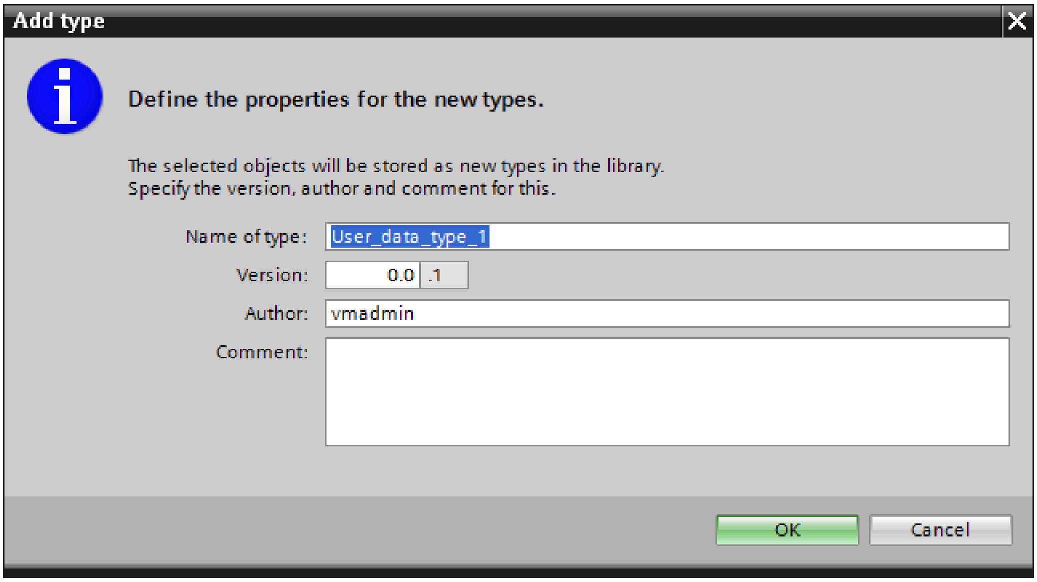
Requirement

- The HMI device WinCC Unified Scada RT has been created.
- A PLC user data type with user data type elements is created in a SIMATIC S7-1500 PLC.
- The "Libraries" task card is open.
- The project library is open.

Adding a PLC user data type to the project library

To add a PLC user data type to the project library, follow these steps:

1. Open the PLC's folder in the project navigation.
2. Open the folder "PLC data types" in the folder of the controller.
   The created PLC user data type is displayed in the "PLC data types" folder.
3. Move the PLC user data type to the "Types" folder in the project library.
   The "Add type" dialog opens.



Add type

Define the properties for the new types.

The selected objects will be stored as new types in the library.
Specify the version, author and comment for this.

Name of type: User_data_type_1

Version: 0.0 .1

Author: vmadmin

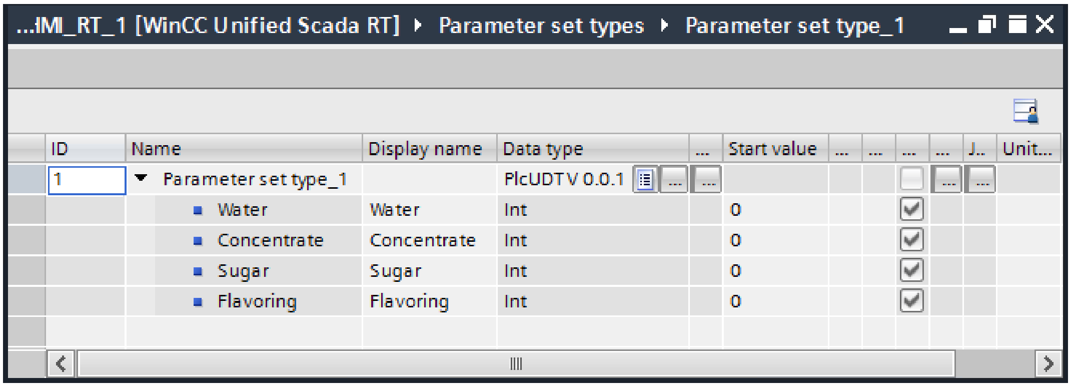
Comment:

4. Make the following settings in the dialog:
   - Enter a unique name for the PLC user data type under "Type name".
   - Enter a valid version number for the PLC user data type under "Version".
   - Enter the author responsible for the PLC user data type under "Author".
   - Enter a comment for the PLC user data type under "Comment".
5. Click the "OK" button.
   The PLC user data type is added with a released version to the "Types" folder in the project library.

Creating a parameter set type with elements via a PLC user data type

Proceed as follows to create a user data type with elements via a PLC user data type:

1. Open the "Parameter set types" folder in the project tree.
2. Double-click "Add new parameter set type".
   A parameter set type with unique standard name and unique ID is created. The "Parameter set types" editor opens.
3. Select the released PLC user data type in the "Data type" column in the "Parameter set types" editor.
   In the editor, parameter set elements which are based on the user data type elements are added to the parameter set type.



...IMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Parameter set type_1

ID	Name	Display name	Data type	...	Start value	...	...	...	...	J..	Unit...
1	▼ Parameter set type_1		PlcUDT V 0.0.1 ...	...				□	...	...	
	■ Water	Water	Int		0			☑			
	■ Concentrate	Concentrate	Int		0			☑			
	■ Sugar	Sugar	Int		0			☑			
	■ Flavoring	Flavoring	Int		0			☑			

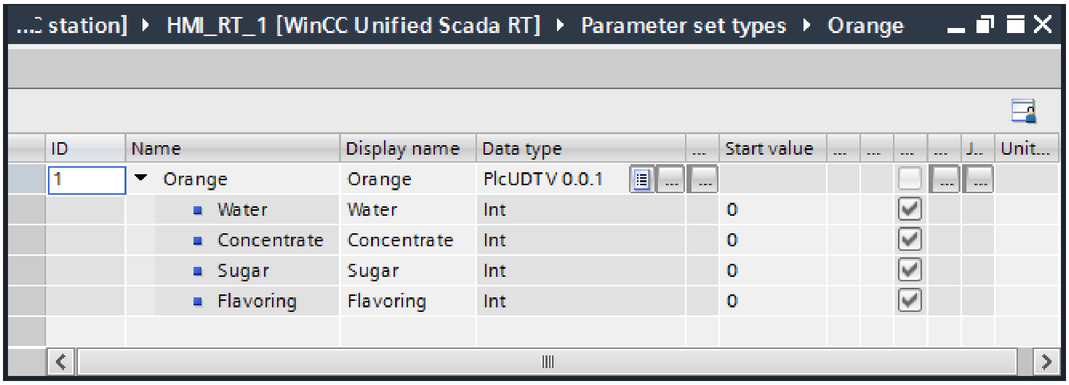
Note

The parameter set type items have the same name and data type as the user data type elements. The name and data type of the parameter set type items are write-protected in the editor.

Configuring a parameter set type

To configure the created parameter set type, proceed as follows:

1. Enter a meaningful name for the parameter set type in the "Name" column in the "Parameter set types" editor
   The name uniquely identifies the parameter set type within the HMI device.
2. Enter a language-dependent name for the parameter set type in the "Display name" column.
   The display name appears in the parameter set control in runtime.
3. If required, enter an own number for the parameter set type in the "ID" column.
   The ID uniquely identifies the parameter set type within the HMI device. The ID appears in the parameter set control in runtime.



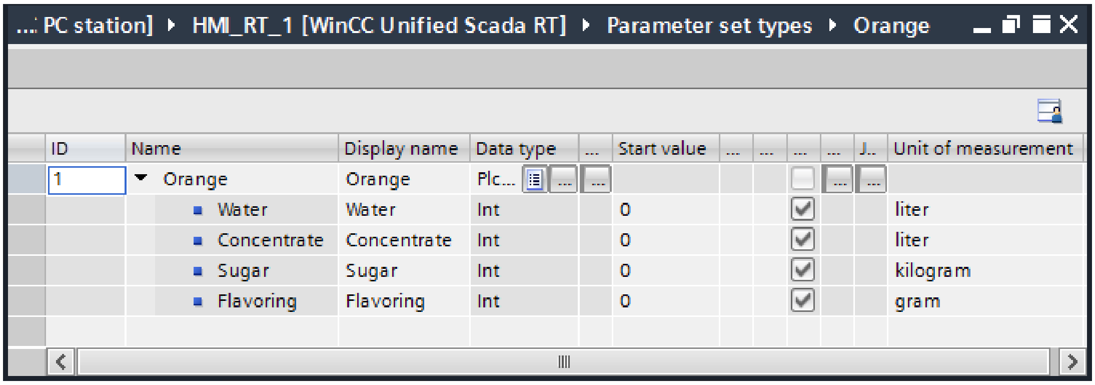
...: station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Orange

ID	Name	Display name	Data type	...	Start value	...	...	...	...	J..	Unit...
1	▾ Orange	Orange	PlcUDT V 0.0.1					☐			
	Water	Water	Int		0			☑			
	Concentrate	Concentrate	Int		0			☑			
	Sugar	Sugar	Int		0			☑			
	Flavoring	Flavoring	Int		0			☑			

Configuring parameter set type items

To configure the created parameter set type items, proceed as follows:

1. In the "Display name" column of the "Parameter set types" editor configure a language-dependent name for the parameter set type items and corresponding parameters in a parameter set.
   The display name appears in the table of the parameter set control in runtime.
2. Configure a unit of measure In the "Unit of measure" column for parameter set type items and corresponding parameters in a parameter set.
   The unit of measure appears in the table of the parameter set control in runtime.
3. Configure a start value in the "Start value" column for parameter set type items.
   The start value is used to pre-assign the corresponding parameters in a newly created parameter set in runtime.



...: PC station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Orange

ID	Name	Display name	Data type	...	Start value	...	...	...	...	J..	Unit of measurement
1	▾ Orange	Orange	Plc...					☐			
	Water	Water	Int		0			☑			liter
	Concentrate	Concentrate	Int		0			☑			liter
	Sugar	Sugar	Int		0			☑			kilogram
	Flavoring	Flavoring	Int		0			☑			gram

Result

You have created a parameter set type with elements via a PLC user data type.

See also

Configuring user data types (Page 166)

"Parameter set types" editor (Page 781)

Changing a parameter set type with elements (Page 795)

Configuring the parameter set view (Page 805)

Assigning a tag of the data type "PLC user data type" to a parameter set type (Page 800)

11.2.3 Changing a parameter set type with elements (RT Uni)

Introduction

You have created one or more parameter set types with elements via a user data type. A parameter set type with elements determines the structure that is used for parameter sets on a machine. You have the following options to change parameter set types with elements:

- You can change one or more parameter set types automatically via a new version of the user data type.
- You can change a parameter set type manually via a new version of the user data type.
- You can change a parameter set type via another user data type.

Requirement

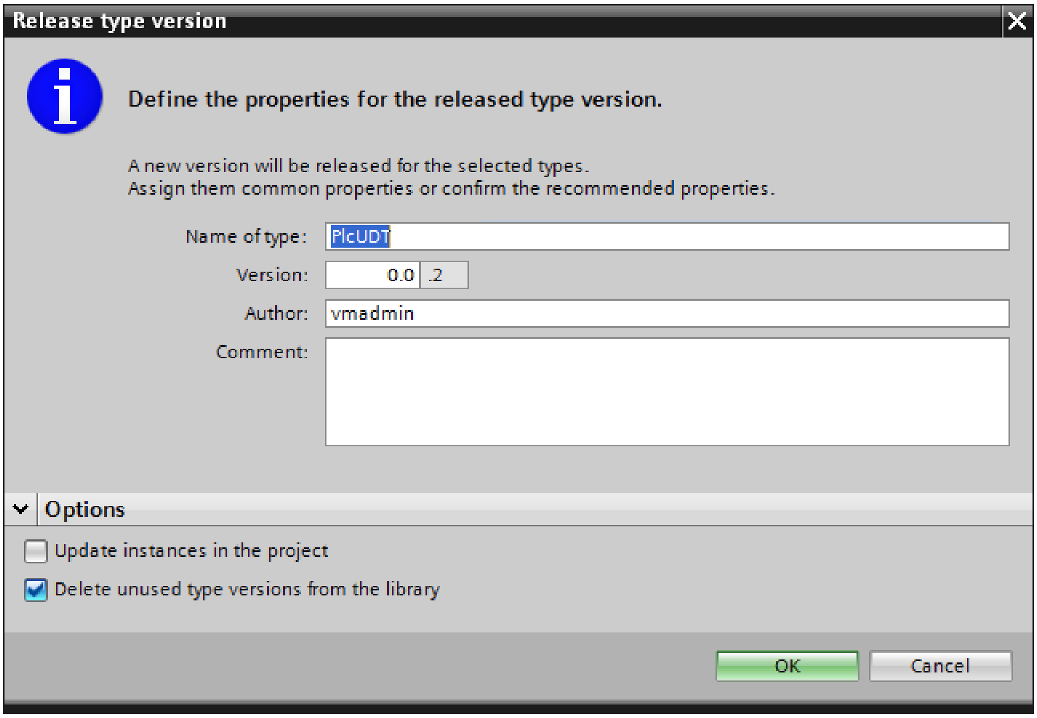
- One or more parameter set types with elements are created on the basis of an HMI or PLC user data type.

Changing parameter set types automatically via a new version of the user data type

To change parameter set types with elements automatically via a new version of the user data type, follow these steps:

1. Select the released user data type in the project library.
2. Select "Edit type" in the shortcut menu.
   In the case of an HMI user data type the "HMI user data types" editor is opened and a new user data type version with the "In process" status is generated in the project library. In the case of a PLC user data type the "Edit type" dialog is opened.
3. In the case of a PLC user data type click "OK" in the "Edit type" dialog.
   The "PLC data type" editor opens. A new user data type version with the "Testing" status is generated in the project library.
4. Change the user data type in the case of an HMI user data type in the "HMI user data types" editor and in the case of a PLC user data type in the "PLC data types" editor.

5. Select the new user data type version in the project library.
6. Select "Release version" in the shortcut menu.
   The "Release type version" dialog box opens.



Release type version

Define the properties for the released type version.

A new version will be released for the selected types.
Assign them common properties or confirm the recommended properties.

Name of type: PlcUDT
Version: 0.0 .2
Author: vmadmin
Comment:

Options

☐ Update instances in the project

☑ Delete unused type versions from the library

7. If necessary, change the properties of the version:
   - Enter a unique name for the user data type in the "Type name" field.
   - Enter a valid version number for the version to be released in the "Version" field.
   - Under "Author" enter the editor of the version to be released.
   - Under "Comment" enter a comment on the version to be released.
8. If you want to revise the version management of the user data type, enable the "Delete unused type versions from the library".

9. To automatically change the parameter set type or types via the new user data type version, activate the option "Update instances in the project".
10. Click the "OK" button.
    The parameter set type or types are changed in the following way via the user data type version:
    - An element that you have not changed in the user data type, also remains in the parameter set type with all its settings.
    - An element that you have added new to the user data type is also added to the parameter set type.
    - The element that you have deleted from the user data type is also deleted from the parameter set type.
    - If you have changed the name of an element in the user data type, the parameter set type of the element's name is changed as well. All other properties of the element remain unchanged such as the start value or the display name.
    - If you have changed the data type of an element in the user data type, the parameter set type of the element's data type is changed as well. Other properties of the elements remain unchanged.

      Note

      An element whose data type you have changed is treated like a new element in runtime. Consequently the values of the element are deleted in the existing parameter sets and the start values are assigned as default to the element.
    - If you have changed a numerical data type of an element into a string data type in the user data type, the minimum and maximum values previously specified of the element are also removed.

Changing the parameter set type manually via a new version of the user data type

Proceed as follows to change a parameter set type with elements manually via a new version of the user data type:

1. Execute Steps 1 to 8 and 10 of the description above.
2. Open the "Parameter set types" folder in the project navigation.
3. Double-click a created parameter set type.
   The "Parameter set types" editor opens.
4. Select the new user data type version in the "Data type" column in the "Parameter set types" editor.
   The parameter set type is changed via the new user data type version as described in Step 10 of the above description.

Changing parameter set type via another user data type

Proceed as follows to change a parameter set type with elements manually via another user data type:

1. Open the "Parameter set types" folder in the project navigation.
2. Double-click a created parameter set type.
   The "Parameter set types" editor opens.
3. Select another released user data type in the "Data type" column in the "Parameter set types" editor.
   The structure of the parameter set type is created completely new in accordance with the structure of the other user data types.

See also

Creating a parameter set type with elements via an HMI user data type (Page 788)

Creating a parameter set type with elements via a PLC user data type (Page 791)

11.2.4 Assigning a tag of the data type HMI user data type to a parameter set type (RT Uni)

Introduction

To exchange parameter sets between the HMI device and control system in runtime, assign an external HMI tag to a parameter set type created via an HMI user data type in the Engineering System. To this purpose the HMI tag uses the HMI user data type as the data type with which you created the parameter set type.

Note

You can also assign an external HMI tag to a parameter set type that was created via a PLC user data type. In doing so the HMI tag uses the PLC user data type as the data type with which you created the parameter set type.

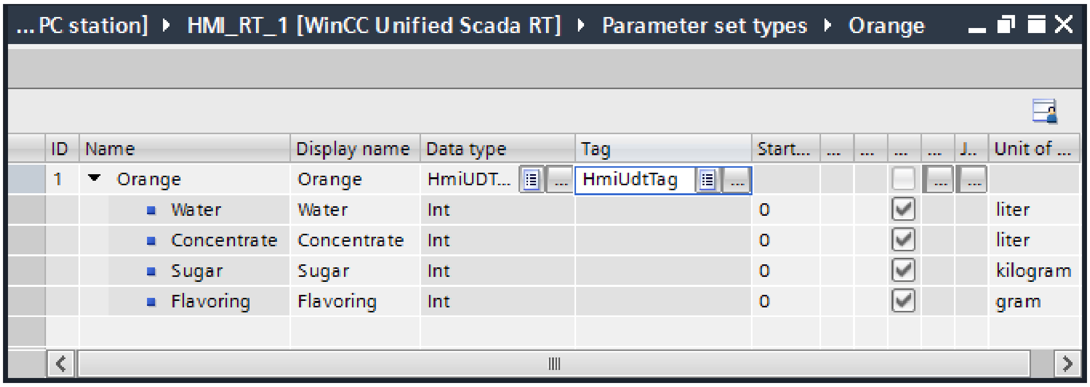
Requirement

- An HMI user data type has been created in the HMI device WinCC Unified Scada RT.
- The communication driver SIMATIC S7-300/400 or SIMATIC S7-1500 is set for the HMI user data type.
- User data type elements are added to the HMI user data type.
- The HMI user data type is enabled.
- A parameter set type with elements is created on the basis of the HMI user data type.
- An external HMI tag is created that uses the HMI user data type as the data type.

Procedure

To assign an external HMI tag of the data type HMI user data type to a parameter set type, proceed as follows:

1. Open the "Parameter set types" folder in the project navigation.
2. Double-click the created parameter set type.
   The "Parameter set types" editor opens. The Inspector window opens.
3. Select the created HMI tag of the data type "HMI user data type" in the column "Tag" in the "Parameter set types" editor.



... PC station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Orange

ID	Name	Display name	Data type	Tag	Start...	...	...	...	...	J..	Unit of ...
1	▾ Orange	Orange	HmiUDT...	HmiUdtTag				☐	...	...	
	Water	Water	Int		0			☑			liter
	Concentrate	Concentrate	Int		0			☑			liter
	Sugar	Sugar	Int		0			☑			kilogram
	Flavoring	Flavoring	Int		0			☑			gram

Result

You have assigned an external HMI tag of the data type "HMI user data type" to a parameter set type.

See also

Creating a parameter set type with elements via an HMI user data type (Page 788)

Creating tags with a user data type data type (Page 170)

Assigning a tag of the data type "PLC user data type" to a parameter set type (Page 800)

Transferring and deleting parameter sets automatically (Page 801)

Transferring parameter sets (Page 816)

11.2.5 Assigning a tag of the data type "PLC user data type" to a parameter set type (RT Uni)

Introduction

To exchange parameter sets between an HMI device and control system in runtime, assign an external HMI tag to a parameter set type created via a PLC user data type in the Engineering System. In doing so the HMI tag uses the PLC user data type as the data type with which you created the parameter set type.

Note

You can also assign an external HMI tag to a parameter set type which was created via a HMI user data type. To this purpose the HMI tag uses the HMI user data type as the data type with which you created the parameter set type.

Requirement

- The HMI device WinCC Unified Scada RT has been created.
- A PLC user data type with user data type elements is created in a SIMATIC S7-1500 control system.
- A parameter set type with elements is created on the basis of the PLC user data type.
- An external HMI tag is created that uses the PLC user data type as the data type.

Note

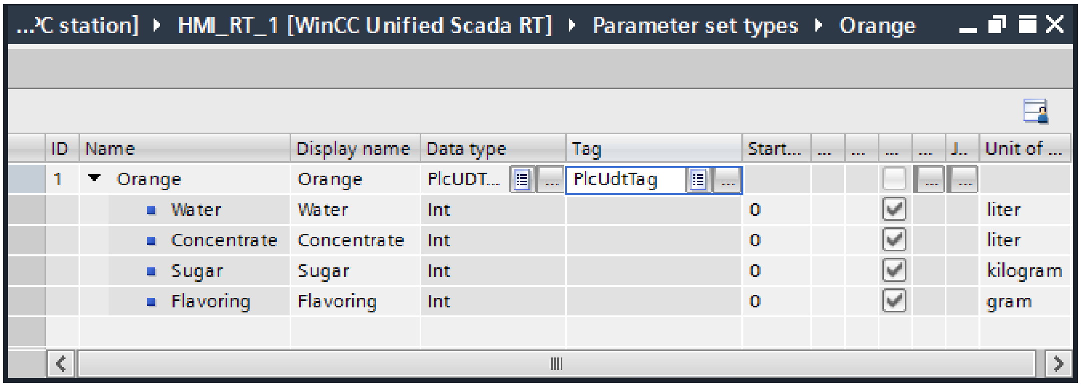
You have the following possibilities to create an external HMI tag of the data type "PLC user data type":

- Assign a PLC tag to an HMI tag, whereby the PLC tag is based on a PLC user data type.
- Assign a PLC data block that is based on a PLC user data type to an HMI tag.
- Assign a PLC data block element that is based on a PLC user data type to an HMI tag.

Procedure

To assign an external HMI tag of the data type "PLC user data type" to a parameter set, follow these steps:

1. Open the "Parameter set types" folder in the project navigation.
2. Double-click the created parameter set type.
   The "Parameter set types" editor opens. The Inspector window opens.
3. Select the created HMI tag of the data type "PLC user data type" in the "Tag" column in the "Parameter set types" editor.



...C station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Orange

ID	Name	Display name	Data type	Tag	Start...	...	...	...	...	J..	Unit of ...
1	▾ Orange	Orange	PlcUDT...	PlcUdtTag				□	...	...	
	■ Water	Water	Int		0			☑			liter
	■ Concentrate	Concentrate	Int		0			☑			liter
	■ Sugar	Sugar	Int		0			☑			kilogram
	■ Flavoring	Flavoring	Int		0			☑			gram

Result

You have assigned an external HMI tag of the data type "PLC user data type" to a parameter set type.

See also

Creating a parameter set type with elements via a PLC user data type (Page 791)

External tags (Page 140)

Creating external tags (Page 152)

Assigning a tag of the data type HMI user data type to a parameter set type (Page 798)

Transferring and deleting parameter sets automatically (Page 801)

Transferring parameter sets (Page 816)

11.2.6 Transferring and deleting parameter sets automatically (RT Uni)

Introduction

In the Engineering System you assign control tags to a parameter set type with elements. The control tags serve to automatically transfer or delete parameter sets between an HMI device and control system in runtime. In the process either the control program or the HMI device controls the automatic transfer or deleting via control requests.

Requirement

- A parameter set type with elements is created on the basis of an HMI or PLC user data type.
- An external HMI tag that uses the HMI or PLC user data type as the data type is assigned to the parameter set type.
- 2 control tags with numerical data type by the name of "ParameterSetIDTag" and "JobIDTag" are created.

Note

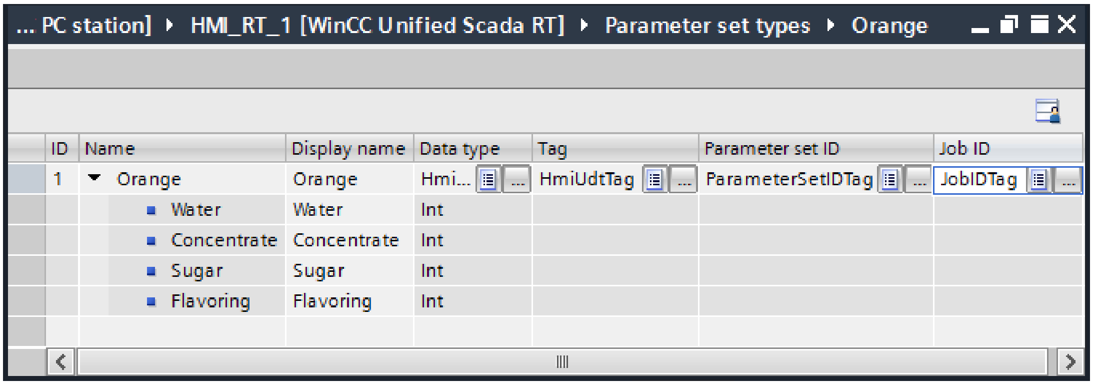
If you want to automatically transfer or delete parameter sets by means of the control program, create 2 external control tags. If you want to automatically transfer or delete parameter sets by means of the HMI device, however, create 2 internal control tags.

- The "Toolbox" task card is open.

Assigning control tags to a parameter set type with elements

Proceed as follows to assign control tags to a parameter set type with elements in the Engineering System:

1. Open the "Parameter set types" folder in the project tree.
2. Double-click on the created parameter set type.
   The "Parameter set types" editor opens. The Inspector window opens.
3. Select the control tag "ParameterSetIDTag" in the "Parameter set ID" column in the "Parameter set types" editor.
4. Select the "JobIDTag" control tag in the "Job ID" column.



... PC station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Parameter set types ▸ Orange

ID	Name	Display name	Data type	Tag	Parameter set ID	Job ID
1	Orange	Orange	Hmi...	HmiUdtTag	ParameterSetIDTag	JobIDTag
	Water	Water	Int			
	Concentrate	Concentrate	Int			
	Sugar	Sugar	Int			
	Flavoring	Flavoring	Int			

Note

Please observe each of the following rules to not obtain any errors when compiling the project:

- If you set a control tag in the "Parameter set ID" column of the "Parameter set types" editor, you also set a control tag in the "Job ID" column.
- Do not set the same control tag in the columns "Parameter set ID" and "Job ID" in the "Parameter set types" editor.
- If you assign a control tag to a parameter set type in the "Parameter set types" editor, do not assign the same control tag to another parameter set type.

Reading a parameter set from the control system

Proceed as follows to automatically read a parameter set from the control system and store it in the parameter set memory in runtime:

1. Automatically set the control tag "ParameterSetIDTag" to an ID of an existing parameter set.
2. Automatically set the control tag "JobIDTag" to the control job ID "6".
   The control job is executed. In the case of success the control tag "JobIDTag" is set to the value "0", otherwise to the value "-1".

Note

If you set a non-existing parameter set ID and the control job ID to "6", a new parameter set is created with the parameter set values available in the control system.

Writing a parameter set to the control system

To automatically load a parameter set from the parameter set memory and write it into the control system, follow these steps:

1. Automatically set the control tag "ParameterSetIDTag" to an ID of an existing parameter set.
2. Set the control tag "JobIDTag" automatically to the control job ID "7".
   The control job is executed. In the case of success the control tag "JobIDTag" is set to the value "0", otherwise to the value "-1".

Deleting a parameter set

To automatically delete a parameter set from the parameter set memory, follow these steps:

1. Automatically set the control tag "ParameterSetIDTag" to an ID of an existing parameter set.
2. Set the control tag "JobIDTag" automatically to the control job ID "8".
   The control job is executed. In the case of success the control tag "JobIDTag" is set to the value "0", otherwise to the value "-1".

Result

You have assigned control tags to a parameter set type with elements in the Engineering Systems and automatically transferred and deleted parameter sets between the HMI device and control system via the control tags in runtime.

See also

Assigning a tag of the data type HMI user data type to a parameter set type (Page 798)

Assigning a tag of the data type "PLC user data type" to a parameter set type (Page 800)

External tags (Page 140)

Creating external tags (Page 152)

Configuring the parameter set view (Page 805)

Transferring parameter sets via scripts (Page 804)

Managing parameter sets (Page 807)

Transferring parameter sets (Page 816)

11.2.7 Transferring parameter sets via scripts (RT Uni)

Introduction

In runtime, you can transfer parameter sets via local scripts between the HMI device and control system. To do so configure the local scripts in the Engineering Systems at events of screen objects or at the "Update" event of tasks.

Code examples for transferring parameter sets

To load a parameter set from the parameter set memory and write it into the control system, use the following code example:

```
let ps1 = ParameterSetTypes('MyPST1').ParameterSets(1);
ps1.LoadAndWrite(true);
```

To read a parameter set from the control system and store it in the parameter set memory, use the following code example:

```
let ps1 = ParameterSetTypes('MyPST1').ParameterSets(1);
ps1.ReadAndSave(HMIRuntime.ParameterSetTypes.Enums.hmiOverwrite.Disabled, true);
```

See also

Local scripts (Page 361)

Transferring and deleting parameter sets automatically (Page 801)

Transferring parameter sets (Page 816)

11.2.8 Configuring the parameter set view (RT Uni)

Introduction

To display, manage and exchange parameter sets with the control system in runtime, use a parameter set control. You configure a parameter set control in the engineering system.

Note

The parameters set view is only supported for Unified PC. If you have configured a parameter set view on a Unified Comfort Panel, you must delete the control before compiling.

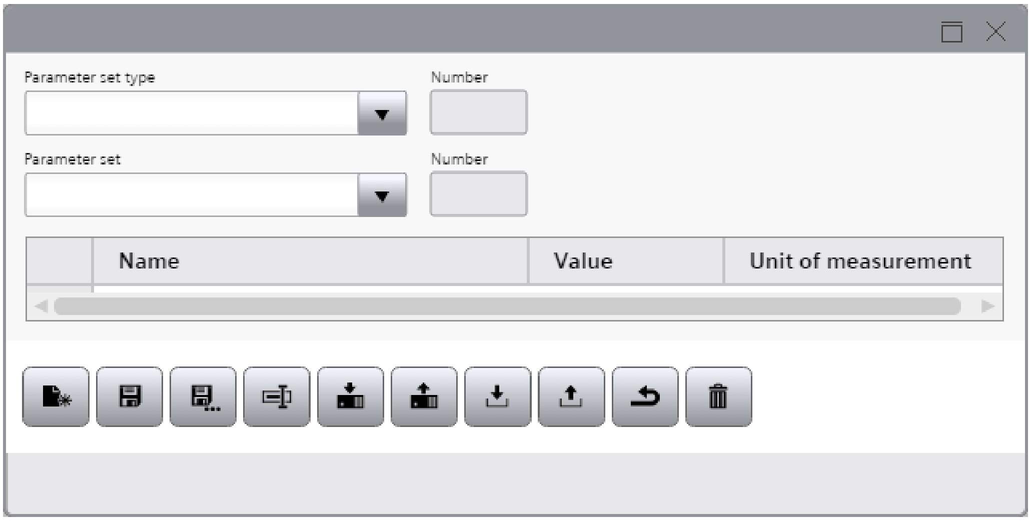
Requirement

- At least one parameter set type with elements has been created.
- A screen is open.
- The "Toolbox" task card is open.

Procedure

To configure a parameter set control, proceed as follows:

1. Insert the "Parameter set control" object from the "Tools" task card into the screen.



2. Go to "Properties" in the Inspector window and set the required height, width and position for the parameter set control.
3. If you only want to use a particular parameter set type with its parameter sets in runtime, select the desired parameter set type under "Properties > Fixed parameter set type".

4. If you want to hide the parameter set type field and the parameter set type ID field with the associated labels, clear the check box under "Properties > Display selection list".

   Note

   If you hide the two fields and do not select a parameter set type under "Properties > Fixed parameter set type", the Parameter set type field is disabled in runtime. In addition, no parameter set ID is displayed in the Parameter set ID field in runtime.

5. Change the labels of the fields, if required.
6. If required, change the display of the parameter table under "Properties > Parameter view".
7. In "Properties > Editing mode", configure the activation status of the toolbar buttons "Create", "Save", "Save as", "Rename" and "Delete" as required.
   These toolbar buttons are used to edit parameter sets.
8. If you want to hide specific buttons in the toolbar, deactivate the "Visibility" property under "Properties > Toolbar > Elements" in the settings of the corresponding button.
9. Under "Properties > Time zone", change the time zone as required by entering a different numerical value.
   The numerical value stands for a time zone, for example:
   - "-1" stands for UTC-1h (Central European Time, standard time)
   - "1" stands for UTC-12h (International Date Line West)
   - "2" stands for UTC-11h (Hawaii)

See also

Parameter set control (Page 785)

Creating a parameter set type with elements via an HMI user data type (Page 788)

Creating a parameter set type with elements via a PLC user data type (Page 791)

Transferring and deleting parameter sets automatically (Page 801)

Managing parameter sets (Page 807)

Exporting and importing parameter sets (Page 813)

Transferring parameter sets (Page 816)

11.3 Using parameter sets in runtime (RT Uni)

11.3.1 Managing parameter sets (RT Uni)

Introduction

You manage parameter sets for different productions in a parameter set control in runtime. You have the following options for managing parameter sets:

- Create new parameter sets
- Copy parameter sets
- Change parameter sets
- Delete parameter sets
- Rename parameter sets

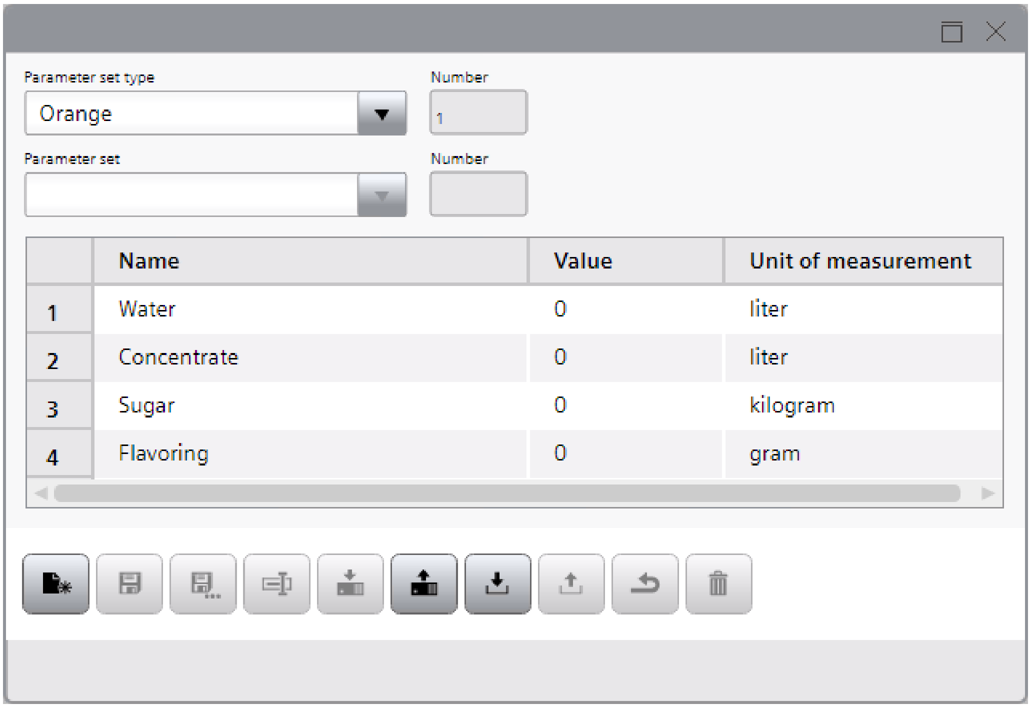
Requirement

- At least one parameter set type with elements has been created.
- A parameter set control has been configured.
- The project is in runtime.

Creating a new parameter set

To create a new parameter set, proceed as follows:

1. In the "Parameter set type" field, select the parameter set type for which you want to create a new parameter set.
   The elements of the selected parameter set type are displayed in the table.



Parameter set type: Orange

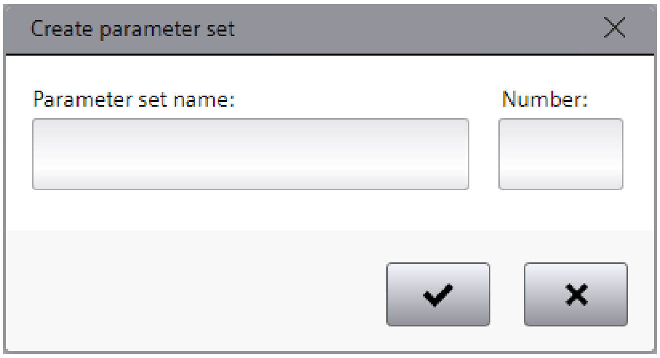
Number: 1

Parameter set:

Number:

	Name	Value	Unit of measurement
1	Water	0	liter
2	Concentrate	0	liter
3	Sugar	0	kilogram
4	Flavoring	0	gram

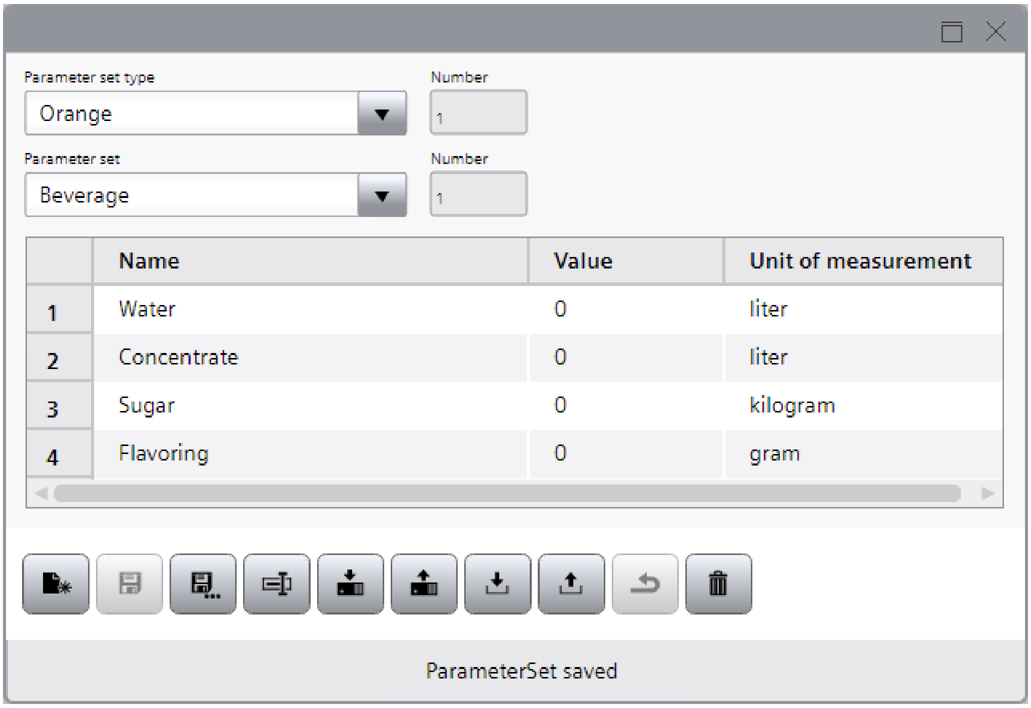
2. Click the "Create" button.
   The "Create parameter set" dialog opens.



3. Enter a unique parameter set name under "Parameter set name".

4. Enter a unique parameter set ID under "Number".

5. Confirm the dialog.
   A new parameter set has been created and saved. The parameters of the new parameter set are displayed in the table. The parameters have the same values in the columns "Name", "Value" and "Unit of measure" as the elements of the previously selected parameter set type.



Parameter set type: Orange
Number: 1
Parameter set: Beverage
Number: 1

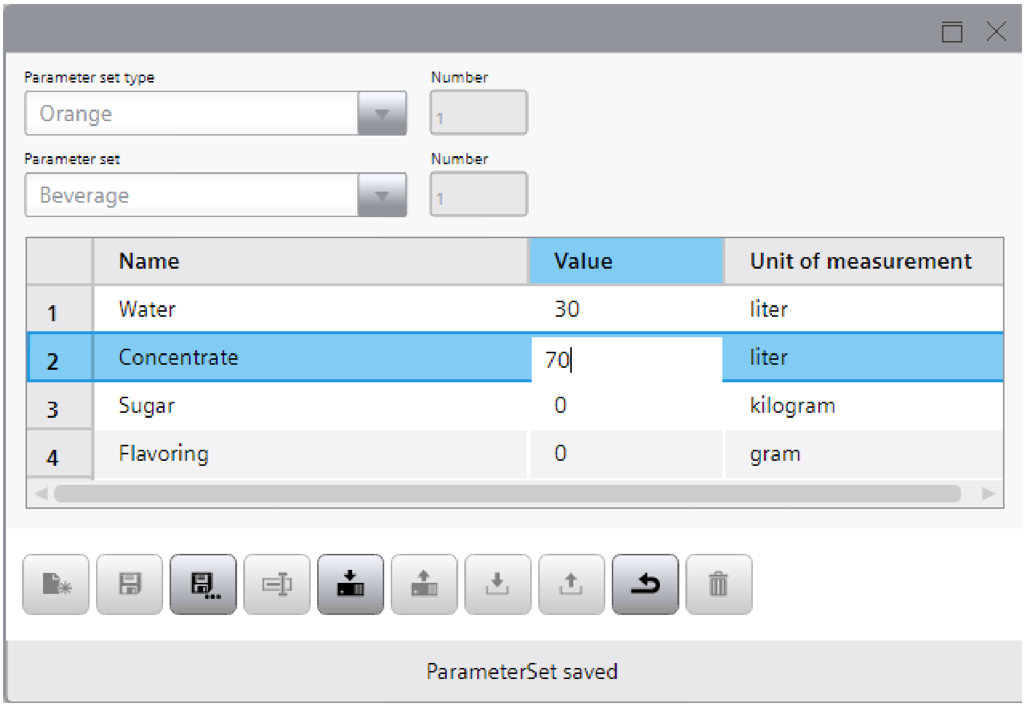
	Name	Value	Unit of measurement
1	Water	0	liter
2	Concentrate	0	liter
3	Sugar	0	kilogram
4	Flavoring	0	gram

ParameterSet saved

Note

If you do not make any entries in the "Create parameter set" dialog and confirm the dialog, a new parameter set is also created and saved. In this case the new parameter set, however, has a unique parameter set name and a unique parameter set ID which were both automatically assigned by the system.

6. Enter values for the parameters in the "Value" column.
   Depending on the configuration, the parameters already contain start values.



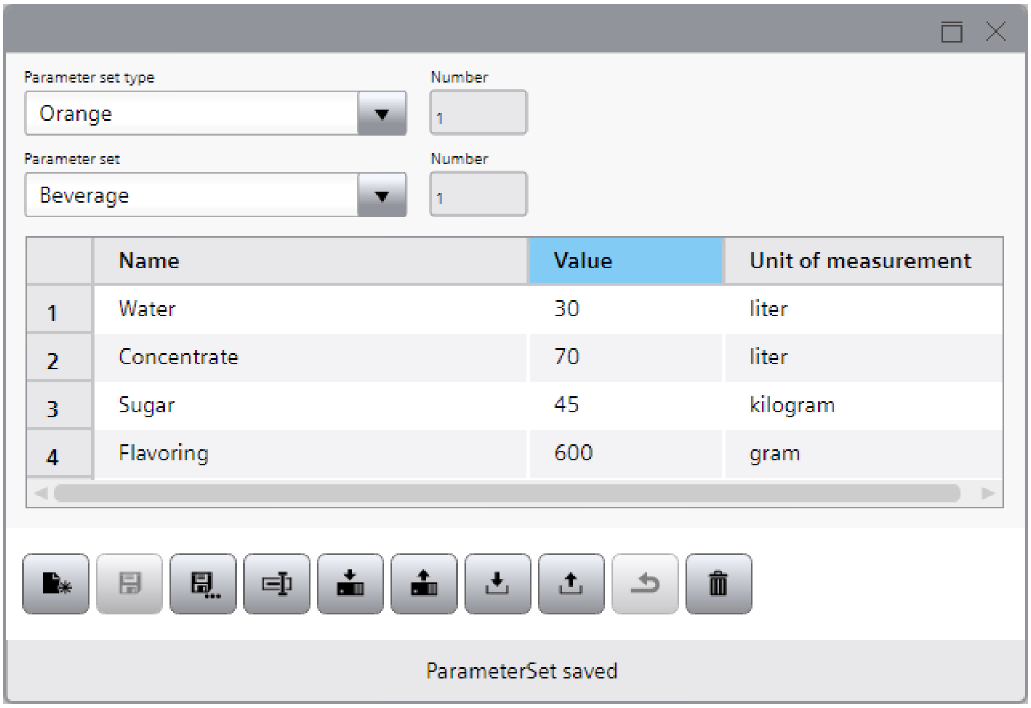
Parameter set type: Orange — Number: 1

Parameter set: Beverage — Number: 1

	Name	Value	Unit of measurement
1	Water	30	liter
2	Concentrate	70	liter
3	Sugar	0	kilogram
4	Flavoring	0	gram

ParameterSet saved

7. Click the "Save" button.



Parameter set type: Orange — Number: 1

Parameter set: Beverage — Number: 1

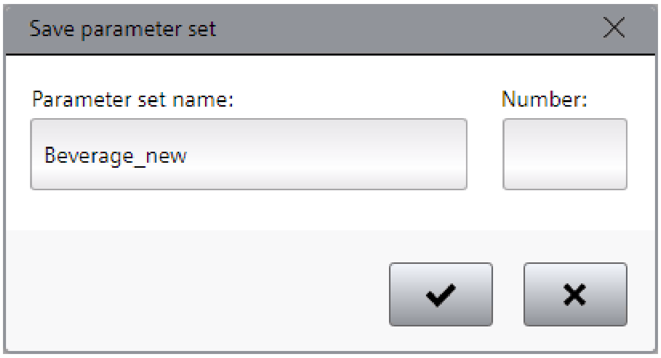
	Name	Value	Unit of measurement
1	Water	30	liter
2	Concentrate	70	liter
3	Sugar	45	kilogram
4	Flavoring	600	gram

ParameterSet saved

Copying a parameter set

To copy a parameter set, proceed as follows:

1. In the "Parameter set type" field, select the parameter set type in which you want to copy an existing parameter set.
   The elements of the selected parameter set type are displayed in the table.
2. In the "Parameter set" field, select the parameter set you want to copy.
   The parameters of the selected parameter set are displayed in the table.
3. Click the "Save as" button.
   The "Save parameter set" dialog opens. A unique parameter set name is pre-assigned to the "Parameter set name" field.


Save parameter set

Parameter set name: Beverage_new

Number:

4. Enter a different unique parameter set name under "Parameter set name" as required.
5. Enter a unique parameter set ID under "Number" as required.
6. Confirm the dialog.

Note

If you do not enter a parameter set ID in the "Save parameter set" dialog and confirm the dialog, a unique parameter set ID is automatically assigned to the new parameter set.

Changing the parameter set

To change a parameter set, proceed as follows:

1. In the "Parameter set type" field, select the parameter set type in which you want to change an existing parameter set.
   The elements of the selected parameter set type are displayed in the table.
2. In the "Parameter set" field, select the parameter set you want to change.
   The parameters of the selected parameter set are displayed in the table.
3. Edit the values of the parameters in the "Value" column.
4. Click the "Save" button.

Deleting a parameter set

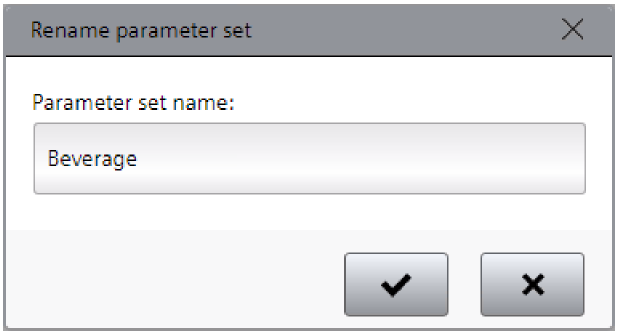
To delete a parameter set, proceed as follows:

1. In the "Parameter set type" field select the parameter set type in which you want to delete an existing parameter set.
   The elements of the selected parameter set type are displayed in the table.
2. In the "Parameter set" field, select the parameter set you want to delete.
   The parameters of the selected parameter set are displayed in the table.
3. Click "Delete".

Renaming a parameter set

To rename a parameter set, proceed as follows:

1. In the "Parameter set type" field, select the parameter set type in which you want to rename an existing parameter set.
   The elements of the selected parameter set type are displayed in the table.
2. In the "Parameter set" field, select the parameter set you want to rename.
   The parameters of the selected parameter set are displayed in the table.
3. Click the "Rename" button.
   The "Rename parameter set" dialog opens.



Rename parameter set

Parameter set name:

Beverage

4. Enter a different unique name for the parameter set under "Parameter set name".
5. Confirm the dialog.

See also

Parameter set control (Page 785)

Configuring the parameter set view (Page 805)

Transferring and deleting parameter sets automatically (Page 801)

Exporting and importing parameter sets (Page 813)

Transferring parameter sets (Page 816)

11.3.2 Exporting and importing parameter sets (RT Uni)

Introduction

In a parameter set control in runtime you export parameter sets from the parameter set memory to a "*.tsv" file to be able to edit them a text editor. In a parameter set control in runtime you furthermore import parameter sets from a "*.tsv" file into the parameter set memory.

Note

A "*.tsv" file is a text file that uses the tabulator as a list separator.

Note

To export and import the parameter sets, you can also use the system functions in the function list or in the scripts:

- With the system function "ExportParameterSets" or "ExportParameterSets", the parameter sets are exported from the parameter set memory to a "*.tsv" file.
- With the system function "ImportParameterSets" or "ImportParameterSets", the parameter records are imported from a "*.tsv" file into the parameter set memory.

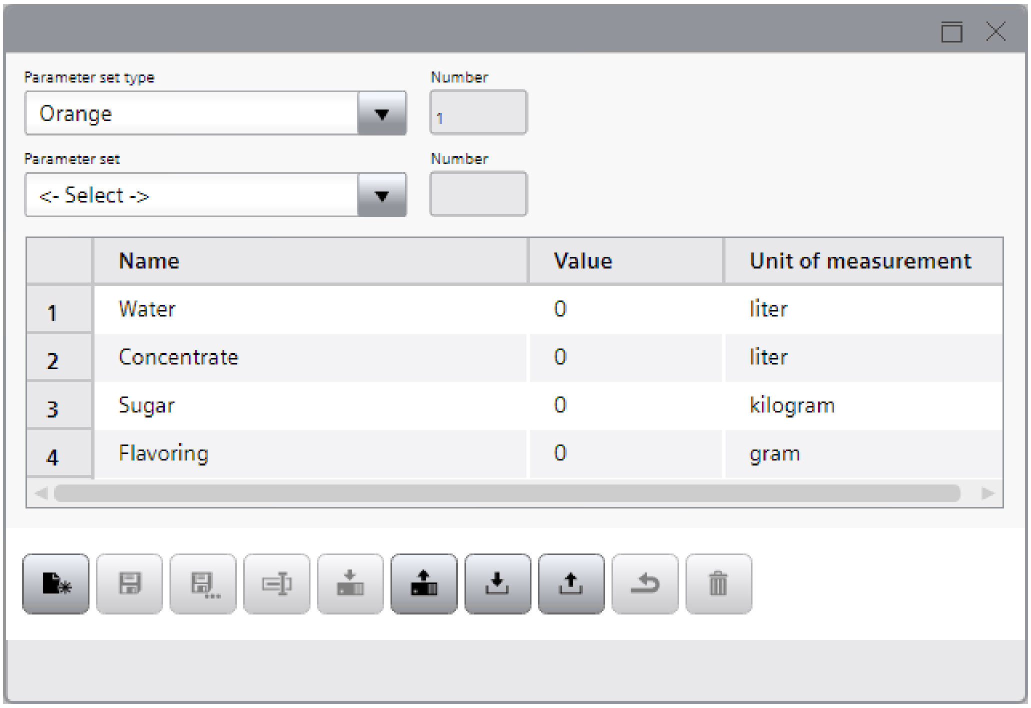
Requirement

- At least one parameter set type with elements has been created.
- A parameter set control has been configured.
- The project is in runtime.

Exporting parameter sets of a parameter set type

Follow these steps to export the parameter sets of a parameter set type:

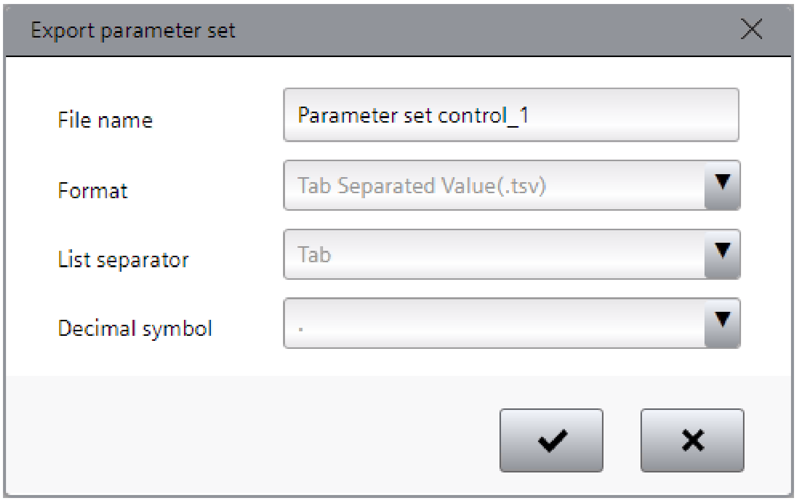
1. In the "Parameter set type" field, select the parameter set type whose parameter sets you want to export.



Parameter set type: Orange
Number: 1
Parameter set: <- Select ->
Number:

	Name	Value	Unit of measurement
1	Water	0	liter
2	Concentrate	0	liter
3	Sugar	0	kilogram
4	Flavoring	0	gram

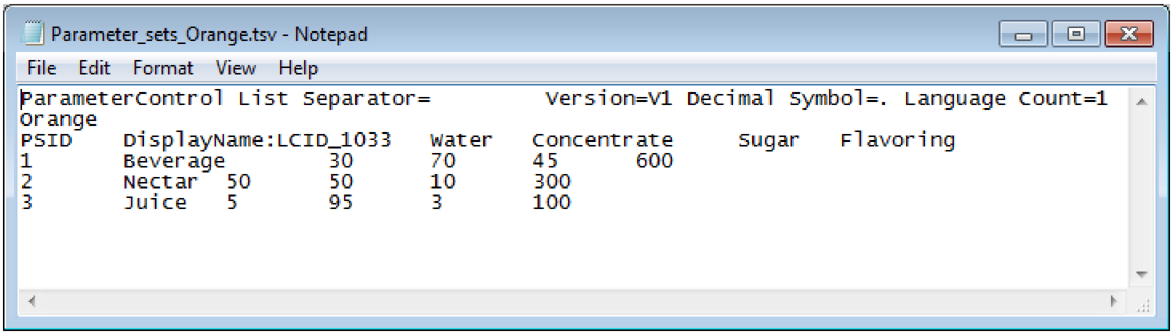
2. Click "Export".
   The "Export parameter set" dialog box opens. The name of the parameter set control is pre-assigned in the "File name" field.



Export parameter set

File name: Parameter set control_1
Format: Tab Separated Value(.tsv)
List separator: Tab
Decimal symbol: .

3. If appropriate, change the name of the file to which you want export the parameter sets under "File name".
4. Confirm the dialog.
   The parameter set types are exported to a ".tsv" file which you can find in the Windows folder "Downloads".
   The first row of the opened file contains the file header. The second line contains the name of the parameter set type. The third row contains the headers for parameter sets. From the fourth line on the parameter sets are listed.

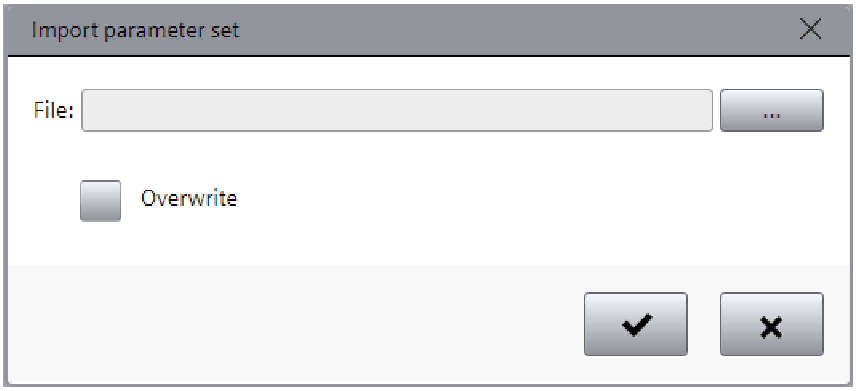

Parameter_sets_Orange.tsv - Notepad
File Edit Format View Help

```
ParameterControl List Separator=        Version=V1 Decimal Symbol=. Language Count=1
Orange
PSID        DisplayName:LCID_1033   Water   Concentrate     Sugar   Flavoring
1           Beverage        30      70      45      600
2           Nectar    50    50      10      300
3           Juice     5     95      3       100
```

Importing parameter sets into a parameter set type

To import parameter sets into a parameter set type, follow these steps:

1. In the "Parameter set type" field, select the parameter set type into which you want to import the parameter sets.
2. Click "Import".
   The "Import parameter set" dialog box opens.



3. Select the file from which you want to import the parameter sets.

Note

The import file must have the same file header and the same header for parameter sets as the export file. Otherwise the import of parameter sets is not possible.

4. To overwrite parameter sets in the parameter set control that have the same ID as the imported parameter sets, activate the "Overwrite" option.

   Note

   If you deactivate overwriting and if a parameter set with the same ID is available in the parameter set control, the import of parameter sets is not possible.

5. Confirm the dialog.
   The parameter sets are imported to the parameter set type.

See also

Parameter set control (Page 785)

Configuring the parameter set view (Page 805)

Managing parameter sets (Page 807)

11.3.3 Transferring parameter sets (RT Uni)

Introduction

You have assigned an external HMI tag of the data type HMI or PLC user data type to a parameter set type. In a parameter set control in runtime you transfer the values of parameter sets to the control system via the HMI tag. The parameter set values are used to set up machines for different productions.

In a parameter set control in runtime you furthermore read active parameter sets from the control system into the parameter set control via the HMI tag. The read parameter set values are stored in the parameter set memory. By reading from the PLC you call up currently used values of production machines for future use.

Note

You can also use system functions in the function list or in scripts to transfer parameter sets between HMI device and PLC:

- With the system function "ReadAndSaveParameterSet" or "ReadAndSaveParameterSet", a parameter set is read from the PLC and saved in the parameter set memory.
- With the system function "LoadAndWriteParameterSet" or "LoadAndWriteParameterSet", a parameter set is loaded from the from the parameter set memory and written to the PLC.

Requirement

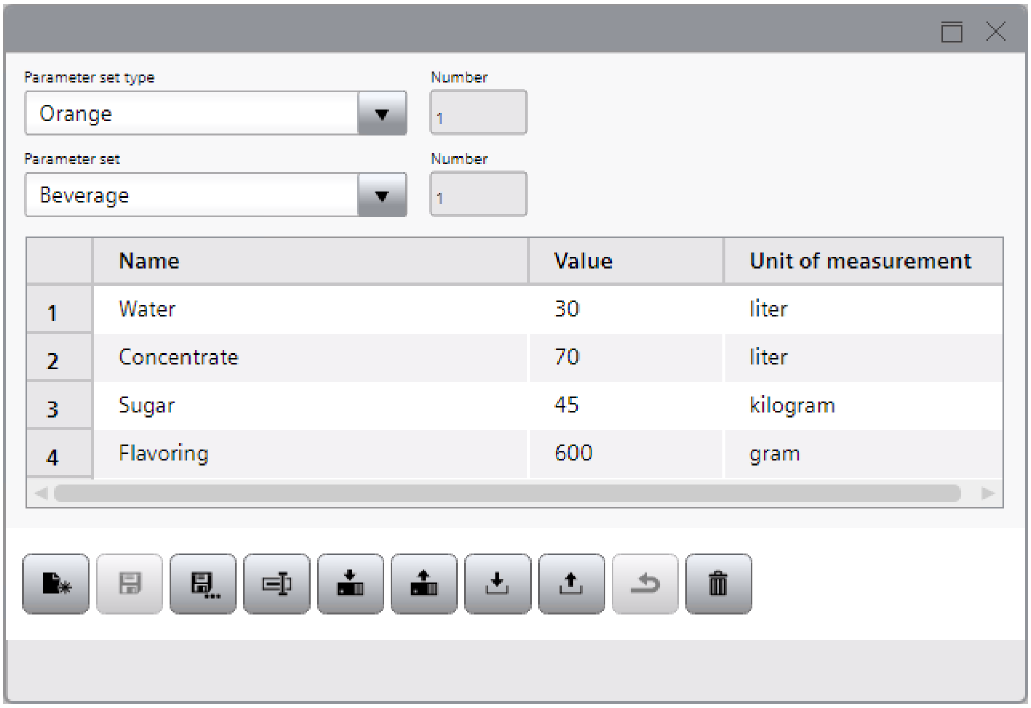
- A parameter set type with elements has been created.
- An external HMI tag of the data type HMI or PLC user data type is assigned to the parameter set type.
- A parameter set control has been configured.

- The project is in runtime.
- At least one parameter set has been created in the parameter set type.

Transferring a parameter set to the PLC.

Proceed as follows to transfer a parameter set to the PLC:

1. In the "Parameter set type" field, select the parameter set type.
2. In the "Parameter set" field, select the parameter set whose values you want to transfer to the PLC.



Parameter set type: Orange
Number: 1
Parameter set: Beverage
Number: 1

	Name	Value	Unit of measurement
1	Water	30	liter
2	Concentrate	70	liter
3	Sugar	45	kilogram
4	Flavoring	600	gram

3. Click the "Write to PLC" button.

Reading a parameter set from PLC

Proceed as follows to read a parameter set from the PLC:

1. In the "Parameter set type" field, select the parameter set type.
2. In the "Parameter set" field, select the parameter set whose values you want to read from the PLC.

   Note

   If do you not select a parameter set in the in the "Parameter set" field, a new parameter set is created in the parameter set control while reading from the control system.

3. Click the "Read from PLC" button.

Result

You have transferred the values of parameter sets between the HMI device and control system.

See also

Assigning a tag of the data type HMI user data type to a parameter set type (Page 798)

Assigning a tag of the data type "PLC user data type" to a parameter set type (Page 800)

Parameter set control (Page 785)

Configuring the parameter set view (Page 805)

Managing parameter sets (Page 807)

Transferring and deleting parameter sets automatically (Page 801)

Transferring parameter sets via scripts (Page 804)

Configuring user administration (RT Uni) 12

12.1 Basics of user administration (RT Uni)

Introduction

The TIA Portal provides the possibility to perform user administration for projects. In this way, for example, a project can be protected against unintentional or unauthorized modification. A user sets up the project protection to activate user management. This user is created as a project administrator. Once the project protection has been activated, the project can only be opened and edited by authorized users. Note that project protection cannot be revoked.

Permissible users are local project users and global users and user groups added by a project administrator:

- Local project users:
  Users who are defined and managed in a TIA Portal project. These user accounts are only valid for this one project. It makes sense to use project user accounts if the entire automation solution is engineered in one project.
- Global users and user groups:
  These user accounts are defined and managed outside the TIA Portal in UMC (User Management Component). Global users and user groups can be imported into the various TIA Portal projects in which these users will work. To add users and user groups from UMC, the corresponding rights are required in UMC.

You can assign specific roles to users or user groups, which in turn can be linked to different function rights. There are the following general engineering role permissions to which a role can be assigned:

- Open the project read-only
  A user who has only this role permission can open the project, but not modify it. However, he cannot open the editors for user administration.
- Open the project with write rights
  A user with this role permission can modify the project. However, he cannot open the editors for user administration with this.
- Managing users and roles
  A user with this role permission can manage the users and roles of the project. In addition, the user still requires the "Open the project with write rights" role permission because the project cannot be opened or edited with the "Manage users and roles" role permission alone.

In addition to these general role permissions, further runtime rights can be assigned. The descriptions of these specific function rights can be found in the corresponding areas of the help. Several function rights can be assigned to a role.

When you enable user administration, the system creates the following two roles:

- "ES Administrator"
  The first created project user is assigned the role "ES Administrator". This role has all three engineering role permissions by default. Each project needs at least one administrator who is allowed to edit the project and the security settings. In addition, other users may be assigned the right to manage users and roles.
- "ES Standard"
  Users who are assigned the role "ES Standard" have the permissions "Open project read-only" and "Open project read/write".

You cannot change or delete these system roles. You can create additional roles and assign them the required role permissions.

Note

Additional local user administrations

In addition to the user administration for projects, there are additional user administrations in certain areas of the TIA Portal, e.g. for WinCC Panels.

Settings for users

A project administrator can make the following settings for a local project user:

- User name:
  Name of the local project user who must be used to log on to the project.
- Password:
  The password of the local project user assigned by the administrator with which the project user can log onto the project. The project user can change the password later.
- Authentication method:
  Authentication methods defined by the administrator. The following two methods are available:
  - Password: Login is via a password which was defined in the TIA Portal or in UMC.
  - Radius: Login is via a RADIUS server on which the password is stored. This option can only be used for devices that support login via a RADIUS server.
- Maximum session duration
  Maximum duration for which the user can be logged on to a device. The user is logged off from the device after the time has expired. This setting is only available for devices that support a session duration. The session duration is not evaluated in the TIA Portal itself.

UMC - User Management Component

In addition, you can install the "User Management Component UMC" software package on one or multiple computers; it provides a central user administration. This creates a system of possibly interactive UMC installations (UMC ring server, UMC server). Then you can define users and user groups in the UMC system or import them from the Windows Active Directory. When UMC is installed, you can access the UMC server from the TIA Portal to add the users and user groups defined there to the user administration of the TIA Portal and use roles to assign them specific role permissions in a TIA Portal project.

However, you cannot change the data of users and user groups added from UMC within the TIA Portal. For example, even as administrator in the project you cannot change passwords or other data of UMC users or UMC user groups. This is only possible in UMC. However, you have the option to synchronize user administration in the TIA Portal with UMC or to check the synchronization status. This enables you to eliminate inconsistencies between the global users and user groups in UMC and the UMC users or UMC user groups already imported in the TIA Portal.

Note

UMC documentation

The UMC installation file and the UMC documentation in English is available on the TIA Portal installation data storage medium ("..\support", "...\Documents\Readme\English").

We highly recommend that you read the UMC documentation completely before you start working with the user administration, especially the sections on "Secure Application Data Support (SADS)". SADS is mandatory for working with the user administration in the TIA Portal.

User administration in the TIA Portal and Multiuser Engineering

Note the following information when working with the user administration in the TIA Portal in combination with Multiuser Engineering:

- You can only make changes to the user administration of a multiuser project in a server session and not in the local sessions.
- When you upload a single-user project that is protected with user administration to a multiuser server, the resulting multiuser project is also protected. The users and user groups contained in the project continue to exist with their passwords and can therefore continue to log into the local sessions of the multiuser project, provided they have the corresponding user rights.
- If you protect a previously unprotected local session using the user administration, this protection is removed once again after checking-in and updating the local session, as the multiuser project on the server is not protected as a result. This also applies when several operators are protecting their local sessions.
- If you protect the multiuser project on the server using the user administration, the local sessions created by this multiuser project are also protected as soon as these are updated.
- When you log off from your local session, which is protected using the user administration, other users remain logged on to their local sessions of the multiuser project. You only log yourself off. Your local session is the closed as a result.
- If the multiuser project on the server is protected using the user administration during processing of your unprotected local session and your user account has not been assigned the "Open project with read/write permission" role permission, you cannot check-in any changes to your local session or update your local session.

More information on Multiuser Engineering can be found in section "Using Multiuser Engineering".

12.2 Configuring user administration (RT Uni)

12.2.1 Setting password policies (RT Uni)

You can specify the structure and the complexity of the project user passwords for the engineering system and Runtime. To do this, you can specify the following policies:

- "Minimum password length"
  The minimum number of characters that the user password must have.
- "Minimum number of numeric characters"
  The minimum number of numeric characters that the user password must contain.
- "Minimum number of special characters"
  The minimum number of special characters that the user password must contain.
- "At least one upper case and one lower case letter"
  Specifies that the user password must contain at least one uppercase and one lowercase letter.
- "Number of recently used passwords blocked for reuse"
  Sets the number of recently used passwords that cannot be used as a new password.
- "Enable password aging"
  Specifies that the password only has a certain validity period and then expires. If this option is selected, the password must be changed within the password validity.
- "Password validity (days)"
  Sets the password validity in days, in which the password must be changed when the password is activated.
- "Prewarning time (days)"
  Sets how many days before the user's password expires, a warning is provided to the user. The warning time must be less than that for the password validity.

Note

Permissible characters for usernames and passwords

You can use the following numbers, letters and special characters for usernames and passwords:

- 0123456789
- A...Z a...z
- !#$%&()*+,-./:;<=>?@ [\]_{|}~^

Requirement

- A project is open.

Procedure

To set the password policies, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click "Settings".
   The "Settings" editor opens in the working area and the "Project protection" area is displayed.
3. Click "Password policies".
   The "Password settings for Runtime and engineering" area is displayed.
4. Make all the desired settings.

12.2.2 Managing project users (RT Uni)

To manage local users in your project you can perform the following actions:

- Create a new project user
- Change the user name of a project user
- Change the password of a project user
- Change the authentication procedure of a project user
- Change the maximum session duration of a project user
- Change the comment for a project user
- Delete project users

Please note the following restrictions:

- A maximum of 256 project users can be added.
- The user name may not exceed 255 characters.
- The password may not exceed 120 characters and must meet the defined password guidelines.
- The comment may not exceed 1000 characters.

In contrast, the following restrictions apply to CPs:

- A maximum of 33 project users can be added.
- The user name may not exceed 32 characters.
- The password may not exceed 32 characters and must meet the defined password guidelines.

Requirement

- A project is open.

Create new project user

To create a new project user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Click "Add new user".
   A submenu is opened in which you can select the user type.
5. Click "Add new project user".
6. Enter a user name in the "User name" column.
7. Click on the arrow in the "Password" column.
8. Enter a password.
9. Re-enter the password to confirm.
10. Open the drop-down list in the "Authentication procedure" column and select the authentication procedure for the project user.
11. In the "Maximum session duration" column configure the maximum session duration for the project user.
12. If necessary, enter a comment for the project user in the "Comment" column.
    A new project user has been created. Next, you can assign roles to the project user.

Note

You can also create a new project user by copying an existing project user. The roles assigned to the original project user are also assigned to the copied project user. However, you must assign a new password for the copied project user.

Changing the user name of a project user

To change the user name of a project user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Change the user name in the "User name" column.

Changing the password of a project user

To change the password of a project user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Click on the arrow next to the password in the "Password" column.
5. Enter the new password.
6. Re-enter the password to confirm.

Changing the authentication procedure of a project user

To change the authentication procedure of a project user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Open the drop-down list in the "Authentication procedure" column and select the preferred authentication procedure.

Changing the maximum session duration of a project user

To change the maximum session duration of a project user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. In the "Maximum session duration" column configure a new maximum session duration.

Change the comment for a project user

To change the comment for a project user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Enter a new comment in the "Comment" column.

Deleting project users

To delete a project user, proceed as follows:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Select the project user.
5. In the shortcut menu select the "Delete" command or use the <Del> key.

12.2.3 Managing global users and user groups (RT Uni)

You can add global users and user groups which were created in UMC (User Management Component) to a project.

Global users can be managed in UMC and the changes are then effective in every project to which the users or user groups belong. You have the option to synchronize the user administration in the TIA Portal with UMC so that no inconsistencies occur if users or user groups were changed in UMC. To determine whether synchronization is necessary, you can check the synchronization status. You can find information on synchronization in the banner of the user administration.

Please note the following restrictions with regard to global users and user groups:

- A maximum of 256 global users can be added.
- A maximum of 50 global user groups can be added.
- Global users and user groups cannot be copied.

You can display the following information for global users and user groups:

- User groups: The users they contain and whether they have already been imported into the user administration.
- User: The user groups of which the user is a member and whether they have already been imported into the user administration.

Requirement

- A project is open.
- The connection to UMC is configured and you have a user account with the corresponding rights in UMC.
  For further information on installation and configuration of UMC, refer to the English documentation on UMC on the installation data carrier in the directory "/document/Readme/English".

Adding global users

To add a global user from UMC, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Click "Add new user".
   A submenu is opened in which you can select the user type.
5. Click "Add user from UMC".
   If you are not yet logged on in UMC, the "UMC login" dialog opens. Then also perform steps 6 and 7.
6. Enter your UMC user name and the corresponding password.
7. Click "OK".
   As soon as you are logged on in UMC, the "Add user from UMC" dialog opens. All the available users from UMC are displayed in this dialog. Already activated users are activated and write-protected.
8. Activate the users you wish to add to your TIA Portal project.

   Note

   To find the required users more quickly, you can filter the table by the columns "Name" and "Long name". To do this, enter part of the name or long name n the first line. All users whose names or long names contain the text entered are displayed. To cancel filtering, click the Filter button or select "*" from the drop-down list.

9. Click "OK" to add the selected users.
   The selected users are added as global users. Your data is write-protected and cannot be changed within the TIA Portal project. However, a global user can change his own password.

Deleting global users

To delete a global user from the TIA Portal project, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Select the global user. You can also delete several users at the same time via multi-selection.
5. In the shortcut menu select the "Delete" command or use the <Del> key.

Adding a global user group

To add a global user from UMC, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "User groups" tab.
4. Click "Add new user group".
   If you are not yet logged on in UMC, the "UMC login" dialog opens. Then also perform steps 5 and 6.
5. Enter your UMC user name and the corresponding password.
6. Click "OK".
   As soon as you are logged on in UMC, the "Add user group from UMC" dialog opens. All the available user groups from UMC are displayed in this dialog. Already activated user groups are activated and write-protected.
7. Activate the user groups that you wish to add to your TIA Portal project.
8. Click "OK" to add the selected user groups.
   The selected user groups are added as global user groups. Your data is write-protected and cannot be changed within the TIA Portal project.

Deleting a global user group

To delete a global user group from the TIA Portal project, proceed as follows:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "User groups" tab.
4. Select the global user group. You can also delete several user groups at the same time via multi-selection.
5. In the shortcut menu select the "Delete" command or use the <Del> key.

Display users of an added global user group

To display the users included in a global user group, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "User groups" tab.
4. Select the user group. Note that you cannot use multiple selection.
5. In the bottom area of the "User groups" tab open the "Users" tab.
   The users of the selected group are displayed. The check box is enabled in the "Imported" column for users who have already been imported into the user administration.

Display user groups of a global user

To display the user groups for an added global users of which they are a member, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Select the global user. Note that you cannot use multiple selection.
5. In the bottom area of the "Users" tab, open the "Assigned user groups" tab.
   The user groups of which the selected global user is a member are displayed. The check box is enabled in the "Imported" column for user groups that have already been imported into the user administration.

Checking the synchronization status

To check the synchronization status, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Show the banner with the synchronization information, and click the "Check status" link.

Or:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab or the "User groups" tab.
4. Click "Check status" in the toolbar.

Performing synchronization

To synchronize, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Show the banner with the synchronization information, and click the "Synchronize" link.

Or:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab or the "User groups" tab.
4. Click "Synchronize" in the toolbar.

12.2.4 Managing roles (RT Uni)

Rights are assigned to users via roles. The following system-defined roles without engineering function rights are created in an unprotected project:

- HMI Administrator
- HMI Operator
- HMI Monitor
- NET Remote Access
- NET Administrator Radius
- NET Radius

When you activate project protection for a project, two additional system-defined roles with engineering function rights are created: the roles "Engineering Administrator" and "Engineering Standard".

You cannot rename or delete system-defined roles. You also cannot change the assignment of the function rights to system-defined roles.

You can, however, create user-defined roles and assign them function rights. You can also perform the following actions for user-defined roles:

- Change name of role
- Change maximum session duration
- Change comment for role
- Changing or deleting the assignment of function rights
- Delete a role

You can display the assigned function rights for each project user or global user and for each global user group. This will give you an overview of the assigned rights at any time.

Requirement

- A project is open.
- An HMI device has been created.

Create new user-defined roles

To create a new user-defined role, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Roles" tab.
4. Double-click "Add new role".
5. Enter a name for the role in the "Name" column.

6. In the "Maximum session duration" column configure the maximum session duration for the role.
7. If necessary, enter a comment for the role in the "Comment" column.
   A new user-defined role has been created. Next, you can assign function rights to the role.

Note

You can also create a new user-defined role by copying an existing role. This means the assigned role permissions are also assigned to the new role.

Assign function rights to a user-defined role

To assign function rights to a user-defined role, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Roles" tab.
4. Select the user-defined role. Please note that you cannot use multi-selection for assignment.
5. In the lower area "Function rights categories", open the categories from which you want to assign the role permissions.
6. In the lower area "Function rights", activate the function rights you wish to link to the role.

Change name of a user-defined role

To change the name of a user-defined role, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Roles" tab.
4. Change the name of the user-defined role in the "Name" column.

Change maximum session duration of a user-defined role

To change the maximum session duration of a user-defined role, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Roles" tab.
4. In the "Maximum session duration" column configure a new maximum session duration.

Change comment for a user-defined role

To change the comment for a user-defined role, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Roles" tab.
4. Enter a new comment in the "Comment" column.

Changing or deleting the assignment of function rights

To change or delete the assignment of the function rights for a user-defined role, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Roles" tab.
4. Select the user-defined role.
5. In the lower area "Function rights categories", open the categories from which you want to assign the function rights or remove the assignment.
6. In the lower area "Function rights", activate the function rights you wish to assign to the role.
7. In the lower area "Function rights", deactivate the function rights that are no longer assigned to the role.

Delete user-defined role

To delete a user-defined role, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Roles" tab.
4. Select the user-defined role.
5. In the shortcut menu select the "Delete" command or use the <Del> key.

Display assigned function rights of a project user or of a global user

To display the assigned function rights of a project user or of a global user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.

4. Select the user. Note that you cannot use multiple selection.
5. In the bottom area of the "Users" tab, open the "Assigned rights" tab.
6. Expand the categories of function rights in the "Categories of function rights" column.
   The function rights that are assigned to the user are displayed in the "Rights list" column. The roles by which the function rights are assigned to the user are displayed in the "Rights derived from role" column.

Display assigned function rights of a global user group

To display the assigned function rights of a global user group, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "User groups" tab.
4. Select the global user group. Note that you cannot use multiple selection.
5. In the bottom area of the "User groups" tab, open the "Assigned rights" tab.
   The function rights that are assigned to the global user group are displayed in the "Assigned rights" column.

12.2.5 Assigning roles (RT Uni)

You can assign roles to project users or global users and user groups that have different function rights. You can revoke the assignments at any time.

Requirement

- A project is open.
- Users and user groups have been created.

Assigning roles to project users and global users

To assign roles to a user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Select the user. Note that you cannot use multiple selection.
5. Enable the desired roles in the "Assigned roles" section.

Assigning roles to global user groups

To assign roles to a global user group, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "User groups" tab.
4. Select the user group. Note that you cannot use multiple selection.
5. Enable the desired roles in the "Assigned roles" section.

Revoking role assignments for project users and global users

To revoke a role for a user, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Users" tab.
4. Select the user. Note that you cannot use multiple selection.
5. In the "Assigned roles" section, clear check marks for the roles that are no longer to be assigned to the user.

Revoking role assignments for global user groups

To revoke a role assignment for a global user group, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "User groups" tab.
4. Select the user group. Note that you cannot use multiple selection.
5. In the "Assigned roles" section, clear check marks for the roles that are no longer to be assigned to the user group.

12.2.6 Activate project protection (RT Uni)

To protect your TIA Portal project from unauthorized access, activate the project protection. A project administrator user is automatically created when you activate the project protection. The project administrator can create additional users. When the project protection is activated, the project can only be opened and changed with a user account with sufficient rights.

Note

Note that project protection cannot be disabled once it has been activated.

Requirement

- A project is open.

Procedure

To activate project protection, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click "Settings".
   The "Settings" editor opens in the working area and the "Project protection" area is displayed.
3. Click "Protect this project".
   The "Protect project" dialog opens in which you specify the logon information for the project administrator.
4. Enter a user name for the project administrator.
5. Enter the password for the project administrator.
6. Re-enter the password to confirm.
7. If necessary, enter a comment for the project administrator.
8. Confirm your inputs with the "OK" button.
   The project protection is activated. The projector administrator and the system-defined roles "Engineering Administrator" and "Engineering Standard" are created. You are logged on as project administrator. The project administrator is assigned the system-defined role "Engineering Administrator".
9. Under "Security settings > Users and roles > Users", assign new passwords for project users that you created before you activated the project protection.

Result

The project protection was activated. As logged on project administrator, you can now create additional local users or add global users or user groups from UMC or create new roles.

Note

After project protection is activated, at least one user with password must always be assigned to the system-defined role "Engineering Administrator" with the "Password" authentication procedure.

12.2.7 Log on to a protected project (RT Uni)

Introduction

You can log on to a protected project with a user account that has the corresponding function rights.

Requirement

- A protected project has been created.
- A user account as project user or as global user is created that is associated with at least one of the following two function rights:
  - Open project read-only
  - Open project with read and write permissions

Procedure

To log on to a protected project, follow these steps:

1. Select the "Open" command in the "Project" menu.
   The "Open project" dialog box opens and the list of recently used projects is displayed.
2. If you want to check whether the project has been inadvertently modified outside the TIA Portal when you open the selected project, select the "Activate basic integrity check" option. The basic integrity check may take some time to complete.
3. Select the protected project from the list that you want to open. If the desired project does not exist in the list, perform the following steps:
   - Click "Browse".
     The "Open existing project" dialog opens.
   - Navigate to the desired project folder and select the project file.
4. Click "Open".
   The "Log on" dialog opens.
5. Select the user type.
6. Enter your user name.
7. Enter your password.
8. If you want to change the password for your user account, click "Change password".

   Note

   If you log in as a global user, the password change is mandatory at the first logon, if this has been defined in UMC. Then perform the steps for changing the password.
9. Click the "OK" button.
   The protected project is opened.

12.2.8 Change password for protected project (RT Uni)

You can change your password for a protected project either when you log on to the project or when you work in the project. If you change your password as a project user, only the password for this project will be changed. If you change your password as a global user, your password will be changed in UMC. This will change your password for all projects for which you are registered as a user.

Requirement

- A protected project has been created.
- A user account as project user or as a global user is created that is associated with at least the function right "Open project with read and write permissions".

Change password when logging on to a protected project

To change your password when logging on to a protected project, follow these steps:

1. Select the "Open" command in the "Project" menu.
   The "Open project" dialog opens and the list of recently used projects is displayed.
2. Select a protected project from the list.
3. Click "Open".
   The "Log on" dialog opens.
4. Click "Change password".
   The "Change password" dialog opens.
5. Enter your user name.
6. Enter your current password.
7. Enter your new password.
8. Enter your new password again for confirmation.
9. Click the "OK" button.
   A dialog opens which informs you that the password change was successful.
10. Click the "OK" button.
    The password change process is complete. You can then use the new password when you log on to a protected project.

Changing a password when working in a protected project

To change your password when working in a protected project, follow these steps:

1. Open the project view.
2. Click the down arrow in the toolbar next to the "User Management" button.
   A menu is opened in which the user administration functions are listed.
3. Select the "Change password" menu command.
   The "Change password" dialog opens.
4. Enter your user name.
5. Enter your current password.
6. Enter your new password.
7. Enter your new password again for confirmation.

8. Click the "OK" button.
   A dialog opens which informs you that the password change was successful.
9. Click the "OK" button.
   The password change process is complete. You can then use the new password when you log on to a protected project.

12.2.9 Log off from a protected project (RT Uni)

You can log off explicitly from a protected project. The project is closed.

Procedure

To log off from a protected project, follow these steps:

1. Open the project view.
2. Click the down arrow in the toolbar next to the "User Management" button.
   A menu is opened in which the user administration functions are listed.
3. Select the "Log off and close project" menu command.
   The "Log off and close project" dialog is opened.
4. Click "Yes".
   If you have not made any changes to the project since it was saved last, you are logged off and the project is closed. If you have made changes to the project since it was saved last, "The project was changed." dialog opens.
5. When the "The project was changed." dialog opens, you either click "Yes" to save the project changes or "No" to close the project without saving the project changes.
   You are logged off and the project is closed either with or without saving the project changes.

12.2.10 Specify user administration used on Unified Comfort Panel (RT Uni)

Introduction

In the engineering system you specify whether you want to use the local or central user administration on a Unified Comfort Panel. This means you specify in the engineering system whether you want to work with local project users or with global users from UMC on a Unified Comfort Panel. By default, the use of the local user administration is specified in the engineering system.

Requirement

- A project is open.
- A Unified Comfort Panel has been created.

Procedure

To specify the user administration that is used on a Unified Comfort Panel, follow these steps:

1. Open the "Runtime settings" of the HMI device in the project tree.
2. Under "User administration > Configuration of user administration" activate either the use of the local user administration or of the central user administration.
3. If you have activated the use of the central user administration, enter the UMC server address, server ID and user name in the corresponding fields.

12.2.11 Limit access to Unified Comfort Panel (RT Uni)

Introduction

You can limit access to the Control Panel of a Unified Comfort Panel in the engineering system. To do so, activate access control for the Control Panel. In addition, you assign the function right for access to the Control Panel to users who are to have access to the Control Panel.

Requirement

- A project is open.
- A Unified Comfort Panel has been created.
- A user has been created.
- A user-defined role has been created.

Activate access control for Control Panel

To activate access control for the Control Panel, follow these steps:

1. Open the "Runtime settings" of the HMI device in the project tree.
2. Under "General > Control Panel" activate access control for the Control Panel.

Assign access right to Control Panel to a user

To assign a user the function right for access to the Control Panel, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the working area.
3. Open the "Roles" tab.
4. Select a user-defined role.
5. In the lower area "Function rights categories", open the category of the Runtime rights.
6. Click on the category of the Unified Comfort Panels.
7. In the lower area "Function rights", activate the function right for access to the Control Panel.

8. Open the "Users" tab.
9. Select a user.
10. In the lower area "Assigned roles", activate the user-defined role to which you have assigned the function right for access to the Control Panel.
    The user receives the access right for the Control Panel.

12.3 Use user administration in Runtime (RT Uni)

12.3.1 Log on to user administration in Runtime (RT Uni)

Introduction

To manage local project users in runtime, log on to user management in runtime.

Requirement

- An administrator account for user management in runtime was created when you installed WinCC Runtime Unified or later in WinCC Unified Configuration.
- A project with a SIMATIC Unified HMI device has been created.
- Project users have been created.
- The project has been downloaded to the HMI device.
- Runtime is active.

Procedure

To log on to user management in runtime, follow these steps:

1. Click "User management" on the start page.
   The "User log on" dialog opens.
2. Enter the user name and password of your administrator account for user management in runtime.
3. If necessary, use the selection list to change the displayed language.
4. Click "Login".
   The user management start page opens in runtime.

Compiling and loading (RT Uni) 13

13.1 Unified Comfort (RT Uni)

13.1.1 Runtime settings (RT Uni)

13.1.1.1 Settings in the runtime software (RT Uni)

Introduction

To edit the runtime settings for an HMI device, select "Runtime settings" under your HMI device in the project tree.

Overview

You can view or edit the following settings:

Setting		Description
General	Identification	Indicates the project identification.
	Encrypted transfer	Make settings for encrypted transfer. You can find additional information under "Encrypted transfer".
	Screen	Specifies the start screen.
Alarms	Controller alarms	Displays the alarms of the controller.
	State texts	Specifies the texts of different states.
Services	Reading/writing tags	Specifies that the HMI device works as an OPC server.
Language & font	Runtime language and font selection	• Specifies the available runtime languages and the sequence of language selection. • Specifies the default font. • Specifies the languages for logging in runtime.
Collaboration	Identification	Specifies the system identification of the collaboration and the collaboration name.

Setting		Description
Storage system	Database type	Specifies the database type for logs: • SQLite
	Database storage location for tag persistence	Specifies the storage medium for the tag persistence: • SD-X51 • USB-X61 • USB-X62
	Main database location for logging	Specifies the storage medium for logs: • SD-X51 • USB-X61 • USB-X62

See also

Start screen (Page 842)

Configuring an HMI device as an OPC UA server (Page 1060)

Languages in runtime (Page 772)

13.1.1.2 Start screen (RT Uni)

Start screen settings

In the runtime settings under "General > Screen", you define which screen of your project is to be displayed as the start screen.

The start screen is the initial screen that is displayed after runtime starts. The screen resolution is automatically adjusted to suit the HMI device when the screen is selected.

Note

Displaying a start screen changed by reloading

You have defined a start screen in the project and started runtime. If you then define another start screen in your project and load the project in the device again, the last active screen is displayed in runtime once connected again.

After reloading the project, refresh the screen in Runtime.

See also

Settings in the runtime software (Page 841)

13.1.1.3 Encrypted transfer (RT Uni)

Introduction

To enable secure loading of the runtime project, assign a password for encrypted transfer. You enter the password both in the engineering system and in runtime.

Settings for encrypted transfer

In the runtime settings under "General > Encrypted transfer", make the following settings:

- Activate encrypted transfer
- Entering and confirming a password in the engineering system
- Allow transfer of initial password via unencrypted loading

The password for encrypted transfer to the HMI device is defined on the HMI device in "Start Center" under "Service and Commissioning > Transfer".

To transfer the password unencrypted once, select the option "Allow initial password transfer via unencrypted download".

If the password assigned in the engineering system does not match the password on the HMI device, you have the option of re-assigning the password in the engineering system directly in the "Load preview" dialog.

13.1.1.4 Setting time base

Setting the time base for the time of day

You set the time in the Control Panel of your HMI device. For more detailed information, refer to the operating instructions for your HMI device.

Synchronize date and time with the PLC

You can find additional information on this subject in the online help for WinCC under "Communicating with the PLC".

13.1.1.5 Printing in Runtime

Print functions

Print functions available in online mode:

- Hardcopy
  You print out the currently displayed screen by means of an operator control that triggers the "PrintScreen" system function.
- Printing alarms
  Every alarm that has occurred and its state changes are reported along on a printer.
- Printing reports
  Reports are output in graphic mode. The use of a serial printer is not recommended because of the accumulated data volume.
  For proper output, the printer must support the paper format and page layout of the report.

Note

The value of a tag in the report is read and output at the moment of printing. A substantial time may elapse between printing out the first and the last page of a report consisting of several pages. This may lead to the same tag on the last page being output with a different value from that on the first page.

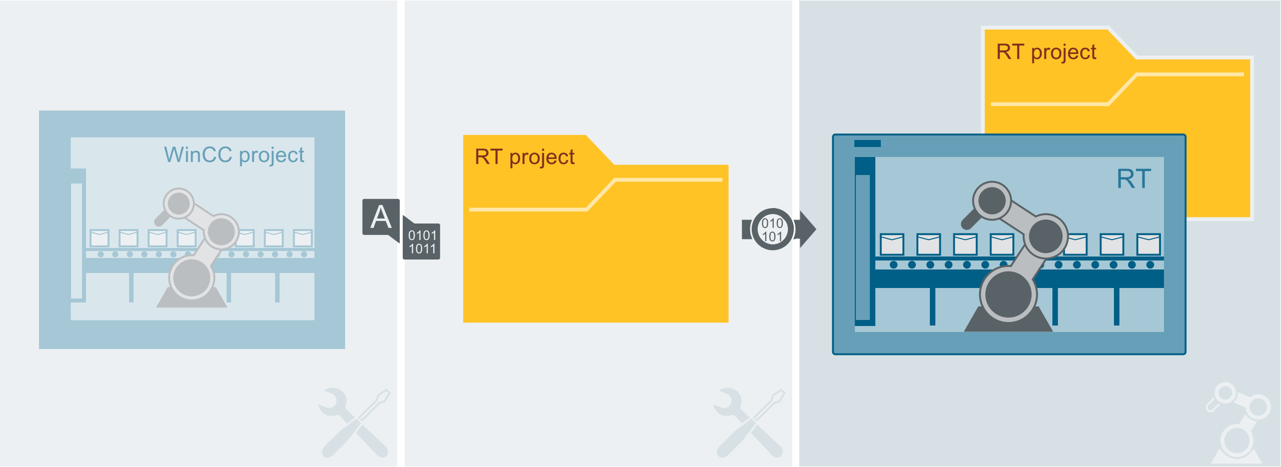
13.1.2 Overview (RT Uni)

The term "project"

The term "project" has two different meanings in the context of "compiling and downloading".

- WinCC project: Contains the configuration data of a HMI device in WinCC
- Runtime project: Contains the compiled configuration data of an HMI device.

The figure below illustrates the link between WinCC projects and runtime projects using the example of the "Compile and Download" process:



A 0101 1011 The configuration data are compiled.

010 101 The runtime project is loaded.

Definitions

To compile a project means generating a runtime project from the WinCC project.

Downloading a project means transferring the runtime project to an HMI device.

The runtime software for process visualization is running on the HMI device. In runtime, you execute the project in process mode.

Simulation

You test your configuration with a simulation. You start simulation without a connection to the running process.

In a simulation, you test configured internal tags or a screen change, for example. You simulate the project on the configuration computer.

See also

Compiling a project (Page 846)

Simulating projects (Page 847)

Downloading projects (Page 851)

13.1.3 Compiling a project (RT Uni)

Scope of the compilation

In the background, the configuration data is continuously checked for consistency and compiled.

If you compile a project manually, only the changes in the configuration made since the last compilation process are compiled in the background.

Requirement

- A project is open.
- The project contains only objects that are supported by the HMI device.

Procedure

Proceed as follows to compile a project:

1. If you want to compile several HMI devices at the same time, select all the relevant HMI devices with multiple selection in the project tree.
2. Compile the project:
   - To only compile changes in the project, select the "Compile > Software (only changes)" command from the shortcut menu of the HMI device.
   - To compile all project data, select the "Compile > Software (compile all)" command from the shortcut menu.

Result

The configuration data of all selected HMI devices is compiled. If errors occur during compilation, the errors are shown in the Inspector window.

If you have configured objects that are not supported on the HMI device, you must delete these objects before you compile again.

Please note the instructions in the Readme file supplied with Runtime Unified.

See also

Overview (Page 844)

Simulating projects (Page 847)

Downloading projects (Page 851)

13.1.4 Simulating projects (RT Uni)

13.1.4.1 Basics of simulation (RT Uni)

Introduction

You can use the simulator to test the performance of your configuration on the configuration PC. This allows you to quickly locate any logical configuration errors before productive operation.

You can start the simulator as follows:

- In the shortcut menu of the HMI device or in a screen: "Start simulation"
- Click "Start simulation" in the toolbar.
- Menu command Online > Simulation > Start
- Under "Visualization > Simulate device" in the portal view.

Field of application

You can use the simulator to test the following functions of the HMI system, for example:

- Screen change and screen navigation
- Internal tags
- Layout
- Configured alarms

See also

Simulating a project (Page 848)

Simulating a screen (Page 850)

13.1.4.2 Skip "Load preview" dialog (RT Uni)

Skip "Load preview" dialog

To permanently skip the "Load preview" dialog when simulating projects and screens, proceed as follows:

1. Open the settings under "Options > Settings".
2. Select "Simulation".
3. In the "HMI Simulation" area, clear the check box "Show 'Load preview' dialog during download to simulation".

Result

- The "Load Preview" dialog is no longer displayed.
- The simulation is opened automatically in the standard browser.

Note

Errors and warnings that occur are displayed in the Inspector window in the "Info" tab.

Note

Settings of the "Load preview" dialog

The following settings are applied from the previous loading process with displayed "Preview Load" dialog:

- Settings for keeping tag values, active alarms and user data (default value: enabled).
- Settings for resetting logs (default value: "No reset")

If the "Load preview" dialog was hidden before the first loading of the project, the default values are used.

See also

Simulating a screen (Page 850)

Simulating a project (Page 848)

13.1.4.3 Simulating a project (RT Uni)

Introduction

You simulate a project on the configuration PC and can download the simulation to the HMI device via an Ethernet connection.

Note

Simulating a project on a configuration PC while runtime is running

Runtime is terminated when a project on the HMI device is running in runtime and you use the option "Full download".

Runtime is not terminated when a project on the HMI device is running in Runtime and you use the option "Delta download". For example, tags keep their value and are not set to the start value.

Ethernet connection

You download your runtime project simulation to the HMI device via an Ethernet connection. The connection uses Ethernet port 20008.

Note

Ethernet port 20008

If an application is using Ethernet port 20008, download is not possible.

If no connection to the target can be established, check the port assignments. If another application is using Ethernet port 20008, close this application.

Requirement

- The "Simulation (SIMATIC WinCC Unified Scada)" component is installed on the configuration PC.
- The project is open in the configuration PC.
- The HMI device and the HMI device have been successfully compiled.

Procedure

Proceed as follows to simulate a project:

1. Click "Start simulation" in the toolbar.
   The "Load Preview" dialog is displayed and the compilation result is displayed.
2. Check the displayed default settings and change the settings as necessary:
   – Specify whether to use the "Full download" or "Delta download" option.
   – Specify whether runtime should start after the download.
   – When you use the "Full download" option again, you specify whether tag values, active alarms, and user data are retained. Only available if you have selected "Start runtime".
   – When you use the "Full download" option again, you specify whether all logs are reset in runtime. Only available if you have selected "Start runtime".
3. Click "Download".
4. Open the browser.
5. Call the URL *"https://localhost"* in the browser.
   Instead of the name "localhost", you can use the computer name.
6. Select "WinCC Unified RT".
7. Enter the user name and password.
   The configured screen is displayed as start screen in the browser.

8. Test, for example:
   – Screen change and screen navigation.
   – Layout
   – Internal tags.
9. To stop the simulation, select "Online > Stop runtime/simulation".

See also

Basics of simulation (Page 847)

Simulating a screen (Page 850)

13.1.4.4 Simulating a screen (RT Uni)

Introduction

If you have only made changes to one screen, you can temporarily specify this screen as the start screen for simulation. In this way, you can debug changes without having to modify the start screen, or opening the screen on the HMI device.

Requirement

You created a project that contains at least one screen.

Procedure

To define a screen as temporary start screen for simulation, follow these steps:

1. In the project tree, select the screen that is to become the temporary start screen in the simulation.
2. Select the "Start simulation" command from the shortcut menu of the screen.
   The "Load Preview" dialog is displayed and the result of the compiling is displayed.
3. Check the displayed default settings and change the settings as necessary:
   – Specify whether to use the "Full download" or "Delta download" option.
   – Specify whether runtime should start after the download.
   – When you use the "Full download" option again, you specify whether tag values, active alarms, and user data are retained. Only available if you have selected "Start runtime".
   – When you use the "Full download" option again, you specify whether all logs are reset in runtime. Only available if you have selected "Start runtime".
4. Click "Download".
5. Open the browser.
6. Call the URL *"https://localhost"* in the browser.
   Instead of the name "localhost", you can use the computer name.

7. Select "WinCC Unified RT".
8. Enter the user name and password.
   The simulated screen is displayed.

Result

If "Start runtime" is selected in the settings for loading, the screen selected in the project tree is displayed in the simulation window instead of the configured start screen.

See also

Basics of simulation (Page 847)

Simulating a project (Page 848)

13.1.5 Downloading projects (RT Uni)

13.1.5.1 Overview for loading of projects (RT Uni)

Overview

The project is automatically compiled before you download it to an HMI device. This always ensures that the latest version of the project is transferred.

If you are using external HMI tags in your project that are connected to controller tags, you should also compile the user program before you compile the HMI device.

Loading a project to an HMI device

The following steps are completed prior to downloading:

1. The download settings are verified. The "Extended download to device" dialog box opens automatically during the initial download of a project to an HMI device. You use this dialog to define the protocol and interface or destination path for the project in accordance with the HMI device Runtime used.
   If the HMI device is part of a subnet, for example, you also select the subnet and the first gateway.
   You can open the "Extended download" dialog at any time with the menu command "Online > Extended download to device...".
   The "Load preview" dialog opens.
2. The project is compiled. Warnings and errors during compilation are displayed in the Inspector window and in the "Load preview" dialog.
3. The "Load preview" dialog shows you the following information for each HMI device:
   - The individual steps for loading
   - Check of the Runtime version of the target HMI device
     A warning is displayed if the WinCC Unified Runtime version installed on the target device does not match the configured device version.
   - Default settings that take effect when loading
     You can change the default settings for this download process, if necessary.
   - Occurring warnings (optional).
     You can download a project while ignoring the "warnings". The functionality may be restricted in runtime.
   - Occurring errors (optional).
     You cannot load the project. WinCC will open the invalid configuration in the corresponding editor if you double-click the error message in the Inspector window. Correct the errors and reload the project.

Note

Interruption of a download

If the download is interrupted, WinCC automatically ensures that no data is lost and that existing data is deleted on the HMI device only after complete transmission.

Loading a project without a connected HMI device

If you cannot establish a direct connection from the configuration PC to the HMI device, copy the compiled project to the HMI device via a storage medium.

Loading with S7 routing

Configure the S7 routing settings in the "Devices & Networks" editor in the relevant PLC. The settings depend on the device configured.

S7 routing supports the following protocols:

- MPI/PROFIBUS
- PN/IE

Transferring Runtime add-ons in WinCC

Projects can contain Runtime add-ons in the form of controls or CSP (Communication Support Packages). These Runtime add-ons are automatically transferred with the project.

See also

Loading a project (Page 853)

Load project from external storage medium (Page 859)

Compiling a project (Page 846)

Simulating projects (Page 847)

Compiling and loading with Multiuser Engineering (Page 862)

13.1.5.2 Loading a project (RT Uni)

Introduction

Before a project can run on an HMI device, you must first load it to the HMI device. During loading, you must specify, in particular, whether existing data on the HMI device such as "User administration" and "Parameter sets" is to be overwritten.

The HMI device name entered in the project tree is used for PROFINET communication. The use of the name corresponds to the default settings of the PROFINET interface of the HMI device. For devices with more than one PROFINET interface, the name of the IE CP is automatically added to the device name with a separating period. The name is written to the HMI device during download. If a device name for the PROFINET communication has already been entered in the HMI device, it will be overwritten.

You can find additional information about these settings in the information system in the "Assigning a device name and IP address" section.

If the device version of the target HMI device does not match the configured device version, you are asked whether you wish to change the device version at this time.

NOTICE

Changing the device version deletes all data on the HMI device.

Data is deleted on the target system if you change the device version. For this reason, you should first back up the following data:

- User administration
- Parameter sets
- Licenses

Resetting to factory settings also deletes the license keys. Back up the license keys before you reset the system to factory settings.

For Unified Comfort devices, licenses are only deleted during resetting to factory settings. Back up the licenses and license keys before you reset the system to factory settings.

Note

Overwriting existing data during loading

During loading, you must specify whether existing data on the HMI device is to be overwritten. By selecting the check boxes in the "Load preview" dialog you can always overwrite the following data during loading:

- Existing data belonging to user administration on the device
- Existing data of the parameter sets on the device

Note

Downloading the project to an HMI device while Runtime is being executed.

Runtime is closed when a project is running in Runtime on the HMI device and you load a project to this HMI device with the command "Download to device > Software (all)".

Runtime is not terminated when a project is running in Runtime on the HMI device and you load a project to this HMI device with the command "Download to device > Software (only changes)". For example, tags keep their value and are not set to the start value.

Project identification

At the start of the configuration, each project receives a project identification which is transferred to the HMI device during loading. If you have already downloaded a project, the download process recognizes the project using the project identification. If the HMI device name is changed in the configuration, the project identification also changes.

Note

Existing runtime projects on the target device

If a runtime project with the same project identification is already available on the HMI device, the project is overwritten. In this case, the options "Software (only changes)" and "Software (all)" are available for loading.

If a runtime project with a new project identification is downloaded to the HMI device, the existing runtime project is replaced by the new project. In this case, only the option "Software (all)" is available for the download.

In both cases, existing project runtime data are overwritten on the HMI device.

Save relevant data before the download.

Controlling the transfer behavior on the HMI device

As a general rule, only one project can active in runtime on an HMI device. An HMI device is generally configured to exit Runtime automatically when loading is started. If this is not the case, you will have to exit runtime manually on the HMI device.

You define how the HMI device reacts when the project is loaded in the "Start Center" under "Settings" on the HMI device:

Transfer mode	Effect
Off	The project cannot be loaded to the HMI device.
Manually	The project can only be loaded to the HMI device if the following requirements are met: ● Runtime is not running ● The HMI device is in "Transfer" mode.
Automatic	The project can always be loaded to the HMI device. If a transfer is started on the configuration PC and a project is in runtime on the HMI device, the running project is automatically closed.

Note

Disable automatic transfer after commissioning

After the commissioning phase, disable the automatic transfer function to prevent the HMI device from switching inadvertently to transfer mode.

Transfer mode can trigger unwanted responses in the plant.

In order to restrict access to the transfer settings and thus avoid unauthorized changes, enter a password in the "Start Center".

Please refer to the documentation for the HMI device used for more detailed information on transfer settings.

Requirement

- You have created an HMI device in the project.
- The HMI device is connected to the configuration PC.
- The "Start Center" has been started on the HMI device.
- The protocol by which the project is loaded is set on the HMI device in the "Start Center" under "Settings".
- Transfer mode is set as "Automatically" or "Manually" in the HMI device.

Procedure

Proceed as follows to load a project:

1. To download a project simultaneously to several HMI devices, select the HMI devices by means of multiple selection in the project tree.
2. Select one of the following commands from the shortcut menu:
   - "Download to device > Software (only changes)".
   - "Download to device > Software (all)"
3. If the "Extended loading" dialog is open, configure the "Settings for loading". Make sure that the "Settings for loading" correspond to the "Transfer settings in the HMI device".
   - Select the protocol used, for example, Ethernet or HTTP.
   - Configure the relevant interface parameters on the configuration PC.
   - Make any interface-specific or protocol-specific settings required in the HMI device.
   - Click "Download".

   You can open the "Extended download" dialog at any time with the menu command "Online > Extended download to device...".
   The "Load Preview" dialog opens. The project is compiled at the same time. The result is displayed in the "Load Preview" dialog.
4. Check the displayed presettings and change them as necessary.
5. Click "Download".

Result

The project with the runtime add-ons it contains is downloaded to the selected HMI devices.

During the download, you can keep track of the files that are transferred.

If errors or warnings occur during the download, corresponding alarms are displayed under "Info > Load" in the Inspector window.

On completion of the successful download of the project, you can execute it on the HMI device.

Note

If the transfer is interrupted, WinCC automatically ensures that no data is lost and that existing data is deleted on the HMI device only after complete transmission.

See also

Updating the operating system on the HMI device (Page 896)

Error messages during loading of projects (Page 865)

Overview for loading of projects (Page 851)

Load project from external storage medium (Page 859)

13.1.5.3 Using external storage medium (RT Uni)

Loading project to external storage medium (RT Uni)

Introduction

If you cannot establish a direct connection from the configuration PC to the HMI device, load the compiled runtime project onto an external storage medium. For example, use a USB stick or SD card.

You load either the complete runtime project or only changes of a runtime project. To load changes, use the "Delta download" option.

As soon as you have connected the external storage medium to your HMI device, load the project on your HMI device.

Requirement

- An HMI device has been created.

Procedure

To create an external storage medium and load a project onto the storage medium, proceed as follows:

1. Jump to the "Devices" tab in the project tree.
2. Double-click "Add user-defined card reader" in the "Card reader/USB storage" folder.
   A selection dialog opens.
3. Select a target directory to save the project.

4. Drag and drop the folder of the HMI device (e.g. "HMI_1 [<Device type>]") to the added folder. Alternatively, use copy and paste.
   The project is checked. If the project has contents that have not yet been compiled, a compile is performed.
   The "Load Preview" dialog opens.

   Note

   If a runtime project with the same project identification already exists in the target directory, only the options "Full download" and "Delta download" are available for download.

5. In the selection menu, specify how your project is to be loaded:
   – "Full download"
   – "Delta download"

   Note

   Only one runtime project at a time with the same project identification can be saved in the target directory via the option "Full download" and "Delta download".

   If a runtime project already exists, it is overwritten during loading.

   Save relevant data before the download.

6. Click "Load" to confirm.

Result

Your project is stored as a compressed ZIP folder in the directory "[<Target directory>]\Simatic.HMI\RT_Projects" :

- Projects that were created with the option "Full download" receive as file name e.g. "[<Project_name>].PC-System_1[SIMATIC PC station - WinCC Unified Scada RT]_full.zip".
- Projects that were created with the option "Delta download" receive as file name e.g. "[<Project_name>].PC-System_1[SIMATIC PC station - WinCC Unified Scada RT]_delta.zip".

Load project from external storage medium (RT Uni)

Introduction

If you cannot establish a direct connection from the configuration PC to the HMI device, copy the compiled project to an external storage medium, such as a USB flash drive, and then load it onto the HMI device.

NOTICE
Data loss
When you activate the "Firmware upgrade" or "Firmware downgrade" options, then the operating system of the HMI device is updated. Existing data on the HMI device including the HMI device password is deleted. Settings in the Start Center are retained, license keys are saved to the external storage medium before update of the operating system.
If required, save this data before loading.

You create the required project data in WinCC by configuring the HMI device and then dragging and dropping or copying and pasting the folder of the HMI device (e.g. "HMI_1 [<*device type*>]") to an external storage medium under "Card Reader/USB memory".

Requirements

- You have started the Start Center on the HMI device.
- The storage medium with the backed up project is inserted in the HMI device.

Procedure

1. Select "Service & Commissioning > Load project from storage".
2. Select a storage medium under "Select storage media for OS update".

   Note

   If there is no storage medium or a defective storage medium in the HMI device, the "0 devices found" message is displayed. Insert the storage medium or replace the storage medium.

   An overview of all projects which are compatible with the HMI device and are located on the storage medium is displayed.
3. Select the project that you want to load into the HMI device.

   Note

   Use "Details" to receive additional information about the selected project.
4. Press "Load Project".

5. Press "Next".
   The HMI device checks whether the project data can be loaded. The result of the check is displayed in the "Load Preview" dialog.
   The project can be loaded to the HMI device if no alarms of the type "Error" are issued.

   Note

   If a downgrade or upgrade is necessary, the "Update OS Image" dialog informs you about the possible loss of data and gives you further instructions.

6. Select "Load", to transfer the project data to the HMI device with the project data.

After the loading process the new project is started on the HMI device.

Alarms in the "Load Preview" dialog

The following messages can be displayed in the "Load Preview" dialog:

- Alarms of the type "Information":
- Alarms of the type "Warning", with options
- Alarms of the type "Error", with options

The following table shows alarms of the type "Information":

Icon	Status	Alarm	Meaning
✓	Info	Firmware version ... Runtime version ...	Firmware and Runtime version on the HMI device
✓	Info	Ready For Loading	Project data is suitable for the HMI device

The following table shows alarms of the type "Warning", with options:

Icon	Status	Alarm	Meaning
⚠	Overwrite	Select project data	The following lines contain options for overwriting data on the HMI device.
□		Parameter sets	Overwrite parameter sets of the HMI device with the parameter sets of the project, optional.
□		User administration data	Overwrite the user administration on the HMI device with the user administration of the project (optional).
□⚠	Upgrade	Runtime upgrade	Runtime version on the HMI device is older than the Runtime version of the project, versions are compatible, upgrade of Runtime version on the HMI device is optional.
□⚠	Upgrade	Firmware upgrade	Firmware version on the HMI device is older than the firmware version of the project, versions are compatible, upgrade of firmware on the HMI device is optional.

Icon	Status	Alarm	Meaning
□ ⚠	Downgrade	Runtime downgrade	Runtime version on the HMI device is newer than the Runtime version of the project, versions are compatible, downgrade of Runtime version on the HMI device is optional.
□ ⚠	Downgrade	Firmware downgrade	Firmware version on the HMI device is newer than the firmware version of the project, versions are compatible, downgrade of firmware on the HMI device is optional.

The following table shows alarms of the type "Error", with options:

Icon	Status	Alarm	Meaning
□ ⊗	Upgrade	Runtime upgrade	Runtime version on the HMI device is older than the Runtime version of the project, versions are incompatible, upgrade of Runtime version on the HMI device is required.
□ ⊗	Upgrade	Firmware upgrade	Firmware version on the HMI device is older than the firmware version of the project, versions are incompatible, upgrade of firmware on the HMI device is required.
□ ⊗	Downgrade	Runtime downgrade	Runtime version on the HMI device is newer than the Runtime version of the project, versions are incompatible, downgrade of Runtime version on the HMI device is required.
□ ⊗	Downgrade	Firmware downgrade	Firmware version on the HMI device is newer than the firmware version of the project, versions are incompatible, downgrade of firmware on the HMI device is optional.
□ ⊗	Download	Runtime download	There is no Runtime software on the HMI device, e.g. after update of the operating system. Runtime software must be downloaded.

Note

When loading projects to your HMI device, please note that there are compatible and incompatible firmware and Runtime versions.

If you are loading a compatible firmware and Runtime version of the project to your HMI device, an upgrade or downgrade is optional. You can download the project to the HMI device without an upgrade or downgrade. In this case, you can ignore the alarm of the type "Warning".

If you are loading an incompatible firmware and Runtime version of the project to your HMI device, an upgrade or downgrade is mandatory. Otherwise, you cannot download the project to the HMI device. An alarm of the type "Error" is displayed:

Note

Alarms of the type "Warning" also appear in the event of a potential overwriting of user data and parameter sets if this setting was not disabled in the configuration.

See also

Overview for loading of projects (Page 851)

Loading a project (Page 853)

Updating the operating system of the HMI device from a data carrier (Page 898)

Error messages during loading of projects (Page 865)

13.1.6 Compiling and loading with Multiuser Engineering (RT Uni)

13.1.6.1 Compiling and loading with multiuser engineering (overview) (RT Uni)

Introduction

When using multiuser engineering for your projects, you should take into account the response when compiling the Runtime projects and the response when downloading them to HMI devices.

You can compile and download to an HMI device in both the server project view and in the local session.

You can find more information about the topic of "multiuser engineering" under "Using Multiuser Engineering".

Basics

The following scenarios are possible for Unified Comfort Panels in multiuser engineering:

- Compiling in the server project view
- Compiling in the local session
- Loading from the server project view
- Loading from the local session

Note

Complete download from the server project view or local session is no different to complete download in a single-user project. With a complete download, the current Runtime project is loaded from the currently active view to an HMI device.

Note

Compiling and downloading in a local session is no different from compiling and downloading in a single-user project.

In principle, you can execute all commands for compiling and loading in multiuser engineering projects:

- "Software (rebuild all)"
- "Compile > Software (only changes)"
- Software (all)

The term "project"

The term "project" has two different meanings in the contexts of compilation and loading. "Project" is the WinCC project on the configuration PC. "Project" is also the Runtime project you create by compiling the configuration data of an HMI device and download to the HMI device.

- WinCC project: contains the configuration data of one or more HMI devices
- Runtime project: contains the compiled configuration data of one HMI device

Rules

The following basic rules apply to compiling and downloading in multiuser engineering:

- The Runtime project that has been compiled in a local session always remains local and is not uploaded to the multiuser server. It cannot be saved in the multiuser server project.
- Only Runtime projects compiled in the server project view can be saved in the multiuser server project.

You can find additional information on Multiuser Engineering on the Siemens YouTube channel: Multiuser Engineering - one team working simultaneously on a project (https://www.youtube.com/watch?v=n4oTZ2Gzg6U).

See also

Compiling in the server project view (Page 863)

Compiling in the local session (Page 864)

Downloading projects (Page 851)

13.1.6.2 Compiling in the server project view (RT Uni)

Basics

Compiling and downloading in the server project view is no different from compiling and downloading in a single-user project.

During the compiling of a project in the server project view, the multiuser server project is blocked. Other users cannot make changes to this server project during this time. The Runtime project compiled in the server project view is stored along with the engineering project in the central multiuser server. Blocking the multiuser server project ensures that the configuration data and the Runtime project remain in sync.

Note

When you compile and save in the server project view, other users obtain the Runtime project you have updated along with the engineering project when they "refresh" their local session. Other users do not have to recompile the changes you have made after an update.

Example: Compiling during check-in

You make changes to a tag in a local session. All prior changes have been compiled in the associated server project.

If there are no compilation errors, both projects - the modified engineering project (with the modified tags) and the compiled Runtime project - are saved in the central multiuser server project with the "Save changes" command.

If you skip compiling during the check-in, the project contains the changes that have been saved on the server.

The next user who creates a local session from the server project or updates an existing local session must compile your two changes in addition to his or her own changes.

Note

Working on a shared project through multiple local sessions increases the probability of error. It is therefore recommended to compile the project at check-in and eliminate any errors that are reported during compiling. In this way, you provide the next user with a project free of errors.

See also

Compiling and loading with multiuser engineering (overview) (Page 862)

Compiling in the local session (Page 864)

13.1.6.3 Compiling in the local session (RT Uni)

Basics

Compiling and downloading projects in the local session is no different from compiling and downloading in a single-user project.

Since the local session is a copy of the server project, the first compilation status of the local session is identical to that of the server project. If the server project contains contents that are not compiled or error messages occurred during the compiling, they are transferred to the local session.

Note

It is recommended to compile the project at check-in and eliminate any errors that are reported during compiling. In this way, you provide the next user with a project free of errors and avoid spreading errors.

Updating in the local session

If you update a project in the local session, the local session - including the compilation status - is completely replaced by the content of the server project. Only the changes marked for check-in are retained in the updated local session and generate additional compiling steps in the local session.

Example: Updating the local session

You make changes to a tag in a local session. All prior changes have been compiled in the associated server project.

You update the content of the local session by clicking the "Update" button. After the update, the local session obtains the compilation status of the server project. There are also compiling tasks for the acquisition of the modified tags.

See also

Compiling and loading with multiuser engineering (overview) (Page 862)

Compiling in the server project view (Page 863)

13.1.7 Error messages during loading of projects (RT Uni)

Possible problems during the download

When a project is being downloaded to the HMI device, status messages regarding the download progress are displayed in the output window.

Problems arising during the download of the project to the HMI device are usually caused by one of the following errors:

- Wrong operating system version on the HMI device
- Incorrect download settings on the HMI device
- Incorrect HMI device type in the project
- The HMI device is not connected to the configuration PC.

The most common download failures and possible causes and remedies are listed below.

The download is cancelled due to a compatibility conflict

Possible cause	Remedy
Conflict between versions of the configuration software and the operating system of the HMI device	Synchronize the operating system of the HMI device with the version of the configuration software. To update the operating system on the HMI device, select the "Update operating system" command from the "Online > HMI device maintenance" menu in WinCC. You can also use ProSave. For additional information, refer to the operating instructions for the HMI device.
The configuration PC is connected to the wrong device, e.g. a controller.	Check the cabling. Correct the communication parameters.

Project download fails

Possible cause	Remedy
Connection to the HMI device cannot be established (alarm in the output window)	Check the physical connection between the configuration PC and the HMI device. Check whether the HMI device is in transfer mode. Exception: Remote control

The configuration is too complex

Possible cause	Remedy
The configuration contains too many different objects or options for the HMI device selected.	Remove all objects of a specific type, for example all HTML browsers. Alternatively, remove options such as Sm@rtServer or OPC server. Reduce the project size.

See also

Reducing the project size (Page 866)

13.1.8 Reducing the project size (RT Uni)

Introduction

When loading a large-scale project to an HMI device, the memory of the HMI device may be insufficient for the project. There are several ways to reduce the size of your project.

Options for reducing the project size

There are several ways to reduce the size of the project and save space:

- Reduce the number of available runtime languages
  Check whether all selected runtime languages are actually needed. You can disable the languages that you do not need under "Runtime settings > Language & Font > Runtime language and font selection".
- Do not use help texts for S7 diagnostic alarms
  To reduce the size of the project, you can disable the download of help texts for the S7 diagnostic alarms. In order to avoid downloading the help texts to the HMI device, disable the option "Download S7 diagnostics help texts" under "Runtime settings > Alarms > General".
- Rebuild all software
  In order to optimize the project data and to clean up obsolete changes, compile the entire project using the "Compile > Software (rebuild all)" command from the shortcut menu of the HMI device.
- Harmonize the presentation using styles
  It is recommended to harmonize screen objects using styles. Standardize the appearance of screen objects in a project to optimize project data. Use the specified preset or customized style for the configuration of the screen objects throughout the project.
- Activate automatic updating of PLC alarms
  To save space, you can specify that the PLC alarm texts are only to be loaded in runtime when needed. To do this, enable the "Automatic update" option under "Runtime settings > Alarms > Controller alarms". Make sure that automatic update of alarms is also enabled in the corresponding controller.
  The "Automatic update" option is not available on Basic Panels.
  The amount of space that can be saved depends on the number of PLC alarms and the number of runtime languages.
- Reduce the number of fonts loaded
  Check whether the number of downloaded user-defined fonts can be reduced. If necessary, configure only the standard fonts for the required Runtime languages under "Runtime settings > Language & font > Runtime language and font selection".
  To save space, use fewer font groups for the configuration.
- Reduce the size of the graphics
  Check the size of the graphics that you use in the project. If necessary, reduce the size of the graphics by reducing the resolution or choose a higher compression format without noticeable loss of quality for the project graphics.
  To keep the project size small for Basic HMI devices, harmonize the sizes of graphics used in the project.
  Select appropriate graphic formats for your screens: For example, use PNG images for drawings that are not vector graphics and JPEG images for photos.

See also

Loading a project (Page 853)

13.1.9 Starting runtime (RT Uni)

Introduction

You can start the project in runtime as soon as you have downloaded the project to the HMI device. The project is generally started automatically on the HMI device.

The project settings defined in the "Runtime settings" of the HMI device are activated when the project is started in Runtime. Make sure when defining the Runtime settings "Lock task switching" and "Full screen" that you will be able to stop Runtime again. You can, for example, configure a button with the system function "StopRuntime".

Note

Response of Runtime when the HMI device is restarted

When the HMI device is restarted, the project is automatically restarted even if the project was stopped before the restart.

Note

Closing runtime automatically

If automatic transfer is activated on the HMI device and a transfer is started on the configuration PC, the running project is automatically terminated.

The HMI device then automatically switches to "Transfer" operating mode.

After the commissioning phase, disable the automatic transfer function to prevent the HMI device from switching inadvertently to transfer mode.

Transfer mode can trigger unwanted responses in the plant.

In order to restrict access to the transfer settings and thus avoid unauthorized changes, enter a password in the "Start Center".

Note

Using encrypted communication connections in runtime

A runtime project with encrypted communication connections to S7 PLCs must be loaded for each Windows user who wants to use these connections in runtime.

Requirement

- The project was downloaded to the HMI device.
- The "Start Center" has been started.

Starting runtime on an HMI device

On an HMI device, the project is stored in the folder specified in the transfer settings of the HMI device. The "Start Center" is displayed when the HMI device is switched on. Depending on the configuration, the loaded project starts automatically after the defined delay time.

If the project does not start automatically, click "Start" in the "Start Center".

Refer to the documentation for the HMI device for additional information on startup of projects.

Result

Runtime is started on the HMI device.

13.1.10 Adapting the project for another HMI device (RT Uni)

Introduction

When you download a WinCC project to an HMI device, WinCC checks whether the HMI device is compatible with the HMI device type used in the project. If the types of HMI device do not match, you will see a message before the download starts.

The download is aborted.

Adapting the project for the HMI device

You need to adapt the project accordingly to be able to download the project to the connected HMI device.

- Add a new HMI device in the project tree. Select the correct type of HMI device from the HMI device selection.
- Copy the configured components from the previous to the new HMI device.
  You copy many components directly in the project tree and the details view.
  For example, copy the "Screens" folder to the screens folder of the new HMI device using the shortcut menu.
- Use the detail view to copy entries in the project tree for which the "Copy" command is not available in the shortcut menu.
- For example, select the "Parameter sets" entry in the project tree. The parameter sets are displayed in the detail view.
- Select the parameter sets in the detail view and drag them to the "Parameter sets" entry of the new HMI device. The parameter sets are copied. You can also select multiple objects in the detail view.
- Configure the components that cannot be copied, e.g. connections, area pointers, and alarms.
- Save the project at various points in time.
- Compile the full project.
- When the compilation is successfully completed, download the project to the HMI device.

Linking references

References to linked objects are included in the copying. The references are once again linked to each other after the linked objects are copied.

Example:

You copy a screen in which objects are linked to tags. The tag names are entered at the individual objects after the screen is added to the new HMI device. The tag names are marked in red because the references are open. When you then copy the tags and insert them into the new HMI device, the open references are closed. The red marking for the tag names disappears.

To complete references to connected objects in the controller, you first need to configure a connection to the controller.

Using the information area

When you compile the project for the new HMI device, errors and warnings are displayed in the "Info" tab of the Inspector window. You can use the shortcut menu command "Go to" to go directly to the location where the error or warning can be corrected.

Work through the list of errors and warnings from top to bottom.

When the compilation is successfully completed, download the project to the HMI device.

13.1.11 Users in runtime (RT Uni)

The operating instructions of the HMI device contains information about users and user administration on your HMI device.

13.1.12 Viewing memory card data (RT Uni)

13.1.12.1 Basics (RT Uni)

Introduction

WinCC provides you with the possibility of viewing data stored on your memory card. The function supports the use of memory cards of the HMI device and of the CPU.

You have the following options:

Viewing a backup (Page 871)

Renaming and deleting backups (Page 872)

Viewing HMI device images (Page 873)

Deleting HMI device images (Page 874)

Creating HMI device images on memory card (Page 875)

See also

Viewing a backup (Page 871)

Renaming and deleting backups (Page 872)

Viewing HMI device images (Page 873)

Deleting HMI device images (Page 874)

Creating HMI device images on memory card (Page 875)

13.1.12.2 Working with backups (RT Uni)

Viewing a backup (RT Uni)

Introduction

If you have stored the backup of an HMI device on a memory card, this backup can also be viewed in the TIA Portal.

Requirements

- WinCC is installed.
- A memory card with a backup is available.
- The card reader is connected to the configuration PC.
- The project view is open.

Backup on the memory card in the card reader

1. Insert the memory card into the card reader.
2. Open "Card Reader/USB storage" in the project tree.
3. Select the card reader drive.
   The "Online card data" folder is displayed.
4. Open the "Online card data" folder.
5. Click the backup to open the shortcut menu.
6. Select "Properties".

Backup on the memory card of the PLC

Proceed as follows if the backup is stored on the memory card of the PLC:

1. Connect the PLC with the configuration PC.
2. Click on the PLC in the project navigation.

3. Select "Connect online" from the shortcut menu.
   A connection to the PLC is established.
   Once the PLC is connected, the "Online card data" folder is displayed.
4. Open the "Online card data" folder.

   Note

   Accessing a password-protected PLC

   When you attempt to access a PLC that is protected by a password, you will be prompted to enter the password.

   You need at least read access rights in order to view the data that is stored on the memory card.

5. Click the backup to open the shortcut menu.
6. Select "Properties".

Result

The backup properties are displayed in a separate dialog:

- General properties
  - Date and time when the backup was created
  - Software version with which the backup was created.
- Supported HMI devices with which the backup is compatible

See also

Renaming and deleting backups (Page 872)

Renaming and deleting backups (RT Uni)

Introduction

You can rename and delete backups from a memory card in the project navigation of the TIA Portal.

Requirements

- WinCC is installed.
- The card reader is connected to the configuration PC.
  Or The PLC is connected online with the configuration PC.
- A memory card with a backup is available.

- The project view is open.
- The backup is displayed in the project navigation.

> **Note**
>
> **Accessing a password-protected PLC**
>
> When you attempt to access a PLC that is protected by a password, you will be prompted to enter the password.
>
> You need write access rights to rename or delete memory card data.

Procedure

1. Click on the backup in the project navigation.
2. Open the shortcut menu.
3. Select "Rename" to rename the file.
4. Enter a new name.
5. Select "Delete" to delete the file.

Result

The backup file is now renamed or deleted.

See also

Viewing a backup (Page 871)

13.1.13 Working with HMI device images (RT Uni)

13.1.13.1 Viewing HMI device images (RT Uni)

Introduction

If you have saved the HMI device image of a Unified Comfort Panel on a memory card, you can display the properties of the HMI device image in the TIA Portal.

Requirements

- WinCC is installed.
- The card reader is connected to the configuration PC.
- A memory card with the HMI device image is available.
- The project view is open.

Procedure

1. Insert the memory card into the card reader.
2. Open "Card Reader/USB storage" in the project tree.
3. Select the card reader drive.
   The "Online card data" folder is displayed.
4. Open the "Online card data" folder.
   The available images of the HMI device are displayed in additional folders.
5. Click the required HMI device image.
6. Select "Properties" in the shortcut menu.

Result

The properties of the HMI device image are displayed in a separate dialog:

- General properties
  - Date and time when the HMI device image was created
  - Software version with which the HMI device image was created
- Supported HMI devices with which the HMI device image is compatible

See also

Deleting HMI device images (Page 874)

Creating HMI device images on memory card (Page 875)

13.1.13.2 Deleting HMI device images (RT Uni)

Introduction

You can delete the HMI device image of a Unified Comfort Panel from a memory card in the project navigation of the TIA Portal.

Requirements

- WinCC is installed.
- The card reader is connected to the configuration PC.
- A memory card with an HMI device image is available.
- The project view is open.
- The HMI device image is displayed in the project navigation.

Procedure

1. Click the HMI device image in the project navigation.
2. Open the shortcut menu.
3. Select "Delete" to delete the file.

Result

The HMI device image is deleted.

See also

Viewing HMI device images (Page 873)

Creating HMI device images on memory card (Page 875)

13.1.13.3 Creating HMI device images on memory card (RT Uni)

Introduction

As an alternative to directly transferring an HMI device image of from a configuration PC to a Unified Comfort Panel, you can create an HMI device image on a memory card or USB flash drive and later transfer the HMI device image to the Unified Comfort Panel.

Requirements

- WinCC is installed.
- The card reader is connected to the configuration PC.
- The project view is open.

Procedure

1. Insert the memory card into the card reader.
2. Click on the memory card in the project navigation.
3. Open the shortcut menu.
4. Select the entry "Card Reader/USB storage > Create HMI OS image on memory card".
   The "Select the memory map of the operating system" dialog opens.

5. Select an HMI device image in the "Images" area or under "Images from another location".
6. Select the restore settings.
   - Select "Keep installed settings of the Control Panel":
     The settings you entered on the Unified Comfort Panel are retained.
   - "Keep installed licenses" enabled:
     The licenses on the Panel are retained.
   - "Lock the settings on the panel" enabled:
     You can no longer change the selected settings for "Keep installed settings of the Control Panel" and "Keep installed licenses" on the Unified Comfort Panel.

Result

You have created an HMI device images on memory card. Alternatively, you may use a USB stick instead of the memory cards.

See also

Viewing HMI device images (Page 873)

Deleting HMI device images (Page 874)

13.1.14 Basics of operating in Unified Runtime (RT Uni)

13.1.14.1 Overview (RT Uni)

Operating options for an HMI device

The following operating options are available:

- Operation via touch screen
  The HMI device has a touch-sensitive touch screen. Use your finger or a suitable touch pen to operate the touch screen.
- Operation via mouse and keyboard
  You can use the mouse and keyboard to operate the device like a PC.

Adhere to the instructions for operating the device in the relevant operating instructions.

Individually configured operation

The configuration engineer has various options available for setting up operation.

Examples of actions whose execution is always determined on a project-specific basis:

- Screen change
- Reporting
- Changing runtime language

There are no specific operating elements to execute certain functions. The configuration engineer specifies the project-specific execution. The screen change can be triggered, for example, via a button.

Information on project-specific operations can be found in the system documentation.

13.1.14.2 Operation with the touch screen (RT Uni)

Overview of operation with the touch screen (RT Uni)

Use the touch screen to operate the HMI device of the project that is running on your HMI device.

Operating the touch screen

NOTICE

Damage to the touch screen

The following operation considerably reduces the service life of the touch screen up to total failure:

- Touching with pointed or sharp objects
- Sudden contact with hard objects. Use your finger or a suitable touch pen to operate the touch screen.

Do not touch the touch screen with sharp or pointed objects. Adhere to the instructions for the touch screen of the device in the corresponding operating instructions.

Special features when operating using the touch screen

Operation with the touch screen is characterized by the following special features:

- Enable
  To enable the operator control, use your finger or a suitable touch pen to operate the touch screen. To generate a double-click, touch the operator control twice in rapid succession.
- Value input
  You enter numbers and letters on the touch screen with a screen keyboard.

Input using the screen keyboard

The screen keyboard is displayed when you select a screen item that requires input. The screen keyboard is hidden again when input is complete.

Further information on the screen keyboard can be found in the operating instructions of the HMI device.

Placing the focus on objects (RT Uni)

You have the following options:

- Click or tap on the object.

  Note

  Giving focus to objects with a transparent background

  If an object has a transparent background, click on a visible area of the object.

- Press <Tab> until the object has the focus.

Operating objects with transparent fill (RT Uni)

The objects displayed on a screen can have transparent ranges.

Example: Sliders, bars and pointer instruments are enclosed by a transparent rectangle.

Trigger event

Requirement

An event which is triggered by operating actions such as typing or clicking has been configured for the object in the engineering.

Procedure

To trigger the event, proceed as follows:

- If the object does not have the focus, click a visible part of the object, e.g. its border.
- If the object already has the focus, the event is also triggered by clicking in the transparent area.

Using multi-touch functions (RT Uni)

Supported gestures (RT Uni)

Definition

Various touch gestures are available for the runtime operation. Some touch gestures have different effects in the process pictures than in the controls.

Note

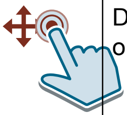
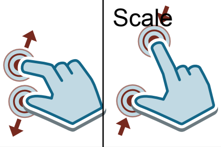
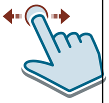
No operation with three or more fingers.

Only use one or two fingers when operating with touch gestures.

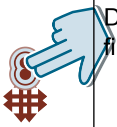
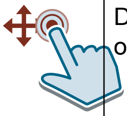
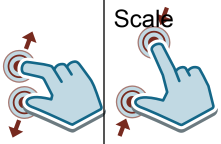
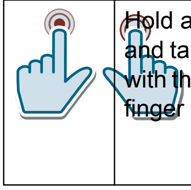
If you use more than two fingers with touch gestures, this can cause incorrect operation.

In the case of multitouch operation with several fingers, you only operate the respectively configured objects.

Supported touch gestures in process pictures

Icon	Gesture	Function
	Tap	To select an object, tip on the corresponding position in the process screen.
	Drag with one finger	To scroll horizontally or vertically, drag the process screen or the object with one finger in the desired direction. You scroll horizontal and vertical at the same time by dragging vertically in screens.
	Scale	To zoom in or zoom out, drag simultaneously with two fingers.
	Swipe	To switch between two process screens, swipe horizontally with one finger.
	Keep pressed	To call the shortcut menu, press for longer than a second on the object or the link. The function corresponds to a right-click.
	Activation gestures	To call the WinCC system dialog, swipe fast vertically from top to bottom.

Supported touch gestures in controls

Icon	Gesture	Behavior	Supported WinCC controls
	Tap	To select a row, tap the row. With corresponding configuration of the control: To select a cell.	Alarm control
	Drag with two fingers	To move table and/or trends and axes, drag with two fingers in the control window.	• Trend control • Process control • Ruler window
	Drag with one finger	To shift the X axis or Y axis. To navigate within a list or table.	• Trend control • Process control • Ruler window
		To select multiple rows, tap a row and drag your finger up or down. With corresponding configuration of the control: To select multiple cells.	Alarm control
		To adapt the column width, tap a column grid line and drag your finger to the right or left.	
		To change the order of the columns, tap a column header and drag your finger to another column header.	
	Keep pressed	To display the tooltip of the picked value or object, press for longer than a second on the value or the object. The function corresponds to a right-click.	Trend control
	Double tap	To toggle between 2 preset zoom settings. Requirement: "Zoom +/-", "Zoom time axis +/-" or "Zoom value axis +/-" was pressed.	Trend control
	Scale	To zoom in or out in the trend control, drag with two fingers in the control window.	Trend control
	Hold a finger and tap twice with the other finger	To restore a zoomed view to 100%, follow these steps: • Hold the control with one finger • Tap twice on the control with the second finger. The operation corresponds to clicking the symbol "Original view" (1:1).	Trend control

See also

Special features for multi-touch operation (Page 881)

Special features for multi-touch operation (RT Uni)

Scrolling in lists and controls

You can scroll through lists and controls by dragging.

You use horizontal dragging to move the content of the screen to the left or right. You use vertical dragging to scroll up or down in the view. An indicator appears during scrolling to indicate your position. When you drag a list diagonally, the content is moved horizontally and vertically at the same time.

You scroll around five times faster up or down on a page when you use two fingers to drag up or down.

Special features of the trend view

You enlarge or reduce the view in "Trend view" and "f(x) trend view" objects by pinch-to-zoom with two fingers.

Double tap to switch from the magnified trend view back to the normal view.

The zooming function is limited to the time axis in the "Trend view" object.

If you have enabled the option "Range > Auto-size" during configuration of the value axes in f(x) trend view, the axes are constantly calculated during zooming.

Horizontal scrolling is not supported in the "Trend view" object.

Note

Current view is not persistent

The changes of zoom factor and position changed by scrolling are not saved.

The trend view is reset to the default setting during a screen change.

See also

Supported gestures (Page 879)

13.1.14.3 Direct Keys (RT Uni)

Introduction

Direct keys on the HMI device are used to set bits in the I/O area of a SIMATIC S7.

Direct keys enable operations with short reaction times that are, for example, a jog mode requirement.

Note

Direct keys are still active when the HMI device is in "offline" mode.

Note

You can only use direct keys when coupling via PROFINET IO.

Direct keys result in additional basic load on the HMI device.

Configuring direct keys

You can configure buttons as direct keys on HMI devices with touchscreen. You can also define screen numbers. This allows the project engineer to define the direct keys for a specific screen.

More detailed information on configuring direct keys can be found at "Visualizing processes > Communicating with controllers".

13.1.14.4 Triggering an action (RT Uni)

Introduction

Triggering an action at an operator control can mean the following:

- A command is executed.
  Example: Touch a button to trigger a script or perform a predefined function.
- An object is enabled.
  Example: Touch a table cell to enter a value in a list.

Requirement

- You have navigated to the operator control on which you want to trigger the action.
- The operator control has the focus.

Procedure

- Touch the operator control on the touch screen once or twice in rapid succession.

Result

The following results are possible:

- The requested command is executed.
- The screen keyboard is opened and/or the cursor blinks in the input area of the operator control.
- The element is selected and can be moved.

For more detailed information, refer to the operating instructions for your HMI device.

13.1.14.5 Entering a value (RT Uni)

Introduction

Depending on the input format, you enter numeric or alphanumeric values in an input field using the screen keyboard.

Requirement

- The object is an input field or table field.
- The operator control is enabled.

Entering a value

1. Enter the desired value.
2. To confirm the value and exit the field, press the <Enter> key.
3. To discard the value and exit the field, press the <Esc> key.

Result

A value is entered or discarded. You navigate as needed to the next operator control.

For more detailed information, refer to the operating instructions for your HMI device.

13.1.14.6 Moving operator controls (RT Uni)

Introduction

You operate movable operator controls of a screen item in runtime via the touchscreen, such as a slider or scroll bar.

Requirement

- A movable operator control is selected.

Procedure

1. Use a corresponding gesture to move the operator control, e.g. "drag" for a slider.
2. To finish the movement, navigate to another screen object or operator control.

Result

The position of the movable operator control and the display in the screen object have changed.

For more detailed information, refer to the operating instructions for your HMI device.

13.1.14.7 Displaying infotext (RT Uni)

Introduction

Depending on the configuration, additional information and operating instructions are available as infotext. The infotext is assigned to an operating element, an alarm or to the open screen. The infotext of an I/O field may contain, for example, information about the value to be entered.

Requirement

- An infotext is configured for the operating element, the screen or an alarm.

Calling the infotext

1. Enable the desired operating element.
2. Press the <Help> button of the screen keyboard.
   The infotext for the operating element is displayed.

If there is no infotext for the selected screen object, the infotext for the current screen is displayed, if it has been configured.

Use the scroll bar for long infotexts.

Depending on the configuration, info text can also be retrieved by means of a configured operating element.

Switching between infotexts

- To switch between the infotexts of the operating elements and the screen, enable the infotext window.

Hiding infotext

- To hide the infotext, press the <Esc> key or press the <Help> key again.

13.1.14.8 Changing Runtime language (RT Uni)

Introduction

The HMI device supports multilingual projects. A corresponding operating element which lets you change the language setting on the HMI device in Runtime has been configured.

The project always starts with the language set in the previous session.

Requirement

- The required language for the project is available on the HMI device.
- The language switching function is linked to an operating element, for example, to a button.

Selecting a language

You can change project languages at any time. Language-specific objects are immediately displayed on the screen in the new language when you switch languages.

You can switch the language in Runtime in one of the following ways:

- Use a configured operating element to switch from one language to the next in a list.
- Use a configured operating element to directly set the required language.

13.1.14.9 Web browser of WebKit engine (RT Uni)

Introduction

If the "Browser" object is configured for an HMI device, then the "Browser" operating object is displayed in the corresponding runtime screen. Only the web browser of the WebKit engine is available on HMI devices. This web browser offers many HTML5 features, but no Active X support.

HTML5 functions

The following HTML5 standard functions are fully or partly supported by the Web browser of the WebKit engine:

- Parsing rules
- Elements
- Forms and fields
- Output
- Communication
- User interactions
- Performance
- Security

- History and Navigation
- 2D graphics
- Memory
- Animations
- Web applications
- Files and file systems

Note

Microdata, input, peer to peer, position and orientation, video, audio, responsive images, 3D graphics, streams and web components are not supported in the Web browser of the WebKit engine.

The table below shows the availability of the HTML5 functions in the Web browser of the WebKit engine in detail:

Parsing rules	Available
<!DOCTYPE html> triggers the standard mode	Yes
HTML5 tokenizer	Yes
HTML 5 tree building	Yes
Parsing Inline SVG	Yes
Parsing Inline MathML	Yes

Elements	Available
Embedded invisible data	Yes
New or modified elements	
Section elements	Yes
Grouping content elements that belong together	Partly
Semantic elements of the text level	Partly
Interactive elements	Partly
Global attributes and methods	
Hidden attributes	Yes
Inserting dynamic markups	Yes

Forms and fields	Available
Field types	
type = text	Yes
type = search	Yes
type = tel	Yes
type = URL	Yes
type = email	Yes
type = date	No

Forms and fields	Available
type = month	No
type = week	No
type = time	No
type = datetime	No
type = datetime-local	No
type = number	Yes
type = range	Yes
type = color	Yes
type = checkbox	Yes
type = image	Yes
type = file	Yes
textarea	Yes
select	Yes
fieldset	Yes
datalist	Yes
keygen	Yes
output	Yes
progress	Yes
meter	Yes
Fields	
Field validation	Yes
Assignment of forms and controls	Yes
Other attributes	Yes
CSS sectors	Yes
Events	Yes
Forms	
Form validation	Yes

Output	Available
Full-screen support	No
Web notifications	Yes

Communication	Available
Server-sent events	Yes
Web beacons	No
XML HttpRequest Level 2	
File upload	Yes
Response type	Yes
WebSocket	
Basic Socket Communication	Yes
ArrayBuffer and Blob	Yes

User interactions	Available
Drag-and-drop	
Attributes	Yes
Events	Yes
Editing HTML	
Editing elements	Yes
Editing documents	Yes
CSS sectors	No
APIs	Yes
Clipboard	
Clipboard for API and events	No
Spell check	
Spelling attributes	Yes

Performance	Available
Native binary data	Yes
Workers	
Web workers	Yes
Shared workers	Yes

Security	Available
Web Cryptography API	No
Content Security Policy 1.0	Yes
Content Security Policy 1.1	No
Cross-Origin Resource Sharing	Yes
Cross-Document Messaging	Yes
Iframes	
Sandboxes Iframe	Yes
Seamless Iframe	Yes
Iframe with Inline contents	Yes

History and Navigation	Available
Session history	Yes

2D graphics	Available
Canvas 2D graphics	Yes
2D primitives	
Text input in graphics	Yes
Path input in graphics	No
Drawing an ellipse in graphics	No

2D graphics	Available
Drawing a dashed line in graphics	Yes
System focus ring	No
Functions	
Hit testing	No
Aperture mode	No
Formats for image export	
PNG	Yes
JPEG	Yes
JPEG-XR	No
WebP	No

Animation	Available
window.requestAnimationFrame	Yes

Web applications	Available
Offline resources	
Application cache	Yes
Service workers	No
Content and scheme handlers	No

Memory	Available
Key value storage	
Session memory	Yes
Local storage	No
Database storage	
IndexedDB	No
Blob object store	No
ArrayBuffer object store	No
Web SQL database	Yes

Files and file systems	Available
Reading files	
Basic support for reading files	Yes
Creating a blob from a file	Yes
Creating a data URL from a blob	Yes
Creating an ArrayBuffer from a blob	Yes
Creating a blob URL from a blob	Yes
Accessing a file system	

Files and file systems	Available
API file system	No
File API: Folders and system	No

Additional functions	Available
Styles	
Style items	No
Scripts	
Asynchronous script execution	Yes
Signaling script errors in Runtime	Yes
Events for script execution	No
Base 64 encoding and decoding	Yes
JSON coding and decoding	Yes
URL API	Yes
MutationObserver	"Yes" (pre-selected)
Promises	No
Page visibility	"Yes" (pre-selected)
Text selection	Yes
Scrolling (Scroll into view)	Yes

13.1.15 Entering barcodes via handheld readers (RT Uni)

Introduction

Optical handheld readers enable you to optically identify components, machines and other objects and to transfer the read-out data on your HMI device directly to certain operating objects.

Optical handheld readers capture codes such as two-dimensional data matrix codes, one-dimensional barcodes and postal barcodes.

Supported optical handheld readers can be found at the following entry on the Internet:

FAQ 19188460 (https://support.industry.siemens.com/cs/ww/en/view/19188460)

You can find templates for the settings and instructions on configuration in the manual for your optical handheld reader.

Note

The optical handheld reader is connected to the USB interface of the HMI device. Only one device at a time can use the USB port. This is why it is not possible to use a USB keyboard and an optical handheld reader or two optical readers at the same time.

Procedure

You use the connected optical reader to read a code into the object that has the focus.

After the read-in, confirm the value with the Enter key or with the "Suffix - Enter" that you have previously configured in the settings of your optical reader.

Objects for input with optical handheld reader

The following objects support input via an optical handheld reader:

Object	Preconditions for input
I/O field Clock	The corresponding data type is selected. The object and the tag length are configured accordingly. The operator element has the cursor focus.
Parameter set view	The parameter set has the cursor focus.
Browser	The operator element has the cursor focus.
Runtime dialogs which support keyboard entry	The dialog is open and the corresponding input field has the cursor focus.
File browser	The field "File path" has the cursor focus.

Result

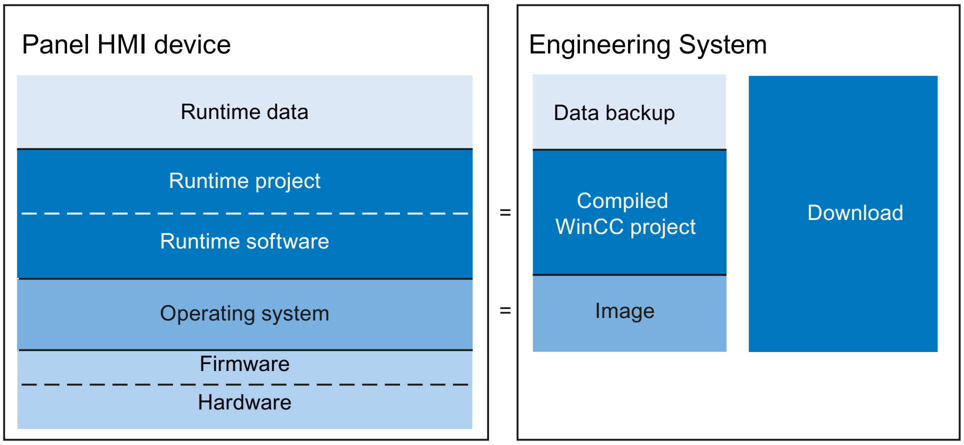
The code is read and entered into the corresponding input field.

13.1.16 Servicing the HMI device (RT Uni)

13.1.16.1 Overview of the service for Unified Comfort Panels (RT Uni)

Structure

The following figure shows the software components of an HMI device and their relation to the engineering system.



Runtime data

The runtime data is generated during operation of the plant and stored on the HMI device. This data includes, for example, parameter sets and data for the user administration. This data is overwritten during loading. If required, save this data before loading a Runtime project.

Runtime project and Runtime software

The Runtime project contains the compiled configuration data for an HMI device. Download the Runtime project along with the Runtime software from WinCC to the HMI device.

Operating system

The operating system of the HMI device is provided as HMI device image via WinCC. Suitable HMI images are supplied with each WinCC version. Depending on the configuration, download the appropriate image along with the Runtime project to the HMI device as required.

Firmware and hardware

The HMI device is delivered with preconfigured firmware and hardware.

13.1.16.2 ProSave (RT Uni)

Introduction

The "ProSave" service tool is included in the WinCC installation. The ProSave functions are accessed in WinCC with the menu "Online > HMI Device maintenance".

Functional scope

ProSave offers numerous functions for data transfer between the configuration PC and HMI device:

- Backing up and restoring the HMI device data
- Updating the operating system of the HMI device
- License Keys transferred
- Installing or uninstalling drivers on an HMI device and providing information about installed and installable options.
- Communication settings (transferred from WinCC)

See also

Overview (Page 844)

13.1.16.3 Backup of HMI data (RT Uni)

Introduction

Data backup is used to create a backup of the data on the HMI device, e.g. before the update of the operating system. You can restore the backed-up data at a later time.

If an HMI device is connected to the configuration PC, you can back up and restore HMI device data from the configuration PC using WinCC.

Alternatively, you can back up the data to an external storage medium supported by the HMI device. If the HMI device is networked, you can also backup the data to a server.

Scope of data backup

The following options are available for data backup:

- Complete backup
  Saves runtime, firmware, operating system, configuration, parameter sets, user administration, options and License Keys
- Firmware/configuration
- Parameter sets only

- Only parameter sets in CSV format
- User administration only

A backup file with the extension *.psb is generated when you backup the data of an HMI device.

Note

Scope of data backup

The selected content of the flash memory is saved during data backup. Alarm logs and process value logs are generally saved on the external storage medium. Alarm logs and process value logs are therefore not backed up. If necessary, back up the contents of the memory card separately.

Note the following for a complete backup and restore of the dataset:

- A full backup includes all options installed. As a rule, the backup includes all options data that is still available after "POWER OFF".
- All data on the device, including License Keys and the operating system, are permanently deleted when you perform complete data restoration.
- If the data restoration was interrupted, execute the command "Reset to factory settings". Restart data restoration.

Note

Use an interface with high bandwidth, such as Ethernet, to back up and restore data via WinCC.

See also

Backing up and restoring data of the HMI device (Page 894)

13.1.16.4 Backing up and restoring data of the HMI device (RT Uni)

Note

Use the restore function for project data only on operating devices which were configured using the same configuration software.

Requirement

- The HMI device is connected to the configuration PC
- The HMI device is selected in the project tree.
- If a server is used for data backup: The configuration PC has access to the server

Backup of the data of the HMI device

Proceed as follows to backup the data of the HMI device:

1. Select the "Backup" command from the "Online > HMI Device maintenance" menu.
   The "Create backup" dialog box opens.
2. Select the type of the PG/PC interface and the target device, and click "Create".
   The "SIMATIC ProSave" dialog box opens.
3. Select the data to backup for the HMI device under "Data type".
4. Enter the name of the backup file under "Save as".
5. Click "Start Backup".

This starts the data backup. The backup operation takes some time, depending on the connection selected.

Restoring the data of the HMI device

Proceed as follows to restore the data of the HMI device:

1. Select the "Restore" command from the "Online > HMI Device maintenance" menu.
   The "Restore backup" dialog box opens.
2. Select the type of the PG/PC interface and the target device, and click "Load".
   The "SIMATIC ProSave" dialog box opens.
3. Enter the name of the backup file under "Save as".
   Information about the selected backup file is displayed under "File information".
4. Click "Start Restore".

This starts the restoration. This operation takes some time, depending on the connection selected.

Backup/Restore from the "Backup/Restore" dialog in the Start Center of the HMI device

The "Backup / Restore" function is enabled for SD memory cards and USB memory media.

For more information, refer to the operating instructions of the HMI device.

See also

Backup of HMI data (Page 893)

13.1.16.5 Updating the operating system (RT Uni)

Introduction

Update the HMI device image if the version is incompatible with the configuration. The version of the image matches the device version.

Update the operating system and Runtime software of the HMI device with the help of the device version. While loading the project, you may be prompted to run an automatic update of the device version, depending on the protocol used.

Loading will then continue. Loading of the project is otherwise aborted. In this case, perform the update of the device version manually.

Updating the device version

Connect the HMI device to the configuration PC to update the device version. If possible, use the interface providing the highest bandwidth for this connection, e.g. Ethernet.

"Reset to factory settings"

If the operating system on the HMI device is no longer operational, update the operating system and reset the HMI device to the factory settings.

See also

Updating the operating system on the HMI device (Page 896)

13.1.16.6 Updating the operating system on the HMI device (RT Uni)

If possible, use the interface providing the highest bandwidth for this connection, e.g. Ethernet. When you update the operating system, the Runtime software on the HMI device is also updated and the device version is changed.

NOTICE
Updating the operating system deletes all data on the HMI device
When you update the operating system you delete data on the target system. For this reason, you should back up the following data beforehand: • User administration • Parameter sets
Resetting to factory settings also deletes the License Keys. Back up the License Keys before you reset the system to factory settings.

Requirement

- The HMI device is connected to the configuration PC.
- The HMI device is selected in the project tree.

Updating the operating system

Proceed as follows to update the operating system:

1. Select the "Update operating system" command from the "Online > HMI Device maintenance" menu.
   The "Update operating system" dialog box opens.
2. Select the type of the PG/PC interface and the target device, and click "Update".
   The "SIMATIC ProSave [OS-Update]" dialog opens. The path to the image is preset.
3. If required, you can select a different path for the image that you want to transfer to the HMI device.
4. Click "Update OS".

This starts the update. The update operation can take time, depending on the connection selected.

Resetting the HMI device to factory settings

To reset the HMI device to factory settings, proceed as follows:

1. Switch off power to the HMI device.
2. Connect the HMI device to the Engineering Station.
3. Select the "Update operating system" command from the menu under "Online > HMI Device maintenance" on the configuration PC in WinCC.
   The "Update operating system" dialog box opens.
4. Select the type of the PG/PC interface and the target device, and click "Update".
   The "SIMATIC ProSave [OS-Update]" dialog opens. The path to the image is preset.
5. If required, you can select a different path for the image that you want to transfer to the HMI device.
6. Activate "Reset to factory settings".
7. Click "Update OS".
8. To reset to factory settings, switch on the power to the HMI device again.
   This operation can take time.

Result

The operating system of the HMI device is operational and up-to-date.

See also

Updating the operating system (Page 895)

Transferring license keys (Page 899)

Managing licenses (Page 899)

Backing up and restoring data of the HMI device (Page 894)

Overview for loading of projects (Page 851)

13.1.16.7 Updating the operating system of the HMI device from a data carrier

Introduction

You can update the operating system using a data storage medium. You can find the HMI image files, for example, in the installation directory of WinCC under: "\Siemens\Automation\Portal V1x\Data\Hmi\Transfer\<HMI device image version>\Images".

NOTICE
Data loss
All data on the HMI device, including the project and HMI device password, is deleted during a restore operation. License keys are only deleted after a security prompt. Back up your data before the restore operation, if necessary.

Requirement

- The HMI device image file is located in the "SIMATIC.HMI\Firmware\" directory on your data carrier, e.g. a SIMATIC HMI Memory card or an industry-grade USB stick.
- The data carrier with the relevant HMI device image file including operating system is inserted in the HMI device.

Procedure

1. Open the "Start Center" on your HMI device.
2. Select "Service & Commissioning > Update OS".
3. Select a storage medium under "Select storage media for OS update".

   Note

   If there is no storage medium or a defective storage medium in the HMI device, the "0 devices found" message is displayed. Insert the storage medium or replace the storage medium.
4. Select the required HMI device image file under "Firmware files on external storage".
5. Press "Update OS".
   The "Update OS Image" dialog opens.
6. To start restoring the operating system, press "Yes".
   The "Transfer" dialog is displayed. A progress bar shows the course of the restore. The HMI device then restarts.

Note

After the restore, it may be necessary to recalibrate the touch screen.

See also

Updating the operating system on the HMI device (Page 896)

13.1.16.8 Transferring license keys (RT Uni)

Introduction

You need a license for certain WinCC Runtime options you may want to install on an HMI device. Usually, the necessary licenses supplied as "License Keys" on a data medium, e.g. USB stick. The "License Keys" can also be made available on a license server.

Use "Automation License Manager" to transfer the "License Keys" to or from an HMI device. The "Automation License Manager" is included automatically when you install WinCC.

> **NOTICE**
>
> **Backing up License Keys**
>
> In the following cases you have to backup the "License Keys" in order to prevent deletion of the "License Keys":
>
> - Before updating the operating system
> - Prior to restoring the data from a full backup
>   "License Keys" on an HMI device are backed up depending on the HMI device configuration. For more information on this topic, refer to the operating instructions of the HMI device.

See also

Managing licenses (Page 899)

13.1.16.9 Managing licenses (RT Uni)

Requirement

- The HMI device is connected to the configuration PC or the PC running the "Automation License Manager".
- If you are using the configuration PC: The HMI device is selected in the project tree.

Procedure

To transfer license keys, follow these steps:

1. Open the "Automation License Manager". Go to the Windows Start menu and start "Automation License Manager" on a PC on which WinCC is not installed.
   The "Automation License Manager" starts.
2. Select the "Connect HMI device" command from the "Edit > Connect target system" menu.
   The "Connect target system" dialog opens.
3. Select the HMI device type in the "Device type" area.
4. Select the "Connection".
5. Configure the "connection parameters" associated with the selected connection.
6. Click "OK".
   The connection to the HMI device is now set up. The connected HMI device is displayed in the left pane of "Automation License Manager".
7. Transfer the "License Keys" to the HMI device:
   - In the left pane, select the drive on which the "License Keys" are located.
     The "License Keys" are displayed on the right pane.
   - Select the "License Keys"
   - Drag-and-drop the "License Keys" to the HMI device.

You can also remove License Keys from the HMI device using drag-and-drop.

Alternative procedure

You can also start the "Automation License Manager" from WinCC on a PC with a WinCC installation: Select the "Authorize/License" command in the "Online > HMI Device maintenance" menu.

Result

The "License Keys" are transferred to the HMI device.

To backup the "License Keys" from the HMI device, drag-and-drop the "License Keys" from the HMI device to an available drive.

See also

Transferring license keys (Page 899)

13.1.16.10 Installing and uninstalling an option (RT Uni)

Introduction

You can install the following options on an HMI device:

- Additional options supplied with WinCC
- Options purchased in addition to WinCC

Which options you can install depends on the HMI device type.

You can find an overview of the installable options in the "Getting started with WinCC".

Requirement

- The HMI device is connected to the configuration PC.
- The PG/PC interface is set.
- The HMI device is selected in the project tree.
- The HMI device is switched on.

Procedure

To install an option on the HMI device, proceed as follows:

1. Select the "Options" command from the "Online > HMI Device maintenance" menu.
   The "Load options" dialog opens.
2. Select the type of the PG/PC interface and the target device, and click "Load".
   The "SIMATIC ProSave" dialog box opens.
   All available options and the previously installed options are displayed.
3. To display the installed options on the HMI device, click "Device status".
4. To install an option on the HMI device, select the option under "Available options" and click ">>".
5. To uninstall an option from the HMI device, select the option under "Available options" and click "<<".

Result

The selected options are installed on or uninstalled from the HMI device.

13.2 Unified PC (RT Uni)

13.2.1 Runtime settings (RT Uni)

13.2.1.1 Settings in the runtime software (RT Uni)

Introduction

To edit the runtime settings for an HMI device, select "Runtime settings" under the HMI device in the project tree.

Overview

You can view or edit the following settings:

Setting		Description
General	Identification	Indicates the project identification.
	Encrypted transfer	Make settings for encrypted transfer. You can find additional information under "Encrypted transfer".
	Screen	Specifies the start screen.
Alarms	Controller alarms	Displays the alarms of the controller.
	State texts	Specifies the texts of different states.
Services	Reading/writing tags	Specifies that the HMI device works as an OPC server.
Language & font	Runtime language and font selection	• Specifies the available runtime languages and the sequence of language selection. • Specifies the default font. • Specifies the languages for logging in runtime.
Collaboration	Identification	Specifies the system identification of the collaboration, collaboration name and the IP address.

Setting		Description
Storage system	Database type	Specifies the database type for logs: • SQLite • Microsoft SQL If "Microfoft SQL" is selected, a separate software installation is required.
	Database storage location for tag persistence	Specifies the storage medium for the tag persistence: • Off • Local • Project folder
	Main database location for logging	Specifies the storage medium for logs: • Off • Local • Project folder

See also

Encrypted transfer (Page 904)

Start screen (Page 903)

Configuring an HMI device as an OPC UA server (Page 1060)

Languages in runtime (Page 772)

13.2.1.2 Start screen (RT Uni)

Start screen settings

You specify the start screen in the runtime settings of the HMI device under "General > Screen".

The start screen is the initial screen that is displayed after runtime starts. The screen resolution is automatically adjusted to suit the HMI device when the screen is selected.

You specify the start screen in the runtime settings of the HMI device under "General > Screen".

Note

Displaying a start screen changed by reloading

You have defined a start screen in the project and started runtime. If you then define another start screen in your project and load the project in the device again, the last active screen is displayed in runtime once connected again.

After reloading the screen, refresh it in runtime in the browser with the <F5> key or the "Update" button in the browser.

See also

Settings in the runtime software (Page 902)

13.2.1.3 Encrypted transfer (RT Uni)

Introduction

To enable secure loading of the runtime project, assign a password for encrypted transfer. You enter the password both in the engineering system and in runtime.

Settings for encrypted transfer

In the runtime settings under "General > Encrypted transfer", make the following settings:

- Activate encrypted transfer
- Entering a password in the engineering system
- Allow transfer of initial password via unencrypted loading

You define the password for encrypted transfer in Runtime in the application "WinCC Unified - Configuration" in the "Secure download" area. Alternatively, to transfer the password unencrypted, activate the option "Allow transfer of the initial password via unencrypted loading".

If the password assigned in the engineering system does not match the password in Runtime, select one of the following options:

- Use the "WinCC Unified - Configuration" application to change the Runtime password.
- In the Runtime settings, allow the transfer of the initial password via unencrypted loading and load the project completely.
  Requirement: The old password is known.

13.2.1.4 Printing in runtime (RT Uni)

Print functions

The following print functions are available in runtime:

- Hardcopy
- Printing alarms
  Every alarm that has occurred and its state changes are logged on a printer.
- Printing reports
  Reports are output in graphic mode. The use of a serial printer is not recommended because of the accumulated data volume.
  For the report to be printed correctly, the printer must support the paper format and page layout of the report.

Note

The value of a tag in the report is read and output at the moment of printing. A substantial period of time may elapse between printing out the first and the last page of a report consisting of several pages. This may lead to the same tag being output with a different value on the last page than on the first page.

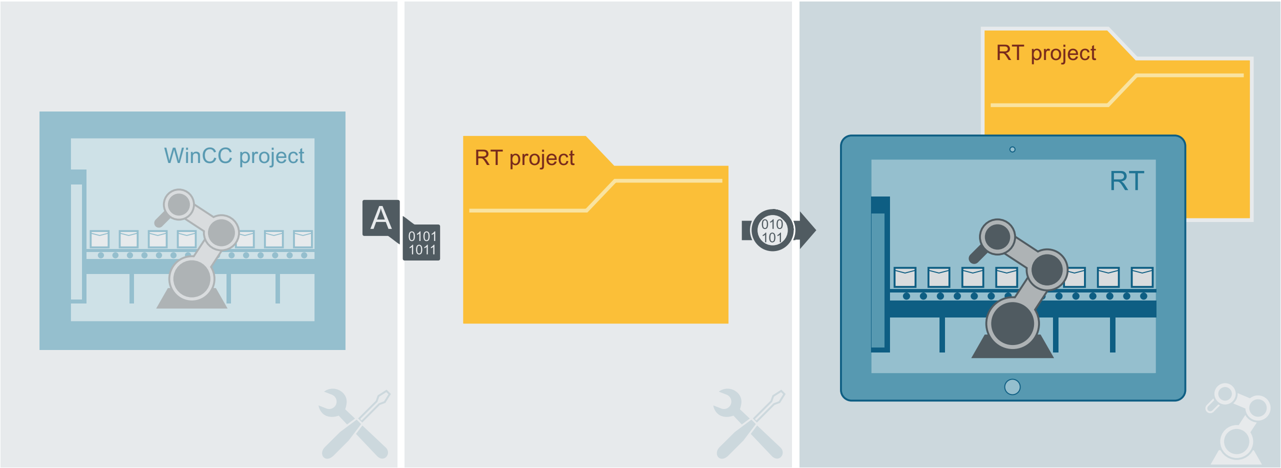
13.2.2 Overview (RT Uni)

The term "project"

The term "project" has two different meanings in the context of "compiling and downloading".

- WinCC project: Contains the configuration data of a HMI device in WinCC
- Runtime project: Contains the compiled configuration data of an HMI device.
  If Runtime has been installed on the configuration PC, you can execute the runtime project directly on the configuration PC.
  If you want to execute the runtime project on a different PC, you have to transfer the runtime project to the PC before startup.

The figure below illustrates the link between WinCC projects and runtime projects using the example of the "Compile and Download" process:



A 0101 1011	The configuration data are compiled.
010 101	The runtime project is loaded.

Definitions

To compile a project means generating a runtime project from the WinCC project.

Downloading a project means transferring the runtime project to an HMI device.

WinCC Unified SCADA RT is the runtime software for process visualization. In runtime, you execute the project in process mode.

Simulation

You test your configuration with a simulation. You start simulation without a connection to the running process.

In a simulation, you test configured internal tags or a screen change, for example. You simulate the project on the configuration computer.

See also

Compiling a project (Page 907)

Sequence of the download process (Page 912)

Loading a project (Page 913)

Simulating projects (Page 907)

13.2.3 Compiling a project (RT Uni)

Scope of the compilation

In the background, the configuration data continuously checked for consistency and compiled.

If you compile a project manually, only the changes in the configuration made since the last compilation process are compiled in the background.

Requirement

- A project is open.

Procedure

Proceed as follows to compile a project:

1. To compile the configuration data of multiple HMI devices at the same time, select all HMI devices using multiple selection in the project tree.
2. To compile the project, click "Compile" in the toolbar.

Result

The configuration data of all selected HMI devices is compiled. If errors occur during compilation, the errors are shown in the Inspector window.

See also

Overview (Page 905)

Sequence of the download process (Page 912)

Loading a project (Page 913)

13.2.4 Simulating projects (RT Uni)

13.2.4.1 Basics of simulation (RT Uni)

Introduction

You can use the simulator to test the performance of your configuration on the configuration PC. This allows you to quickly locate any logical configuration errors before productive operation.

You can start the simulator as follows:

- In the shortcut menu of the HMI device or in a screen: "Start simulation"
- Click "Start simulation" in the toolbar.

- Menu command Online > Simulation > Start
- Under "Visualization > Simulate device" in the portal view.

Field of application

You can use the simulator to test the following functions of the HMI system, for example:

- Screen change and screen navigation
- Internal tags
- Layout
- Configured alarms

See also

Simulating a project (Page 909)

Simulating a screen (Page 911)

13.2.4.2 Skip "Load preview" dialog (RT Uni)

Skip "Load preview" dialog

To permanently skip the "Load preview" dialog when simulating projects and screens, proceed as follows:

1. Open the settings under "Options > Settings".
2. Select "Simulation".
3. In the "HMI Simulation" area, clear the check box "Show 'Load preview' dialog during download to simulation".

Result

- The "Load Preview" dialog is no longer displayed.
- The simulation is opened automatically in the standard browser.

Note

Errors and warnings that occur are displayed in the Inspector window in the "Info" tab.

Note

Settings of the "Load preview" dialog

The following settings are applied from the previous loading process with displayed "Preview Load" dialog:

- Settings for keeping tag values, active alarms and user data (default value: enabled).
- Settings for resetting logs (default value: "No reset")

If the "Load preview" dialog was hidden before the first loading of the project, the default values are used.

See also

Simulating a screen (Page 911)

Simulating a project (Page 909)

13.2.4.3 Simulating a project (RT Uni)

Introduction

You simulate a project on the configuration computer.

Note

Simulating a project on a configuration PC while Runtime is running

Runtime is terminated when a project on the HMI device is running in runtime and you use the option "Full download".

Runtime is not terminated when a project on the HMI device is running in Runtime and you use the option "Delta download". For example, tags keep their value and are not set to the start value.

Ethernet connection

You download your runtime project simulation to the HMI device via an Ethernet connection. The connection uses Ethernet port 20008.

Note

Ethernet port 20008

If an application is using Ethernet port 20008, download is not possible.

If no connection to the target can be established, check the port assignments. If another application is using Ethernet port 20008, close this application.

Requirement

- The "Simulation (SIMATIC WinCC Unified Scada)" component is installed on the configuration PC.
- The project is open in the configuration PC.
- The HMI device and the HMI device have been successfully compiled.

Procedure

Proceed as follows to simulate a project:

1. Click "Start simulation" in the toolbar.
   The "Load Preview" dialog is displayed and the compilation result is displayed.
2. Check the displayed default settings and change the settings as necessary:
   – Specify whether to use the "Full download" or "Delta download" option.
   – Specify whether runtime should start after the download.
   – When you use the "Full download" option again, you specify whether tag values, active alarms, and user data are retained. Only available if you have selected "Start runtime".
   – When you use the "Full download" option again, you specify whether all logs are reset in runtime. Only available if you have selected "Start runtime".
3. Click "Download".
4. Open the browser.
5. Call the URL *"https://localhost"* in the browser.
   Instead of the name "localhost", you can use the computer name.
6. Select "WinCC Unified RT".
7. Enter the user name and password.
   The configured screen is displayed as start screen in the browser.
8. Test, for example:
   – Screen change and screen navigation.
   – Layout
   – Internal tags.
9. To stop the simulation, select "Online > Stop runtime/simulation".

See also

Basics of simulation (Page 907)

Simulating a screen (Page 911)

Skip "Load preview" dialog (Page 908)

13.2.4.4 Simulating a screen (RT Uni)

Introduction

If you have only made changes to one screen, you can temporarily specify this screen as the start screen for simulation. In this way, you can debug changes without having to modify the start screen, or opening the screen on the HMI device.

Requirement

You created a project that contains at least one screen.

Procedure

To define a screen as temporary start screen for simulation, follow these steps:

1. In the project tree, select the screen that is to become the temporary start screen in the simulation.
2. Select the "Start simulation" command from the shortcut menu of the screen.
   The "Load Preview" dialog is displayed and the result of the compiling is displayed.
3. Check the displayed default settings and change the settings as necessary:
   - Specify whether to use the "Full download" or "Delta download" option.
   - Specify whether runtime should start after the download.
   - When you use the "Full download" option again, you specify whether tag values, active alarms, and user data are retained. Only available if you have selected "Start runtime".
   - When you use the "Full download" option again, you specify whether all logs are reset in runtime. Only available if you have selected "Start runtime".
4. Click "Download".
5. Open the browser.
6. Call the URL *"https://localhost"* in the browser.
   Instead of the name "localhost", you can use the computer name.
7. Select "WinCC Unified RT".
8. Enter the user name and password.
   The simulated screen is displayed.

Result

If "Start runtime" is selected in the settings for loading, the screen selected in the project tree is displayed in the simulation window instead of the configured start screen.

See also

Basics of simulation (Page 907)

Simulating a project (Page 909)

Skip "Load preview" dialog (Page 908)

13.2.5 Downloading projects (RT Uni)

13.2.5.1 Sequence of the download process (RT Uni)

Introduction

The project is automatically compiled before you download it to an HMI device. This always ensures that the latest version of the project is transferred.

If you are using external HMI tags in your project that are connected to PLC tags, you should also compile the user program before you compile the HMI device.

Process sequence

If you are downloading a runtime project, follow the procedure below:

1. Select the WinCC project.
2. Set the connection first.
   – Enter the connection data of the target device.
   – The connection is checked and established.
     The connection is thus defined for all subsequent download processes for this WinCC project. To change the connection, open the "Online > Extended upload to project" dialog again.
3. WinCC project is compiled.
   – If the compilation completes with errors, the download is aborted.
   – Runtime project is created.
4. Settings for downloading
   – If the runtime project is already running or is stored on the target device, confirm the closing of Runtime and overwriting the existing data.
   – If another runtime project is running on the target device, confirm the closing of Runtime.
   – You specify whether runtime is started on the HMI device after the download.
5. Triggering the download of the runtime project.

Downloading a runtime project several times

You can download a runtime project one after the other to several connected HMI devices and your configuration PC and start runtime at the same time.

WinCC supports different Runtime versions and configurations.

Downloading different runtime projects

Close the project open on the HMI device to download an additional project to the HMI device.

Example: The project "Mixing" is open on the HMI device but not in runtime. If you change the project, download the changes to the HMI device using "Download to device". To download the "Bottling" project to the HMI device, for example, close the project "Mixing" on the HMI device.

See also

Loading a project (Page 913)

Overview (Page 905)

Compiling a project (Page 907)

13.2.5.2 Loading a project (RT Uni)

Introduction

You download a runtime project from your WinCC project. You always download only one runtime project to a connected HMI device or to your configuration PC.

You load either the complete runtime project or only changes of a runtime project. To load changes, use the "Delta download" option.

How to handle existing runtime projects

If you have already downloaded a project, the download process recognizes the project using the project identification.

The project identification can be found in the runtime settings of the operating device under "General".

Note

Existing runtime projects on the target device

If a runtime project with the same project identification is already available on the HMI device and you select the command "Download to device > Software (all)", the entire project is downloaded again. Existing runtime data from the same project is overwritten.

Save relevant data before the download.

Note

Downloading the project to an HMI device while Runtime is being executed.

Runtime is closed when a project is running in runtime on the HMI device and you load a project to this HMI device with the command "Download to device > Software (all)".

Runtime is not terminated when a project is running in Runtime on the HMI device and you load a project to this HMI device with the command "Download to device > Software (only changes)". For example, tags keep their value and are not set to the start value.

Note

If you have changed the name or data type of a tag, you must load the runtime project completely.

Ethernet connection

You download your runtime project to the HMI device via an Ethernet connection. The connection uses Ethernet port 20008.

Note

Ethernet port 20008

If an application is using Ethernet port 20008, download is not possible.

If no connection to the target can be established, check the port assignments. If another application is using Ethernet port 20008, close this application.

Device version

If the device version of the target HMI device does not correspond to that of the configured device version, runtime cannot be started after the download:

- Verify that the device version of the target HMI device conforms to your configuration before you compile and download your project.
- If necessary, change the device version manually via the properties of the HMI device.

Requirement

- The controller data have been compiled without errors.
- The HMI device has been compiled without errors.
- The device versions of the target HMI device correspond to the configured device version.
- The HMI device is connected to the configuration PC or the configuration PC serves as HMI device.
- Ethernet port 20008 in your network configuration is not allocated.

To download a project using the command "Download to device > Software (download changes)", the following additional requirements must be met:

- The runtime project that is to receive the changes is executed.
- The project versions of the executed runtime project and the runtime project loaded onto the external storage medium using the "Delta download" option are identical.

Loading a project

1. Select the HMI device in the project tree.
2. Select "Download to device > Software (only changes)" or "Download to device > Software (all)".
   When you download the WinCC project for the first time, the "Extended download" dialog opens.
3. Enter the IP address or device name of the target HMI device.
   If you use your configuration PC as a HMI device, enter the IP address 127.0.0.1 or the device name "localhost".
4. Click "Connect".
   The connection is established and a dialog is displayed.
5. When the connection is established, click "Load".
   The WinCC project is compiled again.
   The "Load Preview" dialog is displayed. The compilation result is displayed.
6. Check the displayed default settings and change the settings as necessary:
   - Specify whether to use the "Full download" or "Delta download" option.
   - Specify whether runtime should start on the target system after the download.
   - When you use the "Full download" option again, you specify whether tag values, active alarms, and user data are retained. Only available if you have selected "Start runtime".
   - When you use the "Full download" option again, you specify whether all logs are reset in runtime. Only available if you have selected "Start runtime".
7. Click "Download".
8. Open runtime.
9. If you have used the "Delta download" option, update runtime.

Select another target device

1. Select the HMI device in the project tree.
2. Select "Online > Extended download to device".
   The "Extended download" dialog opens.
3. Enter the IP address or device name of the new target device.
4. Click "Connect" and reload the project.

The runtime project is loaded onto another HMI device.

Result

The project is loaded onto the selected HMI device under the following file path:

C:\ProgramData\SCADAProjects

If errors or warnings occur during the download, corresponding alarms are displayed under "Info > Load" in the Inspector window.

On completion of the successful download of the project, you can execute it on the HMI device. If you have activated the start of runtime on the target system in the "Load Preview" dialog, the project is started in runtime after loading.

See also

Sequence of the download process (Page 912)

Overview (Page 905)

Compiling a project (Page 907)

Settings in the runtime software (Page 902)

13.2.5.3 Using external storage medium (RT Uni)

Loading project to external storage medium (RT Uni)

Introduction

If you cannot establish a direct connection from the configuration PC to the HMI device, load the compiled runtime project onto an external storage medium. For example, use a USB stick or SD card.

You load either the complete runtime project or only changes of a runtime project. To load changes, use the "Delta download" option.

As soon as you have connected the external storage medium to your HMI device, load the project on your HMI device.

Requirement

- An HMI device has been created.

Procedure

To create an external storage medium and load a project onto the storage medium, proceed as follows:

1. Jump to the "Devices" tab in the project tree.
2. Double-click "Add user-defined card reader" in the "Card reader/USB storage" folder.
   A selection dialog opens.
3. Select a target directory to save the project.

4. Drag and drop the folder of the HMI device (e.g. "HMI_1 [<Device type>]") to the added folder. Alternatively, use copy and paste.
   The project is checked. If the project has contents that have not yet been compiled, a compile is performed.
   The "Load Preview" dialog opens.

   Note

   If a runtime project with the same project identification already exists in the target directory, only the options "Full download" and "Delta download" are available for download.

5. In the selection menu, specify how your project is to be loaded:
   - "Full download"
   - "Delta download"

   Note

   Only one runtime project at a time with the same project identification can be saved in the target directory via the option "Full download" and "Delta download".

   If a runtime project already exists, it is overwritten during loading.

   Save relevant data before the download.

6. Click "Load" to confirm.

Result

Your project is stored as a compressed ZIP folder in the directory "[<Target directory>]\Simatic.HMI\RT_Projects" :

- Projects that were created with the option "Full download" receive as file name e.g. "[<Project_name>].PC-System_1[SIMATIC PC station - WinCC Unified Scada RT]_full.zip".
- Projects that were created with the option "Delta download" receive as file name e.g. "[<Project_name>].PC-System_1[SIMATIC PC station - WinCC Unified Scada RT]_delta.zip".

Load project from external storage medium (RT Uni)

Introduction

When you load a project from an external storage medium, the SIMATIC Runtime Manager extracts the repository to a temporary folder on the target system. The transfer to runtime takes place from this folder which is then deleted again.

Requirements

- SIMATIC Runtime Manager is installed on the HMI device.
- The storage medium with the backed-up project is connected to the HMI device.

To download a project that has been downloaded to the external storage medium using the "Delta download" option, the following additional requirements must be met:

- The runtime project that is to receive the changes is executed.
- The project versions of the executed runtime project and the runtime project loaded onto the external storage medium using the "Delta download" option are identical.

Procedure

1. Start the "SIMATIC Runtime Manager"." tool.
2. Click ⊕ "Add project from offline transfer".
   The "Add project" dialog opens.
3. Click "..." under "Select project archive".
   A selection dialog opens.
4. Select the compressed ZIP folder of the runtime project on the storage medium.
5. Click "Open".
   Under "Project information" you can see details of the selected project.
6. To start the project directly, select the option "Start runtime with project" under "Options".
7. Confirm with "Add project".

The project is included in the "Project" list and started directly in the runtime.

If you have not selected the "Start runtime with project" option, you can start the project from the project list of the SIMATIC Runtime Managers:

1. Select the check box in the "Project" column.
2. Select the button ▶ "Start runtime with selected project".

Note

You can execute only one project in runtime at the same time.

13.2.6 Compiling and loading with multiuser engineering (RT Uni)

13.2.6.1 Compiling and loading with multiuser engineering (overview) (RT Uni)

Introduction

When using multiuser engineering for your projects, you should take into account the response when compiling the runtime projects and the response when downloading them to HMI devices.

You can compile and download to an HMI device in both the server project view and in the local session.

You can find more information on Multiuser Engineering in "Using Multiuser Engineering".

Basics

The following scenarios are possible for Unified PC in multiuser engineering:

- Compiling in the server project view
- Compiling in the local session
- Loading from the server project view
- Loading from the local session

Note

The option "Full download" from the server project view or from the local session does not differ from the option "Full download" in a single-user project. With the option "Full download", the current runtime project is loaded from the currently active view to an HMI device.

Note

Compiling and downloading in a local session is no different from compiling and downloading in a single-user project.

In principle, you can execute all commands for compiling and loading in multiuser engineering projects:

- "Software (compile all)"
- "Compile > Software (only changes)"
- "Software (all)"

The term "project"

The term "project" has two different meanings in the context of "compiling and downloading". "Project" is the WinCC project on the configuration PC. "Project" is also the runtime project that you create by compiling the configuration data of an HMI device and downloading it to the HMI device.

- WinCC project: Contains the configuration data of one or more HMI devices
- Runtime project: Contains the compiled configuration data of an HMI device

Rules

The following basic rules apply to compiling and downloading in multiuser engineering:

- The runtime project which was compiled in a local session always remains local and is not uploaded to the multiuser server. It cannot be saved in the multiuser server project.
- Only runtime projects compiled in the server project view can be saved in the multiuser server project.

You can find additional information on Multiuser Engineering on the Siemens YouTube channel: "Multiuser Engineering - one team working simultaneously on a project (https://www.youtube.com/watch?v=n4oTZ2Gzg6U)".

13.2.6.2 Compiling in the server project view (RT Uni)

Basics

Compiling and downloading in the server project view is no different from compiling and downloading in a single-user project.

During the compiling of a project in the server project view, the multiuser server project is blocked. Other users cannot make changes to this server project during this time. The runtime project compiled in the server project view is stored along with the engineering project in the central multiuser server. Blocking the multiuser server project ensures that the configuration data and the runtime project remain in sync.

Note

When you compile and save in the server project view, other users obtain the runtime project you have updated along with the engineering project when they "refresh" their local session. Other users do not have to recompile the changes you have made after an update.

Example: Compiling during check-in

You make changes to a tag in a local session. All prior changes have been compiled in the associated server project.

If there are no compilation errors, both projects - the modified engineering project (with the modified tags) and the compiled runtime project - are saved in the central multiuser server project with the "Save changes" command.

If you skip compiling during the check-in, the project contains the changes that have been saved on the server.

The next user who creates a local session from the server project or updates an existing local session must compile your two changes in addition to his or her own changes.

Note

Working on a shared project through multiple local sessions increases the probability of errors. It is therefore recommended to compile the project at check-in and eliminate any errors that are reported during compiling. In this way, you provide the next user with a project free of errors.

13.2.6.3 Compiling in the local session (RT Uni)

Basics

Compiling and downloading projects in the local session is no different from compiling and downloading in a single-user project.

Since the local session is a copy of the server project,the first compilation status of the local session is identical to that of the server project. If the server project contains contents that are not compiled or error messages occurred during compiling, they are transferred to the local session.

Note

It is recommended to compile the project at check-in and eliminate any errors that are reported during compiling. In this way, you provide the next user with a project free of errors and avoid spreading errors.

Updating in the local session

If you update a project in the local session, the local session - including the compilation status - is completely replaced by the content of the server project. Only the changes marked for check-in are retained in the updated local session and generate additional compiling steps in the local session.

Example: Updating the local session

You make changes to a tag in a local session. All prior changes have been compiled in the associated server project.

You update the content of the local session by clicking the "Update" button. After the update, the local session obtains the compilation status of the server project. There are also compiling tasks for the acquisition of the modified tags.

13.2.7 Error messages during loading of projects (RT Uni)

Possible problems during loading

When a project is being downloaded to the HMI device, status messages regarding the download progress are displayed in the output window.

Problems arising during the download of the project to the HMI device are usually caused by one of the following errors:

- Wrong operating system version on the HMI device
- Incorrect settings for loading on the HMI device
- Incorrect HMI device type in the project
- The HMI device is not connected to the configuration PC.

The most common download failures and possible causes and remedies are listed below.

The download is canceled due to a compatibility conflict

Possible cause	Solution
Conflict between versions of the configuration software and the operating system of the HMI device	Synchronize the operating system of the HMI device with the version of the configuration software. For additional information, refer to the operating instructions for the HMI device.
The configuration PC is connected to the wrong device, e.g. a controller.	Check the cabling. Correct the communication parameters.

Project download fails

Possible cause	Solution
Connection to the HMI device cannot be established (alarm in the output window)	Check the physical connection between the configuration PC and the HMI device.

13.2.8 Starting runtime (RT Uni)

Introduction

If WinCC Runtime is also installed on a configuration PC, you start Runtime during loading. To do this, activate the corresponding option in the "Load Preview" dialog.

If only WinCC Runtime is installed on an HMI device, download the project to the HMI device.

To start runtime, use the "SIMATIC Runtime Manager" or enable the subsequent start of runtime during download.

If you have already downloaded several runtime projects, use "SIMATIC Runtime Manager" to select the project to be started.

Note

Downloading the project to an HMI device while runtime is being executed.

The runtime that is running is closed when a project is in runtime on the HMI device and you download a project to this HMI device using the "Full download" option.

Requirements

- "WinCC Unified Runtime" is installed on the device.
- The HMI device is connected to the configuration PC via Ethernet.

Procedure

To start the runtime of a SIMATIC Unified PC, proceed as follows:

1. Select the desired HMI device in the project tree.
2. Click "Download to device" in the toolbar. Alternatively, select "Download to device > Software (all)" in the shortcut menu.
   When you download the WinCC project for the first time, the "Extended download" dialog opens.
3. Enter the IP address of the target device.
4. Click "Connect"
   The "Load Preview" dialog is displayed.
5. Check the displayed default settings and change the settings as necessary:
   - Specify whether to use the "Full download" or "Delta download" option.
   - Specify whether runtime should start on the target system after the download.
   - When you use the "Full download" option again, you specify whether tag values, active alarms, and user data are retained. Only available if you have selected "Start runtime".
   - When you use the "Full download" option again, you specify whether all logs are reset in runtime. Only available if you have selected "Start runtime".
6. Click "Download".

Result

The project is compiled and loaded.

If you have activated the start of runtime on the target system in the "Load Preview" dialog, the project is started in runtime after loading.

To stop runtime, select "Online > Stop runtime/simulation". Alternatively, use the "SIMATIC Runtime Manager" to stop runtime.

See also

Settings in the runtime software (Page 902)

13.2.9 Adapting the project for another HMI device (RT Uni)

Introduction

When you download a WinCC project to an HMI device, WinCC checks whether the HMI device is compatible with the HMI device type used in the project. If the types of HMI device do not match, you will see a message before the download starts.

The download is aborted.

Adapting the project for the HMI device

You need to adapt the project accordingly to be able to download the project to the connected HMI device.

- Add a new HMI device in the project tree. Select the correct type of HMI device from the HMI device selection.
- Copy the configured components from the previous to the new HMI device.
  You copy many components directly in the project tree and the details view.
  For example, copy the "Screens" folder to the screens folder of the new HMI device using the shortcut menu.
- Use the detail view to copy entries in the project tree for which the "Copy" command is not available in the shortcut menu.
- Select the "Recipes" entry in the project tree, for example. The recipes are displayed in the detail view.
- Select the recipes in the detail view and drag them to the "Recipes" entry of the new HMI device. The recipes are copied. You can also select multiple objects in the detail view.
- Configure the components that cannot be copied, e.g. connections, area pointers, and alarms.
- Save the project at various points in time.
- Compile the full project.
- When the compilation is successfully completed, download the project to the HMI device.

Linking references

References to linked objects are included in the copying. The references are once again linked to each other after the linked objects are copied.

Example:

You copy a screen in which objects are linked to tags. The tag names are entered at the individual objects after the screen is added to the new HMI device. The tag names are marked in red because the references are open. When you then copy the tags and insert them into the new HMI device, the open references are closed. The red marking for the tag names disappears.

To complete references to connected objects in the controller, you first need to configure a connection to the controller.

Using the information area

When you compile the project for the new HMI device, errors and warnings are displayed in the "Info" tab of the Inspector window. You can use the shortcut menu command "Go to" to go directly to the location where the error or warning can be corrected.

Work through the list of errors and warnings from top to bottom.

When the compilation is successfully completed, download the project to the HMI device.

13.2.10 Users in runtime (RT Uni)

13.2.10.1 Changing users in runtime (RT Uni)

Introduction

In runtime, different users can log on, provided these users have been created.

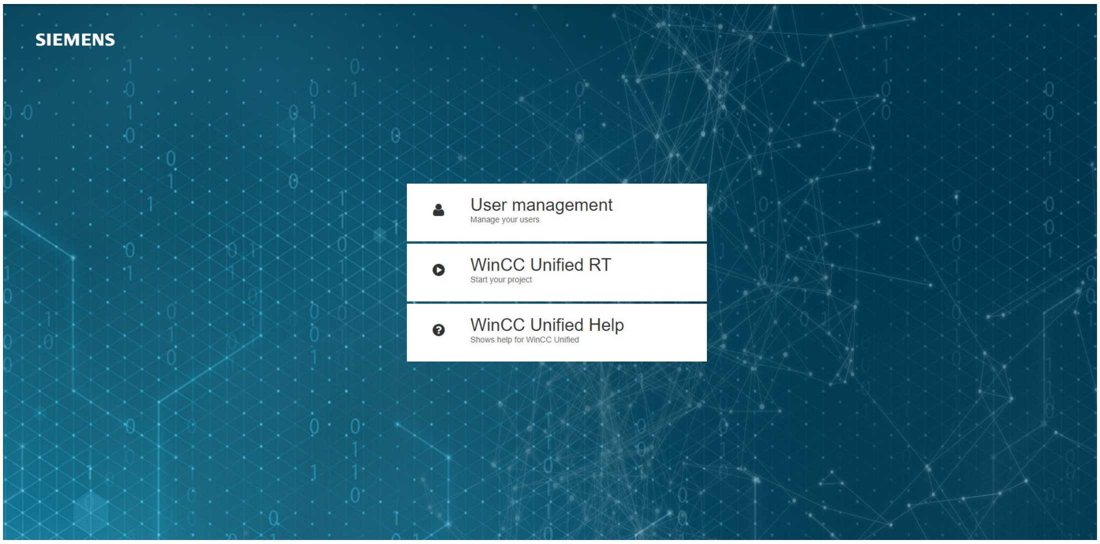
Requirement

- The IP address or the fully qualified name (computer name and domain) of the computer on which Runtime is installed is entered in the browser.
  If Runtime is not installed on the same computer as the browser, the "localhost" designation can also be used.
- A user is logged into runtime.
  You log in by selecting "WinCC Runtime RT" or "User management".

Procedure

To log off a user and then log on a different user, proceed as follows:

1. Select "User management".
2. Expand the menu at the top right.
3. Select "Logout".
4. Log in with a different user.



See also

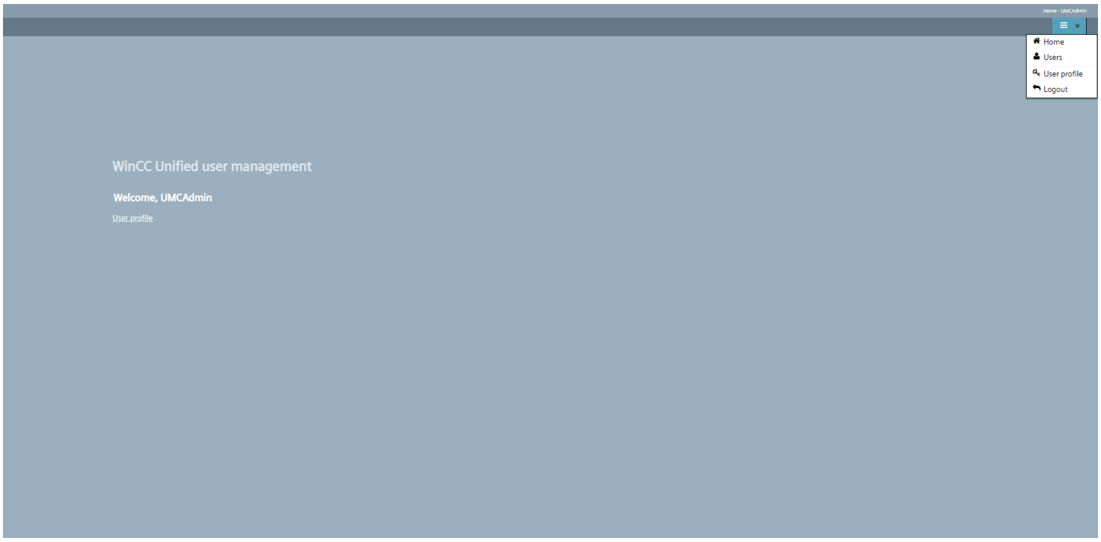
User administration in runtime (Page 926)

13.2.10.2 User administration in runtime (RT Uni)

Overview

Through the "User" selection menu, administrators have the possibility to create new users, edit the properties of the users and assign roles to them.

You can change the password via the "User profile" selection menu.



See also

Changing users in runtime (Page 925)

Configuring cycles (RT Uni) 14

14.1 Basics of cycles (RT Uni)

Introduction

Cycles are used to control actions that regularly occur in runtime. Common applications are the acquisition cycle, the logging cycle and the screen cycle. You can also define your own cycles in addition to those already provided in WinCC.

Principle

In runtime, actions that are performed regularly are controlled by cycles. Typical applications for cycles:

- Acquisition of external tags
  The acquisition cycle determines when the HMI device will read the process value of an external tag from the PLC. Set the acquisition cycle to suit the rate of change of the process values. The temperature of an oven, for example, changes much more slowly than the speed of an electrical drive.
  Do not set the acquisition cycle too low, since this will unnecessarily increase the communication load of the process.
- Triggering scheduled tasks
  In scheduled tasks you have the option to configure a task with a cyclical trigger. Use the cycle time to determine when the scheduled task is executed.
- Logging process values
  The logging cycle determines when a process value is saved in the logging database. The logging cycle is always an integer multiple of the acquisition cycle.

The smallest value for a cycle in Runtime Unified is 100 ms. You can configure all further values with an increment of 50 ms. The smallest default values are 100 ms, 250 ms and 500 ms.

Application example

You can use cycles for the following tasks:

- To record and archive process values.
- To trigger tasks.
- To regularly log a process.
- To draw attention to maintenance intervals.

14.2 Defining cycles (RT Uni)

Introduction

Use cycles to control actions that are run at regular intervals in Runtime. You can also define your own cycles in addition to those already provided in WinCC.

Requirement

The project is open.

Procedure

To define a cycle, follow these steps:

1. Double-click the "Cycles" entry in the project navigation.
   The "Cycles" editor opens.
2. In the "Name" column of the "Cycles" editor, double-click "Add".
   A new cycle time is created.
3. Enter a unique name in the "Name" field.
4. Select the desired cycle unit.
5. Select the desired value for the cycle time.
   The available selection of values varies depending on the cycle unit selected.
6. As an option, you can enter a comment regarding the use of the cycle.
7. Save the project.

Result

The cycle you configured is created and beside the default cycles in WinCC for use during configuration.

Creating production reports (RT Uni) 15

15.1 Basics (RT Uni)

15.1.1 Introduction (RT Uni)

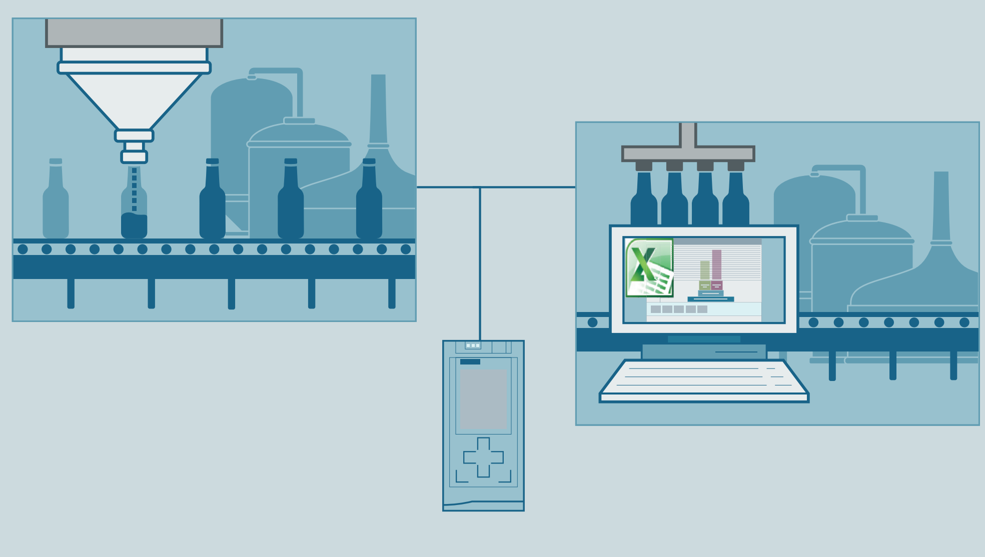
Introduction

With WinCC Unified Reporting, you can generate production reports in the form of Excel reports in Runtime. Reporting covers the following project data:

- Logging tags and online tags
- Log alarms
- With the Performance Insight option package installed: KPIs and operands
  See also Creating production reports for PI options.

You can then continue to edit the data in Excel or save the report as PDF and distribute or archive it.

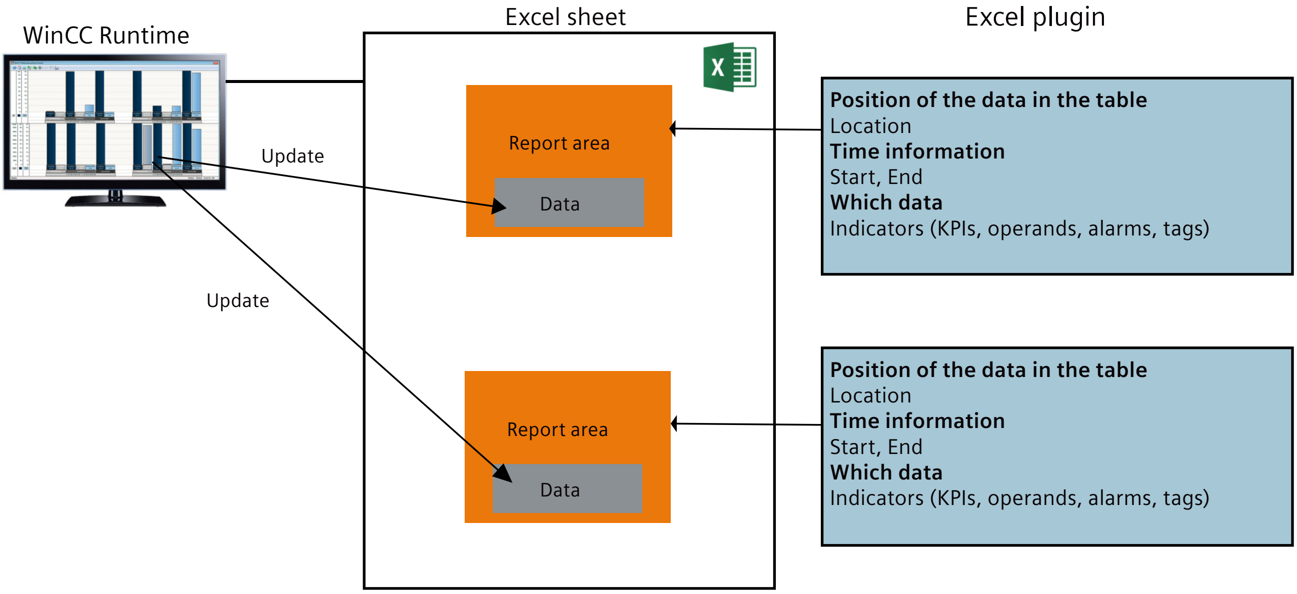
For example, you can generate a report that outputs all alarms occurring in a production line. Prepare the values in the report graphically and then distribute the report for analysis.



Functional scope in the add-in

You create the report templates in an Excel add-in. For this purpose, Reporting offers the following functions in the add-in:

- Selection of the data source of the Runtime project:
  - Online: By means of a connection to a server on which the project is running
  - Offline: By means of a configuration file
- Definition of single value segments and time series segments
- Selection of the data source items of the segments
  Possible data source items:
  - Logging tags and online tags
  - Log alarms
- Definition of the report period (absolute or relative)
- Creation of type-specific data source item configurations
- Execution of individual segments or all segments for test purposes



Functional scope in Runtime

In runtime, you configure report tasks in the "Reporting" control that are based on the templates defined in the add-in. To do so, reporting offers the following functions in Runtime:

- Maintenance of task parameters, especially import and export of report templates
- Creating new report tasks and managing existing report tasks
- Overview of the generated reports
- Download or deletion of the reports

15.1.2 Basics of Reporting (RT Uni)

Report templates

A *report template* is an Excel file that was created with the WinCC Unified Excel add-in. For each report template, you set the data source and the options that are used by the template. You also define which segments of the reports are using the template and which data source items are evaluated by the segments.

After you have imported the report templates into the "Reporting" control in runtime, you can select them for configuring the report tasks.

Data sources

The *data source* is the source from which you select data source items when you configure the report template.

The following connection modes and data sources are available:

- Connection mode: Online
  Data source is the project that is running on the Runtime server to which the report template is connected.
- Connection mode: Offline
  Data source is a configuration file. You create the configuration file by connecting a report template to a runtime server and exporting the project running there to a file. You can use this file to create additional report templates without connecting to a runtime server.

Options and data source items

Options control the types of data source items to which the report template has access.

Data source items are the specific objects whose data is read from the Runtime project during report generation.

The following options and types of data source items are available in Reporting, depending on the installed software:

Software	Option	Types of data source items
WinCC Unified basic installation	Alarm	Log alarms
WinCC Unified basic installation	Logging tag	Logging tags
WinCC Unified basic installation	Tag	Online tag

Software	Option	Types of data source items
WinCC Unified basic installation	User-defined column	User-defined texts or Excel formulas
Performance Insight option package	Performance Insight	Local KPIs and operands of the PI option Performance insight: • KPI • Counters • Cycle time • Machine state • Design speed

Report tasks and task parameters

A *report task* is a job for generating 1 to N reports. *Task parameters* define the details of the generation, for example, the trigger of the report task and which report template the report task uses. When a report task is triggered, a new report is automatically generated.

Reports

A *report* is an Excel file that reads in data from the Runtime system and displays it in form of a table.

Reports are generated:

- Automatic
  When the trigger defined for a report task occurs.
- Manual
  When you execute report tasks manually in the "Report tasks" tab.

The report is created automatically when its report task is executed.

15.2 Procedure (RT Uni)

Project planning in the engineering system

1. Configure the alarms and tags for which you want to generate reports.
2. Place the "Reporting" control in a screen of a WinCC Unified device.
3. Compile the project and load it to the Runtime environment.

Definition of report templates in the Excel add-in

Proceed as follows to define report templates that can be used in runtime for report tasks:

1. Select the data source of the Runtime project.
2. Define templates that use the configured alarms and tags as data source items.
3. Optional: When using an online connection, test the template by reading the runtime data of selected segments or all segments.

Working with reports in runtime

To work with reports in runtime, follow these steps:

1. Configure the task parameters. To do this, import templates that use alarms and tags as data source items.
2. Configure the report tasks. Select one of these templates as the template.

   Note

   Generating reports

   The execution of a report task generates a report. Report tasks are executed automatically when the trigger defined in their task parameters is initiated. You can also have the option to execute report tasks manually.

3. Get an overview of the reports that were generated.
4. Download the reports, if necessary.

See also

Running a report job manually (Page 970)

Configuring report templates (Page 940)

15.3 Configuring production reports in the engineering system (RT Uni)

15.3.1 Inserting a "Reporting" control in a screen (RT Uni)

Procedure

1. Select the HMI device on the "Devices" tab.
2. Open the "Screens" folder.
3. Open the screen.
4. In the "My controls" pane, select the "Reporting" control and place it on the screen.

15.4 Creating templates for production reports (RT Uni)

15.4.1 Requirements (RT Uni)

15.4.1.1 Installation of the Reporting add-in (RT Uni)

Installing Excel manifest on a computer

Procedure

1. In the installation package of WinCC Unified on "DVD_2", double-click the file "Support \Reporting\SIMATIC_WinCC_Unified_Reporting_<Version number>.exe".
2. Select the target directory to which the underlying ZIP file is extracted and confirm your input. The ZIP file is extracted and setup starts automatically.

> **Note**
>
> **Start setup manually**
>
> To start the setup manually after the file was extracted, select the option "Extract the setup files without being installed".
>
> Start the setup later by running the "Setup.exe" file as administrator in the target directory.

3. Follow the setup instructions.
4. In the "Configuration" step, select the options for the Excel add-in and the PDF add-in.
5. Select whether PDF generation of the reports runs via Excel or Libre Office.
6. When Excel creates the PDFs, specify the user name and the password under which the PDF is created.
   Use a user that does not exist in the Windows user administration yet.
7. When Libre Office creates the PDFs, select the installation directory of Libre Office.
8. Click "Next" and follow the setup instructions.

Setting up read access to the Excel manifest

Give the users that create templates with the Excel add-in read access to the installation path of the Excel manifest: <target directory>\WinCCUAReporting\Excelmanifest

Adding add-in in Excel

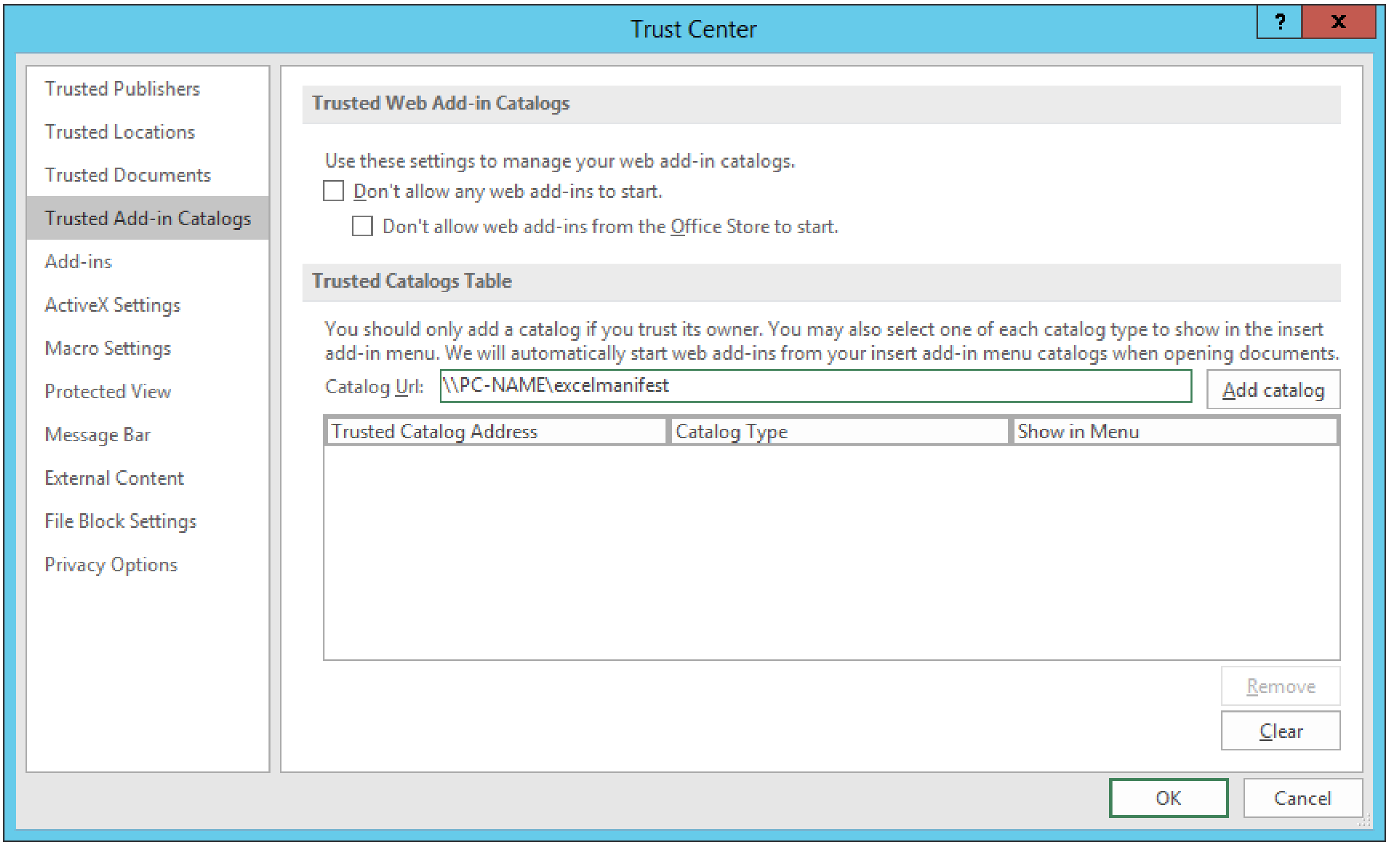
Requirement

The following software is installed on the device on which you want to use the Excel add-in:

Excel	Operating system	Browser
MS Excel 2016 - Build 16.0.6769 (32- or 64-bit)	Windows 10 Version < 1903	Internet Explorer 11
	Windows 10 Version > 1903	
	Windows 10 Version >= 1903	Microsoft Edge

Procedure

1. Open Microsoft Excel.
2. Open the "Trust Center" under "File" > "Options".
3. Click on "Trust Center Settings".
4. Click on "Trusted Add-In Catalogs".
5. Add the catalog using the URL "\\<Computer name>\excelmanifest".



Trust Center

Trusted Web Add-in Catalogs

Use these settings to manage your web add-in catalogs.

□ Don't allow any web add-ins to start.

□ Don't allow web add-ins from the Office Store to start.

Trusted Catalogs Table

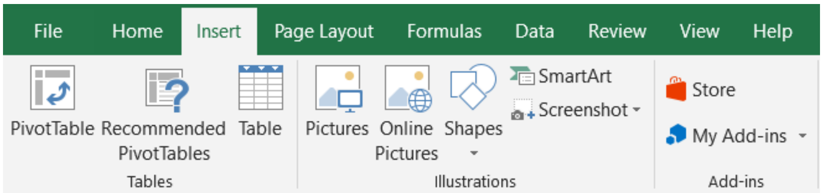
You should only add a catalog if you trust its owner. You may also select one of each catalog type to show in the insert add-in menu. We will automatically start web add-ins from your insert add-in menu catalogs when opening documents.

Catalog Url: \\PC-NAME\excelmanifest

Trusted Catalog Address	Catalog Type	Show in Menu

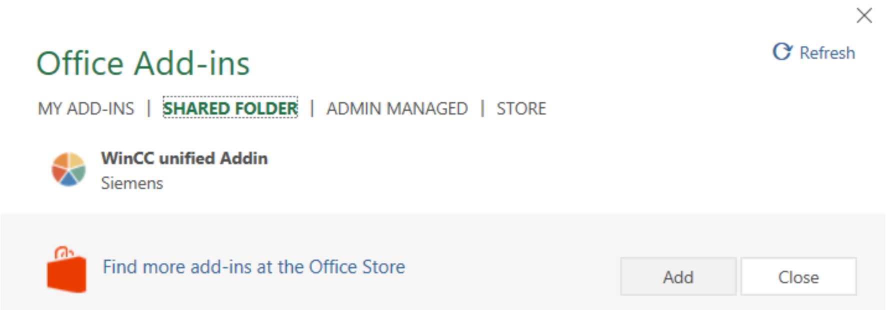
6. Make sure that the check mark in the "Show in Menu" column is set.
7. End and restart Excel.

8. In the "Insert" menu, click "My Add-ins".



In the "Office Add-ins" dialog box, the Siemens add-in is displayed under "Shared folders".

9. Select the add-in and click on "Add".


Office Add-ins
Refresh
MY ADD-INS | SHARED FOLDER | ADMIN MANAGED | STORE
WinCC unified Addin
Siemens
Find more add-ins at the Office Store
Add
Close

15.4.1.2 Configuring Internet Explorer and Edge (RT Uni)

The Reporting Excel add-in uses the certificate that was selected during installation of WinCC Unified Runtime or later in "WinCC Unified Configuration".

Some browsers do not consider self-signed certificates as trusted. If you use a self-signed certificate for WinCC Unified Runtime, you must add the certificate to the list of trusted certificates in Internet Explorer or Edge on the device on which the Excel add-in is installed.

For detailed information on handling certificates, see the Runtime Readme.

Procedure

The following section describes the procedure for adding a self-signed certificate to the list of trusted certificates, using Internet Explorer as an example:

1. Start Internet Explorer.
2. In the address line, enter the host name or the IP when creating the certificate.
   You will receive a security warning.
3. Click "Continue to this website (not recommended)".
4. Click "Install certificate".
5. Click "Place all certificates in the following store" and "Browse".
6. Click "Trusted Root Certification Authorities" followed by "OK".

Note

Do not use the preset options for automatic selection of the certificate store.

7. Exit the dialog.
8. If you receive a security warning as to whether you want to trust the certificate, confirm it with "Yes".
9. Load the page again.

See also

MicrosoftHelp_IE_ZertifikatInstallieren (https://medium.com/@ali.dev/how-to-trust-any-self-signed-ssl-certificate-in-ie11-and-edge-fa7b416cac68)

15.4.2 Setting up a data source (RT Uni)

15.4.2.1 Using an online connection (RT Uni)

Setting up an online connection (RT Uni)

Requirements

- The Runtime server is accessible.
- WinCC Unified Runtime and the desired options are installed on the server.
- A project with the desired project data is available on the server.
- The project is downloaded to Runtime and is in RUN.
- The add-in is not connected to a server.

Setting up connection settings

1. In the "Data sources" group, click on "Connections" in the "WinCC Unified" tab.
2. Under "Connections", click on "Online".
3. Under "Server", enter the server name or the IP.
4. Click "Load".
5. Select the desired options.
6. Confirm your entries.

Result

The server node is created and the connection is established.

You can edit the connection settings at a later time, for example, to add an option.

Diagnostics

If no connection can be established or an incorrect server name has been entered, the add-in will display a corresponding error message.

Editing an online connection (RT Uni)

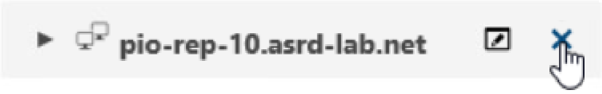
You can edit the connection settings to the connected server.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A connection to a server is set up as data source.

Deleting a connected server

1. In the "Data sources" group, click on "Connections".
2. Under "Connections", click on "Online".
3. Select the server node.
4. Click "Delete" next to the server node.



Changing a connected server

To connect Reporting to a server other than the one currently set up, follow these steps:

1. In the "Data sources" group, click on "Connections".
2. Under "Connections", click on "Online".
3. Delete the existing server connection.
4. Set up a new server connection.

Adding an option

1. In the "Data sources" group, click on "Connections".
2. Under "Connections", click on "Online".
3. Select the server node.
4. Click "Edit" next to the server node.
5. Select the desired option.
6. Confirm your entries with "OK".

Deleting an option

1. In the "Data sources" group, click on "Connections".
2. Under "Connections", click on "Online".
3. Select the server node.
4. Select an option under the server node.
5. Deactivate the option.
6. Confirm your entries.

See also

Setting up an online connection (Page 937)

15.4.2.2 Using an offline connection (RT Uni)

Creating a configuration file in the add-in (RT Uni)

Requirement

An online connection is loaded.

Procedure

1. In the "Data sources" group, click on "Connections" in the "WinCC Unified" tab.
2. Under "Connections", click on "Online".
3. Click "Download configuration".
4. Under "Options", select which options are to be part of the configuration file.
5. Click "Download".
6. Determine the storage location of the configuration file in the "Show downloads - Internet Explorer" window.

Result

The configuration file is stored in JSON format in the specified directory.

To use the configuration file for a report template, download the configuration file to the add-in.

See also

Setting up an online connection (Page 937)

Setting up an offline connection (Page 940)

Setting up an offline connection (RT Uni)

Requirement

A configuration file is available on the device.

Procedure

1. In the "Data sources" group, click on "Connections" in the "WinCC Unified" tab.
2. Under "Connections", click on "Offline".
3. Click "Open offline configuration".
4. Select the desired file in the window that opens and confirm your entries.
5. Click "Load".
6. Select the desired options.
7. Confirm your entries.

Result

The configuration file is loaded to the add-in. The data of the configuration file is available for configuring the report template.

See also

Creating a configuration file in the add-in (Page 939)

15.4.3 Configuring report templates (RT Uni)

15.4.3.1 Sequence of events (RT Uni)

Requirement

An online connection or offline connection has been established.

Procedure

To create a new report template, proceed as follows:

1. Open a new Excel file.
2. Add a segment.
   You can choose between time series segments and single value segments.
3. Add data source items to the segment.
   The exact procedure depends on the type of the data source item.

4. Optional: If you do not want the data source item to use the default configuration, specify its configuration.
   You have the following options:
   – Select an existing configuration.
   – Create a new configuration and select it.
   – Define a local configuration.
5. Optional: To define additional segments, repeat steps 2 to 4.
6. Optional: When using an online connection, test the template by reading the runtime data of selected segments or all segments.

15.4.3.2 Create segments (RT Uni)

Definition

A report template consists of any number of segments. Each *segment* is a container to which you can add any number of data source items. The segment reads the data from its data source items.

You can choose between time series segments and single value segments.

Time series segments

Time series segments consist of a legend table and a data table:

- The legend table lists general information about the data source items of the segment.
  Example of logging tags: Name, Option, Parent and Description
- The data table lists several values for each data source item in the segment.
  Example of logging tags: All values logged for the tags in the evaluation period, including their time stamp.

Possible data source items:

- Log alarms
- Logging tags
- User-defined columns

Single value segments

Single value segments consist of a data table that lists exactly one value for each data source item in the segment.

Note

Output additional information

For the data source items of the single value segment, you can set in the configurations of the data source items whether the data table outputs additional information about the value.

Example of logging tags:

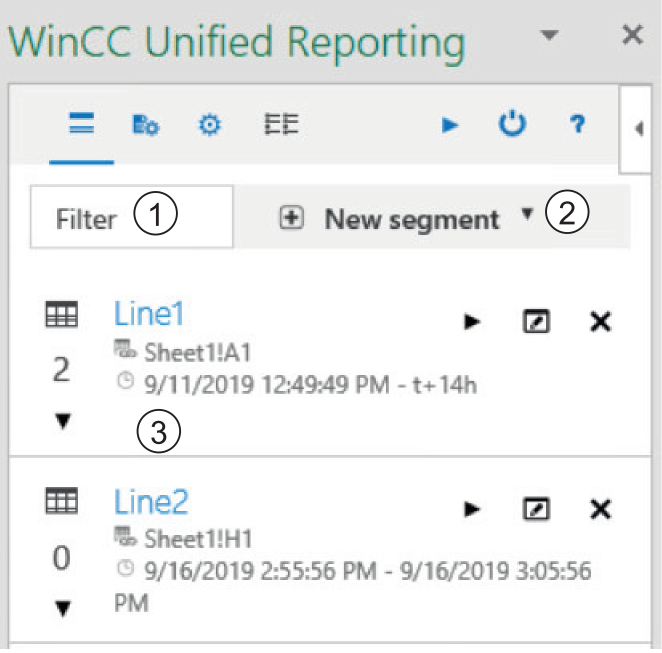
- Quality code and time stamp of the tag value
- Labels

Possible data source items:

- Logging tags
- Online tags

User interface

The interface for creating and editing segments has the following structure:



1 Filter

Filters the list of segments by name.

2 Button for creating a segment

3 List of segments

Each segment has buttons for reading in, editing and deleting the segment.

The following configuration is displayed for each segment:

- Segment name
- Number of data source items
- Insertion location of the segment in the Excel file
- Time interval

A click on the segment opens the area with the data source items.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- The data source is set up.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "New segment".
3. Select "New time series segment" or "New single value segment".

4. Enter a segment name. Note the Excel restrictions for naming tables (for example, do not use blanks).
5. For a time series segment, make the following settings in addition:
   - Under "Location", determine where the segment is inserted in the file. Enter the name of the worksheet and the cell.
     Alternatively, click and use the cell currently highlighted in the Excel file.

     Note

     Arranging segments horizontally

     Place the segments horizontally toward each other.

     Because the tables grow dynamically, tables can overlap in vertical placement. This causes an error of the class OfficeExtension.Error.

   - Under "Start" and "End", you determine the time period for which values are read into the segment.

	Absolute time information	Select a date and a time. The information is absolute to the current date.
	Relative time information	Select a reference time and a time interval. The information is relative to the current date.
	Read time information from cell	Applies the value of the cell currently highlighted in the Excel file. Make sure that the cell supplies a valid time.
	Read time information from tag	Applies the value of the set online tag. Make sure that the tag supplies a valid time. Possible data types: ● DateTime ● String ● Integer

6. Confirm your entries with "OK".

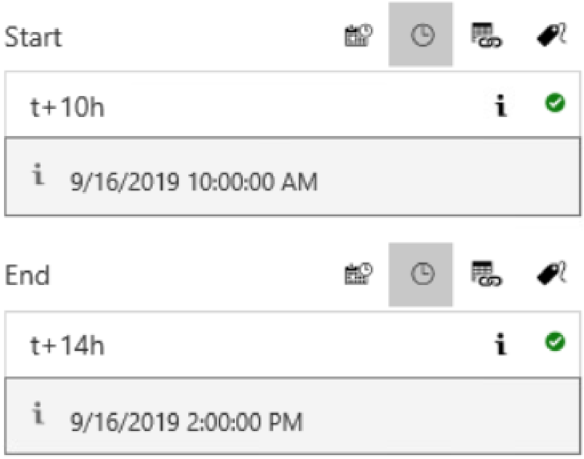
Result

The segment is created and added to the list of segments:

Next, add data source items to the segment. Your procedure depends on the type of the new data source item.

Format for relative time information

The relative times are entered using a reference time and a time interval.



Start

t+10h

9/16/2019 10:00:00 AM

End

t+14h

9/16/2019 2:00:00 PM

Reference time

Use one of the following characters for the reference time:

- "*" - Now
- "t" (today) - Today at midnight
- "y" (yesterday) - Yesterday at midnight
- "1-31" - Specific day of the current month

Time interval

- "y" (year): +1y = plus 1 year
- "mo" (month): +1mo = plus 1 month
- "w" (week): +1w = plus 1 week
- "d" (day): +1d = plus 1 day
- "h" (hour): +1h = plus 1 hour
- "m" (minute): +1m = plus 1 minute
- "s" (second): +1s = plus 1 second
- "ms" (milliseconds): +250ms = plus 250 milliseconds

Examples

- *-1y: One year ago today
- t+8h: Today at 8:00 am
- y+8h: Yesterday at 8:00 am

- 1+8h: The first day of the current month at 8:00 am
- *-1d: One day ago
- *-2h-30m-30s: 2 hours, 30 minutes and 30 seconds ago

See also

Adding data source elements (Page 947)

Working with configurations (Page 953)

15.4.3.3 Edit segments (RT Uni)

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Edit" next to a segment in the list of segments.
3. Edit the segment.
   You can make the same settings as when creating the segment.

15.4.3.4 Delete segments (RT Uni)

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Delete" next to a segment in the list of segments.
3. Confirm your entries with "OK".

15.4.3.5 Adding data source elements (RT Uni)

Add log alarms (RT Uni)

Requirement

- There are logging alarms in the project that runs on the connected Runtime server or is the basis of the configuration file.
- The "Alarm" option is activated in the connection settings.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Adding log alarms

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Alarm" option.
5. Select the "Alarm" entry under "Select alarms".
6. To remove alarms from the report, select the entry "Alarms" under "Selected data source items" and click "Delete".

   Note

   Change selection criteria

   After you have added notifications, you can change the selection criteria and add more data source items.

   For example: Output tags and alarms in the same segment.
7. Confirm with "OK".

Note

Displayed alarms

First, the data table shows all logging alarms of the project. You filter the alarms using the configuration of the data source item.

Result

The added data source item for alarms is displayed below the segment and inserted into the Excel file.

If you do not want the data source item to use the default configuration, select a configuration next.

See also

Create or edit configurations for an alarm (Page 953)

Select configuration (Page 957)

Working with configurations (Page 953)

Add logging tags (RT Uni)

Requirement

- The project on which the connected Runtime server runs or the basis of the configuration file has logging tags.
- The "Logging tag" option was selected while setting up of the connection.
- The "WinCC Unified" tab is visible in Excel.
- A single value segment or time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Logging tag" option.

5. Optional: To reduce the load time, limit which tags are loaded to the selection under "Fill filter form".
   The preset filters "*" return all logging tags of the project.
   – "Tag name": Enter the name of the online tag whose logging tags you want to add.
   – "Logging tag name": Enter the name of the logging tags you want to add.

   Note that the entry is case-sensitive.

   Note

   Filter by partial string

   You use the wildcard "*" to filter by partial strings.

   For example:
   - *T* returns all tags with a "T" in their name.
   - *T returns all tags that end in "T".
   - T* returns all tags that start with "T".

   When filtering for structures, the separators must be part of the filter string.

   For example: The following filters return the logging tags for all tags of the device "HMI_RT_1":
   - Filter for tag: "HMI_RT_1::*"
   - Filter for logging tag: "*"

6. Click "Load".
   The filters are applied to the logging tags contained in the project.
7. Optional: Further reduce the number of tags that are offered for selection by clicking next to "Select logging tag" and entering another filter string.
   The list of tags you are being offered is filtered while you type.
8. Select one or more tags under "Select logging tag".
   The tags are added to the "Selected data source items" list.

   Note

   Change selection criteria

   After you have added a tag, you can select a different option or a different filter and add additional data source items.

   For example: Output KPIs and logging tags in the same segment.

9. To remove one or more data source items from "Selected data source items", select them and click "Delete".
10. Confirm with "OK".
    The added logging tags are shown below the segment and added to the Excel table.
11. If you have added the logging tag to a single value segment:
    – In the Excel worksheet, select the cell in which the logging tag is to be inserted.
    – Click the "Select a cell" button on the data source item of the logging tag.
    Alternatively, enter the name of the worksheet and the cell.

See also

Create segments (Page 941)

Select configuration (Page 957)

Create or edit configurations for logging tags (Page 954)

Working with configurations (Page 953)

Adding online tags (RT Uni)

Requirement

- The project on which the connected Runtime server runs or that is the basis of the configuration file has online tags.
- The "Tag" option was enabled when the connection was set up.
- The "WinCC Unified" tab is visible in Excel.
- A single value segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select the single value segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Tag" option.
5. Optional: To reduce the load time, limit which tags are loaded to the selection under "Fill filter form".
   Under "Tag name", enter a filter, e.g. the name of the online tag. Note that the entry is case-sensitive.
   The preset filter "*" returns all online tags of the project.

   Note

   Filter by partial string

   You use the wildcard "*" to filter by partial strings.

   For example:

   - *T* returns all tags with a "T" in their name.
   - *T returns all tags that end in "T".
   - T* returns all tags that start with "T".

   When filtering for structures, the separators must be part of the filter string.

   For example: The "HMI_RT_1::*" filter returns all online tags of the "HMI_RT_1" device.

6. Click "Load".
   The filters are applied to the online tags contained in the project.

7. Optional: Further reduce the number of tags that are offered for selection by clicking next to "Select tag" and entering another filter string.
   The list of tags you are being offered is filtered while you type.
8. Select one or more tags under "Select tag".
   The tags are applied to the "Selected data source items" list.

   Note

   Change selection criteria

   After you have added a tag, you can select a different option or a different filter and add additional data source items.

9. To remove one or more data source items from "Selected data source items", select them and click "Delete".
10. Confirm with "OK".

The added online tags are shown below the segment and added to the Excel file.

See also

Creating or editing configurations for online tags (Page 956)

Working with configurations (Page 953)

Adding user-defined columns (RT Uni)

Introduction

User-defined columns supplement the data of the other data source items of a time series segment with additional information:

- With a fixed string
  The string appears in each cell of the column.
  Example: Display measurement unit of the tag values in report
- With a formula
  The formula is calculated during generation for each cell in the dynamic column.
  Example: The sum of the tag values output in the report.

The configuration of the user-defined column controls which string or formula it uses.

Requirement

- The "User-defined column" option was enabled when the connection was set up.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the option "User-defined column".
5. Enter the name of the column under "name".
6. Click "Select" or press <ENTER>.
   The column is included in the list "Selected data source items".

   Note

   Change selection criteria

   After you have added a column, you can select a different option or a different filter and add additional data source items.
7. Select a configuration for the user-defined column.
8. To remove one or more data source items from "Selected data source items", select them and click "Delete".
9. Confirm with "OK".

The added columns are shown below the segment and added to the Excel table.

See also

Creating and editing configurations for user-defined columns (Page 956)

Working with configurations (Page 953)

15.4.3.6 Delete data source elements (RT Uni)

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source element is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Expand a segment by clicking on it.
   The area for adding and editing data source elements appears.
3. Move the mouse pointer over a data source element and click "Delete".

15.4.3.7 Working with configurations (RT Uni)

Basics of configuration (RT Uni)

The *configuration* of a data source item defines how the values of the data source item are calculated and displayed in a segment.

There are specific configuration settings for each data-source-item type.

Data source items used in time series segments use a different configuration than data source items used in single-value segments.

You have the following options:

- Use a default configuration.
  There is a default configuration for all types of data source items. Once added, data source items use the default configuration of their type.
  You can edit the default configurations.
- Use user-defined configuration.
  You can create any number of user-defined configurations for all types of data source items.
  You can select one of the user-defined configurations on the data source item.
- Overwrite a configuration locally.
  You can overwrite the configuration selected at the data source item locally.

Create or edit configurations for an alarm (RT Uni)

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item segment configuration".
3. Click "New segment > Log alarm configuration".
4. Enter the name of the configuration under "Name".
5. To determine which alarm properties are displayed, activate the options for the desired columns under "Columns".
6. To filter which logging alarms are displayed, define a filter query. The filter query can consist of any number of conditions.
   Follow these steps:
   – Under "Filter query", click "+" or "Add new condition line".
   – Select an alarm property, an operator, and enter a value.
   – Optional: Use "+" or "Add new condition row" to create further conditions and select whether the conditions are to be linked with a logical AND or OR.

7. Activate the option "Use system colors" so that the alarms are highlighted with the same colors as in the alarm control.
8. Confirm your entries with "OK".

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item segment configuration".
3. Click a configuration for logging alarms.
4. Edit the configuration settings. You have the same options as when creating the configuration.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the runtime data.

See also

Select configuration (Page 957)

Calculation modes for data source elements (Page 960)

Create or edit configurations for logging tags (RT Uni)

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item segment configuration".
3. Click "New segment".
4. To create a configuration for logging tags in a time series segment, select the entry "Logging tag configuration".
   To create a configuration for logging tags in a single value segment, select the entry "Single value configuration logging tag".
5. Set the settings for the configuration.
6. Confirm your entries with "OK".

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item segment configuration".
3. Click a configuration for logging tags.

4. Edit the configuration settings.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the runtime data.

Settings for time series segments

The following settings are available for logging tags in time series segments:

Setting	Description
"Name"	Enter the name of the configuration.
"Calculation mode"	Select which data are to be written if there is no current value.
"Interval"	Only for the calculation modes "Keep last value" and "Interpolate".
"Show quality code"	Select whether the quality code is output with the value.

Settings for single value segments

The following settings are available for logging tags in single value segments:

Setting	Description
"Name"	Enter the name of the configuration.
"Time stamp"	Determine the date and time for which the value is read.
"Calculation mode"	Determine which data is to be written if there is no current value.
"Show captions"	Define whether a header is displayed in the columns for the time stamp, the data source item and the quality code.
"Show time stamp"	Determine whether and where this information is displayed in the table. The information is always in relation to the value cell.
"Show data source item"	
"Show quality code"	

Possible values for "Time stamp":

📅	Absolute time information	Select a date and a time. The information is absolute.
🕒	Relative time information	Select a reference time and a time interval. The information is relative to the current date.
🔗	Read time information from cell	Applies the value of the cell currently highlighted in the Excel file. Make sure that the cell supplies a valid time.
🏷	Read time information from tag	Applies the value of the set online tag. Make sure that the tag supplies a valid time. Possible data types: • DateTime • String • Integer

Creating or editing configurations for online tags (RT Uni)

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item segment configuration".
3. Click "New segment > Single value configuration tag".
4. Set the settings for the configuration.
5. Confirm your entries with "OK".

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item segment configuration".
3. Click a configuration for online tags.
4. Edit the configuration settings.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the runtime data.

Settings for single value segments

The following settings are available for online tags in single value segments:

Setting	Description
"Name"	Enter the name of the configuration.
"Show captions"	Select whether a header is displayed in the columns for the time stamp, the data source item and the quality code.
"Show time stamp"	Select whether the time stamp is output with the value.
"Show data source item"	Select whether the quality code is output with the value.
"Show quality code"	Select whether the quality code is output with the value.

Creating and editing configurations for user-defined columns (RT Uni)

Requirement

- The "WinCC Unified" tab is visible in Excel.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item segment configuration".
3. Click "New segment > Configuration of user-defined column".
4. Enter the name of the configuration under "Name".
5. Under "Formula", select one of the following options:
   - Enter a fixed string.
     The string is transferred into each cell of the column.
   - Enter an Excel formula.
     The formula is copied into each cell of the user-defined column and adapted to the respective row.
     To prevent a part of the formula from being adjusted, place the character "$" in front of it.
     Example

Formula in configuration		=B2+C2	=B$2+C2
Adapting the formula in the report	in line 2	=B2+C2	=B2+C2
	in line 3	=B3+C3	=B2+C3
	in line 4	=B4+C4	=B2+C4

Note

No validity check

The formula is not tested for correctness during either input or dynamic adaptation.

6. Confirm your entries with "OK".

Select configuration (RT Uni)

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source item is available.
- There is a user-defined configuration for the type of the data source item.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Select the segment.
   The segment is extended by the area for the data source items.
3. Select the desired configuration from a data source item in the drop-down list.
4. Click "OK".

Result

The changes are applied the next time you read in the runtime data.

Define local configuration (RT Uni)

A local configuration is only available at the data source item where it was entered.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source item is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Select the segment.
   The segment is extended by the area for the data source items.
3. Move the mouse over a data source item and click "Edit".
   You create a local configuration that first adopts the values of the original configuration.
4. Enter a name for the local configuration.
5. Change the desired settings.
6. Confirm your entries with "OK".

Result

The changes are applied the next time you read in the runtime data.

Delete configuration (RT Uni)

Requirement

A configuration is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item segment configuration".

3. Move the mouse to a configuration.

   Note

   Default configurations cannot be deleted

   You can edit default configurations but not delete them.

4. Click "Delete".

Result

- The configuration is deleted.
- Data source elements with this configuration obtain a local configuration with the same settings.

15.4.3.8 Reading Runtime data in Excel (RT Uni)

Importing runtime data into Excel is only possible with an online connection.

Requirement

An online connection is established.

Reading in all segments

1. Select "WinCC Unified > Segments".
2. Click "Update all" ▶.

Reading in individual segments

1. Select "WinCC Unified > Segments".
2. Click "Update" ▶ next to a segment in the list of segments.

Result

The segment or segments are run. Your data is read into Excel.

You can further process the read values with the known Excel functions, for example, by outputting them as a diagram.

When you save the Excel file, your segments and the read out values are retained.

Diagnostics during the data query

Successful execution of the data query is documented by the add-in with a status message in the table:

If an error occurs during the data query, a general error message is displayed under status. In addition, detailed error messages are displayed in the "ErrorLog" worksheet.

15.4.3.9 Calculation modes for data source elements (RT Uni)

If there is no current value for a data source item for a requested point in time, the following calculation modes are available.

Calculation modes for tags

The following calculation modes are available for tags of a time series segment:

Calculation mode	Description
Raw	The actual value available for the specified period. If no data are available, no value is output.
Keep last value:	If no data are available, the last value is used. With this mode you can also use values with an invalid quality code.
Interpolate	The values are interpolated linearly for the specified time period. With this mode you can only use values with a valid quality code.

The following calculation modes are available for tags of a single value segment:

Calculation mode	Description
Interpolate	The values are interpolated linearly for the specified time period. With this mode you can only use values with a valid quality code.
Left	If no data are available, the last value to the left of the specified time period is used.
Right	If no data are available, the last value to the right of the specified time period is used.

15.4.4 Making general settings (RT Uni)

15.4.4.1 Changing the language (RT Uni)

Changing the add-in language

The Excel add-in automatically uses the same interface language as Excel. If you are using a language for Excel that is not included in the Unified options, English is used as the default language.

You can select the language for the contents of the report independently of the interface. To select another language, the language must be configured in Runtime.

Selecting the language for the report

1. Select "WinCC Unified > Segments".
2. Click ⚙ "Settings".
3. Under "Runtime language", select the language of the report content.
4. Under "Query language" you select which language data queries have that require user input, e.g. filter definitions.

15.4.4.2 Adapting the work area (RT Uni)

Undocking and moving the add-in

To enlarge your workspace, you can undock the Excel add-in:

1. Open the drop-down list in the header of the add-in.
2. Click "Move".
3. Move the mouse pointer to the desired location and click the left mouse button.
4. To move the add-in again, keep the left mouse button pressed in the header of the add-in and move the mouse.
5. To dock the add-in again, double-click in the header of the add-in.

Adapting the size of the add-in

1. Open the drop-down list in the header of the add-in.
2. Click "Size".
3. Move the mouse pointer to the left to make the add-in wider or to the right to make it narrower.
4. Left-click when you have reached the desired size.

15.4.4.3 Zooming in the add-in (RT Uni)

Procedure

To zoom in or out of the display in the add-in, press <CTRL> and move the mouse wheel.

15.4.5 Undo and redo (RT Uni)

The Excel functions "Undo" and "Redo" are not available in the add-in.

15.5 Working with production logs in runtime (RT Uni)

15.5.1 The user interface of the "Reports" control (RT Uni)

Note

With version V16, the "Reports" control is supported only for Unified PC. If you use the control under Unified Comfort Panel, an error message of the compiler is returned. Existing projects under Unified Comfort Panel in which the control is configured must delete the control before compilation to version V16.

Note

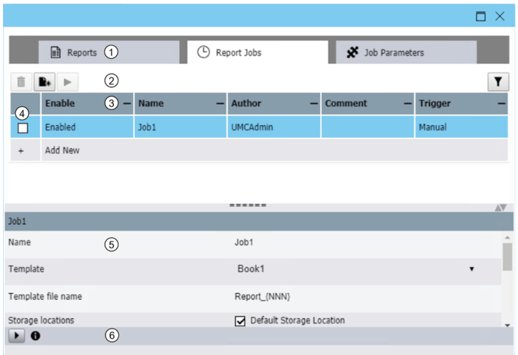
Automatic data transfer

Changes in the "Reports" control are saved automatically.

Layout

You create and manage report jobs in the "Reports" control. You also have access to the reports generated by the report jobs.

The control has the following structure:



	Enable	Name	Author	Comment	Trigger
□	Enabled	Job1	UMCAdmin		Manual
+	Add New				

Job1

Name	Job1
Template	Book1
Template file name	Report_{NNN}
Storage locations	☑ Default Storage Location

1 Tabs for configuring and managing reports, report jobs and job parameters

2 Toolbar

The buttons you see depends on the tab.

3 Work area

A list of elements available in the tab

4 Options to select elements

5 Detail area

Shows the properties of the selected element.

6 Status bar

Tab

"Reports" tab

Here you can see which reports have already been generated. You can download reports or delete them from the toolbar.

"Report jobs" tab

Here you create new report jobs, manage existing report jobs or start a report job manually.

"Job parameters" tab

Here you manage the parameters with which you configure the report jobs in the "Report jobs" tab.

Toolbar

The following buttons are available in the toolbars of the tabs:

Icon	Button	
🗑	Delete	Deletes the elements whose option is enabled in the work area.
📄✱	• Add new • Import	• Creates a new element. • "Job parameters > Templates" tab: To import one or more templates to runtime.
▶	Run	In the "Report jobs" tab. Manually creates reports for the report jobs whose option is enabled in the work area.
📄→	Export	• In the "Job parameters > Templates" tab: To export templates • In the "Reports" tab: To download reports onto the client

Status bar

The button in the status bar displays general information sent by the reporting service, for example on whether a report job has been executed.

15.5.2 Configuring task parameters (RT Uni)

Job parameters define the details of a report job.

You configure the following parameters on the "Job parameters" tab:

- Templates
- Trigger
- Storage location
  The storage location determines where the reports generated by the execution of the report job are stored on the Runtime server.
  Storage location is the default project directory.
  The parameter is read-only.

You define the remaining job parameters while configuring a report job in the "Report jobs" tab.

See also

Import and export templates (Page 964)

Deleting templates (Page 965)

Configure trigger (Page 965)

15.5.2.1 Import and export templates (RT Uni)

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Templates" tab is visible in the control.
- Import: You have access to the storage location of the templates.
- Export: Templates have been imported into the control.

Importing templates

1. Click "Add new" in the toolbar.
   Alternative: In the work area, click "Add new".
2. Select one or more template files in the dialog box that opens.
3. Confirm your input.

   Note

   No validation

   The templates are not validated during import.

4. Optional: In the work area, select one of the imported templates and enter a comment describing the template in the detail area.

Exporting templates

1. In the work area, select the options next to the templates you want to export.
2. Click "Export" in the toolbar.

The templates are downloaded to the download folder or a user-defined directory according to the device settings.

15.5.2.2 Deleting templates (RT Uni)

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Templates" tab is visible in the control.
- Templates have been imported into the control.

Procedure

1. In the work area, select the options next to the templates you want to delete.
2. Click "Delete" in the toolbar.

Deleting used templates

The "In use" column shows whether the template is used by a report job.

If you delete a template that is used by a report job, the report job is marked as inconsistent and no longer executed.

15.5.2.3 Configure trigger (RT Uni)

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Trigger" tab is visible in the control.

Add trigger

1. In the work area of the tab, click "Add new".
   A new trigger is created and displayed in the detail area.
2. Assign a unique name to the trigger.
3. Set the trigger type:
   - "Tag trigger"
     Report jobs with this trigger are executed when a configured value condition occurs for a tag.
   - "Serial trigger"
     Report jobs with this trigger are executed in the user-defined period as standard.

4. For triggers with "Variable trigger" trigger type:
   - Click "Select tag".
   - Click "Load".
   - Select the required tag and click "OK".
   - Set the condition and the condition value.
     Example:

Set tag	Type UInt32
Condition	>
Condition value	100

The trigger will be initiated when the tag receives a value greater than 100.

5. For triggers with "Serial trigger" trigger type:
   - Select the serial pattern.
     The series pattern defines the occurrence and time at which the trigger is initiated.
     Example: Weekly > Every 2 weeks > Fridays
   - Select the series area.
     The series range defines the period in which the trigger is initiated.

"Start"	Specify the start date
"Time"	Specify the time at which the trigger is initiated.
"End on"	Specify the end date. The trigger will be executed for the last time on this day.
"End after"	Determine the number of dates after which the series ends.
"No end date"	The series runs indefinitely.

Note

Generate report once

If a report is to be generated automatically exactly once at a fixed time, select the start date, the time and set the value "1" under "End after".

6. Optional: Enter a comment for the trigger.

Delete trigger

Activate the option of the desired trigger in the work area of the tab and click "Delete" in the toolbar.

Edit trigger

1. Activate the option of the desired trigger in the work area of the tab.
2. In the detail area, edit the settings of the trigger.

Note

No change of the trigger type

The trigger type can only be set when adding the trigger.

15.5.3 Configuring report tasks (RT Uni)

15.5.3.1 Creating a report job (RT Uni)

A report job is a job for generating reports. The configuration of a report job controls the details of the generation.

N reports can be generated for each report job. Each time a report job is executed, a new report is generated.

A report job is executed in the following cases:

- Automatic
  This is the case if the trigger defined for the report job occurs.
- Manual
  When you execute report jobs manually in the "Report jobs" tab.

Requirement

- The "Reports" control is configured on a screen of the project.
- Job parameters have been configured in the control:
  - At least one template has been imported.
  - To automatically execute a report job: Triggers are configured in the "Job parameters > Trigger" tab.
- For a report job that generates reports as PDF files: During the reporting setup, the PDF add-in was installed and the information required for PDF creation was provided.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. Select "Add new" In the work area or click "Add new" in the toolbar.

3. In the detail area, enter a name for the report job.
4. In the detail area, configure the parameters of the report job:
   - Template
     Select the template on which the report generated by the report job is based.
   - Target name
     Enter the name of the generated reports.
     Use placeholders to dynamize the name and generate unique names.
     The string `LineA_{YYYYMMDD}_{HHMMSS}_{NNN}`, for example, generates a name, consisting of a descriptive part, a time stamp and a counter: `LineA_20181210_170641_667`
   - Storage location
     Read-only: Project folder
     The directory for the generated reports.
   - Trigger
     If the report job is only to be executed manually, select "Manual".
     If the report job is to be executed automatically, select the required trigger.
   - Target type
     Define the target format of the report. You have the following options:

Format	Requirement
Excel	None
Excel and PDF	Microsoft Office Excel 2016 (Build 16.0.6769 or higher) or Libre Office with a PDF plug-in are installed on the Runtime server
PDF	

   - Comment
     Describe the report job in more detail.

Result

The report job is saved automatically. When it is triggered, a report is generated.

List of placeholders

The following placeholders are available for defining report names

Place-holder	Description	Example		Area
		Config-uration	Result	
{N}	Automatic enumeration	Rep_{N}	Rep_1	1..9
{NN}	Automatic enumeration	Rep_{NN}	Rep_01	01..99
{NNN}	Automatic enumeration	Rep_{NNN}	Rep_001	001..999
{YYYY}	Current year with 4 digits	Rep_{YYYY}	Rep_2018	Valid year with 4 digits

Place-holder	Description	Example		Area
		Config-uration	Result	
{YY}	Current year with 2 digits	Rep_{YY}	Rep_18	Valid year with 2 digits
{MM}	Current month	Rep_{MM}	Rep_12	Valid month with 2 digits
{DD}	Current day of the month	Rep_{DD}	Rep_10	Valid day with 2 digits
{HH}	Current hour	Rep_{HH}	Rep_17	Valid hour with 2 digits
{MM}	Current minute	Rep_{MM}	Rep_06	Valid minute with 2 digits
{SS}	Current second	Rep_{SS}	Rep_41	Valid second with 2 digits

See also

Running a report job manually (Page 970)

Configure trigger (Page 965)

15.5.3.2 Managing report jobs (RT Uni)

Requirement

- The "Reports" control is configured on a screen of the project.
- Report jobs have been configured in the control.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. To edit a report job, proceed as follows:
   – Select the report job in the work area.
   – In the detail area, edit the settings of the report job.
   You have the same options as when creating a report job.
3. To delete report jobs, proceed as follows:
   – In the work area, enable the options next to the report job.
   – Click "Delete" in the toolbar.

15.5.4 Running a report job manually (RT Uni)

Requirement

Report jobs have been configured in the "Reports" control.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. In the work area, select the options next to the report jobs you want to run manually.
3. Click "Run" in the toolbar.

Result

The reports are generated. You can download them in the "Reports" tab.

15.5.5 Downloading reports (RT Uni)

You have the option of downloading the reports generated by the report jobs to your device.

Depending on which file formats have been set in the report job, you can download the report either as an Excel file or as a PDF file.

Requirement

- Report jobs have been configured and executed in the "Reports" control.

Procedure

1. Select the "Reports" tab in the "Reports" control.
2. In the "Files" column, select the target format for the desired reports.
3. For each report that you want to download, select the options to the left of the report.
4. Click "Export" in the toolbar.

The reports are downloaded into the download directory of the browser.

You can further edit, distribute, or log the reports. If a report has the target format .xlsx, you can add the report's data to a pivot table in Excel.

15.5.6 Inconsistencies and error diagnostics (RT Uni)

Note

Inconsistent report jobs are not executed.

The templates available in the "Reports" control are not validated.

Display of inconsistencies and errors

Errors and inconsistencies are displayed as follows:

In the control	If job parameters are affected. Examples: • No template is set for a report job. • A tag that triggers a report job is deleted in the engineering system. The project is reloaded into the device.
In the "Error log" worksheet of the report	Errors or inconsistencies affecting the content of the report. Example: The report evaluates data from a tag that is no longer available in runtime.
As system alarm	For errors and inconsistencies that do not affect job parameters or the contents of the report. Example: The ExecuteReport system function transfers a report job that does not exist.

Job parameters

The following values lead to errors and inconsistencies:

Parameter	Invalid values	Default setting
"Name"	Zero, empty or already assigned name	"New report job"
"Template"	Zero, empty or "None". Name of a template that is not imported	"None"
"Target name"	Zero or empty	"NewReportJob[NN]"

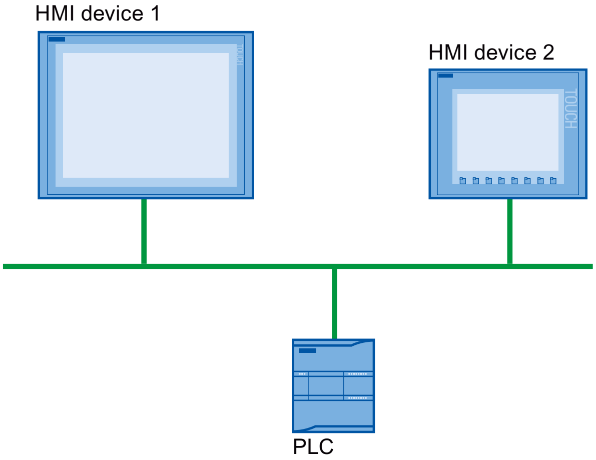
16 Communicating with controllers

16.1 Basics of communication (RT Uni)

16.1.1 Communication between devices (RT Uni)

Communication

The data exchange between two devices is known as communication. The devices can be interconnected directly or via a network. The networked devices in communication are referred to as communication partners.



Data transferred between the communication partners is used for various purposes:

- Display processes
- Operate processes
- Output alarms
- Archive process values and alarms
- Document process values and alarms
- Administer process parameters and machine parameters

Communication partners in the automation system

An automation system consists of the following communication partners:

- PLC
  The PLC controls a process by means of a user program.
- HMI device
  You use the HMI device to operate and monitor the process.
  Communication between the communication partners PLC and HMI device is described below.
  Additional information on other forms of communication is available in the online help of the TIA Portal in the section "Editing devices & networks".
  If the following requirements are met, the PLC and HMI device form an automation system:
  - The PLC and HMI device are interconnected.
  - The network between the PLC and HMI device is configured.

Network configuration

The basis for all types of communication is a network configuration.

- Every device in a network has a unique address.
- The devices carry out communication with consistent transmission characteristics.

Data exchange using tags

Process values such as temperatures and levels are transferred by tags in Runtime. Process values are stored in the memory of one of the connected automation systems.

To access the process data with the HMI device, link the external HMI tags to the PLC tags.

For additional information on configuring tags, refer to "Configuring tags (Page 137)".

Communication via a uniform and vendor-neutral interface

With OPC (Openess Productivity Collaboration), WinCC has a uniform and manufacturer-neutral software interface. This interface enables standardized data exchange between industrial, office, and manufacturing applications.

For more detailed information, refer to the documentation for OPC.

See also

Supported PLCs (Page 975)

Configuring communication (Page 975)

16.1.2 Supported PLCs (RT Uni)

Overview

Your HMI device can communicate with PLCs from the following lines of controllers:

SIMATIC line of controllers	Supported communication channels	Comment
SIMATIC S7-1500	PROFINET	Parallel communication with multiple PLCs is possible
SIMATIC S7-300/400	PROFINET	Parallel communication with multiple PLCs is possible

See also

Communication between devices (Page 973)

Configuring communication (Page 975)

16.1.3 Configuring communication (RT Uni)

Introduction

To set up an automation system, you work integrated in the TIA Portal. You configure the connections in the "Devices & Networks" editor. You use the graphic and tabular network view to set up the connections.

Requirement

- The network configuration is complete.
- HMI device and controller are available in the hardware catalog.

Note

Non-integrated configuration

If integrated configuration of the HMI connections is not possible, create non-integrated HMI connections of the HMI device in the "Connections" editor.

Procedure

To set up an automation system, always follow the steps below:

1. **Inserting devices**
   You drag a PLC and an HMI device from the hardware catalog to the network view of the "Devices & networks" editor.
2. **Configuring devices**
   Depending on the HMI device used, you add the required communication modules to your operator panel.
3. **Networking devices**
   In the networking step, you configure the physical connection of the devices.
   To connect the devices to the network, you connect the interfaces of the devices with communication capability using drag and drop.
4. **Connecting devices**
   To set up a logical communication connection between the communication partners, you create an HMI connection between the networked devices.
   The tabular network overview supplements the graphical network view.
   In addition, the created HMI connection is also visible in the "Connections" editor of the HMI device where it can be configured.

See also

Networking HMI device and PLCs (Page 982)

Creating an integrated HMI connection (Page 983)

Creating a non-integrated HMI connection (Page 985)

Communication between devices (Page 973)

Supported PLCs (Page 975)

16.2 Networks and connections (RT Uni)

16.2.1 SIMATIC communication networks (RT Uni)

16.2.1.1 Communication networks (RT Uni)

Overview

Communication networks are a central component of an automation solutions. Industrial networks fulfill special requirements:

- Coupling of automation systems as well as simple sensors, actuators, and PCs
- Correct transfer of information at the right time

- Robustness against electromagnetic interference, mechanical stresses and soiling
- Flexible adaptation to the production requirements

Industrial networks belong to the LANs (Local Area Networks) and allow communication within a limited area.

Industrial networks fulfill the following communication functions:

- Process and field communication of the automation systems including sensors and actuators
- Data communication between automation systems
- IT communication for integrating information technology

Overview of the networks

Your HMI device can communicate over the following networks:

- Industrial Ethernet
  The industrial network standard for all levels
- PROFINET
  The open Industrial Ethernet standard for automation

The figure below shows an example of a plant configuration:

HMI device in the plant network

You connect an HMI device in the network to SIMATIC S7 modules that have an integrated interface of the corresponding communication channel.

You can connect multiple HMI devices to one SIMATIC S7 PLC and multiple SIMATIC S7 PLCs to one HMI device. The maximum number of communication partners that you can connect to an HMI device is dependent on the HMI device used.

Additional information is available in the documentation for the respective HMI device.

See also

PROFINET (Page 977)

16.2.1.2 PROFINET (RT Uni)

PROFINET

PROFINET is an open standard for industrial automation defined by IEEE 61158 and based on Industrial Ethernet. PROFINET makes use of IT standards all the way to the field level and enables plant-wide engineering.

With PROFINET, you realize high-performance automation solutions with stringent real-time requirements.

Industrial Ethernet

Industrial Ethernet, which is based on IEEE 802.3, enables you to connect your automation system to your office networks. Industrial Ethernet provides IT services that you can use to access production data from the office environment.

See also

Communication networks (Page 976)

16.2.2 Connections (RT Uni)

16.2.2.1 HMI connection (RT Uni)

Definition

An HMI connection is a logical connection between an HMI device and a PLC. The HMI connection enables communication between the communication partners.

Unlike an S7 connection, the HMI connection is assigned to the HMI device.

Layout

The HMI connection defines the following within the plant network:

- Communication partners
  The HMI connection identifies the devices in the plant configuration.
- Communication channel over which these communication partners communicate.
  The HMI connection requires a configured network.
- Communication path
  The HMI connection defines the interface parameters and the network addresses of the communication partners.

HMI connection types

The options for addressing external tags depend on the type of HMI connection between WinCC and the PLC in question. The TIA Portal supports the following types of connection:

- Integrated HMI connection
  In the TIA Portal you configure integrated HMI connections between the devices in the "Devices & Networks" editor. An integrated HMI connection enables an optimized data exchange.
- Non-integrated HMI connection
  In case of a non-integrated connection, the control program can be created outside the WinCC project. You configure the PLC and the WinCC project independent of each other. For configuration in WinCC, you only need to know the addresses used in the PLC and their function.
  You use a non-integrated HMI connection, for example, in the following application cases:
  - You configure a WinCC project for external PLCs.
  - You do not have access to the device configuration of a SIMATIC controller, for example, because you are working without a STEP 7 license.

  You configure non-integrated HMI connections for the HMI device in the "Connections" editor of the WinCC project.

See also

Creating an integrated HMI connection (Page 983)

Creating a non-integrated HMI connection (Page 985)

Additional connection types (Page 979)

Setting up switch on/switch off of a connection in runtime (Page 986)

16.2.2.2 Additional connection types (RT Uni)

Overview

The following table provides an overview of the connection types that you can use in addition to the HMI connection for communication to specific device types and areas of application.

Additional information on connection types is available in the online help of the TIA Portal in the section "Editing devices & networks".

Connection type	Description	Application
S7 connections	Connection type can be used in all S7 devices	Data exchange between SIMATIC S7 stations
FDL connection	Fieldbus Data Link Security layer Based on PROFIBUS	Communication with a partner that supports sending and receiving according to the SDA function (Send Data with Acknowledge), e.g. SIMATIC S5 or PC.

Connection type	Description	Application
ISO transport connection	Suitable for large amounts of data Based on ISO transport	Communication with a partner that supports sending and receiving data in accordance with ISO transport, e.g. SIMATIC S5 or PC.
ISO-on-TCP connection	Transmission Control Protocol/Internet Protocol with the extension RFC 1006 Corresponds to the standard TCP/IP	Communication with a partner that supports sending and receiving data in accordance with ISO-on-TCP, e.g. PC or external system.
TCP connection	Transmission Control Protocol/Internet Protocol Corresponds to the standard TCP/IP	Communication with a partner that supports sending and receiving data in accordance with TCP/IP, e.g. PC or external system.
UDP connection	User Datagram Protocol Subnet: Industrial Ethernet	Unsecured transmission of related data fields between two nodes
E-mail connection	The mail server with which all e-mails sent by an IT-CP are served is defined by the e-mail connection.	For example, enables the sending of process data, for example, from data blocks via e-mail using a CP with IT functionality (IT-CP);
PtP connection	Pert-to-Peer Communication between two equal devices	Communication with external devices. e.g. a printer.

See also

HMI connection (Page 978)

16.3 Device configuration (RT Uni)

16.3.1 Layout of a PC-based HMI device (RT Uni)

Definition

A PC-based HMI device represents the plant process, shows the process values and enables access to the plant controller via operator inputs.

The HMI device needs a WinCC Runtime software for process visualization and operation. The device must have the corresponding interfaces and communication drivers to connect the HMI device to the plant network and the PLC. The hardware basis of a PC-based HMI device is a PC.

Layout in the "Devices & Networks" editor

To configure a PC-based HMI device in the "Devices & Networks" editor, define multiple components as HMI device:

- WinCC Runtime software
  The WinCC Runtime software visualizes your WinCC Runtime project and enables process operation. The WinCC Runtime software is available in the hardware catalog.
- PC station
  The selected PC provides the operating system and the firmware as well as other hardware equipment. Select a PC that meets your requirements.
- Communication processors
  If the selected PC station does not have the necessary interfaces, install the required communication processors in the PC station.

See also

Configuring HMI device (Page 981)

16.3.2 Configuring HMI device (RT Uni)

Introduction

A corresponding interface is required at both ends to connect the HMI device to the PLC.

Requirements

- The networks are configured.
- HMI device and PLC each support the communication channel of the respective network.
- The network view is open in the "Devices & Networks" editor.

Using WinCC Runtime to configure the configuration PC

1. Drag the WinCC Runtime from the hardware catalog to the work area.
   A SIMATIC PC station with WinCC Runtime is shown as graphic in the network view.
2. Select a communication processor for the required interface type in the hardware catalog.
3. Drag the communication processor to the WinCC Runtime.
4. Drag the matching PLC from the hardware catalog to the work area.

Result

You have configured an HMI device with the matching interfaces. You can network the devices.

See also

Layout of a PC-based HMI device (Page 980)

16.4 Configuring an HMI connection (RT Uni)

16.4.1 Integrated HMI connection (RT Uni)

16.4.1.1 Networking HMI device and PLCs (RT Uni)

Introduction

You can network an HMI device with multiple PLCs. The networking of devices is depicted by lines that are colored depending on the interface type.

The number of available interfaces and interface types depends on the device.

Requirement

- The "Devices & Networks" editor is open.
- The networks are configured.
- An HMI device is configured in the "Devices & Networks" editor.
- The PLC is configured in the "Devices & Networks" editor.

Procedure

To network an HMI device and a PLC, follow these steps:

1. Open the network view of the "Devices & Networks" editor.
2. Enable the "Networking" mode.
3. Use a drag-and-drop operation to interconnect the interfaces of the desired communication network of the devices.
   A connection is shown as graphic and table in the network view.
   The tabular network overview supplements the graphical network view with the following additional functions:
   - You obtain detailed information on the structure and parameter settings of the devices.
   - Using the "Subnet" column, you can connect communication-capable components to subnets that have been created.

See also

Creating an integrated HMI connection (Page 983)

SIMATIC communication networks (Page 976)

16.4.1.2 Creating an integrated HMI connection (RT Uni)

Introduction

An integrated HMI connection connects an HMI device and a SIMATIC S7 PLC in your project.

Connection resources

Each connection requires connection resources for the end point or transition point on the devices involved. The number of connection resources is device-specific.

If all connection resources of a communication partner are allocated, no new connection can be configured.

Requirement

- The networks are configured.
- An HMI device and a SIMATIC PLC are configured and networked.
- The network view is open in the "Devices & Networks" editor.

Create an integrated HMI connection

1. Enable the "Connections" mode.
2. Select the "HMI connection" connection type.
   The devices available for connection are highlighted in color.
3. Use a drag-and-drop operation to interconnect the interfaces of the desired communication channel of the HMI device and PLC with each other.
   The HMI connection is shown as graphic and table in the network view.
   The HMI connection is shown in the tabular area of the editor in the "Connections" tab.

   Note

   Change local connection names

   You can change the local name for the connection only in the tabular area of the editor.

4. Change the connection parameters in the tabular area according to the requirements of your project.

Open the graphic view of the connection partners

1. Select the HMI connection.
2. Click "Highlight HMI connection" and select the HMI connection.
   The connection path is shown in the Inspector window under "Properties > General > General".

Change the connection path

1. Open the graphic view display of the connection partners.
2. Select a different interface in the Inspector window under "Properties > General > General > Interface".
   The existing connection parameters are highlighted as invalid.
3. To validate the connection parameters, click "Find connection path".
   The connection parameters are reassigned and validated.

Create an integrated HMI connection in the "Tags" editor

1. Double-click "Tags" in the project tree below your HMI device.
   The "Tags" editor opens.
2. Create an HMI tag.
3. Connect the HMI tag to an existing PLC tag of the matching data type.
4. The integrated HMI connection to the PLC is established automatically.

See also

HMI connection (Page 978)

Configuring PROFINET interfaces of a non-integrated HMI connection (Page 989)

Networking HMI device and PLCs (Page 982)

S7-1500 | Integrated HMI connection (Page 996)

S7-300/400 | Integrated HMI connection (Page 998)

16.4.2 Non-integrated HMI connection (RT Uni)

16.4.2.1 Configuring non-integrated connections (RT Uni)

Introduction

A non-integrated HMI connection requires a communication driver and a good understanding of the address structure of the communication partner.

Addressing with non-integrated connections

In the case of a project with a non-integrated connection, you always configure a tag connection exclusively with absolute addressing.

Select the valid data type yourself. If the address of a PLC tag changes in a project with a non-integrated connection during the course of the project, you also have to make the change in WinCC. The tag connection is not checked for validity in Runtime. An error message is not displayed.

Communication drivers

A communication driver is a software component that establishes a connection between a PLC and an HMI device. The communication driver thus enables the assignment of process values to HMI tags.

Depending on the HMI device used and the connected communication partner, you select the interface used as well as the profile and transmission speed.

Basic procedure

The following steps are required to work in your project in a non-integrated connection:

1. Create an HMI connection
2. Select communication drivers and interfaces
3. Address the communication partners
4. Assign the communication network
5. Close the connection

See also

Creating a non-integrated HMI connection (Page 985)

16.4.2.2 Creating a non-integrated HMI connection (RT Uni)

Introduction

A non-integrated connection connects an HMI device to a PLC that is configured outside your project. You create the non-integrated HMI connection in the "Connections" editor of the HMI device.

Requirements

- A project is open.
- An HMI device has been created.

Procedure

To create a non-integrated connection, follow these steps:

1. Double-click "Connections" in the project tree below your HMI device.
   The "Connections" editor opens.
   Existing integrated connections are identified with .
   Existing non-integrated connections are identified with .
2. Create a new connection with "Add".
3. Select the communication driver. Use the communication driver of the required family of PLCs.
4. Select the required interface of the HMI device in the graphic area of the editor under "Parameters > [HMI device type] > Interface".
   The number of available interfaces on the HMI device depends on the communication driver.
5. Change the connection parameters according to the requirements of your project.

See also

HMI connection (Page 978)

Configuring non-integrated connections (Page 984)

S7-1500 | Non-integrated HMI connection (Page 997)

S7-300/400 | Non-integrated HMI connection (Page 999)

16.4.3 Setting up switch on/switch off of a connection in runtime (RT Uni)

Introduction

If a connection between an HMI device and a PLC is not always required, terminate the connection in Runtime and establish it again when necessary. This reduces the load on the communication channel.

Configure a Runtime script for enabling/disabling a connection in runtime.

Note

Alarm system and system diagnostics

After switching off the connection to a PLC, the alarms from this PLC are still being displayed. The system diagnostics for this PLC is also available.

Requirement

- An HMI connection is configured.
- A button is configured in the HMI device of the HMI connection.
- The "Screens" editor is open.

Procedure

To configure enabling/disabling of a connection in Runtime, follow these steps:

1. Select a button.
2. Select the event that is to trigger enabling/disabling in runtime under "Properties > Events" in the Inspector window.
3. Program a script to the event which enables or disables the connection via the "Set Connection mode" snippet.

Result

Pressing this button triggers enabling/disabling of the connection in Runtime.

See also

Runtime scripting (Page 355)

HMI connection (Page 978)

16.5 Configuring interfaces (RT Uni)

16.5.1 PLCs and Interfaces (RT Uni)

Introduction

A PLC has various types of interfaces.

Interfaces for integrated HMI connections

Your HMI device can communicate with PLCs from the following lines of controllers over the shown subnets and interfaces:

SIMATIC line of controllers	Subnet	Interface types of the PLC
SIMATIC S7-1500	PN/IE	PROFINET
SIMATIC S7-300/400	PN/IE	PROFINET

Interfaces for non-integrated HMI connections

Your HMI device can communicate over the following communication drivers and interfaces:

SIMATIC line of controllers	Communication drivers	Interface types on the HMI device
SIMATIC S7-1500	SIMATIC S7-1500	Industrial Ethernet
SIMATIC S7-300/400	SIMATIC S7 300/400	Industrial Ethernet

See also

Requirements for interface configuration (Page 988)

16.5.2 Requirements for interface configuration (RT Uni)

Before you configure the interfaces of an HMI connection, observe the following requirements:

- The configuration of the devices and networks is complete.
- You have access to the data of the network configuration.
- The HMI connection is created.

See also

PLCs and Interfaces (Page 987)

Configuring PROFINET interfaces of a non-integrated HMI connection (Page 989)

16.5.3 PROFINET (RT Uni)

16.5.3.1 PROFINET interfaces (RT Uni)

Introduction

You enter the IP address of the communication partner at the PROFINET interface. The data you need for this is available in the network configuration of your plant.

PROFINET parameter assignment for integrated HMI connections

You configure the following communication functions depending on the line of controllers:

SIMATIC line of controllers	Function
SIMATIC S7-1500	Parallel communication with multiple PLCs
	Set IP address in the project or on the device

SIMATIC line of controllers	Function
SIMATIC S7-300/400	Parallel communication with multiple PLCs
	Set IP address in the project or on the device

PROFINET parameter assignment for non-integrated HMI connections

The following parameters are required to create a non-integrated HMI connection:

- IP address of both communication partners
- Access point for local or network access of the HMI device

You configure the following communication functions depending on the line of controllers:

SIMATIC line of controllers	Function
SIMATIC S7-1500	Parallel communication with multiple PLCs
SIMATIC S7-300/400	Parallel communication with multiple PLCs
	Slot and rack
	Cyclic mode

See also

Configuring PROFINET interfaces of a non-integrated HMI connection (Page 989)

Interface and communication parameters (Page 996)

16.5.3.2 Configuring PROFINET interfaces of a non-integrated HMI connection (RT Uni)

Introduction

The network and device configuration for Industrial Ethernet and the address structure of the connection partners are described in the online help of the TIA Portal under "Configuring devices and networks".

> ⚠ **CAUTION**
>
> **Communication via Ethernet**
>
> In Ethernet-based communication, the end user is responsible for the security of his data network.
>
> Targeted attacks can overload the device and interfere with proper functioning.

When setting parameters for the PROFINET interface, you have the following options:

- Set the IP address in the TIA Portal project.
- Set the IP address on the device.

Requirements

- A network is configured.
- An HMI device and at least one PLC are configured.
- The communication partners are networked over a PROFINET subnet in the "Devices & Networks" editor.
- The integrated HMI connection between the communication partners is created.
- The network view is open in the "Devices & Networks" editor.

Configure the PROFINET interface of the PLC

1. Click the green PROFINET interface in the PLC.
2. Select the Ethernet addresses for the interface in the Inspector window under "Properties > General".
   The PROFINET interface parameters are displayed in the Inspector window.
3. Configure the IP address according to the data from the network configuration of the plant.
   If you set the IP address directly at the device, select the option "IP address is set directly at the device" instead.
4. If you want to use your own PROFINET device name for the PLC, enable the option "PROFINET device name is set directly at the device".

Configure the PROFINET interface of the HMI device

1. Click the green PROFINET interface of your HMI device.
2. Select the Ethernet addresses for the interface in the Inspector window under "Properties > General".
   The PROFINET interface parameters are displayed in the Inspector window.
3. Configure the IP address according to the data from the network configuration of the plant.
   When you transfer the WinCC project to the HMI device, this IP address is set up directly in the HMI device.
   If you want to fetch the IP address using a different path, enable the option "Use IP protocol".
4. To assign the PROFINET device name for the PLC yourself, deselect the option "Generate PROFINET device name automatically".

Result

You have configured the PROFINET interfaces of an HMI connection. The HMI device and the PLC can communicate in Runtime.

Connection resources for HMI connections

HMI connections that are created over the integrated PROFINET interface on the HMI device, one connection resource is allocated for the endpoint per HMI connection.

One connection resource is also required for the PLC.

See also

PROFINET interfaces (Page 988)

Requirements for interface configuration (Page 988)

Interface and communication parameters (Page 996)

16.6 Configuring communication (RT Uni)

16.6.1 Communicating with SIMATIC S7-1500 (RT Uni)

16.6.1.1 Communication with SIMATIC S7-1500 (RT Uni)

Overview

You configure the following communication channel for communication between the HMI device and the SIMATIC S7-1500 controller.

- PROFINET

See also

Valid data types for SIMATIC S7-1500 (Page 991)

Symbolic addressing (Page 992)

16.6.1.2 Valid data types for SIMATIC S7-1500 (RT Uni)

Valid data types for connections with SIMATIC S7-1500

The table below lists the data types that can be used when configuring tags.

Data type	Length
BOOL	1 bit
BYTE	1 byte
CHAR	1 byte
DATE	2 bytes
DATE_AND_TIME	8 bytes
DINT	4 bytes
DTL	12 bytes
DWORD	4 bytes
INT	2 bytes
LDT	8 bytes

Data type	Length
LINT	8 bytes
LREAL	8 bytes
LTIME	8 bytes
LTIME_OF_DAY	8 bytes
REAL	4 bytes
S5TIME	2 bytes
SINT	1 byte
STRING	(2+n) bytes, n = 0 to 254
TIME	4 bytes
TIME_OF_DAY	4 bytes
UDINT	4 bytes
UINT	2 bytes
ULINT	8 bytes
USINT	1 byte
WCHAR	2 bytes
WORD	2 bytes
WSTRING	(2+n) 2 bytes, n = 0 to 254

See also

Communication with SIMATIC S7-1500 (Page 991)

Symbolic addressing (Page 992)

16.6.1.3 Symbolic addressing (RT Uni)

Introduction

Data is exchanged between HMI device and PLC via tags.

Depending on the addressed data blocks, you address these tags absolute or symbolic in the PLC.

- Symbolic addressing
  For symbolic addressing, a validity check of the tag connection is performed in runtime. If an address is changed in the PLC, the change is registered and an error message is issued. For symbolic addressing, you select the PLC tag via its name and connect it to an external HMI tag. The valid data type for the external HMI tag is automatically selected by the system.
- Absolute addressing
  The linking of tags is not checked in runtime. You select the valid data type of the tags. If the tag address changes in the PLC, compile and load the HMI device again so that the change is registered in runtime.

Data blocks and symbolic access

Data blocks with optimized access support only symbolic addressing.

During the symbolic addressing of a data block, the address of an element in the data block is dynamically assigned and is automatically adopted in the HMI tag in the event of a change.

Neither the connected data block nor the WinCC project must be compiled.

For symbolic addressing of elements in a data block, you only need to recompile and reload the WinCC project in case of the following changes:

- Name or data type of the connected data block element or of the global PLC tag
- Name or data type of a higher-level structure node of the connected element in the data block element or global PLC tag
- Number of the connected data block

HMI connections and symbolic access

With symbolic addressing of tags, you create an integrated HMI connection:

- Integrated connection
  You address the tags symbolically as well as absolutely over an integrated connection.
- Non-integrated connection
  You address the tags only absolutely over a non-integrated connection.
  A non-integrated connection is available for all supported PLCs.

Disabling symbolic access

In WinCC, symbolic addressing is the default method.

To change the default setting, follow these steps:

1. Select "Options > Settings > Visualization > HMI tags" in the menu.
2. Activate the "Symbolic access" option.

See also

Communication with SIMATIC S7-1500 (Page 991)

Valid data types for SIMATIC S7-1500 (Page 991)

16.6.2 Communicating with SIMATIC S7-300 / S7-400 (RT Uni)

16.6.2.1 Communication with SIMATIC S7-300 / S7-400 (RT Uni)

Introduction

The S7-300 and S7-400 PLCs are referred to jointly as SIMATIC S7-300/400.

You configure the following communication channel for communication between an HMI device and the SIMATIC S7-300/400 controller.

- PROFINET

See also

Valid data types for SIMATIC S7-300 / S7-400 (Page 994)

Cyclic operation (Page 995)

16.6.2.2 Valid data types for SIMATIC S7-300 / S7-400 (RT Uni)

Permitted data types for connections with SIMATIC S7-300/400

The table lists the data types that can be used when configuring tags.

Data type	Length
BOOL	1-bit
BYTE	1 byte
CHAR	1 byte
DATE	2 bytes
DATE_AND_TIME	8 bytes
DINT	4 bytes
DWORD	4 bytes
INT	2 bytes
REAL	4 bytes
S5TIME	2 bytes
STRING	(2+n) bytes, n = 0 to 254
TIME	4 bytes
TIME_OF_DAY, TOD	4 bytes
WORD	2 bytes

See also

Communication with SIMATIC S7-300 / S7-400 (Page 993)

Cyclic operation (Page 995)

16.6.2.3 Cyclic operation (RT Uni)

Basics of cyclic operation (RT Uni)

Operating principle of cyclic operation

When the "Cyclic operation" option is enabled, the HMI device sends a message frame to the CPU at the beginning of communication indicating that certain tags are required on a recurring basis.

The CPU then always transmits the data at the same cyclic interval. This saves the HMI device from having to output new requests for the data.

When cyclic operation is disabled, the HMI device sends a request whenever information is required.

Advantages und properties of cyclic operation

The list below shows the advantages und properties of "Cyclic operation" option:

- Cyclic operation reduces data transmission load at the HMI device. The PLC resources are used to relieve load on the HMI device.
- The PLC only supports a certain number of cyclic services. The HMI device handles the operation if the PLC cannot provide any further resources for cyclic services.
- The HMI device generates the cycle if the PLC does not support cyclic operation.
- Screen tags are not integrated into cyclic operation.
- Cyclic operation is only set up at the restart of Runtime.
- The HMI device transfers several jobs to the PLC if cyclic operation is enabled, depending on the PLC.
- The HMI device only transfers one job to the PLC if cyclic operation is disabled.

See also

Configuring cyclic operation (Page 995)

Configuring cyclic operation (RT Uni)

Introduction

You configure cyclic operation for an HMI connection at an HMI connection in the "Connections" editor of the HMI device.

Requirement

- The devices and networks are configured.
- An HMI connection is created in the "Connections" editor.

Procedure

To enable an HMI connection for cyclic operation, follow these steps:

1. Double-click "Connections" in the project tree below your HMI device.
   The "Connections" editor opens.
2. Select the desired HMI connection.
   The parameters of the connection are displayed in the graphic overview.
3. Activate "PLC > Cyclic operation".

Result

The PLC then always transmits the required data at the same cyclic interval.

See also

Basics of cyclic operation (Page 995)

16.7 Interface and communication parameters (RT Uni)

16.7.1 S7-1500 (RT Uni)

16.7.1.1 S7-1500 | Integrated HMI connection (RT Uni)

PROFINET interface parameters (RT Uni)

The following table shows the interface parameters of an integrated HMI connection:

Table 16-1 PROFINET parameters of the HMI device

Parameters	Description
Subnet	Specifies the subnet of the HMI connection via which the HMI device is connected to the network.
MAC address	Specifies the MAC address for the connection type "ISO connection". This option is only available when the "Use ISO protocol" option is selected.
IP address	Specifies the IP address of the communication partner. This property is only available when the "Set IP address in the project" option is selected.

Parameters	Description
Subnet mask	Specifies the subnet mask. This property is only available when the "Set IP address in the project" option is selected. The subnet mask determines which part of the IP address addresses the network and which part of the IP address addresses the device.
Router address	Specifies the router address. This property is only available when the "Use router" option is selected.

Table 16-2 PROFINET parameters of the PLC

Parameters	Description
Subnet	Specifies the subnet of the HMI connection via which the HMI device is connected to the network.
IP address	Specifies the IP address of the communication partner. This property is only available when the "Set IP address in the project" option is selected.
Subnet mask	Specifies the subnet mask. This property is only available when the "Set IP address in the project" option is selected. The subnet mask determines which part of the IP address addresses the network and which part of the IP address addresses the device.
Router address	Specifies the router address. This property is only available when the "Use router" option is selected.
PROFINET device name	Shows the PROFINET device name or specifies it. This property is only available when the "Generate PROFINET device name automatically" option is deactivated.
Converted name	Shows the name that is automatically generated from the PROFINET device name and satisfies the DNS conventions.
Device number	Shows the device number by which an IO device can be identified.

16.7.1.2 S7-1500 | Non-integrated HMI connection (RT Uni)

PROFINET interface parameters (RT Uni)

The following table shows the interface parameters of a non-integrated HMI connection:

Table 16-3 PROFINET parameters of the HMI device

Parameters	Description
Interface	Specifies the communication channel.
Address	Specifies the IP address of the device.
Access point	Specifies the device name through which the communication partner can be reached.

Table 16-4 PROFINET parameters of the PLC

Parameters	Description
Address	Specifies the IP address of the device.

16.7.2 S7-300/400 (RT Uni)

16.7.2.1 S7-300/400 | Integrated HMI connection (RT Uni)

PROFINET interface parameters (RT Uni)

The following table shows the interface parameters of an integrated HMI connection:

Table 16-5 PROFINET parameters of the HMI device

Parameters	Description
Subnet	Specifies the subnet of the HMI connection via which the HMI device is connected to the network.
MAC address	Specifies the MAC address for the connection type "ISO connection". This option is only available when the "Use ISO protocol" option is selected.
IP address	Specifies the IP address of the communication partner. This property is only available when the "Set IP address in the project" option is selected.
Subnet mask	Specifies the subnet mask. This property is only available when the "Set IP address in the project" option is selected. The subnet mask determines which part of the IP address addresses the network and which part of the IP address addresses the device.
Router address	Specifies the router address. This property is only available when the "Use router" option is selected.

Table 16-6 PROFINET parameters of the PLC

Parameters	Description
Subnet	Specifies the subnet of the HMI connection via which the HMI device is connected to the network.
IP address	Specifies the IP address of the communication partner. This property is only available when the "Set IP address in the project" option is selected.
Subnet mask	Specifies the subnet mask. This property is only available when the "Set IP address in the project" option is selected. The subnet mask determines which part of the IP address addresses the network and which part of the IP address addresses the device.
Router address	Specifies the router address. This property is only available when the "Use router" option is selected.

Parameters	Description
PROFINET device name	Shows the PROFINET device name or specifies it. This property is only available when the "Generate PROFINET device name automatically" option is deactivated.
Converted name	Shows the name that is automatically generated from the PROFINET device name and satisfies the DNS conventions.
Device number	Shows the device number by which an IO device can be identified.

16.7.2.2 S7-300/400 | Non-integrated HMI connection (RT Uni)

PROFINET interface parameters (RT Uni)

The following table shows the interface parameters of a non-integrated HMI connection:

Table 16-7 PROFINET parameters of the HMI device

Parameters	Description
Interface	Specifies the communication channel.
Address	Specifies the IP address of the device.
Access point	Specifies the device name through which the communication partner can be reached.

Table 16-8 PROFINET parameters of the PLC

Parameters	Description
Address	Specifies the IP address of the device.
Expansion slot	Specifies the number of the expansion slot of the PLC to be addressed.
Rack	Specifies the rack number of the PLC to be addressed.
Cyclic operation	Enables/disables cyclic operation.

16.8 Troubleshooting of connection errors (commissioning) (RT Uni)

16.8.1 Troubleshooting for SIMATIC S7-300/400 (RT Uni)

16.8.1.1 Procedure for the localization of errors (RT Uni)

S7 channel rectification in case of connection problems (RT Uni)

Introduction

If you encounter problems during the commissioning of WinCC S7, the following description provides initial information. The description is for error localization and error correction and does not replace the WinCC description.

Follow the following instructions to locate the error.

- Create a new project
- Select the correct communication subsystem
- Select the correct channel unit
- Configure the correct AS network address
- Check the system bus
- Check AS with STEP 7
- S7 log function
- Special settings of the S7 channel
- Error codes for connection fault
- API error texts

Error code

The most important error codes are listed in the section "Error codes for connection fault and errors in functions". If an error occurs with an error code that is not in the table, please call the WinCC hotline.

For error codes for a connection fault (SIMATIC S7-300/400) see under numbers 0112 to D406.

You will find errors which occur during editing of channel functions with the display "Error xx occurred in "function name" function!" under the numbers 1 to 100.

See also

Creating a new project (Page 1001)

Selection of the correct communication subsystem (Page 1001)

Selection of the correct channel unit (Page 1002)

Configure the correct AS network address (Page 1003)

Checking the bus system (Page 1003)

Checking the AS with STEP 7 (Page 1004)

S7 log function (Page 1004)

Creating a new project (RT Uni)

If you communicate in your current project with several automation systems, create a new project with a single connection to a automation system. The errors can be easier localized here.

See also

S7 channel rectification in case of connection problems (Page 1000)

Selection of the correct communication subsystem (Page 1001)

Selection of the correct channel unit (Page 1002)

Configure the correct AS network address (Page 1003)

Checking the bus system (Page 1003)

Checking the AS with STEP 7 (Page 1004)

S7 log function (Page 1004)

Selection of the correct communication subsystem (RT Uni)

The S7 channel communicates via the communication subsystem SAPI-S7 or S7DOS.

SAPI-S7

SAPI-S7 is a single product and is used for the communication to the S7 automation systems. SAPI-S7 products are adjusted to the corresponding CP. The correct SAPI-S7 product has to be installed to the available CP during communication via SAPI-S7. The S7 channel uses the tag services for reading and writing of tags from SAPI-S7.

For more complex functions, e.g. PMC error message processing or the BSEND/BRCV function, activate the communication subsystem S7DOS.

S7DOS

S7DOS is a part of STEP 7 and is also used for the communication to S7. In addition to the read and write jobs, the more complex PMC services are also activated during communication via S7DOS. For clients who have no STEP 7 installed on the WinCC target machine, S7DOS is also on the WinCC-DVD delivered. For installation of S7DOS see WinCC setup description.

For the single communication subsystems, limitations for supported communication cards are possible. For information, refer to your current product information.

The S7 channel uses the following method to recognize which communication subsystem will be used by the S7 channel for communication: It first attempts to load the S7DOS communication subsystem. If S7DOS cannot be loaded, it attempts to load SAPI-S7. The error message "Error during loading of S7 communication driver" will be displayed if SAPI-S7 cannot be loaded either.

See also

S7 channel rectification in case of connection problems (Page 1000)

Creating a new project (Page 1001)

Selection of the correct channel unit (Page 1002)

Configure the correct AS network address (Page 1003)

Checking the bus system (Page 1003)

Checking the AS with STEP 7 (Page 1004)

S7 log function (Page 1004)

Selection of the correct channel unit (RT Uni)

The selection of the channel unit depends on the communication card in your PC. A specific type of communication card is assigned to each channel unit.

Communication with the S7 takes place via so-called 'Logical devices'. The name of a logical device is assigned during installation of the communication card. In the case of some installations you can assign the name also via "Busprofile".

Depending on the installed communication card, the following defaults are standard:

- "SIMATIC NET PROFIBUS" communication cards: e.g., CP5611, CP5613, CP5623
- "SIMATIC Industrial Ethernet" communication cards: CP_H1_1:
- Slot PLCs: SLOT_PLC
- TCP/IP communication cards: CP-TCPIP

The S7 communication driver differentiates accordingly the following units:

- PROFIBUS communication via the SIMATIC NET PROFIBUS cards (CP5623), which means logical device: CP_L2_1:
- Industrial Ethernet for communication via SIMATIC Industrial Ethernet cards (e.g., CP1623 or CP1628); which means logical device: CP_H1_1:
- Slot PLC for communication via a Slot PLC;i.e. logical device: SLOT_PLC
- TCP/IP for communication via the TCP/IP protocol; i.e. logical device: CP-TCPIP

During installation of the communication drivers the physical device is assigned via which the communication takes place.

You can change the default "logical device names" of the channel unit via the unit-specific setting of the system parameter.

See also

S7 channel rectification in case of connection problems (Page 1000)

Creating a new project (Page 1001)

Selection of the correct communication subsystem (Page 1001)

Configure the correct AS network address (Page 1003)

Checking the bus system (Page 1003)

Checking the AS with STEP 7 (Page 1004)

S7 log function (Page 1004)

Configure the correct AS network address (RT Uni)

For example, the TSAPs have a fixed setting for the S7 network address. This does not usually need to be configured. The set address as well as the rack and the slot number still have to be configured.

Check the configured network address with the AS configured network address.

The rack number and the slot number have only to be specified, if the communication takes place on AS side via a communication processor and not via the MPI interface installed on the CPU. In this case specify the rack and the slot number of the CPU that is to be addressed. If MPI interface installed in the CPU is used, enter the value 0 for rack and slot number.

See also

S7 channel rectification in case of connection problems (Page 1000)

Creating a new project (Page 1001)

Selection of the correct communication subsystem (Page 1001)

Selection of the correct channel unit (Page 1002)

Checking the bus system (Page 1003)

Checking the AS with STEP 7 (Page 1004)

S7 log function (Page 1004)

Checking the bus system (RT Uni)

Check the correct setup of the bus system.

SIMATIC NET PROFIBUS

- Check the network for communication stations with the same station address.
- Check if the highest station address (HSA) is correctly set for all communication stations.
- Check if an enabled device, e.g. an AS, is connected at the start and the end of the PROFIBUS.
- Check if the terminating resistors are correctly enabled or disabled. The terminating resistors may only be enabled at the first and the last bus terminal.
- If you have an MPI cable, connect the AS directly to your MPI card or your CP5623.

SIMATIC Industrial Ethernet

- Check the network for communication stations with the same Ethernet addresses.

See also

S7 channel rectification in case of connection problems (Page 1000)

Creating a new project (Page 1001)

Selection of the correct communication subsystem (Page 1001)

Selection of the correct channel unit (Page 1002)

Configure the correct AS network address (Page 1003)

Checking the AS with STEP 7 (Page 1004)

S7 log function (Page 1004)

Checking the AS with STEP 7 (RT Uni)

If you have installed STEP 7, check if you can access the automation system with STEP 7. If problems occur, follow the STEP 7 instructions for error correction.

See also

S7 channel rectification in case of connection problems (Page 1000)

Creating a new project (Page 1001)

Selection of the correct communication subsystem (Page 1001)

Selection of the correct channel unit (Page 1002)

Configure the correct AS network address (Page 1003)

Checking the bus system (Page 1003)

S7 log function (Page 1004)

S7 log function (RT Uni)

The S7 channel has a logbook function. You can use this file to write the most important status changes and errors to an ASCII file on an external memory. These messages can be used to analyze a communication problem.

The individual messages are described below:

Start and end messages

- S7 channel DLL started
- S7 channel DLL terminated

Unit activated or deactivated

- S7 channel unit "unitname" activated
- S7 channel unit "unitname" deactivated

Version information

- S7DOS version: Version string
- S7CHN version: Version string

Communication error

- Cannot connect to "connectionname": Errorcode 0xhhhh ffff

The message is displayed if communication to the corresponding automation device cannot be established immediately after WinCC is activated. If the connection was established correctly at least once, the following message is displayed if errors occur:

- Connectionerror nnn " connectionname": Errorcode 0xhhhh ffff

	Description
nnn	Number of connection terminations for this connection
Connectionname	Connection name
hhhh	1. Error display in hexa S7DOS / SAPI-S7
ffff	2. Error display in hexa S7DOS / SAPI-S7

Channel API error

- Channel API error: Errorstring

The error string 'errorstring' is passed by the channel to the WinCC Explorer. Depending on the significance of the error, the error string may or may not be displayed in a notice box. For a description of the error strings, please see the API Error Text.

- Max. count of API errors reached - API logbook deactivated

Errors on the API interface can cyclically occur depending on the error and function. To avoid filling the logbook file with these error messages, a maximum of 32 messages are output for an API error.

General channel error messages

- Cannot write storage data
- Cannot read storage data / use default data
- Storage data illegal or destroyed / use default data
- No storage data / use default data

Initialization messages

- Devicename in unit "unitname" changed from "old devicename" to "new devicename"

Message when maximum file length is exceeded

- Max. logbooksize reached - Logbook deactivated

In order to avoid filling of the data medium only with the logbook file, the logbook output has a length monitoring. As soon as the specified length is reached, the logbook is deactivated. The message is only output, when message output causes the max. file length to be exceeded. No message is output, if the file length is changed with an editor or the maximum file length is reduced in the INI file.

The output is not time critical, because the logbook functions are called only during activate/ deactivate and in the case of problems of the communication process.

Example of a logbook protocol:	
17.09.1996/15:16:18.330	S7 channel DLL started
17.09.1996/15:18:05.380	S7DOS version: @(#)TIS-Block Library DLL Version X3.060-DEB-BASIS
17.09.1996/15:18:05.490	S7CHN version: V2.0 / Sep 16 1996 / 18:42:48
17.09.1996/15:18:05.600	S7 channel unit "S7-MPI" activated
17.09.1996/15:36:48.490	Cannot connect to "MPI_CPU_3": Errorcode 0xFFDF 4107
17.09.1996/15:43:29.060	Connectionerror 1 "MPI_CPU_4": Errorcode 0xFFDF 410E
17.09.1996/16:43:44.830	Connectionerror 2 "MPI_CPU_4": Errorcode 0xFFDF 410E
17.09.1996/17:45:29.960	Connectionerror 3 "MPI_CPU_4": Errorcode 0xFFDF 410E
17.09.1996/18:54:02.630	Connectionerror 4 "MPI_CPU_4": Errorcode 0xFFDF 410E
17.09.1996/22:56:47.680	S7 channel unit "S7-MPI" deactivated
17.09.1996/22:56:56.740	S7 channel DLL terminated

The logbook texts are not language-dependent with the exception of "Channel API Errorstring" and are always displayed in English.

The S7 logbook is enabled by default. The logbook is stored in WinCC diagnose folder.

To settings and assigning parameters of the logbook function, see "Setup of an S7 logbook.

See also

S7 channel rectification in case of connection problems (Page 1000)

Creating a new project (Page 1001)

Selection of the correct communication subsystem (Page 1001)

Selection of the correct channel unit (Page 1002)

Configure the correct AS network address (Page 1003)

Checking the bus system (Page 1003)

Checking the AS with STEP 7 (Page 1004)

Special settings of the S7 channel (RT Uni)

Special settings of the S7 channel (RT Uni)

Edit with an ASCII-Editor, e. g. NOTEPAD.EXE, a file with name S7CHN.INI. Via the entries in the INI file you can modify the properties of the S7 channel. The S7CHN.INI file is expected in the directory which also contains the S7 communication driver "SIMATIC S7-300/400.CHN". If no file S7CHN.INI is found, the properties are not modified. The file is set to Windows INI file conventions.

[section]

key=string

In the following the relevant parameters for modification of S7 channel are described. You just have to apply the lines in the INI file which create the setting you want. Comply with the specified examples:

- Change the default of the logical device name of the channel unit
- Change CP type of a channel unit
- Setup S7 logbook

Additional specific settings:

- Modify channel characteristics via the S7chn.ini file

See also

Change the default of the logical device name of a channel unit (Page 1007)

Changing the CP type of a channel unit (Page 1008)

Setup of an S7 logbook (Page 1009)

s7chn.ini (Page 1010)

Change the default of the logical device name of a channel unit (RT Uni)

```
[Unit:unitname]
```

```
CpName = name
```

Name

Logical device name of the communication card (for example: CP_L2_1:).

Example: You want to set the channel unit "PROFIBUS" to a new logical device name "mein_cp".

```
 [Unit:PROFIBUS]
```

```
CpName = mein_cp
```

Note

The modification via INI file is only for setup of the default. The setting of the logical device name takes place via the unit specific system parameter setting.

See also

Special settings of the S7 channel (Page 1007)

Changing the CP type of a channel unit (Page 1008)

Setup of an S7 logbook (Page 1009)

s7chn.ini (Page 1010)

Changing the CP type of a channel unit (RT Uni)

```
[Unit:unitname]

CpName = name

CpType = type
```

Name

Logical device name of the communication card (for example: CP_L2_1:).

Type

Type of the communication card (for example: L2). The following settings are available:

- MPI
- L2
- H1

Example: You want to set the channel unit "PROFIBUS" to a new logical device name "cp_neu". You want to activate the connection configuration mask to configure the AS network address to enter a special AS address according to SCI convention.

```
CpName = cp_neu

CpType = NNN
```

Note

Only change the CP type if you want to activate the connection configuration mask to configure the AS network address to enter a special AS address according to SCI convention.

See also

Special settings of the S7 channel (Page 1007)

Change the default of the logical device name of a channel unit (Page 1007)

Setup of an S7 logbook (Page 1009)

s7chn.ini (Page 1010)

Setup of an S7 logbook (RT Uni)

```
[Logbook]

FileName = C:\TMP\S7CHN.LOG

MaxKbFileSize = 32

MaxCycleBuffer= 2

Selection = FF.FF.FF.FF
```

FileName

Name of the diagnostics file incl. path specification. If no path is specified, the path of the called module is used. The name of the diagnostic file should be unique in the first 6 characters. In the circular buffer mode the file name is limited to maximum 6 characters and completed with the current cycle number 01 to 99. (Example: FileName = C:\TMP\S7CHN.LOG). The specified folder has to be available when the communication driver is started. The folder is not automatically created by S7 channel DLL.

MaxKbFileSize

Maximal length of the diagnostic file in k byte. If the maximal length is reached, the saving process is completed or switched to another file, depending on the "MaxCycleBuffer" setting. (Example: MaxKbFileSize = 32)

MaxCycleBuffer

Maximum number of circular buffer files. If the parameter is not specified or "MaxCycleBuffer=0" is specified, only one file is created. For input of a number 2..99 a new file is created if the maximum file length is reached. In the circular buffer mode the file name is limited to maximum 6 characters and completed with the current cycle number 01..99. If the maximum buffer number is reached, the saving process of the files continues with the file 01. (Example: MaxCycleBuffer = 2) ==> writing in files: S7CHN01.LOG -> S7CHN02.LOG -> S7CHN01.LOG ...

Selection

Number of the message to be saved. The selection is specified as 32 bit hexastring. Set for each desired kind of message the corresponding bit. (Example: Selection=FFFFFFFF). In order to simplify the input, an ' ' or '.' character can be inserted byte by byte. (Example: Selection = FF.FF.FF.FF)

The individual selection ID has the following values:

Selection ID	**Value**
00.00.00.01	Start and end error message
00.00.00.02	Error message if unit activated/deactivated
00.00.00.04	Version information
00.00.00.08	Communication errors
00.00.00.10	Channel API error

Selection ID	Value
00.00.00.20	General channel error messages
00.00.00.40	Initialization messages

The selection = FF.FF.FF.FF causes all logbook error messages to be output. The output is not time critical, because the logbook functions are called only during activate/deactivate and in the case of problems of the communication process.

See also

Special settings of the S7 channel (Page 1007)

Change the default of the logical device name of a channel unit (Page 1007)

Changing the CP type of a channel unit (Page 1008)

s7chn.ini (Page 1010)

s7chn.ini (RT Uni)

The characteristics of the S7 channel can be modified via the S7CHN.INI file. The file is only for test or diagnostics and is not required for the normal operation.

Since the runtime characteristics of the S7 channel can be significantly changed using the INI file described below, the options for modifying this file should only be used for internal testing or error analysis!

The interpretation and modification of the parameters is only intended for "insiders". Therefore, we have not gone into more detail on the meaning and application of the parameters and constants.

```
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
; !!!!!    A T T E N T I O N    !!!!!      ===========================
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
; The S7CHN.INI is not required for the runtime!  Only for
; for testing and diagnostics. *) default without S7CHN.INI

;[Debug]
;DebugMode     = 1      ; 0 = off *), 1 = DebugMessage, 2 = S7 Console
;DebugLevel    = 101    ; 100 = Info, 101 = Error *), 102 = CHN API, 103 = Trace, 104 = List
;DebugTime     = 1      ; 0 = off, 1 = on *)

;[SAPI]
;TraceTarget   = 1      ; 0 = BUFFER *), 1 = NEW_FILE
;TraceSelect   = 65535  ; 1 = ADMIN_SERVICES *), 65535 = SELECT_ALL
;TraceDepth    = 104    ; 0 = OFF *), 101 = INTERFACE, 103 = LIB0, 104 = LIB0_PDU

;[Timer]
;TimerMode     = 1      ; 0 = external, 1 = internal *)
```

```
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;TimerCycle    = 1000   ; cycle time in milliseconds
;CycleConnInit = 4000   ; Cycle time for autom. connection establishment
;FunTimeout    = 10000  ; Timeout time for function calls
;CloseTimeout  = 10000  ; Timeout time Close for unit deactivation
;MaxAsCycle    = 5000   ; Max. cycle for cyclic read jobs by AS

;[Channel]
;UnitStorage   = 1      ; Storage Read access: 0 = off, 1 = on
;LocalSlotId   = 0      ; Setting of the local slot number in the TSAP
;ThreadModel   = 1      ; Thread Model (0 ohne, 1 mit, 2 CHNCAP_REENTRANT)
;S7String      = 1      ; String processing (0 with, 1 without control word)

;[Driver]
;ComSystem     = 0      ; 0 = auto *), 1 = SAPI, 2 = S7DOS
;CnfAmqCalling = 0      ; max. rec.-jobs without acknowledgement (0 default)
;CnfAmqCalling = 0      ; max. send.-jobs without acknowledgement (0 default)

;[Unit:Industrial Ethernet]
;CpName        = CP_H1_1:
;CpType        = NNN

;[Diagnosis]
;ChkStopState  = 0      ; Stop check: 0 = off, 1 = on *)
;ChkLifeTime   = 60000  ; cycle time for lifebeat monitoring in milliseconds 0 = off
;LifeTimeAck   = 15000  ; lifebeat acknowledgement timeout in milliseconds

;[Logbook]
;FileName      = C:\TMP\S7CHN.LOG
;MaxKbFileSize = 20
;MaxCycleBuffer= 2
;Selection     = FF.FF.FF.FF

;[ChnTrace]
;FileName      = C:\TMP\S7CHN.TRC
;MaxKbFileSize = 50     ; max. length of the diagnostic file in k byte
;MaxCycleBuffer= 10     ; max. number circular buffer files
;DebugMode     = 0      ; 0 = Output in file, 1 = Output via 'OutputDebugString()’
;TimeStampMode = 2      ; 0 = no time stamp, 1 oder 2 = with time stamp
;Selection     = 00.00.30.27
               ; 00 00 00 01        // General alarms
               ; 00 00 00 02        // Connection status
               ; 00 00 00 04        // S7 Function calls # Tags services
               ; 00 00 00 08        // S7 Function calls # Tags services
```

	; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
	; 00 00 00 10 // S7 Function call data
	; 00 00 00 20 // Message event # Tags services
	; 00 00 00 40 // Message event Tags services
	; 00 00 00 80 // Message event data
	; 00 00 01 00 // Message on start processing
	; 00 00 10 00 // S7 raw data function
	; 00 00 20 00 // S7 raw data
	; 00 01 00 00 // S7 Channel API # Tags services
	; 00 02 00 00 // S7 Channel API Tags services
	;MaxDump = 512 ; Max. number bytes on data trace (Hex-Dump) of the channel diagnosis

See also

Special settings of the S7 channel (Page 1007)

Change the default of the logical device name of a channel unit (Page 1007)

Changing the CP type of a channel unit (Page 1008)

Setup of an S7 logbook (Page 1009)

16.8.1.2 Error codes (RT Uni)

Error codes for connection faults (SIMATIC S7-300/400) (RT Uni)

The most important error codes are listed in this section. If an error occurs with an error code that is not in the table, please call the WinCC hotline.

- Error 0112 (Page 1014)
- Error 4022 (Page 1014)
- Error 4102 (Page 1014)
- Error 4107 (Page 1014)
- Error 410E (Page 1015)
- Error 4110 (Page 1015)
- Error 4116 (Page 1015)
- Error 411A (Page 1015)
- Error 411C (Page 1016)
- Error 4230 (Page 1013)

- Error 4231 (Page 1016)
- Error 4232 (Page 1018)
- Error 42B0 (Page 1016)
- Error 42B1 (Page 1016)
- Error 42B2 (Page 1016)
- Error 42B3 (Page 1017)
- Error 42B5 (Page 1017)
- Error 42B7 (Page 1017)
- Error 42B8 (Page 1017)
- Error 42C0 (Page 1017)
- Error 42C2 (Page 1020)
- Error 42D2 (Page 1020)
- Error 7000 (Page 1020)
- Error 7001 (Page 1020)
- Error 7002 (Page 1020)
- Error 7003 (Page 1021)
- Error 7004 (Page 1021)
- Error 7005 (Page 1021)
- Error 7006 (Page 1021)
- Error 7007 (Page 1022)
- Error 7008 (Page 1022)
- Error 7101 (Page 1022)
- Error 7102 (Page 1022)
- Error 7900 (Page 1022)
- Error 8204 (Page 1023)
- Error 8305 (Page 1024)
- Error 8404 (Page 1024)
- Error 8405 (Page 1024)
- Error 8500 (Page 1024)
- Error D405 (Page 1024)
- Error D406 (Page 1025)

Error 4230 - L4_SCI_E_UNPLUGGED (RT Uni)

No further active partner available.

- PC not switched to the bus or power plug not correctly plugged.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 0112 - INVALID_PARAM (RT Uni)

An incorrect parameter was transferred to the communication subsystem S7DOS.

- Message processing or CPU Stop Monitoring enabled and an outdated S7DOS version used.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 4022 - S7DOS_SRV_TIMEOUT (RT Uni)

No feedback from AS:

- AS did not respond to job within the timeout period.
- Station address available several times on the bus.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 4102 - L4_INVALID_REQ (RT Uni)

Request block not permitted.

- An invalid request block was returned by the communication driver.
- A not yet processed request block was returned after a CLOSE_REQ.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 4107 - L4_OK_CLOSED_RESP (RT Uni)

Connection was terminated.

- Rack/Slot not right configured.
- Number of maximum permitted connections in AS exceeded.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 410E - L4_REM_ABORT (RT Uni)

Connection was aborted.

- AS or CP disabled or reset.
- AS not connected to bus or error in bus system.
- Number of maximum permitted connections in AS exceeded.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 4110 - L4_LOC_TIMEOUT (RT Uni)

No connection established. AS denies connection setup.

- Wrong internet address configured.
- AS disabled.
- AS not connected to bus or error in bus system.
- Number of maximum permitted connections in AS exceeded.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 4116 - L4_CONN_REJECT (RT Uni)

No connection established. AS denies connection setup.

- Rack/Slot not right configured. For external CP module, the slot of the CPU module has to be indicated.
- Number of maximum permitted connections in AS exceeded.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 411A - L4_ILLEGAL_ADDRESS (RT Uni)

Invalid address.

- An invalid network address was specified during connection configuration.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 411C - L4_NETWORK_ERROR (RT Uni)

Network error.

- A fatal error was encountered in the network.
- The network address or the segment ID was configured incorrectly.
- Communication module defective.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 4231 - L4_SCI Error 4231 - L4_SCI (RT Uni)

BUS disrupted.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42B0 - L4_DLL_E_NO_HW (RT Uni)

No communication hardware found.

- Communication module defective.
- Communication module not installed correctly.
- Wrong port address defined.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42B1 - L4_DLL_E_HW_DEFECT (RT Uni)

Communication module defective.

- Communication module not installed correctly.
- Wrong port address defined.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42B2 - L4_DLL_E_CNF (RT Uni)

- Driver configured incorrectly or invalid parameter in the registry.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42B3 - L4_DLL_E_BAUDRATE (RT Uni)

Incorrect baudrate or incorrect interrupt vector defined.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42B5 - L4_DLL_E_TS (RT Uni)

The defined local participant number (TS_ADR) is already assigned.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42B7 - L4_DLL_E_INT_NOT_PROV (RT Uni)

The defined interrupt vector (IRQ) is not available on the communication module.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42B8 - L4_DLL_E_INT_BUSY (RT Uni)

The defined interrupt vector (IRQ) is already allocated on the communication module.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42C0 - L4_DLL_E_NO_FILE (RT Uni)

The selected communication driver cannot be installed. File not found.

- Communication driver not installed correctly.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 4232 - L4_DLL_E_HSA (RT Uni)

Incorrect maximum number of set stations.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

1. S7DOS Error display (RT Uni)

This value indicates the error detecting module. The value is not relevant for the initial error analysis.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

1. SAPI-S7 Error display (RT Uni)

This value indicates the "S7_MINI_DB_RESPONSE_CODE". The value is not relevant for the initial error analysis.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

2. SAPI-S7 Error display (RT Uni)

This value indicates the "s7_last_detailed_err_no". The table "s7_last_detailed_err_nos7_last_detailed_err_no" includes a list of SAPI-S7 error codes without additional details.

In case of problems with SAPI-S7, read the SAPI-S7 documentation or contact the SINEC or WinCC hotline.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

s7_last_detailed_err_no (RT Uni)

Table 16-9

Error	Decimal	Hexa
S7_ERR_NO_ERROR	0	00
S7_ERR_UNKNOWN_ERROR	1	01
S7_ERR_WRONG_CP_DESCR	2	02
S7_ERR_NO_RESOURCE	3	03

Error	Decimal	Hexa
S7_ERR_INVALID_PARAMETER	7	07
S7_ERR_TOO_LONG_DATA	8	08
S7_ERR_TOO_MANY_DLL_USERS	9	09
S7_ERR_WRONG_IND_CNF	10	0A
S7_ERR_SERVICE_NOT_SUPPORTED	11	0B
S7_ERR_INVALID_CREF	20	14
S7_ERR_CONN_NAME_NOT_FOUND	23	17
S7_ERR_INVALID_ORDERID	30	1E
S7_ERR_ORDERID_USED	31	1F
S7_ERR_OBJ_UNDEFINED	50	32
S7_ERR_OBJ_ATTR_INCONSISTENT	51	33
S7_ERR_OBJ_ACCESS_DENIED	53	35
S7_ERR_INVALID_DATA_SIZE	80	50
S7_ERR_RECEIVE_BUFFER_FULL	81	51
S7_ERR_FW_ERROR	90	5A
S7_ERR_MINI_DB_TYPE	100	64
S7_ERR_MINI_DB_VALUE	101	65
S7_ERR_SERVICE_VFD_ALREADY_USED	112	70
S7_ERR_SERVICE_CONN_ALREADY_USED	113	71
S7_ERR_CONN_ABORTED	120	78
S7_ERR_INVALID_CONN_STATE	121	79
S7_ERR_MAX_REQ	122	7A
S7_ERR_CONN_CNF	123	7B
S7_ERR_INVALID_CYCL_READ_STATE	130	82
S7_ERR_INSTALL	140	8C
S7_ERR_INTERNAL_ERROR	141	8D
S7_ERR_NO_SIN_SERV	142	8E
S7_ERR_NO_LICENCE	143	8F
S7_ERR_SYMB_ADDRESS	150	96
S7_ERR_SYMB_ADDRESS_INCONSISTENT	151	97

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

No help found for this error. (RT Uni)

The help function contains only the most important error codes. Please contact the WinCC hotline.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42C2 - L4_DLL_E_LOGDEV (RT Uni)

The logical device is not defined in the registry.

- Communication driver not installed correctly.
- Entry damaged or deleted in the registry.
- Check the setting of the logical device name with the "Set PG/PC interface" program.
- Check the setting for the logical device name in the "System parameter - Device" mask.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 42D2 - L4_DLL_E_NO_SMC (RT Uni)

The connection to the adapter is disturbed.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7000 - CEC_UDEACT (RT Uni)

- Unit will be deactivated.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7001 - CEC_STPCHK (RT Uni)

Connection closed due to Stop check.

- The connection was terminated by the channel because a "STOP", "HOLD" or "DEFECTIVE" VMD status was detected for the CPU.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7002 - CEC_LACKTO (RT Uni)

Lifebeat acknowledgement error.

- The connection was terminated because the acknowledgment of the lifebeat frame was not received within the expected period.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7003 - CEC_RAWFKT (RT Uni)

Connection closed via raw data function

- The connection function was activated via a raw data function.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7004 - CEC_OIDERR (RT Uni)

Order ID invalid

- The order ID that has to be processed is invalid.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7005 - CEC_DTLERR (RT Uni)

Data length incorrect.

- In the received PDU the data length is greater than the expected data length or 0. This display is output, for example, when an odd number of bytes is read from an AS314 prototype CPU (314-1AE00-0AB0).

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7006 - CEC_CONERR (RT Uni)

Connection closed or deactivated.

- Connection was terminated. That is why the job was rejected.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7007 - CEC_VMDERR (RT Uni)

It is not permitted to determine VMD status via lifebeat frame.

- The connection was terminated by the channel because a "STOP", "HOLD" or "DEFECTIVE" VMD status was detected for the CPU.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7008 - CEC_CONPRJ (RT Uni)

Connection data incorrect or corrupted.

- No connection parameters have been configured.
- The configured connection parameters are not permitted.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7101 - CEC_NORAM1 (RT Uni)

No memory available.

- No free memory could be allocated during creation of order list.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7102 - CEC_NORAM2 (RT Uni)

No memory available.

- No free memory could be allocated during creation of tag feedback data area.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 7900 - CEC_S7DOSE (RT Uni)

Necessary S7DOS version not loaded.

- A function was configured (for example, PMC message processing) that requires a specific version of the communication subsystem S7DOS.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

The connection is ready to use. (RT Uni)

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

The connection is currently being initialized (RT Uni)

Communication is only possible after the completion of the initialization.

- If the state is pending for several seconds, an overloading of the communication on the AS side might be the reason for this.
- Check the PM configuration in your AS.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

The connection to the AS is established (RT Uni)

The CPU is in STOP mode.

- To communicate with the CPU via WinCC, switch the CPU to RUN mode.
- You have the possibility to also monitor the PLC in STOP mode. System Diagnostics and Alarming can continue to be run in this way.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 8204 - L7_DEF_OBJ_INCONSISTENT (RT Uni)

The object's type format is inconsistent.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 8305 - L7_FUNC_NOT_AVAIL (RT Uni)

Functionality not available.

- AS-CPU type does not support the configured function.
- AS CPU does not contain the most recent firmware status.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 8404 - L7_FATAL_ERR (RT Uni)

AS protocol error.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 8405 - L7_CPU_IN_PROTECTED_STATE (RT Uni)

The remote block is in status DISABLE (PBK). The function cannot be executed.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error 8500 - L7_PDU_SIZE_ERR (RT Uni)

Wrong PDU size.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error D405 - L7_DGS_FKT_PAR_SYNTAX_ERR (RT Uni)

Service is not supported or syntax error for function parameters.

- AS-CPU type does not support the configured function.
- AS CPU does not contain the most recent firmware status.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

Error D406 - L7_NO_INFO (RT Uni)

Desired information not available.

- The requested data are not available.
- In the case of AS300 the message also occur, if two station at the same time access the desired data.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

S7 connection status (RT Uni)

Internal function correctly completed.

In this Property Page, the current connection status of the corresponding unit is displayed. You can use the displayed error code to quickly localize a communication error.

Select the corresponding connection and press the <F1> key, or double-click the error code.

Column	Description
Name	Connection name
Status	Connection status: ✓ Ready The connection is ready to use. 🛑 Faulty The connection is faulty. An error code output in the Error column. ❗ CPU Stop The AS is in STOP mode. ❗ init The connection is being initialized or uninitialized.
Fault	Error code in hexadecimal. Note: In each case the error code that occurred first is displayed.
PDU length	The PDU length is displayed here in bytes after the commection has been established.
Number of errors	Number of connection problems that occurred. Note: A connection problem is only counted if an available connection is disturbed , i.e. switches from 'available' to 'disturbed' or 'CPU Stop'.
Asyn. Events	Number of asynchronous S7DOS events.

Note

The table can contain additional columns that are not described here. These data are intended for additional error diagnostics and not relevant for the user.

See also

Error codes for connection faults (SIMATIC S7-300/400) (Page 1012)

16.8.1.3 Internal error codes and constants (RT Uni)

Internal error codes and constants (RT Uni)

The following tables contain the most important error codes and constants. The information is intended for "insiders". Therefore, we have not gone into more detail on the meanings of the codes and constants.

- iNA960 messages
- SCI messages
- S7DOS Function types
- S7DOS Error codes
- S7DOS Trace function

iNA960 messages (RT Uni)

Table 16-10

General iNA960 messages:			
OK_RESP	1	0x01	Request executed with no errors
OK_EOM_RESP	3	0x03	Data block received with no errors
OK_DECIDE_REQ_RESP	5	0x05	Request executed with no errors
OK_CLOSED_RESP	7	0x07	Connection terminated by local user

iNA960 Error messages:			
INVALID_REQ	2	0x02	Incorrect request block
NO_RESOURCES	4	0x04	No resources free in CP
UNKNOWN_REFERENCE	6	0x06	Incorrect OPEN reference defined
BUFFER_TOO_SHORT	8	0x08	User buffer too short
BUFFER_TOO_LONG	10	0x0A	User buffer too long
ILLEGAL_REQ	12	0x0C	Incorrect "negot_options" defined

iNA960 Error messages:			
REM_ABORT	14	0x0E	Connection terminated by remote station
LOC_TIMEOUT	16	0x10	Timeout
UNKNOWN_CONN_CLASS	18	0x12	Unknown connection class
DUP_REQ	20	0x14	Connection already established
CONN_REJECT	22	0x16	Connection request rejected by remote
NEGOT_FAILED	24	0x18	Connection termination incorrect negot option
ILLEGAL_ADDRESS	26	0x1A	Incorrect transport address
NETWORK_ERROR	28	0x1C	Bus or CP faulted
PROTOCOL_ERR	30	0x1E	Protocol error
ILLEGAL_RB_LENGTH	32	0x20	Incorrect request block length

See also

Internal error codes and constants (Page 1026)

SCI messages (RT Uni)

See description in the "SINEC Communication Interface SCI" manual (A/5-15).

SCI messages			
SCP_OK	0	0x00	No error
SCP_INCONS	201	0xC9	Minor device number is not 00
SCP_RESOURCE	202	0xCA	DPRAM memory request invalid
SCP_CONFIG	203	0xCB	Configuration error (NUM_PROCS)
SCP_NOCONFIG	204	0xCC	SCP driver not configured
SCP_PARAM	206	0xCE	Incorrect mode
SCP_DEVOPEN	207	0xCF	Open already performed
SCP_BOARD	208	0xD0	Board not inserted/recognized
SCP_SOFTWARE	209	0xD1	IRQ error or software not found
SCP_MEM	210	0xD2	Low memory in DPRAM
SCP_MODE	211	0xD3	Download process not yet completed
SCP_LOADER	212	0xD4	No response from loader
SCP_SIGNAL	213	0xD5	Process started asynchronously
SCP_NOMESS	215	0xD7	No message arrived for the process
SCP_USERMEM	216	0xD8	Buffer length length_of_buffer too small
SCP_WINDOW	217	0xD9	Too many SEND calls
SCP_TIMEOUT	219	0xDB	Timeout on SCP
SCP_ATTACH	220	0xDC	Reset not executed/channel still active
SCP_ILLEGAL_REQUEST	221	0xDD	Incorrect request
SCP_ERECOVERF	223	0xDF	Buffer not retrieved with scp_receive
SCP_ECLOSED	224	0xE0	All buffers assigned for the connection
EUSERMAX	225	0xE1	

SCI messages			
SCP_EINTR	226	0xE2	
SCP_BOARD_OPEN	231	0xE7	
SCP_NO_WIN_SERV	233	0xE9	
EPROTECT	234	0xEA	License not found

SCI messages		
SCP_DB_FILE_DOES_NOT_EXIST	240	0xF0
SCP_DB_FILE_CLOSE_NOT_OK	241	0xF1
SCP_SEND_NOT_SUCCESSFUL	242	0xF2
SCP_RECEIVE_NOT_SUCCESSFUL	243	0xF3
SCP_NO_DEVICE_AVAILABLE	244	0xF4
SCP_ILLEGAL_SUBSYSTEM	245	0xF5
SCP_ILLEGAL_OPCODE	246	0xF6
SCP_BUFFER_TOO_SHORT	247	0xF7
SCP_BUFFER_1_TOO_SHORT	248	0xF8
SCP_ILLEGAL_PROTOCOL_SEQUENCE	249	0xF9
SCP_ILLEGAL_PDU_ARRIVED	250	0xFA
SCP_REQUEST_ERROR	251	0xFB
SCP_NO_LICENSE	252	0xFC

Additional online DLL messages on the SCP interface			
E_TIMER_INIT	768	0x0300	WIN Settimer call unsuccessful
E_INIT_COM	769	0x0301	
E_NO_HW	784	0x0310	MPI module not found
E_HW_DEFECT	785	0x0311	Problem with the hardware
E_CNF	786	0x0312	Incorrect configuration parameter
E_BAUDRATE	787	0x0313	Incorrect baudrate/incorrect IntVector
E_HSA	788	0x0314	Incorrect HSA configured
E_TS	789	0x0315	Configured address already assigned
E_OCC	790	0x0316	HW_Device already assigned
E_INT_NOT_PROV	791	0x0317	Interrupt not available
E_INT_BUSY	792	0x0318	Interrupt allocated
E_SAP	793	0x0319	SAP deactivate: SAP not allocated
E_UNPLUGGED	794	0x031a	No remote station found
E_SYNI	795	0x031b	Syni error occurred
E_AMPRO	796	0x031c	AMPRO 2 reported a system error
E_BUFFSIZE	797	0x031d	No buffer of this size created
E_NO_FILE	800	0x0320	DLL/VxD File not found or entries in registry damaged
E_NO_ENTRY	801	0x0321	Address does not exist in DLL
E_VERSION	816	0x0330	Version conflict between SMC driver and SMC firmware
E_COMCNF	817	0x0331	Problem with the COM port configuration

Additional online DLL messages on the SCP interface			
E_NO_SMC	818	0x0332	SMC no longer responds
E_COMMBADID	819	0x0333	COM port is not configured
E_COMMOPEN	820	0x0334	COM port is not available
E_SMCBUSY	821	0x0335	Serial driver is currently in use with another configuration
E_SMCMODEM	822	0x0336	No connection currently exists to a PC/MPI cable
E_SMCNOLEG	823	0x0337	PC/MPI cable refuses job, required authorization is missing
E_ONLINE	896	0x0380	Internal error at the IOCTL interface
E_LOGDEV	897	0x0381	Logical device not in registry
E_L2DRIVER	898	0x0382	L2DRIVER entry is missing in the registry
E_L4DRIVER	900	0x0384	L4DRIVER entry is missing in the registry
E_SYSERROR	1023	0x03FF	System error

See also

Internal error codes and constants (Page 1026)

S7DOS Function types (RT Uni)

Table 16-11

S7DOS Function types	
S7O_S7_EVENT	1
S7O_S7AG_BESY_UPDATE	2
S7O_S7AG_BUB_CYCL_READ_CREATE	3
S7O_S7AG_BUB_CYCL_READ_START	4
S7O_S7AG_BUB_CYCL_READ_STOP	5
S7O_S7AG_BUB_CYCL_READ_DELETE	6
S7O_S7AG_BUB_READ_VAR	7
S7O_S7AG_BUB_WRITE_VAR	8
S7O_S7AG_COMPRESS	9
S7O_S7AG_LINK_IN	10
S7O_S7AG_MEM_MODE	11
S7O_S7AG_MSG_MODE	12
S7O_S7AG_PASSWORD	13
S7O_S7AG_READ_SZL	14
S7O_S7AG_READ_TIME	15
S7O_S7AG_RESUME	16
S7O_S7AG_START	17
S7O_S7AG_STOP	18
S7O_S7AG_TEST	19
S7O_S7AG_TEST_DELETE	20
S7O_S7AG_WRITE_TIME	21

S7DOS Function types	
S7O_S7BLK_DELETE	22
S7O_S7BLK_FINDFIRST	23
S7O_S7BLK_FINDNEXT	24
S7O_S7BLK_READ	25
S7O_S7BLK_WRITE	26
S7O_S7DB_CREATE	27
S7O_S7DB_OPEN	28
S7O_S7DB_COPY	29
S7O_S7DB_CLOSE	30
S7O_S7DB_DELETE	31
S7O_S7DOS_RELEASE	32
S7O_S7NET_GET_LIFE_LIST	33
S7O_S7NET_GET_DIRECT_PLC	34
S7O_S7DP_SET_SLAVE_ADDRESS	35
S7O_S7DP_SLAVE_DIAGNOSE	36
S7O_S7AG_PMC_MSG_MODE	37
S7O_S7AG_PMC_ON_OFF	38
S7O_S7AG_BRCV	39
S7O_S7AG_BSND	40
S7O_S7AG_PMC_ACK	41
S7O_S7AG_PMC_MLDG	42
S7O_S7AG_BUB_CYCL_READ_CREATE_CNF	43
S7O_S7AG_MSG_MODE_CNF	44
S7O_S7AG_PMC_UPDATE	45
S7O_S7L7_DOWNLOAD_DOMAIN	128
S7O_S7L7_UPLOAD_DOMAIN	129

See also

Internal error codes and constants (Page 1026)

S7DOS Error codes (RT Uni)

Table 16-12

Define	Dec	Hex	Error Message (Stand 7.1.99)
INVALID_BLOCK_TYPE_NUM	272	0110	Invalid block type or invalid block number.
INVALID_PARAM	274	0112	Invalid parameter.
INVALID_BLOCK_TYPE	275	0113	Invalid block type.
BLOCK_NOT_FOUND	276	0114	Block not found.
BLOCK_ALREADY_EXIST	277	0115	The block is already available.
BLOCK_IS_PROTECTED	278	0116	The block is read-only.
BLOCK_TOO_LARGE	279	0117	The block is too large.

Define	Dec	Hex	Error Message (Stand 7.1.99)
INVALID_BLOCK_NUM	280	0118	Invalid block number.
INCORRECT_PASSWORD	281	0119	You have entered an incorrect password.
PG_RESOURCE_ERROR	282	011A	PG-Resources error.
PLC_RESOURCE_ERROR	283	011B	AS resource error.
PROTOCOL_ERROR	284	011C	Protocol error.
TOO_MANY_CPU_BLKS	285	011D	Too many blocks (module specific limitation).
DATABASE_DISCONNECTED	286	011E	No connection to the database or S7DOS handle is invalid.
USER_BUFFER_TOO_SHORT	287	011F	The application buffer is too small.
END_OF_BLOCK_LIST	288	0120	End of block list.
CALLOC_ERROR	320	0140	Insufficient memory available.
REQ_INI_ERR	321	0141	The job cannot be processed because of missing resources.
SIMULATION_ERR	368	0170	The simulator could not be found.
DRIVER_NOT_INST	384	0180	The driver is not installed: Incompatible parameter or invalid driver handle.
DRIVER_ALREADY_OPEN	385	0181	The driver is already open or too many channels are open.
VERSION_MISMATCH	448	01C0	The versions are incompatible.
TABLE_STRUCT_MISMATCH	449	01C1	The field setup to the opened database does not correspond to the expected setup.
NO_VALID_SELECTION	450	01C2	s7blk_findnext() was called before s7blk_findfirst().
HEADER_MEMO_INCONSISTENCY	451	01C3	The length format in the block header does not correspond to the actual length of the section in the data storage system.
ID_FILE_ERROR	452	01C4	A problem occurred during processing of Last-ID file.
WRONG_BLOCK_FORMAT	453	01C5	Incorrect block format.
FILE_NOT_FOUND	454	01C6	File not found.
INVALID_BESY_KOMP	455	01C7	Invalid Besy Update Component.
DBCPY_TARGET_EXISTS	456	01C8	The database specified as target is already available.
WR_TABLE_ALREADY_LOCKED	457	01C9	The database is already locked by another application.
SKIP_UNEXPECTED_ERROR	458	01CA	An unexpected error occurred during positioning in the database.
DHCLOSE_ERROR	459	01CB	An invalid DATA4 pointer was transferred to the dhclose() call.
LIB_NOT_FOUND	460	01CC	The TBI-DLL could not be loaded.
FKT_NOT_FOUND	461	01CD	A function in the dynamic downloaded DLL could not be found.
STPCPY_NOT_FOUND	462	01CE	The StopCopy-DLL could not be loaded.
FILE_ACCESS_ERROR	463	01CF	Access protection violation.

Define	Dec	Hex	Error Message (Stand 7.1.99)
PASSWORD_NOT_FOUND	464	01D0	The password was not found.
DIAGNOSE_NOT_STARTED	480	01E0	Diagnostics are disabled.
DIAGNOSE_DATA_NOT_AVAILABLE	481	01E1	No diagnostics data are available.
DIAGNOSE_DATA_INCONSISTENT	482	01E2	The diagnostic data are inconsistent.
NOT_IMPLEMENTED	496	01F0	Function is not implemented.
INTERN_ERR	511	01FF	System error.
L7_UNKNOWN_ROSCTR	2049	0801	Internal error code, is being mapped.
L7_UNKNOWN_CN_ID	2050	0802	Internal error code, is being mapped
L7_UNKNOWN_DIENSTKENN	2051	0803	Internal error code, is being mapped
L7_TOO_MUCH_BLOCKS	2052	0804	Internal error code, is being mapped
L7_WRONG_FLAGS	2053	0805	S7 protocol: Invalid flags.
L7_INTERNAL_ERR	2054	0806	Internal error code, is being mapped
L7_UNKNOWN_ID1_ERR	2055	0807	Internal error code, is being mapped
L7_SEND_ERR	2064	0810	S7 protocol: Data could not be correctly sent.
L7_RECEIVE_ERR	2065	0811	S7 protocol: No job for the received data could be found.
L7_DIAG_ERR_UPDATE	3456	0D80	A diagnostic error has occurred.
EPR_VCC_ERROR	8208	2010	Internal error code, is being mapped
EPR_VPP_ERROR	8209	2011	Internal error code, is being mapped
EPR_NO_ADAPT	8210	2012	EPROM: The memory card adapter is missing.
EPR_REM_EXT_ADAPT	8211	2013	EPROM: The external memory card adapter has to be removed.
EPR_NO_MEM_CARD	8212	2014	EPROM: The memory card is missing.
EPR_CHECK_SUM_ERR	8213	2015	Internal error code, is being mapped
EPR_LEN_ERR	8214	2016	Internal error code, is being mapped
EPR_MOD_ADR_ERR	8215	2017	Internal error code, is being mapped
EPR_READ_ONLY	8216	2018	Internal error code, is being mapped
EPR_NOT_IMPLEMENT	8217	2019	Internal error code, is being mapped
EPR_AREA_NOT_PRES	8218	201A	Internal error code, is being mapped
EPR_ALGO_UNKNOWN	8219	201B	EPROM: Unknown configuration algorithm.
EPR_MOD_CHANGE	8220	201C	EPROM: The memory card was changed without authorization.
EPR_NO_OPEN	8221	201D	Internal error code, is being mapped
EPR_ILEG_BREAK	8222	201E	Internal error code, is being mapped
EPR_NO_FLASH	8223	201F	Internal error code, is being mapped
EPR_LEN_SEC_ERR	8224	2020	Internal error code, is being mapped
EPR_ADR_SEC_ERR	8225	2021	Internal error code, is being mapped
EPR_ADR_ERR	8226	2022	Internal error code, is being mapped
EPR_DB_SHRT_ERR	8227	2023	Internal error code, is being mapped
EPR_MODE_ERR	8228	2024	Internal error code, is being mapped
EPR_PROG_ERR	8229	2025	EPROM: Hardware error in the memory card.
EPR_PROG0_ERR	8230	2026	Internal error code, is being mapped
EPR_ERA_ERR	8231	2027	Internal error code, is being mapped

Define	Dec	Hex	Error Message (Stand 7.1.99)
EPR_RD_KENN_ERR	8232	2028	Internal error code, is being mapped
EPR_CMP_ERR	8233	2029	Internal error code, is being mapped
EPR_EMP_ERR	8234	202A	EPROM: The memory card is not empty.
EPR_EE_WR_ERR	8235	202B	Internal error, is being mapped
EPR_EE_TIMOUT_ERR	8236	202C	Internal error, is being mapped
EPR_SYS_ERR	8237	202D	EPROM: Internal error.
EPR_WR_PROT	8238	202E	EPROM: Read-only memory card.
EPR_NO_EXPOWER	8239	202F	EPROM: The external power supply is missing.
EPR_MINUS_12V_ERR	8240	2030	Internal error, is being mapped
EPR_REF_SP_ERR	8241	2031	Internal error, is being mapped
EPR_DEV_NOT_SUPPORTED	8242	2032	EPROM: The EPROM driver cannot support the set configuration interface on this computer.
EPR_PROM_ENTR_ERR	8256	2040	EPROM: External prommer is missing.
EPR_NO_BIOS_CENTADR	8257	2041	Internal error, is being mapped
EPR_QOUT_FACT_ERR	8258	2042	EPROM: Hardware error on the external prommer.
EPR_QUEUE_BUSY	8259	2043	Internal error, is being mapped
EPR_TIMER_ERR	8272	2050	Internal error, is being mapped
EPR_WRONG_WIN_MODE	8288	2060	Internal error, is being mapped
EPR_NO_GLOBAL_MEM	8289	2061	Internal error, is being mapped
EPR_PROGAS_ERR	8324	2084	EPROM: Configuration interface error.
EPR_FATAL_ERR	8328	2088	Internal error, is being mapped
EPR_ERR_KEYNOTFOUND	8452	2104	Internal error, is being mapped
EPR_ERR_RECBUFLEN	8470	2116	Internal error code, is being mapped
EPR_ERR_BLK_LEN	8488	2128	Internal error, is being mapped
EPR_ERR_FILEFULL	8534	2156	Internal error, is being mapped
EPR_DRV_NOT_INST	8560	2170	Internal error, is being mapped
EPR_DRV_ALREADY_OPEN	8561	2171	Internal error, is being mapped
EPR_INVALID_DRV_HANDLE	8562	2172	Internal error, is being mapped
EPR_DRIVER_NOT_OPEN	8563	2173	Internal error, is being mapped
EPR_WRONG_DEVICE_NAME	8564	2174	Internal error, is being mapped
EPR_NO_FLASH_AREA	8566	2176	EPROM: No flash area is available on the memory card.
EPR_WRONG_APPL_KEN	8567	2177	EPROM: The memory card has an incorrect application ID.
EPR_MOD_CONTENS_UNKNOWN	8568	2178	EPROM: The content of the memory card cannot be interpreted.
EPR_MOD_CHKSUM_ERR	8569	2179	A growler error has occurred in the content of the memory card.
EPR_PROGRAMMING_ERR	8592	2190	EPROM: A configuration error has occurred.
SRVERR_NO_CONV_ESTABLISHED	16394	400A	No communication with the S7OTBL server is possible. The server is not started.
SRVERR_SYS_ERROR	16399	400F	System error during the S7OTBL server client communication.

Define	Dec	Hex	Error Message (Stand 7.1.99)
NO_LIST_ENTRY	16415	401F	No corresponding entry is available in the asynchronous list.
LOCAL_ALLOC_ERR	16416	4020	No local memory is available.
LOCAL_LOCK_ERR	16417	4021	The local memory cannot be locked.
S7DOS_SRV_TIMEOUT	16418	4022	Timeout during waiting on WM_ENDE_L7.
WRONG_WIN_MODE	16480	4060	Incorrect operating mode in Windows.
NO_GLOBAL_MEM	16481	4061	No global memory is available.
L4_OK_RESP	16641	4101	Internal error code, is being mapped
L4_INVALID_REQ	16642	4102	Online: Internal error.
L4_OK_EOM_RESP	16643	4103	Online: OK_EOM_RESPONSE.
L4_NO_RESOURCES	16644	4104	Online: No resources in the driver are available.
L4_OK_DECIDE_REQ_RESP	16645	4105	Internal error code, is being mapped
L4_UNKNOWN_REFERENCE	16646	4106	Online: Unknown shortcut.
L4_OK_CLOSED_RESP	16647	4107	Online: The connection was terminated.
L4_BUFFER_TOO_SHORT	16648	4108	Internal error code, is being mapped
L4_NO_SRD_RSP	16649	4109	Online: Sending and receiving of data is not acknowledged.
L4_BUFFER_TOO_LONG	16650	410A	Internal error code, is being mapped
L4_OK_REJECT_CONN_RESP	16651	410B	Internal error code, is being mapped
L4_ILLEGAL_REQ	16652	410C	Internal error code, is being mapped
L4_REM_ABORT	16654	410E	Online: Connection was terminated.
L4_LOC_TIMEOUT	16656	4110	Online: No connection was established. The station does not respond.
L4_UNKNOWN_CONN_CLASS	16658	4112	Internal error code, is being mapped
L4_DUP_REQ	16660	4114	Online: The connection already exists.
L4_CONN_REJECT	16662	4116	Online: No connection was established. The partner refuses to establish the connection.
L4_NEGOT_FAILED	16664	4118	Internal error code, is being mapped
L4_ILLEGAL_ADDRESS	16666	411A	Online: Invalid address.
L4_NETWORK_ERROR	16668	411C	Online: Network error.
L4_PROTOCOL_ERR	16670	411E	Internal error code, is being mapped
L4_ILLEGAL_RB_LENGTH	16672	4120	Internal error code, is being mapped
L4_PPI_T_TI_ERROR	16674	4122	Internal error code, is being mapped
L4_NO_RESOURCE	16897	4201	Online: No resources in the driver are available.
L4_CONNECTION_EXIST	16898	4202	Internal error code, is being mapped
L2_OPEN_DONE	16899	4203	Internal error code, is being mapped
L4_SCI_STATION_NA	16913	4211	Online: No station is available on the subnet.
L4_SCI_STATION_DS	16914	4212	Online: The station is not online.
L4_SCI_STATION_NO	16915	4213	Online: Too many stations.
L4_SCI_NOTIMPLEMENTED	16917	4215	Online: The function is not implemented or is invalid in the current context.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L4_SCI_E_DPX2_ERROR	16918	4216	Online: Incorrect station address of the DP Slave or error message of the DP Slave.
L4_NO_CONFIG_FOUND	16936	4228	The bus parameters could not be retrieved automatically (ONLINE). No stations at the bus that send bus parameter frames. Set your MPI/PROFIBUS interface manually.
L4_ONLY_BAUDRATE_FOUND	16937	4229	Online: Only the baud rates could be retrieved.
L4_ONLY_PROFILE_FOUND	16938	422A	Online: Only the profile could be retrieved.
L4_SCI_E_UNPLUGGED	16944	4230	Online: No other active station is available.
L4_SCI_E_SYNI	16945	4231	Online: The bus is faulted.
L4_DLL_E_HSA	16946	4232	Online: Incorrect maximal number of stations (HIGHEST_ADR).
L4_SCI_E_AMPRO	16947	4233	Online: System error.
L4_DLL_E_TIMER_INIT	17056	42A0	Online: The Windows timer cannot be set.
L4_DLL_E_INIT_COM	17057	42A1	Online: The COM interface cannot be initialized or opened.
L4_DLL_E_NO_HW	17072	42B0	Online: No hardware found.
L4_DLL_E_HW_DEFECT	17073	42B1	Online: The hardware is defective.
L4_DLL_E_CNF	17074	42B2	Online: The driver is configured incorrectly or the registry contains invalid parameters.
L4_DLL_E_BAUDRATE	17075	42B3	Online: An incorrect baud rate or an incorrect interrupt vector is set, or the local MPI address is greater than the maximum station address.
L4_DLL_E_TS	17077	42B5	Online: The set local station address (TS_ADR) is already assigned.
L4_DLL_E_OCC	17078	42B6	Online: The hardware device (DEVICE) cannot be used several times.
L4_DLL_E_INT_NOT_PROV	17079	42B7	Online: The defined interrupt vector (IRQ) is not available on this module.
L4_DLL_E_INT_BUSY	17080	42B8	Online: The set interrupt vector (IRQ) is already allocated.
L4_DLL_E_SAP	17081	42B9	Internal error code, is being mapped
L4_DLL_E_NO_FILE	17088	42C0	Online: The selected communication driver cannot be loaded; the file was not found.
L4_DLL_E_NO_ENTRY	17089	42C1	Online: The function is not realized on the loaded communication driver.
L4_DLL_E_LOGDEV	17090	42C2	Online: The logical device is not defined in the registry.
L4_DLL_E_VERSION	17104	42D0	Online: The output inventory of driver and adapter or PC/MPI cable are incompatible.
L4_DLL_E_COMCNF	17105	42D1	Online: No interrupt of PC/MPI cable was received.
L4_DLL_E_NO_SMC	17106	42D2	Online: The connection to the adapter is disturbed.
L4_DLL_E_COMMBADID	17107	42D3	Online: The COM interface is not configured under Windows.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L4_DLL_E_COMMOPEN	17108	42D4	Online: The COM interface is currently unavailable.
L4_DLL_E_SMCBUSY	17109	42D5	Online: The serial driver is currently used by another application with another configuration.
L4_DLL_E_SMCMODEM	17110	42D6	Online: There is still no remote connection or logical connection to the TS Adapter.
L4_DLL_E_SMCNOLEG	17111	42D7	Online: The TS Adapter refuses the job due to missing required legitimation.
L4_LIB_WIN_SYS_ERR	17120	42E0	Online: Windows system error in communication driver.
L4_LIB_NO_GLOBAL_MEM	17134	42EE	Online: No global memory is available.
L4_LIB_NO_SIN_SERV	17135	42EF	Online: SIN_SERV is not started.
L4_SCI_STATION_NOT_ONLINE	17146	42FA	Online: The station is not online.
L4_SCI_RB_ERR	17147	42FB	Internal error code, is being mapped
L4_SCI_MAX_REQ_NR	17148	42FC	Internal error code, is being mapped
L4_SCI_DRV_ALREADY_OPEN	17149	42FD	Internal error code, is being mapped
L4_SCI_DRV_ERR	17150	42FE	Internal error code, is being mapped
L4_SCI_DRV_NOT_INST	17151	42FF	Internal error code, is being mapped
NO_LOC_SUBNET_ENTRY	17153	4301	Invalid local subnet number in the file S7DPMPI.INI.
NO_TABLE_ENTRY	17154	4302	The station on this subnet is unavailable.
WRONG_RACK_SLOT	17155	4303	Incorrect rack/slot in the module table.
WRONG_NODENAME	17156	4304	Incorrect format of the node name in S7db_open (online).
INVALID_S7_TABUF	17157	4305	Invalid S7-Transport_Address_Buffer.
INVALID_S7_WUSERID	17158	4306	The job cannot be found. Incorrect wUserID or incorrect Window handle.
L7_DPT_ERROR	17456	4430	The data record cannot be read/written.
RES_VON	28672	7000	The area from 0x7000 to 0x7fff is reserved for applications.
RES_BIS	32767	7FFF	The area from 0x7000 to 0x7fff is reserved for applications.
L7_INVALID_CPU_STATE	32769	8001	The function is invalid in this operating mode, or an incorrect minimum scan cycle time is set.
L7_DOMAIN_LOADING_ERR	32771	8003	S7 protocol error: A domain transfer error has occurred. The domain content (e.g. block) is incorrect.
L7_S5_INTERN_ERR	33023	80FF	S7 protocol: Internal error.
L7_COMMON_ERR	33024	8100	Application, general error: Unknown service for the remote module.
L7_MISSING_CONTEXT	33028	8104	The service is not supported. An unknown error in PDU adapter or service has occurred.
L7_DEF_OBJ_INCONSISTENT	33284	8204	The object's type format is inconsistent.
L7_ALREADY_COPIED	33285	8205	A copied object's variant is available.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L7_NO_RAM	33537	8301	The memory space on the module is not enough or the specified memory medium is not available.
L7_NO_RESOURCE	33538	8302	A resource bottleneck is available or the processor resources are unavailable.
L7_ERRCOD_INI_NO_RES	33540	8304	Domain: An additional simultaneous uploading is no longer possible. A resources bottleneck is available.
L7_FUNC_NOT_AVAIL	33541	8305	The functionality is not available. A resources bottleneck is available.
L7_INVALID_SEQUENCE	33793	8401	S7 protocol error: Incorrect sequence of services (e.g. during loading or uploading of a block).
L7_WRONG_PI_STATE	33794	8402	Error during the service sequence. The service cannot be performed due to the mode of the addressed object.
L7_FATAL_ERR	33796	8404	S7 protocol: A significant error has occurred. The function cannot be executed.
L7_CPU_IN_PROTECTED_STATE	33797	8405	The remote block is in status DISABLE (PBK). The function cannot be executed.
L7_PDU_SIZE_ERR	34048	8500	S5 protocol error: Wrong PDU size.
L7_SERVICE_CANCELED	34051	8503	The service has been prematurely canceled.
L7_NO_OBJ_ACCESS	34561	8701	The object access is not supported.
L7_INVALID_ADDRESS	34562	8702	S7 protocol error: Access to a remote object was rejected.
L7_OBJECT_ACCESS_DENIED	34563	8703	Access error: Access to a remote object was rejected.
L7_ACCESS_ERR_OBJ	34564	8704	Access error: The object is damaged.
L7_INVALID_REQ_NB	53249	D001	D001: Protocol error: Job number is invalid.
L7_INVALID_REQ_VER	53250	D002	D002: Parameter error: Job variant is invalid.
L7_INVALID_FKT	53251	D003	D003: Parameter error: The function is invalid.
L7_INVALID_REQ_STAT	53252	D004	D004: Parameter error: Job status is invalid.
L7_INVALID_END_OF_REQ	53253	D005	D005: Parameter error: Job termination is invalid.
L7_INVALID_ABORT_OF_CONN	53254	D006	D006: Parameter error: The ID for the connection termination is invalid.
L7_INVALID_NB_OF_BUF	53255	D007	D007: Parameter error: The number of buffer elements is invalid.
L7_INVALID_GEAR_DOWN	53256	D008	D008: Parameter error: The reduction factor is invalid.
L7_INVALID_NB_OF_EXEC	53257	D009	D009: Parameter error: The execution number is invalid.
L7_INVALID_TRIG_EVENT	53258	D00A	D00A: Parameter error: The trigger event is invalid.
L7_INVALID_TRIG_COND	53259	D00B	D00B: Parameter error: The trigger requirement is invalid.
L7_TRIG_EVENT_ERR_NO_BLK	53265	D011	D011: Parameter error: The block is not available.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L7_TRIG_EVENT_WRONG_OFF	53266	D012	D012: Parameter error: Incorrect address in the block.
L7_TRIG_EVENT_ERR_BLKTYP	53267	D013	D013: Parameter error: The block type is invalid.
L7_TRIG_EVENT_BLK_IS_MOD	53268	D014	D014: Parameter error: The block is being deleted/overloaded.
L7_TRIG_EVENT_ERR_VARADDR	53269	D015	D015: Parameter error: The tag address is invalid.
L7_TRIG_EVENT_ERR_FOUND_BL	53270	D016	D016: Parameter error: No block can be activated.
L7_TRIG_EVENT_ERR_SYS_TRIGGER	53271	D017	D017: Parameter error: The SYS trigger number is invalid.
L7_TRIG_COND_PATH_ERR	53285	D025	D025: Parameter error: The path is invalid.
L7_TRIG_COND_ACC_ERR	53286	D026	D026: Parameter error: The access type is invalid.
L7_TRIG_COND_NB_OF_DB_ERR	53287	D027	D027: Parameter error: The number of DBs is invalid.
L7_SEGMENT_NEXT_ERR	53297	D031	D031: Protocol error: CN-ID, S-ID or ID1 in the following segment do not match the first segment.
L7_SEGMENT_LEN_ERR	53298	D032	D032: Parameter error: Incorrect length of the buffer result.
L7_REQUEST_LEN_ERR	53299	D033	D033: Protocol error: Job length is invalid.
L7_PARAM_CODING_ERR	53311	D03F	D03F: Coding error: Other error in the parameter part (e.g. reserve bytes do not equal NULL).
L7_STALI_ID_ERR	53313	D041	D041: Data error: The stali ID is invalid.
L7_VAR_ADDR_ERR	53314	D042	D042: Data error: The tag address is invalid.
L7_REQ_DOES_NOT_EXIST	53315	D043	D043: Data error: The referenced job is not available.
L7_INVALID_VAR_VALUE	53316	D044	D044: Data error: The tag value is invalid.
L7_BASP_ERR	53317	D045	D045: Data error: Quitting the BASP control (ODIS) during HOLD is invalid.
L7_MEASUREMENT_ERR	53318	D046	D046: Data error: The measuring level during runtime measuring is invalid.
L7_HIERARCHY_ERR	53319	D047	D047: Data error: The hierarchy for 'Read job list' is invalid.
L7_INVALID_DEL_ID	53320	D048	D048: Data error: The delete ID for 'Delete job' is invalid.
L7_INVALID_SUB_ID	53321	D049	D049: The replace ID for 'Replace job' is invalid.
L7_INVALID_ENTRANCE_DATA	53322	D04A	D04A: Error during execution of 'programstatus'.
L7_DATA_CODING_ERR	53343	D05F	D05F: Coding error: Other error in the data part (e.g. reserve bytes do not equal NULL, ...).
L7_REQ_MEM_ERR	53345	D061	D061: Resource error: No memory for the job is available.
L7_FULL_REQ_LIST	53346	D062	D062: Resource error: The job list is full.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L7_TRIG_EVENT_IS_USED	53347	D063	D063: Resource error: The trigger event is allocated.
L7_TOO_SMALL_BUF	53348	D064	D064: Resource error: The memory space for an element of the result buffer is too small.
L7_TOO_MANY_BUF	53349	D065	D065: Resource error: The memory space for several elements of the result buffer is too small.
L7_NO_TIMER_AVAIL	53350	D066	D066: Resource error: Another job allocates the timer for the runtime measurement.
L7_PBUS_ALREADY_USED	53351	D067	D067: Resource error: Another job allocates the P bus during 'Control selection'.
L7_INVALID_FKT_IN_STATE	53377	D081	D081: Operation mode error: The function in the current operation mode is invalid.
L7_OPERATION_STATE_ERR	53378	D082	Operation mode error: Operating mode HOLD cannot be quit.
L7_PROT_LEVEL_ERR	53409	D0A1	D0A1: Protection and coding: The function in the current protection level is not allowed.
L7_MOD_FKT_ERR	53410	D0A2	D0A2: Protection and coding: The memory-modifying OVS function is running.
L7_CTRL_IS_INST	53411	D0A3	D0A3 Protection and coding: "Control selection" is already set up.
L7_FORCE_IS_INST	53412	D0A4	D0A4: Protection and coding: "Forcing" is already set up.
L7_NO_REF_TO_REF	53413	D0A5	D0A5: Protection and coding: The current job references another job in the communication partner which is not available.
L7_TE_AT_ONCE_REQ	53414	D0A6	D0A6: Protection and coding: The job cannot be locked/unlocked.
L7_REQ_NO_DELETE	53415	D0A7	D0A7: Protection and coding: The job cannot be deleted because e.g. it is currently being read.
L7_REQ_NO_CHANGE	53416	D0A8	D0A8: Protection and coding: The job cannot be replaced because e.g. it is currently being read or deleted.
L7_REQ_NO_READ	53417	D0A9	D0A9: Protection and coding: The job cannot be read because e.g. it is currently being deleted.
L7_TIME_LIMIT	53418	D0AA	D0AA: Timeout in process mode is exceeded.
L7_INVALID_INSTRUCT_PARAM	53419	D0AB	D0AB: The job parameter are invalid in process moe.
L7_INVALID_INSTRUCT_DATA	53420	D0AC	D0AC: The job data are invalid in process moe.
L7_OPERATIONS_MODE_REACHED	53421	D0AD	D0AD: Operating mode is already set.
L7_TRIG_EVENT_ON_OTHER_CONN	53422	D0AE	D0AE: The breakpoint is set using another connection. The job cannot be manipulated.
L7_VAR_ACC_ERR	53441	D0C1	D0C1: Processing warning: During accessing tags at least one error was detected.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L7_BZUE_TO_STOP	53442	D0C2	D0C2: Processing warning: Operating mode transition in STOP/HOLD during function call.
L7_BZUE_AFTER_VAR_ACC	53443	D0C3	D0C3: Processing warning: Operating mode transition and access error: Access errors occurred during loading of the tag(s).
L7_FKT_TIMEOUT	53444	D0C4	D0C4: Processing warning: Timer overflow during the runtime measuring.
L7_MOD_BLK_ERR	53445	D0C5	D0C5: Processing warning: The blocks have been deleted/uploaded since transition STOP.
L7_REF_REQ_WAS_DEL	53446	D0C6	D0C6: Processing error: The referenced job was deleted, because all jobs, that referenced it, were deleted.
L7_REQ_WAS_DEL_AFTER_BZUE	53447	D0C7	D0C7: Processing error: The job was deleted due to quitting of the operating mode STOP.
L7_EMPTY_STALI_ID	53448	D0C8	D0C8: Processing error: The status block was interrupted, because an empty stali ID was detected during processing.
L7_VERL_STAT_RES_HGOB	53449	D0C9	D0C9: Processing warning: Quitting the status area via reset of the background OB.
L7_VERL_STAT_RES_HGOB_ZUGERR	53450	D0CA	D0CA: Processing warning: Quitting the status area via reset of the background OB and access error during reading of the tags before quitting.
L7_OUTPUT_LOCK_PA_ON	53451	D0CB	D0CB: The output lock of the I/O outputs is again enabled.
L7_SHORT_RESULT	53452	D0CC	D0CC: The data range for the test functions is limited by the timeout.
L7_INVALID_BL_NAME	53761	D201	D201: Syntax error in block name.
L7_INVALID_ARGUMENT	53762	D202	D202: Syntax error in the function parameters.
L7_INVALID_BL_TYPE	53763	D203	Internal error code, is being mapped
L7_NO_LINKED_BLK	53764	D204	Internal error code, is being mapped
L7_BL_ALREADY_INSERTED	53765	D205	D205: A linked block is already available in the RAM; no conditional copying is possible.
L7_INVALID_BL_NUMBER	53766	D206	D206: A linked block is already available in the EPROM; no conditional copying is possible.
L7_BL_IN_ROM	53767	D207	At least one block is already available in the EPROM.
L7_TOO_MANY_COPIED_BL	53768	D208	The maximum number of copied (unlinked) blocks on the module has been exceeded.
L7_MISSING_BL	53769	D209	D209: At least one unspecified block on the module is unavailable.
L7_TOO_MANY_BL_INS	53770	D20A	D20A: The maximum number of modules that could be linked with one job was exceeded.
L7_TOO_MANY_BL_DEL	53771	D20B	D20B: The maximum number of modules that could be deleted with one job was exceeded.
L7_NO_AE	53772	D20C	D20C: The OB cannot be copied, as the corresponding runtime level is not available.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L7_INVALID_YB	53773	D20D	The SDB cannot be evaluated (e.g. unknown number).
L7_NO_MORE_BL	53774	D20E	D20E: No block available.
L7_INVALID_BL_LEN	53775	D20F	D20F: The module-specific maximum block size was exceeded.
L7_BLK_NB_ERR	53776	D210	D210: The block number is invalid.
L7_RUNTIME_REL_ATTRIB_ERR	53778	D212	D212: Incorrect header attribute (runtime relevant).
L7_TOO_MANY_SDB	53779	D213	D213: Too many SDBs.
L7_INVALID_BL	53780	D214	D214: Incorrect context in the block.
L7_INVALID_USER_PROG	53781	D215	D215: Incorrect user program.
L7_USER_PROG_ERR	53782	D216	D216: Invalid user program- clear/reset module.
L7_SDB0_PROTECTION_ERR	53783	D217	D217: The specified protection level in the SDB0 is invalid.
L7_ACT_PAS_ATTRIB_ERR	53784	D218	D218: Incorrect attribute (active/passive).
L7_BL_LEN_ERR	53785	D219	D219: Incorrect block lengths (e.g. incorrect length of the first section or the whole block).
L7_LOCAL_DATA_LEN_ERR	53786	D21A	D21A: The length of the local data (e.g. odd. for OB<20, local data length/ data stack too large) or read-only ID is incorrect.
L7_COMPRESS_ERR	53787	D21B	D21B: The module cannot compress, or the compress process was prematurely interrupted.
L7_WRONG_CPU_LIMITS	53789	D21D	D21D: The transferred dynamic qualified project specifications are invalid. They do not match the removal of CPU or the current user program. Check your settings and transfer them again.
L7_SDB_LINK_IN_ERR	53790	D21E	D21E: Error during configuration of external modules inside linking of a SDB.
L7_INVALID_LANGUAGE	53792	D220	D220: The language setting is invalid.
L7_MPI_PARAM_ERR	53793	D221	D221: Error in the SDB for the connection management (incorrect MPI parameter in the SDB0 or error in the connection description (SDBs)).
L7_IK_PARAM_ERR	53794	D222	D222: Error in the SDB with GD configuration (incorrect parameter in the GD-SDB).
L7_PBK_PARAM_ERR	53795	D223	D223: Error in instance DB for PBK, or the maximum number of instance DBs was exceeded.
L7_PMC_ERR	53796	D224	There is an error in the setup of the SCAN-SDB.
L7_DP_ERR	53797	D225	There is an error in the setup of the DP-SDB.
L7_BLK_STRUCT_WRONG	53798	D226	D226: A structure error occurred in a module.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L7_BL_TOO_LONG	53808	D230	D230: Incorrect resource specification. The block is longer than specifed, or the information provided at start of loading does not match.
L7_OB_WITHOUT_AE	53809	D231	D231: At least one loaded the OB cannot be copied, because the corresponding runtime level is not available.
L7_LOAD_PRG_BLK_NB_ERR	53810	D232	D232: At least one block number of a loaded module is invalid.
L7_EPR_BLK_ALREADY_EXIST	53811	D233	At least one loaded module is already available in EPROM.
L7_BLK_LOADING_2X	53812	D234	D234: The block is available twice in the specified memory medium or in the job.
L7_BLK_WRONG_CRC	53813	D235	D235: The block contains a checksum error.
L7_BLK_NO_CRC	53814	D236	D236: The block contains no checksum error.
L7_COORD_RULE	53824	D240	D240: The coding regulations were violated.
L7_FEW_PROT_LEVEL	53825	D241	D241: The protection level of the function is too small.
L7_SECURITY_LEVEL	53826	D242	AS protection error.
L7_OSUPDATE_WRONG_VER	53840	D250	D250: The update and module ID or the output release do not match.
L7_OSUPDATE_ORDER_ERR	53841	D251	D251: Incorrect sequence in the operation system components.
L7_OSUPDATE_CHECKSUM_ERR	53842	D252	D252: Checksum error.
L7_OSUPDATE_NO_LOADER	53843	D253	D253: There is no runtime download available, an update is only via the memory card possible.
L7_OSUPDATE_MEM_ERR	53844	D254	D254: Memory error in operating system.
L7_COMPILE_ERR	53888	D280	D280: Compiling error in AS 300.
L7_KOOR1_TRIGGER_ACTIVE	53921	D2A1	D2A1: An additional block function or a trigger on a block is active. Close the other online function.
L7_KOOR2_TRIGGER_ACTIVE	53922	D2A2	D2A2: A trigger is active on a block. Terminate the test function.
L7_KOOR3_TRIGGER_NOT_ACTIVE	53923	D2A3	D2A3: The block is not active (sequenced) or the block has to be deleted. Repeat the function later.
L7_KOOR4_BLOCK_IN_WORK	53924	D2A4	D2A4: The block is currently processed by another block function. Repeat the function later.
L7_KOOR6_PRG_SAVE	53926	D2A6	D2A6: Saving and changing the user program are simultaneously impossible. Repeat the function later.
L7_KOOR7_BLOCK_NOT_RUNNING	53927	D2A7	D2A7: The block has the attribute ´Unlinked´or is not being processed. A test function on this block is impossible.
L7_KOOR8_TST_RUNNING	53928	D2A8	D2A8: A running test function prevents the CPU configuration. Terminate the test function.
L7_KOOR9_CPU_PARAM	53929	D2A9	D2A9: The CPU is currently being re-configured. "Load user program" is simultaneously impossible. Repeat the function later.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L7_KOOR4_BGR_PARAM	53930	D2AA	D2AA: Modules are being currently re-configured. Repeat the function later.
L7_H_CPU_REORG_MEM	53931	D2AB	D2AB: Modifying the dynamic qualified project specifications is active. The user program is being re-evaluated. Wait until the end of the reevaluation and repeat your job.
L7_INVALID_SZL_ID	54273	D401	D401: Invalid SZL-ID.
L7_INVALID_INDEX	54274	D402	D402: Invalid INDEX.
L7_DGS_CONN_ALREADY_ANNOU	54275	D403	D403: The service is already signed in/signed out (diagnose/PMC).
L7_MAX_USER_NB	54276	D404	D404: The maximum station number was reached.
L7_DGS_FKT_PAR_SYNTAX_ERR	54277	D405	D405: Service is not supported or syntax error for the function parameters.
L7_NO_INFO	54278	D406	D406: The desired information is not available.
L7_DIAGNOSE_ERR	54279	D407	A diagnostics error has occurred.
L7_DIAG_ERR_UPDATE	54280	D408	A diagnostics error has occurred.
L7_DIAG_ERR_DPBUS	54281	D409	A diagnostic error has occurred on the DB bus.
L7_PRT_FKT_PAR_SYNTAX_ERR	54785	D601	D601: Syntax error in function parameter.
L7_PASSWORD_ERR	54786	D602	D602: You have entered an incorrect password.
L7_PRT_CONN_ALREADY_ANNOU	54787	D603	D603: The connection is already allowed.
L7_PRT_CONN_ALREADY_FREE	54788	D604	D604: The connection is already approved.
L7_NO_PASSWORD	54789	D605	D605: Not allowed due to unavailable passord.
L7_INVALID_VAR_ADDR	55297	D801	D801: At least one tag address is invalid..
L7_UNKNOWN_REQ	55298	D802	D802: The specified job is not available.
L7_INVALID_REQ_STATUS	55299	D803	D803: Invalid job status.
L7_INVALID_CYCLIC_TIME	55300	D804	D804: Invalid cycle time (time basis or multiple is invalid).
L7_NO_MORE_CYCLIC_REQ	55301	D805	D805: An additional cyclic reading job cann not be set up.
L7_INVALID_CYCLIC_REQ_STATE	55302	D806	D806: The referenced job is in a status, in which the requested function cannot be executed.
L7_CYCLIC_TIME_TOO_SHORT	55303	D807	D807: Interruption of the function due to overload, i.e., the processing of the reading cycle takes longer than the set cycle time.
L7_WRONG_TSY_FORMAT	56321	DC01	DC01: Error in the specification of date and time.
L7_H_CPU_IS_MASTER	57857	E201	E201: CPU is already master:
L7_H_CPU_AUA_MODULE	57858	E202	E202: Link-up and update impossible due to different user program in flash module.
L7_H_CPU_AUA_FW	57859	E203	E203: Link-up and update impossible due to different firmware.
L7_H_CPU_AUA_MEM	57860	E204	E204: Link-up and update impossible due to different memory expansion.

Define	Dec	Hex	Error Message (Stand 7.1.99)
L7_H_CPU_AUA_SYNC	57861	E205	E205: Abort link-up/update due to synchronization error.
L7_H_CPU_AUA_KOORD	57862	E206	E206: Reject link-up/update due to coordination violation.
L7_INVALID_ID2	61185	EF01	S7 protocol error: Error in ID2: In the job only 00H is valid.
L7_MISSING_CAPABILITY	61186	EF02	S7 protocol error: Error in ID2: The production facility record is not available.
DEMO_RETURN_CODE	65281	FF01	The function in the demo version is invalid.

See also

Internal error codes and constants (Page 1026)

S7DOS Trace function (RT Uni)

The S7DOS trace can be controlled via the following registry entry:

HKEY_LOCAL_MACHINE/SOFTWARE/SIEMENS/SINEC/TrcParams
"Format"="long"
"Output"="file"
"Level"="0x71000000"

Format

The "Format" parameter can be set to "long" or "short". This affects the line layout of a trace line. A time stamp is input for "long".

Output

The "Output" parameter determines where the output is saved. If output is set to "file", the outputs are written to C:\tmp\S7FILE1.TRC or C:\tmp\S7FILE2.TRC. These files are limited to approx. 2 MB and are written according to the alternating buffer principle.

Level

The parameter level determines what kind of outputs are saved in the tracefile. This parameter is a bit-by-bit "OR" logic operation of the following constants:

Level		
S7TRCAI	0x10000000UL	Application Interface
S7TRCL7	0x01000000UL	Lower Interface (Layer 7 PDUs)
S7TRCL4	0x40000000UL	Lower Interface (Layer 4)
S7TRCL2	0x20000000UL	Lower Interface (Layer 2)

If all possible trace outputs have to be activated, the level has to be set to ‘0x7FFFFFFF’. The information is evaluated on activation of S7DOS. At this time the folder C:\tmp has to be available. The folder is not automatically created by S7DOS.

See also

Internal error codes and constants (Page 1026)

16.8.1.4 API error texts (RT Uni)

API error messages

The most important error messages of S7 channel are listed in this section. If an error with an error code that is not in the table occurs, please call the WinCC hotline.

- Error during initialization of the S7 communication driver for 'unitname' unit 'devicename' device.
- Error during loading the S7 communication driver.
- For the configured functions is a S7DOS version xx or higher necessary.
- Error xx occurred in 'functionname' function

Error during initialization of the S7 communication driver for 'unitname' unit 'devicename' device. (RT Uni)

An error occurred while the communication drivers were being initialized.

- The configured device is not available.
- An error occurred during the device driver is activated.
- Copy protection error occurred.
- The bus connector to the MPI card is not inserted.

See also

API error messages (Page 1045)

Error during loading the S7 communication driver (RT Uni)

The communication subsystem SAPI-S7 or S7DOS could not be loaded.

- Communication driver not installed correctly.
- The DLLs required by the communication driver could not be loaded correctly.
- Path setting incorrect or required communication drivers deleted or moved.

See also

API error messages (Page 1045)

For the configured functions is a S7DOS version xx or higher necessary. (RT Uni)

Functions were configured (for example, PMC message processing) that are not supported by the installed communication subsystem.

See also

API error messages (Page 1045)

Error xx occurred in 'functionname' function (RT Uni)

The error indication xx occurred during the processing of the specified channel function.

The most important error codes are listed in this section. If an error occurs with an error code that is not included in this table, please call the hotline.

See also

API error messages (Page 1045)

Error 1 - EC_NOIMPL - Function not implemented (RT Uni)

The called function is not implemented in the S7 channel.

See also

API error messages (Page 1045)

Error 2 - EC_STRUFE - structure error (RT Uni)

The language DLLs do not belong to the installed S7 channel.

See also

API error messages (Page 1045)

Error 3 - EC_ILEGAL - Function call not permitted (RT Uni)

The call of the function with the transferred parameters in S7 channel is not allowed

See also

API error messages (Page 1045)

Error 4 - EC_NO_RAM - No free memory (RT Uni)

The S7 channel could not create the memory that is required for the function.

See also

API error messages (Page 1045)

Error 5 - EC_NOFILE - File not available (RT Uni)

The specified file is not available.

See also

API error messages (Page 1045)

Error 6 - EC_LNGERR - Error during switching languages (RT Uni)

An error occurred while switching languages.

See also

API error messages (Page 1045)

Error 7 - EC_UNITNV - Invalid unit ID (RT Uni)

The specified UNIT-ID is invalid.

See also

API error messages (Page 1045)

Error 8 - EC_NOUNIT - No unit available (RT Uni)

The specified UNIT is not available.

See also

API error messages (Page 1045)

Error 9 - EC_UNITNA - Unit not active (RT Uni)

The specified UNIT is not active.

See also

API error messages (Page 1045)

Error 10 - EC_PTRERR - Incorrect pointer (RT Uni)

The pointer that is transferred in the function call is incorrect.

See also

API error messages (Page 1045)

Error 11 - EC_TIMERR - Error during starting the internal timer (RT Uni)

An error occurred during starting of the internal timer.

See also

API error messages (Page 1045)

Error 12 - EC_S7LERR - Error during loading of the S7 communication driver (RT Uni)

The communication subsystem SAPI-S7 or S7DOS could not be loaded.

- Communication driver not installed correctly.
- The DLLs that are required by the communication driver could not be loaded correctly.
- Path setting incorrect or required communication drivers deleted or moved.

See also

API error messages (Page 1045)

Error 13 - EC_S7IERR - Error during initializing of the S7 communication driver (RT Uni)

An error occurred during initializing of the S7 communication driver.

- The configured device is not available.
- An error occurred during the device driver is activated.
- Copy protection error occurred.
- The bus connector to the MPI card is not inserted.

See also

API error messages (Page 1045)

Error 14 - EC_CONERR - Connection fault (RT Uni)

A read/write job was sent to a disrupted connection.

See also

API error messages (Page 1045)

Error 15 - EC_PARERR - Incorrect parameter supply (RT Uni)

An incorrect parameter was specified during a function call.

See also

API error messages (Page 1045)

Error 16 - EC_DATERR - Data error occurred (RT Uni)

The transferred connection data are invalid or damaged.

See also

API error messages (Page 1045)

Error 17 - EC_CONDAT - Incorrect connection data (RT Uni)

The transferred connection data are invalid or damaged.

See also

API error messages (Page 1045)

Error 18 - EC_WNDERR - SINEC Windows incorrect (RT Uni)

The SINEC Windows required for the communication could not be created.

See also

API error messages (Page 1045)

Error 19 - EC_RAWERR - Error occurred in the raw data structure (RT Uni)

The transferred raw data job is invalid or incorrect parameters were specified.

See also

API error messages (Page 1045)

Error 20 - EC_INTRDY - Internal function completed (RT Uni)

Internal function correctly completed.

See also

API error messages (Page 1045)

Error 21 - EC_EVNERR - EventNumber in MemberName n incorrect (RT Uni)

Internal function correctly completed.

Tag configuration for status processing incorrect.

See also

API error messages (Page 1045)

Error 22 - EC_GETERR - Error during GETVALUECB (RT Uni)

Internal function correctly completed.

An error occurred during the function GETVALUECB

See also

API error messages (Page 1045)

Error 23 - EC_EVMERR - EventMember variables incomplete (RT Uni)

Internal function correctly completed.

Tag configuration for status processing incorrect.

See also

API error messages (Page 1045)

Error 24 - EC_EIDERR - WinCC EV_ID for status processing incorrect (RT Uni)

Internal function correctly completed.

Tag configuration for status processing incorrect.

See also

API error messages (Page 1045)

Error 25 - EC_S7DOSV - function available after S7DOS version xx (RT Uni)

Internal function correctly completed.

Functions were configured (for example, PMC message processing) that are not supported by the installed communication subsystem.

See also

API error messages (Page 1045)

Error 26 - EC_EVSTDE - data not permitted during writing of .EventState (RT Uni)

Internal function correctly completed.

During writing of an .EventState tag incorrect data were transferred.

See also

API error messages (Page 1045)

Error 27 - EC_CONSTR - Configured connection too long as string (RT Uni)

Internal function correctly completed.

The data of the configured connection result in string with invalid length.

See also

API error messages (Page 1045)

Error 28 - EC_PDULEN - Configured data length greater than PDU length (RT Uni)

Internal function correctly completed.

The data length of a configured tag exceeds the maximum PDU length.

The permitted data length for a PDU length of 240 byte is 208 byte. A PDU length of 240 bytes is usual for AS300 and for communication via SAPI-S7.

For a PDU length of 480 bytes, 448 bytes can be transferred. A PDU length of 480 bytes is usual for AS400.

See also

API error messages (Page 1045)

Error 29 - EC_OBJERR - configured data area is not supported (RT Uni)

Internal function correctly completed.

The configured data area (for example area 0x80 I/O) is not supported.

See also

API error messages (Page 1045)

Error 30 - EC_SYSPAR - Error during the setting of the system parameters (RT Uni)

Internal function correctly completed.

An error occurred during the setting of the system parameters.

See also

API error messages (Page 1045)

Error 31 - EC_SYPWRT - error during writing of the system parameters (RT Uni)

Internal function correctly completed.

An error occurred during the writing of the system parameters in the storage file.

See also

API error messages (Page 1045)

Error 32 - EC_NOVARI - Error during GetTagInfo call (RT Uni)

Internal function correctly completed.

An error occurred during call GetTagInfo.

See also

API error messages (Page 1045)

Error 33 - EC_ACKERR - Error during acknowledge via EventState (RT Uni)

Internal function correctly completed.

An error was reported during the sending of jobs to the AS.

See also

API error messages (Page 1045)

Error 34 - EC_LCKERR - Error during Lock/Release via EventState (RT Uni)

Internal function correctly completed.

An error was reported during the sending of jobs to the AS.

See also

API error messages (Page 1045)

Error 35 - EC_AUAOVL - buffer overflow during link-up and update (RT Uni)

Internal function correctly completed.

A buffer overflow due to excess number of jobs occurred during link-up and update.

See also

API error messages (Page 1045)

Error 100 - EC_VATERR - Incorrect type of transferred tag (RT Uni)

Internal function correctly completed.

Transferred tag not from the right type

See also

API error messages (Page 1045)

Communicating with OPC (RT Uni) 17

17.1 OPC UA (RT Uni)

OPC UA (Unified Architecture) is platform-independent technology.

You use the OPC interface to link the devices and applications from various manufacturers in a standardized manner.

See also

Using OPC in WinCC (Page 1055)

Basics of the WinCC OPC UA server (Page 1056)

Compatibility (Page 1057)

Security concept of OPC UA (Page 1058)

Configuring an HMI device as an OPC UA server (Page 1060)

17.2 Using OPC in WinCC (RT Uni)

Configuration

You can use an HMI device as OPC UA server. The OPC server provides process values from the WinCC data management for one or more OPC clients.

Application

An OPC client accesses process values and their properties over the OPC interface. Properties of a process value are, for example, time stamp and quality code.

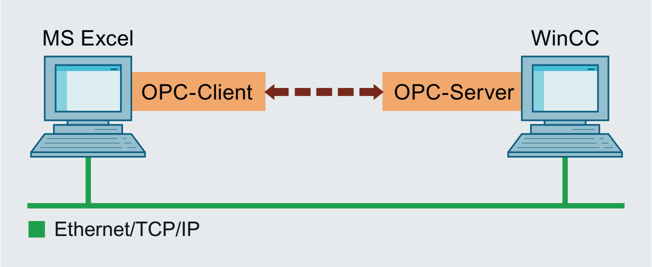
The OPC server supports the following types of access by the OPC client:

- Read/write process values
- Filter process values
- Monitor process values

HMI device as OPC server

An HMI device as OPC server makes the data available to other applications. The applications can run on the same HMI device or on HMI devices in the connected network environment.

The following schematic diagram shows the use of MS Excel as an OPC client that displays process values of the OPC server:



See also

OPC UA (Page 1055)

Basics of the WinCC OPC UA server (Page 1056)

Compatibility (Page 1057)

Security concept of OPC UA (Page 1058)

Configuring an HMI device as an OPC UA server (Page 1060)

OPC server configuration (Page 1061)

17.3 Basics of the WinCC OPC UA server (RT Uni)

Supported certificates

The WinCC OPC UA server is installed as Windows service with WinCC and started automatically.

The WinCC OPC UA server supports only the "UA-TCP UA-SC UA Binary" communication profile and the following server profiles:

- Embedded UA Server Profile
- Standard UA Server Server

For detailed information on these profiles refer to the website of the OPC Foundation (https://opcfoundation.org/).

URL of the WinCC OPC UA server

You access the WinCC OPC UA server via the following URL:

- "opc.tcp://[HostName]:[Port]"

Parameter	Description
HostName	Placeholder for the computer name. Is used automatically
Port	Port number. "4890" is set by default.

The used port number is adjustable.

Discovery server

WinCC supports the "Discovery server". The Discovery server is by default installed on the HMI device as Windows service.

The Discovery server provides information about the OPC UA server to the UA clients registered on the server.

On the WinCC OPC UA server you configure whether and on which Discovery servers the WinCC OPC UA server registers during runtime start.

See also

Configuring an HMI device as an OPC UA server (Page 1060)

OPC UA (Page 1055)

Using OPC in WinCC (Page 1055)

Compatibility (Page 1057)

Security concept of OPC UA (Page 1058)

OPC UA services support (Page 1063)

Permitted data types (OPC) (Page 1064)

OPC server configuration (Page 1061)

17.4 Compatibility (RT Uni)

Support of the mentioned specifications is checked regularly by the "Compliance Test Tool" (CTT) of the OPC Foundation. Interoperability with OPC products of other manufacturers is ensured through the participation in "OPC Interoperability Workshops".

The test results submitted are published on the website of the OPC Foundation. The results can be called up from there using the search term "OPC Self-Certified Products".

See also

OPC UA (Page 1055)

Using OPC in WinCC (Page 1055)

Basics of the WinCC OPC UA server (Page 1056)

Security concept of OPC UA (Page 1058)

Configuring an HMI device as an OPC UA server (Page 1060)

17.5 Security concept of OPC UA (RT Uni)

Introduction

The WinCC OPC UA server uses different communication protocols for encrypted data exchange, e.g. TCP/IP or HTTP. For authorization between WinCC OPC UA server and OPC UA client certificates are exchanged.

Note

Notes on IT security

When configuring the OPC UA server, observe the information on IT Security in the "Readme" section of the TIA Portal online help.

Security concept

The WinCC OPC UA server and each OPC UA client authorize themselves mutually by exchanging certificates.

By default, the WinCC OPC UA server creates during installation a self signed instance certificate.

Certificates represent the authentication mechanism of OPC UA applications. Each application has its own instance certificate and thereby identifies itself within the public key infrastructure.

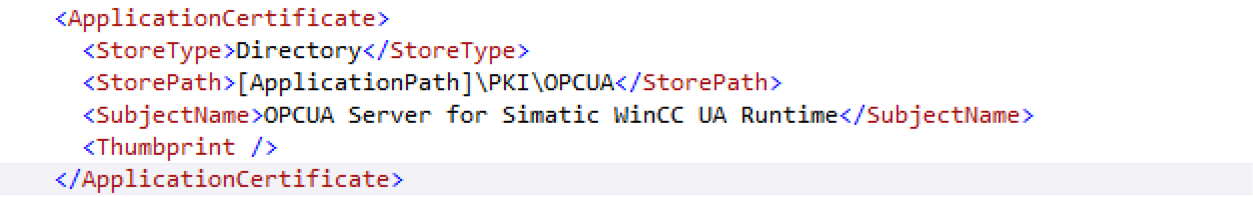
The certificates used by the WinCC OPC UA server are stored via the settings in the configuration file "OpcUaServerWinCCUA.xml".

Instance certificate of WinCC OPC UA server

Each WinCC OPC UA server for secure operation requires a separate instance certificate with a private key. The certificate is only valid on the respective PC and may be used only by the WinCC OPC UA server installed there.

- Instance certificate
  When you install the server, a self-signed instance certificate of the server is created and stored in the certificate folder of the server. You can distribute the instance certificate to other secure OPC clients.
- Private key
  The private key for this certificate is only stored in the certificate folder. Access to the folder with the private key is restricted to the administrator. The storage location of the private key is specifiedi n the configuration file in the XML path "`SecuredApplication/ApplicationtCertificate/`". You can change the storage location, if necessary.
  The following path is set by default:
  `[ApplicationPath]\PKI\WINCC-OPC-UA-Server`
  Storage path for das instance certificate: \WinCCUA\bin\PKI\OPCUA\certs
  Storage location for the private key: \WinCCUA\bin\PKI\OPCUA\private

Example of instance certificate configuration

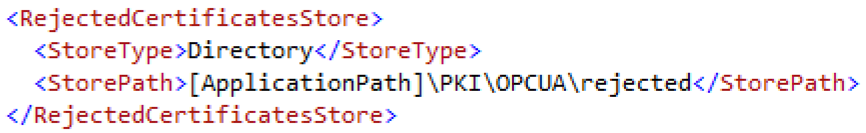


```
<ApplicationCertificate>
  <StoreType>Directory</StoreType>
  <StorePath>[ApplicationPath]\PKI\OPCUA</StorePath>
  <SubjectName>OPCUA Server for Simatic WinCC UA Runtime</SubjectName>
  <Thumbprint />
</ApplicationCertificate>
```

Client certificates not accepted

If a UA client accesses the WinCC OPC UA server without its trusted certificate, the WinCC OPC UA server rejects the secured communication. The server copies the client certificate to the folder for rejected certificates.

You define the storage for rejected certificates with the configuration file of the WinCC OPC UA server, for example,



```
<RejectedCertificatesStore>
  <StoreType>Directory</StoreType>
  <StorePath>[ApplicationPath]\PKI\OPCUA\rejected</StorePath>
</RejectedCertificatesStore>
```

To enable secured communication with this client, you will have to move the rejected certificate to the certificate memory for trusted certificates.

Security settings

The following table lists the security settings supported by the WinCC OPC UA server:

SecurityPolicy	Message Security Mode	
Basic128Rsa15[1]	Sign	SignAndEncrypt
Basic256[2]	Sign	SignAndEncrypt
Basic256Sha256[3]	Sign	SignAndEncrypt

[1] Certificate exchange with depth of encryption of 128 bit.

[2] Certificate exchange with depth of encryption of 256 bit.

Sign: The data packages are signed with the certificates, but not encoded

SignAndEncrypt: The data packages are signed with the certificates and encoded

User identification

For user account identification of an OPC UA client, the WinCC OPC UA server supports the following methods:

- "Anonymous"
  To disable support of anonymous users, delete the entry in the configuration file of the WinCC OPC UA server.
- "User name / Password".
  The respective user account must be known in the user administration of the operating system of the WinCC OPC UA server. In the TIA Portal user administration you assign the global access right "OPCUAServer_GlobalAccess" to the user.

See also

OPC UA (Page 1055)

Using OPC in WinCC (Page 1055)

Basics of the WinCC OPC UA server (Page 1056)

Compatibility (Page 1057)

Configuring an HMI device as an OPC UA server (Page 1060)

OPC UA services support (Page 1063)

Permitted data types (OPC) (Page 1064)

17.6 Configuring an HMI device as an OPC UA server (RT Uni)

Requirement

An HMI device has been created and networked.

Procedure

To configure an HMI device as an OPC UA server, follow these steps:

1. Open the "Runtime settings" of the HMI device in the project tree.
2. Enable "Services > Operate as OPC server".
3. Configure the server settings in the configuration file of the OPC UA server.
4. Save the project.
5. Download the project to the HMI device.
6. Start runtime on the HMI device.

Result

The HMI device can be reached as OPC server in runtime.

See also

OPC UA (Page 1055)

Using OPC in WinCC (Page 1055)

Basics of the WinCC OPC UA server (Page 1056)

Compatibility (Page 1057)

Security concept of OPC UA (Page 1058)

17.7 OPC server configuration (RT Uni)

17.7.1 Structure of the configuration file (RT Uni)

Introduction

You configure the WinCC OPC UA server in the configuration file "OpcUaServerRTIL.xml".

File location

The project-specific configuration file "OpcUaServerRTIL.xml" is stored in the WinCC Unified installation directory under:

"<WinCCUnified>\bin"

`<SecuredApplication>` section

In this section the OPC UA application security is set.

Section	Description
`<Secured Application>`	

`<BaseAddresses>`
`<...></...>`
`</BaseAddresses>`
	Address and port number
	The parameter [`HostName`] is the placeholder for the computer name and is determined during runtime.
	Example:
	`<BaseAdresses>`
	`<ua:String>opc.tcp://`
	`[HostName]:5210</ua:String>`
	`</BaseAdresses>`

Section	Description

`<SecurityProfileUris>`
`<SecurityProfile>`
`<...></...>`
`</SecurityProfile>`
`...`
`</SecurityProfileUris>`
	Security policies
	● You enable the setting with "`true`".
	● You disable the setting with "`false`".
	All active OPC clients with this certificate are thus disabled.
	Example:
	`<SecurityProfile>`
	`<ProfileUri>http://opcfoundation.org/`
	`UA/SecurityPolicy#Basic128Rsa15</ProfileUri>`
	`<Enabled>false</Enabled>`
	`</SecurityProfile>`

`<ApplicationCertificate>`
`<TrustedCertificateStore>`
`<RejectedCertificatesStore>`
`<...>`
	Storage location of the certificates

`</Secured Application>`	

`<ServerConfiguration>` section

The parameters for data transmission and authentication are set in this section.

`<ServerConfiguration>`	

`<SecurityPolicies>`
`<SecurityPolicy>`
`<...></...>`
`</SecurityPolicy>`
`...`
`</SecurityPolicies>`
	Message Security Modes
	To deactivate a security setting, delete the entire entry.
	Example:
	`<SecurityPolicy>`
	`<ProfileUri>http://opcfoundation.org/`
	`UA/SecurityPolicy#Basic128Rsa15`
	`</ProfileUri>`
	`<MessageSecurityModes>SignAndEncrypt`
	`</MessageSecurityModes>`
	`</SecurityPolicy>`

`<UserTokenPolicies>`
`<UserTokenPolicy>`
`<...></...>`
`</UserTokenPolicy>`
`...`
`</UserTokenPolicies>`
	User authentication
	To deactivate a setting, delete the entire entry.
	Example
	`<UserTokenPolicy>`
	`<TokenType>`
	`<!--[User]-->`
	`</TokenType>`
	`</UserTokenPolicy>`
	Use the "Anonymous" setting only for test and diagnostics purposes.

`</ServerConfiguration>`	

See also

Configuring an OPC UA server (Page 1063)

17.7.2 Configuring an OPC UA server (RT Uni)

Opening the configuration file

1. Open the Windows Explorer.
2. Navigate to the directory "<WinCCUnified-InstallationDirectory>\bin".
3. Open the configuration file "OpcUaServerRTIL.xml".

Changing the port number of the WinCC OPC UA server

1. If necessary, change the port number 4890 under `<BaseAdresses>`.
   Do not use a port number that is already assigned to another application.

Configuring a WinCC OPC UA server

1. Configure the OPC UA application security in the section `<SecuredApplication>.`.
2. Configure the data transmission and authentication in the section `<ServerConfiguration>`.

See also

Structure of the configuration file (Page 1061)

17.8 OPC UA services support (RT Uni)

Introduction

The WinCC OPC UA server supports the following described functionality based on the "OPC UA 1.02" specification of the OPC Foundation.

OPC UA Service Sets

The following table shows the supported OPC UA Service Sets:

OPC UA Service Sets	Services
Discovery Service Set	FindServers GetEndpoints
Secure Channel Service Session Service Set	All

OPC UA Service Sets	Services
View Service Set	Browse BrowseNext
Attribute Service Set	Read Write
Subscription Service Set	CreateSubscription SetPublishingMode Publish RePublish DeleteSubscription
MonitoredItem Service Set	CreateMonitoredItems SetMonitoringMode DeleteMonitoredItems

See also

Basics of the WinCC OPC UA server (Page 1056)

Security concept of OPC UA (Page 1058)

17.9 Permitted data types (OPC) (RT Uni)

Permitted data types

The following table lists the data types supported by the WinCC OPC servers:

OPC data type	WinCC data type
BOOLEAN	BOOL
SBYTE	SINT
INT16	INT
INT32	DINT
INT64	LINT
BYTE	USINT
UINT16	UINT
UINT32	UDINT
UINT64	ULINT
FLOAT	REAL
DOUBLE	LREAL
DOUBLE	LTIME*
CDATETIME	DATETIME
INT64	-
BYTE	BYTE
UINT16	WORD
UINT32	DWORD

OPC data type	WinCC data type
UINT64	LWORD
BYTESTRING	RAW
STRING	STRING
STRING	CHAR*
STRUCTURE	STRUCT

* Read-only access

See also

Basics of the WinCC OPC UA server (Page 1056)

Performance features (RT Uni) 18

18.1 General technical data (RT Uni)

18.1.1 Permitted special characters (RT Uni)

Introduction

The following table shows the restrictions that must be observed when allocating names.

Permitted characters

Name	Restriction
Device name	The following constraints apply to the assignment of the device name: • Do not use the following characters: – , ; : ! ? " ' ^ ´ ` ~ _+ = / \ ¦ @ * # $ % & § ° ( ) [ ] { } < > – Spaces • Use upper case only. • The first character must be a letter. • The first 12 characters of the device name must be unique.
Object names	The use of the following special characters is not supported: • Pipe (|) • Slash (/), inverted slash (\ ) • Dot (.) , Comma (,), Semicolon (;), Colon (:) • Quotation marks ("), Apostrophe (') • Angle brackets (<), (>) • Tilde (~), Hash (#), Dollar sign ($), Asterisk (*) • Question mark (?) The use of the following control characters is not supported: • \x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F \x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F When creating scripts, also consider the restrictions relating to special characters of the programming language.

18.2 SIMATIC Unified Comfort Panel (RT Uni)

Unified Comfort Panel

The following tables of performance features help you to assess whether your project conforms to the system limits of a given HMI device.

The specified maximum values are not additive. It cannot be guaranteed that configurations running on the devices at the full system limits will be functional.

Furthermore, the complexity of configuring the screens, such as the number of objects per screen, the number of tag connections, cycle times and scripts, has a significant influence on the open screen times and the performance in runtime.

In addition to the specified limits, allowances must be made for restrictions imposed by configuration memory resources.

Tags

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of tags in the project	8000	8000
Number of elements per array	1600	1600

Alarms

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of alarm classes	32	32
Number of discrete alarms	9000	9000
Number of analog alarms	300	300
Length of an alarm in characters	512	512
Number of alarm texts per interrupt	10	10
Number of process values per alarm	10	10
Number of queued alarm events	750	750

Screens

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of screens	1200	1200
Number of lower-level screen windows	10	10
Number of objects per screen	800	1200
Number of objects from the "Controls" area per screen	40	80
Number of tags per screen	600	800

Parameter sets

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of parameter set types	--	--
Number of parameter set type elements	--	--
Number of parameter sets	--	--
Reserved memory for data records in the internal Flash	--	--

Logs

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of logs	50	50
Number of logging tags, SQLite	5000	5000
Number of logging tags, Microsoft SQL	--	--
Number of entries per log (including all log segments)	500000	500000

Trends

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of trends	600	600
Number of trends per trend view	20	20
Number of trend areas per trend view	2	5

Text lists and graphics lists

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of graphics lists	750	750
Number of text lists	750	750
Number of entries per text or graphics list	750	750
Number of graphic objects	6000	6000
Number of text elements	60000	60000

Scripts

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of scripts	600	600
Number of functions per function list	25	25

Scheduler

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of tasks, time- or event-triggered	70	70

Communication

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of S7 connections [1]	16	16
1) SIMATIC NET is required to use more than 11 connections.		

Reporting

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of templates	--	--
Number of report tasks	--	--
Number of report tasks started at the same time	--	--
Number of reports executed at the same time	--	--

OPC UA

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of connected OPC UA clients	3	3

Languages

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of runtime languages	32	32

User administration

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of roles	50	50
Number of predefined function rights	20	20
Number of users	200	200

Project

	Unified Comfort 7-12"	Unified Comfort 15-22"
Size of the project files on the device	< 100 MB	< 100 MB

18.3 SIMATIC Unified PC

Unified PC based

The following tables of performance features help you to assess whether your project conforms to the system limits of a given HMI device.

The specified maximum values are not additive. It cannot be guaranteed that configurations running on the devices at the full system limits will be functional.

Furthermore, the complexity of configuring the screens, such as the number of objects per screen, the number of tag connections, cycle times and scripts, has a significant influence on the open screen times and the performance in runtime.

In addition to the specified limits, allowances must be made for restrictions imposed by configuration memory resources.

Tags

	SIMATIC Unified PC
Number of PowerTags	600000 (depends on the license)
Number of internal tags	200000
Number of elements per array	2000

Alarms

	SIMATIC Unified PC
Number of alarm classes	32
Number of discrete alarms	200000
Number of analog alarms	10000
Length of an alarm in characters	512
Number of alarm texts per interrupt	10
Number of process values per alarm	10
Number of alarms for every second (continuous load)	20
Number of queued alarm events	unlimited
Number of alarms for every 10 seconds (alarm burst)	8000

Screens

	SIMATIC Unified PC
Maximum size in the engineering system	10,860 * 6,110 pixels
Maximum size in Runtime	4,096 * 2,160 pixels
Number of screens	2000
Number of lower-level screen windows	unlimited
Number of objects per screen	1500
Number of tags per screen	1000

Parameter sets

	SIMATIC Unified PC
Number of parameter set types	750
Number of parameter set type elements	1000
Number of parameter sets	2000

Logs

	SIMATIC Unified PC
Number of logs	1000
Number of logging tags, SQLite	5000
Number of logging tags, Microsoft SQL	Maximum number of PowerTags
Number of entries per log (including all log segments)	500000
Number of entries per second	30000

Trends

	SIMATIC Unified PC
Number of trends	1000
Number of trends per trend view	60
Number of trend areas per trend view	5

Text lists and graphics lists

	SIMATIC Unified PC
Number of graphics lists	1000
Number of text lists	2000
Number of entries per text or graphics list	3500
Number of graphic objects	unlimited
Number of text elements	unlimited

Scripts

	SIMATIC Unified PC
Number of scripts	unlimited
Number of functions per function list	50

Scheduler

	Unified PC based
Number of tasks, time- or event-triggered	200

Communication

	SIMATIC Unified PC
Number of S7 connections [1)]	128
1) SIMATIC NET is required to use more than 11 connections.	

Reporting

	SIMATIC Unified PC
Number of templates	500
Number of report tasks	500
Number of report tasks started at the same time	20
Number of reports executed at the same time	5

OPC UA

	Unified PC based
Number of connected OPC UA clients	10

Languages

	SIMATIC Unified PC
Number of runtime languages	32

User administration

	SIMATIC Unified PC
Number of roles	50
Number of predefined function rights	20
Number of users	200

Plant objects

	SIMATIC Unified PC
Number of plant object types	400
Number of plant object instances	65000
Number of hierarchy levels	unlimited

Runtime API (RT Uni) 19

19.1 Basics (RT Uni)

Task of the runtime API

Runtime API describes the open programming interface of WinCC Unified Scada RT. With the Runtime API, you can use the internal functions of WinCC in your own applications. With an ODK client, you can read out all the objects of the Runtime system and change their Runtime attributes, for example, for tags or alarms.

The ODK is optimized for the processing of mass data when special objects are used, for example the reading or writing of 1000 tags in one pass.

Note

Siemens is not liable for and does guarantee the compatibility of the data and information transported via the API interfaces with third-party software.

We expressly point out that improper use of the API interface can result in data loss or production downtimes.

Requirement

- Programming environment is installed, e.g. MS Visual Studio
- WinCC Runtime Unified Scada RT is installed.

Application of C++ and .NET

The Runtime API makes all the interfaces available for the access to the runtime system in the languages C++ and C#.

Name-based addressing of objects

The objects of the Runtime system are addressed by their name and the full name path.

The name path of objects consists of several components and has the following syntax:

```
[SystemName::][ObjectName][.ElementPath][:SubElementName]
```

- `SystemName`
  Name of a Runtime systems (optional)
  If the "SystemName" is omitted, the object is searched for on the local runtime system.
- `ObjectName`
  Name of a tag or a structure

- `ElementPath`
  Element of a structure
- `SubElementName`
  Subelement of an object, e.g. alarm or logging tag of a tag.

Examples for access to different object types:

- Simple tag: `MyRTSystem::MySimpleTag`
- Structure tag: `MyRTSystem::Motor.Temperature`
- Alarm of a simple tag: `MyDiscreteTag:MyDiscreteAlarm`
- Alarm of a structure tag: `Motor.Temperature:MyAnalogAlarm`
- Logging tag: `MySimpleTag:MyLoggingTag`
- Connection: `MyRTSystem::MyHmiConnection`

19.2 Changes to the API (RT Uni)

Changes compared to previous version

The following changes were made at the following interfaces:

Version 15 to 15.1

Context	Version 15	Version 15.1	Language	Description
Assemblies	Siemens.Runtime.HmiIL.Interfaces.dll Siemens.Runtime.HmiIL.Alarms.dll Siemens.Runtime.HmiIL.dll Siemens.Runtime.HmiIL.Tags.dll Siemens.Runtime.HmiIL.PlantModel.dll Siemens.Runtime.HmiIL.TagLogging.dll Siemens.Runtime.HmiIL.AlarmLogging.dll Siemens.Runtime.HmiIL.Connections.dll Siemens.Runtime.HmiIL.Umc.dll	Siemens.Runtime.HmiUnified.Interfaces.dll Siemens.Runtime.HmiUnified.dll Siemens.Runtime.HmiUnified.dll Siemens.Runtime.HmiUnified.Tags.dll Siemens.Runtime.HmiUnified.PlantModel.dll Siemens.Runtime.HmiUnified.TagLogging.dll Siemens.Runtime.HmiUnified.AlarmLogging.dll Siemens.Runtime.HmiUnified.Connections.dll Siemens.Runtime.HmiUnified.Umc.dll	C#	Assemblies renamed.
	HmiILRt.lib HmiILRt.dll HmiILRtAlarms.dll HmiILRtTags.dll HmiILRtPlantModel.dll HmiILRtTagLogging.dll HmiILRtAlarmLogging.dll HmiILRtConnections.dll HmiILRtUmc.dll	HmiUnifiedRt.lib HmiUnifiedRt.dll HmiUnifiedRtAlarms.dll HmiUnifiedRtTags.dll HmiUnifiedRtPlantModel.dll HmiUnifiedRtTagLogging.dll HmiUnifiedRtAlarmLogging.dll HmiUnifiedRtConnections.dll HmiUnifiedRtUmc.dll	C++	
		Siemens.Runtime.HmiIL.Pma.Interfaces.dll	C#	New: Assemblies for the PI Option Performance Insight
		OneOeeODK.dll	C++	

Context	Version 15	Version 15.1	Language	Description
Namespaces	Siemens.Runtime.HmiIL.Interfaces	Siemens.Runtime.HmiUnified	C#	Namespaces renamed.
	Siemens::Runtime::HmiIL::Cpp Siemens::Runtime::HmiIL::Common::Cpp Siemens::Runtime::HmiIL::Alarms::Cpp Siemens::Runtime::HmiIL::Logging::Cpp Siemens::Runtime::HmiIL::Connection::Cpp Siemens::Runtime::HmiIL::PlantModel::Cpp Siemens::Runtime::HmiIL::Tags::Cpp Siemens::Runtime::HmiIL::Umc::Cpp	Siemens::Runtime::HmiUnified::Rt Siemens::Runtime::HmiUnified::Common Siemens::Runtime::HmiUnified::Alarms Siemens::Runtime::HmiUnified::Logging Siemens::Runtime::HmiUnified::Connection Siemens::Runtime::HmiUnified::PlantModel Siemens::Runtime::HmiUnified::Tags Siemens::Runtime::HmiUnified::Umc	C++	
		Siemens::Runtime::HmiUnified::Pma	C#	New: Namespaces for the PI Option Performance Insight
		Siemens::Runtime::HmiUnified::PMA	C++	
BrowseLoggingTag		BrowseLoggingTag.cs has been removed und replaced by ILoggingTags	C#	BrowseLoggingTag has been replaced.
		IOdkRtBrowseLoggingTag.h has been removed und replaced by ILoggingTags	C++	
ILoggedTagValue	Int32 Quality { get; set; }	UInt16 Quality { get; set; }	C#	Data type changed
	GetQuality(OUT int32_t* value)	GetQuality(OUT uint16_t* value)	C++	
IAlarmResult	UInt64 Id { get; }	UInt32 Id { get; }	C#	Data type changed
	GetId(OUT uint64_t * value)	GetId(OUT uint32_t * value)	C++	
	Byte State { get; }	HmiAlarmState State { get; }	C#	Enumerations introduced for State und SourceType.
	UInt16 SourceType { get; }	HmiAlarmSourceType SourceType { get; }		
ILoggedAlarmResult	Byte State { get; }	HmiAlarmState State { get; }	C#	Enumerations introduced for State und SourceType.
	Byte SourceType { get; }	HmiAlarmSourceType SourceType { get; }		
Enum.cs		Added to the enumerations "None = 0"	C#	All enumerations begin with 0.
IPlantObject		GetLoggingTags method has been removed	C#	GetLoggingTags has been removed.
		GetLoggingTags method has been removed	C++	

Context	Version 15	Version 15.1	Language	Description
IPlantObject-PropertySet	void Write();	IList<IErrorResult> Write();	C#	Supplies a list with instances of "IErrorResult" instances.
	Write();	Write(HmiUnified::Rt::IErrorResultEnumerator** ppEnumerator);	C++	Supplies an "IErrorResultEnumerator" instance.
IPlantModel		GetPlantObjectByPath has been removed	C#	Obsolete method GetPlantObjectByPath has been removed. Can be replaced by GetPlantObject.
			C++	
UMC.cs		Added to the enumeration ServerStatus "None = 0"	C#	All enumerations begin with 0.

19.3 Creating a minimal ODK client (RT Uni)

Introduction

An ODK client uses the ODK API to access objects of the WinCC Unified system.

In the following, an ODK client is created for use of the Runtime API in the C# and C++ languages.

The programs only contain the most needed components of a simple client. They form the framework for all the subsequent runtime code examples in this documentation. See also section AUTOHOTSPOT.

Note

You will find additional programming examples on the installation medium in the file "Support\Openness\Siemens.Unified.Openness_SDK_<version number>.zip" in the subdirectory"ODK\samples".

Requirement

- Development environment is installed.
- The ODK SDK was extracted locally on your computer. You will find the ODK SDK in the "Support\Openness" folder on the WinCC Unified DVD in the file "Siemens.Unified.Openness_SDK_<version number>.zip".

Note

If you create a C++ ODK client, you must set the system tag "PATH=C:\Program Files\Siemens\Automation\WinCCUnified\bin" and perform a restart.

Procedure C# client

1. Create a new .NET project in the development environment.
2. Carry out the following project settings:
   - Target framework is .NET 4.6.
   - ODK client is "Release" version for the x64 platform.
3. Create references to the following assembly: Siemens.Runtime.HmiUnified.Interfaces.dll (Copy Local = False)
   You will find the assembly in the local folder to which you have extracted Openness_SDK.zip, in the subfolder "ODK\bin".
4. Create a program with the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Siemens.Runtime.HmiUnified;

namespace Siemens.Runtime.HmiUnified.TestClient
{
    class Program
    {

        static void Main(string[] args)
        {
            try
            {
                using (IRuntime runtime = Runtime.Connect())
                {
                    //do runtime operations
                }
            }
            catch (Exception ex)
            {
                System.Console.WriteLine(string.Format("Exception occured
{0}", ex.Message));
            }
        }
    }
}
```

Procedure C++ client

1. Create a new C++ project in the development environment.
2. Set the following include directories for required headers:
   - <Local folder to which you have extracted the ODK SDK>\ODK\include\ODK
   - <Local folder to which you have extracted the ODK SDK>\ODK\include\ODK\include\CF

3. Create references to the following libraries in "<Local folder to which you have extracted the ODK SDK>\ODK\include\ODK\lib":
   - HmiUnifiedRt.lib
   - CfCore.lib
4. Create a reference to the following directory as "Additional Library Directory":
   - <Local folder to which you have extracted the ODK SDK>\ODK\lib
5. Create a program with the following code:

```
#include <CfTL>

#include "IOdkRt.h"
#include "IOdkRtTag.h"
#include "IOdkRtTagLogging.h"
#include "IOdkRtAlarm.h"
#include "IOdkRtAlarmLogging.h"
#include "IOdkRtCpm.h"
#include "IOdkRtConnection.h"
#include "IOdkRtUmc.h"

#include <stdio.h>
#include <tchar.h>
#include <iostream>

using namespace Siemens::Runtime::HmiUnified;
using namespace Siemens::Runtime::HmiUnified::Common;
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    CCfString projectName = L"";
    IRuntimePtr pRuntime;

    if(CF_SUCCEEDED(Connect(projectName, &pRuntime)
    {
        // do runtime operations here
    }
    return 0;
}
```

Result

The program core of an ODK client is created.

You can complete the program with the fragments from the following code examples for ODK-API.

19.4 Authorizing users (RT Uni)

Introduction

Before the ODK client can be used, all users that run the ODK client must be authorized.

Note

Only authenticated users can access the data of a project over the Runtime API with an ODK client.

Requirement

WinCC Unified RT setup has been carried out completely on the Runtime computer.

Procedure

Add the user who corresponds to the logged-on Windows user on the Runtime computer to the Windows user group "SIMATIC HMI" in the Windows user administration.

Result

When the ODK client is connected to the Runtime system, the logged-on Windows user is authenticated via the user administration of the Runtime computer.

If one of the checks fails, the ODK client does not establish a connection (error: "Authentication error" or "User has no access right").

19.5 Startup and shutdown behavior of an ODK application (RT Uni)

19.5.1 Autostart of an ODK application (RT Uni)

You have the possibility of starting ODK applications automatically on start-up of the device.

Requirement

Runtime is configured in such a way that it automatically started on start-up of the device without a user having to be logged on. (Default setting)

Procedure

In the Windows Task Scheduler, create a task which starts the ODK application on start-up of the device.

Note

User Service Mode

To use the ODK application in Service Mode, configure the security options in the dialog "Create task" in such a way that a user does not have to be logged on for starting the task.

Under Windows 10 activate the option "Run whether user is logged on or not".

Result

The "Connect" method of IRuntime wait for a maximum of ten minutes after the start of the ODK application until the Runtime has started up.

19.5.2 Shutdown behavior (RT Uni)

You have the option to be notified by the system on shutdown of Runtime, for example, to start cleanup work on the client.

For this purpose, subscribe the system tag "@SystemActivationState" for monitoring. "@SystemActivationState" signals whether Runtime is active and can have the following values:

- System startup in progress (1)
- System started (activated) (2)
- System stopped (3)
- System shutdown in progress (4)
- System restart in progress (5)

Value 4 is the trigger to start cleanup work.

Note

Interface calls on shutdown

Do not call any functions of the ODK interfaces while the system is shut down.

19.5.3 Restart behavior (RT Uni)

Tags subscribed for monitoring

After Runtime is restarted, you can continue to use the existing subscriptions.

19.6 Syntax of the alarm filter (RT Uni)

With an AlarmSubscription, a filter can be transferred so that not all active alarms of the alarm system are notified, but only those which match the filter. The filter syntax is based on SQL syntax. However, only the WHERE instruction is relevant. The keyword "WHERE" must be omitted.

Operators

The following operators can be used in the filter string of the alarm filter:

Operator	Description	Example
`=`	equal to	`Name = 'Recipe246'`
`<>`	not equal	`Value <> 0.0`
`>`	greater than	`Value > 25.0`
`<`	less than	`Value < 75.0`
`>=`	greater than or equal to	`Value >= 25.0`
`<=`	less than or equal to	`Value <= 75.0`
`OR, ||`	logical OR	`State = 1 OR State = 3`
`AND, &&`	logical AND	`Value >= 25.0 AND Value <= 75.0`
`BETWEEN`	within a range	`Value BETWEEN 25.0 AND 75.0`
`NOT BETWEEN`	outside a range	`Value NOT BETWEEN 25.0 AND 75.0`
`LIKE string`	corresponds to the string *string*	`Name LIKE 'Motor*'`
`NOT LIKE string`	does not correspond to the string *string*	`Name NOT LIKE 'Valve*'`
`IN (v1, v2, …)`	corresponds to one or more values	`State IN (1, 4, 7)`
`NOT IN (v1, v2, …)`	does not correspond to one or more values	`State NOT IN (0, 2, 3, 5, 6)`
`(…)`	brackets expressions	`Value <= 75.0 AND (State = 1 OR State = 3)`

Precedence of the operators:

Rank	Operators
1	• Relational operators: =, <>, >, <, >=, <= • LIKE • IN • BETWEEN
2	NOT
3	AND, &&
4	OR, ||
5	

Permitted wildcards:

Wildcard	Description	Example

*	Replaces 0 to more characters	`Name LIKE 'Motor*'`
		`Reference = <1.*.15>1`

?	Replaces 1 character	`Name = 'Recipe?'`

19.7 Locale IDs of the supported languages (RT Uni)

At the AlarmSubscription, there is a Language property which defines the language of the alarm filter and the language of the alarm texts. In this case, a locale ID from the table below must be entered.

The following table contains the Microsoft locale IDs of the languages supported in the TIA Portal:

Language	Country/Region	Locale ID
Afrikaans	South Africa	1078
Albanian	Albania	1052
Armenian	Armenia	1067
Azerbaijani (Cyrillic)	Azerbaijan	2092
Azerbaijani (Latin)	Azerbaijan	1068
Basque	Basque country	1069
Belarusian	Belarus	1059
Bulgarian	Bulgaria	1026
Chinese	Chinese (Hong Kong S.A.R.)	3076
Chinese	Chinese (Macao S.A.R.)	5124
Chinese	Chinese (Singapore)	4100
Chinese	Chinese (Taiwan)	1028
Chinese	Chinese (PR China)	2052
Danish	Denmark	1030
German	Germany	1031
German	Liechtenstein	5127
German	Luxembourg	4103
German	Austria	3079
German	Switzerland	2055
English	Australia	3081
English	Belize	10249
English	United Kingdom	2057
English	Ireland	6153
English	Jamaica	8201
English	Canada	4105
English	Caribbean	9225
English	New Zealand	5129
English	Philippines	13321

Language	Country/Region	Locale ID
English	Zimbabwe	12297
English	South Africa	7177
English	Trinidad and Tobago	11273
English	USA	1033
Estonian	Estonia	1061
Faroese	Faroe Islands	1080
Finnish	Finland	1035
French	Belgium	2060
French	France	1036
French	Canada	3084
French	Luxembourg	5132
French	Monaco	6156
French	Switzerland	4108
Galician	Galicia	1110
Georgian	Georgia	1079
Greek	Greece	1032
Hindi	India	1081
Indonesian	Indonesia	1057
Icelandic	Iceland	1039
Italian	Italy	1040
Italian	Switzerland	2064
Japanese	Japan	1041
Kazakh	Kazakhstan	1087
Catalan	Catalonia	1027
Kyrgyz	Kyrgyzstan	1088
Konkani	India	1111
Korean	Korea	1042
Croatian	Croatia	1050
Latvian	Latvia	1062
Malay	Brunei Darussalam	2110
Malay	Malaysia	1086
Macedonian	Macedonia, FYRM	1071
Mongolian (Cyrillic)	Mongolia	1104
Dutch	Belgium	2067
Dutch	Netherlands	1043
Norwegian (Bokmal)	Norway	1044
Norwegian (Nynorsk)	Norway	2068
Polish	Poland	1045
Portuguese	Brazil	1046
Portuguese	Portugal	2070
Romanian	Romania	1048
Russian	Russia	1049
Sanskrit	India	1103

Language	Country/Region	Locale ID
Swedish	Finland	2077
Swedish	Sweden	1053
Serbian (Cyrillic)	Serbia and Montenegro (formerly)	3098
Serbian (Latin)	Serbia and Montenegro (formerly)	2074
Slovakian	Slovakia	1051
Slovenian	Slovenia	1060
Spanish	Argentina	11274
Spanish	Bolivia	16394
Spanish	Chile	13322
Spanish	Costa Rica	5130
Spanish	Dominican Republic	7178
Spanish	Ecuador	12298
Spanish	El Salvador	17418
Spanish	Guatemala	4106
Spanish	Honduras	18442
Spanish	Columbia	9226
Spanish	Mexico	2058
Spanish	Nicaragua	19466
Spanish	Panama	6154
Spanish	Paraguay	15370
Spanish	Peru	10250
Spanish	Puerto Rico	20490
Spanish	Spain	3082
Spanish	Uruguay	14346
Spanish	Venezuela	8202
Swahili	Kenya	1089
Tatar	Russia	1092
Thai	Thailand	1054
Czech	Czech Republic	1029
Turkish	Turkey	1055
Ukrainian	Ukraine	1058
Hungarian	Hungary	1038
Uzbek (Cyrillic)	Uzbekistan	2115
Uzbek (Latin)	Uzbekistan	1091
Vietnamese	Vietnam	1066

19.8 Code samples (RT Uni)

ODK is supplied with code samples for using the Runtime interfaces. Open the local folder to which you have extracted the file "Support\Openness\Siemens.Unified.Openness_SDK_<version number>.zip".

You will find the code samples in the subfolder "\ODK\samples".

Using the code samples in the help

To reduce the complexity of the code samples and enable better readability, the examples in the help deliberately exclude troubleshooting and freeing up memory.

The application programmer must add these elements during programming.

Constructs affected under C#

- Exception handling with try…catch…finally
```
try
{
   …
}
catch (Exception ex)
{
   …
}
finally
{
}
```
- Freeing up memory with Dispose or using (…)
  See also section AUTOHOTSPOT.

For a description of how to evaluate ODK-specific errors, see section AUTOHOTSPOT.

Constructs affected under C++

- Freeing up allocated memory
- Null pointer check: `if (pObject != nullptr) {…}`
- Error code check: `if (CF_SUCCEEDED(errorCode) {…}`

For a description of how to evaluate ODK-specific errors, see section AUTOHOTSPOT.

19.9 Description of the C# interfaces (RT Uni)

19.9.1 Releasing objects (RT Uni)

Creating objects with GetObject

In .NET-ODK, you create the objects with the "GetObject" method, for example:

```
ITagSet odkTagSet = runtime.GetObject<ITagSet>();
```

Memory is created internally for the object. In .NET, the Garbage Collector automatically releases the memory when an object is no longer needed. However, the memory is released at an indefinite time.

Note

The indefinite execution of the Garbage Collector may cause the memory to increase and appear as if it is not being released again. The real reason for this is that the Garbage Collector has not yet started!

Releasing objects created with GetObject

The ODK client should release the memory of objects created using the "GetObject" method as soon as the object is no longer needed.

The following cases must be distinguished here:

- Using the objects by synchronous ODK methods
- Using the objects by asynchronous ODK methods

Example when using synchronous ODK methods

When releasing objects used by synchronous ODK methods, use the keyword `using`:

Copy code

```
try
{
    using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
    {
        IProcessValue value = myTag.Read(HmiReadType.Cache);   // Reads synchronous
    }
}
catch (OdkException ex)
{
    System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
}
```

Example when using asynchronous ODK methods

When you release objects that are used by asynchronous ODK methods, you can use the "Dispose" method. This can be called in the callback method:

Copy code

```
try
{
    ITagSet odkTagSet = runtime.GetObject<ITagSet>();
    odkTagSet.Add(new string[] { "Tag1", "Tag2" });

    // Assign callback function
    odkTagSet.OnReadResult += odkTagSet_OnReadResult;
    odkTagSet.ReadAsync();// Reads asynchronous
}
catch (OdkException ex)
{
    System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
}

void odkTagSet_OnReadResult(ITagSet sender, IList<IProcessValue> values, bool completed)
{
    try
    {
        …
    }
    catch (OdkException ex)
    {
        …
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose(); // Release memory
        }
    }
}
```

19.9.2 Interfaces of the Runtime environment (RT Uni)

19.9.2.1 IRuntime (RT Uni)

Description

The C# interface "IRuntime" specifies properties and methods for handling the Runtime system. The "Connect" method of the "Runtime" class is called to establish the connection to the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members "Runtime"

The class implements the following method:

"Connect" method

Connect to a Runtime project.

- Connect to locally run Runtime project. Logged-on Windows user is authenticated.
  `IRuntime Connect()`
- Connect to locally run Runtime project. The logged-on Windows user is not authenticated; instead, the user is specified as a parameter.

  Note

  Can only be used in a future version!

  ```
  IRuntime Connect(string userName, string password)
  ```

  – `user`
    User name

  – `password`
    Password
- Connect to a specific Runtime project. Logged-on Windows user is authenticated.

  Note

  Can only be used in a future version!

  ```
  IRuntime Connect(string value)
  value
  ```
  Name of a Runtime project
- Connect to a specific Runtime project. The logged-on Windows user is not authenticated; instead, the user is specified as a parameter.

  Note

  Can only be used in a future version!

  ```
  IRuntime Connect(string value, string userName, string password)
  ```

  – `value`
    Name of a Runtime project

  – `user`
    User name

  – `password`
    Password

"Dispose" method

Enable Runtime system with all resources.

```
void Dispose()
```

Members "IRuntime"

The following properties and methods are specified in the interface:

"ProjectName" property

Name of the current project

Note

Can only be used in a future version!

```
string ProjectName { get; }
```

"UserName" property

Name of the logged-on user

```
string UserName { get; }
```

"Product" property

Return version information and installed options of the Runtime system as "IProduct" object.

```
IProduct Product { get; }
```

"GetObject" method

Create a new instance of an object type T in a project.

```
T GetObject<T>(params object[] parameters)
```

The object type T adopts the following values:

- `ITag`, `ITagSet` or `ITagSetQCD`
  Access to tags
- `IAlarm`, `IAlarmSet` or `IAlarmSubscription`
  Access to alarm logging
- `ILoggedTag` or `ILoggedTagSet`
  Access to logging tags
- `IAlsrmLogging` or `IAlarmLoggingSubscription`
  Access to logged alarms
- `IUserManagement`
  Access to user management
- `IConnection` or `IConnectionSet`
  Access to connections

`parameters`
Optional: A name or array with names of objects of the respective object type

"GetOption" method

Return an installed option of the Runtime system as "IOption" object using the name.

```
IOption GetOption(string optionName)
```

`optionName`
Name of the installed option

Example

Initialize the ODK and establish a connection to the active project of the Runtime system.

Copy code

```
public static IRuntime runtime = null;

public void Connect()
{
    try
    {
        // Connect to running project
        runtime = Siemens.Runtime.HmiUnified.Runtime.Connect();
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
```

Initialize an "IProduct" object and output the technical product version of the Runtime system:

```
public void GetVersionInfo(IRuntime runtime)
{
    IProduct product = runtime.Product;
    IVersionInfo version = product.Version;
    System.Console.WriteLine(string.Format("Product version: {0}.{1}.{2}.{3}",
version.Major, version.Minor, version.ServicePack, version.Update));

    ...

}
```

Access a tag with the name "Tag1":

```
public void ReadSingleTagSync()
{
    try
    {
        using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
        {
        ...//further tag processing
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
```

See also

IProduct (Page 1094)

IErrorResult (Page 1098)

19.9.2.2 IProduct (RT Uni)

Description

The C# interface "IProduct" specifies properties for handling product information of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties are specified in the interface:

"Options" property

Return installed options of the Runtime system as a list of "IOption" objects.

```
IList<IOption> Options { get; }
```

"Version" property

Return version structure of the Runtime system as "IVersionInfo" object.

```
IVersionInfo Version { get; }
```

Example

The version of the "IProduct" object is read out and iterated via the installed "IOption" objects:

Copy code

```
public void GetVersionInfo(IRuntime runtime)
{
    try
    {
        IProduct product = runtime.Product;
        IVersionInfo version = product.Version;

        if (product.Options.Count > 0)
        {
            foreach (IOption op in product.Options)
            {
                IVersionInfo opVersion = op.Version;

                // Iterate through options and get version
                ...
            }
        }
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(string.Format("Exception occured {0}", ex.Message));
    }
}
```

See also

IRuntime (Page 1090)

IOption (Page 1095)

IVersionInfo (Page 1097)

19.9.2.3 IOption (RT Uni)

Description

The C# interface "IOption" specifies properties and methods for handling installed product options of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of an installed option of the Runtime system

```
String Name { get; }
```

"Version" property

Return version structure of an installed option of the Runtime system as "IVersionInfo" object.

```
IVersionInfo Version { get; }
```

"GetObject" method

Create new instance of an object type T of the option.

```
T GetObject<T>(params object[] parameters)
```

- `T`
  The value defines a specific object type of the option.
- `parameters`
  Optional: A parameter or array with parameters for the object type of the option

Example

Instantiate and use installed options with name "MyOptionName":

Copy code

```
public void GetOptionObject()
{
    //load option component by name
    IMyOption rtOption = (IMyOption)runtime.GetOption("MyOptionName");

    //create a instance of the option object IMyOptionObject
    IMyOptionObject optionObject = rtOption.GetObject<IMyOptionObject>();
    try
    {
        string strMethod = optionObject.MyMethod();
        string strProperty = optionObject.MyProperty;
    }
    catch (OdkException ex)
    {
        //It is an option error?
        if (ex.ErrorSubCategory == MyOptionConstants.MYOPTION_ERRORCATEGORY)
        {
            //Handle option specific error
            if (ex.ErrorCode == (uint)MyErrorCodes.E_UNKNOWN_NAME)
            {
                //get error description
                string errorDescription = ex.Message;
            }
        }
    }
}
```

See also

IProduct (Page 1094)

IVersionInfo (Page 1097)

19.9.2.4 IVersionInfo (RT Uni)

Description

The C# interface "IVersionInfo" specifies properties for handling version information of the Runtime system.

Members

The following properties are specified in the interface:

"Major" property

Main version of the Runtime system or of an installed option

```
uint16 Major { get; }
```

"Minor" property

Minor version of the Runtime system or of an installed option

```
uint16 Minor { get; }
```

"ServicePack" property

Service pack of the Runtime system or of an installed option

```
uint16 ServicePack { get; }
```

"Update" property

Update version of the Runtime system or of an installed option

```
uint16 Update { get; }
```

Example

Information about the installed "IOption" options of the runtime system is output:

Copy code

```
public void GetVersionInfo(IRuntime runtime)
{
    try
    {
        IProduct product = runtime.Product;
        IVersionInfo version = product.Version;
        System.Console.WriteLine(string.Format("Product version: {0}.{1}.{2}.{3}",
version.Major, version.Minor, version.ServicePack, version.Update));

        if (product.Options.Count > 0)
        {
            foreach (IOption op in product.Options)
            {
                IVersionInfo opVersion = op.Version;

                // Iterate through options and get version
                System.Console.WriteLine(string.Format("Option name: {0}", op.Name));
                System.Console.WriteLine(string.Format("Option version: {0}.{1}.{2}.{3}",
opVersion.Major, opVersion.Minor, opVersion.ServicePack, opVersion.Update));
            }
        }
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(string.Format("Exception occured {0}", ex.Message));
    }
}
```

See also

IProduct (Page 1094)

IOption (Page 1095)

19.9.3 Error-handling interfaces (RT Uni)

19.9.3.1 IErrorResult (RT Uni)

Description

The C# interface "IErrorResult" specifies properties of error results in runtime.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties are specified in the interface:

"Error" property

Error code of an error

```
int32 Error { get; }
```

"Name" property

Name of the associated object of the data source

```
string Name { get; }
```

Example

Error output when writing a TagSet:

Copy code

```
public void WritePartlyNotExistingTagSetSync()
{
    try
    {
        using (ITagSet odkTagSet = runtime.GetObject<ITagSet>())
        {
            odkTagSet.Add("Tag1", 1);
            odkTagSet.Add("Tag2", 2);
            odkTagSet.Add("NotExistingTag1", 1);
            odkTagSet.Add("NotExistingTag2", 2);

            IList<IErrorResult> writeResult = odkTagSet.Write();

            foreach (var result in writeResult)
            {
                if (result.Error != 0)
                {
                    System.Console.WriteLine(string.Format("Write tag '{0}' failed, error 
code {1}", result.Name, result.Error));
                }
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
```

See also

IRuntime (Page 1090)

IErrorInfo (Page 1100)

19.9.3.2 IErrorInfo (RT Uni)

Description

The C# interface "IErrorInfo" specifies methods and properties for handling error codes.

Members

The following properties and methods are specified in the interface:

"Error" property

Error code of an error

```
int32 Error { get; }
```

"GetErrorDescription" method

Output an error description for the error code.

```
string GetErrorDescription(uint32 Error)
```

`Error`
Error code that is passed by the ODK client.

See also

IErrorResult (Page 1098)

19.9.3.3 OdkException (RT Uni)

Description

In the case of exceptions, the ODK triggers an OdkException in the .Net environment. The OdkException can be caught by try-catch blocks and evaluated.

The "OdkException" class inherits all properties and methods of the .NET class "Exception".

Members "OdkException"

The following objects and methods are also implemented in the "OdkException" class for all properties and methods of the .NET class "Exception".

"OdkException" method

- Trigger exception without message.
  `OdkException()`
- Trigger exception with message and error description.
  `OdkException(string message)`
  `message`
  Description of the error

- Trigger exception with message and error description. Trigger additional exception with reference to triggering exception.
  `OdkException(string message, Exception innerException)`
  - `message`
    Description of the error
  - `innerException`
    Triggering exception
- Trigger exception with serialized data.
  `OdkException(SerializationInfo info, StreamingContext context)`
  - `info`
    Serialized data of the exception
  - `context`
    Describes the origin or target of the serialized data.

"ErrorCode" property

Error code of the exception

```
UInt32 ErrorCode { get; }
```

"ErrorSubCategory" property

Subcategory of an error code

```
UInt32 ErrorSubCategory { get; }
```

Example

Exception handling using the example of optional components:

Copy code

```
public void GetOptionObject()
{
    //load option component by name
    IMyOption rtOption = (IMyOption)runtime.GetOption("MyOptionName");
    //create a instance of the option object IMyOptionObject
    IMyOptionObject optionObject = rtOption.GetObject<IMyOptionObject>();
    try
    {
        string strMethod = optionObject.MyMethod();
        string strProperty = optionObject.MyProperty;
    }
    catch (OdkException ex)
    {
        //It is an option error?
        if (ex.ErrorSubCategory == MyOptionConstants.MYOPTION_ERRORCATEGORY)
        {
            //Handle option specific error
            if (ex.ErrorCode == (uint)MyErrorCodes.E_UNKNOWN_NAME)
            {
                //get error description
                string errorDescription = ex.Message;
            }
        }
    }
}
```

19.9.4 Interfaces of the tags (RT Uni)

19.9.4.1 IProcessValue (RT Uni)

Description

The C# interface "IProcessValue" specifies properties and methods for values of process tags of the Runtime system. The "IProcessValue" interface provides values from the result of a read operation or monitoring.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; }
```

"Value" property

Value of the tag at the moment of the read operation.

```
object Value { get; }
```

"Quality" property

Quality code of the read operation of the tag.

```
uint32 Quality { get; }
```

"TimeStamp" property

Time stamp of the last successful read operation of the tag.

```
DateTime TimeStamp { get; }
```

"Error" property

Error code of the last read or write operation of the tag.

```
int32 Error { get; }
```

Example

Output properties of the "IProcessValue" object that is returned by the ITag.Read method:

Copy code

```
public void ReadSingleTagSync()
{
    try
    {
        using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
        {
            IProcessValue value = myTag.Read(HmiReadType.Cache); // Reads value from Cache
            System.Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}
Quality: {3}", value.Name, value.TimeStamp, value.Value, value.Quality));
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
```

See also

ITag (Page 1104)

ITagSet (Page 1106)

ITagSetQCD (Page 1114)

19.9.4.2 ITag (RT Uni)

Description

The C# interface "ITag" specifies properties and methods for handling tags of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the tag that is read with the "Read" method.

```
string Name { get; set; }
```

"Read" method

Read process value and properties of the tag synchronously from the Runtime system.

```
IProcessValue Read(HmiReadType type = HmiReadType.Cache)
```

`type`
The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default parameter): Reads the tag value from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag value directly from the AS. The tag image is not used.

"Write" method

Write process value of the tag synchronously in the Runtime system.

```
void Write(
     object value,
     HmiWriteType type = HmiWriteType.NoWait)
```

- `value`
  Value of the tag
- `type`
  The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType.NoWait` (default parameter): Writes the tag value without waiting. Errors for the write operation are not detected.
  - `HmiWriteType.Wait`: Waits until the tag value is written in the AS. If an error occurs during the write operation, an exception is triggered.

"WriteQCD" method

Write process value with quality code of the tag synchronously in the Runtime system. The tag also has a freely definable time stamp. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

```
void Write(
     object value,
     DateTime timeStamp,
     uint32 qualityCode,
     HmiWriteType type = HmiWriteType.NoWait)
```

- `value`
  Value of the tag
- `timeStamp`
  Time stamp of the tag
- `qualityCode`
  Quality code of the tag
- `type`
  The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType.NoWait` (default parameter): Writes the tag value without waiting. Errors for the write operation are not detected.
  - `HmiWriteType.Wait`: Waits until the tag value is written in the AS. If an error occurs during the write operation, an exception is triggered.

"WriteWithOperatorMessage" method

Write process value of the tag synchronously in the Runtime system and create operator message. In addition to the reason, the operator message contains the old and new value, the user and host names and the unit.

```
void WriteWithOperatorMessage(
     object value,
     string reason)
```

- `value`
  Value of the tag
- `reason`
  Reason of the value change for message

Example

Read and write tag synchronously:

Copy code

```
public void ReadSingleTagSync()
{
    try
    {
        using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
        {
            IProcessValue value = myTag.Read(HmiReadType.Cache); // Reads value from Cache
            System.Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}
Quality: {3}", value.Name, value.TimeStamp, value.Value, value.Quality));
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}

public void WriteSingleTagSync()
{
    try
    {
        using (ITag odkTag = runtime.GetObject<ITag>("Tag1"))
        {
            int value = 5;
           odkTag.Write(value, HmiWriteType.NoWait);   // Writes value without waiting that
value has been written to PLC
            IProcessValue pvalue = odkTag.Read(HmiReadType.Cache);
        }
    }
    catch (Exception ex)
    {
        System.Console.WriteLine(string.Format("Exception occured {0}", ex.Message));
    }
}
```

See also

IProcessValue (Page 1102)

ITagSet (Page 1106)

19.9.4.3 ITagSet (RT Uni)

Description

The C# interface "ITagSet" specifies properties, methods and events for optimized access to several tags of the Runtime system.

After initialization of the "ITagSet" object, you can execute read and write access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Note

You have the option to enumerate via a TagSet and to access a special element via the name. This is useful if you want to change the value of a tag in the TagSet for a write operation. To write the values in the process image, a "Write" or "WriteAsync" method must first be called:

```
MyTagSet.Add("MyTag1", "MyTag2");

MyTagSet["MyTag1"] = 5; // Set value to 5 for write operation

MyTagSet.Write();
```

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identification characteristics of a TagSet. If several TagSets are used to read tags, you can assign the response to the request via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of tags of a TagSet list

```
int32 Count { get; }
```

"Add" method

Add tag to a TagSet.

Add tag with or without process value to the TagSet:

```
void Add(ICollection<string> tagNames)
```

`tagNames`
List with tag names for TagSet

or

```
void Add(string tagName, object value = null)
```

- `TagName`
  Name of the tag for TagSet
- `value`
  New value of the tag, default: No value

"Remove" method

Remove individual tag from a TagSet.

```
void Remove(string tagName)
```

`tagName`
Name of the tag that is removed from TagSet.

"Clear" method

Remove all tags from a TagSet.

```
void Clear()
```

"Read" method

Read process values and properties of all the tags of a TagSet synchronously from the Runtime system.

```
IList<IProcessValue> Read(HmiReadType type = HmiReadType.Cache)
```

`type`
The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag values directly from the automation system. The tag image is not used.

"ReadAsync" method

Read process values and properties of all the tags of a TagSet asynchronously from the Runtime system.

```
void ReadAsync(HmiReadType type = HmiReadType.Cache)
```

`type`
The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag values directly from the automation system. The tag image is not used.

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
IList<IErrorResult> Write(HmiWriteType type = HmiWriteType.NoWait)
```

`type`
The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:

- `HmiWriteType.NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
- `HmiWriteType.Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType.Wait` type and waits until the tag value has been written in the automation system. If an error occurs during the write operation, it is reported via the AsyncHandler.

```
void WriteAsync()
```

"WriteWithOperatorMessage" method

Write process values of all tags of a TagSet synchronously in the Runtime system and create operator messages. In addition to the reason, the operation messages contain the old and new value, the user and host names and the unit.

```
void WriteWithOperatorMessage(string reason)
```

`reason`
Reason of the value change for message

"Subscribe" method

Subscribe all tags of a TagSet asynchronously for cyclic monitoring of the process values.

Note

Tags from IO devices with the "Cyclic in operation" acquisition mode

For a tag with the acquisition mode "Cyclic in operation", the value stored in the process image when Subscribe is called might be outdated. OnAdd therefore only provides the QualityCode "uncertain". Only value changes made after the Subscribe call provide the current value and the QualityCode "good".

```
void Subscribe()
```

"CancelSubscribe" method

Cancel monitoring of all tags of a TagSet.

```
void CancelSubscribe()
```

"OnReadResult" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnReadResultHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnReadResultHandler OnReadResult
```

"OnWriteResult" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnWriteResultHandler" delegate.

Declares the event and the event handler for asynchronous write operations.

```
event OnWriteResultHandler OnWriteResult
```

"OnDataChanged" event

After a process value change of a monitored TagSet, the event calls an instance of the "OnDataChangedHandler" delegate.

Declares the event and the event handler for process value changes.

```
event OnDataChangedHandler OnDataChanged
```

"OnReadResultHandler" delegate

Specifies the signature of the event handling method for the "OnReadResult" event of a TagSet.

```
void OnReadResultHandler(
        ITagSet sender,
        IList<IProcessValue> values,
        bool completed)
```

- `sender`
  The read out "TagSet" object
- `values`
  Event data as a list of "IProcessValue" objects of the read tag
- `completed`
  Status of the asynchronous transfer:
  - True: All values of the TagSet are read.
  - False: Not all values of the TagSet are read yet.

"OnWriteResultHandler" delegate

Specifies the signature of the event handling method for the "OnWriteResult" event of a TagSet.

```
void OnWriteResultHandler(
        ITagSet sender,
        IList<IErrorResult> values,
        bool completed)
```

- `sender`
  The written "TagSet" object
- `values`
  Error during write operations of tags as "IErrorResult" object
- `completed`
  Status of the asynchronous transfer:
  - True: All values of the TagSet are written.
  - False: Not all values of the TagSet are written yet.

"OnDataChangedHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event of a TagSet.

```
void OnDataChangedHandler(
        ITagSet sender,
        IList<IProcessValue> values)
```

- `sender`
  The monitored "TagSet" object
- `values`
  Event data as a list of "IProcessValue" objects of the changed process values

Example

Read TagSet synchronously and write with change:

Copy code

```
public void ReadTagSetSync()
{
    try
    {
        using (ITagSet odkTagSet = runtime.GetObject<ITagSet>())
        {
            odkTagSet.Add(new string[] { "Tag1", "Tag2" });
            IList<IProcessValue> values = odkTagSet.Read();
            foreach (var value in values)
               System.Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}
Quality: {3}", value.Name, value.TimeStamp, value.Value, value.Quality));
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
public void WriteTagSetSyncWithChange()
{
    try
    {
        using (ITagSet odkTagSet = runtime.GetObject<ITagSet>())
        {
            odkTagSet.Add(new string[] { "Tag1", "Tag2" });
            // Modify the value of a tag in the tagset and write
            odkTagSet["Tag1"] = 5;
            odkTagSet.Write();
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
```

Define functions for asynchronous write operations and define the monitoring of TagSets.

Monitor events "OnWriteResult" or "OnDataChanged" and call the event handling methods "odkTagSet_OnWriteComplete" or "odkTag_OnDataChanged" on occurrence:

Copy code

```
public void WriteTagSetAsync()
{
    try
    {
        ITagSet odkTagSet = runtime.GetObject<ITagSet>();
        odkTagSet.Add("Tag1", 1);
        odkTagSet.Add("Tag2", 2);
        // Assign callback function
        odkTagSet.OnWriteResult += odkTagSet_OnWriteResult;
        odkTagSet.WriteAsync();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
public void odkTagSet_OnWriteResult(ITagSet sender, IList<KeyValuePair<string, int>>
values, bool completed)
{
    System.Console.WriteLine("odkTagSet_OnWriteComplete");
    sender.Dispose();
}
public void SubscribeTagSet()
{
    try
    {
        ITagSet odkTagSet = runtime.GetObject<ITagSet>();
        odkTagSet.Add(new string[] { "Tag1", "Tag2" });
        // Assign callback function
        odkTagSet.OnDataChanged += odkTagSet_OnDataChanged;
        odkTagSet.Subscribe();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
public void odkTagSet_OnDataChanged(ITagSet sender, IList<IProcessValue> pItems)
{
    try
    {
        foreach (var value in pItems)
            System.Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}
Quality: {3}", value.Name, value.TimeStamp, value.Value, value.Quality));
        // For test purpose: Cancel subscription after first notification
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
    finally
    {
        if (null != sender)
        {
            sender.CancelSubscribe();
```

Copy code

```
            sender.Dispose();
        }
    }
}
```

See also

IProcessValue (Page 1102)

ITag (Page 1104)

19.9.4.4 ITagSetQCD (RT Uni)

Description

The C# interface "ITagSetQCD" specifies properties, methods and events for optimized writing of several tags of the Runtime system. The tags also have a freely definable time stamp and quality code. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

After initialization of the "ITagSetQCD" object, you can have read access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Note

You have the option to enumerate via a TagSet and to access a special element via the name. This is useful if you want to change the value of a tag in the TagSet for a write operation. To write the values in the process image, a "Write" or "WriteAsync" method must first be called:

```
MyTagSet.Add("MyTag1", "MyTag2");
```

```
MyTagSet["MyTag1"] = 5; // Set value to 5 for write operation
```

```
MyTagSet.Write();
```

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identification characteristics of a TagSet. If several TagSets are used to read tags, you can assign the response to the request via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of tags of a TagSet list

```
int32 Count { get; }
```

"Add" method

Add user-defined tag with process value, quality code and time stamp to the TagSet.

```
void Add(string tagName, object value, DateTime timeStamp, uint32
qualityCode)
```

- `TagName`
  Name of the tag for TagSet
- `value`
  New value of the tag
- `timeStamp`
  Time stamp of the tag
- `qualityCode`
  Quality code of the tag

"Remove" method

Remove individual tag from a TagSet.

```
void Remove(string tagName)
```

`tagName`
Name of the tag that is removed from TagSet.

"Clear" method

Remove all tags from a TagSet.

```
void Clear()
```

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
IList<IErrorResult> Write(HmiWriteType type = HmiWriteType.NoWait)
```

`type`
The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:

- `HmiWriteType.NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
- `HmiWriteType.Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType.Wait` type and waits until the tag value has been written in the automation system. If an error occurs during the write operation, it is reported via the AsyncHandler.

```
void WriteAsync()
```

"OnWriteComplete" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnWriteCompleteTagSetQCDHandler" delegate.

Declares the event and the event handler for asynchronous write operations with quality codes and time stamps.

```
event OnWriteCompleteTagSetQCDHandler OnWriteComplete
```

"OnWriteCompleteTagSetQCDHandler" delegate

Specifies the signature of the event handling method for the "OnWriteComplete" event of a TagSet with quality code and time stamps.

```
void OnWriteCompleteTagSetQCDHandler(
        ITagSetQCD sender,
        IList<IErrorResult> pItems)
```

- `sender`
  Source of the event
- `pItems`
  Error during write operations of tag as "IErrorResult" object

Example

Write TagSet with time stamp and quality code synchronously:

Copy code

```
public void WriteTagSetQCDSync()
{
    try
    {
        using (ITagSetQCD odkTagSet = runtime.GetObject<ITagSetQCD>())
        {
            odkTagSet.Add("Tag1", 1, DateTime.Now, 128);
            odkTagSet.Add("Tag2", 2, DateTime.Now, 128);

            IList<IErrorResult> writeResult = odkTagSet.Write();
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}
```

See also

IProcessValue (Page 1102)

ITagSetQCDItem (Page 1117)

19.9.4.5 ITagSetQCDItem (RT Uni)

Description

The C# interface "ITagSetQCDItem" specifies properties for user-defined values of tags in a TagSetQCD. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, only the tag value is set. The QualityCode and time stamp are set internally by the system.

The objects "ITagSetQCDItem" are not used by the "ITagSetQCD" object and can also be edited with the methods of the "ITagSetQCD" interface.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; set; }
```

"Value" property

Tag value

```
object Value { get; set; }
```

"Quality" property

Quality code of the tag

```
int32 Quality { get; set; }
```

"TimeStamp" property

Time stamp of the tag

```
DateTime TimeStamp { get; set; }
```

See also

ITagSetQCD (Page 1114)

19.9.4.6 ILoggedTagValue (RT Uni)

Description

The C# interface "ILoggedTagValue" specifies the properties for process values of logging tags of a logging system. The "ILoggedTagValue" interface displays historical process values.

An "ILoggedTagValue" object is a pure data object which maps all properties of the logged process values.

Members

The following properties are specified in the interface:

"Name" property

Name of the logging tag

```
string Name { get; set; }
```

"Value" property

Process value of the logging tag

```
object Value { get; set; }
```

"Quality" property

Quality code of the process value

```
int16 Quality { get; set; }
```

"TimeStamp" property

Time stamp of the process value

```
DateTime TimeStamp { get; set; }
```

"Error" property

Error code of the process value

```
int32 Error { get; }
```

"Flags" property

Context information from the read operation for the process value

```
HmiTagLoggingValueFlags Flags { get; set; }
```

The "HmiTagLoggingValueFlags" enumeration contains the following bit-by-bit-coded values:

- 0: `Extra`
  There are still additional values at the time of the process value.
- 2: `Calculated`
  Process value is calculated.
- 16: `Bounding`
  Process value is a limit value.

- 32: `NoData`
  No additional information available
- 64: `FirstStored`
  Process value is the first value stored in the logging system.
- 128: `LastStored`
  Process value is the last value stored in the logging system.

See also

ILoggedTag (Page 1119)

ILoggedTagSet (Page 1120)

19.9.4.7 ILoggedTag (RT Uni)

Description

The C# interface "ILoggedTag" specifies properties and methods for handling logging tags of a logging system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the logging tag

```
string Name { get; set; }
```

"Read" method

Read logged process values of a logging tag of a time period synchronously from the logging system.

```
IList<ILoggedTagValue> Read(DateTime begin, DateTime end, bool
boundingValue)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `boundingValue`
  True, to additionally return high and low limits.

Example

Output process values of the logging tag "Tag1:Tag1Logging1" for a time period:

Copy code

```
public void ReadSingleTag()
{
    var tag = _runtime.GetObject<ILoggedTag>("Tag1:Tag1Logging1");
    var begin = DateTime.UtcNow.AddMinutes(-10);
    var end = DateTime.UtcNow;
    var values = tag.Read(begin, end, true);
    Print(values);
    tag.Dispose();
}
```

See also

ILoggedTagValue (Page 1118)

ILoggedTagSet (Page 1120)

19.9.4.8 ILoggedTagSet (RT Uni)

Description

The C# interface "ILoggedTagSet" specifies properties, methods and events for optimized access to several logging tags of a logging system.

After initialization of the "ILoggedTagSet" object, you can execute read and write access to multiple logging tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple logging tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identification characteristics of a LoggedTagSet. If several LoggedTagSets are used to read logging tags, you can assign the response to the request via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of logging tags of a LoggedTagSet list

```
int32 Count { get; }
```

"Add" method

Add logging tag to a LoggedTagSet.

Add logging tags with or without context to the LoggedTagSet:

```
void Add(ICollection<string> tagNames)
```

`tagNames`
List with names of logging tags for the LoggedTagSet

or

```
void Add(string tagName)
```

`tagName`
Name of a logging tag

"Remove" method

Remove individual logging tag from a LoggedTagSet.

```
void Remove(string tagName)
```

`tagName`
Name of the logging tag that is removed from the LoggedTagSet.

"Clear" method

Remove all logging tags from a LoggedTagSet.

```
void Clear()
```

"Read" method

Read all logging tags of a LoggedTagSet synchronously from the logging system.

```
IList<ILoggedTagValue> Read(DateTime begin, DateTime end, bool
boundingValue)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `boundingValue`
  True, to additionally return high and low limits.

"ReadAsync" method

Read all logging tags of a LoggedTagSet asynchronously from the logging system.

```
void ReadAsync(DateTime begin, DateTime end, bool boundingValue)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `boundingValue`
  True, to additionally return high and low limits.

"Subscribe" method

Subscribe all logging tags of a LoggedTagSet asynchronously for updating the process values following a change. When new process values are logged, they can be processed with the "OnDataChanged" event.

```
void Subscribe()
```

"CancelSubscribe" method

Cancel updating of process values following a change for all logging tags of a LoggedTagSet.

```
void CancelSubscribe()
```

"OnReadComplete" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnReadCompleteTagSetHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnReadCompleteTagSetHandler OnReadComplete
```

"OnDataChanged" event

After a process value change of a monitored LoggedTagSet, the event calls an instance of the "OnDataChangedTagSetHandler" delegate.

Declares the event and the event handler for process value changes.

```
event OnDataChangedTagSetHandler OnDataChanged
```

"OnReadCompleteTagSetHandler" delegate

Specifies the signature of the event handling method for the "OnReadComplete" event of a LoggedTagSet.

```
void OnReadCompleteTagSetHandler(
        ILoggedTagSet sender,
        uint32 errorCode,
        IList<ILoggedTagValue> values,
        bool completed)
```

- `sender`
  Source of the event
- `errorCode`
  Error during asynchronous transfer
- `values`
  Event data as a list of "ILoggedTagValue" objects of the read logging tag
- `completed`
  Status of the asynchronous transfer:
  - True: All values of the LoggedTagSet are read.
  - False: Not all values of the LoggedTagSet are read.

"OnDataChangedTagSetHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event of a LoggedTagSet.

```
void OnDataChangedTagSetHandler(
        ILoggedTagSet sender,
        uint32 errorCode,
        IList<ILoggedTagValue> value)
```

- `sender`
  Source of the event
- `errorCode`
  Error during asynchronous transfer
- `value`
  Event data as a list of "ILoggedTagValue" objects with process values of the changed logging tag

Example

Read and output process values of logging tags of a specific time period asynchronously in LoggedTagSet "tagSet":

Copy code

```
public void ReadTagSetAsync()
{
    try
    {
        var begin = DateTime.UtcNow.AddHours(-1);
        var end = DateTime.UtcNow;
        var tagSet = _runtime.GetObject<ILoggedTagSet>();
        tagSet.OnReadComplete += TagSet_OnReadComplete;
        tagSet.Add("Tag1:Tag1Logging1");
        tagSet.Add("Tag2:Tag2Logging1");
        tagSet.ReadAsync(begin, end, true);
    }
    catch (OdkException ex)
    {
        Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

private void TagSet_OnReadComplete(ILoggedTagSet sender, uint errorCode,
IList<ILoggedTagValue> values, bool completed)
{
    Print(values);
    sender.Dispose();
}
```

Output process values of logging tag "Tag1:Tag1Logging1" in case of change:

Copy code

```
public void SubscribeTagSet()
{
    try
    {
        ILoggedTagSet tagSet = _runtime.GetObject<ILoggedTagSet>();
        tagSet.Add("Tag1:Tag1Logging1");
        tagSet.OnDataChanged += tagSet_OnDataChanged;
        tagSet.Subscribe();
    }
    catch (OdkException ex)
    {
        Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

void tagSet_OnDataChanged(ILoggedTagSet sender, UInt32 errorCode, IList<ILoggedTagValue>
values)
{
    Print(values);
    sender.Dispose();
}
```

See also

ILoggedTag (Page 1119)

ILoggedTagValue (Page 1118)

19.9.4.9 ITags (RT Uni)

Description

The C# interface "ITags" defines methods with which you can access configured tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

"Find" method

Supplies a collection with "ITagAttributes" instances of the specified tag.

```
ICollection<ITagAttributes>  Find(ICollection<string> SystemNames,
string Filter = null, UInt32 LanguageID = 0)
```

- `SystemNames`
  Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
  Filter by name of the tags to restrict the search.
  Supports searching with wildcards (*).
- (Optional) `LanguageID`
  Language code ID of filter

"FindAsync" method

For asynchronous searching for "ITagAttributes" instances of the specified tags.

```
void FindAsync(IList<string> SystemNames, string Filter = null,
UInt32 LanguageID = 0)
```

- `SystemNames`
  Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
  Filter by name of the tags to restrict the search.
  Supports searching with wildcards (*)
- (Optional) `LanguageID`
  Language code ID of filter

"OnTagAttributesRecieved" event

After completion of the "FindAsync" method, the event calls an instance of the "OnTagAttributesRead" delegate.

```
event OnTagAttributesRead OnTagAttributesRecieved;
```

"OnTagAttributesRead" delegate

Specifies the signature of the event handling method for the "OnTagAttributesReceived" event of an "ITagAttributes" instance.

```
public delegate void OnTagAttributesRead(ITags sender,
ICollection<ITagAttributes> tagAttributes, bool completed)
```

- `sender`
  Source of the event
- `tagAttributes`
  Event data as collection of "ITagAttributes" instances
- `completed`
  Status of the asynchronous transfer:
  - True: All tags have been read out.
  - False: Not all tags have been read out yet.

19.9.4.10 ITagAttributes (RT Uni)

Description

The C# interface "ITagAttributes" defines properties of a tag.

Member

"Name" property

The name of the tag. Must be unique throughout the device.

```
string Name { get; }
```

"DisplayName" property

The display name of the tags

```
string DisplayName { get; }
```

"DataType" property

The data type of the tag

```
HmiDataType DataType { get; }
```

"Connection" property

The connection of the tag

The memory location of the tag in the controller is accessed via the connection.

```
string Connection { get; }
```

"AcquisitionCycle" property

The acquisition cycle of tags

If you configure a tag at an object, the acquisition cycle of the tag is displayed at the object.

```
string AcquisitionCycle { get; }
```

"AcquisitionMode" property

The acquisition mode of the tag

```
HmiAcquisitionMode AcquisitionMode { get; }
```

The enumeration "HmiAcquisitionMode" can contain the following values:

- Undefined (0)
- CyclicOnUse (1)
- CyclicContinous (2)
- OnDemand (3)
- OnChange (4)

"MaxLength" property

The length of the tags.

The length is only available with a string tag. The length is preset for structure tags and cannot be changed.

```
UInt32 MaxLength { get; }
```

"Persistent" property

The persistence of the tags

```
bool Persistent { get; }
```

"InitialValue" property

The start value of the tag

After Runtime starts, the tag retains the start value until an operator or the PLC changes the value.

```
object InitialValue { get; }
```

"InitialMaxValue" property

The start value for the event "On exceeding".

```
object InitialMaxValue { get; }
```

"InitialMinValue" property

The start value for the event "On falling below"

```
object InitialMinValue { get; }
```

"SubstituteValue" property

The substitute value of the tag

If the selected condition occurs, the tag is filled with the substitute value in runtime.

```
object SubstituteValue { get; }
```

"SubstituteValueUsage" property

The condition for using the substitute value of the tag

```
HmiSubstituteValueUsage SubstituteValueUsage { get; }
```

The enumeration "HmiSubstituteValueUsage" can contain the following values:

- None (0)
- InvalidValue (1)
- RangeViolation (2)
- InvalidValueOrRangeViolation (3)

19.9.4.11 ILoggingTags (RT Uni)

Description

The C# interface "ILoggingTags" defines methods with which you can access configured logging tags.

The interface inherits the Dispose() method of the "IDisposable" interface of the .NET framework.

Members

"Find" method

Supplies a collection with "ITagAttributes" instances of the specified logging tag.

```
ICollection<ILoggingTagAttributes>  Find(IList<string> SystemNames,
string Filter = null, UInt32 LanguageID = 0)
```

- `SystemNames`
  Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
  Filter by name of the tags to restrict the search.
  Supports searching with wildcard (*).
  Example:
  `Tag1:*` Supplies all logging tags of "Tag1".
- (Optional) `LanguageID`
  Language code ID of filter

"FindAsync" method

For asynchronous searching for "ILoggingTagAttributes" instances.

```
void FindAsync(IList<string> SystemNames, string Filter = null,
UInt32 LanguageID = 0)
```

- `SystemNames`
  Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
  Filter by name of the tags to restrict the search.
  Supports searching with wildcard (*).
  Example:
  `Tag1.*`: Supplies all logging tags of "Tag1".
- (Optional) `LanguageID`
  Language code ID of filter

"OnLoggingTagAttributesReadReceived" event

After completion of the "FindAsync" method, the event calls an instance of the "OnLoggingTagAttributesRead" delegate.

```
event OnLoggingTagAttributesRead OnLoggingTagAttributesReadRecieved;
```

"OnLoggingTagAttributesRead" delegate

Specifies the signature of the event handling method for the "OnLoggingTagAttributesRead" event of an "ILoggingTags" instance.

```
public delegate void OnLoggingTagAttributesRead(ILoggingTags sender,
ICollection<ILoggingTagAttributes> tagAttributes, bool completed);
```

- `sender`
  Source of the event
- `tagAttributes`
  Event data as collection of "ILoggingTagAttributes" instances
- `completed`
  Status of the asynchronous transfer:
  - True: All logging tags are read out.
  - False: Not all logging tags are read out yet.

19.9.4.12 ILoggingTagAttributes (RT Uni)

Description

The C# interface "ILoggingTagAttributes" defines properties of a logging tag.

Member

"Name" property

The name of the tag

```
string Name { get; }
```

"DisplayName" property

The display name of the tags

```
string DisplayName { get; }
```

"DataType" property

The data type of the tag

```
HmiDataType DataType { get; }
```

19.9.5 Interfaces of the alarms (RT Uni)

19.9.5.1 IAlarmResult (RT Uni)

Description

The C# interface "IAlarmResult" specifies properties for accessing properties of active alarms of the Runtime system.

An "IAlarmResult" object is a pure data object which maps all properties of an active alarm.

Members

The following properties are specified in the interface:

"InstanceID" property

InstanceID for an alarm with multiple instances

```
uint32 InstanceID { get; }
```

"SourceID" property

Source at which the alarm was triggered

```
string SourceID { get; }
```

"Name" property

Name of the alarm

```
string Name { get; }
```

"AlarmClassName" property

Name of the alarm class

```
string AlarmClassName { get; }
```

"AlarmClassSymbol" property

Symbol of the alarm class

```
string AlarmClassSymbol { get; }
```

"AlarmParameterValues" property

Parameter values of the alarm

```
IReadOnlyList<object> AlarmParameterValues { get; }
```

"AlarmText1" … "AlarmText9" properties

Additional texts 1-9 of the alarm

```
string AlarmText1 { get; }
…
```

```
string AlarmText9 { get; }
```

"ChangeReason" property

Trigger event for modification of alarm state

```
uint16 ChangeReason { get; }
```

"Connection" property

Connection by which the alarm was triggered

```
string Connection { get; }
```

"State" property

Current alarm state

```
HmiAlarmState State { get; }
```

The enumeration "HmiAlarmState" can contain the following values:

- Standard (0x00)
- Raised (0x01)
- RaisedCleared (0x02)
- RaisedAcknowledged (0x05)
- RaisedAcknowledgedCleared (0x06)
- RaisedClearedAcknowledged (0x07)
- Removed (0x80)

"StateText" property

Current alarm state as text, for example "Incoming" or "Outgoing"

```
string StateText { get; }
```

"EventText" property

Text that describes the alarm event.

```
string EventText { get; }
```

"InfoText" property

Text that describes an operator instruction for the alarm.

```
string InfoText { get; }
```

"TextColor" property

Text color of the alarm state

```
uint32 TextColor { get; }
```

"BackColor" property

Background color of the alarm state

```
uint32 BackColor { get; }
```

"Flashing" property

Indicates whether the alarm is flashing.

```
bool Flashing { get; }
```

"InvalidFlags" property

Marking of the alarm in case of invalid data

```
byte InvalidFlags { get; }
```

"ModificationTime" property

Time of the last modification to the alarm state

```
DateTime ModificationTime { get; }
```

"RaiseTime" property

Trigger time of the alarm

```
DateTime RaiseTime { get; }
```

"AcknowledgementTime" property

Time of alarm acknowledgment

```
DateTime AcknowledgementTime { get; }
```

"ClearTime" property

Time of alarm reset

```
DateTime ClearTime { get; }
```

"ResetTime" property

Time of alarm reset

```
DateTime ResetTime { get; }
```

"SuppressionState" property

Status of alarm visibility

```
byte SuppressionState { get; }
```

"SystemSeverity" property

Severity of the system error

```
UInt8 SystemSeverity { get; }
```

"Priority" property

Relevance for display and sorting of the alarm

```
UInt8 Priority { get; }
```

"Origin" property

Origin for display and sorting of the alarm

```
string Origin { get; }
```

"Area" property

Origin area for display and sorting of the alarm

```
string Area { get; }
```

"Value" property

Current process value of the alarm

```
object Value { get; }
```

"ValueQuality" property

Quality of the process value of the alarm

```
uint16 ValueQuality { get; }
```

"ValueLimit" property

Limit of the process value of the alarm

```
object ValueLimit { get; }
```

"UserName" property

User name of the operator input alarm

```
string UserName { get; }
```

"UserResponse" property

Expected or required user response to the alarm

```
byte UserResponse { get; }
```

"HostName" property

Name of the host that triggered the alarm.

```
string HostName { get; }
```

"Id" property

ID of the alarm that is also used in the display.

```
Uint32 Id { get; }
```

"AlarmGroupID" property

ID of the alarm group to which the alarm belongs.

```
UInt32 AlarmGroupID { get; }
```

"SourceType" property

Source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
HmiAlarmSourceType SourceType { get; }
```

The enumeration "HmiAlarmSourceType" can contain the following values:

- `Undefined` (0)
- `Tag` (1)
- `Controller` (2)
- `System` (3)
- `Alarm` (4)

"DeadBand" property

Range of the triggering tag in which no alarms are generated.

```
object DeadBand { get; }
```

"LoopInAlarm" property

Function that navigates from the alarm view to its origin.

```
string LoopInAlarm { get; }
```

"NotificationReason" property

Reason for the notification

```
HmiNotificationReason NotificationReason { get; }
```

The enumeration "HmiNotificationReason" can contain the following values:

- `Unknown` (0)
- `Add` (1)
  The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring.
- `Modify` (2)
  Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- `Remove` (3)
  The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

Note

Changes to the alarm only lead to notifications again when the alarm meets the filter criteria again. In this case, "NotificationReason" is set to `Add`.

Note

Removing alarm from business logic

The use case of the client determines whether the client ignores notifications via alarms with the "NotificationReason" `Modify` or `Remove`.

For example:

- State-based monitoring: The client wants to show a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
- Event-based monitoring: The client wants to send an email when an alarm comes in. Only the notification reason `Add` is relevant.

Example:

An ODK client begins monitoring with the filter criterion "State" = 1. An alarm is triggered. Runtime notifies the ODK client of the "NotificationReason" as follows:

Notification-Reason	Description
Add	• The "State" property is 1. The alarm has come in.
Modify	• The "State" property has not changed. • Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	The "State" property has changed, e.g. alarm has gone out.

Example

When alarm are incoming, output a selection of properties of the "IAlarmResult" objects:

Copy code

```
public void alarm_OnAlarmHandler(IAlarmSubscription sender, UInt32 nGlobalError, String
systemName, IList<IAlarmResult> value, bool completed)
{
    if (0 == nGlobalError)
    {
        try
        {
            AlarmList = new List<IALarmResult>();
            foreach (var item in value)
            {
               System.Console.WriteLine(string.Format("InstanceID: {0}", item.InstanceID));
                System.Console.WriteLine(string.Format("AcknowledgementTime: {0}",
item.AcknowledgementTime));
                System.Console.WriteLine(string.Format("AlarmClass: {0}",
item.AlarmClassName));
                System.Console.WriteLine(string.Format("AlarmClassSymbol: {0}",
item.AlarmClassSymbol));
                System.Console.WriteLine(string.Format("Id: {0}", item.Id));
                System.Console.WriteLine(string.Format("AlarmParameterValues: {0}",
item.AlarmParameterValues));
               System.Console.WriteLine(string.Format("AlarmText1: {0}", item.AlarmText1));
               System.Console.WriteLine(string.Format("AlarmText9: {0}", item.AlarmText9));
                System.Console.WriteLine(string.Format("Area: {0}", item.Area));
                System.Console.WriteLine(string.Format("BackColor: {0}", item.BackColor));
                System.Console.WriteLine(string.Format("ChangeReason: {0}",
item.ChangeReason));
                System.Console.WriteLine(string.Format("ClearTime: {0}", item.ClearTime));
               System.Console.WriteLine(string.Format("Connection: {0}", item.Connection));
                System.Console.WriteLine(string.Format("DeadBand: {0}", item.DeadBand));
                System.Console.WriteLine(string.Format("EventText: {0}", item.EventText));
                System.Console.WriteLine(string.Format("InfoText: {0}", item.InfoText));
                System.Console.WriteLine(string.Format("Flashing: {0}", item.Flashing));
                System.Console.WriteLine(string.Format("HostName: {0}", item.HostName));
                System.Console.WriteLine(string.Format("InvalidFlags: {0}",
item.InvalidFlags));
                System.Console.WriteLine(string.Format("LoopInAlarm: {0}",
item.LoopInAlarm));
                System.Console.WriteLine(string.Format("ModificationTime: {0}",
item.ModificationTime));
                System.Console.WriteLine(string.Format("Name: {0}", item.Name));
                System.Console.WriteLine(string.Format("Origin: {0}", item.Origin));
                System.Console.WriteLine(string.Format("Priority: {0}", item.Priority));
                System.Console.WriteLine(string.Format("RaiseTime: {0}", item.RaiseTime));
                System.Console.WriteLine(string.Format("ResetTime: {0}", item.ResetTime));
                System.Console.WriteLine(string.Format("SourceID: {0}", item.SourceID));
               System.Console.WriteLine(string.Format("SourceType: {0}", item.SourceType));
                System.Console.WriteLine(string.Format("State: {0}", item.State));
                System.Console.WriteLine(string.Format("StateText: {0}", item.StateText));
                System.Console.WriteLine(string.Format("SuppressionState: {0}",
item.SuppressionState));
                System.Console.WriteLine(string.Format("SystemSeverity: {0}",
item.SystemSeverity));
                System.Console.WriteLine(string.Format("TextColor: {0}", item.TextColor));
                System.Console.WriteLine(string.Format("User: {0}", item.UserName));
```

Copy code

```
                System.Console.WriteLine(string.Format("UserResponse: {0}",
item.UserResponse));
                System.Console.WriteLine(string.Format("Value: {0}", item.Value));
                System.Console.WriteLine(string.Format("ValueLimit: {0}",
item.ValueQuality));

                AlarmList.Add(item);
            }
            ...
        }
        ...
    }
}
```

See also

IAlarm (Page 1138)

IAlarmSubscription (Page 1156)

19.9.5.2 IAlarm (RT Uni)

Description

The C# interface "IAlarm" specifies properties and methods for handling active alarms of the Runtime system.

The methods trigger an exception in the case of an error.

Members

The following properties and methods are specified in the interface:

"Error" property

Error code of the alarm

```
uint32 Error { get; set; }
```

"Name" property

Name of the active alarm

```
string Name { get; set; }
```

"Acknowledge" method

Acknowledge active alarm or instance of an active alarm synchronously.

```
void Acknowledge(UInt32 instanceId)
```

- `instanceID`
  - Value "0": Acknowledge active alarm.
  - Value > "0": Acknowledge instance with this ID.

"Disable" method

Deactivate generation of the alarm in the alarm source synchronously.

```
void Disable()
```

"Enable" method

Activate generation of the alarm in the alarm source synchronously again.

```
void Enable()
```

"Reset" method

Acknowledge outgoing state of an active alarm or an instance of an active alarm synchronously.

```
void Reset(UInt32 instanceId)
```

- `instanceID`
  - Value "0": Acknowledge the counter state of the active alarm.
  - Value > "0": Acknowledge the counter state of an instance with this ID.

"Shelve" method

Hide active alarm synchronously.

```
void Shelve()
```

"Unshelve" method

Display hidden alarm synchronously again.

```
void Unshelve()
```

Example

Acknowledge active alarm synchronously:

Copy code

```
public void AcknowledgeAlarms(IList<IAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        if (AlarmList.Count == 1)
        {
            IAlarm alarmAck = runtime.GetObject<IAlarm>(AlarmList[0].Name);
            alarmAck.Acknowledge(AlarmList[0].Name);
        }
        else
        {
            //process multiple alarms
        }
    }
}
```

See also

IAlarmResult (Page 1130)

IAlarmSet (Page 1140)

IAlarmSubscription (Page 1156)

19.9.5.3 IAlarmSet (RT Uni)

Description

The C# interface "IAlarmSet" specifies properties, methods and events for optimized access to several active alarms of the Runtime system.

After initialization of the "IAlarmSet" object, you can execute asynchronous operations with multiple alarms in one call, e. g. acknowledgment. Simultaneous access demonstrates better performance and lower communication load than single access to multiple alarms.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Members

The following properties, methods and events are only specified in the interface:

Access operator "this[string]"

Accessing an element of an alarm set via the alarm name.

```
IAlarm this[string alarmName] { get; set; }
```

`alarmName`
Name of the alarm that is changed in the AlarmSet.

"Count" property

Number of alarms of an AlarmSet list.

```
int32 Count { get; }
```

"Add" method

Add an active alarm or an instance of the alarm to the AlarmSet.

```
IAlarm Add(string alarmName, UInt32 instanceId)
```

- `alarmName`
  Name of the alarm that is added to the AlarmSet.
- `instanceId`
  Value = "0": Add active alarm.
  Value > "0": Add instance with this ID.

"Remove" method

Remove a single alarm or an instance of an alarm from the AlarmSet.

```
void Remove(string alarmName, UInt32 instanceId)
```

- `alarmName`
  Name of the alarm that is removed from the AlarmSet.
- `instanceID`
  value = "0": Remove alarm
  Value > "0": Remove instance with this ID.

"Clear" method

Remove all alarms from an AlarmSet.

```
void Clear()
```

"Acknowledge" method

Acknowledge alarms of the AlarmSet asynchronously.

```
int Acknowledge()
```

"Disable" method

Deactivate generation of alarms of the AlarmSet in the alarm source asynchronously.

```
void Disable()
```

"Enable" method

Activate generation of alarms of the AlarmSet in the alarm source asynchronously again.

```
void Enable()
```

"Reset" method

Acknowledge outgoing state of the alarms of the AlarmSet asynchronously.

```
void Reset()
```

"Shelve" method

Hide alarms of the AlarmSet asynchronously.

```
void Shelve()
```

"Unshelve" method

Display hidden alarms of the AlarmSet again.

```
void Unshelve()
```

"OnAcknowledgeHandler" event

When an AlarmSet is acknowledged with the "Acknowledge" and "AckknowledgeInstance" methods, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous acknowledgment of the alarms.

```
event OnAlarmSetEventHandler OnAcknowledgeHandler
```

"OnResetHandler" event

Event calls an instance of the "OnAlarmSetEventHandler" delegate with the "Reset" and "ResetInstance" methods when the outgoing state of the alarms of an AlarmSet is acknowledged.

Declares the event and the event handler for the asynchronous removal of the alarms.

```
event OnAlarmSetEventHandler OnResetHandler
```

"OnDisableHandler" event

When the alarms of an AlarmSet are deactivated with the "Disable" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous deactivation of the alarms.

```
event OnAlarmSetEventHandler OnDisableHandler
```

"OnEnableHandler" event

When the alarms of an AlarmSet are reactivated with the "Enable" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous reactivation of the alarms.

```
event OnAlarmSetEventHandler OnEnableHandler
```

"OnShelveHandler" event

When the alarms of an AlarmSet are hidden with the "Shelve" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous hiding of the alarms.

```
event OnAlarmSetEventHandler OnShelveHandler
```

"OnUnshelveHandler" event

When the alarms of an AlarmSet are displayed with the "Unshelve" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the new asynchronous displaying of the alarms.

```
event OnAlarmSetEventHandler OnUnshelveHandler
```

"OnAlarmSetEventHandler" delegate

Specifies the signature of the event handling method for all events of an AlarmSet.

```
void OnAlarmSetEventHandler(
     IAlarmSet sender,
     uint32 errorCode,
     IList<IAlarmSetResult> values,
     bool completed)
```

- `sender`
  Source of the event
- `errorCode`
  Error during asynchronous transfer
- `values`
  Event data as a list of "IAlarmSetResult" objects of the processed alarms.
- `completed`
  Status of the asynchronous transfer:
  - True: All alarms of the AlarmSet are processed.
  - False: Not all alarms of the AlarmSet are processed yet.

Example

Acknowledge active alarms from the "AlarmList" list as an AlarmSet asynchronously:

Copy code

```
public void AcknowledgeAlarms(IList<IAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        if (AlarmList.Count == 1)
        {
            IAlarm alarmAck = runtime.GetObject<IAlarm>(AlarmList[0].Name);
            alarmAck.Acknowledge();
        }
        else
        {
            AlarmAckList = new List<IAlarmResult>();
            IAlarmSet alarmSet = runtime.GetObject<IAlarmSet>(); ;
            foreach (var alarmResult in AlarmList)
            {
                IAlarm pAlarm = null;
                pAlarm = alarmSet.Add(alarmResult.Name);
                AlarmAckList.Add(alarmResult);
            }

            // Assign callback function
            alarmSet.OnAcknowledgeHandler += alarmSet_OnAcknowledgeHandler;
            try
            {
                alarmSet.Acknowledge();
            }
            catch (OdkException ex)
            {
                System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
            }
        }
    }
}

public void alarmSet_OnAcknowledgeHandler(IAlarmSet sender, uint errorCode,
IList<IAlarmSetResult> values, bool completed)
{
    try
    {
        foreach (var item in values)
        {
            System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName:
{2} ",
                item.InstanceId, item.Name, item.SystemName));
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

Copy code

Remove active alarms from the "AlarmList" list as an AlarmSet asynchronously:

Copy code

```
public void ResetAlarms(IList<IAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        if (AlarmList.Count == 1)
        {
            IAlarm alarmReset = runtime.GetObject<IAlarm>(AlarmList[0].Name);
            alarmReset.Reset();
        }
        else
        {
            IAlarmSet alarmSet = runtime.GetObject<IAlarmSet>(); ;
            AlarmResetList = new List<IAlarmResult>();
            foreach (var alarmResult in AlarmList)
            {
                IAlarm pAlarm = null;
                pAlarm = alarmSet.Add(alarmResult.Name);
                AlarmResetList.Add(alarmResult);
            }
            alarmSet.OnResetHandler += alarmSet_OnReseteHandler;
            try
            {
                alarmSet.Reset();
            }
            catch (OdkException ex)
            {
                System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
            }
        }
    }
}

public void alarmSet_OnReseteHandler(IAlarmSet sender, uint errorCode,
IList<IAlarmSetResult> values, bool completed)
{
    try
    {
        foreach (var item in values)
        {
            System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName:
{2} ",
                item.InstanceId, item.Name, item.SystemName));
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

See also

IAlarmSetResult (Page 1148)

IAlarm (Page 1138)

19.9.5.4 IAlarmSetResult (RT Uni)

Description

The C# interface "IAlarmSetResult" specifies properties of alarms in AlarmSets. These properties are returned by the EventHandler for all events of AlarmSets.

Members

The following properties are specified in the interface:

"SystemName" property

System name of the Runtime system of an alarm in the AlarmSet

```
string SystemName { get; }
```

"Name" property

Name of an alarm in the AlarmSet

```
string Name { get; }
```

"InstanceId" property

InstanceID of an alarm in the AlarmSet

```
uint32 InstanceId { get; }
```

"Error" property

Error code of an alarm in the AlarmSet

```
uint32 Error { get; }
```

Example

Read out alarm of an AlarmSet after acknowledgment

Copy code

```
public void alarmSet_OnAcknowledgeHandler(IAlarmSet sender, uint errorCode,
IList<IAlarmSetResult> values, bool completed)
{
    try
    {
        foreach (var item in values)
        {
            System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName:
{2} ",
                item.InstanceId, item.Name, item.SystemName));
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

See also

IAlarmSet (Page 1140)

19.9.5.5 IAlarmTrigger (RT Uni)

Description

The C# interface "IAlarmTrigger" specifies methods for triggering alarms.

Members

"CreateSystemAlarm" method

Generates an alarm of the class SystemAlarmWithoutClearEvent with the state machine alarm without outgoing status with acknowledgment.

```
void CreateSystemAlarm(object alarmText, string area, object
alarmParameterValue1, ...,  object alarmParameterValue7)
```

- `alarmText`:
  The alarm text. You have the following options:
  - Transferring a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-definedtext lists

    This method processes only user-defined text lists.
  - Transfer static string with monolingual text.
- `area`:
  The area of origin of the alarm
- `alarmParameterValue1` to `alarmParameterValue7`:
  User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm
  texts:
  `@ScriptingSystemEvents.SystemAlarmWithoutClearEvent:SystemAlarmWithoutClearEvent`
- For monolingual alarm
  text:
  `@ScriptingSystemEvents.SystemAlarmWithoutClearEventText:SystemAlarmWithoutClearEvent`

"CreateSystemInformation" method

Generates an alarm of the class SystemInformation with the state machine alarm without outgoing status without acknowledgment.

```
void CreateSystemInformation(object alarmText, string area, object
alarmParameterValue1, ...,  object alarmParameterValue7)
```

- `alarmText`:
  The alarm text. You have the following options:
  - Transferring a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-definedtext lists

    This method processes only user-defined text lists.
  - Transfer static string with monolingual text.
- `area`:
  The area of origin of the alarm
- `alarmParameterValue1` to `alarmParameterValue7`:
  User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts: `@ScriptingSystemEvents.SystemInformation:SystemInformation`
- For monolingual alarm text: `@ScriptingSystemEvents.SystemInformationText:SystemInformation`

"CreateOperatorInputInformation" method

Generates an alarm of the class OperatorInputInformation with the state machine alarm without outgoing status without acknowledgment.

```
void CreateOperatorInputInformation(object alarmText, string area,
object alarmParameterValue1, ...,  object alarmParameterValue7)
```

- `alarmText`:
  The alarm text. You have the following options:
  - Transferring a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-definedtext lists

    This method processes only user-defined text lists.
  - Transfer static string with monolingual text.
- `area`:
  The area of origin of the alarm
- `alarmParameterValue1` to `alarmParameterValue7`:
  User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts: `@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputInformation`
- For monolingual alarm text: `@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputInformation`

Example

The following code examples show how to trigger alarms of the class SystemAlarmWithoutClearEvent. Alarms of the classes SystemInformation and OperatorInputInformation are triggered in the same way.

Copying code

```
public void CreateSystemAlarm()
{
   //Create SystemAlarm with monolingual alarm text
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
    systemAlarm.CreateSystemAlarm(alarmText: "Alarm Text", area: "Area",
        alarmParameterValue1: "Param1",
        alarmParameterValue2: "Param2",
        alarmParameterValue3: "Param3",
        alarmParameterValue4: "Param4",
        alarmParameterValue5: "Param5",
        alarmParameterValue6: "Param6",
        alarmParameterValue7: "Param7");
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}


public void CreateSystemAlarmWithAlarmTextAsTextList()
{
   //Create SystemAlarm with multilingual alarm text; the tranlsations are directly stored
in the text list
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
     var texlistforAlarmText = runtime.GetObject<ITextList>();
    // Text list: AlarmTextTemplate
    // Test list entry index: 101
    // Text: "My input msg. input value = @1%d@"
    texlistforAlarmText.Name = "AlarmTextTemplate";
    texlistforAlarmText.TextListEntryIndex = 101;
    systemAlarm.CreateSystemAlarm(alarmText: texlistforAlarmText, area: "Area",
        alarmParameterValue1: "125", // dynamic value for format specifier @1%d@
        alarmParameterValue2: "Param2",
        alarmParameterValue3: "Param3",
        alarmParameterValue4: "Param4",
```

Copying code

```
        alarmParameterValue5: "Param5",
        alarmParameterValue6: "Param6",
        alarmParameterValue7: "Param7");
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}


public void CreateSystemAlarmWithTextListAsParameterValue()
{
   //Create SystemAlarm with multilingual alarm text; the text list references other text
lists with tranlsations
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
    var textList1 = runtime.GetObject<ITextList>("Text_List_1");
    var textList2 = runtime.GetObject<ITextList>("Text_List_2");
    textList1.TextListEntryIndex = 1; //Eng TL @1%t#2T@ Val: @3%s@
    systemAlarm.CreateSystemAlarm(alarmText: textList1, area: "Area",
    alarmParameterValue1: 1,
    alarmParameterValue2: textList2, // text list object
    alarmParameterValue3: "Hello"); // Dynamic value of @3%s@
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}
```

See also

ITextList (Page 1156)

19.9.5.6 ITextList (RT Uni)

Description

The C# interface "ITextList" is used to transfer multilingual alarm texts for system alarms and operator input alarms. See section IAlarmTrigger (Page 1149), CreateSystemInformation method. An ITextList instance is passed to the alarm text. When the operator input alarm is generated, it is replaced by the configured text from the text list.

Members

"Name" property

The name of the text list.

```
string Name { get; set; }
```

"TextListEntryIndex" property

The index of the list entry

```
UInt32 TextListEntryIndex { get; set; }
```

19.9.5.7 IAlarmSubscription (RT Uni)

Description

The C# interface "IAlarmSubscription" specifies methods for monitoring alarms of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following methods and events are specified in the interface:

"SystemNames" property

System names of Runtime systems for the monitoring of active alarms.

```
IList<string> SystemNames { get; set; }
```

"Language" property

Country identification of the language of the monitored alarms

```
uint32 Language { get; set; }
```

"Filter" property

SQL-type string for filtering the result set of active alarms

```
string Filter { get; set; }
```

All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 1084).

"Start" method

Subscribe systems for monitoring of changes of active alarms.

```
void Start()
```

"Stop" method

Unsubscribe monitoring of active alarms.

Note

Start and Stop in Windows Forms applications

Do not call the "Stop" method for a Windows forms application in the same thread in which you called "Start".

```
void Stop()
```

"OnAlarmHandler" event

Following a change of alarm state on subscribed systems, the event calls an instance of the "OnAlarmHandler" delegate.

Declares the event and the event handler for asynchronous monitoring of alarms.

```
event OnAlarmHandler OnAlarmHandler
```

"OnPendingAlarmCompleteHandler" event

After completion of handling of all active alarms of a system, the event calls an instance of the "OnPendingAlarmCompleteHandler" delegate.

Declares the event and the event handler for confirmation of the asynchronous operations.

```
event OnPendingAlarmCompleteHandler OnPendingAlarmCompleteHandler
```

"OnAlarmHandler" delegate

Specifies the signature of the event handling method for the "OnAlarmHandler" event.

```
void OnAlarmHandler(
     IAlarmSubscription sender,
     uint32 systemError,
     string systemName,
     IList<IAlarmResult> values,
     bool completed)
```

- `sender`
  Reference to the "IAlarmSubscription" object
- `systemError`
  Error code for the asynchronous operation
- `systemName`
  Name of the Runtime system that is subscribed for alarm monitoring by the user.

- `values`
  Event data as a list of "IAlarmResult" objects of the monitored active alarms.
- `completed`
  Status of the asynchronous transfer:
  – True: All alarms are read out.
  – False: Not all alarms are yet read out.

"OnPendingAlarmCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPendingAlarmCompleteHandler" event.

```
void OnPendingAlarmCompleteHandler(IAlarmSubscription sender)
```

- `sender`
  Source of the event

Example

Define functions for asynchronous monitoring of active alarms. Monitor "OnAlarmHandler" event and when occurring call the event handling method "alarm_OnAlarmHandler":

Copy code

```
public void SubscribeAlarmWithFilterOnOriginAndAlarmclass()
{
    try
    {
        //  object filter = "AlarmClass = 'AlarmWithOptionalAcknowledgement' AND Origin = 
'Boiler'";
        IAlarmSubscription alarm = runtime.GetObject<IAlarmSubscription>();
        List<string> systemNames = new List<string>();
        systemNames.Add("SYSTEM1");

        // Assign alarm handler
        alarm.OnAlarmHandler += alarm_OnAlarmHandler;
        alarm.Filter = "AlarmClassName = 'AlarmWithOptionalAcknowledgement' AND Origin = 
'Boiler'";
        alarm.Language = 1033;
        alarm.SystemNames = systemNames;
        alarm.Start();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}

public void alarm_OnAlarmHandler(IAlarmSubscription sender, UInt32 nGlobalError, String 
systemName, IList<IAlarmResult> value, bool completed)
{
    if (0 == nGlobalError)
    {
        try
        {
            AlarmList = new List<IAlarmResult>();
            foreach (var item in value)
            {
               System.Console.WriteLine(string.Format("InstanceID: {0} SourceID: {1} Name: 
{2} State: {3} EventText: {4} StateText: {5} BackColor: {6} Flashing: {7} Quality: {8} 
ModificationTime: {9}",
                    item.InstanceID, item.SourceID, item.Name, item.State,
                    item.EventText, item.StateText, item.BackColor, item.Flashing, 
item.Quality, item.ModificationTime));
                AlarmList.Add(item);
            }

            // For test purpose: Cancel subscription after first notification
            AcknowledgeAlarms(AlarmList);
        }
        finally
        {
            if (null != sender)
            {
                sender.Stop();
                sender.Dispose();
            }
        }
    }
```

Copy code

```
    else
    {
        System.Console.WriteLine("AlarmSubscription Failed");
    }
}
```

See also

IAlarmResult (Page 1130)

IAlarm (Page 1138)

19.9.5.8 ILoggedAlarmResult (RT Uni)

Description

The C# interface "ILoggedAlarmResult" specifies methods for accessing properties of logged alarms of a logging system.

Members

The following properties are specified in the interface:

"InstanceID" property

InstanceID for a logged alarm with multiple instances

```
uint32 InstanceID { get; }
```

"Name" property

Name of the logged alarm.

```
string Name { get; }
```

"AlarmClassName" property

Name of the alarm class of the logged alarm.

```
string AlarmClassName { get; }
```

"AlarmClassSymbol" property

Symbol of the alarm class of the logged alarm.

```
string AlarmClassSymbol { get; }
```

"AlarmParameterValues" property

Parameter values of the logged alarm.

```
IReadOnlyList<object> AlarmParameterValues { get; }
```

"AlarmText1" … "AlarmText9" properties

Additional texts 1-9 of the logged alarm.

```
string AlarmText1 { get; }
…
string AlarmText9 { get; }
```

"ChangeReason" property

Trigger event of the change of the alarm state of the logged alarm.

```
uint16 ChangeReason { get; }
```

"Connection" property

Connection via which the logged alarm was triggered.

```
string Connection { get; }
```

"State" property

Alarm state of the logged alarm.

```
HmiAlarmState State { get; }
```

The enumeration "HmiAlarmState" can contain the following values:

- Standard (0x00)
- Raised (0x01)
- RaisedCleared (0x02)
- RaisedAcknowledged (0x05)
- RaisedAcknowledgedCleared (0x06)
- RaisedClearedAcknowledged (0x07)
- Removed (0x80)

"StateText" property

Alarm state of the logged alarm as text, e.g. "incoming" or "outgoing".

```
string StateText { get; }
```

"EventText" property

Text of the logged alarm that describes the alarm event.

```
string EventText { get; }
```

"TextColor" property

Text color of the alarm state of the logged alarm.

```
uint32 TextColor { get; }
```

"BackColor" property

Background color of the alarm state of the logged alarm.

```
uint32 BackColor { get; }
```

"Flashing" property

Indicates whether the logged alarm flashes.

```
bool Flashing { get; }
```

"InvalidFlags" property

Marking of the logged alarm in case of invalid data

```
byte InvalidFlags { get; }
```

"ModificationTime" property

Time of the last change of the alarm state of the logged alarm.

```
DateTime ModificationTime { get; }
```

"RaiseTime" property

Trigger time of the logged alarm.

```
DateTime RaiseTime { get; }
```

"AcknowledgementTime" property

Time at which the logged alarm was acknowledged.

```
DateTime AcknowledgementTime { get; }
```

"ClearTime" property

Time at which the logged alarm was cleared.

```
DateTime ClearTime { get; }
```

"ResetTime" property

Time at which the logged alarm was reset.

```
DateTime ResetTime { get; }
```

"SuppressionState" property

Status of the visibility of the logged alarm.

```
byte SuppressionState { get; }
```

"SystemSeverity" property

Severity of the system error

```
byte SystemSeverity { get; }
```

"Priority" property

Relevance for display and sorting of the logged alarm.

```
byte Priority { get; }
```

"Origin" property

Origin for display and sorting of the logged alarm.

```
string Origin { get; }
```

"Area" property

Origin area for display and sorting of the logged alarm.

```
string Area { get; }
```

"Value" property

Process value of the logged alarm.

```
object Value { get; }
```

"AlarmGroupName" property

Name of the group to which the alarm belongs. Blank if the alarm does not belong to a group.

```
string AlarmGroupName { get; }
```

"DeadBand" property

Range of the triggering tag in which no alarms are generated.

```
object DeadBand { get; }
```

"HostName" property

Name of the host that triggered the alarm.

```
string HostName { get; }
```

"InfoText" property

Localizable text for the alarm that contains the associated work instruction.

```
string InfoText { get; }
```

"StateMachine" property

StateMachine model of the alarm. The StateMachine represents the behavior of alarms through arrangement of alarm states and alarm events, e.g. "RaiseClear", "RaiseRequiresAcknowledgment" or "RaiseClearOptionalAcknowledgment".

```
byte StateMachine { get; }
```

"ValueQuality" property

Quality of the process value of the alarm

```
int32 ValueQuality { get; }
```

"ValueLimit" property

Limit of the process value of the logged alarm.

```
object ValueLimit { get; }
```

"SingleAcknowledgement" property

Indicates whether an alarm may be acknowledged only individually or may be acknowledged as a group or multiple selection.

```
bool SingleAcknowledgement { get; }
```

"LoggedAlarmStateObjectID" property

ID of alarm state for referencing within the logging system.

```
string LoggedAlarmStateObjectID { get; }
```

"ID" property

User-defined ID of the alarm that is also used in the display.

```
uint32 ID { get; }
```

"SourceType" property

Source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
HmiAlarmSourceType SourceType { get; }
```

The enumeration "HmiAlarmSourceType" can contain the following values:

- `Undefined` (0)
- `Tag` (1)
- `Controller` (2)
- `System` (3)
- `Alarm` (4)

"UserName" property

User name of the logged operator input alarm.

```
string UserName { get; }
```

"UserResponse" property

Expected or required user response to the logged alarm.

```
byte UserResponse { get; }
```

"LoopInAlarm" property

Function that navigates from the alarm view to its origin.

```
string LoopInAlarm { get; }
```

"Error" property

Error code of the logged alarm.

```
unit32 Error { get; }
```

See also

IAlarmLogging (Page 1166)

IAlarmLoggingSubscription (Page 1168)

19.9.5.9 IAlarmLogging (RT Uni)

Description

The C# interface "IAlarmLogging" specifies properties and methods for reading out logged alarms of a logging system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following methods and events are specified in the interface:

"Read" method

Read out logged alarms of a time period synchronously from logging system.

```
IList<ILoggedAlarmResult> Read(
     DateTime begin,
     DateTime end,
     string filter,
     uint32 language,
     IList<string> systemNames)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `filter`
  Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
  Country identification of the language of the logged alarm text
- `systemNames`
  System names of the Runtime systems of the logged alarms. Default: local system

"ReadAsync" method

Read out logged alarms of a time period asynchronously from logging system.

```
void ReadAsync(
     DateTime begin,
     DateTime end,
     string filter,
     uint32 language,
     IList<string> systemNames)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period

- `filter`
  Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
  Country identification of the language of the logged alarm text
- `systemNames`
  System names of the Runtime systems of the logged alarms. Default: local system

"OnReadComplete" event

After readout of the logged alarms, event calls an instance of the "OnReadCompleteAlarmLoggingHandler" delegate.

Declares the event and the event handler for the asynchronous readout of logged alarms.

```
event OnReadCompleteAlarmLoggingHandler OnReadComplete
```

"OnReadCompleteAlarmLoggingHandler" delegate

Specifies the signature of the event handling method for the "OnReadComplete" event.

```
void OnReadCompleteAlarmLoggingHandler(
     IAlarmLogging sender,
     uint32 errorCode,
     IList<ILoggedAlarmResult> values,
     bool completed)
```

- `sender`
  Source of the event
- `errorCode`
  Error code for the asynchronous operation
- `values`
  Event data as a list of "ILoggedAlarmResult" objects of the read-out logged alarms.
- `completed`
  Status of the asynchronous transfer:
  - True: All alarms are read out.
  - False: Not all alarms are yet read out.

Example

Reading out and outputting logged alarms asynchronously:

Copy code

```
public void ReadAsync()
{
    try
    {
        var alarm = _runtime.GetObject<IAlarmLogging>();
        var now = DateTime.UtcNow;
        var begin = now.AddMinutes(-5);
        var end = now.AddMinutes(-2);
        List<string> systemNames = new List<string>();
        systemNames.Add("SYSTEM1");
        alarm.OnReadComplete += Alarm_OnReadComplete;
        alarm.ReadAsync(begin, end, "", 1033, systemNames);
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}

private void Alarm_OnReadComplete(IAlarmLogging sender, uint errorCode,
IList<ILoggedAlarmResult> values, bool completed)
{
    PrintValues(values);
    sender.Dispose();
}

private void PrintValues(IList<ILoggedAlarmResult> values)
{
    foreach (var v in values)
    {
        Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}", v.Name,
v.RaiseTime, v.Value));
    }
}
```

See also

ILoggedAlarmResult (Page 1161)

IAlarmLoggingSubscription (Page 1168)

19.9.5.10 IAlarmLoggingSubscription (RT Uni)

Description

The C# interface "IAlarmLoggingSubscription" specifies properties and methods for monitoring logged alarms of a logging system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"SystemNames" property

System names of the Runtime systems of the logged alarms.

```
IList<string> SystemNames { get; set; }
```

"Language" property

Country identification of the language of the logged alarms.

```
uint32 Language { get; set; }
```

"Filter" property

SQL-like string for filtering the result set of the logged alarms.

```
string Filter { get; set; }
```

"Start" method

Start monitoring of logged alarms.

```
void Start()
```

"Stop" method

Stop monitoring of all logged alarms.

```
void Stop()
```

"OnDataChanged" event

Following a change of a monitored alarm in the logging systems, the event calls an instance of the "OnDataChangedAlarmLoggingHandler" delegate.

Declares the event and the event handler for the monitoring of logged alarms.

```
event OnDataChangedAlarmLoggingHandler OnDataChanged
```

"OnDataChangedAlarmLoggingHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event.

```
void OnDataChangedAlarmLoggingHandler(
     IAlarmLoggingSubscription sender,
uint32 errorCode,
IList<ILoggedAlarmResult> values)
```

- `sender`
  Source of the event
- `errorCode`
  Error code for the asynchronous operation
- `values`
  Event data as a list of "ILoggedAlarmResult" objects of the monitored logged alarms.

Example

Output logged alarms following a change.

Copy code

```
public void SubscribeAlarm()
{
    try
    {
        _alarm = _runtime.GetObject<IAlarmLoggingSubscription>();
        List<string> systemNames = new List<string>();
        systemNames.Add("SYSTEM1");
        // Assign alarm handler
        _alarm.OnDataChanged += Alarm_OnDataChanged;
        _alarm.Filter = "";
        _alarm.Language = 1033;
        _alarm.SystemNames = systemNames;
        _alarm.Start();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
    }
}

private void Alarm_OnDataChanged(IAlarmLoggingSubscription sender, uint errorCode,
IList<ILoggedAlarmResult> values)
{
    PrintValues(values);
}

private void PrintValues(IList<ILoggedAlarmResult> values)
{
    foreach (var v in values)
    {
        Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}", v.Name,
v.RaiseTime, v.Value));
    }
}
```

Cancel monitoring of logged alarms:

Copy code

```
public void CancelSubscription()
{
    try
    {
        if (_alarm != null)
        {
            _alarm.Stop();
            _alarm.Dispose();
        }
    }
    catch(Exception ex)
    {
        System.Console.WriteLine(ex.Message);
    }
}
```

See also

IAlarmLogging (Page 1166)

ILoggedAlarmResult (Page 1161)

19.9.6 Interfaces for connections (RT Uni)

19.9.6.1 IConnectionResult (RT Uni)

Description

The C# interface "IConnectionResult" provides access to the details of a connection.

Members

The following properties are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; }
```

"ConnectionState" property

Status of the connection

```
HmiConnectionState ConnectionState { get; }
```

The "HmiConnectionState" enumeration contains the following values:

- `Disabled` (0)
- `Connecting` (1)
- `Connected` (2)
- `Disconnecting` (3)
- `Disconnected` (4)
- `Reconnecting` (5)

"EstablishmentMode" property

Mode in which the connection will be established.

```
HmiConnectionEstablishmentMode EstablishmentMode { get; }
```

The "HmiConnectionEstablishmentMode" enumeration contains the following values:

- `None` (0)
- `AutomaticActive` (1)
- `AutomaticPassive` (2)
- `OnDemandActive` (3)
- `OnDemandPassive` (4)

"TimeSynchronizationMode" property

Mode of time synchronization between HMI system and automation system

```
HmiTimeSynchronizationMode TimeSynchronizationMode { get; }
```

The "HmiTimeSynchronizationMode" enumeration contains the following values:

- `None` (0)
- `Slave` (1)
- `Master` (2)

"DisabledAtStartup" property

Indicates whether the connection is disabled at the start of Runtime.

```
bool DisabledAtStartup { get; }
```

- `true`: Connection is disabled at the connection start.
- `false`: Connection is active at the connection start.

"Enabled" property

Indicates whether the connection is active.

```
bool Enabled { get; }
```

- `true`: Connection is active.
- `false`: Connection is not active.

"ConnectionType" property

Protocol of a communication driver, e.g. "S7 Classic".

```
string ConnectionType { get; }
```

"Error" property

Error code of the connection

```
uint32 Error { get; }
```

Example

Output connection details:

Copy code

```
public void Connection_Read()
{
    var (connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    if (connection != null)
    {
        var con = connection.Read();
        if (con != null)
        {
            Console.WriteLine("Connection Name is  {0} ", con.Name);
            Console.WriteLine("ConnectionState is  {0}", con.ConnectionState);
           Console.WriteLine("TimeSynchronizationMode is {0} ", con.TimeSynchronizationMod
e);
            Console.WriteLine("EstablishMentMode is  {0} ", con.EstablishmentMode);
            Console.WriteLine("Enabled is  {0} ", con.Enabled);
            Console.WriteLine("DisabledAtStartup is  {0} ", con.DisabledAtStartup);
            Console.WriteLine("ConnectionType is  {0} ", con.ConnectionType);
            Console.WriteLine(" Error is {0} ", con.Error);
        }
    }
}
```

See also

IConnection (Page 1175)

IConnectionSet (Page 1176)

19.9.6.2 IConnectionStatusResult (RT Uni)

Description

The C# interface "IConnectionStatusResult" provides access to the status of a connection.

Members

The following properties are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; }
```

"ConnectionState" property

Status of the connection

```
HmiConnectionState ConnectionState { get; }
```

The "HmiConnectionState" enumeration contains the following values:

- `Disabled` (0)
- `Connecting` (1)
- `Connected` (2)
- `Disconnecting` (3)
- `Disconnected` (4)
- `Reconnecting` (5)

"Error" property

Error code of the connection

```
uint32 Error { get; }
```

Example

Output status of a certain connection:

Copy code

```
public void Connection_GetConnectionState()
{
    var connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    var con = connection.GetConnectionState();
    if (con != null)
    {
        Console.WriteLine("Connection Name is : {0} ", con.Name);
        Console.WriteLine(" Error is : {0} ", con.Error);
        Console.WriteLine("Connection status is: {0}", con.ConnectionStatus);
    }
}
```

See also

IConnection (Page 1175)

IConnectionSet (Page 1176)

19.9.6.3 IConnection (RT Uni)

Description

The C# interface "IConnection" provides properties and methods for the access to a connection. A connection is a configured, logical assignment of two communication partners.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; set; }
```

"Read" method

Read connection details synchronously from the Runtime system.

```
IConnectionResult Read()
```

"GetConnectionState" method

Return connection status of a connection.

```
IConnectionStatusResult GetConnectionState()
```

"SetConnectionMode" method

Change connection status of a connection.

```
void SetConnectionMode(ConnectionMode mode)
```

The "ConnectionMode" enumeration contains the following values:

- `Disabled` (0)
- `Enabled` (1)

Examples

Disable connection:

Copy code

```
public void Connection_SetConnectionStateDisable()
{
    var connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    if (connection != null)
    {
        connection.SetConnectionMode(ConnectionMode.Disabled);
    }
}
```

Enable connection:

Copy code

```
public void SetConnectionStateEnable()
{
    using (IConnection connection = runtime.GetObject<IConnection>("HMI-ConnectionS7Plus"))
    {
        if (connection != null)
        {
            connection.SetConnectionMode(ConnectionMode.Enabled);
        }
    }
}
```

See also

IConnectionSet (Page 1176)

IConnectionResult (Page 1171)

IConnectionStatusResult (Page 1173)

19.9.6.4 IConnectionSet (RT Uni)

Description

The C# interface "IConnectionSet" specifies properties, methods and events for optimized access to several connections of the Runtime system.

After initialization of the "IConnectionSet" object, you have read/write access to multiple connections in one call. Simultaneous access takes place with better performance and lower communication load than single access to multiple connections.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

Access operator "this[string]"

Accessing or changing an element of a ConnectionSet via the connection name.

```
IConnection this[string connectionName] { get; set; }
```

`connectionName`
Name of the connection

"Count" property

Number of connections of a connection set list

```
int32 Cont { get; }
```

"Add" method

Add connections to a connection set.

```
void Add(ICollection<string> connections)
```

`connections`
List of connections for the connection set

"Remove" method

Remove individual connection from connection set.

```
void Remove(string connection)
```

`connection`
Name of connection that is removed from the connection set.

"Clear" method

Remove all connections from connection set.

```
void Clear()
```

"Read" method

Read connection details of all connections of the connection set synchronously from the Runtime system.

```
IList<IConnectionResult> Read()
```

"ReadAsync" method

Read connection details of all connections of the connection set asynchronously from the Runtime system.

```
void ReadAsync()
```

"GetConnectionState" method

Read connection status of all connections of the connection set synchronously from the Runtime system.

```
IList<IConnectionStatusResult> GetConnectionState()
```

"Subscribe" method

Subscribe all connections of a connection set asynchronously for change monitoring.

```
void Subscribe()
```

"CancelSubscribe" method

Cancel change monitoring of all connections of a connection set.

```
void CancelSubscribe()
```

"OnConnectionRead" event

After establishment of the connection, event calls an instance of the "OnConnectionHandler" delegate.

Declares the event and the event handler for the establishment of a connection.

```
event OnConnectionHandler OnConnectionRead
```

"OnConnectionStateChange" event

After change of the connection status, event calls an instance of the "OnConnectionStateChangeHandler" delegate.

Declares the event and the event handler for the change of the connection status.

```
event OnConnectionStateChangeHandler OnConnectionStateChange
```

"OnConnectionHandler" delegate

Specifies the signature of the event handling method for the "OnConnectionHandler" event of a connection set.

```
void OnConnectionHandler(
     IConnectionSet sender,
     uint32 systemError,
     IList<IConnectionResult> values)
```

- `sender`
  Source of the event
- `systemError`
  Error code
- `values`
  List of connections

"OnConnectionStateChangeHandler" delegate

Specifies the signature of the event handling method for the "OnConnectionStateChangeHandler" event of a connection set.

```
void OnConnectionStateChangeHandler(
    IConnectionSet  sender,
    uint32 systemError,
    IList<IConnectionStatusResult> values)
```

- `sender`
  Source of the event
- `systemError`
  Error code
- `values`
  List of status changes of the connections

Examples

Monitor connection status:

Copy code

```
public void ConnnectionSet_Subscribe()
{
    Console.WriteLine(" ......Connection Set:Subscription start ......");
    using (IConnectionSet subscribe = runtime.GetObject<IConnectionSet>())
    {
        if (subscribe != null)
        {
            ICollection<string> list = new string[] { "HMI-Connection", "HMI-
ConnectionS7Plus" };
            subscribe.Add(list);
            subscribe.OnConnectionStateChanged += Subscribe_OnConnectionStateChanged;
            subscribe.Subscribe();
            Thread.Sleep(500);
        }
    }
}

private void Subscribe_OnConnectionStateChanged(IConnectionSet sender, uint systemError,
IList<IConnectionStatusResult> values)
{
    if (0 == systemError)
    {
        try
        {
            foreach (var item in values)
            {
                Console.WriteLine("Name:{0} ", item.Name);
                Console.WriteLine("State:{0} ", item.ConnectionStatus.ToString());
                Console.WriteLine("Error:{0} ", item.Error);
            }
        }

        catch (OdkException ex)
        {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
        }

        finally
        {
            if (null != sender)
            {
                sender.CancelSubscribe();
                Console.WriteLine("Subscription cancelled");
                sender.Dispose();
            }
        }
    }
    else
    {
        System.Console.WriteLine("Connection set subscription  Failed");
    }
}
```

Copy code

Read out connection synchronously:

Copy code

```
public void ConnectionSet_Read()
{
    Console.WriteLine(" .......Connection Set: Read ...... ");
    using (IConnectionSet read = runtime.GetObject<IConnectionSet>())
    {
        if (read != null)
        {
            ICollection<string> list = new string[] { "HMI-Connection", "HMI-
ConnectionS7Plus" };
            read.Add(list);

            IList<IConnectionResult> connectionResult = read.Read();
            foreach (var item in connectionResult)
            {
                System.Console.WriteLine(string.Format("Connection Name is  {0} ",
item.Name));
                System.Console.WriteLine(string.Format("ConnectionState is  {0}",
item.ConnectionState.ToString()));
                System.Console.WriteLine(string.Format("TimeSynchronizationMode is {0} ",
item.TimeSynchronizationMode.ToString()));
                System.Console.WriteLine(string.Format(" Error is {0} ", item.Error));
                System.Console.WriteLine(string.Format("EstablishMentMode is  {0} ",
item.EstablishmentMode));
                System.Console.WriteLine(string.Format("Enabled is  {0} ", item.Enabled));
                System.Console.WriteLine(string.Format("DisabledAtStartup is  {0} ",
item.DisabledAtStartup));
                System.Console.WriteLine(string.Format("ConnectionType is  {0} ",
item.ConnectionType));
            }
        }
    }
}
```

Read out connection asynchronously:

Copy code

```
public void ConnectionSet_ReadAsync()
{
    Console.WriteLine(" ......Connection Set: ReadAsync start ......");
    IConnectionSet readAsync = runtime.GetObject<IConnectionSet>();
    if (readAsync != null)
    {
       ICollection<string> list = new string[] { "HMI-Connection", "HMI-ConnectionS7Plus" };
        readAsync.Add(list);
        readAsync.OnConnectionRead += Read_OnConnectionComplete;
        readAsync.ReadAsync();
        Thread.Sleep(5000);
    }
}

private void Read_OnConnectionComplete(IConnectionSet sender, uint systemError,
IList<IConnectionResult> values)
{
    foreach (var item in values)
    {
        Console.WriteLine("Name:{0} ", item.Name);
        Console.WriteLine("State:{0} ", item.ConnectionState);
        Console.WriteLine("establishmentMode:{0} ", item.EstablishmentMode);
        Console.WriteLine("TimeSynchronizationMode:{0} ", item.TimeSynchronizationMode);
        Console.WriteLine("ConnectionType:{0} ", item.ConnectionType);
        Console.WriteLine("Enabled:{0} ", item.Enabled);
        Console.WriteLine("DisabledAtStartup:{0} ", item.DisabledAtStartup);
        Console.WriteLine("Error:{0} ", item.Error);
    }
}
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

See also

IConnection (Page 1175)

IConnectionResult (Page 1171)

IConnectionStatusResult (Page 1173)

19.9.7 Interfaces of the Plant Model (RT Uni)

19.9.7.1 IPlantModel (RT Uni)

Description

The C# interface "IPlantModel" specifies methods for access to object instances of the plant model of a Runtime system. The "IPlantModel" object represents the plant model of the graphical Runtime system.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Formatting a path in the hierarchy

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Note

Fixed code HierarchyName

In the current version, `HierarchyName` has a fixed code which is "hierarchy".

Members

The class implements the following methods:

"GetPlantObject" method

Supplies an object instance of "IPlantObject".

```
IPlantObject GetPlantObject(string  plantObject)
```

- `plantObject`
  Identifies an `IPlantObject` instance by its name or its path in the hierarchy.

"GetPlantObjectsByType" method

Supplies a list with instances of "IPlantObject" that have a specific type.

```
IList<IPlantObject> GetPlantObjectsByType(string
plantObjectTypeFilter, string viewFilter = null)
```

- `plantObjectTypeFilter`
  Filter for the "IPlantObject" type on which the instances are based.
- Optional: `viewFilter`
  Filter for the path in the hierarchy. Only instances from a specific node are returned.

"GetPlantObjectsByExpression" method

Supplies a list with instances of "IPlantObject" instances. The instances are filtered by type and property values.

```
IList<IPlantObject> GetPlantObjectsByExpression(ICollection<string>
propertyNames, string plantObjectTypeFilter, string
expressionFilter, string viewFilter)
```

- `propertyNames`
  A list with property names
- `plantObjectTypeFilter`
  Filter for the object type on which the instances are based.
- `expressionFilter`
  An expression that is a filter for the property values.
- Optional: `viewFilter`
  Filter for a hierarchy path.

Example:

```
var plantObjectArr =
PlantModel.GetPlantObjectsByExpression("Temperature", "Motor",
"Temperature>100");
```

"GetPlantObjectsByPropertyNames" method

Supplies a list with "IPlantObject" instances that have specific properties and originate in a specific plant node.

```
IList<IPlantObject>
GetPlantObjectsByPropertyNames(ICollection<string> propertyNames,
string viewFilter = null)
```

- `propertyNames`
  A list with property names
  If the list contains multiple values, all properties must be available at the object.
- Optional: `viewFilter`
  Filter for a hierarchy path.

Example

Example of a hierarchy path:

Hierarchy path	Referenced object instance
`System2.TechnologicalHierarchy::P1/S1/L2/LeftPump`	References the "LeftPump" object instance in the "TechnologicalHierarchiy" of system2.
`.TechnologicalHierarchy::P1/S1/L2/LeftPump`	References the "LeftPump" object instance in the "TechnologicalHierarchiy" of the local system.
`U4711`	References the "U4711" object instance of the local system.
`System2::U4711`	References the "U4711" object instance of System2.

Sample code

Copy code

```
public void Odk_GetPlantObjectsByType()
{
    using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
    {    //gets node for specified Node path
        IList<IPlantObject> plantObject =
myPlantModel.GetPlantObjectsByType("RUNTIME_1::NodeType1",
        ".hierarchy::RootNodeName\\Node1");

        if (plantObject.Count() > 0)
        {
             foreach (IPlantObject item in plantObject)
             {
                 System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
item.CurrentPlantView, item.Name));
             }
        }
    }
}
```

19.9.7.2 IPlantObject (RT Uni)

Description

The C# interface "IPlantObject" specifies properties and methods for handling object instances of the plant model of a Runtime system.

An object instance in the plant model is based on an object type and its data structure. Each object instance receives its position within the hierarchy by assigning it to a hierarchy node.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

`[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID`

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following properties, methods and events are specified in the interface:

"Name" property

The name for unique identification of the object instance.

```
string Name { get; }
```

"Parent" property

The parent object instance in the hierarchy.

```
IPlantObject Parent { get; }
```

"Children" property

List of child object instances in the hierarchy

```
IReadOnlyList<IPlantObject> Children { get; }
```

"PlantViewPaths" property

The dictionary with string/string pairs that maps the hierarchy names to the hierarchy paths for all hierarchies that contain the "IPlantObject" instance (hierarchy name to hierarchy path).

```
IReadOnlyDictionary<string, string> PlantViewPaths { get; }
```

"CurrentPlantView" property

Path and name of the object instance in the currently selected hierarchy, e.g. "Maintenance".

The "CurrentPlantView" property is used as basis for navigation with the "Parent" or "Children" properties. If the object instance is only contained in one hierarchy, "CurrentPlantView" contains its path. If the object is contained in several views, the hierarchy path must be set via this property before the "Parent" or "Children" property can be used.

```
string CurrentPlantView { get; set; }
```

"GetProperty" method

Supplies a property of the object instance.

```
IPlantObjectProperty GetProperty(string propertyName)
```

`propertyName`
Name of an object instance property

"GetProperties" method

Supplies a two-dimensional list (name-object pairs) of the data structure of the object instance. The list allows access to the instance properties.

```
IPlantObjectPropertySet GetProperties(ICollection<string>
propertyNames = null)
```

- Optional: `propertyNames`
  List with names of one or multiple object instance properties.
  Without parameters, all properties of an object instance are returned

"GetActiveAlarms" method

Supplies all active alarms that the object instance contains at the time it is called in the active hierarchy. Unlike with an AlarmSubscription, no status changes or new alarms are signaled that occur after the function call. Users can filter the alarms or specify a SystemName if they only want to receive the active alarms of a specific system.

```
void GetActiveAlarms(UInt32 languageId, bool includeChildren =
false, string filter = null);
```

- `languageID`
  Language code of the language for all alarm texts and the filters. See chapter Locale IDs of the supported languages (Page 1085).
- Optional: `includeChildren`
  The active alarms of the child instances are returned as well.
- Optional: `filter`
  SQL-type string for filtering the alarm texts. The filter can contain operators. See also Syntax of the alarm filter (Page 1084).

"CreateAlarmSubscription" method

Supplies a "PlantObjectAlarmSubscription" that can be used to start and stop an alarm subscription.

```
IPlantObjectAlarmSubscription CreateAlarmSubscription();
```

"PlantObjectAlarmHandler" event

The event calls an instance of the "OnPlantObjectAlarmHandler" delegate.

```
event OnPlantObjectAlarmHandler PlantObjectAlarmHandler;
```

"OnPlantObjectAlarmHandler" delegate

Specifies the signature of the event handling method for the "`PlantObjectAlarmHandler`" event of an "IPlantObject" instance.

```
public delegate void OnPlantObjectAlarmHandler(
        IPlantObject sender,
        UInt32 systemError,
        string systemName,
        IList<IAlarmResult> values,
        bool completed);
```

- `sender`
  Source of the event
- `systemError`
  Supplies an error code when a global error has occurred. When the error code is set, `values` is irrelevant.
- `systemName`
  Name of the Runtime system that is subscribed for alarm monitoring by the user.
- `values`
  Event data as a list of "IAlarmResult" instances of the monitored active alarm.
- `completed`
  Status of the asynchronous transfer:
  - `True`: All alarms are read out.
  - `False`: Not all alarms are yet read out.

Example

Copy code

```
public void Odk_PlantObjectGetProperties()
{
    var myPlantModel = _runtime.GetObject<IPlantModel>();
    var strNodeName = ".hierarchy::RootNodeName/Node1";
    var plantObject = myPlantModel.GetPlantObject(strNodeName);
    Console.WriteLine("ViewName: {0} Name: {1}", plantObject.CurrentPlantView, plantObject.Name);
    // get the  plant objectproperties by propeyty names
    var plantObjectProperties = plantObject.GetProperties();
    if (plantObjectProperties != null)
    {
        var nCount = plantObjectProperties.Count;
        var listPropValues = plantObjectProperties.Read();
        Console.WriteLine("Number of Properties {0}", nCount);
        foreach (var item in listPropValues)
        {
            Console.WriteLine("Property Name is {0}  ", item.Name);
            Console.WriteLine("Property Value is {0}  ", item.Value);
            Console.WriteLine("Property Quality is {0} ", item.Quality);
            Console.WriteLine("Property Error is {0} ", item.Error);
        }
    }
}
```

19.9.7.3 IPlantObjectProperty (RT Uni)

Description

The C# interface "IPlantObjectProperty" specifies the handling of properties of object instances of the plant model of a Runtime system. The properties represent the data structure of an object instance.

The object instance communicates with the automation system through the properties of the data structure. The values of the properties are obtained from linked process tags or internal tags.

You reference an "IPlantObjectProperty" object using the `IPlantObject.GetProperty` or `IPlantObject.GetProperties` method.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The interface has the following properties and methods:

"Name" property

Name of the property

```
string Name { get; }
```

"Read" method

Reads the value of the "IPlantObjectProperty" instance synchronously and returns it as an "IPlantObjectPropertyValue" object. The value, the quality code and the time stamp of the property are determined when the property is read.

```
IPlantObjectPropertyValue Read()
```

"Write" method

Writes the value synchronously to the "IPlantObjectProperty" instance.

```
void Write(object Value)
```

- `Value`
  New process value of the property

Example

Copy code

```
public void Odk_PlantObjectGetPropertyWrite()
{
    using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
    {
       string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        using (IPlantObject plantobject = myPlantModel.GetPlantObject(strNodeName))
        {
            System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantobject.CurrentPlantView, plantobject.Name));
            if (plantobject != null)
                {
                    string strName = "NodeProperty_1";
                    using (IPlantObjectProperty plantObjectProperty =
plantobject.GetProperty(strName))
                    {
                        if (plantObjectProperty != null)
                        {
                            IPlantObjectPropertyValue pValue = plantObjectProperty.Read();
                            System.Console.WriteLine(string.Format("Property Name: {0}
property value before write
                            operation: {1}", strName, pValue.Value));
                            Object value = 400; // Write Cpm Node Property
                            plantObjectProperty.Write(value);
                           IPlantObjectPropertyValue pValues = plantObjectProperty.Read();
                            System.Console.WriteLine(string.Format("Property Name: {0}
property value after write Operration: {1}",strName, pValues.Value));
                        }
                    }
                }
            }
        }
    }
}
```

19.9.7.4 IPlantObjectPropertyValue (RT Uni)

Description

The C# interface "IPlantObjectPropertyValue" specifies the properties of process tags that are connected to an object instance property of the Runtime system.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; }
```

"Value" property

Value of the tag at the moment of the read operation.

```
object Value { get; }
```

"Quality" property

Quality code of the read operation of the tag.

```
Int32 Quality { get; }
```

"TimeStamp" property

Time stamp of the last successful read operation of the tag.

```
DateTime TimeStamp { get; }
```

"Error" method

Error code of the last read or write operation of the tag.

```
UInt32 Error { get; }
```

"OnPlantModelPropertySubscriptionHandler" delegate

Specifies the signature of the event handling method for the "`OnPlantModelPropertySubscriptionHandler`" event of an "IPlantObjectPropertySet" instance.

```
public delegate void OnPlantModelPropertySubscriptionHandler(
        IPlantObjectPropertySet sender,
         IList<IPlantObjectPropertyValue> values);
```

- `sender`
  Source of the event
- `values`
  Event data as a list of "IPlantObjectPropertyValue" instances of the monitored active alarm.

Example

Copy code

```
public void Odk_PlantObjectGetPropertyRead()
{
    using (var myPlantModel = _runtime.GetObject<IPlantModel>())
    {
        var strNodeName = ".hierarchy::RootNodeName/Node1";
        //gets node for specified Node path
        using (var plantobject = myPlantModel.GetPlantObject(strNodeName))
        {
            Console.WriteLine("ViewName: {0} Name: {1}", plantobject.CurrentPlantView, plantobject.Name);
            var strName = "NodeProperty_1";
            using (var plantObjectProperty = plantobject.GetProperty(strName))
            {
                if (plantObjectProperty != null)
                {
                    // Read Cpm Node Property
                    var plantObjectPropertyValue = plantObjectProperty.Read();
                    if (null != plantObjectPropertyValue)
                    {
                        Console.WriteLine(
                        "Name= {0} TimeStamp {1} QualityCode {2} Error {3} Value {4}",
                        plantObjectPropertyValue.Name, plantObjectPropertyValue.TimeStamp,
                        plantObjectPropertyValue.Quality, plantObjectPropertyValue.Error,
                        plantObjectPropertyValue.Value);
                    }
                }
            }
        }
    }
}
```

19.9.7.5 IPlantObjectPropertySet (RT Uni)

Description

The C# interface "IPlantObjectPropertySet" specifies properties, methods and events for optimized access to several IPlantObjectProperty instances of the Runtime system.

After initialization of the "IPlantObjectPropertySet" object, you have read/write access to multiple IPlantObjectProperty instances in one call. Simultaneous access has better performance and a lower communication load than single access to multiple properties.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Member

The following properties, methods and events are specified in the interface:

"ContextID" property

"ContextID" can be useful for asynchronous methods. Users can assign "ContextID" if they have to assign the answer from the system to a read/write job.

Default value -1: "ContextId" is not used.

```
Int32 ContextId { get; set; }
```

"[propertyName]" property

Change the process value of a property of the "IPlantObjectPropertySet" instance.
The value is changed by the property only in the local "IPlantObjectPropertySet" instance. To write the values in the process image, a "Write" or "WriteAsync" method must be called.

```
object this[string propertyName] { get; set; }
```

- `propertyName`
  Name of the property that is changed in the PropertySet.

"Count" property

The number of properties of the "IPlantObjectPropertySet" instance.

```
Uint32 Count { get; }
```

"Read" method

Supplies a list with all values of the "IPlantObjectProperty" instances contained in the "IPlantObjectPropertySet" instance. The values are read synchronously.

```
IList<IPlantObjectPropertyValue> Read();
```

"ReadAsync" method

Reads the values of all "IPlantObjectProperty" instances of the "IPlantObjectPropertySet" instance asynchronously.

```
void ReadAsync()
```

"Write" method

Writes the values of the "IPlantProperty" instances of the "PlantObjectPropertySet" instance synchronously to the Runtime system. Write operation errors are returned in a list with "IErrorResult" instances.

```
IList<IErrorResult> Write()
```

"WriteAsync" method

Writes the values of all "IPlantObjectProperty" instances of the "PlantObjectPropertySet" instance asynchronously to the Runtime system.

```
void WriteAsync()
```

"Subscribe" method

Subscribes all properties of the "IPlantObjectPropertySet" instance asynchronously for change monitoring.

```
void Subscribe()
```

"Add" method

Adds one or more "IPlantObjectProperty" instances to the "IPlantObjectPropertySet" instance.

The method can be called as follows:

- Adding multiple "IPlantObjectProperty" instances without value:
  `void Add(ICollection<string> propertyNames)`
  - `name`
    A collection with the names of the "IPlantObjectProperty" instances.
- Adding a "IPlantObjectProperty" instance with value:
  `void Add(string propertyName, object value);`
  - `propertyName`
    Name of the "IPlantObjectProperty" instance
  - `value`
    New process value of the "IPlantObjectProperty" instance

"Remove" method

Removes a property from the "IPlantObjectPropertySet" instance.

```
void Remove(string propertyName)
```

- `propertyName`
  Name of the property that is being removed.

"Clear" method

Removes all properties from the "IPlantObjectPropertySet" instance.

```
void Clear()
```

"OnPropertySetReadComplete" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnPropertySetReadCompleteHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnPropertySetReadCompleteHandler OnPropertySetReadComplete
```

"OnPropertySetWriteComplete" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnPropertySetWriteCompleteHandler" delegate.

Declares the event and the event handler for asynchronous write operations.

```
event OnPropertySetWriteCompleteHandler OnPropertySetWriteComplete
```

"OnPlantModelPropertySubscriptionNotification" event

After the change of a monitored PropertySet, the event calls an instance of the "OnPlantModelPropertySubscriptionHandler" delegate.

Declares the event and the event handler when changing a PropertySet.

```
event OnPlantModelPropertySubscriptionHandler
OnPlantModelPropertySubscriptionNotification
```

"OnPropertySetReadCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPropertySetReadComplete" event of an "IPlantObjectPropertySet" instance.

```
void OnPropertySetReadCompleteHandler(
        IPlantObjectPropertySet sender,
        UInt32 errorCode,
        IList<IPlantObjectPropertyValue> values)
```

- `sender`
  Source of the event
- `errorCode`
  Supplies an error code when a global error has occurred.
- `values`
  Event data as a list of "IPlantObjectPropertyValue" instances of the read property.

"OnPropertySetWriteCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPropertySetWriteComplete" event of an "IPlantObjectPropertySet" instance.

```
void OnPropertySetWriteCompleteHandler(
        IPlantObjectPropertySet sender,
        UInt32 errorCode,
        IList<IPlantObjectPropertyValue> values)
```

- `sender`
  Source of the event
- `errorCode`
  Supplies an error code when a global error has occurred. When the error code is set, `values` is irrelevant.
- `values`
  Event data as a list of "IPlantObjectPropertyValue" instances of the read property.

"OnPlantModelPropertySubscriptionHandler" delegate

Specifies the signature of the event handling method for the "OnPlantModelPropertySubscriptionNotification" event of an "IPlantObjectPropertySet" instance.

```
void OnPlantModelPropertySubcriptionHandler(
        IPlantObjectPropertySet sender,
        IList<IPlantObjectPropertyValue> values)
```

- `sender`
  Source of the event
- `values`
  Event data as a list of the changed "IPlantObjectPropertyValue" instances.

Example

Copy code

```
public void Odk_PlantObjectGetPropertySetReadAsync()
{
    try
    {
        IPlantModel myPlantModel = runtime.GetObject<IPlantModel>();
       string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        IPlantObject plantObject = myPlantModel.GetPlantObject(strNodeName);
        System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantObject.CurrentPlantView,
        plantObject.Name));
        if (plantObject != null)
        {
            ICollection<string> PropNames = null;// get the plant objectproperties by
propeyty names
            IPlantObjectPropertySet plantObjectPropertyset =
plantObject.GetProperties(PropNames);
            if (plantObjectPropertyset != null)
            {
                 plantObjectPropertyset.OnPropertySetReadComplete +=
odkPlantModel_onReadComplete; // Read Plant
                 Object properties values asynchronously
                 plantObjectPropertyset.ReadAsync();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
public void odkPlantModel_onReadComplete(IPlantObjectPropertySet sender, UInt32
SystemError, IList<IPlantObjectPropertyValue> Values)
{
    try
    {
        foreach (var value in Values)
        {
            Console.WriteLine("Name {0}", value.Name);
            Console.WriteLine("TimeStamp {0}", value.TimeStamp);Console.WriteLine("Value
{0}", value.Value);
            Console.WriteLine("Quality {0}", value.Quality);Console.WriteLine("Error {0}",
value.Error);
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

Copy code

Copy code

```
public void odk_plantModelSubscribe()
{
    try
    {
        IPlantModel myPlantmodel = runtime.GetObject<IPlantModel>();
       string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        IPlantObject plantObject = myPlantmodel.GetPlantObject(strNodeName);
        System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantObject.CurrentPlantView,
        plantObject.Name));
        if (plantObject != null)
        {
            ICollection<string> PropNames = null;
            IPlantObjectPropertySet plantObjectPropertyset =
plantObject.GetProperties(PropNames);
            if (plantObjectPropertyset != null)
            {
                // Assign callback function
                plantObjectPropertyset.OnPlantModelPropertySubscriptionNotification +=
                odkPlantModelPropertySet_OnDataChanged;
                plantObjectPropertyset.Subscribe();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
public void odkPlantModelPropertySet_OnDataChanged(IPlantObjectPropertySet sender,
IList<IPlantObjectPropertyValue> Values)
{
    try
    {
        foreach (var value in Values)
        {
            Console.WriteLine("Name {0}", value.Name);
            Console.WriteLine("TimeStamp {0}", value.TimeStamp);
            Console.WriteLine("Value {0}", value.Value);
            Console.WriteLine("Quality {0}", value.Quality);Console.WriteLine("Error {0}",
value.Error);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
```

Copy code

```
}
```

19.9.7.6 IPlantObjectAlarmSubscription (RT Uni)

Description

The C# interface "IPlantObjectAlarmSubscription" specifies methods for monitoring alarms of "IPlantObject" instances.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Member

The following properties, methods and events are specified in the interface:

"Filter" property

SQL-type string for filtering the result set of active alarms.

```
string Filter { get; set; }
```

All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 1084).

"Language" property

Country identifier of the language of the monitored alarms. See also section Locale IDs of the supported languages (Page 1085).

```
UInt32 Language { get; set; }
```

"IncludeChildren" property

If "true" is transferred, the alarm subscription only applies to the alarms of the "IPlantObject" instance and all its children in the hierarchy. If "false" is transferred, it only applies for the alarms of the "IPlantObject" instance.

```
bool IncludeChildren { get; set; }
```

"Start" method

Subscribe systems for monitoring of changes of active alarms.

```
void Start()
```

"Stop" method

Unsubscribe monitoring of active alarms.

```
void Stop()
```

"OnPlantObjectSubscribeAlarmHandler" event

Declares the event for the monitoring of alarms of an "IPlantObject" instance.

The event calls an instance of the "OnPlantObjectSubscribeAlarmHandler" delegate.

```
event OnPlantObjectSubscribeAlarmHandler
OnPlantObjectSubscribeAlarmHandler;
```

"OnPlantObjectSubscribeAlarmHandler" delegate

Specifies the signature of the event handling method for the "OnPlantObjectSubscribeAlarmHandler" event of an "IPlantObject" instance.

```
public delegate void OnPlantObjectSubscribeAlarmHandler(
        IPlantObjectAlarmSubscription sender,
        UInt32 systemError,
        string systemName,
        IList<IAlarmResult> values);
```

- `sender`
  Source of the event
- `systemError`
  Supplies an error code when a global error has occurred. When the error code is set, `values` is irrelevant.
- `systemName`
  Name of the Runtime system that is subscribed for alarm monitoring by the user.
- `values`
  Event data as a list of "IAlarmResult" instances of the monitored active alarm.

Example

Copy code

```
public void Odk_GetAlarmSubscription()
{
    try
    {
        using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
        {
            string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for
specified Node path
            IPlantObject plantobject = myPlantModel.GetPlantObject(strNodeName);
           IPlantObjectAlarmSubscription alarmsub = plantobject.CreateAlarmSubscription();
  //Assign alarm handler
            if (alarmsub != null)
            {
                alarmsub.OnPlantObjectSubscribeAlarmHandler +=
alarm_OnPlantObjectSubscribeAlarmHandler;
                alarmsub.Filter = "";
                alarmsub.Language = 1033;
                alarmsub.IncludeChildren = false;
                alarmsub.Start();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
public void alarm_OnPlantObjectSubscribeAlarmHandler(IPlantObjectAlarmSubscription sender,
UInt32 nGlobalError, String systemName, IList<IAlarmResult> value)
{
    try
    {
        foreach (var item in value)
        {
            System.Console.WriteLine(string.Format("Name: {0}", item.Name));
            System.Console.WriteLine(string.Format("InstanceID: {0}", item.InstanceID));
          System.Console.WriteLine(string.Format("AlarmClass: {0}", item.AlarmClassName));
            System.Console.WriteLine(string.Format("AlarmParameterValues: {0}",
item.AlarmParameterValues));
            System.Console.WriteLine(string.Format("AlarmText1: {0}", item.AlarmText1));
            System.Console.WriteLine(string.Format("Area: {0}", item.Area));
        }
    }
    finally
    {
        if (null != sender )
        {
            sender.Stop();
        }
    }
}
```

19.9.8 Interfaces of the Calendar option (RT Uni)

19.9.8.1 ISHCCalendar (RT Uni)

Description

The C# interface "ISHCCalendar" specifies the properties and methods of a calendar. The calendar is integrated via an "IPlantObject" instance.

The interface inherits the "Dispose() method of the "IDisposable" interface of the .NET framework and the methods of the "ISHCGetObject" interface

Members

"Settings" property

Reference to an "ISHCCalenderSettings" instance which saves the settings of the calendar.

```
ISHCCalendarSettings Settings { get; }
```

"Category" property

Reference to an "ISHCCategoryProvider" instance with which you access categories of the calendar.

```
ISHCCategoryProvider Category { get; }
```

"DayTemplate" property

Reference to an "ISHCDayTemplatesProvider" instance with which you create, read, update and delete day templates for the calendar.

```
ISHCDayTemplatesProvider DayTemplate { get; }
```

"ShiftTemplate" property

Reference to an "ISHCShiftTemplatesProvider" instance with which you create, read, update and delete shift templates for the calendar.

```
ISHCShiftTemplatesProvider ShiftTemplate { get; }
```

"ActionTemplate" property

Reference to an "ISHCActionTemplatesProvider" instance with which you create, read, update and delete action templates for the calendar.

```
ISHCActionTemplatesProvider ActionTemplate { get; }
```

"Day" property

Reference to an "ISHCDayProvider" instance with which you create, read, update and delete days for the calendar.

```
ISHCDayProvider Day { get; }
```

Example

The following example serves as a basis for the other examples for the C# interfaces of the Calendar option.

It shows how you can obtain the "IPlantObject" instance and also an "ISHCCalendar" instance. The "ISHCCalendar" instance referenced via `calendar` is also used in the other examples.

Copy code

```
using Siemens.Runtime.HmiUnified.SHC;
using Siemens.Runtime.HmiUnified;

ISHCCalendar calendar = null;
// Connect to Runtime
IRuntime runtime = Runtime.Connect();
if (runtime != null)
{
    IPlantModel myPlantModel = runtime.GetObject<IPlantModel>();
    IPlantObject po = myPlantModel.GetPlantObject(".hierarchy::Plant/Unit1");
    if (po != null)
    {
        calendar = po.Calendar();
    }
}
```

19.9.8.2 ISHCCategory (RT Uni)

Description

The C# interface "ISHCCategory" specifies the properties and methods of a time category of the time model.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the category

```
string Name { get; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the category and its language code ID.

```
IDictionary<UInt32, string> Descriptions { get;  }
```

"Color" property

The color of the category

```
Color Color { get; }
```

"Deleted" property

Saves the information on whether an already used category was deleted in Engineering.

```
bool Deleted { get; }
```

Example

```
static void PrintCategory(ISHCCategory category)
{
     if (null != category)
     {
          Console.WriteLine(" \nName:{0} \nColor:{1} \n Deleted:{2}\n", category.Name,
category.Color, category.Deleted);
     }
     IDictionary<uint, string> displayNames = category.DisplayNames;
     foreach (var item in displayNames)
     {
          Console.WriteLine("Language:{0} DisplayName:{1}", item.Key, item.Value);
     }
     IDictionary<uint, string> description = category.Descriptions;
     foreach (var item in description)
     {
          Console.WriteLine("Language:{0} Description:{1}", item.Key, item.Value);
     }
}
```

19.9.8.3 ISHCCategoryProvider (RT Uni)

Description

The C# interface "ISHCCategoryProvider" provides you with read access to the "ISHCCategory" instances of an "ISHCCalendar" instance.

Members

"Browse" method

Supplies a collection with the categories of the calendar.

```
IReadOnlyCollection<ISHCCategory> Browse();
```

Example

Copy code

```
IReadOnlyCollection<ISHCCategory> categories = calendar.Category.Browse();
foreach (var cat in categories)
{
    // do something
}
```

19.9.8.4 ISHCCalendarSettings (RT Uni)

Description

The C# interface "ISHCCalendarSettings" specifies properties and methods for access to the calendar settings of an "ISHCCalendar" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"PlantObject" property

Reference to the "IPlantObject" instance to which the calendar belongs.

```
string PlantObject { get; }
```

"FirstDayOfWeek" property

Reference to the "ShcWeekDay" instance which is set as the first day of the week.

```
ShcWeekDay FirstDayOfWeek { get; }
```

"FirstWeekOfYear" property

Reference to the "ShcWeekStart" instance which is set as the first week of the year.

```
ShcWeekStart FirstWeekOfYear { get; }
```

"FiscalYearStartDay" property

The first day of the fiscal year

Default setting: 1

```
Byte FiscalYearStartDay { get; }
```

"FiscalYearStartMonth" property

The first month of the fiscal year

Default setting: 1

```
Byte FiscalYearStartMonth { get; }
```

"DayOffset" property

The offset with which the workday begins, calculated from midnight.

Default setting: 0

Maximum value: 24 hours

```
TimeSpan DayOffset { get; }
```

"Workdays" property

Number of workdays

```
UInt32 Workdays { get; }
```

"TimeZone" property

The Microsoft time zone

```
UInt32 TimeZone { get; }
```

Example

```
static void PrintCalendar(ISHCCalendarSettings calendar)
{
     if (null != calendar)
     {
          string cal = string.Format(" \n Workdays: {0} \n FirstDayOfWeek: {1} \n
FirstWeekOfYear: {2} \n FiscalYearStartDay: {3} \n FiscalYearStartMonth: {4} \n DayOffset:
{5} \n PlantObject: {6} \n \n TimeZone:{7} \n", calendar.Workdays, calendar.FirstDayOfWeek,
calendar.FirstWeekOfYear, calendar.FiscalYearStartDay, calendar.FiscalYearStartMonth,
calendar.DayOffset, calendar.PlantObject, calendar.TimeZone);
          Console.WriteLine(cal);
     }
}
```

19.9.8.5 ISHCTimeSlice (RT Uni)

Description

The C# interface "ISHCTimeSlice" specifies the properties and methods of a time slice.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"StartTime" property

Time stamp with the start time of the time slice

```
DateTime StartTime { get; set; }
```

"Duration" property

The duration of the time slice

```
TimeSpan Duration { get; set; }
```

"Category" property

The time category of the time slice

```
string Category { get; set; }
```

19.9.8.6 ISHCDay (RT Uni)

Description

The C# interface "ISHCDay" specifies the properties and methods of a day.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Comments" property

A dictionary from UInt32/string pairs with the comments of the "ISHCDay" instance and their language code IDs.

```
IDictionary<UInt32, string> Comments { get; }
```

"StartTime" property

Time stamp with the start time of the "ISHCDay" instance.

```
DateTime StartTime { get; set; }
```

"IsCustomized" property

Saves information on whether the "ISHCDay" instance was edited by users.

```
bool IsCustomized { get; }
```

"DayTemplate" property

The "ISHCDayTemplate" instance from which the "ISHCDay" instance is derived.

```
string DayTemplate { get; set; }
```

"CreateShift" method

Instantiates an "ISHCShift" instance at the "ISHCDay" instance.

```
ISHCShift CreateShift(
       ISHCShiftTemplate shcShiftTemplate,
       TimeSpan startTime);
```

- `shcShiftTemplate`
  Reference to the shift template from which the shift is derived
- `startTime`
  Time stamp with the start time of the "ISHCShift" instance.

"DeleteShift" method

Deletes a shift of the "ISHCDay" instance.

```
void DeleteShift(
       ISHCShift shcShift);
```

- `shcShift`
  Reference to the shift to be deleted

"GetShifts" method

Supplies a list with all the shifts of the "ISHCDay" instance.

```
IReadOnlyList<ISHCShift> GetShifts();
```

"SetComment" method

Adds a new entry to the dictionary of the "Comment" property.

```
void SetComment(
       UInt32 languageId,
       string comment);
```

- `languageId`
  The language code ID of the comment
- `comment`
  A comment

19.9.8.7 ISHCDayProvider (RT Uni)

Description

The C# interface "ISHCDayProvider" provides you with access to the days of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete days.

Members

"Browse" method

Supplies a collection with the "ISHCDay" instances of the calendar.

```
IReadOnlyCollection<ISHCDay> Browse(
       DateTime startTime, DateTime end);
```

- `startTime`
  Defines the start of the time period whose days are returned.
- `end`
  Defines the end of the time period whose days are returned.

Example:

```
static void ReadDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(3);
        IReadOnlyCollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            foreach (var day in days)
            {
                // PrintDays(day);
                if (null != day)
               { string strDays = string.Format("\n StartTime :{0} \n IsCustomized :{1} \n
DayTemplate :{2} ", day.StartTime,                 day.IsCustomized, day.DayTemplate);
                Console.WriteLine(strDays);
                IDictionary<uint, string> Comments = day.Comments;foreach (var item in
Comments)
                {Console.WriteLine("\n Language:{0} DayComment:{1} \n", item.Key,
item.Value);  }
                }

            }
        }
    }
    catch (OdkException ex)
    {
       System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Create" method

Adds new "ISHCDay" instances to the calendar.

```
void  Create(
       IList<ISHCDay> days);
```

- `days`
  List with the new "ISHCDay" instances

Example:

```
static void CreateDayWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> Daytemplates =
calendar.DayTemplate.Browse(false);
        if (Daytemplates.Count > 0)
        {
            ISHCDayTemplate dayTemplate = Daytemplates.ElementAt(0);
            List<ISHCDay> DayList = new List<ISHCDay>();
            ISHCDay day = calendar.GetObject<ISHCDay>();
            day.DayTemplate = dayTemplate.Name;
            DateTime dtday = DateTime.Now;
            day.StartTime = dtday;
            day.SetComment(1033, "DaywithShift");
            DayList.Add(day);
            calendar.Day.Create(DayList);
            IReadOnlyCollection<ISHCShiftTemplate> ShiftTemplates =
calendar.ShiftTemplate.Browse(false);
            if (ShiftTemplates.Count > 0)
            {
                ISHCShiftTemplate ShiftTemplate = ShiftTemplates.ElementAt(0);
              ISHCShift dayShift = day.CreateShift(ShiftTemplate, new TimeSpan(18, 0, 0));
            }
        }
    }
    catch (OdkException ex)
    {
       System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Update" method

Updates "ISHCDay" instances of the calendar.

```
void Update(
         IList<ISHCDay> days);
```

- `days`
  List of the "ISHCDay" instances to be updated

Example:

```
static void UpdateDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(1);
        IReadOnlyCollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            List<ISHCDay> list = new List<ISHCDay>();
            foreach (var day in days)
            {
                IReadOnlyCollection<ISHCShift> shifts = day.GetShifts();
                if (shifts != null)
                {
                    ISHCShift shift = shifts.ElementAt(0);
                    shift.GetTimeSlices().ElementAt(0).Category = "Maintenance";
                }
                list.Add(day);
            }
            calendar.Day.Update(list);
        }
    }
    catch (OdkException ex)
    {
       System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Delete" method

Deletes "ISHCDay" instances of the calendar.

```
void Delete(
      IList<ISHCDay> days);
```

- `days`
  List of "ISHCDay" instances to be deleted

Example:

```
static void DeleteDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(3);
        IReadOnlyCollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            List<ISHCDay> list = new List<ISHCDay>();
            foreach (var day in days)
            {
                list.Add(day);
            }
            calendar.Day.Delete(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

19.9.8.8 ISHCDayTemplate (RT Uni)

Description

The C# interface "ISHCDayTemplate" specifies the properties and methods of a day.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCDayTemplate" instance.

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCDayTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCDayTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"Deleted" property

Saves information on whether the day template was deleted by users.

```
bool Deleted { get; }
```

"SetDisplayName" method

Sets the display name of the "ISHCDayTemplate" instance and its language code ID.

```
void SetDisplayName(
       UInt32 languageId,
       string displayName);
```

- `languageId`
  The language code ID of the display name
- `displayName`
  The display name

"SetDescription" method

Sets the description of the "ISHCDayTemplate" instance and its language code ID.

```
 void SetDescription(
       UInt32 languageId,
       string description);
```

- `languageId`
  The language code ID
- `description`
  The description

"GetShifts" method

Supplies a collection with the shifts of the "ISHCDayTemplate" instances.

```
IReadOnlyList<ISHCShift> GetShifts();
```

"CreateShift" method

Adds a shift to the "ISHCDayTemplate" instance.

```
ISHCShift CreateShift(
       ISHCShiftTemplate template,
       TimeSpan startTime);
```

- `template`
  Reference to the shift template on which the shift is based.
- `startTime`
  Time stamp with the start time of the shift

"DeleteShift" method

Deletes a shift of the "ISHCDayTemplate" instance.

```
void DeleteShift(
       ISHCShift shift);
```

- `shift`
  Reference to the shift to be deleted

19.9.8.9 ISHCDayTemplatesProvider (RT Uni)

Description

The C# interface "ISHCDayTemplatesProvider" provides you with access to the day templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete day templates.

Members

"Browse" method

Supplies a collection with the "ISHCDayTemplate" instances of the calendar.

```
IReadOnlyCollection<ISHCDayTemplate> Browse(
       bool includeDeleted);
```

- `includeDeleted`
  Saves information on whether the collection also contains the deleted day templates.

Example:

```
static void ReadDayTemplateWithShift()
{
    try
    {
         IReadOnlyCollection<ISHCDayTemplate> dayTemplate =
calendar.DayTemplate.Browse(false);
         if (dayTemplate.Count > 0)
             {
                 foreach (var template in dayTemplate)
                 {
                      PrintDayTemplates(template);
                      ReadShiftsforDayTemplate(template);
                 }
             }
     }
     catch (OdkException ex)
     {
         System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

"Create" method

Adds new "ISHCDayTemplate" instances to the calendar.

```
void  Create(
       ICollection<ISHCDayTemplate> dayTemplates);
```

- `dayTemplates`
  Collection with the new "ISHCDayTemplate" instances

Example:

```
static void CreateDayTemplateWithShift()
{
    try
    {
        ISHCDayTemplate Daytemplate = calendar.GetObject<ISHCDayTemplate>();
        if (null != Daytemplate)
        {
            List<ISHCDayTemplate> ListDayTemplate = new List<ISHCDayTemplate>();
            Daytemplate.Name = "DayTemplateName";
            Daytemplate.SetDescription(1033, "DayTemplateDescription");
            Daytemplate.SetDisplayName(1033, "DayTemplateDisplayName");
            ListDayTemplate.Add(Daytemplate);
            calendar.DayTemplate.Create(ListDayTemplate);
            IReadOnlyCollection<ISHCShiftTemplate> ShiftTemplates =
calendar.ShiftTemplate.Browse(false);
            if (ShiftTemplates.Count > 0)
            {
                ISHCShiftTemplate ShiftTemplate = ShiftTemplates.ElementAt(0);
               ISHCShift shift = Daytemplate.CreateShift(ShiftTemplate, new TimeSpan(1, 0,
0));
            }
        }
    }
    catch (OdkException ex)
    {
       System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Update" method

Updates the "ISHCDayTemplate" instances of the calendar.

```
void Update(
        ICollection<ISHCDayTemplate> dayTemplates);
```

- `dayTemplates`
  Collection with the "ISHCDayTemplate" instances to be updated

Example:

```
static void UpdateDayTemplateWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> dayTemplate =
calendar.DayTemplate.Browse(false);
        if (dayTemplate.Count > 0)
        {
            List<ISHCDayTemplate> list = new List<ISHCDayTemplate>();
            foreach (var dayTemplates in dayTemplate)
            {
                dayTemplates.Name = "UpdatedDayTemplate";
                dayTemplates.SetDisplayName(1033, "UpdatedDayTemplateDisplayName");
                dayTemplates.SetDescription(1033, "UpdatedDayTemplateDescription");
                IReadOnlyCollection<ISHCShift> shifts = dayTemplates.GetShifts();
                if (shifts != null)
                {
                    ISHCShift shift = shifts.ElementAt(0);
                    shift.Duration = new TimeSpan(6, 0, 0);
                }
                list.Add(dayTemplates);
            }
            calendar.DayTemplate.Update(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Delete" method

Deletes "ISHCDayTemplate" instances of the calendar.

```
void Delete(
      ICollection<ISHCDayTemplate> dayTemplates);
```

- `dayTemplates`
  Collection with the "ISHCDayTemplate" instances to be deleted

Example:

```
static void DeleteDayTemplateWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> dayTemplates =
calendar.DayTemplate.Browse(false);
        if (dayTemplate.Count > 0)
        {
            List<ISHCDayTemplate> list = new List<ISHCDayTemplate>();
            foreach (var dayTemplate in dayTemplates)
            {
                list.Add(dayTemplate);
            }
            calendar.DayTemplate.Delete(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

19.9.8.10 ISHCShiftTemplate (RT Uni)

Description

The C# interface "ISHCShiftTemplate" specifies the properties and methods of a shift.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCShiftTemplate" instance

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCShiftTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"Deleted" property

Saves information on whether the shift template was deleted by users.

```
bool Deleted { get; }
```

"Duration" property

The duration of the "ISHCShiftTemplate" instance.

```
TimeSpan Duration { get; set; }
```

"SetDisplayName" method

Sets the display name of the "ISHCShiftTemplate" instance and its language code ID.

```
void SetDisplayName(
       UInt32 languageId,
       string displayName);
```

- `languageId`
  The language code ID of the display name
- `displayName`
  The display name

"SetDescription" method

Sets the description of the "ISHCShiftTemplate" instance and its language code ID.

```
 void SetDescription(
       UInt32 languageId,
       string description);
```

- `languageId`
  The language code ID
- `description`
  The description

"GetTimeSlices" method

Supplies a list with the time slices of the "ISHCShiftTemplate" instance.

```
IReadOnlyList<ISHCTimeSlice> GetTimeSlices();
```

"CreateTimeSlice" method

Adds a time slice to the "ISHCShiftTemplate" instance.

```
void CreateTimeSlice(
       ISHCTimeSlice slice);
```

- `slice`
  Reference to the new time slice

"DeleteTimeSlice" method

Deletes a time slice of the "ISHCShiftTemplate" instance.

```
void DeleteTimeSlice(
       ISHCTimeSlice slice);
```

- `slice`
  Reference to the time slice to be deleted

19.9.8.11 ISHCShiftTemplatesProvider (RT Uni)

Description

The C# interface "ISHCShiftTemplatesProvider" provides you with access to the shift templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete shift templates.

Members

"Browse" method

Supplies a collection with the "ISHCShiftTemplate" instances of the calendar.

```
IReadOnlyCollection<ISHCShiftTemplate> Browse(
       bool includeDeleted);
```

- `includeDeleted`
  Saves information on whether the collection also contains the deleted shift templates.

Example:

```
static void ReadShiftTemplatesWithTimeslice()
{
      try
      {
            Console.WriteLine("ReadShiftTemplate With Timeslice");
            IReadOnlyCollection<ISHCShiftTemplate> template =
calendar.ShiftTemplate.Browse(false);
            if (template.Count > 0)
            {
                  foreach (var shift in template)
                  {
                        PrintShiftTemplates(shift);
                        ReadTimeslicesforShiftTemplate(shift);
                  }
            }
      }
      catch (OdkException ex)
      {
            System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
      }
}
```

"Create" method

Adds new "ISHCShiftTemplate" instances to the calendar.

```
void Create(
       ICollection<ISHCShiftTemplate> shiftTemplates);
```

- `shiftTemplates`
  Collection with the new "ISHCShiftTemplate" instances of the calendar.

Example:

```
static void CreateShiftTemplateWithTimeSlice()
{
      try
      {
         using (ISHCShiftTemplate pShiftTemplate = calendar.GetObject<ISHCShiftTemplate>())
            {
                pShiftTemplate.Name = "ShiftTemplateName";
                pShiftTemplate.SetDisplayName(1033, "ShiftTemplateDisplayName");
                pShiftTemplate.SetDescription(1033, "ShiftTemplateDescriptions");
                pShiftTemplate.Duration = new TimeSpan(8, 0, 0);
                List<ISHCShiftTemplate> ShiftList = new List<ISHCShiftTemplate>();
                ShiftList.Add(pShiftTemplate);
                calendar.ShiftTemplate.Create(ShiftList);
                IReadOnlyCollection<ISHCCategory> categories = calendar.Category.Browse();
                if (categories.Count > 0)
                {
                    ISHCCategory pCat = categories.ElementAt(0);
                    ISHCTimeSlice pTimeSlice = calendar.GetObject<ISHCTimeSlice>();
                    pTimeSlice.StartTime = DateTime.Now.StartOfDay();
                    pTimeSlice.Duration = new TimeSpan(3, 0, 0);
                    pTimeSlice.Category = pCat.Name;
                    pShiftTemplate.CreateTimeSlice(pTimeSlice);
                }
            }
      }
      catch (OdkException ex)
      {
            System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
      }
}
```

"Update" method

Updates "ISHCShiftTemplate" instances of the calendar.

```
void Update(
       ICollection<ISHCShiftTemplate> shiftTemplates);
```

- `shiftTemplates`
  Collection with the "ISHCShiftTemplate" instances to be updated

Example:

```
static void UpdateShiftTemplateWithTimeslice()
{
      try
      {
            IReadOnlyCollection<ISHCShiftTemplate> shiftTemplates = 
calendar.ShiftTemplate.Browse(false);
            List<ISHCShiftTemplate> list = new List<ISHCShiftTemplate>();
            IReadOnlyCollection<ISHCCategory> categories = calendar.Category.Browse();
            ISHCCategory pCat = categories.ElementAt(1);
            foreach (var shifttemplate in shiftTemplates)
            {
                  shifttemplate.Name = "UpdateShiftTemplate";
                  shifttemplate.SetDisplayName(1033, "Updated DisplayName");
                  shifttemplate.SetDescription(1033, "UpdatedDescription");
                  shifttemplate.Duration = new TimeSpan(10, 0, 0);
                  list.Add(shifttemplate);
                  IReadOnlyCollection<ISHCTimeSlice> Timeslices = 
shifttemplate.GetTimeSlices();
                  if (Timeslices.Count > 0)
                  {
                        ISHCTimeSlice timeslice = Timeslices.ElementAt(0);
                        timeslice.Duration = new TimeSpan(5, 0, 0);
                        timeslice.Category = pCat.Name;
                  }
            }
            calendar.ShiftTemplate.Update(list);
      }
      catch (OdkException ex)
      {
            System.Console.WriteLine(string.Format("OdkException occurred {0}", 
ex.Message));
      }
}
```

"Delete" method

Deletes "ISHCShiftTemplate" instances of the calendar.

```
void Delete(
      ICollection<ISHCShiftTemplate> shiftTemplates);
```

- `shiftTemplates`
  Collection with the "ISHCShiftTemplate" instances of the calendar to be deleted.

Example:

```
static void DeleteShiftTemplateWithTimeslice()
{
      try
      {
            IReadOnlyCollection<ISHCShiftTemplate> shiftTemplates =
calendar.ShiftTemplate.Browse(false);
            if (shiftTemplates.Count > 0)
            {
                  List<ISHCShiftTemplate> list = new List<ISHCShiftTemplate>();
                  foreach (var shifttemplate in shiftTemplates)
                  {
                        list.Add(shifttemplate);
                  }
                  calendar.ShiftTemplate.Delete(list);
            }
      }
      catch (OdkException ex)
      {
            System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
      }
}
```

19.9.8.12 ISHCShift (RT Uni)

Description

The C# interface "ISHCShift" specifies the properties and methods of a shift.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"StartTime" property

Time stamp with the start time of the "ISHCShift" instance.

```
DateTime StartTime { get; set; }
```

"Duration" property

The duration of the "ISHCShift" instance.

```
TimeSpan Duration { get; set; }
```

"ShiftTemplate" property

The shift template of the "ISHCShift" instance.

```
string ShiftTemplate { get; }
```

"IsCustomized" property

Saves information on whether the "ISHCShift" instance was edited by users.

```
bool IsCustomized { get; }
```

"DeltaKind" property

Saves information on how the time slices of the "ISHCShift" instance deviate from the shift template.

```
ShcDeltaType DeltaKind { get; }
```

The enumeration "ShcDeltaType" can contain the following values:

- `Added` (0)
- `Modified` (1)
- `Deleted` (2)

"ShiftId" property

Saves the ShiftID of the "ISHCShift" instance.

```
UInt32 ShiftId { get; set; }
```

"Comments" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCShift" instance and their language code IDs.

```
IDictionary<UInt32, string> Comments { get; }
```

"GetTimeSlices" method

Supplies a collection with the time slices of the "ISHCShift" instances.

```
IReadOnlyList<ISHCTimeSlice> GetTimeSlices();
```

"CreateTimeSlice" method

Adds a time slice to the "ISHCShift" instance.

```
void CreateTimeSlice(
       ISHCTimeSlice slice);
```

- `slice`
  Reference to the new time slice

"DeleteTimeSlice" method

Deletes a time slice of the "ISHCShift" instance.

```
void DeleteTimeSlice(
       ISHCTimeSlice slice);
```

- `slice`
  Reference to the time slice to be deleted

"SetComment" method

Adds a comment with language code ID to the "Comments" property.

```
void SetComment(
      UInt32 languageId,
      string comment);
```

- `languageId`
  The language code ID of the comment
- `comment`
  The comment

"CreateAction" method

Adds an action to the "ISHCShift" instance.

```
ISHCAction CreateAction(
      ISHCActionTemplate actionTemplate,
      TimeSpan offset);
```

- `actionTemplate`
  The action template of the new action
- `offset`
  The offset for the anchor point of the action, in relation to the starting point of the shift. Positive and negative value allowed.

Example:

```
static void CreateActionUsingShift()
{
     try
     {
          DateTime start = DateTime.Now.StartOfDay();
          DateTime end = DateTime.Now.EndOfDay();
          end = end.AddDays(3);
          ISHCShift Shift = calendar.Day.Read(start,
end).ElementAt(0).GetShifts().ElementAt(0);
          if (Shift != null)
          {
               ISHCAction Action =
Shift.CreateAction(calendar.ActionTemplate.Read(false).ElementAt(0), new TimeSpan(5, 0,
0));
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

"DeleteAction" method

Deletes an action of the "ISHCShift" instance.

```
void DeleteAction(ISHCAction shcAction);
```

- `shcAction`
  Reference to the action to be deleted

"GetActions" method

Supplies a list with the actions of the "ISHCShift" instance.

```
IReadOnlyList<ISHCAction> GetActions();
```

Example:

```
static void ReadActionUsingShift()
{
     try
     {
          DateTime start = DateTime.Now.StartOfDay();
          DateTime end = DateTime.Now.EndOfDay();
          end = end.AddDays(3);
          ISHCShift Shift = calendar.Day.Read(start,
end).ElementAt(0).GetShifts().ElementAt(0);
          if (Shift != null)
          {
               IReadOnlyList<ISHCAction> action = Shift.GetActions();
               if (action != null)
               {
                    ISHCAction actions = action.ElementAt(0);
                    string Action = string.Format("\n Offset:{0} \n IsCustomized:
{1} ,actionTemplate:{2}", actions.Offset, actions.IsCustomized, actions.ActionTemplate);
                    System.Console.WriteLine(Action);
               }
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

19.9.8.13 ISHCAction (RT Uni)

Description

The C# interface "ISHCAction" specifies the properties and methods of an action.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Offset" property

The offset for the anchor point of the "ISHCAction" instance in 100 nanoseconds in relation to the start point of its shift instance. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"ActionTemplate" property

The action template of the "ISHCAction" instance

```
string ActionTemplate { get; }
```

"IsCustomized" property

Saves information on whether the "ISHCAction" instance was edited by users.

```
bool IsCustomized { get; }
```

"GetElements" method

Supplies a list with the action elements of the "ISHCAction" instance.

```
IReadOnlyList<ISHCActionElement> GetElements();
```

19.9.8.14 ISHCActionElement (RT Uni)

Description

The C# interface "ISHCActionElement" specifies the properties and methods of an action element of an "ISHCAction" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"ElementType" property

The type of the "ISHCActionElement" instance.

```
ShcActionElementType ElementType { get; }
```

The enumeration "ShcActionElementType" can contain the following values:

- `Tag` (0)
  The action element controls a tag.

"Enabled" property

Saves the information on whether the "ISHCActionElement" instance is activated.

```
bool Enabled { get; set; }
```

"Offset" property

The offset of the "ISHCActionElement" instance in 100 nanoseconds in relation to the anchor point of its action. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"Value" property

Value of the tag controlled by the "ISHCActionElement" instance

```
object Value { get; set; }
```

"ElementName" property

Name of the tag controlled by the "ISHCActionElement" instance

```
string ElementName { get; set; }
```

19.9.8.15 ISHCActionTemplate (RT Uni)

Description

The C# interface "ISHCActionTemplate" specifies the properties and methods of the action template of an "ISHCAction" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCActionTemplate" instance.

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCActionTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Deleted" property

Saves information on whether the action template was deleted by users.

```
bool Deleted { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCActionTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"SetDisplayName" method

Sets the display name of the "ISHCActionTemplate" instance and its language code ID.

```
void SetDisplayName(
       UInt32 languageId,
       string displayName);
```

- `languageId`
  The language code ID of the display name
- `displayName`
  The display name

"SetDescription" property

Sets the description of the "ISHCActionTemplate" instance and its language code ID.

```
void SetDescription(
       UInt32 languageId,
       string description);
```

- `languageId`
  The language code
- `IDdescription`
  The description

"CreateElement" property

Adds an "ISHCActionTemplateElement" instance to the "ISHCActionTemplate" instance.

```
void CreateElement(
       ISHCActionTemplateElement actiontemplateElement);
```

- `actiontemplateElement`
  Reference to the new action template element

"DeleteElement" property

Deletes an "ISHCActionTemplateElement" instance of the "ISHCActionTemplate" instance.

```
void DeleteElement(
       ISHCActionTemplateElement actiontemplateElement);
```

- `actiontemplateElement`
  Reference to the action template element to be deleted

"GetElements" property

Supplies a list with action template elements of the "ISHCActionTemplate" instances.

```
IReadOnlyList<ISHCActionTemplateElement> GetElements();
```

19.9.8.16 ISHCActionTemplateElement (RT Uni)

Description

The C# interface "ISHCActionTemplateElement" specifies the properties and methods of an action element of an "ISHCActionTemplate" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"ElementType" property

The type of the "ISHCActionTemplateElement" instance.

```
ShcActionElementType ElementType { get; }
```

The enumeration "ShcActionElementType" can contain the following values:

- `Tag` (0)
  The action element controls a tag.

"Offset" property

The offset of the "ISHCActionTemplateElement" instance in 100 nanoseconds in relation to the anchor point of its action template. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"Value" property

Value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
object Value { get; set; }
```

"ElementName" property

Name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
string ElementName { get; set; }
```

19.9.8.17 ISHCActionTemplatesProvider (RT Uni)

Description

The C# interface "ISHCActionTemplatesProvider" provides you with access to the action templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete action templates.

Members

"Browse" method

Supplies a collection with the "ISHCActionTemplate" instances of the calendar.

```
IReadOnlyCollection<ISHCActionTemplate> Browse(
       bool includeDeleted);
```

- `includeDeleted`
  Saves information on whether the collection also contains the deleted action templates.

Example:

```
static void ReadActionTemplate()
{
     try
     {
          Console.WriteLine("ReadActionTemplate");
          IReadOnlyCollection<ISHCActionTemplate> actionTemplate =
calendar.ActionTemplate.Browse(false);
          foreach (var template in actionTemplate)
          {
               PrintActionTemplates(template);
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

"Create" method

Adds new "ISHCActionTemplate" instances to the calendar.

```
void  Create(
       ICollection<ISHCActionTemplate> actionTemplates);
```

- `actionTemplates`
  Collection with the new "ISHCActionTemplate" instances

Example:

```
static void CreateActionTemplateWithActionTemplateElement()
{
     try
     {
          ISHCActionTemplate pActionTemplate = calendar.GetObject<ISHCActionTemplate>();
          if (pActionTemplate != null)
          {
               pActionTemplate.Name = "ActionTemplate";
               pActionTemplate.SetDisplayName(1033, "ActionDisplayName");
               pActionTemplate.SetDescription(1033, "ActionDescription");
               List<ISHCActionTemplate> ActionList = new List<ISHCActionTemplate>();
               ActionList.Add(pActionTemplate);
               calendar.ActionTemplate.Create(ActionList);
               ISHCActionTemplateElement pActionTemplateElement =
calendar.GetObject<ISHCActionTemplateElement>();
               if (pActionTemplateElement != null)
               {
                    pActionTemplateElement.ElementName = "HMI_RT_1::Unit1.Member_1";
                    pActionTemplateElement.Value = false;
                    pActionTemplateElement.Offset = new TimeSpan(4, 0, 0);
                    pActionTemplate.CreateElement(pActionTemplateElement);
               }
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

"Update" method

Updates "ISHCActionTemplate" instances of the calendar.

```
void Update(
        ICollection<ISHCActionTemplate> actionTemplates);
```

- `actionTemplates`
  Collection with the "ISHCActionTemplate" instances to be updated

Example:

```
static void UpdateActionTemplateWithActionTemplateElement()
{
     IReadOnlyCollection<ISHCActionTemplate> actionTemplates =
calendar.ActionTemplate.Browse(false);
     List<ISHCActionTemplate> list = new List<ISHCActionTemplate>();
     foreach (var actionTemplate in actionTemplates)
     {
          actionTemplate.Name = "UpdatedActionTemplate";
          actionTemplate.SetDisplayName(1033, "UpdatedDisplayName ");
          actionTemplate.SetDescription(1033, "UpdatedDescription");
          IReadOnlyCollection<ISHCActionTemplateElement> action =
actionTemplate.GetElements();
          ISHCActionTemplateElement templateElement = action.ElementAt(0);
          templateElement.Offset = new TimeSpan(6, 0, 0);
          list.Add(actionTemplate);
     }
     calendar.ActionTemplate.Update(list);
}
```

"Delete" method

Deletes "ISHCActionTemplate" instances of the calendar.

```
void Delete(
      ICollection<ISHCActionTemplate> actionTemplates);
```

- `actionTemplates`
  Collection with the "ISHCActionTemplate" instances to be deleted

Example:

```
static void DeleteActionTemplateWithActionTemplateElement()
{
     try
     {
          IReadOnlyCollection<ISHCActionTemplate> actionTemplates =
calendar.ActionTemplate.Browse(false);
          if (actionTemplates.Count > 0)
          {
               List<ISHCActionTemplate> list = new List<ISHCActionTemplate>();
               foreach (var actionTemplate in actionTemplates)
               {
                    list.Add(actionTemplate);
               }
               calendar.ActionTemplate.Delete(list);
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

19.10 Description of the C++ interfaces (RT Uni)

19.10.1 Error codes of the C++ interfaces (RT Uni)

All methods that have defined a `CFRESULT` return `CFSUCCESS` if the method was run through successfully. Otherwise, they return a corresponding error code.

19.10.2 Interfaces of the Runtime environment (RT Uni)

19.10.2.1 IOdkRt (RT Uni)

Description

The C++ interface "IOdkRt" specifies methods for the connection to the Runtime system and the error handling.

Members

The following methods are specified in the interface:

"Connect" method

Connect to a Runtime project.

```
CFRESULT Connect(
         const CFSTR context,
         IRuntime **ppRuntime,
         const CTSTR user = nullptr,
         const CTSTR password = nullptr)
```

- `context`
  [in]: Name of the runtime project

  Note

  The name of the Runtime project is not used in the current version. An empty string must be passed in order to connect to the locally run Runtime project.

- `IRuntime`
  [out]: Points to the initialized "IRuntime" object that the ODK object model makes available.

- `user`
  [in]: User name

  Note

  Can only be used in a future version!

- `password`
  [in]: Password

  Note

  Can only be used in a future version!

"Close" method

Enable configuration files and plug-ins of the Runtime system.

```
CFRESULT Close()
```

"GetErrorHandler" method

Transfers an "IErrorInfo" object for error handling.

```
CFRESULT GetErrorHandler(IErrorInfo** pErrorInfo)
```

`IErrorInfo`
[out]: Points to an "IErrorInfo" object.

Example

Connect to the Runtime system of the active project:

Copy code

```
IRuntimePtr pRuntime;

CFRESULT Connect()
{
    // Connect to running project
    CCfString projectName = L"";
    CFRESULT retVal = Connect(projectName, &pRuntime);

    if(CF_FAILED(retVal))
        PrintErrorInformation(retVal, L"Connect", pRuntime);

    return retVal;
}
```

Error handling when reading out installed options of the Runtime system:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    //load option component by name
    CFRESULT errorCode = pRuntime->GetOption(CCfString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnknown;
    //create a instance of the option object "MyOptionObject"
    errorCode = pOdkOption->GetObject(CCfString("MyOptionObject"), &pUnknown);

    if (CF_SUCCEEDED(errorCode))
    {
        IMyOptionObjectPtr pMyOptionObject(pUnknown);
        CCfString strProperty;
        pMyOptionObject->GetMyProperty(&strProperty);
    }
    else
    {
        IErrorInfoPtr pInfo;
        if (CF_SUCCEEDED(GetErrorHandler(&pInfo)))
        {
            SCCfString errorDescription;
            //get error description
            pInfo->GetErrorDescription(errorCode, &errorDescriptionStr);
        }
    }
}
```

See also

IRuntime (Page 1234)

IErrorInfo (Page 1249)

19.10.2.2 IRuntime (RT Uni)

Description

The C++ interface "IRuntime" specifies methods for information and the addressing of the Runtime system.

Members

The following methods are specified in the interface:

"GetObject" method

Create new instance of an object of the Runtime system. Possible object types are defined in the configuration file OdkObjectModel.xml.

```
CFRESULT GetObject(const CFSTR value, ICfUnknown **ppObject)
```

- `value`
  [in]: Name of the object type, for example "Tag" for tags
- `ppObject`
  [out]: Points to the initialized object of the runtime system.

"GetProduct" method

Return an "IProduct" object that allows access to the version information and installed options of the Runtime system.

```
CFRESULT GetProduct(IProduct **ppProduct)
```

`ppProduct`
[in/out]: Points to an "IProduct" object that contains the product information of the runtime system.

"GetOption" method

Referencing installed option of the Runtime system.

```
CFRESULT GetOption(
         const CFSTR optionName,
         IOption **ppOption)
```

- `optionName`
  [in]: Name of the installed option
- `ppProduct`
  [out]: Points to an installed option of the Runtime system as "IOption" object.

"GetUserName" method

Return the name of the logged-on user.

```
CFRESULT GetUserName(CFSTR* name)
```

`name`
[out]: Displays the user name.

Example

Initialize an object of the "Tag" type of the Runtime system:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        ... //further tag processing
    }

    return errCode;
}
```

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);

    if(pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
        IVersionInfoPtr pVersion;
        pProduct->GetVersion(&pVersion);

        pVersion->GetMajor(&uintMajor);
        pVersion->GetMinor(&uintMinor);
        pVersion->GetServicePack(&uintServicePack);
        pVersion->GetUpdate(&uintUpdate);

        wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
uintServicePack << L"-" << uintUpdate << endl;
    }
}
```

Use installed options:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    CFRESULT errCode = pRuntime->GetOption(CCfString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnk;
    errCode = pOdkOption->GetObject("MyOptionObject2", &pUnk);

    if (CF_SUCCEEDED(errorCode))
    {
        IMyOptionObjectPtr pMyOptionObject(pUnknown);
        CCfString strProperty;
        pMyOptionObject->GetMyProperty(&strProperty);
    }
    else
    {
        IErrorInfoPtr pInfo;
        if (CF_SUCCEEDED(GetErrorHandler(&pInfo)))
        {
            CCfString errorDescriptionStr;
            //get error description
            pInfo->GetErrorDescription(errorCode, &errorDescriptionStr);
        }
    }

    //using extension methods for CPM node
    ICpmPtr pCpm;
    errorCode = pRuntime->GetObject(CCfString("Cpm"), (ICfUnknown**) &pCpm);

    ICpmNodePtr pCpmNode;
    CCfString strNode(".hierarchy::PlantView\\Unit1");
    errorCode = pCpm->GetNode(strNode, &pCpmNode);

    //using specific option interface
    IMyOptionPtr pMyOption(pOdkOption);

    IMyCpmNodeFormulaPtr pFormula;
    pMyOption->GetObject(pCpmNode, CCfString("Formula"), (ICfUnknown**) &pFormula);
    pFormula->SetName(CCfString("Quality"));
    int32_t result;
    pFormula->Calc(&result);
}
```

See also

IOdkRt (Page 1232)

IProduct (Page 1238)

IOption (Page 1239)

19.10.2.3 IProduct (RT Uni)

Description

The C++ interface "IProduct" specifies methods for handling product information of the Runtime system.

Members

The following methods are specified in the interface:

"GetOptions" method

Return installed options of the Runtime system as array of "IOption" objects.

```
CFRESULT GetOptions(IOptionEnumerator **ppEnumerator)
```

`ppEnumerator`
[out]: Points to the installed options as "IOptionEnumerator" object.

"GetVersion" method

Return version structure of the installed Runtime system as "IVersionInfo" object.

```
CFRESULT GetVersion(IVersionInfo** versionInfo)
```

`versionInfo`
[out]: Points to a structure with version information of the installed Runtime system.

Example

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);

    if(pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
        IVersionInfoPtr pVersion;
        pProduct->GetVersion(&pVersion);

        pVersion->GetMajor(&uintMajor);
        pVersion->GetMinor(&uintMinor);
        pVersion->GetServicePack(&uintServicePack);
        pVersion->GetUpdate(&uintUpdate);

        wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
uintServicePack << L"-" << uintUpdate << endl;
    ]
}
```

Output name of all installed options:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);
    if (pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        IOptionEnumeratorPtr pItems;
        errCode = pProduct->GetOptions(&pItems);
        if (CF_SUCCEEDED(errCode))
        {
            while (pItems->MoveNext() == CF_SUCCESS)
            {
                IOptionPtr pValue;
                pItems->Current(&pValue);
                CCfString module;
                pValue->GetName(&module);
                wcout << L"Option name: " << module << endl;
            }
        }
        else
        {
            wcout << L"No option installed." << endl;
            PrintErrorInformation(errCode, L"GetOptions", pRuntime);
        }
    }
    else
    {
        PrintErrorInformation(errCode, L"GetProduct", pRuntime);
    }
}
```

See also

IOdkRt (Page 1232)

IRuntime (Page 1234)

IOption (Page 1239)

IOptionEnumerator (Page 1241)

IVersionInfo (Page 1243)

19.10.2.4 IOption (RT Uni)

Description

The C++ interface "IOption" specifies properties and methods for handling installed product options of the Runtime system.

Members

The following methods are specified in the interface:

"GetName" method

Return name of an installed option of the Runtime system.

```
CFRESULT GetName(CFSTR *pValue)
```

`pValue`
[out]: Points to the name of an installed option of the runtime system.

"GetObject" method

Referencing installed option of the Runtime system.

```
CFRESULT GetObject(
         const CFSTR Value,
         ICfUnknown** ppObject)
```

- `Value`
  [in]: Name of the installed option of the Runtime system
- `ppObject`
  [out]: Points to the installed option of the Runtime system as an "ICfUnknown" object.

"GetVersion" method

Reference version structure of an installed option of the Runtime system as "IVersionInfo" object.

```
CFRESULT GetVersion(IVersionInfo** versionInfo)
```

`versionInfo`
[out]: Points to a structure with version information of an installed option of the Runtime system.

Example

Read out installed option:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    CFRESULT errCode = pRuntime->GetOption(CCfString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnk;
    errCode = pOdkOption->GetObject("MyOptionObject2", &pUnk);

    if (CF_SUCCEEDED(errorCode))
    {
        IMyOptionObjectPtr pMyOptionObject(pUnknown);
        CCfString strProperty;
        pMyOptionObject->GetMyProperty(&strProperty);
    }
    else
    {
        IErrorInfoPtr pInfo;
        if (CF_SUCCEEDED(GetErrorHandler(&pInfo)))
        {
            CCfString errorDescriptionStr;
            //get error description
            pInfo->GetErrorDescription(errorCode, &errorDescriptionStr);
        }
    }
}
```

See also

IProduct (Page 1238)

IOptionEnumerator (Page 1241)

IVersionInfo (Page 1243)

19.10.2.5 IOptionEnumerator (RT Uni)

Description

The "IOptionEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of installed product options of the Runtime system.

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

`CFRESULT Current(IOption **ppItem)`

`ppItem`
[out]: Points to the current "IOption" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

`CFRESULT MoveNext()`

"Reset" method

Reset the current position in the enumeration of a list.

`CFRESULT Reset()`

The "MoveNext"method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

`CFRESULT Count(uint32_t *value)`

`value`
[out]: Points to the value for the number of the elements of the list.

Example

Access the installed options "IOption" of the runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);
    if(pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        IOptionEnumeratorPtr pItems;
        if(CF_SUCCEEDED(pProduct->GetOptions(&pItems)))
        {
            while(pItems->MoveNext() == CF_SUCCESS)
            {
                IOptionPtr pValue;
                pItems->Current(&pValue);
                ...
            }
        }
        else
        {
            wcout << L"No option installed." <<endl;
        }
    }
}
```

See also

IOption (Page 1239)

19.10.2.6 IVersionInfo (RT Uni)

Description

The C++ interface "IVersionInfo" specifies methods for reading out version information of the runtime system.

Members

The following methods are specified in the interface:

"GetMajor" method

Return main version of an installed option of the Runtime system.

```
CFRESULT GetMajor(uint16_t *pValue)
```

`pValue`
[out]: Points to the main version of an installed option of the Runtime system.

"GetMinor" method

Return minor version of an installed option of the Runtime system.

```
CFRESULT GetMinor(uint16_t *pValue)
```

`pValue`
[out]: Points to the minor version of an installed option of the Runtime system.

"GetServicePack" method

Return service pack of an installed option of the Runtime system.

```
CFRESULT GetServicePack(uint16_t *pValue)
```

`pValue`
[out]: Points to the service pack of an installed option of the Runtime system.

"GetUpdate" method

Return update version of an installed option of the Runtime system.

```
CFRESULT GetUpdate(uint16_t *pValue)
```

`pValue`
[out]: Points to the update version of an installed option of the Runtime system.

Example

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
    IProductPtr pProduct;
    CFRESULT errCode = pRuntime->GetProduct(&pProduct);

    if(pProduct != NULL && CF_SUCCEEDED(errCode))
    {
        uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
        IVersionInfoPtr pVersion;
        pProduct->GetVersion(&pVersion);

        pVersion->GetMajor(&uintMajor);
        pVersion->GetMinor(&uintMinor);
        pVersion->GetServicePack(&uintServicePack);
        pVersion->GetUpdate(&uintUpdate);

        wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
uintServicePack << L"-" << uintUpdate << endl;
    ]
}
```

See also

IProduct (Page 1238)

IOption (Page 1239)

19.10.2.7 IErrorResult (RT Uni)

Description

The "IErrorResult" interface is a C++ interface that specifies methods for reading out error details.

Members

The following methods are specified in the interface:

"GetError" method

Read out error code of an error message.

`CFRESULT GetError(CFRESULT *value)`

`value`
[out]: Points to an error code.

"GetName" method

Read out name of the associated object of the data source.

```
CFRESULT GetError(CFSTR *value)
```

`value`
[out]: Points to an object name.

Example

Read out details of "IErrorResult" error messages:

Copy code

```
IErrorResultEnumerator* WriteTagSetSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
    IErrorResultEnumerator* pEnumerator = nullptr;
    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->AddWithValue(CCfString(tags[i]._tagName), tags[i]._tagValue);
        }

        errCode = pTagSet->Write(&pEnumerator);
        if (CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }

        if (pEnumerator != nullptr)
        {
            while (pEnumerator->MoveNext() == CF_SUCCESS)
            {
                IErrorResult* pValue;
                CFRESULT errorCode = pEnumerator->Current(&pValue);
                if (pValue != nullptr && CF_SUCCEEDED(errorCode))
                {
                    pValue->GetError(&errorCode);
                    CCfString str;
                    pValue->GetName(&str);

                    if (CF_FAILED(errorCode))
                    {
                     std::wcout << L"Write Tag failed, Tag name: " << str << L", ErrorCode:
" << errorCode << std::endl;
                        PrintErrorInformation(errorCode, L"Write Tag", pRuntime);
                    }
                }
            }
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
    return pEnumerator;
}
```

See also

IErrorResultEnumerator (Page 1247)

19.10.2.8 IErrorResultEnumerator (RT Uni)

Description

The "IErrorResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of error messages of the Runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IErrorResult **ppItem)
```

`ppItem`
[out]: Points to the current "IErrorResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of elements of the list.

Example

Access the "IErrorResult" error messages when writing a TagSet:

Copy code

```
IErrorResultEnumerator* WriteTagSetSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
    IErrorResultEnumerator* pEnumerator = nullptr;
    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->AddWithValue(CCfString(tags[i]._tagName), tags[i]._tagValue);
        }

        errCode = pTagSet->Write(&pEnumerator);
        if (CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }

        if (pEnumerator != nullptr)
        {
            while (pEnumerator->MoveNext() == CF_SUCCESS)
            {
                IErrorResult* pValue;
                CFRESULT errorCode = pEnumerator->Current(&pValue);
                if (pValue != nullptr && CF_SUCCEEDED(errorCode))
                {
                    pValue->GetError(&errorCode);
                    CCfString str;
                    pValue->GetName(&str);

                    if (CF_FAILED(errorCode))
                    {
                     std::wcout << L"Write Tag failed, Tag name: " << str << L", ErrorCode:
" << errorCode << std::endl;
                        PrintErrorInformation(errorCode, L"Write Tag", pRuntime);
                    }
                }
            }
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
    return pEnumerator;
}
```

See also

IErrorResult (Page 1244)

ITagSet (Page 1264)

ITagSetQCD (Page 1272)

19.10.2.9 IErrorInfo (RT Uni)

Description

The "IErrorInfo" interface is a C++ interface that specifies methods for handling error codes.

Members

The following methods are specified in the interface:

"GetErrorDescription" method

Output an error description for the error code.

You have to use the "GetErrorHandler" method to instantiate an "IErrorInfo" object beforehand.

```
CFRESULT GetErrorDescription(
        uint32_t errorCode,
        CFSTR *errorDescription)
```

- `errorCode`
  [in]: Error code that is handed over by the ODK client.
- `errorDescription`
  [out]: Points to the error description of the error code.

Example

Output object name and description of an error:

Copy code

```
void PrintErrorInformation(CFRESULT errorCode, CCfSmartString objectName, IRuntimePtr 
pRuntime)
{
    if (pRuntime != nullptr)
    {
        IErrorInfoPtr pInfo;
        CFRESULT result = pRuntime->GetObject(CCfString(L"ErrorHandler"), 
(ICfUnknown**)&pInfo);

        if (CF_FAILED(result))
        {
            std::wcout << "Error occurred: Can not create 'ErrorHandler' object " << 
std::endl;
            return;
        }

        CCfString resStr;
        result = pInfo->GetErrorDescription(errorCode, &resStr);

        if (CF_SUCCEEDED(result))
        {
            CCfSmartString errorDescription(resStr);
           std::wcout << "Error occurred: '" << errorDescription.Get() << "', ObjectName = 
" << objectName.Get() << std::endl;
        }
        else
        {
           std::wcout << "Error occurred: 'GetErrorDescription' failed, Error number: " << 
result << std::endl;
        }
    }
    else
    {
        std::wcout << "IRuntimePtr is NULL "<< std::endl;
    }
}
```

See also

IOdkRt (Page 1232)

19.10.3 Interfaces of the tags (RT Uni)

19.10.3.1 IProcessValue (RT Uni)

Description

The C++ interface "IProcessValue" specifies properties and methods for values of process tags of the Runtime system. The "IProcessValue" interface represents values from the result of a read operation or monitoring.

Members

The following methods are specified in the interface:

"GetTagName" method

Return the name of the tag.

```
CFRESULT GetTagName(CFSTR *value)
```

`value`
[out]: Points to the name of the tag belonging to the process value.

"GetValue" method

Return value of the tag at the moment of the read operation.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the process value of the tag.

"GetQuality" method

Return quality code of the read operation of the tag.

```
CFRESULT GetQuality(int32_t *value)
```

`value`
[out]: Points to the quality code of the process tag.

"GetTimeStamp" method

Return the time stamp of the last successful read operation of the tag.

```
CFRESULT GetTimeStamp(CFDATETIME64 *value)
```

`value`
[out]: Points to the time stamp of the read operation of the process tag.

"GetError" method

Return error code of the last read or write operation of the tag.

```
CFRESULT GetError(int32_t *value)
```

`value`
[out]: Points to the error code of the process tag.

Example

Read out a process tag and output the properties of the "IProcessValue" object:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        IProcessValuePtr pValue;

        // Read value of tag
        errCode = pTag->Read(&pValue);

        if (pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            CCfString timeStamp;
            CFDATETIME64 cfTimeStamp;
            pValue->GetTimeStamp(&cfTimeStamp);
            CCfDateTime64 time(cfTimeStamp);
            timeStamp = time.GetDateTimeString(false);

            CCfString strName;
            pValue->GetTagName(&strName);

            CCfVariant varValue;
            pValue->GetValue(&varValue);

           std::wcout << strName.ToUTF8().c_str() << L" " << timeStamp << L" " << L" Value:
" << (double)(varValue) << std::endl;
        }
        else
        {
            std::wcout << L"Read operation failed." << std::endl;
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
    }
    return errCode;
}
```

See also

IProcessValueEnumerator (Page 1253)

ITag (Page 1254)

ITagSet (Page 1264)

ITagSetQCD (Page 1272)

19.10.3.2 IProcessValueEnumerator (RT Uni)

Description

The "IProcessValueEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of process values of the Runtime system. The enumeration is, for example, used when reading out process values of a TagSet.

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IProcessValue **ppItem)
```

`ppItem`
[out]: Points to the current "IProcessValue" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext"method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of the elements of the list.

Example

Output process values of TagSets:

Copy code

```
...
IProcessValueEnumeratorPtr pItems;
errCode = pTagSet->Read(&pItems);

if(pItems != nullptr && CF_SUCCEEDED(errCode))
{
    std::wcout << "Read finished " << std::endl;

    // Iterate over the process value objects
    while(CF_SUCCEEDED(pItems->MoveNext()))
    {
        IProcessValuePtr pValue;
        errCode = pItems->Current(&pValue); // get current process value
        if(pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            CCfString timeStamp;
            CFDATETIME64 cfTimeStamp;
            pValue->GetTimeStamp(&cfTimeStamp);
            CCfDateTime64 time(cfTimeStamp);
            timeStamp = time.GetDateTimeString(false);
            CCfString strName;
            pValue->GetTagName(&strName);
            CCfVariant varValue;
            pValue->GetValue(&varValue);

           std::wcout << strName << L" " <<timeStamp << L" " << contextId << L" Value: " <<
(double)(varValue) << std::endl;
        }
    }
}
else
{
    std::wcout << L"Read operation failed." << std::endl;
    PrintErrorInformation(errCode, L"GetObject", pRuntime);
}
...
```

See also

IProcessValue (Page 1251)

19.10.3.3 ITag (RT Uni)

Description

The C++ interface "ITag" specifies methods for handling tags of the Runtime system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetTagName" method

Set name of the tag.

```
CFRESULT SetTagName(const CFSTR tagName)
```

`tagName`
[in]: Name of the tag

"Write" method

Write process value of the tag synchronously in the Runtime system.

```
CFRESULT Write(
         const CFVARIANT value,
         CFENUM type = HmiWriteType::NoWait)
```

- `value`
  [in]: Tag value
- `type`
  [in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType::NoWait` (default): Writes the tag value without waiting. Errors for the write operation are not detected.
  - `HmiWriteType::Wait`: Waits until the tag value is written in the AS. The associated errors are written.

"WriteQCD" method

Write process value with quality code of the tag synchronously in the Runtime system. The tag also has a freely definable time stamp. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

```
CFRESULT WriteQCD(
         const CFVARIANT value,
         const CFDATETIME64 timeStamp,
         const int16_t qualityCode,
         CFENUM type = HmiWriteType::NoWait)
```

- `value`
  [in]: Tag value
- `timeStamp`
  [in]: Time stamp of the tag. Also in the past.

- `qualityCode`
  [in]: Quality code of the tag
- `type`
  [in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  – `HmiWriteType::NoWait` (default): Writes the tag value without waiting. Errors for the write operation are not detected.
  – `HmiWriteType::Wait`: Waits until the tag value is written in the AS. The associated errors are written.

"WriteWithOperatorMessage" method

Write process value of the tag synchronously in the Runtime system and create operator message. In addition to the reason, the operator message contains the old and new value, the user and host names and the unit.

```
CFRESULT WriteWithOperatorMessage(
         const CFVARIANT value,
         const CFSTR reason)
```

- `value`
  [in]: Value of the tag
- `reason`
  [in]: Reason of the value change for message

"Read" method

Read process value and properties of the tag synchronously from the Runtime system.

```
CFRESULT Read(
         IProcessValue **ppValue,
         CFENUM type = HmiReadType::Cache)
```

- `ppValue`
  [out]: Points to the properties and the value of the tag as an "IProcessValue" object.
- `type`
  [in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
  – `HmiReadType::Cache` (default): Reads the tag value from the tag image. If no registration exists, the tag is registered.
  – `HmiReadType::Device`: Reads the tag value directly from the AS. The tag image is not used.

Example

Write tags synchronously:

Copy code

```
CFRESULT WriteSingleTagSync(IRuntimePtr pRuntime, CCfString tag, CCfVariant& value)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        // Write value of tag
        errCode = pTag->Write(value);
        if(CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
    return errCode;
}
```

Write tag with time stamp and quality code synchronously:

Copy code

```
void WriteSingleTagQCDSync(IRuntimePtr pRuntime, CCfString tag, CCfVariant& value)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        // Write value of tag
        errCode = pTag->WriteQCD(value, CCfDateTime64::Now(), 128);
        if(CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}
```

Read tags synchronously:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        IProcessValuePtr pValue;

        // Read value of tag
        errCode = pTag->Read(&pValue);

        if(pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            ...
        }
        else
        {
            std::wcout << L"Read operation failed." << std::endl;
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
    }
    return errCode;
}
```

See also

IProcessValue (Page 1251)

ITagCallback (Page 1259)

19.10.3.4 ITagCallback (RT Uni)

Description

The "ITagCallback" interface and the "COdkTagSourceCBBase" and "COdkTagSetCB" classes define methods for implementing asynchronous read and write operations with tags. The methods are used by the C++-interface "ITagSet".

Members of the interface

The following methods are specified in the "ITagCallback" interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

The "OnReadComplete" callback method is called when the "ITagSet.ReadAsync" method is used.

```
CFRESULT OnReadComplete(
         IProcessValueEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId,
         CFBOOL completed)
```

- `pEnumerator`
  [out]: Points to an "IProcessValueEnumerator" object that contains the enumeration of the read process values.
- `systemError`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the tag.
- `completed`
  [out]: Status of the asynchronous transfer:
  - True: All alarms are read out.
  - False: Not all alarms are yet read out.

"OnWriteComplete" method

Callback method is called on completion of asynchronous write operations.

The "OnWriteComplete" callback method is called when the "ITagSet.WriteAsync" method is used.

```
CFRESULT OnWriteComplete(
         IErrorResultEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId
         CFBOOL completed)
```

- `pEnumerator`
  [out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations of the tag.
- `systemError`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the tag.
- `completed`
  [out]: Status of the asynchronous transfer:
  - True: All alarms are read out.
  - False: Not all alarms are yet read out.

"OnDataChanged" method

Callback method is called when a monitored tag value is changed.

The callback method is called after the process value change of a monitored TagSet ("ITagSet.Subscribe" method).

```
CFRESULT OnDataChanged(
         IProcessValueEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "IProcessValueEnumerator" object that contains the enumeration of the read process values.
- `systemError`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the tag.

Members of the classes

The following methods are implemented in the "COdkTagSourceCBBase" and "COdkTagSetCB" classes:

"SetEvent" method

Signals an event.

```
CFBOOL SetEvent()
```

"ResetEvent" method

Resets the signaling of an event.

```
CFBOOL ResetEvent()
```

"WaitForcompletion" method

Waits for the signaling of an event.

```
uint32_t WaitForcompletion(uint32_t dwMilliseconds)
```

`dwMilliseconds`
[in]: Time interval in milliseconds for which an event is waited for.

"GetValues" method

Return process values of the asynchronous read operation.

```
std::vector<IProcessValue*> GetValues()
```

Example

In the following section tags in a TagSet are read asynchronously. To this purpose the "ReadTagSetAsync" function uses a "COdkTagSetCB" object that implements the "ITagCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "COdkTagSetCB" object is determined via reference counting.

Copy code

```
void ReadTagSetAsync(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);

    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for(int i = 0; i < tags.size(); i++)
        {
            pTagSet->Add(tags[i]);
        }

        COdkTagSetCB* pTagSetCB = new COdkTagSetCB();

        if(pTagSetCB != nullptr)
        {
            pTagSetCB->AddRef();

            // Read the tag set asynchronously, result comes via callback
            if(CF_SUCCEEDED(pTagSet->ReadAsync(pTagSetCB)))
            {
                if (CF_SUCCESS == pTagSetCB-
>WaitForcompletion(std::numeric_limits<uint32_t>::max()))
                {
                    vector<IProcessValuePtr> pValues = pTagSetCB->GetValues();

                    std::wcout << L"Read finished " << std::endl;

                    // display tag values
                    for(int i = 0; i < pValues.size(); i++)
                    {
                        IProcessValue* pValue = pValues[i];
                        CCfString timeStamp;
                        CFDATETIME64 cfTimeStamp;
                        pValue->GetTimeStamp(&cfTimeStamp);
                        CCfDateTime64 time(cfTimeStamp);
                        timeStamp = time.GetDateTimeString(false);
                        CCfString strName;
                        pValue->GetTagName(&strName);
                        CCfVariant varValue;
                        pValue->GetValue(&varValue);

                       std::wcout << strName << L" " << timeStamp << L" " << L" Value: " <<
(double)(varValue) << std::endl;
                    }
                }
                else
                {
                  std::wcout << L"Error, couldn't create callback interface." << std::endl;
                    PrintErrorInformation(errCode, L"WaitForcompletion", pRuntime);
                }
            }
```

Copy code

```
            else
            {
                std::wcout << L"ReadAsync request failed." << std::endl;
                PrintErrorInformation(errCode, L"ReadAsync", pRuntime);
            }
        }
        else
        {
            std::wcout << L"General error" << std::endl;
            PrintErrorInformation(errCode, L"COdkTagSetCB", pRuntime);
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}
```

See also

ITag (Page 1254)

ITagSet (Page 1264)

ITagSetQCD (Page 1272)

19.10.3.5 ITagSet (RT Uni)

Description

The C++ interface "ITagSet" specifies properties and methods for an optimized access to several tags of the Runtime system.

After initialization of the "ITagSet" object, you can execute read and write access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

`value`
[in]: ContextId of the tag:

"GetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t *value)
```

`value`
[out]: Points to the ContextId of the tag.

"Remove" method

Remove individual tag from a TagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

`tagName`
[in]: Name of the tag that is removed from TagSet.

"Add" method

Add tag to a TagSet.

```
CFRESULT Add(const CFSTR tagName)
```

`tagName`
[in]: Name of the tag for TagSet

"AddWithValue" method

Add tag with process value to the TagSet.

```
CFRESULT AddWithValue(const CFSTR tagName, const CFVARIANT value)
```

- `tagName`
  [in]: Name of the tag
- `value`
  [in]: New value of the tag

"GetValue" method

Read process value of a tag of a TagSet.
To fill the local TagSet with process values, a "Read", "ReadAsync" or "AddWithValue" method must be called beforehand.

The values of the "IProcessValue" object are not available until after execution of the methods "Read", "ReadAsync" or "AddWithValue".

```
CFRESULT GetValue(const CFSTR tagName, CFVARIANT *pValue)
```

- `tagName`
  [in]: Name of the tag from the TagSet
- `pValue`
  [out]: Points to the process value of the tag.

"SetValue" method

Change the process value of a tag of a TagSet.

The "SetValue" method changes only the values of the local TagSet. In order to write the changed values into the automation system you must additionally execute the "Write" or "WriteAsync" method.

```
CFRESULT SetValue(const CFSTR tagName, const CFVARIANT value)
```

- `tagName`
  [in]: Name of the tag from the TagSet
- `value`
  [in]: New value of the tag

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator
**ppEnumerator, CFENUM type = HmiWriteType::NoWait)
```

- `ppEnumerator`
  [out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations.
- `type`
  [in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
  - `Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteWithOperatorMessage" method

Write process values of all tags of a TagSet synchronously in the Runtime system and create operator messages. In addition to the reason, the operation messages contain the old and new value, the user and host names and the unit.

```
CFRESULT WriteWithOperatorMessage(const CFSTR reason)
```

`reason`
[in]: Reason of the value change for message

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType::Wait` type and waits until the tag value has been written in the automation system. Associated errors are written.

```
CFRESULT WriteAsync(
         ITagCallback* pTagSetCb)
```

- `pTagSetCb`
  [in]: Points to the "ITagCallback" object that implements the callback interface.

"Read" method

Read process values and properties of all the tags of a TagSet synchronously from the Runtime system.

```
CFRESULT Read(
         IProcessValueEnumerator **ppEnumerator,
         CFENUM type = HmiReadType::Cache)
```

- `ppEnumerator`
  [in/out]: Points to the properties and process values of the tags as an "IProcessValueEnumerator" object.
- `type`
  [in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
  - `Cache` (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
  - `Device`: Reads the tag values directly from the automation system. The tag image is not used.

"ReadAsync" method

Read process values and properties of all the tags of a TagSet asynchronously from the Runtime system.

```
CFRESULT ReadAsync(
         ITagCallback *pTagSetCb,
         CFENUM type = HmiReadType::Cache)
```

- `pTagSetCb`
  [in]: Points to the "ITagCallback" object that implements the callback interface.
- `type`
  [in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
  - `Cache` (default): Reads the tag value from the tag image. If no subscription exists, the tag is subscribed.
  - `Device`: Reads the tag value directly from the AS. The tag image is not used.

"Subscribe" method

Subscribe all tags of a TagSet asynchronously for cyclic monitoring of the process values.

Note

Tags from IO devices with the "Cyclic in operation" acquisition mode

For a tag with the acquisition mode "Cyclic in operation", the value stored in the process image when Subscribe is called might be outdated. OnAdd therefore only provides the QualityCode "uncertain". Only value changes made after the Subscribe call provide the current value and the QualityCode "good".

```
CFRESULT Subscribe(ITagCallback *pTagSetCb)
```

`pTagCb`
[in]: Points to the "ITagCallback" object that implements the callback interface.

"CancelSubscribe" method

Cancel monitoring of all tags of a TagSet.

```
CFRESULT CancelSubscribe()
```

"GetCount" method

Return the number of tags of a TagSet list.

```
CFRESULT GetCount(int32_t *value)
```

`value`
[out]: Points to the value for the number of tags of the TagSet list.

"Clear" method

Remove all tags from a TagSet.

```
CFRESULT Clear()
```

Example

Write TagSet asynchronously:

Copy code

```
struct TagTuple_T
{
    CCfSmartString _tagName;
    CCfVariant _tagValue;
};

void WriteTagSetAsync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
      ICfUnknownPtr pUnk;
      CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
      vector<IErrorResultPtr> pErrors;
      if(pUnk != nullptr && CF_SUCCEEDED(errCode))
      {
            ITagSetPtr pTagSet(pUnk);
            // add tags to tag set
            for(int i = 0; i < tags.size(); i++)
            {
                  pTagSet->AddWithValue(CCfString(tags[i]._tagName), tags[i]._tagValue);
            }
            COdkTagSetCB* pTagSetCB = new COdkTagSetCB();
            if(pTagSetCB != nullptr)
            {
                  pTagSetCB->AddRef();
                  // Write value of tag asynchronously
                  pTagSet->WriteAsync(pTagSetCB);
                  errCode = pTagSetCB-
>WaitForcompletion(std::numeric_limits<uint32_t>::max());
                  if (CF_SUCCESS == errCode)
                  {
                        pErrors = pTagSetCB->GetErrors();

                        for (int i = 0; i < pErrors.size(); i++)
                        {
                              IErrorResult* pError = pErrors[i];
                              CFRESULT tagError;
                              pError->GetError(&tagError);
                              CCfString strName;
                              pError->GetName(&strName);
                              if (CF_FAILED(tagError))
                              {
                                   PrintErrorInformation(tagError, L"Tag Write", pRuntime);
                              }
                        }
                  } else
                  {
                        PrintErrorInformation(errCode, L"WaitForcompletion", pRuntime);
                  }
            }
            else
            {
                  PrintErrorInformation(errCode, L"COdkTagSetCB", pRuntime);
            }
      }
      else
      {
```

Copy code

```
            PrintErrorInformation(errCode, L"GetObject", pRuntime);
      }
}
```

Start monitoring for tags of a TagSet:

Copy code

```
void SubscribeTagSet(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->Add(tags[i]);
        }

        COdkTagSetCB* pTagSetCB = new COdkTagSetCB();

        if (pTagSetCB != nullptr && CF_SUCCEEDED(errCode))
        {
            pTagSetCB->AddRef();

            // subscribe tags
            errCode = pTagSet->Subscribe(pTagSetCB);
            if(CF_FAILED(errCode))
            {
                std::wcout << L"Error, couldn't create callback interface." << std::endl;
                PrintErrorInformation(errCode, L"Subscribe", pRuntime);
            }

        }
        else
        {
            std::wcout << L"Error, couldn't create callback interface." << std::endl;
            PrintErrorInformation(errCode, L"COdkTagSetCB", pRuntime);
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}
```

See also

IProcessValue (Page 1251)

ITagCallback (Page 1259)

IErrorResultEnumerator (Page 1247)

19.10.3.6 ITagSetQCD (RT Uni)

Description

The C++ interface "ITagSetQCD" specifies methods for optimized writing of several tags of the Runtime system. The tags also have a freely definable time stamp and quality code. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

After initialization of the "ITagSetQCD" object, you can have read access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

`value`
[in]: ContextId of the tag:

"GetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t *value)
```

`value`
[out]: Points to the ContextId of the tag.

"Remove" method

Remove individual tag from a TagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

`tagName`
[in]: Name of the tag that is removed from TagSet.

"Add" method

Add tag with process value, quality code and time stamp to the TagSet.

```
CFRESULT Add(
         const CFSTR tagName,
         const CFVARIANT value,
         const CFDATETIME64 timeStamp,
         const int16_t qualityCode)
```

- `tagName`
  [in]: Name of the tag for TagSet
- `value`
  [in]: New process value of the tag
- `timeStamp`
  [in]: Time stamp of the process value. Also in the past.
- `qualityCode`
  [in]: Quality code for process value

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator
**ppEnumerator, CFENUM type = HmiWriteType::NoWait)
```

- `ppEnumerator`
  [out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations.
- `type`
  [in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType::NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
  - `HmiWriteType::Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType::Wait` type and waits until the tag value has been written in the automation system. Associated errors are written.

```
CFRESULT WriteAsync(
         ITagCallback* pTagSetCb)
```

- `pTagSetCb`
  [in]: Points to the "ITagCallback" object that implements the callback interface.

"GetCount" method

Return the number of tags of a TagSet list.

```
CFRESULT GetCount(int32_t *value)
```

`value`
[out]: Points to the value for the number of tags of the TagSet list.

"GetItem" method

Return tag of the TagSet for changing or reading out process value, QualityCode and time stamp.

```
CFRESULT GetItem(
         const CFSTR name,
         ITagSetQCDItem **pTagSetQCDItem)
```

- `name`
  [in]: Name of the tag in the TagSet
- `pTagSetQCDItem`
  [out]: Points to tag as "TagSetQCDItem" object

"Clear" method

Remove all tags from a TagSet.

```
CFRESULT Clear()
```

Example

Write TagSet with time stamp and quality code synchronously:

Copy code

```
struct TagTuple_T
{
    CCfSmartString _tagName;
    CCfVariant _tagValue;
};

void WriteTagSetQCDSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSetQCD"), &pUnk);
    if(pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ITagSetQCDPtr pTagSetQCD(pUnk);

        // add tags to tag set
        for(int i = 0; i < tags.size(); i++)
        {
            pTagSetQCD->Add(CCfString(tags[i]._tagName), tags[i]._tagValue,
CCfDateTime64::Now(), 128);
        }

        IErrorResultEnumerator* pEnumerator;
        errCode = pTagSetQCD->Write(&pEnumerator);
        if(CF_FAILED(errCode))
        {
            std::wcout << L"Write operation failed." << std::endl;
            PrintErrorInformation(errCode, L"Write", pRuntime);
        }

        if (pEnumerator != nullptr)
        {
            while (pEnumerator->MoveNext() == CF_SUCCESS)
            {
                IErrorResult* pValue;
                CFRESULT errorCode = pEnumerator->Current(&pValue);
                if (pValue != nullptr && CF_SUCCEEDED(errorCode))
                {
                    pValue->GetError(&errorCode);
                    CCfString str;
                    pValue->GetName(&str);
                    if (CF_FAILED(errorCode))
                    {
                     std::wcout << L"Write Tag failed, Tag name: " << str << L", ErrorCode:
" << errorCode << std::endl;
                        PrintErrorInformation(errorCode, L"Write Tag", pRuntime);
                    }
                }
            }
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
```

Copy code

```
}
```

See also

IProcessValue (Page 1251)

ITagCallback (Page 1259)

ITagSetQCDItem (Page 1277)

IErrorResultEnumerator (Page 1247)

19.10.3.7 ITagSetQCDItem (RT Uni)

Description

The C++ interface "ITagSetQCDItem" specifies methods for adapting tags of the Runtime system. You can read in tags into a TagSetQCD and then change all names, values, time stamps and QualityCodes of the tags.

Note

Reaction to external tags

For external tags, only the tag value is set. The QualityCode and time stamp are set internally by the system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"GetName" method

Return the name of the tag.

```
CFRESULT GetName(CFSTR *name)
```

`name`
[out]: Points to the name of the tag.

"SetName" method

Change name of the tag.

```
CFRESULT SetName(const CFSTR name)
```

`name`
[in]: New name of the tag.

"GetValue" method

Return value of the tag at the moment of the read operation.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the process value of the tag.

"SetValue" method

Change value of the tag.

```
CFRESULT SetValue(const CFVARIANT value)
```

`value`
[in]: New process value of the tag.

"GetTimeStamp" method

Return time stamp of the tag.

```
CFRESULT GetTimeStamp(CFDATETIME64 *timeStamp)
```

`timeStamp`
[out]: Points to the time stamp of the tag.

"SetTimeStamp" method

Change time stamp of the tag.

```
CFRESULT SetTimeStamp(const CFDATETIME64 timeStamp)
```

`timeStamp`
[in]: New time stamp of the tag

"GetQuality" method

Return quality code of the read operation of the tag.

```
CFRESULT GetQuality(int32_t *qualityCode)
```

`qualityCode`
[out]: Points to the quality code of the tag.

"SetQuality" method

Change quality code of the tag.

```
CFRESULT SetQuality(const int32_t qualityCode)
```

`qualityCode`
[in]: New quality code of the tag

See also

ITagSetQCD (Page 1272)

19.10.3.8 ILoggedTagValue (RT Uni)

Description

The C++ interface "ILoggedTagValue" specifies the properties for process values of logging tags of a logging system. The "ILoggedTagValue" interface displays historical process values.

Members

The following methods are specified in the interface:

"GetTagName" method

Return name of the logging tag.

```
CFRESULT GetTagName(CFSTR *value)
```

`value`
[out]: Points to the name of the process value belonging to the logging tag.

"GetValue" method

Return process value of the logging tag.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the process value of the logging tag.

"GetQuality" method

Return quality code of the process value.

```
CFRESULT GetQuality(int16_t *value)
```

`value`
[out]: Points to the quality code of the process value.

"GetTimeStamp" method

Return time stamp of the process value.

```
CFRESULT GetTimeStamp(CFDATETIME64 *value)
```

`value`
[out]: Points to the time stamp of the process value.

"GetError" method

Return error code of the process value.

```
CFRESULT GetError(uint32_t *value)
```

`value`
[out]: Points to the error code of the process value.

"GetFlags" method

Return context information from the read operation for the process value.

```
CFRESULT GetFlags(HmiTagLoggingValueFlags *value)
```

`value`
[out]: Points to the context information.

The "HmiTagLoggingValueFlags" enumeration contains the following bit-by-bit-coded values:

- 0: `Extra`
  There are still additional values at the time of the process value.
- 2: `Calculated`
  Process value is calculated.
- 16: `Bounding`
  Process value is a limit value.
- 32: `NoData`
  No additional information available
- 64: `FirstStored`
  Process value is the first value stored in the logging system.
- 128: `LastStored`
  Process value is the last value stored in the logging system.

Example

Output process values of a logging tag:

Copy code

```
void PrintValues(ILoggedTagValueEnumeratorPtr pItems)
{
    // Iterate over the process value objects
    while (CF_SUCCEEDED(pItems->MoveNext()))
    {
        ILoggedTagValuePtr pValue;
        CFRESULT errCode = pItems->Current(&pValue); // get current process value
        if (pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            CCfString timeStamp;
            CFDATETIME64 cfTimeStamp;
            pValue->GetTimeStamp(&cfTimeStamp);
            CCfDateTime64 time(cfTimeStamp);
            timeStamp = time.GetDateTimeString(false);
            CCfString strName;
            pValue->GetTagName(&strName);
            CCfVariant varValue;
            pValue->GetValue(&varValue);

            std::wcout << strName << L" " << timeStamp << L" " << L" Value: " << (double)
(varValue) << std::endl;
        }
    }
}
```

See also

ILoggedTagValueEnumerator (Page 1281)

ILoggedTag (Page 1287)

19.10.3.9 ILoggedTagValueEnumerator (RT Uni)

Description

The "ILoggedTagValueEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of process values of a logging tag of the Runtime system. The methods are used by the C++-interfaces "ILoggedTag" and "ILoggedTagSet".

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(ILoggedTagValue **ppItem)
```

`ppItem`
[out]: Points to the current "ILoggedTagValue" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext"method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of the elements of the list.

Example

Output process values of a logging tag:

Copy code

```
void PrintValues(ILoggedTagValueEnumeratorPtr pItems)
{
    // Iterate over the process value objects
    while (CF_SUCCEEDED(pItems->MoveNext()))
    {
        ILoggedTagValuePtr pValue;
        CFRESULT errCode = pItems->Current(&pValue); // get current process value
        if (pValue != nullptr && CF_SUCCEEDED(errCode))
        {
            ...
        }
    }
}
```

See also

ILoggedTagValue (Page 1279)

ILoggedTag (Page 1287)

ILoggedTagSet (Page 1290)

ILoggedTagCallback / ILoggedTagSetCallback (Page 1282)

19.10.3.10 ILoggedTagCallback / ILoggedTagSetCallback (RT Uni)

Description

The interfaces "ILoggedTagCallback" and "ILoggedTagSetCallback" and the classes "COdkTagSourceCBBase" and "COdkTagSetLoggingCB" define methods for implementing asynchronous read and write operations with logging tags. The methods are used by the C++-interfaces "ILoggedTag" and "ILoggedTagSet".

All the methods return CF_SUCCESS following successful execution.

Members of "ILoggedTagCallback"

The following methods are specified in the interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

`CFRESULT OnReadComplete(ILoggedTagValueEnumerator *pEnumerator)`

`pEnumerator`
[out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.

"OnDeleteComplete" method

Callback method is called on completion of asynchronous delete operations.

```
CFRESULT OnDeleteComplete(ILoggedTagValueEnumerator *pEnumerator)
```

`pEnumerator`
[out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.

"OnWriteComplete" method

Callback method is called on completion of asynchronous write operations. Can only be applied to individual logging tags in the "ILoggedTagCallback" interface.

```
CFRESULT OnWriteComplete(ILoggedTagValueEnumerator *pEnumerator)
```

`pEnumerator`
[out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.

"OnDataChanged" method

Callback method is called when a monitored logging tag is changed.

```
CFRESULT OnDataChanged(ILoggedTagValueEnumerator *pEnumerator)
```

`pEnumerator`
[out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.

Members of "ILoggedTagSetCallback"

The following methods are specified in the interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

The "OnReadComplete" callback method is called when the "ReadAsync" method is used.

```
CFRESULT OnReadComplete(ILoggedTagValueEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.
- `errorCode`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the logging tag.

"OnDataChanged" method

Callback method is called when a monitored logging tag is changed.

The callback method is called after the process value change of a monitored logging tag or a LoggedTagSet.

```
CFRESULT OnDataChanged(ILoggedTagValueEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.
- `errorCode`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the logging tag.

Members of the classes

The following methods are implemented in the "COdkTagSourceCBBase" and "COdkTagSetLoggingCB" classes:

"SetEvent" method

Signals an event.

```
CFBOOL SetEvent()
```

"ResetEvent" method

Resets the signaling of an event.

```
CFBOOL ResetEvent()
```

"WaitForcompletion" method

Waits for the signaling of an event.

```
uint32_t WaitForcompletion(uint32_t dwMilliseconds)
```

`dwMilliseconds`
[in]: Time interval in milliseconds for which an event is waited for.

"GetValues" method

Return process values of the asynchronous read operation.

```
std::vector<ILoggedTagValuePtr> GetValues()
```

Example

Output LoggedTagSet asynchronously:

Copy code

```
void LogggingReadTagSetAsync(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"LoggedTagSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ILoggedTagSetPtr pTagSet(pUnk);

        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->Add(tags[i]);
        }

        CCfDateTime64 begin, end;
        begin = CCfDateTime64::Now(true);
        end = begin;
        begin.SubtractTimeSpan(Get1Minute() * 30);

        ILoggedTagValueEnumeratorPtr pItems;

        COdkTagSetLoggingCB* pTagSetCB = new COdkTagSetLoggingCB();

        if (pTagSetCB != nullptr)
        {
            pTagSetCB->AddRef();

            // Read the tag set asynchronously, result comes via callback
            if (CF_SUCCEEDED(pTagSet->ReadAsync(pTagSetCB, begin, end, true)))
            {
                if (CF_SUCCESS == pTagSetCB-
>WaitForcompletion(std::numeric_limits<uint32_t>::max()))
                {
                    vector<ILoggedTagValuePtr> pValues = pTagSetCB->GetValues();

                    std::wcout << L"Read finished " << std::endl;

                    // display tag values
                    for (int i = 0; i < pValues.size(); i++)
                    {
                        ILoggedTagValue* pValue = pValues[i];
                        CCfString timeStamp;
                        CFDATETIME64 cfTimeStamp;
                        pValue->GetTimeStamp(&cfTimeStamp);
                        CCfDateTime64 time(cfTimeStamp);
                        timeStamp = time.GetDateTimeString(false);
                        CCfString strName;
                        pValue->GetTagName(&strName);
                        CCfVariant varValue;
                        pValue->GetValue(&varValue);

                      std::wcout << strName << L" " << timeStamp << L" " << L" Value: " <<
(double)(varValue) << std::endl;
                    }
```

Copy code

```
                }
                else
                {
                  std::wcout << L"Error, couldn't create callback interface." << std::endl;
                    PrintErrorInformation(errCode, L"WaitForcompletion", pRuntime);
                }
            }
            else
            {
                std::wcout << L"ReadAsync request failed." << std::endl;
                PrintErrorInformation(errCode, L"ReadAsync", pRuntime);
            }
        }
        else
        {
            std::wcout << L"General error" << std::endl;
            PrintErrorInformation(errCode, L"COdkTagSetCB", pRuntime);
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}
```

See also

ILoggedTag (Page 1287)

ILoggedTagSet (Page 1290)

ILoggedTagValueEnumerator (Page 1281)

19.10.3.11 ILoggedTag (RT Uni)

Description

The C++ interface "ILoggedTag" specifies methods for handling logging tags of a logging system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetTagName" method

Set name of the logging tag.

```
CFRESULT SetTagName(const CFSTR tagName)
```

`tagName`
[in]: Name of the logging tag

"Read" method

Read logged process values of a logging tag of a time period synchronously from the logging system.

```
CFRESULT Read(
         const CFTIMEDATE64 begin,
         const CFTIMEDATE64 end,
         ILoggedTagValueEnumerator **ppEnumerator,
         CFBOOL boundingValue)
```

- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `ppEnumerator`
  [in/out]: Points to the enumeration of process values of the logging tag as "ILoggedTagValueEnumerator" object.
- `boundingValue`
  [in]: True, in order to additionally return high and low limits.

Example

Read out process values of a logging tag synchronously from a logging system:

Copy code

```
void LoggingReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"LoggedTag"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ILoggedTagPtr pTag(pUnk);
        pTag->SetTagName(tag);

        CCfDateTime64 begin, end;
        begin = CCfDateTime64::Now(true);
        end = begin;
        begin.SubtractTimeSpan(Get1Minute() * 3);

        ILoggedTagValueEnumeratorPtr pItems;
        // Read value of tag
        errCode = pTag->Read(begin, end, &pItems, true);

        if (pItems != nullptr && CF_SUCCEEDED(errCode))
        {
            std::wcout << "Read finished " << std::endl;
            PrintValues(pItems);
        }
        else
        {
            std::wcout << L"Read operation failed." << std::endl;
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
    }
}
```

See also

ILoggedTagValue (Page 1279)

ILoggedTagSet (Page 1290)

ILoggedTagCallback / ILoggedTagSetCallback (Page 1282)

ILoggedTagValueEnumerator (Page 1281)

19.10.3.12 ILoggedTagSet (RT Uni)

Description

The C++ interface "ILoggedTagSet" specifies methods for optimized access to several logging tags of a logging system.

After initialization of the "ILoggedTagSet" object, you can execute read and write access to multiple logging tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple logging tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

ID as additional identification feature of the logging tag. The ContextId can, for example, be used to recognize identically named logging tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

`value`
[in]: ContextId of the logging tag

"GetContextId" method

ID as additional identification feature of the logging tag. The ContextId can, for example, be used to recognize identically named logging tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t *value)
```

`value`
[out]: Points to the ContextId of the logging tag.

"Remove" method

Remove individual logging tag from a LoggedTagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

`tagName`
[in]: Name of the logging tag that is removed from the LoggedTagSet.

"Add" method

Add logging tag to a LoggedTagSet.

```
CFRESULT Add(const CFSTR tagName)
```

`tagName`
[in]: Name of the logging tag for the LoggedTagSet

"Subscribe" method

Subscribe all logging tags of a LoggedTagSet asynchronously for updating the process values following a change. When new process values are logged, they can be processed with the "OnDataChanged" event.

```
CFRESULT Subscribe(ILoggedTagSetCallback *pTagSetLoggedCb)
```

`pTagSetLoggedCb`
[in]: Points to the "ILoggedTagSetCallback" object that implements the callback interface.

"CancelSubscribe" method

Cancel updating of process values following a change for all logging tags of a LoggedTagSet.

```
CFRESULT CancelSubscribe()
```

"Read" method

Retrieve all logging tags of a LoggedTagSet for a period of time synchronously from the logging system.

```
CFRESULT Read(
         const CFTIMEDATE64 begin,
         const CFTIMEDATE64 end,
         ILoggedTagValueEnumerator **ppEnumerator,
         CFBOOL boundingValue)
```

- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `ppEnumerator`
  [in/out]: Points to the enumeration of process values of the logging tags of the LoggedTagSet as "ILoggedTagValueEnumerator" object.
- `boundingValue`
  [in/optional]: True, in order to additionally return high and low limits.

"ReadAsync" method

Retrieve all logging tags of a LoggedTagSet for a period of time synchronously from the logging system.

```
CFRESULT ReadAsync(
         ILoggedTagSetCallback *pTagLoggedCb,
         const CFTIMEDATE64 begin,
         const CFTIMEDATE64 end,
         CFBOOL boundingValue)
```

- `pTagLoggedCb`
  [in]: Points to the "ILoggedTagSetCallback" object that implements the callback interface.
- `begin`
  [in]: Start date of the time period

- `end`
  [in]: End date of the time period
- `boundingValue`
  [in/optional]: True, in order to additionally return high and low limits.

"GetCount" method

Return the number of logging tags of a LoggedTagSet list.

```
CFRESULT GetCount(int32_t *value)
```

`value`
[out]: Points to the value for the number of logging tags of the LoggedTagSet list.

"Clear" method

Remove all logging tags from a LoggedTagSet.

```
CFRESULT Clear()
```

Example

Subscribe logging tags of a LoggedTagSet for change monitoring:

Copy code

```
void LoggingSubscribeTagSet(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"LoggedTagSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        ILoggedTagSetPtr pTagSet(pUnk);
        // add tags to tag set
        for (int i = 0; i < tags.size(); i++)
        {
            pTagSet->Add(tags[i]);
        }

        COdkTagSetLoggingCB* pTagSetCB = new COdkTagSetLoggingCB();

        if (pTagSetCB != nullptr && CF_SUCCEEDED(errCode))
        {
            pTagSetCB->AddRef();

            // subscribe tags
            errCode = pTagSet->Subscribe(pTagSetCB);
            if (CF_FAILED(errCode))
            {
                std::wcout << L"Error, couldn't create callback interface." << std::endl;
                PrintErrorInformation(errCode, L"Subscribe", pRuntime);
            }
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}
```

See also

ILoggedTag (Page 1287)

ILoggedTagCallback / ILoggedTagSetCallback (Page 1282)

ILoggedTagValueEnumerator (Page 1281)

19.10.3.13 ITags (RT Uni)

Description

The C++ interface "ITags" defines methods with which you can access configured tags.

The interface inherits from the "ICfDispatch" interface.

All the methods return `CF_SUCCESS` following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"Find" method

Supplies an enumerator for access to the "ITagAttributes" instances of the specified tag.

```
CFRESULT Find(
         CFVARIANT systemIDs,
         int32_t language,
         CFSTR filter,
         ITagAttributesEnumerator** ppITagAttributesEnumerator)
```

```
Find(CFVARIANT systemIDs, uint32_t language, CFSTR filter,
ITagAttributesEnumerator** ppITagAttributesEnumerator)
```

- `systemIDs`
  [in]: Collection of SystemNames on which the tags were configured.
- `language`
  [in]: Language code ID of filter
- `filter`
  [in]: Filter by name of the tags to restrict the search.
  Supports searching with wildcard (*)
- `ppITagAttributesEnumerator`
  [out]: Enumerator which supplies access to the "ITagAttributes" instances.

"FindAsync" method

For asynchronous searching for "ITagAttributes" instances of the specified tags.

```
FindAsync(
         CFVARIANT systemIDs
         uint32_t language
         CFSTR filter
         ITagAttributesCallback* pCallback)
```

- `systemIDs`
  [in]: Collection of SystemNames on which the tags were configured.
- `language`
  [in]: Language code ID of filter
- `filter`
  [in]: Filter by name of the tags to restrict the search.
  Supports searching with wildcard (*)
- `filterpCallback`:
  [in]: Callback pointer to an "ITagAttributesCallback" instance

19.10.3.14 ITagAttributes (RT Uni)

Description

The C++ interface "ITagAttributes" defines methods for access to the attributes of a tag.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Supplies the name of the tags. Must be unique throughout the device.

```
GetName(
        CFSTR* name)
```

- `name`
  [out]: The name

"GetDisplayName" method

Supplies the display name of the tags.

```
GetDisplayName(
        CFSTR* name)
```

- `name`
  [out]: The display name

"GetDataType" method

Supplies the data type of the tags.

```
GetDataType(
        CFENUM* datatype)
```

- `datatype`
  [out]: The data type of the tag

"GetConnection" method

Supplies the connection of the tag.

The memory location of the tag in the controller is accessed via the connection.

```
GetConnection(
        CFSTR* connection)
```

- `connection`
  [out]: The connection

"GetAcquisitionCycle" method

Specifies the tag acquisition cycle.

If you configure a tag at an object, the acquisition cycle of the tag is displayed at the object.

```
GetAcquisitionCycle(
        CFSTR* acquisitionCycle)
```

- `acquisitionCycle`
  [out]: The acquisition cycle

"GetAcquisitionMode" method

Supplies the acquisition mode of the tag.

```
GetAcquisitionMode(
        CFENUM* acquisitionMode)
```

- `acquisitionMode`
  [out]: Value of the enumeration "HmiAcquisitionMode".
  The enumeration "HmiAcquisitionMode" can contain the following values:
  - Undefined (0)
  - CyclicOnUse (1)
  - CyclicContinous (2)
  - OnDemand (3)
  - OnChange (4)

"GetMaxLength" method

Supplies the length of the tags.

The length is only available with a string tag. The length is preset for structure tags and cannot be changed.

```
GetMaxLength(
        uint32_t* maxLength)
```

- `maxLength`
  [out]: Maximum length

"GetPersistent" method

Supplies the persistence of the tags.

```
GetPersistent(
       CFBOOL* persistent)
```

- `persistent`
  [out]: The persistence

"GetInitialValue" method

Supplies the start value of the tag.

After Runtime starts, the tag retains the start value until an operator or the PLC changes the value.

```
GetInitialValue(
       CFVARIANT * initialValue)
```

- `initialValue`
  [out]: The start value

"GetInitialMaxValue" method

Supplies the start value for the event "On exceeding".

```
InitialMaxValue(
        CFVARIANT * initialValue)
```

- `initialValue`
  [out]: The start value

"GetInitialMinValue" method

Supplies the start value for the event "On falling below".

```
InitialMinValue(
        CFVARIANT * initialValue)
```

- `initialValue`
  [out]: The start value

"GetSubstituteValue" method

Supplies the substitute value of the tags.

If the selected condition occurs, the tag is filled with the substitute value in runtime.

```
GetSubstituteValue(
        CFVARIANT * substituteValue)
```

- `substituteValue`
  [out]: The substitute value

"GetSubstituteValueUsage" method

Supplies the condition for the use of the substitute value of the tag.

```
GetSubstituteValueUsage(
        CFVARIANT * substituteValueUsage)
```

- `substituteValueUsage`
  [out]: The condition

19.10.3.15 ITagAttributesEnumerator (RT Uni)

Description

The "ITagAttributesEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of tag attributes. The enumerator enables access to individual attributes from a set of tag attributes.

The interface inherits from the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumerator.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumerator

```
CFRESULT Current(
        ITagAttributes** ppItem)
```

- `ppItem`
  [out]: The current "ITagAttributes" instance

"Reset" method

Reset the current position in the enumerator. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumerator or the number of its elements.

```
CFRESULT Count(
        uint32_t* value)
```

- `value`
  [out]: Number of attributes

19.10.3.16 ITagAttributesCallback (RT Uni)

Description

The C++ interface "ITagAttributesCallback" defines the callback method "OnTagAttributesRead". The method is used for implementing asynchronous read operations of tag attributes. The method is used by the C++ interface "ITags".

The interface implements the methods of the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"OnTagAttributesRead" method

Callback method, is called on completion of an asynchronous search for "ITagAttributes" instances.

```
OnTagAttributesRead(
        ITagAttributesEnumerator* pEnumerator,
        CFBOOL bIsCompleted)
```

- `ITagAttributesEnumerator`
  [in/out]: Reference to an "ITagAttributesEnumerator" instance which provides access to the tag attributes.
- `bIsCompleted`
  [out]: Supplies information on whether the read operation was successfully completed.

19.10.3.17 ILoggingTags (RT Uni)

Description

The C++ interface "ILoggingTags" defines methods with which you can access configured logging tags.

The interface inherits from the "ICfDispatch" interface.

All the methods return `CF_SUCCESS` following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"Find" method

Supplies an enumerator for access to the "ITagAttributes" instances of the specified logging tag.

```
CFRESULT Find(
         CFVARIANT systemIDs,
         uint32_t language,
         CFSTR filter,
         ILoggingTagAttributesEnumerator**
ppILoggingTagAttributesEnumerator)
```

- `systemIDs`
  [in]: Collection of SystemNames on which the logging tags were configured.
- `language`
  [in]: Language code ID of filter
- `filter`
  [in]: Filter by name of the logging tags to restrict the search.
  Supports searching with wildcard (*).
  Example:
  `Tag1:*` Supplies all logging tags of "Tag1".
- `ppILoggingTagAttributesEnumerator`
  [out]: Enumerator which supplies access to the "ILoggingTagAttributes" instances.

"FindAsync" method

For asynchronous searching for "ILoggingTagAttributes" instances.

```
CFRESULT FindAsync(CFVARIANT systemIDs,
         uint32_t language,
        CFSTR filter,
        ILoggingTagAttributesCallback* pCallback)
```

- `systemIDs`
  [in]: Collection of SystemNames on which the logging tags were configured.
- `language`
  [in]: Language code ID of filter
- `filter`
  [in]: Filter by name of the logging tags to restrict the search.
  Supports searching with wildcard (*).
- pCallback
  [in]: Callback pointer to an "ILoggingTagAttributesCallback" instance

19.10.3.18 ILoggingTagAttributes (RT Uni)

Description

The C++ interface "ITagAttributes" defines methods for access to the attributes of a logging tag.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Supplies the name of the logging tag.

```
GetName(
        CFSTR* name)
```

- `name`
  [out]: The name

"GetDisplayName" method

Supplies the display name of the logging tags.

```
GetDisplayName(
        CFSTR* name)
```

- `name`
  [out]: The display name

"GetDataType" method

Supplies the data type of the logging tags.

```
GetDataType(
        CFENUM* datatype)
```

- `datatype`
  [out]: The data type

19.10.3.19 ILoggingTagAttributesEnumerator (RT Uni)

Description

The "ILoggingTagAttributesEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of logging tag attributes. The enumerator enables access to individual attributes from a set of tag attributes.

The interface inherits from the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumerator.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumerator

```
CFRESULT Current(
         ITagAttributes** ppItem)
```

- `ppItem`
  [out]: The current "ITagAttributes" instance

"Reset" method

Reset the current position in the enumerator. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumerator or the number of its elements.

```
CFRESULT Count(
         uint32_t* value)
```

- `value`
  [out]: Number of attributes

19.10.3.20 ILoggingTagAttributesCallback (RT Uni)

Description

The C++ interface "ILoggingTagAttributesCallback" defines the callback method "OnTagAttributesRead". The method is used for implementing asynchronous read operations of logging tag attributes. The method is used by the C++ interface "ILoggingTags".

The interface implements the methods of the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"OnTagAttributesRead" method

Callback method, is called on completion of an asynchronous search for "ITagAttributes" instances.

```
OnTagAttributesRead(
        ILoggingTagAttributesEnumerator* pEnumerator,
        CFBOOL bIsCompleted)
```

- `ILoggingTagAttributesEnumerator`
  [in]: Reference to an "ILoggingTagAttributesEnumerator" instance which provides access to the tag attributes.
- `bIsCompleted`
  Supplies information on whether the result of the read operation is complete or whether other events will come.

19.10.4 Interfaces of the alarms (RT Uni)

19.10.4.1 IAlarmResult (RT Uni)

Description

The C++ interface "IAlarmResult" specifies methods for accessing properties of active alarms of the Runtime system.

An "IAlarmResult" object is a pure data object that maps all properties of an active alarm.

Members

The following methods are specified in the interface:

"GetInstanceID" method

Return InstanceID for an alarm with multiple instances.

```
CFRESULT GetInstanceID(uint32_t *value)
```

`value`
[out]: Points to the InstanceID of the alarm.

"GetSourceID" method

Return source at which the alarm was triggered.

```
CFRESULT GetSourceID(CFSTR *value)
```

`value`
[out]: Points to the source of the alarm.

"GetName" method

Return name of the alarm.

```
CFRESULT GetName(CFSTR *value)
```

`value`
[out]: Shows the name of the alarm.

"GetAlarmClassName" method

Return name of the alarm class.

```
CFRESULT GetAlarmClassName(CFSTR *value)
```

`value`
[out]: Points to the symbol of the associated alarm class.

"GetAlarmClassSymbol" method

Return symbol of the alarm class

```
CFRESULT GetAlarmClassSymbol(CFSTR *value)
```

`value`
[out]: Shows the name of the associated alarm class.

"GetState" method

Return current alarm state.

```
CFRESULT GetState(int32_t *value)
```

`value`
[out]: Shows the current alarm status.

"GetStateText" method

Return current alarm state as text, for example "Incoming" or "Outgoing".

```
CFRESULT GetStateText(CFSTR *value)
```

`value`
[out]: Shows the current alarm status as text.

"GetEventText" method

Return text that describes the alarm event.

```
CFRESULT GetEventText(CFSTR *value)
```

`value`
[out]: Points to the text that describes the alarm event.

"GetInfoText" method

Return text that describes the alarm event.

```
CFRESULT GetInfoText(CFSTR *value)
```

`value`
[out]: Points to the text that describes an operator instruction for the alarm.

"GetAlarmText1GetAlarmText9" method

Return additional texts 1-9 of the alarm.

```
CFRESULT GetAlarmText1(CFSTR *value)
```

…

```
CFRESULT GetAlarmText9(CFSTR *value)
```

`value`
[out]: Points to the additional text of the alarm.

"GetTextColor" method

Return text color of the alarm state.

```
CFRESULT GetTextColor(uint32_t *value)
```

`value`
[out]: Points to the text color of the alarm state.

"GetBackColor" method

Return background color of the alarm state.

```
CFRESULT GetBackColor(uint32_t *value)
```

`value`
[out]: Points to the background color of the alarm state.

"GetFlashing" method

Return flashing background color of the alarm state.

```
CFRESULT GetFlashing(CFBOOL *value)
```

`value`
[out]: Points to the flashing background color of the alarm state.

"GetModificationTime" method

Return time of the last modification to the alarm state.

```
CFRESULT GetModificationTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the last change of the alarm state.

"GetChangeReason" method

Return trigger event for change of the alarm state.

```
CFRESULT GetChangeReason(uint16_t *value)
```

`value`
[out]: Points to the trigger event for the change of the alarm state.

"GetRaiseTime" method

Return trigger time of the alarm.

```
CFRESULT GetRaiseTime(CFDATETIME64 *value)
```

`value`
[out]: Shows the triggering time of the alarm.

"GetAcknowledgementTime" method

Return time of alarm acknowledgment.

```
CFRESULT GetAcknowledgementTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of alarm acknowledgment.

"GetClearTime" method

Return time of alarm reset

```
CFRESULT GetClearTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of alarm reset.

"GetResetTime" method

Return time of alarm reset.

```
CFRESULT GetResetTime(CFDATETIME64 *value)
```

`value`
[out]: Points out the time of the alarm reset.

"GetSuppressionState" method

Return status of alarm visibility.

```
CFRESULT GetSuppressionState(uint8_t *value)
```

`value`
[out]: Points to the status of the alarm visibility.

"GetPriority" method

Return relevance for display and sorting of the alarm.

```
CFRESULT GetPriority(uint8_t *value)
```

`value`
[out]: Points to the relevance of the alarm.

"GetOrigin" method

Return origin for display and sorting of the alarm

```
CFRESULT GetOrigin(CFSTR *value)
```

`value`
[out]: Points to the origin of the alarm.

"GetArea" method

Return origin area for display and sorting of the alarm.

```
CFRESULT GetArea(CFSTR *value)
```

`value`
[out]: Points to the origin area of the alarm.

"GetValue" method

Return current process value of the alarm.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the current process value of the alarm.

"GetValueQuality" method

Return quality of the process value of the alarm.

```
CFRESULT GetValueQuality(uint16_t *value)
```

`value`
[out]: Points to the quality of the process value of the alarm.

"GetValueLimit" method

Return limit of the process value of the alarm.

```
CFRESULT GetValueLimit(CFVARIANT *value)
```

`value`
[out]: Points to the limit of the process value of the alarm.

"GetUserName" method

Return user name of the operator input alarm.

```
CFRESULT GetUserName(CFSTR *value)
```

`value`
[out]: Points to the user name of the operator input alarm.

"GetLoopInAlarm" method

Return function that navigates from the alarm view to its origin.

```
CFRESULT GetLoopInAlarm(CFSTR *value)
```

`value`
[out]: Points to the function name that navigates to the origin of the alarm.

"GetAlarmParameterValues" method

Return parameter values of the alarm.

```
CFRESULT GetAlarmParameterValues(CFVARIANT *value)
```

`value`
[out]: Points to the parameter values of the alarm.

"GetInvalidFlags" method

Return marking of the alarm with invalid data.

```
CFRESULT GetInvalidFlags(uint8_t *value)
```

`value`
[out]: Points to the invalid data of the alarm.

"GetConnection" method

Return connection by which the alarm was triggered.

```
CFRESULT GetConnection CFSTR *value)
```

`value`
[out]: Points to the connection of the alarm.

"GetSystemSeverity" method

Return severity of the system error.

```
CFRESULT GetSystemSeverity(uint8_t *value)
```

`value`
[out]: Points to the severity of the system error.

"GetUserResponse" method

Return expected or required user response to the alarm.

```
CFRESULT GetUserResponse(uint8_t *value)
```

`value`
[out]: Points to the expected or required user response to the alarm.

"GetSourceType" method

Return source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
CFRESULT GetSourceType(uint16_t *value)
```

`value`
[out]: Points to the type of source.

"GetDeadBand" method

Return range of the triggering tag in which no alarms are generated.

```
CFRESULT GetDeadBand(CFVARIANT *value)
```

`value`
[out]: Points to the non-triggering range.

"GetId" method

Return ID of the alarm that is also used in the display.

```
CFRESULT GetId(uint32_t *value)
```

`value`
[out]: Points to the ID of the alarm.

"GetAlarmGroupID" method

Return the ID of the alarm group to which the alarm belongs.

```
CFRESULT GetAlarmGroupID(uint32_t* groupId)
```

`groupId`
[out]: The ID of the alarm group

"GetHostName" method

Return name of the host that triggered the alarm.

```
CFRESULT GetHostName(CFSTR *value)
```

`value`
[out]: Points to the host name.

"GetNotificationReason" method

Return the reason for the notification.

```
CFRESULT GetNotificationReason(CFENUM* value)
```

`value`:
[out]: Points to the enumeration of the notification reason. The notification reason can have the following values:

- `Unknown` (0)
- `Add` (1)
  The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring.
- `Modify` (2)
  Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- `Remove` (3)
  The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

  Note

  Changes to the alarm only lead to notifications again when the alarm meets the filter criteria again. In this case, "NotificationReason" is set to `Add`.

Note

Removing alarm from business logic

The use case of the client determines whether the client ignores notifications via alarms with the "NotificationReason" `Modify` or `Remove`.

For example:

- State-based monitoring: The client wants to show a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
- Event-based monitoring: The client wants to send an email when an alarm comes in. Only the notification reason `Add` is relevant.

Example:

An ODK client begins monitoring with the filter criterion "State" = 1. An alarm is triggered. Runtime notifies the ODK client of the "NotificationReason" as follows:

Notification-Reason	Description
Add	• The "State" property is 1. The alarm has come in.
Modify	• The "State" property has not changed. • Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	The "State" property has changed, e.g. alarm has gone out.

Example

When alarm are incoming, output a selection of properties of the "IAlarmResult" objects:

Copy code

```
// Callback for alarm notifications
CFRESULT CFCALLTYPE CAlarmValue::OnAlarm(IAlarmResultEnumerator* pItems, uint32_t
systemError, CFSTR systemName, CFBOOL completed)
{
    CFRESULT hr = CF_FALSE;
    CFBOOL bSet = false;
    uint32_t nsize;
    pItems->Count(&nsize);

    if (nsize > 0 && CF_SUCCEEDED(systemError)) {
        m_AlarmValue.clear();
        int index = 0;

        while (CF_SUCCEEDED(pItems->MoveNext()))
        {
            IAlarmResultPtr ppValues;
            if (CF_SUCCEEDED(pItems->Current(&ppValues)))
            {
                AlarmAttributes AlarmValue;
                AlarmValue.m_nInstanceID;
                ppValues->GetInstanceID(&AlarmValue.m_nInstanceID);
                AlarmValue.m_strSourceID;
                CCfString strId;
                ppValues->GetSourceID(&strId);
                AlarmValue.m_strSourceID = CCfSmartString(strId);
                CCfString strName;
                ppValues->GetName(&strName);
                AlarmValue.m_strName = CCfSmartString(strName);
                CCfString strClassName;
                ppValues->GetAlarmClassName(&strClassName);
                AlarmValue.m_strAlarmClassName = CCfSmartString(strClassName);
                ppValues->GetState(&AlarmValue.m_nState);
                CCfString strEvent;
                ppValues->GetEventText(&strEvent);
                AlarmValue.m_strEventText = CCfSmartString(strEvent);
                CCfString strText;
                ppValues->GetStateText(&strText);
                AlarmValue.m_strStateText = CCfSmartString(strText);
                ppValues->GetBackColor(&AlarmValue.m_nBackColor);
                ppValues->GetTextColor(&AlarmValue.m_nTextColor);
                ppValues->GetFlashing(&AlarmValue.m_bFlashing);

                ...

                AlarmValue.m_nAlarmsSize = nsize;

                AlarmValue.m_strSystemName = systemName;
               std::cout << "System name = " << AlarmValue.m_strSystemName.ToUTF8().c_str()
<< std::endl;

                m_AlarmValue.push_back(AlarmValue);
                std::cout << "Alarm name = " << strName.ToUTF8().c_str() << std::endl;
            }
            index++;
        }
```

Copy code

```
        this->SetEvent();
        return CF_SUCCESS;
    }
    return hr;
}
```

See also

IAlarmResultEnumerator (Page 1312)

IAlarm (Page 1313)

IAlarmSubscription (Page 1338)

19.10.4.2 IAlarmResultEnumerator (RT Uni)

Description

The "IAlarmResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of active alarms of the Runtime system. Through the enumeration you access individual alarms from the quantity of all active alarm of the runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

`CFRESULT Current(IAlarmResult **ppItem)`

`ppItem`
[out]: Points to the current "IAlarmResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

`CFRESULT MoveNext()`

"Reset" method

Reset the current position in the enumeration of a list.

`CFRESULT Reset()`

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of elements of the list.

Example

When alarms are incoming, the "IAlarmResult" objects enumerate:

Copy code

```
// Callback for alarm notifications
CFRESULT CFCALLTYPE CAlarmValue::OnAlarm(IAlarmResultEnumerator* pItems, uint32_t
systemError, CFSTR systemName, CFBOOL completed)
{
    CFRESULT hr = CF_FALSE;
    CFBOOL bSet = false;
    uint32_t nsize;
    pItems->Count(&nsize);

    if (nsize > 0 && CF_SUCCEEDED(systemError)) {
        m_AlarmValue.clear();
        int index = 0;

        while (CF_SUCCEEDED(pItems->MoveNext()))
        {
            IAlarmResultPtr ppValues;
            if (CF_SUCCEEDED(pItems->Current(&ppValues)))
            {
                //AlarmResult processing...
            }
            index++;
        }
        this->SetEvent();
        hr = CF_SUCCESS;
    }
    return hr;
}
```

See also

IAlarmResult (Page 1302)

19.10.4.3 IAlarm (RT Uni)

Description

The C++ interface "IAlarm" specifies properties and methods for handling active alarms of the Runtime system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetSourceID" method

Set the ID of the source at which the active alarm was triggered.

```
CFRESULT SetSourceID(CFVARIANT sourceID)
```

`sourceID`
[in]: ID of the source of the active alarm

"SetName" method

Set the name of the active alarm.

```
CFRESULT SetName(CFSTR name)
```

`name`
[in]: Name of the active alarm

"GetName" method

Read out name of the active alarm.

```
CFRESULT GetName(CFSTR *name)
```

`name`
[out]: Points to a name of the active alarm.

"SetErrorCode" method

Set error code of the alarm.

```
CFRESULT SetErrorCode(uint32_t errorCode)
```

`errorCode`
[in]: Error code

"GetError" method

Read out error code of the alarm.

```
CFRESULT GetError(uint32_t *error)
```

`error`
[out]: Error code

"Disable" method

Deactivate generation of the alarm in the alarm source synchronously.

```
CFRESULT Disable()
```

"Enable" method

Activate generation of the alarm in the alarm source synchronously again.

```
CFRESULT Enable()
```

"Shelve" method

Hide active alarm synchronously.

```
CFRESULT Shelve()
```

"Unshelve" method

Display hidden alarm synchronously again.

```
CFRESULT Unshelve()
```

"Acknowledge" method

Acknowledge active alarm or instance of an active alarm synchronously.

```
CFRESULT Acknowledge(uint32_t instanceId)
```

- `instanceId`
  Value "0": Acknowledge active alarm.
  Value > "0": Acknowledge instance with this ID.

"Reset" method

Acknowledge outgoing state of an active alarm or an instance of an active alarm synchronously.

```
CFRESULT Reset(uint32_t instanceId)
```

- `instanceId`
  Value "0": Acknowledge the counter state of the active alarm.
  Value > "0": Acknowledge the counter state of an instance with this ID.

Example

Acknowledge list of active alarm synchronously:

Copy code

```
vector<AlarmAttributes> g_vecAlarmList;

void AcknowledgeAlarmSync(IRuntimePtr pRuntime)
{
    SubscribeAlarm(pRuntime);
    ICfUnknownPtr pUnk;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"Alarm"), &pUnk)))
    {
        IAlarmPtr  pAlarm(pUnk);
        if (g_vecAlarmList.size() > 0)
        {
            vector<AlarmAttributes>::iterator it = g_vecAlarmList.begin();

            // iterate through list of notified alarms and acknowledge each alarm
            while (it != g_vecAlarmList.end())
            {
                CCfVariant vtAlarmName = it->m_strName;
                pAlarm->SetName(vtAlarmName);
                CFRESULT errCode = pAlarm->Acknowledge();
                if (CF_FAILED(errCode))
                {
                    std::wcout << L"Sync Acknowledge failed" << endl;
                    PrintErrorInformation(errCode, L"Acknowledge", pRuntime);
                }
                it++;
            }
        }
        g_vecAlarmList.clear();
    }
}
```

See also

IAlarmResult (Page 1302)

19.10.4.4 IAlarmCallback (RT Uni)

Description

The C++ interface "IAlarmCallback" defines methods for implementing asynchronous operations for monitoring active alarms. The methods are used by the "IAlarmSubscription" interface.

Members

The following methods are specified in the "IAlarmCallback" interface:

"OnAlarm" method

Callback method is called when active alarms on monitored systems change the alarm state.

The "OnAlarm" callback method is called when the "IAlarmSubscription.Start" method is used.

```
CFRESULT OnAlarm(
         IAlarmResultEnumerator* pItems,
         uint32_t systemError,
         CFSTR systemName,
         CFBOOL completed)
```

- `pItems`
  [in]: Points to an "IAlarmResultEnumerator" object that contains the enumeration of the changed active alarms.
- `systemError`
  [in]: Error code for the asynchronous operation
- `systemName`
  [in]: System of the associated Runtime system.
- `completed`
  [in]: True, if no more data from the callback can be expected.

"OnPendingAlarmsComplete" method

Callback method is called to display the completion of the handling of all the active alarms of a system.

The "OnPendingAlarmsComplete" callback method is called when the "IAlarmSubscription.Start" method is used.

```
CFRESULT OnPendingAlarmsComplete()
```

Example

In the following section monitored active alarms are written asynchronously into the "g_vecAlarmList" map at a change. To this purpose the "SubscribeAlarm" function uses a "CAlarmValue" object that implements the "IAlarmCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "CAlarmValue" object is determined via reference counting.

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void SubscribeAlarm(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk)))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback *pCB = pAlarmValue;

        CCfDafeArrayBound bounds(1UL, 0);

        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;

        CCfSmartString daFilter = L"";

        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);

        CCfVariant daLanguage = 1033;
        CFRESULT errCode;

        daSystemID.Detach(&vSystemIDs);

        CCfSmartString strFilter = L"";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);

        errCode = pAlarm->Start(pCB);
        if (errCode == CF_SUCCESS)
        {
            std::wcout<< "Subscription Success"<<endl;
        }

        // Wait for alarm nofications
        if (pAlarmValue->WaitForcompletion(g_nMaxWaitTime) == CF_SUCCESS)
        {
            errCode = pAlarm->Stop();
            if (CF_FAILED(errCode))
            {
                std::wcout << "CancelSubscribe failed" << endl;
                PrintErrorInformation(errCode, L"CancelSubscribe", pRuntime);
            }

            // Get current alarms from callback
            pAlarmValue->GetAlarmAttributes(g_vecAlarmList);
```

Copy code

```
            }
        }
}
```

See also

IAlarmSubscription (Page 1338)

19.10.4.5 IAlarmSourceCommandCallback (RT Uni)

Description

The C++ interface "IAlarmSourceCommandCallback" defines methods for implementing asynchronous operations with active alarms. The methods are used by the "IAlarmSet" interface.

Members

The following methods are specified in the interface:

"OnAcknowledge" method

Callback method is called when an active alarm was acknowledged.

The "OnAcknowledge" callback method is called when the "IAlarmSet.Acknowledge" and "IAlarmSet.AcknowledgeInstance" methods are used.

```
CFRESULT OnAcknowledge(
        CFRESULT SystemError,
        IAlarmSetResultEnumerator *AlarmSetResult,
        CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnReset" method

Callback method is called when an active alarm was removed.

The "OnReset" callback method is called when the "IAlarmSet.Reset" and "IAlarmSet.ResetInstance" methods are used.

```
CFRESULT OnReset(
        CFRESULT SystemError,
         IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnDisable" method

Callback method is called when the generation of the alarm in the alarm source was deactivated.

The "OnDisable" callback method is called when the "IAlarmSet.Disable" method is used.

```
CFRESULT OnDisable(
         CFRESULT SystemError,
         IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnEnable" method

Callback method is called when the generation of the alarm in the alarm source was reactivated.

The "OnEnable" callback method is called when the "IAlarmSet.Enable" method is used.

```
CFRESULT OnEnable(
         CFRESULT SystemError,
         IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- SystemError
  [in]: Basic errors that occurred during the asynchronous transfer.
- AlarmSetResult
  [in]: Points to an enumeration with the alarms of the callback
- MoreFollows
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnShelve" method

Callback method is called when an active alarm was hidden.

The "OnShelve" callback method is called when the "IAlarmSet.Shelve" method is used.

```
CFRESULT OnShelve(
         CFRESULT SystemError,
         IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- SystemError
  [in]: Basic errors that occurred during the asynchronous transfer.
- AlarmSetResult
  [in]: Points to an enumeration with the alarms of the callback
- MoreFollows
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnUnShelve" method

Callback method is called when an active alarm was displayed.

The "OnUnshelve" callback method is called when the "IAlarmSet.Unshelve" method is used.

```
CFRESULT OnUnshelve(
         CFRESULT SystemError,
        IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- SystemError
  [in]: Basic errors that occurred during the asynchronous transfer.
- AlarmSetResult
  [in]: Points to an enumeration with the alarms of the callback
- MoreFollows
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

Example

In the following section monitored active alarms of the "g_vecAlarmList" vector are acknowledged asynchronously. To this purpose the "AcknowledgeAlarmAsync" function uses a "CAlarmSourceCommandCB" object that implements the "IAlarmSourceCommandCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "CAlarmSourceCommandCB" object is determined via reference counting.

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void AcknowledgeAlarmAsync(IRuntimePtr pRuntime)
{
    SubscribeAlarm(pRuntime);
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSet"), &pUnk)))
    {
        IAlarmSetPtr pAlarmSet(pUnk);
        CAlarmSourceCommandCB* pAlarmSoureCommand = new CAlarmSourceCommandCB();
        pAlarmSoureCommand->AddRef();
        IAlarmSourceCommandCallback* pCB = pAlarmSoureCommand;
        if (g_vecAlarmList.size() > 0)
        {
            vector<AlarmAttributes>::iterator it = g_vecAlarmList.begin();
            // iterate through list of notified alarms and acknowledge each alarm
            while (it != g_vecAlarmList.end())
            {
                IAlarm* pAlarm = nullptr;
                CCfVariant vtAlarmName = it->m_strName;
                errCode = pAlarmSet->Add(vtAlarmName, &pAlarm);
                if (CF_FAILED(errCode))
                {
                    PrintErrorInformation(errCode, L"AlarmSet", pRuntime);
                }
                it++;
            }
            // acknowledged the AlarmSet
            errCode = pAlarmSet->Acknowledge(pCB);
            // wait for acknowledge callback
            if (CF_SUCCEEDED(errCode) && pAlarmSoureCommand-
>WaitForcompletion(g_nMaxWaitTime) == CF_SUCCESS)
            {
                //Check if an alarm could not be acknowledged
                pAlarmSoureCommand->PrintError(pRuntime, L"Acknowledge");
            }
            g_vecAlarmList.clear();
        }
    }
}
```

See also

IAlarmSet (Page 1324)

19.10.4.6 IAlarmSet (RT Uni)

Description

The C++ interface "IAlarmSet" specifies properties and methods for optimized access to several active alarms of the Runtime system.

After initialization of the "IAlarmSet" object, you can execute asynchronous operations with multiple alarms in one call, e. g. acknowledgment or commenting. Simultaneous access demonstrates better performance and lower communication load than single access to multiple alarms.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"GetCount" method

Return the number of alarms of an AlarmSet list.

```
CFRESULT GetCount(uint32_t *value)
```

`value`
[out]: Points to the value for the number of alarms of the AlarmSet list.

"Remove" method

Remove a single alarm or an instance of an alarm from an AlarmSet.

```
CFRESULT Remove(CFSTR name, uint32_t instanceId)
```

- `name`
  [in]: Name of the alarm that is removed from the AlarmSet.
- `instanceId`
  [in]:
  Value = "0": Remove active alarm.
  Value > "0": Remove instance with this ID.

"Add" method

Add an active alarm or instance of the alarm to an AlarmSet.

```
CFRESULT Add(
        CFVARIANT p_varName,
         IAlarm** pAlarm,
         uint32_t instanceId)
```

- `p_varName`
  [in]: Name of alarm tag that is added to the AlarmSet.
- `pAlarm`
  [out]: Points to the added "IAlarm" object of the AlarmSet.
- `instanceId`
  [in]:
  Value = "0": Add active alarm.
  Value > "0": Add instance with this ID.

"Get" method

Reference an alarm or an instance of an alarm from an AlarmSet.

```
CFRESULT Get(
         const CFSTR p_varName,
         IAlarm** pAlarm,
         uint32_t instanceId)
```

- `p_varName`
  [in]: Name of the alarm tag.
- `pAlarm`
  [out]: Points to the "IAlarm" object of the AlarmSet.
- `instanceId`
  [in]:
  Value = "0": Reference active alarm.
  Value > "0": Reference instance with this ID.

"Disable" method

Deactivate generation of the alarms of the AlarmSet in the alarm source asynchronously.

```
CFRESULT Disable(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Enable" method

Reactivate generation of the alarms of the AlarmSet in the alarm source asynchronously.

```
CFRESULT Enable(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Shelve" method

Hide alarms of the AlarmSet asynchronously.

```
CFRESULT Shelve(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Unshelve" method

Display hidden alarms of the AlarmSet once again.

```
CFRESULT Unshelve(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Acknowledge" method

Acknowledge alarms of the AlarmSet asynchronously.

```
CFRESULT Acknowledge(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Reset" method

Acknowledge outgoing state of the alarms of the AlarmSet asynchronously.

```
CFRESULT Reset(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Clear" method

Remove all alarms from an AlarmSet.

```
CFRESULT Clear()
```

Example

Remove active alarms from the "g_vecAlarmList" list as an AlarmSet asynchronously:

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;
SubscribeAlarm(pRuntime);
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSet"), &pUnk)))
    {
        IAlarmSetPtr pAlarmSet(pUnk);
        CAlarmSourceCommandCB* pAlarmSoureCommond = new CAlarmSourceCommandCB();
        pAlarmSoureCommond->AddRef();
        IAlarmSourceCommandCallback* pCB = pAlarmSoureCommond;
        if (g_vecAlarmList.size() > 0)
        {
            vector<AlarmAttributes>::iterator it = g_vecAlarmList.begin();
            // iterate through list of notified alarms and reset each alarm
            while (it != g_vecAlarmList.end())
            {
                IAlarm* palarm = nullptr;
                CCfVariant vtAlarmName = it->m_strName;
                errCode = pAlarmSet->Add(vtAlarmName, &palarm);
                if (CF_FAILED(errCode))
                {
                    PrintErrorInformation(errCode, L"AlarmSet", pRuntime);
                }
                it++;
            }
            // Reset the AlarmSet
            errCode = pAlarmSet->Reset(pCB);
            if (CF_SUCCEEDED(errCode) && pAlarmSoureCommond-
>WaitForcompletion(g_nMaxWaitTime) == CF_SUCCESS)
            {
                //Check if an alarm could not be Reset
                pAlarmSoureCommond->PrintError(pRuntime, L"Reset");
            }
            g_vecAlarmList.clear();
        }
    }
}
```

See also

IAlarmSourceCommandCallback (Page 1320)

IAlarmSetResult (Page 1328)

IAlarmSetResultEnumerator (Page 1329)

19.10.4.7 IAlarmSetResult (RT Uni)

Description

The C++ interface "IAlarmSetResult" specifies methods for accessing properties of monitored alarms in AlarmSets.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"GetSystemName" method

Return system name of the Runtime system of an alarm in the AlarmSet.

```
CFRESULT GetSystemName(CFSTR *value)
```

`value`
[out]: Points to the associated system name.

"GetName" method

Return name of an alarm in the AlarmSet.

```
CFRESULT GetName(CFSTR *value)
```

`value`
[out]: Points to the name of an alarm.

"GetInstanceID" method

Return InstanceID of an alarm in the AlarmSet.

```
CFRESULT GetInstanceID(uint32_t *value)
```

`value`
[out]: Points to the InstanceID of an alarm

"GetErrorCode" method

Return error code of an alarm in the AlarmSet.

```
CFRESULT GetErrorCode(uint32_t *value)
```

`value`
[out]: Points to the error code of an alarm

See also

IAlarmSet (Page 1324)

IAlarmSetResultEnumerator (Page 1329)

IAlarmSubscription (Page 1338)

19.10.4.8 IAlarmSetResultEnumerator (RT Uni)

Description

The "IAlarmSetResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of monitored alarms of the Runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IAlarmSetResult **ppItem)
```

`ppItem`
[out]: Points to the current "IAlarmSetResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of elements of the list.

See also

IAlarmSet (Page 1324)

IAlarmSetResult (Page 1328)

19.10.4.9 IAlarmTrigger (RT Uni)

Description

The C++ interface "AlarmTrigger" specifies methods for triggering alarms.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"CreateSystemAlarm" method

Generates an alarm of the class SystemAlarmWithoutClearEvent with the state machine alarm without outgoing status with acknowledgment.

```
CFRESULT CreateSystemAlarm(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- `alarmText`
  [in]: The alarm text. You have the following options:
  – Transferring a text list of type "ITextList".
    The list entries of the text list can contain the multilingual text or references to other text lists.

    Note

    Only user-definedtext lists

    This method processes only user-defined text lists.

  – Transfer static string with monolingual text.
- `area`
  [in]: The area of origin of the alarm
- `p_AlarmParameterValue`<Number>
  [in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm
  texts:
  `@ScriptingSystemEvents.SystemAlarmWithoutClearEvent:SystemAlarmWithoutClearEvent`
- For monolingual alarm
  text:
  `@ScriptingSystemEvents.SystemAlarmWithoutClearEventText:SystemAlarmWithoutClearEvent`

"CreateSystemInformation" method

Generates an alarm of the class SystemInformation with the state machine alarm without outgoing status without acknowledgment.

```
CFRESULT CreateSystemInformation(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- `alarmText`
  [in]: The alarm text. You have the following options:
  - Transferring a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-definedtext lists

    This method processes only user-defined text lists.
  - Transfer static string with monolingual text.
- `area`
  [in]: The area of origin of the alarm
- `p_AlarmParameterValue`<Number>
  [in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts: `@ScriptingSystemEvents.SystemInformation:SystemInformation`
- For monolingual alarm text: `@ScriptingSystemEvents.SystemInformationText:SystemInformation`

"CreateOperatorInputInformation" method

Generates an alarm of the class OperatorInputInformation with the state machine alarm without outgoing status without acknowledgment.

```
CFRESULT CreateOperatorInputInformation(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- `alarmText`
  [in]: The alarm text. You have the following options:
  - Transferring a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-definedtext lists

    This method processes only user-defined text lists.
  - Transfer static string with monolingual text.
- `area`
  [in]: The area of origin of the alarm
- `p_AlarmParameterValue`<Number>
  [in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm
  texts:
  `@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputInformation`
- For monolingual alarm
  text:
  `@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputInformation`

Example

The following code examples show how to trigger alarms of the class SystemAlarmWithoutClearEvent. Alarms of the classes SystemInformation and OperatorInputInformation are triggered in the same way.

Copying code

```
void CreateSystemAlarm(IRuntimePtr pRuntime)
{
   //Create SystemAlarm with monolingual alarm text
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk)))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback* pCB = pAlarmValue;
        CCfSafeArrayBound bounds(1UL, 0);
        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);
        CCfSmartString strFilter = L"AlarmClassName = 'SystemAlarmWithoutClearEvent'";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);
        errCode = pAlarm->Start(pCB);
        if (CF_SUCCEEDED(errCode))
        {
            ICfUnknownPtr pUnktrig;
            if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnktrig)))
            {
                IAlarmTriggerPtr pTrigger(pUnktrig);
                if (pTrigger != nullptr)
                {
                    errCode = pTrigger->CreateSystemAlarm(CCfVariant(L"Alarm Text"),
CCfString(L"Alarm Area"), CCfVariant(L"param1"), CCfVariant(L"param2"),
CCfVariant(L"param3"), CCfVariant(L"param4"), CCfVariant(L"param5"), CCfVariant("param6"),
CCfVariant("param7"));
                    if (CF_FAILED(errCode))
                    {
                        PrintErrorInformation(errCode, L"AlarmTrigger", pRuntime);
                    }
                }
            }
        }
        // Wait for alarm nofications
        if (CF_SUCCEEDED(errCode) && pAlarmValue->WaitForcompletion(g_nMaxWaitTime) ==
CF_SUCCESS)
        {
            errCode = pAlarm->Stop();
            if (CF_FAILED(errCode))
            {
                PrintErrorInformation(errCode, L"AlarmSubscription", pRuntime);
            }
            pAlarmValue->Release();
```

Copying code

```
        }
    }
}


void CreateSystemAlarmWithAlarmTextAsTextList(IRuntimePtr pRuntime)
{
   //Create SystemAlarm with multilingual alarm text; the tranlsations are directly stored
in the text list
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk)))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback* pCB = pAlarmValue;
        CCfSafeArrayBound bounds(1UL, 0);
        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);
        CCfSmartString strFilter = L"AlarmClassName = 'SystemAlarmWithoutClearEvent'";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);
        errCode = pAlarm->Start(pCB);
        if (CF_SUCCEEDED(errCode))
        {
            ICfUnknownPtr pUnktrig;
            if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnktrig)))
            {
                IAlarmTriggerPtr pTrigger(pUnktrig);
                if (pTrigger != nullptr)
                {
                   if (CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"), &pUnktrig)))
                    {
                        ITextListPtr ptextList(pUnktrig);
                        if (nullptr != ptextList)
                        {
                            ptextList->SetName(CCfString(L"AlarmTextTemplate"));
                            ptextList->SetTextListEntryIndex(101);
                            errCode = pTrigger->CreateSystemAlarm(CCfVariant(ptextList),
CCfString(L"Alarm Area"), CCfVariant(L"param1"), CCfVariant(L"param2"),
CCfVariant(L"param3"), CCfVariant(L"param4"), CCfVariant(L"param5"), CCfVariant("param6"),
CCfVariant("param7"));
                            if (CF_FAILED(errCode))
                            {
                                PrintErrorInformation(errCode, L"AlarmTrigger", pRuntime);
                            }
                        }
```

Copying code

```
                    }
                }
            }
        }
        // Wait for alarm nofications
        if (CF_SUCCEEDED(errCode) && pAlarmValue->WaitForcompletion(g_nMaxWaitTime) == 
CF_SUCCESS)
        {
            errCode = pAlarm->Stop();
            if (CF_FAILED(errCode))
            {
                PrintErrorInformation(errCode, L"AlarmSubscription", pRuntime);
            }
            pAlarmValue->Release();
        }
    }
}


void CreateSystemAlarmWithTextListAsParameterValue(IRuntimePtr pRuntime)
{
    //
Create SystemAlarm with multilingual alarm text; the text list references other text lists 
with tranlsations
    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk)))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback* pCB = pAlarmValue;
        CCfSafeArrayBound bounds(1UL, 0);
        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);
        CCfSmartString strFilter = L"AlarmClassName = 'SystemAlarmWithoutClearEvent'";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);
        errCode = pAlarm->Start(pCB);
        if (CF_SUCCEEDED(errCode))
        {
            ICfUnknownPtr pUnktrig;
            if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnktrig)))
            {
                IAlarmTriggerPtr pTrigger(pUnktrig);
                if (pTrigger != nullptr)
                {
                    ICfUnknownPtr pUnktextList, pUnktextList1;
```

Copying code

```
                    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"),
&pUnktextList)) && CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"),
&pUnktextList1)))
                    {
                        ITextListPtr pTextList_1(pUnktextList);
                        if (nullptr != pTextList_1)
                        {
                            pTextList_1->SetName(CCfString(L"Text_List_2"));
                       pTextList_1->SetTextListEntryIndex(1); //Eng TL @1%t#2T@ Val: @3%s@
                        }
                        ITextListPtr pTextList_2(pUnktextList1);
                        if (nullptr != pTextList_2)
                        {
                            pTextList_2->SetName(CCfString(L"Text_List_2"));
                        }
                        errCode = pTrigger->CreateSystemAlarm(CCfVariant(pTextList_1),
CCfString(L"Alarm Area"),
                            CCfVariant(1), // Index for Text_list_2
                            CCfVariant(pTextList_2), // text list object
                            CCfVariant(L"Hello"), // Dynamic value of @3%s@
                            CCfVariant(), CCfVariant(), CCfVariant(), CCfVariant());
                        if (CF_FAILED(errCode))
                        {
                            PrintErrorInformation(errCode, L"AlarmTrigger", pRuntime);
                        }
                    }
                }
            }
        }
        // Wait for alarm nofications
        if (CF_SUCCEEDED(errCode) && pAlarmValue->WaitForcompletion(g_nMaxWaitTime) ==
CF_SUCCESS)
        {
            errCode = pAlarm->Stop();
            if (CF_FAILED(errCode))
            {
                PrintErrorInformation(errCode, L"AlarmSubscription", pRuntime);
            }
            pAlarmValue->Release();
        }
    }
}
```

19.10.4.10 ITextList (RT Uni)

Description

The C++ interface "ITextList" is used to transfer multilingual alarm texts for system alarms and operator input alarms. See section IAlarmTrigger (Page 1329), CreateSystemInformation method. An ITextList instance is passed to the alarm text. When the operator input alarm is generated, it is replaced by the configured text from the text list.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Returns the name of the text list.

```
GetName(CFSTR* name)
```

- `name`:
  [out]: The name

"SetName" method

Set the name of the text list.

```
SetName(CFSTR name)
```

- `name`:
  [in]: The name

"GetTextListEntryIndex" method

Return the index of the list entry.

```
GetTextListEntryIndex)(OUT uint32_t* pIndex)
```

- `pIndex`
  [out]: The index

"SetTextListEntryIndex" method

Set the index of the list entry.

```
SetTextListEntryIndex)(IN uint32_t pIndex)
```

- `pIndex`
  [in]: The index

19.10.4.11 IAlarmSubscription (RT Uni)

Description

The C++ interface "IAlarmSubscription" specifies methods for monitoring tags of the Runtime system. The subscribed tags are monitored for a change of alarm state.

Members

The following methods are specified in the interface:

"Start" method

Start monitoring of active alarms.

```
CFRESULT Start(IAlarmCallback* callbackPtr)
```

`callbackPtr`
[in/out]: Points to an "IAlarmCallback" object that implements the asynchronous monitoring.

"Stop" method

Stop monitoring of active alarms.

Note

Start and Stop in Windows Forms applications

Do not call the "Stop" method for a Windows Forms application in the same thread where you called "Start".

```
CFRESULT Stop()
```

"SetSystemNames" method

Set system names of Runtime systems for monitoring of active alarms.

```
CFRESULT SetSystemNames(CFVARIANT systemIDs)
```

`systemIDs`
[in]: System names of Runtime systems

"GetSystemNames" method

Read out system names of Runtime systems for monitoring of active alarms.

```
CFRESULT GetSystemNames(CFVARIANT* systemIDs)
```

`systemIDs`
[out]: Points to system names of Runtime systems.

"SetLanguage" method

Set country identification of the language for monitored alarms.

```
CFRESULT SetLanguage(uint32_t language)
```

`language`
[in]: Country identification of the language

"GetLanguage" method

Read out country identification of the language for monitored alarms.

```
CFRESULT GetLanguage(uint32_t* language)
```

`language`
[out]: Points to country identification of the language.

"GetFilter" method

Supplies the string by which the result set is filtered.

```
CFRESULT GetFilter(CFSTR* filter)
```

- `filter`
  [out]: SQL-type string for filtering the result set of active alarms.

"SetFilter" method

Sets the string for filtering the result set of active alarms.

```
CFRESULT SetFilter(IN CFSTR filter)
```

- `filter`
  [in]: SQL-type string for filtering the result set of active alarms.
  All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 1084).

Example

In the following section monitored active alarms are written asynchronously into the "g_vecAlarmList" map at a change. To this purpose the "SubscribeAlarm" function uses a "CAlarmValue" object that implements the "IAlarmCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "CAlarmValue" object is determined via reference counting.

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void SubscribeAlarm(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk)))
    {
        IAlarmSubscriptionPtr pAlarm(pUnk);
        CAlarmValue* pAlarmValue = new CAlarmValue();
        pAlarmValue->AddRef();
        IAlarmCallback *pCB = pAlarmValue;

        CCfSafeArrayBound bounds(1UL, 0);

        CCfSafeArray daAttribute;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
        CCfVariant daDataSource = 0;
        CCfVariant vSystemIDs = 0;

        CCfSmartString daFilter = L"";

        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);

        CCfVariant daLanguage = 1033;
        CFRESULT errCode;

        daSystemID.Detach(&vSystemIDs);

        CCfSmartString strFilter = L"";
        // Start Subscription
        pAlarm->SetFilter(strFilter.AllocCFSTR());
        pAlarm->SetLanguage(1033);
        pAlarm->SetSystemNames(vSystemIDs);

        errCode = pAlarm->Start(pCB);
        if (errCode == CF_SUCCESS)
        {
            std::wcout<< "Subscription Success"<<endl;
        }

        // Wait for alarm nofications
        if (pAlarmValue->WaitForcompletion(g_nMaxWaitTime) == CF_SUCCESS)
        {
            errCode = pAlarm->Stop();
            if (CF_FAILED(errCode))
            {
                std::wcout << "CancelSubscribe failed" << endl;
                PrintErrorInformation(errCode, L"CancelSubscribe", pRuntime);
            }

            // Get current alarms from callback
            pAlarmValue->GetAlarmAttributes(g_vecAlarmList);
```

Copy code

```
        }
    }
}
```

See also

IAlarmResult (Page 1302)

IAlarmCallback (Page 1316)

IAlarmSetResult (Page 1328)

19.10.4.12 ILoggedAlarmResult (RT Uni)

Description

The C++ interface "ILoggedAlarmResult" specifies methods for accessing properties of logged alarms of a logging system.

An "ILoggedAlarmResult" object is a pure data object that maps all properties of a logged alarm.

Members

The following methods are specified in the interface:

"GetInstanceID" method

Return InstanceID of a logged alarm.

```
CFRESULT GetInstanceID(uint32_t *value)
```

`value`
[out]: Points to the InstanceID of the logged alarm.

"GetName" method

Return name of the logged alarm.

```
CFRESULT GetName(CFSTR *value)
```

`value`
[out]: Points to the name of the logged alarm.

"GetAlarmClassName" method

Return name of the alarm class.

```
CFRESULT GetAlarmClassName(CFSTR *value)
```

`value`
[out]: Points to the symbol of the alarm class of the logged alarm.

"GetAlarmClassSymbol" method

Return symbol of the alarm class

```
CFRESULT GetAlarmClassSymbol(CFSTR *value)
```

`value`
[out]: Points to the name of the alarm class of the logged alarm.

"GetState" method

Return alarm state of the logged alarm.

```
CFRESULT GetState(int32_t* *value)
```

`value`
[out]: Points to the alarm state.

"GetStateText" method

Return alarm state of the logged alarm as text, for example "Incoming" or "Outgoing".

```
CFRESULT GetStateText(CFSTR *value)
```

`value`
[out]: Points to the alarm state as text.

"GetEventText" method

Return text that describes the alarm event of the logged alarm.

```
CFRESULT GetEventText(CFSTR *value)
```

`value`
[out]: Points to the text that describes the alarm event.

"GetAlarmText1GetAlarmText9" method

Return alarm texts 1-9 of the logged alarm.

```
CFRESULT GetAlarmText1(CFSTR *value)

…

CFRESULT GetAlarmText9(CFSTR *value)
```

`value`
[out]: Points to the additional text of the logged alarm.

"GetTextColor" method

Return text color of the alarm state of the logged alarm.

```
CFRESULT GetTextColor(uint32_t *value)
```

`value`
[out]: Points to the text color of the alarm state.

"GetBackColor" method

Return background color of the alarm state of the logged alarm.

```
CFRESULT GetBackColor(uint32_t *value)
```

`value`
[out]: Points to the background color of the alarm state.

"GetFlashing" method

Return flashing background color of the alarm state of the logged alarm.

```
CFRESULT GetFlashing(CFBOOL *value)
```

`value`
[out]: Points to the flashing background color of the alarm state.

"GetModificationTime" method

Time of the last change of the alarm state of the logged alarm.

```
CFRESULT GetModificationTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the last change of the alarm state.

"GetChangeReason" method

Return trigger event for the change of the alarm state of the logged alarm.

```
CFRESULT GetChangeReason(uint16_t *value)
```

`value`
[out]: Points to the trigger event for the change of the alarm state.

"GetRaiseTime" method

Return time at which the logged alarm was triggered.

```
CFRESULT GetRaiseTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the logged alarm trigger.

"GetAcknowledgementTime" method

Return time at which the logged alarm was acknowledged.

```
CFRESULT GetAcknowledgementTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the logged alarm acknowledgment.

"GetClearTime" method

Return time at which the logged alarm was cleared.

```
CFRESULT GetClearTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the logged alarm clearing.

"GetResetTime" method

Return time at which the logged alarm was reset.

```
CFRESULT GetResetTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the logged alarm reset.

"GetSuppressionState" method

Return status of the visibility of the logged alarm.

```
CFRESULT GetSuppressionState(uint8_t *value)
```

`value`
[out]: Points to the status of the visibility of the logged alarm.

"GetPriority" method

Return relevance for display and sorting of the logged alarm.

```
CFRESULT GetPriority(uint8_t *value)
```

`value`
[out]: Points to the relevance of the logged alarm.

"GetOrigin" method

Return origin for display and sorting of the logged alarm.

```
CFRESULT GetOrigin(CFSTR *value)
```

`value`
[out]: Points to the origin of the logged alarm.

"GetArea" method

Return origin area for display and sorting of the logged alarm.

```
CFRESULT GetArea(CFSTR *value)
```

`value`
[out]: Points to the origin area of the logged alarm.

"GetValue" method

Return process value of the logged alarm.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the process value of the logged alarm.

"GetValueQuality" method

Return quality of the process value of the logged alarm.

```
CFRESULT GetValueQuality(uint16_t *value)
```

`value`
[out]: Points to the quality of the process value of the logged alarm.

"GetValueLimit" method

Return limit of the process value of the logged alarm.

```
CFRESULT GetValueLimit(CFVARIANT *value)
```

`value`
[out]: Points to the limit of the process value of the logged alarm.

"GetUserName" method

Return user name of the logged operator input alarm.

```
CFRESULT GetUserName(CFSTR *value)
```

`value`
[out]: Points to the user name of the logged operator input alarm.

"GetLoopInAlarm" method

Return function that navigates from the alarm view to its origin.

```
CFRESULT GetLoopInAlarm(CFSTR *value)
```

`value`
[out]: Points to the function name that navigates to the origin of the logged alarm.

"GetAlarmParameterValues" method

Return parameter values of the logged alarm.

```
CFRESULT GetAlarmParameterValues(CFVARIANT *value)
```

`value`
[out]: Points to the parameter values of the logged alarm.

"GetInvalidFlags" method

Return marking of the logged alarm with invalid data.

```
CFRESULT GetInvalidFlags(uint8_t *value)
```

`value`
[out]: Points to the invalid data of the logged alarm.

"GetConnection" method

Return connection via which the logged alarm was triggered.

```
CFRESULT GetConnection CFSTR *value)
```

`value`
[out]: Points to the connection of the logged alarm.

"GetSystemSeverity" method

Return severity of the system error.

```
CFRESULT GetSystemSeverity(uint8_t *value)
```

`value`
[out]: Points to the severity of the system error.

"GetUserResponse" method

Return expected or required user response to the logged alarm.

```
CFRESULT GetUserResponse(uint8_t *value)
```

`value`
[out]: Points to the expected or required user response to the logged alarm.

"GetDeadBand" method

Return range of the triggering tag in which no alarms are generated.

```
CFRESULT GetDeadBand(CFVARIANT *value)
```

`value`
[out]: Points to the DeadBand of the logged alarm.

"GetHostName" method

Return name of the host that triggered the alarm.

```
CFRESULT GetHostName(CFSTR *value)
```

`value`
[out]: Points to host name.

"GetInfoText" method

Return text for the alarm that contains the associated work instruction.

```
CFRESULT GetInfoText(CFSTR *value)
```

`value`
[out]: Points to the text of the operator instruction.

"GetStateMachine" method

Return StateMachine model of the alarm. The StateMachine represents the behavior of alarms through arrangement of alarm states and alarm events, e.g. "RaiseClear", "RaiseRequiresAcknowledgment" or "RaiseClearOptionalAcknowledgment".

```
CFRESULT GetStateMachine(uint8_t *value)
```

`value`
[out]: Shows the model of the StateMachine of the logged alarm.

"GetSingleAcknowledgement" method

Returns whether an alarm may be acknowledged only individually or may be acknowledged as a group or multiple selection.

```
CFRESULT GetSingleAcknowledgement(CFBOOL *value)
```

`value`
[out]: Points to the acknowledgement specification.

"GetLoggedAlarmStateObjectID" method

Return ID of alarm state for referencing within the logging system.

```
CFRESULT GetLoggedAlarmStateObjectID(CFSTR *value)
```

`value`
[out]: Points to the ID of the alarm state of the logged alarm.

"GetID" method

Return user-defined ID of the alarm that is also used in the display.

```
CFRESULT GetID(uint32_t *value)
```

`value`
[out]: Points to the ID of the logged alarm.

"GetSourceType" method

Return source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
CFRESULT GetSourceType(uint16_t *value)
```

`value`
[out]: Points to the type of source of the logged alarm.

See also

ILoggedAlarmResultEnumerator (Page 1349)

IAlarmLogging (Page 1350)

IAlarmLoggingSubscription (Page 1354)

19.10.4.13 ILoggedAlarmResultEnumerator (RT Uni)

Description

The "ILoggedAlarmResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of logged alarms of a logging system. Through the enumeration you access individual alarms from the set of logged alarm of a logging system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(ILoggedAlarmResult **ppItem)
```

`ppItem`
[out]: Points to the current "ILoggedAlarmResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of elements of the list.

See also

ILoggedAlarmResult (Page 1343)

IAlarmLogging (Page 1350)

19.10.4.14 IAlarmLogging (RT Uni)

Description

The C++ interface "IAlarmLogging" specifies methods for reading out logged alarms of a logging system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"Read" method

Read out logged alarms of a time period synchronously from logging system.

```
CFRESULT Read(
         CFDATETIME64 begin,
         CFDATETIME64 end,
         CFSTR filter,
         uint32_t language,
         CFVARIANT systemIDs,
         ILoggedAlarmResultEnumerator **ppEnumerator)
```

- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `filter`
  [in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
  [in]: Country identification of the language of the logged alarm text

- `systemIDs`
  [in]: System names of the Runtime systems of the logged alarms. Default: local system
- `ppEnumerator`
  [out]: Points to the logged alarms as an "ILoggedAlarmResultEnumerator" object.

"ReadAsync" method

Read out logged alarms of a time period asynchronously from logging system.

```
CFRESULT ReadAsync(
         CFDATETIME64 begin,
         CFDATETIME64 end,
         CFSTR filter,
         uint32_t language,
         CFVARIANT systemIDs,
         IAlarmLoggingCallback *pLoggedAlarmCb)
```

- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `filter`
  [in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
  [in]: Country identification of the language of the logged alarm text
- `systemIDs`
  [in]: System names of the Runtime systems of the logged alarms. Default: local system
- `pLoggedAlarmCb`
  [in]: Points to the "IAlarmLoggingCallback" object that implements the callback interface.

Example

Read historical alarms synchronously from the logging system:

Copy code

```
void LoggingReadAlarmSync(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"LoggedAlarm"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        CCfVariant vSystemIDs = 0;
        CFSAFEARRAYBOUND* bounds = nullptr;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, bounds);
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);

        IAlarmLoggingPtr pAlarm(pUnk);

        CCfDateTime64 begin, end;
        begin = CCfDateTime64::Now(true);
        end = begin;
        begin.SubtractTimeSpan(Get1Minute() * 3);

        ILoggedAlarmResultEnumeratorPtr pItems;
        // Read value of tag
        errCode = pAlarm->Read(begin, end, CCfString(""), 1033, vSystemIDs, &pItems);

        if (pItems != nullptr && CF_SUCCEEDED(errCode))
        {
            std::wcout << "Read finished " << std::endl;
            PrintValues(pItems);
        }
        else
        {
            std::wcout << L"Read operation failed." << std::endl;
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
    }
}
```

See also

ILoggedAlarmResult (Page 1343)

ILoggedAlarmResultEnumerator (Page 1349)

IAlarmLoggingCallback (Page 1353)

IAlarmLoggingSubscription (Page 1354)

19.10.4.15 IAlarmLoggingCallback (RT Uni)

Description

The C++ interface "IAlarmLoggingCallback" defines methods for implementing asynchronous operations for monitoring active alarms. The methods are used by the "IAlarmLogging" and "IAlarmLoggingSubscription" interfaces.

All the methods return CF_SUCCESS after execution.

Members

The following methods are specified in the "IAlarmCallback" interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations in logging systems.

The "OnReadComplete" callback method is called when the "IAlarmLogging.ReadAsync" method is used.

```
CFRESULT OnReadComplete(ILoggedAlarmResultEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "ILoggegAlarmResultEnumerator" object that contains the enumeration of the logged alarms.
- `errorCode`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the logged alarms.

"OnDataChanged" method

Callback method is called upon a change of a monitored alarm in logging systems.

The "OnDataChanged" callback method is called when the "IAlarmLoggingSubscription.Start" method is used.

```
CFRESULT OnDataChanged(ILoggedAlarmResultEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "ILoggegAlarmResultEnumerator" object that contains the enumeration of the logged alarms.
- `errorCode`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the logged alarms.

See also

IAlarmLogging (Page 1350)

IAlarmLoggingSubscription (Page 1354)

19.10.4.16 IAlarmLoggingSubscription (RT Uni)

Description

The C++ interface "IAlarmLoggingSubscription" specifies methods for monitoring logged alarms of an archive system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetFilter" method

Set SQL-like string for filtering the result set of the logged alarms.

```
CFRESULT SetFilter(CFSTR filter)
```

`filter`
[in]: Filter string for logged alarms

"SetLanguage" method

Set country identifier of the language for monitoring of logged alarms.

```
CFRESULT SetLanguage(uint32_t language)
```

`language`
[in]: Country identification of the language

"SetSystemName" method

Set system names of Runtime systems for monitoring of logged alarms.

```
CFRESULT SetSystemName(CFVARIANT systemIDs)
```

`systemIDs`
[in]: System name of Runtime systems

"Start" method

Start monitoring of logged alarms.

```
CFRESULT Start(IAlarmLoggingCallback* pLoggedAlarmCb)
```

- `filter`
  [in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `pLoggedAlramCb`
  [in/out]: Points to an "IAlarmLoggingCallback" object that implements asynchronous monitoring.

"Stop" method

Stop monitoring of all logged alarms.

```
CFRESULT Stop()
```

Example

Monitoring logged alarms. The values are returned by the "IAlarmLoggingCallback" object:

Copy code

```
void LoggingSubscribeAlarm(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"LoggedAlarmSubscribtion"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errCode))
    {
        CCfVariant vSystemIDs = 0;
        CFSAFEARRAYBOUND* bounds = nullptr;
        CCfSafeArray daSystemID(CF_VT_SREF, 1, bounds);
        CCfSREF id(L"SYSTEM1");
        int32_t index = 0;
        daSystemID.PutElement(&index, &id);
        daSystemID.Detach(&vSystemIDs);

        IAlarmLoggingSubscriptionPtr pAlarm(pUnk);
        pAlarm->SetSystemName(vSystemIDs);
        pAlarm->SetLanguage(1033);

        COdkAlarmLoggingCB* pAlarmCB = new COdkAlarmLoggingCB();

        if (pAlarmCB != nullptr && CF_SUCCEEDED(errCode))
        {
            pAlarmCB->AddRef();

            // subscribe tags
            errCode = pAlarm->Start(pAlarmCB);
            if (CF_FAILED(errCode))
            {
                std::wcout << L"Error, couldn't create callback interface." << std::endl;
                PrintErrorInformation(errCode, L"Start", pRuntime);
            }
        }
    }
    else
    {
        std::wcout << L"Error, couldn't create ODK object." << std::endl;
        PrintErrorInformation(errCode, L"GetObject", pRuntime);
    }
}
```

See also

IAlarmLogging (Page 1350)

IAlarmLoggingCallback (Page 1353)

ILoggedAlarmResult (Page 1343)

19.10.5 Interfaces for connections (RT Uni)

19.10.5.1 IConnectionResult (RT Uni)

Description

The C++ interface "IConnectionResult" provides methods for access to the details of connections.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName(CFSTR *pName)
```

`pName`
[out]: Points to the name of the connection.

"GetConnectionState" method

Return status of the connection.

```
CFRESULT GetConnectionState(CFENUM *pConnectionState)
```

`pConnectionState`
[out]: Points to the enumeration, which can contain the following values:

- `Disabled` (0)
- `Connecting` (1)
- `Connected` (2)
- `Disconnecting` (3)
- `Disconnected` (4)
- `Reconnecting` (5)

"GetEstablishmentMode" method

Return mode in which the connection is established.

```
CFRESULT GetEstablishmentMode(CFENUM *pEstablishmentMode)
```

`pEstablishmentMode`
[out]: Points to the enumeration, which can contain the following values:

- `None` (0)
- `AutomaticActive` (1)
- `AutomaticPassive` (2)
- `OnDemandActive` (3)
- `OnDemandPassive` (4)

"GetTimeSynchronizationMode" method

Mode of time synchronization between HMI system and AS.

```
CFRESULT GetTimeSynchronizationMode(CFENUM
*pTimeSynchronizationMode)
```

`pTimeSynchronizationMode`
[out]: Points to the enumeration, which can contain the following values:

- `None` (0)
- `Slave` (1)
- `Master` (2)

"GetDisabledAtStartup" method

Indicates whether the connection is disabled at the start of Runtime.

```
CFRESULT GetDisabledAtStartup(CFBOOL *pDisabledAtStartup)
```

`pDisabledAtStartup`
[out]: Points to a Boolean value.

"GetEnabled" method

Indicates whether the connection is active.

```
CFRESULT GetEnabled(CFBOOL *pEnabled)
```

`pbEnabled`
[out]: Points to a Boolean value.

"GetConnectionType" method

Return protocol of a communication driver, e.g. "S7 Classic".

```
CFRESULT GetConnectionType(CFSTR *pConnectionType)
```

`pConnectionType`
[out]: Points to the name of the protocol.

"GetError" method

Return error code of the connection.

```
CFRESULT GetError(uint32_t *pError)
```

`pError`
[out]: Points to the error code.

Example

Output connection details:

Copy code

```
void DisplayConnectionInfo(IConnectionResultPtr pConnectionresult)
{
    if (nullptr != pConnectionresult)
    {
        CCfString strName;
        pConnectionresult->GetName(&strName);
        std::cout << "ConnectionName:" << strName.ToUTF8() << std::endl;

        CCfString strConnectiontype;
        pConnectionresult->GetConnectionType(&strConnectiontype);
        std::cout << "ConnectionType:" << strConnectiontype.ToUTF8() << std::endl;

        CFENUM enConnectionState;
        pConnectionresult->GetConnectionState(&enConnectionState);
        HmiConnectionState enumconnectionState =
static_cast<HmiConnectionState>(enConnectionState);

        ConnectionState(enumconnectionState);
        CFENUM enEstablishmentMode;
        pConnectionresult->GetEstablishmentMode(&enEstablishmentMode);
        HmiConnectionEstablishmentMode enumEstablishmentMode =
static_cast<HmiConnectionEstablishmentMode>(enEstablishmentMode);
        Establishmentmode(enumEstablishmentMode);

        CFENUM enTimeSynchronizationMode;
        pConnectionresult->GetTimeSynchronizationMode(&enTimeSynchronizationMode);
        HmiTimeSynchronizationMode enumTimeSynchronizationMode =
static_cast<HmiTimeSynchronizationMode>(enTimeSynchronizationMode);
        TimeSynchronizationmode(enumTimeSynchronizationMode);

        CFBOOL bDisableatStartup;
        pConnectionresult->GetDisabledAtStartup(&bDisableatStartup);
        std::cout << "DisableStartup:" << (int)bDisableatStartup << std::endl;

        CFBOOL bEnabled;
        pConnectionresult->GetEnabled(&bEnabled);
        std::cout << "Enabled:" << (int)bEnabled << std::endl;

        uint32_t nerror;
        pConnectionresult->GetError(&nerror);
        std::cout << "Error:" << nerror << std::endl;
    }
    std::cout << std::endl;
}
```

See also

IConnection (Page 1364)

IConnectionResultEnumerator (Page 1359)

19.10.5.2 IConnectionResultEnumerator (RT Uni)

Description

The "IConnectionResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of connection details of the Runtime system. The enumeration is used, for example, when reading out connections of a connection set.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IConnectionResult **ppItem)
```

`ppItem`
[out]: Points to the current "IConnectionResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *pCount)
```

`pCount`
[out]: Points to the value for the number of elements of the list.

See also

IConnectionResult (Page 1356)

IConnectionSet (Page 1370)

19.10.5.3 IConnectionStatusResult (RT Uni)

Description

The C++ interface "IConnectionStatusResult" provides methods for access to the status of connections.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName(CFSTR *pName)
```

`pName`
[out]: Points to the name of the connection.

"GetConnectionStatus" method

Return status of the connection.

```
CFRESULT GetConnectionStatus(CFENUM *pConnectionStatus)
```

`pConnectionStatus`
[out]: Points to the enumeration, which can contain the following values:

- `Disabled` (0)
- `Connecting` (1)
- `Connected` (2)
- `Disconnecting` (3)
- `Disconnected` (4)
- `Reconnecting` (5)

"GetError" method

Return error code of the connection.

```
CFRESULT GetError(uint32_t *pError)
```

`pError`
[out]: Points to the error code.

Example

Output status of a certain connection:

Copy code

```
void ConnectionSet_GetConnectionState(IRuntimePtr pRuntime)
{
    std::cout << "ConnectionSet_GetConnectionState  Start" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;
    CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionsetPtr(pUnk);
        CCfSmartString strName(L"HMI-Connection");
        CCfString strName1(L"HMI-ConnectionS7Plus");
        CCfArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);
        hr = pConnectionsetPtr->Add(connectionNames);

        if (nullptr != pConnectionsetPtr)
        {
            IConnectionStatusResultEnumeratorPtr pConnectionStatusResultEnum;
            IConnectionStatusResultPtr pConnectionStatusresult;
           errorCode = pConnectionsetPtr->GetConnectionState(&pConnectionStatusResultEnum);
            if (CF_SUCCEEDED(errorCode))
            {
                uint32_t nCount;
                pConnectionStatusResultEnum->Count(&nCount);
                for (int32_t i = 0; i < nCount; i++)
                {
                    pConnectionStatusResultEnum->MoveNext();
                    if (CF_SUCCEEDED(pConnectionStatusResultEnum-
>Current(&pConnectionStatusresult)))
                    {
                        if (nullptr != pConnectionStatusresult)
                        {
                            CCfString strName;
                            pConnectionStatusresult->GetName(&strName);
                           std::cout << "ConnectionName:" << strName.ToUTF8() << std::endl;
                            uint32_t nerror;
                            pConnectionStatusresult->GetError(&nerror);
                            std::cout << "Error:" << nerror << std::endl;
                            CFENUM enConnectionState;
                          pConnectionStatusresult->GetConnectionStatus(&enConnectionState);
                            HmiConnectionState enumconnectionState =
static_cast<HmiConnectionState>(enConnectionState);
                            ConnectionState(enumconnectionState);
                        }
                    }
                }
            }
            else
            {
                std::cout << L" ConnectionSet_GetConnectionState  failed." << "errCode = "
<< errorCode << std::endl;
            }
```

Copy code

```
        }
    }
    std::cout << std::endl;
}
```

See also

IConnection (Page 1364)

IConnectionStatusResultEnumerator (Page 1363)

19.10.5.4 IConnectionStatusResultEnumerator (RT Uni)

Description

The "IConnectionStatusResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of connection status of the Runtime system. The enumeration is used, for example, when reading out connections of a connection set.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

`CFRESULT Current(IConnectionStatusResult **ppItem)`

`ppItem`
[out]: Points to the current "IConnectionStatusResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

`CFRESULT MoveNext()`

"Reset" method

Reset the current position in the enumeration of a list.

`CFRESULT Reset()`

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

`CFRESULT Count(uint32_t *pCount)`

`pCount`
[out]: Points to the value for the number of elements of the list.

See also

IConnectionStatusResult (Page 1360)

IConnectionSet (Page 1370)

19.10.5.5 IConnection (RT Uni)

Description

The C++ interface "IConnection" provides properties and methods for access to a connection.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName(CFSTR *pName)
```

`pName`
[out]: Points to the name of the connection.

"SetName" method

Change name of the connection.

```
CFRESULT SetName(CFSTR name)
```

`name`
[in]: Name of the connection

"Read" method

Read connection details synchronously from the Runtime system.

```
CFRESULT Read(IConnectionResult **ppConnectionResult)
```

`ppConnectionResult`
[out]: Points to an object of type "IConnectionResult" that contains the connection details.

"GetConnectionState" method

Return connection status of a connection.

```
CFRESULT GetConnectionState(IConnectionStatusResult
**ppConnectionStatusResult)
```

`ppConnectionStatusResult`
[out]: Points to an object of type "IConnectionStatusResult" that contains the status of connections.

"SetConnectionMode" method

Change connection status of a connection.

```
CFRESULT SetConnectionMode(CFENUM connectionmode)
```

`connectionmode`
[in]: Enumeration which contains the mode of connections:

- `Disabled` (0)
- `Enabled` (1)

Examples

Change status of a connection:

Copy code

```
void Connection_SetConnectionState(IRuntimePtr pRuntime, ConnectionMode connectionMode)
{
    std::cout << "Connection_SetConnectionState Start :" << std::endl;
    ICfUnknownPtr pUnk;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"Connection"), &pUnk)))
    {
        IConnectionPtr pConnection(pUnk);

        CCfSmartString daName = L"HMI-Connection";
        pConnection->SetName(daName.AllocCFSTR());
        CFRESULT hr = pConnection->SetConnectionMode(int32_t(connectionMode));
        if(CF_SUCCEEDED(hr))
            std::cout << "Connection_SetConnectionState Succeeded" << std::endl;
        else
        {
            std::cout << "Connection_SetConnectionState failed" << std::endl;
        }
    }
    std::cout << std::endl;
}
```

See also

IConnectionResult (Page 1356)

IConnectionStatusResult (Page 1360)

IConnectionSet (Page 1370)

19.10.5.6 IConnectionReadNotification (RT Uni)

Description

The C++ interface "IConnectionReadNotification" defines a callback method for implementation of operations following read operations of connections.

Members

The following method is specified in the interface:

"OnReadComplete" method

Callback method is called following read operations of connections.

The "OnReadComplete" callback method is called when the `IConnectionSet.ReadAsync` method is used.

```
CFRESULT OnReadComplete(IConnectionResultEnumerator *pEnumerator,
uint32_t errorCode, int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "IConnectionResultEnumerator" object that contains the enumeration of the connection details.
- `errorCode`
  [out]: Error code for the asynchronous operation.
- `contextId`
  [out]: ContextID as additional identification feature of the connections.

Example

Read out connection asynchronously:

Copy code

```
void ConnectionSet_ReadAsync(IRuntimePtr pRuntime)
{
    std::cout << "ConnectionSet_ReadAsync Start :" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionset(pUnk);
        IConnectionResultPtr pConnectionResult;
        CCfString strName(L"HMI-Connection");
        CCfString strName1(L"HMI-ConnectionS7Plus");
        CCfArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);
        hr = pConnectionset->Add(connectionNames);
        if (nullptr != pConnectionset)
        {
           CCfRefPtr<CConnectionReadNotification>pRead = new CConnectionReadNotification();
            IConnectionReadNotification *pNotification = pRead;
            errorCode = pConnectionset->ReadAsync(pNotification);
            if (CF_SUCCEEDED(errorCode))
            {
                if (pRead->WaitForcompletion(dwMilliseconds) == CF_SUCCESS)
                {
                 std::cout << " OnRead Succeded." << "errCode = " << errorCode << std::endl;
                }
            }
            else
            {
              std::cout << L" ConnectionSet_ReadAsync failed." << "errCode = " << errorCode
<< std::endl;
            }
        }
    }
    std::cout << std::endl;
}
```

See also

IConnectionSet (Page 1370)

19.10.5.7 IConnectionStateChangeNotification (RT Uni)

Description

The C++ interface "IConnectionStateChangeNotification" defines a callback method for implementing asynchronous change monitoring of connections.

Members

The following method is specified in the interface:

"OnDataChanged" method

Callback method is called after changes of a monitored connection.

The "OnDataChanged" callback method is called when the `IConnectionSet.Subscribe` method is used.

```
CFRESULT OnDataChanged(IConnectionStatusResultEnumerator*
pEnumerator, uint32_t errorCode, int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "IConnectionStatusResultEnumerator" object that contains the enumeration of the connection status.
- `errorCode`
  [out]: Error code for the asynchronous operation.
- `contextId`
  [out]: ContextID as additional identification feature of the connections.

Example

Monitor connection status:

Copy code

```
void ConnnectionSet_Subscribe(IRuntimePtr pRuntime)
{
    std::cout << "ConnnectionSet_Subscribe Start" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);
    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionsetPtr(pUnk);
        CCfString strName(L"RUNTIME_1::Connection3");
        CCfString strName1(L"HMI-ConnectionS7Plus");
        CCfArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);

        hr = pConnectionsetPtr->Add(connectionNames);

        if (nullptr != pConnectionsetPtr)
        {
            CConnectionSubscriptionNotification *pSubscribe = new
CConnectionSubscriptionNotification();
            IConnectionStateChangeNotification *pNotification = pSubscribe;
            errorCode = pConnectionsetPtr->Subscribe(pNotification);
            if (CF_SUCCEEDED(errorCode))
            {
                if (pSubscribe->WaitForcompletion(dwMilliseconds) == CF_SUCCESS)
                {
                    //cancel subscription
                    pConnectionsetPtr->CancelSubscribe();
                }
            }
            else
            {
             std::cout << L" ConnnectionSet_Subscribe failed." << "errCode = " << errorCode
<< std::endl;
            }
        }
    }
    std::cout << std::endl;
}
```

See also

IConnectionSet (Page 1370)

19.10.5.8 IConnectionSet (RT Uni)

Description

The C++ interface "IConnectionSet" specifies properties and methods for optimized access to several connections of the Runtime system.

After initialization of the "IConnectionSet" object, you have read/write access to multiple connections in one call. Simultaneous access takes place with better performance and lower communication load than single access to multiple connections.

Members

The following methods are specified in the interface:

"SetContextId" method

Change ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

```
CFRESULT SetContextId(uint32_t id)
```

`id`
[in]: ContextID of the connection

"GetContextId" method

Return ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

```
CFRESULT GetContextId(uint32_t *pId)
```

`pId`
[out]: Points to the ContextID of the connection.

"Add" method

Add connections to a connection set.

```
CFRESULT Add(ICfArrayVariant *connectionNames)
```

`connectionNames`
[in]: Points to an array that contains the names of connections.

"Remove" method

Remove individual connection from connection set.

```
CFRESULT Remove(CFSTR connectionName)
```

`connectionName`
[in]: Name of the connection.

"Clear" method

Remove all connections from connection set.

```
CFRESULT Clear()
```

"GetCount" method

Return number of connections of a connection set list.

```
CFRESULT GetCount(int32_t *pCount)
```

`pCount`
[out]: Points to the number of connections in the connection set.

"Read" method

Read connection details of all connections of the connection set synchronously from the Runtime system.

```
CFRESULT Read(IConnectionResultEnumerator
**ppConnectionResultEnumerator)
```

`ppConnectionResultEnumerator`
[out]: Points to the enumeration of the details of the individual connections.

"ReadAsync" method

Read connection details of all connections of the connection set asynchronously from the Runtime system.

```
CFRESULT ReadAsync(IConnectionReadNotification *pReadReply)
```

`pReadReply`
[in]: Points to the "IConnectionReadNotification" callback interface for read operations and returns the "IConnectionResultEnumerator" object.

"GetConnectionState" method

Read connection status synchronously from the Runtime system.

```
CFRESULT GetConnectionState(IConnectionStatusResultEnumerator
**ppConnectionStatusResultEnumerator)
```

`ppConnectionStatusResultEnumerator`
[out]: Points to an object of type "IConnectionStatusResultEnumerator" that contains the enumeration of the connection status.

"Subscribe" method

Subscribe all connections of a connection set asynchronously for change monitoring.

```
CFRESULT Subscribe(IConnectionStateChangeNotification
*ppNotificationCB)
```

`ppNotificationCB`
[in]: Points to the "IConnectionStateChangeNotification" callback interface for monitoring and returns the "IConnectionStatusResultEnumerator" object following a change.

"CancelSubscribe" method

Cancel change monitoring of all connections of a connection set.

```
CFRESULT CancelSubscribe()
```

Examples

Monitor connection:

Copy code

```
void ConnnectionSet_Subscribe(IRuntimePtr pRuntime)
{
    std::cout << "ConnnectionSet_Subscribe Start" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionsetPtr(pUnk);
        CCfString strName(L"RUNTIME_1::Connection3");
        CCfString strName1(L"HMI-ConnectionS7Plus");
        CCfArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);

        hr = pConnectionsetPtr->Add(connectionNames);

        if (nullptr != pConnectionsetPtr)
        {
            CConnectionSubscriptionNotification *pSubscribe = new
CConnectionSubscriptionNotification();
            IConnectionStateChangeNotification *pNotification = pSubscribe;
            errorCode = pConnectionsetPtr->Subscribe(pNotification);
            if (CF_SUCCEEDED(errorCode))
            {
                if (pSubscribe->WaitForcompletion(dwMilliseconds) == CF_SUCCESS)
                {
                    //cancel subscription
                    pConnectionsetPtr->CancelSubscribe();
                }
            }
            else
            {
             std::cout << L" ConnnectionSet_Subscribe failed." << "errCode = " << errorCode
<< std::endl;
            }
        }
    }
    std::cout << std::endl;
}
```

Read out connection asynchronously:

Copy code

```
void ConnectionSet_ReadAsync(IRuntimePtr pRuntime)
{
    std::cout << "ConnectionSet_ReadAsync Start :" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionset(pUnk);
        IConnectionResultPtr pConnectionResult;
        CCfString strName(L"HMI-Connection");
        CCfString strName1(L"HMI-ConnectionS7Plus");
        CCfArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);

        hr = pConnectionset->Add(connectionNames);

        if (nullptr != pConnectionset)
        {
           CCfRefPtr<CConnectionReadNotification>pRead = new CConnectionReadNotification();
            IConnectionReadNotification *pNotification = pRead;
            errorCode = pConnectionset->ReadAsync(pNotification);
            if (CF_SUCCEEDED(errorCode))
            {
                if (pRead->WaitForcompletion(dwMilliseconds) == CF_SUCCESS)
                {
                 std::cout << " OnRead Succeded." << "errCode = " << errorCode << std::endl;
                }
            }
            else
            {
              std::cout << L" ConnectionSet_ReadAsync failed." << "errCode = " << errorCode
<< std::endl;
            }
        }
    }
    std::cout << std::endl;
}
```

Add/remove connection for connection set

Copy code

```
void Connection_AddRemove(IRuntimePtr pRuntime)
{
    std::cout << "Connection_AddRemove Start :" << std::endl;
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;

    CFRESULT errorCode = pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);

    if (pUnk != nullptr && CF_SUCCEEDED(errorCode))
    {
        IConnectionSetPtr pConnectionsetPtr(pUnk);
        CCfString strName(L"HMI-Connection");
        CCfSmartString strName1(L"HMI-ConnectionS7Plus");
        CCfArrayVariant vtArrayVaiarnt;
        vtArrayVaiarnt.Append(strName);
        vtArrayVaiarnt.Append(strName1);
        ICfArrayVariantPtr connectionNames;
        vtArrayVaiarnt.DetachEnumerator(&connectionNames);
        hr = pConnectionsetPtr->Add(connectionNames);
        int32_t count;
        pConnectionsetPtr->GetCount(&count);
        std::cout << "Count = " << count << std::endl;
        pConnectionsetPtr->Remove(strName1.AllocCFSTR());
        pConnectionsetPtr->GetCount(&count);
        std::cout << "Count = " << count << std::endl;
        pConnectionsetPtr->Clear();
        pConnectionsetPtr->GetCount(&count);
        std::cout << "Count = " << count << std::endl;
    }
    std::cout << std::endl;
}
```

See also

IConnectionReadNotification (Page 1365)

IConnectionStateChangeNotification (Page 1368)

IConnection (Page 1364)

IConnectionResultEnumerator (Page 1359)

IConnectionStatusResultEnumerator (Page 1363)

19.10.6 Interfaces of the Plant Model (RT Uni)

19.10.6.1 IPlantModel (RT Uni)

Description

The C++ interface "IPlantModel" specifies methods for access to object instances of the plant model of a Runtime system. The "IPlantModel" object represents the plant model of the graphical Runtime system.

The interface inherits from the "ICfDispatch" interface.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following methods are specified in the interface:

"GetPlantObject" method

Supplies an "IPlantObject" instance.

```
CFRESULT GetPlantObject(const CFSTR Node, IPlantObject**
ppPlantObject)
```

- `Node`
  [in]: Identifies an `IPlantObject` instance by its name or its path in the hierarchy.
- `ppPlantObject`
  [out]: Points to an "IPlantObject" instance.

"GetPlantObjectsByType" method

Supplies an enumeration with "IPlantObject" instances that have a specific type.

```
GetPlantObjectsByType(const CFSTR plantObjectTypeFilter, const CFSTR
ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- `plantObjectTypeFilter`
  [in]: Filter for the "IPlantObject" type on which the instances are based.
- `ViewFilter`
  [in]: Filter for the path in the hierarchy. Only instances from a specific node are returned.
- `ppPlantObjects`
  [out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

"GetObjectsByPropertyName" method

Supplies an enumeration with "IPlantObject" instances that have specific properties and originate in a specific plant node.

```
GetPlantObjectsByPropertyNames(const CFVARIANT PropertyNames, const
CFSTR ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- `PropertyNames`
  [in]: Property names
- `ViewFilter`
  [in]: Filter for a hierarchy path.
- `ppPlantObjects`
  [out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

"GetPlantObjectsByExpression" method

Supplies an enumeration with "IPlantObject" instances. The instances are filtered by type and property values.

```
GetPlantObjectsByExpression(const CFVARIANT PropertyNames, const
CFSTR plantObjectTypeFilter, const CFSTR expressionFilter, const
CFSTR ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- `PropertyNames`
  [in]: Property names
  If the list contains multiple values, all properties must be available at the object.
- `plantObjectTypeFilter`
  [in]: Filter for the object type on which the instances are based.
- `expressionFilter`
  [in]: An expression that is a filter for the property values.
- `ViewFilter`
  Filter for a hierarchy path.
- `ppCpmNodes`
  [out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

Example

Hierarchy path	Referenced object instance
`System2.TechnologicalHierarchy::P1/S1/L2/LeftPump`	References the "LeftPump" object instance in the "TechnologicalHierarchiy" of system2.
`.TechnologicalHierarchy::P1/S1/L2/LeftPump`	References the "LeftPump" object instance in the "TechnologicalHierarchiy" of the local system.
`U4711`	References the "U4711" object instance of the local system.
`System2::U4711`	References the "U4711" object instance of System2.

Copy code

```
void PlantModelGetPlantObjectsByType(IRuntimePtr pRuntime)
{
    if (nullptr != pRuntime)
    {
        ICfUnknownPtr pUnk;
        CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
        if (CF_SUCCEEDED(errCode) && nullptr != pUnk)
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectEnumeratorPtr pPlantObjectEnum;
            errCode = pPlantModel->GetPlantObjectsByType(CCfString(L"BLOWER").Get(), 
CCfString(L"").Get(),
            &pPlantObjectEnum);
            if (CF_SUCCEEDED(errCode) && nullptr != pPlantObjectEnum)
            {
               IPlantObjectPtr pItem; pPlantObjectEnum->MoveNext();
               while (CF_SUCCEEDED(pPlantObjectEnum->Current(&pItem)))
               {
                 displayNodeInfo(pItem);
                 pPlantObjectEnum->MoveNext();
               }
            }
            else
            {
             std::wcout << L"PlantModelGetPlantObjectsByType operation Failed" << L"ErrCode
= " << errCode <<
               endl;
              PrintErrorInformation(errCode, CCfSmartString(L"objectbytype"), pRuntime);
            }
        }
        else
        {
           std::cout << L"Error, couldn't create ODK object." << "errCode= " << errCode << 
std::endl;
        }
    }
}


void PlantModelGetPlantObjectsByExpressionByAllFilter(IRuntimePtr pRuntime)
{
       ICfUnknownPtr pUnk;
       CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
       if (pUnk != nullptr&& CF_SUCCEEDED(errCode))
       {
             IPlantModelPtr pPlantModel(pUnk);
             IPlantObjectEnumeratorPtr pPlantObjectEnum;
             CCfString str1 = L"NumberOfNodes";
             CCfString str2 = L"Quality";
             CCfSafeArray daSafeArray(CF_VT_STR, 1U, 2U);
             int32_t index = 0;
             daSafeArray.PutElement(&index, (void*)&str1);
             ++index;
             daSafeArray.PutElement(&index, (void*)&str2);
             ++index;
```

Copy code

```
             CCfVariant vtProperties;
             daSafeArray.Detach(&vtProperties);
             errCode = pPlantModel->GetPlantObjectsByExpression(vtProperties,
CCfString(L"BLOWER").Get(), CCfString(L"NumberOfNodes>=0"),
                          CCfString(L"RUNTIME_1.hierarchy").Get(), &pPlantObjectEnum);
             if (pPlantObjectEnum != nullptr && CF_SUCCEEDED(errCode))
             {
                    pPlantObjectEnum->MoveNext();
                    IPlantObject* pPlantObject;
                    while (CF_SUCCEEDED(pPlantObjectEnum->Current(&pPlantObject)))
                    {
                           displayNodeInfo(pPlantObject);
                           pPlantObjectEnum->MoveNext();
                    }
             }
             else
             {
                 std::cout << L"GetPlantObjectsByExpression operation failed." << "errCode
= " << errCode << std::endl;
             }
       }
       else
       {
             std::cout << L"Error, couldn't create ODK object." << "errCode = " << errCode
<< std::endl;
       }
}
```

19.10.6.2 IPlantObject (RT Uni)

Description

The C++ interface "IPlantObject" specifies methods for handling object instances of the plant model of a Runtime system.

An object instance in the plant model is based on an object type and its data structure. Each object instance receives its position within the plant hierarchy by assigning it to a hierarchy node.

The interface inherits from the "ICfDispatch" interface.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

`[SystemName].HierarchyName::[NodeID/.../]NodeID`

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following methods are specified in the interface:

"GetName" method

Supplies the name of the "IPlantObject" instance for unique identification.

```
CFRESULT GetName(CFSTR* pName)
```

- `pName`
  [out]: Name of the instance

"GetParent" method

Supplies the parent of the "IPlantObject" instance in the hierarchy.

```
CFRESULT GetParent(IPlantObject** ppParent)
```

- `ppParent`
  [out]: The parent as "IPlantObject" instance.

"GetChildren" method

Supplies an enumeration with the children of the "IPlantObject" instance in the hierarchy.

```
CFRESULT GetChildren(IPlantObjectEnumerator** ppChildren)
```

- `ppChildren`
  [out]: An enumeration of the type "IPlantObjectEnumerator" with child object instances.

"GetPlantViewPaths" method

Supplies a map with hierarchy names and hierarchy paths that the "IPlantObject" instance has in all hierarchies in which it is included.

```
CFRESULT GetPlantViewPaths(ICfMapStringToVariant **pViewPaths)
```

- `pViewPaths`
  [out]: A map with String/String pairs (hierarchy name to hierarchy path).

"GetCurrentPlantView" method

Supplies the path and names of the "IPlantObject" instance in the current hierarchy.

If the "IPlantObject" instance is only contained in one hierarchy, this path is returned.

```
CFRESULT GetCurrentPlantView)(CFSTR* pView)
```

- `pView`
  [out]: Hierarchy path and name of the "IPlantObject" instance.

"SetCurrentPlantView" method

"CurrentPlantView" is the basis for navigation with the "GetParent" or "GetChildren" methods. If the "IPlantObject" instance is contained in several hierarchies, the path must be set via "SetCurrentPlantView" before the "GetParent" or "GetChildren" method can be used.

```
CFRESULT SetCurrentPlantView)(CFSTR const& View)
```

- `View`
  [in]: The current hierarchy

"GetProperty" method

Supplies a property of the "IPlantObject" instance.

```
CFRESULT GetProperty(const CFSTR propertyName,
IPlantObjectProperty** ppPlantObjectProperty)
```

- `propertyName`
  [in]: Name of a property of the "IPlantObject" instance
- `ppPlantObjectProperty`
  [out]: Points to an "IPlantObjectProperty" instance.

"GetProperties" method

Supplies a two-dimensional list (name-object pairs) of the data structure of the "IPlantObject" instance. The list allows access to the instance properties.

```
CFRESULT GetProperties(const CFVARIANT propertyNames,
IPlantObjectPropertySet** ppPlantObjectPropertySet)
```

- Optional: `propertyNames`
  [in]: List with names of one or multiple properties of the "IPlantObject" instance.
  Without "propertyNames" parameter, all properties of the instance are referenced in the list.
- `ppPlantObjectPropertySet`
  [out]: Points to the list of the type "IPlantObjectPropertySet" that contains the names of one or multiple properties of the "IPlantObject" instance.

"GetActiveAlarms" method

Supplies all active alarms that the "IPlantObject" instance contains at the time it is called in the active hierarchy. Unlike with an AlarmSubscription, no status changes or new alarms are signaled that occur after the function call. Users can filter the alarms or specify a SystemName if they only want to receive the active alarms of a specific system.

```
CFRESULT GetActiveAlarms(uint32_t language, CFBOOL IncludeChildren,
CFSTR filter,IN IPlantObjectAlarmCallback* pCallback)
```

- `language`
  [in]: Language code of the language for all alarm texts and the filters. See chapter Locale IDs of the supported languages (Page 1085).
- `IncludeChildren`
  [in]: The active alarms of the child instances are returned as well.
- `filter`
  [in]: SQL-type string for filtering the alarm texts. The filter can contain operators. See also Syntax of the alarm filter (Page 1084).
- `pCallback`
  [in]: Callback pointer.

"CreateAlarmSubscription" method

Supplies a "PlantObjectAlarmSubscription" that can be used to start and stop an alarm subscription.

```
CFRESULT CreateAlarmSubscription)(IPlantObjectAlarmSubscription**
ppPlantObjectAlarmSubscription)
```

- `ppPlantObjectAlarmSubscription`
  [out] Points to a "PlantObjectAlarmSubscription" instance.

Example

Copy code

```
void PlantObjectGetProperties(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
        {
            if (nullptr != pUnk && CF_SUCCEEDED(errCode))
            {
                IPlantModelPtr pPlantModel(pUnk);
                IPlantObjectPtr pPlantObject;
                CCfString strNode = L".hierarchy::PlantView\\Unit1";

                //gets node for specified Node path
                errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
                if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
                {
                    //empty property - so all node members should be read
                    CCfVariant vtProperties;
                    IPlantObjectPropertySetPtr pPlantObjectPropertySet;
                    // get the PlantObject properties by property names
                    errCode = pPlantObject->GetProperties(vtProperties,
&pPlantObjectPropertySet);
                    if (nullptr != pPlantObjectPropertySet && CF_SUCCEEDED(errCode))
                    {
                        uint32_t nCount = 0;
                        pPlantObjectPropertySet->GetCount(&nCount);
                        std::cout << "Count :" << nCount << std::endl;
                    }
                    else
                    {
                       std::cout << L" GetProperties operation failed." << "errCode = " <<
errCode << std::endl;
                        PrintErrorInformation(errCode, L"Getproperties", pRuntime);
                    }
                }
                else
                {
                    std::cout << L"GetPlantObject operation failed." << "errCode = " <<
errCode << std::endl;
                }
            }
            else
            {
                    std::cout << L"Error, couldn't create ODK object." << "errCode = " <<
errCode << std::endl;
            }
        }
}
```

19.10.6.3 IPlantObjectProperty (RT Uni)

Description

The C++ interface "IPlantObjectProperty" specifies the handling of properties of object instances of the plant model of a Runtime system. The properties represent the data structure of an object instance.

The object instance communicates with the automation system through the properties of the data structure. The values of the properties are obtained from linked process tags or internal tags.

You reference an object using the `IPlantObject.GetProperty` or `IPlantObject.GetProperties` methods.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"GetName" method

Supplies the name of the property.

```
CFRESULT GetName(CFSTR *pName)
```

- `pName`
  [out]: Points to the name of the property.

"Read" method

Reads the value of the "IPlantObjectProperty" instance synchronously and returns it as an "IPlantObjectPropertyValue" object. The value, the quality code and the time stamp of the property are determined when the property is read.

```
CFRESULT Read(IPlantObjectPropertyValue**
ppPlantObjectPropertyValue)
```

- `ppPlantObjectPropertyValue`
  [out]: Values of the property as "IPlantObjectPropertyValue" instance

"Write" method

Writes the value synchronously to the "IPlantObjectProperty" instance.

```
CFRESULT Write)(const CFVARIANT value)
```

- `value`
  [in]: New process value of the property

Example

Copy code

```
void PlantObjectReadproperty(IRuntimePtr pRuntime)
{
    std::cout << "PlantObjectReadproperty Start" << std::endl;
    ICfUnknownPtr pUnk;CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"),
&pUnk);
    {
        if (nullptr != pUnk && CF_SUCCEEDED(errCode))
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectPtr pPlantObject;CCfString strNode = L".hierarchy::PlantView\
\Unit1";

            //gets node for specified Node path
            errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
            if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
            {

                CCfString strName(L"NumberOfNodes");
                IPlantObjectPropertyPtr PlantObjectProperty;
                //get the PlantObject property by name
                pPlantObject->GetProperty(strName.Get(), &PlantObjectProperty);

                IPlantObjectPropertyValuePtr pPlantObjectPropertyValue;
                // Read PlantObject Property
                PlantObjectProperty->Read(&pPlantObjectPropertyValue);
                if (nullptr != pPlantObjectPropertyValue)
                {
                    CCfString strName;
                    pPlantObjectPropertyValue->GetPlantObjectPropertyName(&strName);
                   std::cout << "PlantModelPropertyName:" << strName.ToUTF8() << std::endl;
                    int64_t qc;
                    pPlantObjectPropertyValue->GetQuality(&qc);
                    std::cout << "Quality:" << qc << std::endl;
                    int64_t ec;
                    pPlantObjectPropertyValue->GetError(&ec);
                    std::cout << "Error:" << ec << std::endl;
                    CCfDateTime64 dt;
                    pPlantObjectPropertyValue->GetTimeStamp(&dt);
                  std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() << std::endl;
                    CCfVariant vtVal;
                    pPlantObjectPropertyValue->GetValue(&vtVal);
                    std::cout << "Value:" << vtVal.uint64 << std::endl;
                }
                else
                {
                    std::cout << L" GetProperties operation failed." << "errCode = " <<
errCode << std::endl;
                    PrintErrorInformation(errCode, L"GetProperties", pRuntime);
                }
            }
            else
            {
               std::cout << L"GetPlantObject operation failed." << "errCode = " << errCode
<< std::endl;
```

Copy code

```
                PrintErrorInformation(errCode, L"GetPlantObject", pRuntime);
            }
        }
        else
        {
            std::cout << L"Error, couldn't create ODK object." << "errCode = " << errCode
<< std::endl;
        }
    }
}
```

19.10.6.4 IPlantObjectPropertyValue (RT Uni)

Description

The C++ interface "IPlantObjectPropertyValue" specifies the handling of process tag properties of the Runtime system.

Members

The following methods are specified in the interface:

"GetPlantObjectPropertyName" method

Supplies the name of the tag.

`CFRESULT GetPlantObjectPropertyName(CFSTR* pName)`

- `pName`
  [out]: Points to the name.

"GetValue" method

Supplies the tag value.

`CFRESULT GetValue(CFVARIANT* pValue)`

- `pValue`
  [out]: Points to the process value.

"GetQuality" method

Supplies the quality code of the tag.

`CFRESULT GetQuality(int64_t* pQualityCode)`

- `pQualityCode`
  [out]: Points to the quality code.

"GetTimeStamp" method

Supplies the time stamp of the last modification to the tag.

```
CFRESULT GetTimeStamp(CFDATETIME64* pTimeStamp)
```

- `pTimeStamp`
  [out]: Points to the time stamp.

"GetError" method

Supplies the error code of the tag.

```
CFRESULT GetError(int64_t* pErrorCode)
```

- `pErrorCode`
  [out]: Points to the error code.

Example

For example, see IPlantObjectProperty (Page 1383).

19.10.6.5 IPlantModelPropertySubscriptionNotification (RT Uni)

Description

The C++ interface "IPlantModelPropertySubscriptionNotification" defines methods for implementing asynchronous change monitoring of object instance properties. The methods are used by the C++ interface "ICpmNodePropertySet".

All the methods return CF_SUCCESS following successful execution.

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"OnDataChanged" method

Callback method is called when a monitored object instance property is changed.

```
CFRESULT OnDataChanged(IPlantObjectPropertyValueEnumerator*
pEnumerator)
```

- `pEnumerator`:
  [in]: Points to the "IPlantObjectPropertyValueEnumerator" object that provides access to an enumeration of "IPlantObjectPropertyValue" instances.

Example

Register a PropertySet for monitoring:

Copy code

```
void PlantObjectSubscribePropertySet(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
    {
        if (nullptr != pUnk && CF_SUCCEEDED(errCode))
        {
            IPlantModelPtr pPlantModel(pUnk);
            IPlantObjectPtr pPlantObject;
            CCfString strNode = L".hierarchy::PlantView\\Unit1\\Filler1";

            //gets node for specified Node path
            errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
            if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
            {
                //empty property - so all node members should be read
                CCfVariant vtProperties;
                IPlantObjectPropertySetPtr pPlantObjectPropertySet;
                // get the PlantObject properties by property names
                errCode = pPlantObject->GetProperties(vtProperties,
&pPlantObjectPropertySet);
                if (nullptr != pPlantObjectPropertySet && CF_SUCCEEDED(errCode))
                {
                    CPlantModelPropertySubscriptionNotification* pSubscribe = new
CPlantModelPropertySubscriptionNotification();
                   IPlantModelPropertySubscriptionNotification* pNotification = pSubscribe;
                    //Subscribe for all PlantObject properties errCode =
pPlantObjectPropertySet->Subscribe(pNotification);
                }
                else
                {
                    std::cout << L" GetProperties operation failed." << "errCode = " <<
errCode << std::endl;
                }
            }
            else
            {
               std::cout << L"GetPlantObject operation failed." << "errCode = " << errCode
<< std::endl;
            }
        }
    }
}
CFRESULT CFCALLTYPE
CPlantModelPropertySubscriptionNotification::OnDataChanged(IPlantObjectPropertyValueEnumer
ator* pEnumerator)
{
    if (nullptr != pEnumerator)
    {
        uint32_t nCount = 0;
        pEnumerator->Count(&nCount);
        for (int i = 0;i < (int32_t)nCount; i++)
        {
            pEnumerator->MoveNext();
            IPlantObjectPropertyValue* pPlantModelPropertyValue;
```

Copy code

```
          if (CF_SUCCEEDED(pEnumerator->Current(&pPlantModelPropertyValue)) && nullptr !=
pPlantModelPropertyValue)
           {
               CCfString strName;
               pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
               std::cout << "Name:" << strName.ToUTF8() << std::endl;
               int64_t qc;
               pPlantModelPropertyValue->GetQuality(&qc);
               std::cout << "Quality:" << qc << std::endl;
               int64_t ec;
               pPlantModelPropertyValue->GetError(&ec);
               std::cout << "Error:" << ec << std::endl;
               CCfDateTime64 dt;
               pPlantModelPropertyValue->GetTimeStamp(&dt);
              std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() << std::endl;
               CCfVariant vtVal;
               pPlantModelPropertyValue->GetValue(&vtVal);
               std::cout << "Value:" << vtVal.uint64 << std::endl;
           }
       }
    }
    return CF_SUCCESS;
}
```

19.10.6.6 IPlantObjectPropertyValueEnumerator (RT Uni)

Description

The C++ interface "IPlantObjectPropertyValueEnumerator" specifies methods for handling the enumeration of "IPlantObjectPropertyValue" instances.

All the methods return `CF_SUCCESS` following successful execution.

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"Current" method

Supplies the current element of the enumeration.

`Current(IPlantObjectPropertyValue** ppItem)`

- `ppItem`
  [out]:

"MoveNext" method

Move to the next element of the enumeration.

`CFRESULT MoveNext()`

"Reset" method

Resets the current position in the enumeration to the first element.

```
CFRESULT Reset()
```

"MoveNext" then supplies the first element of the enumeration.

"Count" method

Output the size of the enumeration or the number of elements in an enumeration.

```
CFRESULT Count(uint32_t *pValue)
```

- `value`
  [out]: Points to the value for the number of elements in the enumeration.

Example

For example, see IPlantModelPropertySubscriptionNotification (Page 1386).

19.10.6.7 IPlantObjectPropertySet (RT Uni)

Description

The C++ interface "IPlantObjectPropertySet" specifies methods for optimized access to several "IPlantObjectProperty" instances of an "IPlantObject" instance of the Runtime system.

After initialization of the "IPlantObjectPropertySet" object, you have read/write access to multiple "IPlantObjectProperty" instances in one call. Simultaneous access has better performance and a lower communication load than single access to multiple properties.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"GetContextID" method

Supplies an ID as additional identification feature of a property. "ContextId" can, for example, be used to recognize properties having the same name but from different systems.

Default value -1: "ContextId" is not used.

```
CFRESULT GetContextId)(int32_t* pContextId)
```

- `pContextId`
  [out]: "ContextId" of the property

"SetContextID" method

Sets an ID as additional identification feature of a property. "ContextId" can, for example, be used to recognize properties having the same name but from different systems.

Default value -1: "ContextId" is not used.

```
CFRESULT SetContextId(int32_t contextId)
```

- `contextId`
  [in]: "ContextId" of the property

"Read" method

Supplies all values of the "IPlantObjectProperty" instances contained in the "IPlantObjectPropertySet" instance. The values are read synchronously.

```
CFRESULT Read(IPlantObjectPropertyValueEnumerator**
ppPlantObjectPropertyValueEnumerator)
```

- `ppPlantObjectPropertyValueEnumerator`
  [out]: Points to the enumeration of the tag values as an "IPlantObjectPropertyValueEnumerator" object.

"ReadAsync" method

Reads the values of all "IPlantObjectProperty" instances of the "IPlantObjectPropertySet" instance asynchronously.

```
CFRESULT ReadAsync(IPlantObjectPropertySetReadReply* pReadReply)
```

- `pReadReply`
  [in]: Points to the "IPlantObjectPropertySetReadReply" object that implements the callback interface for read operations.

"Write" method

Writes the values of the "IPlantProperty" instances of the "PlantObjectPropertySet" instance synchronously to the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator**
ppEnumerator)
```

- `pWriteReply`
  [out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations of the "IPlantObjectProperty" instances.

"WriteAsync" method

Writes the values of all "IPlantObjectProperty" instances of the "PlantObjectPropertySet" instance asynchronously to the Runtime system.

```
CFRESULT WriteAsync(IPlantObjectPropertySetWriteReply *pWriteReply)
```

- `pWriteReply`
  [in]: Points to the "IPlantObjectPropertySetWriteReply" object that implements the callback interface for write operations.

"GetCount" method

Supplies the number of properties of the "IPlantObjectPropertySet" instance.

```
CFRESULT GetCount(uint32_t *pCount)
```

- `pCount`
  [out]: Points to the number of properties.

"Subscribe" method

Subscribes all properties of the "IPlantObjectPropertySet" instance asynchronously for change monitoring.

```
CFRESULT Subscribe(IPlantModelPropertySubscriptionNotification*
pPlantModelSubscriptionCallback)
```

- `pPlantModelSubscriptionCallback`
  [in]: Points to the "IPlantObjectPropertySubscriptionNotification" object that implements the callback interface of the change monitoring.

"CancelSubscribe" method

Cancels change monitoring for all properties of the "IPlantObjectPropertySet" instance.

```
CFRESULT CancelSubscribe()
```

"Add" method

Adds an "IPlantObjectProperty" instance with property value or more "IPlantObjectProperty" instances to the "IPlantObjectPropertySet" instance.

```
CFRESULT Add(ICfArrayVariantPtr propertyNames)
```

- `propertyNames`
  [in]: Array with names of several "IPlantObjectProperty" instances

or

```
CFRESULT Add(const CFSTR propertyName, const CFVARIANT value)
```

- `propertyName`
  [in]: Name of the "IPlantObjectProperty" instance
- `value`
  [in]: New process value of the "IPlantObjectProperty" instance

"Remove" method

Removes a property from the "IPlantObjectPropertySet" instance.

```
CFRESULT Remove(const CFSTR propertyName)
```

- `propertyName`
  [in]: Name of the property that is being removed.

"Clear" method

Removes all properties from the "IPlantObjectPropertySet" instance.

```
CFRESULT Clear()
```

New example

Copy code

```
void PlantObjectWriteAsyncPropertySet(IRuntimePtr pRuntime)
{
    CFRESULT hr = CF_ERROR;
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
    {
        if (nullptr != pUnk && CF_SUCCEEDED(errCode))
        {
                IPlantModelPtr pPlantModel(pUnk);
                IPlantObjectPtr pPlantObject;
                CCfString strNode = L".hierarchy::PlantView\\Unit1";

                //gets node for specified Node path
                errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
                if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
                {
                    //empty property - so all node members should be read
                    CCfVariant vtProperties;
                    IPlantObjectPropertySetPtr pPlantObjectPropertySet;
                    // get the PlantObject properties by property names
                    errCode = pPlantObject->GetProperties(vtProperties,
&pPlantObjectPropertySet);
                    if (nullptr != pPlantObjectPropertySet && CF_SUCCEEDED(errCode))
                    {
                        CCfString strName(L"NumberOfNodes");
                        CCfVariant vtValue(1000);
                        hr = pPlantObjectPropertySet->Add(strName, vtValue);
                        if (CF_SUCCEEDED(hr))
                        {
                            CPlantObjectPropertySetWriteReply* pReply = new
CPlantObjectPropertySetWriteReply();
                            IPlantObjectPropertySetWriteReplyPtr pWritReply = pReply;

                            //Write PlantObject properties values asynchronously
                            pPlantObjectPropertySet->WriteAsync(pWritReply);
                        }
                    }
                    else
                    {
                       std::cout << L" GetProperties operation failed." << "errCode = " <<
errCode << std::endl;
                    }
                }
                else
                {
                       std::cout << L"GetPlantObject operation failed." << "errCode = " <<
errCode << std::endl;
                }
        }
        else
        {
              std::cout << L"Error, couldn't create ODK object." << "errCode = " << errCode
<< std::endl;
        }
```

Copy code
```
        }
}
```

19.10.6.8 IPlantObjectPropertySetReadReply (RT Uni)

Description

The C++ interface "IPlantObjectPropertySetReadReply" defines the "OnReadComplete" method as callback method of a "ReadAsync" call. The method is used by the C++ interface "IPlantObjectPropertySet".

Members

The following methods are specified in the interface:

"OnReadComplete" method

The "OnReadComplete" callback method is called when the "ReadAsync" method is used.

```
CFRESULT OnReadComplete(CFVARIANT systemError,
IPlantObjectPropertyValueEnumerator*
pPlantObjectPropertyValueEnumerator)
```

- `systemError`
  [in]: Supplies an error code when a global error has occurred.
- `ppPlantObjectPropertyValueEnumerator`
  [in]: Points to an "IPlantObjectPropertyValueEnumerator" object that contains the enumeration of the process values of object instance properties.

Example

Copy code

```
void PlantObjectReadAsycPropertySet(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
    {
          if (nullptr != pUnk && CF_SUCCEEDED(errCode))
          {
                IPlantModelPtr pPlantModel(pUnk);
                IPlantObjectPtr pPlantObject;
                CCfString strNode = L".hierarchy::PlantView\\Unit1";

                //gets node for specified Node path
                errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
                if (nullptr != pPlantObject && CF_SUCCEEDED(errCode))
                {
                        //empty property - so all node members should be read
                        CCfVariant vtProperties;
                        IPlantObjectPropertySetPtr pPlantObjectPropertySet;

                        // get the PlantObject properties by property names
                        errCode = pPlantObject->GetProperties(vtProperties,
&pPlantObjectPropertySet);
                        if (nullptr != pPlantObjectPropertySet && CF_SUCCEEDED(errCode))
                        {
                            CPlantObjectPropertySetReadReply* pReply = new
CPlantObjectPropertySetReadReply();
                            IPlantObjectPropertySetReadReplyPtr pReadReply = pReply;

                            // Read PlantObject properties values asynchronously
                            pPlantObjectPropertySet->ReadAsync(pReadReply);
                        }
                        else
                        {
                            std::cout << L" GetProperties operation failed." << "errCode =
" << errCode << std::endl;
                        }
                }
                else
                {
                       std::cout << L"GetPlantObject operation failed." << "errCode = " <<
errCode << std::endl;
                }
         }
         else
         {
              std::cout << L"Error, couldn't create ODK object." << "errCode = " << errCode
<< std::endl;
         }
    }
}
CFRESULT CFCALLTYPE CPlantObjectPropertySetReadReply::OnReadComplete(IN CFVARIANT
SystemError, IN IPlantObjectPropertyValueEnumerator* pPlantObjectPropertySet)
{
    if (nullptr != pPlantObjectPropertySet)
```

Copy code

```
    {
          uint32_t nCount = 0;pPlantObjectPropertySet->Count(&nCount);
          for (int i = 0;i < (int32_t)nCount; i++)
          {
                pPlantObjectPropertySet->MoveNext();
                IPlantObjectPropertyValue* pPlantModelPropertyValue;
                if (CF_SUCCEEDED(pPlantObjectPropertySet-
>Current(&pPlantModelPropertyValue)) && nullptr !=              pPlantModelPropertyValue)
                {
                        CCfString strName;
                        pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
                        std::cout << "Name:" << strName.ToUTF8() << std::endl;
                        int64_t qc;
                        pPlantModelPropertyValue->GetQuality(&qc);
                        std::cout << "Quality:" << qc << std::endl;
                        int64_t ec;
                        pPlantModelPropertyValue->GetError(&ec);
                        std::cout << "Error:" << ec << std::endl;
                        CCfDateTime64 dt; pPlantModelPropertyValue->GetTimeStamp(&dt);
                        std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() <<
std::endl;
                        CCfVariant vtVal;
                        pPlantModelPropertyValue->GetValue(&vtVal);
                        std::cout << "Value:" << vtVal.uint64 << std::endl;
                }
          }
    }
    return CF_SUCCESS;
}
```

19.10.6.9 IPlantObjectPropertySetWriteReply (RT Uni)

Description

The C++ interface "IPlantObjectPropertySetWriteReply" defines the "OnWriteComplete" method as callback method of a "WritedAsync" call. The method is used by the C++ interface "IPlantObjectPropertySet".

Members

The following methods are specified in the interface:

"OnWriteComplete" method

The "OnWriteComplete" callback method is called when the "WriteAsync" method is used.

```
CFRESULT OnWriteComplete(CFVARIANT systemError,
IPlantObjectPropertyValueEnumerator*
pPlantObjectPropertyValueEnumerator)P
```

- `systemError`
  [in]: Supplies an error code when a global error has occurred.
- `ppPlantObjectPropertyValueEnumerator`
  [in]: Points to an "IPlantObjectPropertyValueEnumerator" object that contains the enumeration of the process values of "IPlantObjectProperty" instances.

Example

Copy code

```
CFRESULT CFCALLTYPE CPlantObjectPropertySetWriteReply::OnWriteComplete(IN CFVARIANT
SystemError, IN IPlantObjectPropertyValueEnumerator* pPlantObjectPropertySet)
{
    if (nullptr != pPlantObjectPropertySet)
    {
          uint32_t nCount = 0;
          pPlantObjectPropertySet->Count(&nCount);
          for (int i = 0;i < (int32_t)nCount; i++)
          {
                pPlantObjectPropertySet->MoveNext();
                IPlantObjectPropertyValue* pPlantModelPropertyValue;
                if (CF_SUCCEEDED(pPlantObjectPropertySet-
>Current(&pPlantModelPropertyValue)) && nullptr !=
pPlantModelPropertyValue)
                {
                        CCfString strName;
                        pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
                        std::cout << "Name:" << strName.ToUTF8() << std::endl;
                        int64_t qc;
                        pPlantModelPropertyValue->GetQuality(&qc);
                        std::cout << "Quality:" << qc << std::endl;
                        int64_t ec;
                        pPlantModelPropertyValue->GetError(&ec);
                        std::cout << "Error:" << ec << std::endl;
                        CCfDateTime64 dt;
                        pPlantModelPropertyValue->GetTimeStamp(&dt);
                        std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() <<
std::endl;
                        CCfVariant vtVal;
                        pPlantModelPropertyValue->GetValue(&vtVal);
                        std::cout << "Value:" << vtVal.uint64 << std::endl;
                }
          }
    }
    return CF_SUCCESS;
}
```

19.10.6.10 IPlantObjectEnumerator (RT Uni)

Description

The "IPlantObjectEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of object instances of the plant model of a Runtime system.

All the methods return CF_SUCCESS following successful execution.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"Current" method

Supplies the current element of the enumeration.

```
CFRESULT Current(IPlantObject **ppItem)
```

- `ppItem`
  [out]: Points to the current "IPlantObject" object as an element of the list.

"MoveNext" method

Move to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Reset" method

Resets the current position in the enumeration to the first element.

```
CFRESULT Reset()
```

"MoveNext" then supplies the first element of the enumeration.

"Count" method

Output the size of the enumeration or the number of elements in an enumeration.

```
CFRESULT Count(uint32_t *pValue)
```

- `value`
  [out]: Points to the value for the number of elements in the enumeration.

Example

Copy code

```
void PlantObjectCurrentPlantViewPath(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"),
&pUnk);
    if (pUnk != nullptr&& CF_SUCCEEDED(errCode))
    {
          IPlantModelPtr pPlantModel(pUnk);
          IPlantObjectEnumeratorPtr pPlantObjectEnum;
          CCfString str1 = L"NumberOfNodes";
          CCfString str2 = L"Quality";
          CCfString str3 = L"Quantity";
          CCfString str4 = L"DateActivation";
          CCfSafeArray daSafeArray(CF_VT_STR, 1U, 4U);
          int32_t index = 0;
          daSafeArray.PutElement(&index, (void*)&str1);
          ++index;
          daSafeArray.PutElement(&index, (void*)&str2);
          ++index;
          daSafeArray.PutElement(&index, (void*)&str3);
          ++index;
          daSafeArray.PutElement(&index, (void*)&str4);
          CCfVariant vtProperties;
          daSafeArray.Detach(&vtProperties);

          // gets PlantObjects for specified filter
          errCode = pPlantModel->GetPlantObjectsByExpression(vtProperties,
CCfString                    (L"BLOWER").Get(),
CCfString(L"NumberOfNodes>=0"), CCfString
(L"RUNTIME_1.hierarchy::PlantView\\Unit1\\Blower1").Get
(),&pPlantObjectEnum);
          if (pPlantObjectEnum != nullptr && CF_SUCCEEDED(errCode))
          {
                pPlantObjectEnum->MoveNext();
                IPlantObject* pPlantObject;
                while (CF_SUCCEEDED(pPlantObjectEnum-
>Current(&pPlantObject)))
                {
                        displayNodeInfo(pPlantObject);
                        pPlantObjectEnum->MoveNext();
                }
          }
          else
          {
              std::cout << L"GetPlantObjectsByExpression operation failed."
<< "errCode =                                 " << errCode << std::endl;
          }
    }
    else
    {
          std::cout << L"Error, couldn't create ODK object." << "errCode =
" << errCode <<                    std::endl;
    }
```

Copy code

```
}
```

19.10.6.11 IPlantObjectAlarmSubscription (RT Uni)

Description

The C++ interface "IPlantObjectAlarmSubscription" specifies methods for starting and stopping an "AlarmSubscription".

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"Start" method

Subscribe systems for monitoring of changes of active alarms.

```
CFRESULT Start(IPlantObjectAlarmSubscriptionCallback* callbackPtr)
```

- `callbackPtr`
  [in]: Points to the "IPlantObjectAlarmSubscriptionCallback" object that implements the callback interface of the change monitoring.

"Stop" method

Unsubscribe monitoring of active alarms.

```
CFRESULT Stop(void)
```

"GetFilter" method

Supplies the string by which the result set is filtered.

```
CFRESULT GetFilter(CFSTR* filter)
```

- `filter`
  [out]: SQL-type string for filtering the result set of active alarms.

"SetFilter" method

Sets the string for filtering the result set of active alarms.

```
CFRESULT SetFilter(IN CFSTR filter)
```

- `filter`
  [in]: SQL-type string for filtering the result set of active alarms.
  All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 1084).

"GetLanguage" method

Supplies the country identifier of the language of the monitored alarms.

```
CFRESULT GetLanguage(uint32_t* language)
```

- `language`
  [out]: Code of the country identification. See also sectionLocale IDs of the supported languages (Page 1085).

"SetLanguage" method

Sets the country identifier of the language of the monitored alarms.

```
CFRESULT SetLanguage)(uint32_t language)
```

- `language`
  [in]: Code of the country identification See also sectionLocale IDs of the supported languages (Page 1085).

"GetIncludeChildren" method

Supplies the setting for the child instances.

```
CFRESULT GetIncludeChildren(CFBOOL* bIsIncludeChildren)
```

- `bIsIncludeChildren`
  [out]: Reads out whether the child instances are part of monitoring.

"SetIncludeChildren" method

Determines the setting for the child instances.

```
CFRESULT SetIncludeChildren(CFBOOL bIsIncludeChildren)
```

- `bIsIncludeChildren`
  [in]: Controls whether the child instances are part of monitoring.

Example

Copy code

```
void PlantObjectSubscription(IRuntimePtr pRuntime)
{
    if (nullptr != pRuntime)
    {
          ICfUnknownPtr pUnk;
          CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel").Get(), &pUnk);
          if (CF_SUCCEEDED(errCode) && nullptr != pUnk)
          {
                   IPlantModelPtr pPlantModel(pUnk);
                   CAlarmValue* pAlarmValue = new CAlarmValue();
                   pAlarmValue->AddRef();
                   IPlantObjectAlarmCallback *pCB = pAlarmValue;
                   CCfString strNode = L".hierarchy::RootNodeName\\Node1";
                   IPlantObjectPtr pPlantObject;
                   errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
                   if (pPlantObject != nullptr && CF_SUCCEEDED(errCode))
                   {
                        CCfSmartString daFilter; //= L"AlarmClassName = 'Alarm'";
                        uint32_t daLanguage = 1033;
                        //IPlantObjectAlarmCallbackPtr pAlarmCallback;
                        IPlantObjectAlarmSubscriptionPtr pPlantObjectAlarmSubscription;
                        errCode = pPlantObject-
>CreateAlarmSubscription(&pPlantObjectAlarmSubscription);
                    if (errCode == CF_SUCCESS && nullptr != pPlantObjectAlarmSubscription)
                        {
                          pPlantObjectAlarmSubscription->SetFilter(daFilter.AllocCFSTR());
                            pPlantObjectAlarmSubscription->SetLanguage(1033);
                            pPlantObjectAlarmSubscription->SetFilter(false);
                            pPlantObjectAlarmSubscription->SetIncludeChildren(false);
                            CCfRefPtr<CPlantObjectAlarmSubscriptionCallback> pCallback =
                                                        new
CPlantObjectAlarmSubscriptionCallback();
                            pPlantObjectAlarmSubscription->Start(pCallback);
                        }
                   }
          }
    }
}
```

19.10.6.12 IPlantObjectAlarmCallback (RT Uni)

Description

The C++ interface "IPlantObjectAlarmCallback" defines the callback method "OnAlarm".

Member

The following methods are defined in the interface:

"OnAlarm" method

Specifies the signature of the event handling method for the "OnAlarm" event of an "IPlantObject" instance.

```
OnAlarm(uint32_t systemError,
        CFSTR systemName,

Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator*
pItems,
        CFBOOL completed)
```

- `systemError`
  Supplies an error code when a global error has occurred. When the error code is set, `pItems` is irrelevant.
- `systemName`
  Name of the Runtime system that is subscribed for alarm monitoring by the user.
- `pItems`
  Supplies a pointer to "IAlarmResultEnumerator" that can be used to enumerate the active alarms.
- `completed`
  Status of the asynchronous transfer:
  - `True`: All alarms are read out.
  - `False`: Not all alarms are yet read out.

Example

Copy code

```
CFRESULT CFCALLTYPE CPlantModelAlarmValue::OnAlarm( uint32_t systemError, CFSTR
systemName, Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator* pItems, CFBOOL
completed)
{
    CFRESULT hr = CF_FALSE;
    if (!completed)
    {
          uint32_t nsize;
          pItems->Count(&nsize);
          if (nsize > 0 && CF_SUCCEEDED(systemError))
          {
                while (CF_SUCCEEDED(pItems->MoveNext()))
                {
                        IAlarmResultPtr ppValues;
                        if (CF_SUCCEEDED(pItems->Current(&ppValues)))
                        {
                            CCfString strId;
                            ppValues->GetSourceID(&strId);
                         std::cout << "String ID = " << strId.ToUTF8().c_str() << std::endl;
                            CCfString strName;
                            ppValues->GetName(&strName);
                          std::cout << "Name = " << strName.ToUTF8().c_str() << std::endl;

                            CCfString strClassName;
                            ppValues->GetAlarmClassName(&strClassName);
                        std::cout << "Alarm Class Name = " << strClassName.ToUTF8().c_str()
<< std::endl;
                        }
                }
          }
    }
    else
    {
          this->SetEvent();
    }
    hr = CF_SUCCESS;
    return hr;
}
```

19.10.6.13 IPlantObjectAlarmSubscriptionCallback (RT Uni)

Description

The C++ interface "IPlantObjectAlarmCallbackSubscription" defines the callback method "OnAlarm".

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"OnAlarm" method

Specifies the signature of the event handling method for the "OnAlarm" event of an "IPlantObject" instance.

```
OnAlarm(uint32_t systemError,
        CFSTR systemName,

Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator*
pItems)
```

- `systemError`
  Supplies an error code when a global error has occurred. When the error code is set, `pItems` is irrelevant.
- `systemName`
  Name of the Runtime system that is subscribed for alarm monitoring by the user.
- `pItems`
  - Supplies a pointer to "IAlarmResultEnumerator" that can be used to enumerate the active alarms.

19.10.7 Interfaces of the Calendar option (RT Uni)

19.10.7.1 ISHCCalendarOption (RT Uni)

Description

The C++ interface "ISHCCalendarOption" specifies the "GetObject" method. The method supplies the calendar object of an "IPlantObject" instance. A calendar is always integrated via an "IPlantObject" instance.

Members

"GetObject" method

Supplies an error code of the type `CFRESULT`.

```
CFRESULT
GetObject(Siemens::Runtime::HmiUnified::PlantModel::IPlantObject *
pCpmNode, CFSTR objectName, ICfUnknown** ppObject)
```

- `cpmNode`
  [in]: Reference to the "IPlantObject" instance currently selected in the hierarchy
- `objectName`
  [in]: The name of the "IPlantObject" instance
- `ppObject`
  [out]: Returns an object of the type "ICfUnknown". The object is cast to a calendar object using the "QueryInterface" method.
  Example:

```
ISHCCalendarOptionPtr pShcOption;
ISHCCalendarPtr pCalendar;
pRuntime->GetOption(CCfString(ODK_SHC_OPTION),
(IOption**)&pShcOption);
if (nullptr != pShcOption)
{
     ICfUnknownPtr pUnk;
     pShcOption->GetObject(pPlantObject, ODK_SHC_CALENDAR, &pUnk);
     if (nullptr != pUnk)
     {
          pUnk->QueryInterface(IID_ISHCCalendar,
(ICfUnknown**)&pCal);
     }
}
```

Example

The following example serves as a basis for the other examples for the C++ interfaces of the Calendar option.

It shows how you can obtain the "IPlantObject" instance and also an "ISHCCalendar" instance. The "ISHCCalendar" instance referenced via `pCalendar` is also used in the other examples.

Copy code

```
#include <iostream>
#include "IOdkShcInterface.h"
#include "IOdkRt.h"
#include "IOdkRtAlarm.h"
using namespace Siemens::Runtime::HmiUnified::Rt;
using namespace Siemens::Runtime::HmiUnified::Common;
using namespace Siemens::Runtime::HmiUnified::PlantModel;
IRuntimePtr pRuntime;
ISHCCalendarPtr pCalendar;
ISHCCalendarSettingsProviderPtr pCalendarProvider;
ISHCCategoryProviderPtr pShcCategoryProvider;
ISHCShiftTemplatesProviderPtr pShcShiftTemplateProvider;
ISHCDayProviderPtr pShcDayProvider;
ISHCDayTemplatesProviderPtr pShcDayTemplateProvider;
ISHCActionTemplatesProviderPtr pShcActionTemplateProvider;
CCfString projectName = L"";
if (CF_SUCCEEDED(Connect(projectName, &pRuntime)))
{
     ICfUnknownPtr pPlantModelUnk;
     CFRESULT errCode = pRuntime->GetObject(CCfString(L"PlantModel"), &pPlantModelUnk);
     IPlantObjectPtr pPlantObject;
     if (pPlantModelUnk != nullptr && CF_SUCCEEDED(errCode))
     {
          IPlantModelPtr pPlantModel(pPlantModelUnk);
          CCfString strNode = L".hierarchy::Plant/Unit1";
          //gets Object for specified Node path
          errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
          if (CF_SUCCEEDED(errCode) && pPlantObject != nullptr)
          {
               IOptionPtr pOdkOption;
               pRuntime->GetOption(CCfString(ODK_SHC_OPTION), &pOdkOption);
               if (nullptr != pOdkOption)
               {
                   ISHCCalendarOptionPtr pShcOption;
                   pOdkOption->QueryInterface(IID_ISHCCalendarOption,
(ICfUnknown**)&pShcOption);
                   if (nullptr != pShcOption)
                   {
                       ICfUnknownPtr pUnk;
                       pShcOption->GetObject(pPlantObject, ODK_SHC_CALENDAR, &pUnk);
                       if (nullptr != pUnk)
                       {
                          pUnk->QueryInterface(IID_ISHCCalendar, (ICfUnknown**)&pCalendar);
                           if (pCalendar != nullptr)
                           {
                               // Get all data provider
                               pCalendar-
>GetActionTemplatesProvider(&pShcActionTemplateProvider);
                               pCalendar->GetCategoryProvider(&pShcCategoryProvider);
                               pCalendar->GetDayProvider(&pShcDayProvider);
                              pCalendar->GetDayTemplateProvider(&pShcDayTemplateProvider);
                          pCalendar->GetShiftTemplateProvider(&pShcShiftTemplateProvider);
                               pCalendar->GetSettings(&pCalendarProvider);
                           }
                       }
```

Copy code

```
                    }
                }
            }
        }
}
```

19.10.7.2 ISHCCalendar (RT Uni)

Description

The C++ interface "ISHCCalendar" specifies the methods of a calendar.

The interface inherits from the "ICfUnknown" interface.

Members

"GetSettings" method

Supplies the "ISHCCalendarSettings" instances of the calendar.

`CFRESULT GetSettings(ISHCCalendarSettings** calendar)`

- `calendar`
  [out]: The "ISHCCalendarSettings" instance

"GetCategoryProvider" method

Supplies an "ISHCCategoryProvider" instance. The provider enables access to the "ISHCategory" instances of the calendar.

`CFRESULT GetCategoryProvider(ISHCCategoryProvider** ppCatgoryProvider)`

- `ppCatgoryProvider`
  [out]: The "ISHCCategoryProvider" instance

"GetShiftTemplateProvider" method

Supplies an "ISHCShiftTemplatesProvider" instance. The provider enables access to the "ISHCShiftTemplate" instances of the calendar.

`CFRESULT GetShiftTemplateProvider(ISHCShiftTemplatesProvider** ppShiftTemplateProvider)`

- `ppShiftTemplateProvider`
  [out]: The "ISHCShiftTemplatesProvider" instance

"GetDayTemplateProvider" method

Supplies an "ISHCDayTemplatesProvider" instance. The provider enables access to the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT GetDayTemplateProvider(ISHCDayTemplatesProvider**
ppDayTemplateProvider)
```

- `ppDayTemplateProvider`
  [out]: The "ISHCDayTemplatesProvider" instance

"GetActionTemplatesProvider" method

Supplies an "ISHCActionTemplatesProvider" instance. The provider enables access to the "ISHCActionTemplate" instances of the calendar.

```
CFRESULT GetActionTemplatesProvider)(OUT
ISHCActionTemplatesProvider** ppActionTemplatesProvider)
```

- `ppActionTemplatesProvider`
  [out]: The "ISHCActionTemplatesProvider" instance

"GetDayProvider" method

Supplies an "ISHCDayProvider" instance. The provider enables access to the "ISHCDay" instances of the calendar.

```
CFRESULT GetDayProvider(ISHCDayProvider** ppDayProvider)
```

- `ppDayProvider`
  [out]: The "ISHCDayProvider" instance

"GetObject" method

Creates an instance of the type defined in `value` and supplies a reference to the instance.

```
CFRESULT GetObject(const CFSTR value, ICfUnknown** ppObject)
```

- `value`
  [in]: Possible values:
  - "ODK_SHC_OPTION"
  - "ODK_SHC_CALENDAR"
  - "ODK_SHC_TIME_SLICE"
  - "ODK_SHC_DAY_TEMPLATE"
  - "ODK_SHC_DAY"
  - "ODK_SHC_DAY_TEMPLATE"
  - "ODK_SHC_ACTION_TEMPLATE"
  - "ODK_SHC_ACTION_TEMPLATE_ELEMENT"
- `ppObject`
  [out]: Reference to an object of the type "ICfUnknown", which can be cast to the corresponding type.
  Example:

```
ICfUnknownPtr pUnk;
ISHCShiftTemplatePtr pShcShiftTemplate;
pCalendar->GetObject(ODK_SHC_SHIFT_TEMPLATE, &pUnk);
pShcShiftTemplate = (ISHCShiftTemplatePtr)pUnk;
```

19.10.7.3 ISHCCalendarSettings (RT Uni)

Description

The C++ interface "ISHCCalendarSettings" specifies methods for the calendar settings of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetPlantObject" method

Supplies the name of the "IPlantObject" instance to which the calendar belongs.

```
GetPlantObject(CFSTR* pCpmNode)
```

- `pCpmNode`
  [out]: The name of the "IPlantObject" instance

"GetFirstDayOfWeek" method

Supplies the first day of the week.

```
CFRESULT GetFirstDayOfWeek(CFENUM * pvarRet)
```

- `pvarRet`
  Points to the enumeration "ShcWeekDay", which can contain the following values:
  - Sunday (0)
  - Monday (1)
  - Tuesday (2)
  - Wednesday (3)
  - Thursday (4)
  - Friday (5)
  - Saturday (6)

"GetFirstWeekOfYear" method

Supplies the first week of the year.

```
CFRESULT GetFirstWeekOfYear(CFENUM * pvarRet)
```

- `pvarRet`
  [out]: Points to the enumeration "ShcWeekStart", which can contain the following values:
  - FirstOfJanuary (0): The first week starts on the first of January.
  - AtLeastFourDays (1): The first week must have at least four days.
  - WholeWeek (2): The first week must have at least seven days.

"GetFiscalYearStartDay" method

Supplies the first day of the fiscal year.

```
CFRESULT GetFiscalYearStartDay(uint8_t* pvarRet)
```

- `pvarRet`
  [out]: The first day of the fiscal year of the calendar.

"GetFiscalYearStartMonth" method

Supplies the first month of the fiscal year.

```
CFRESULT GetFiscalYearStartMonth(uint8_t* pvarRet)
```

- `pvarRet`
  [out]: The first month of the fiscal year of the calendar.

"GetDayOffset" method

Supplies the offset with which the workday begins, calculated from midnight.

```
CFRESULT GetDayOffset(CFTIMESPAN64 * pvarRet)
```

- `pvarRet`
  [out]: Supplies the number of hours after midnight with which the day begins.

"GetWorkDays" method

Supplies the workdays of the calendar.

```
CFRESULT GetWorkDays(uint8_t* pvarRet)
```

- `pvarRet`
  [out]: The workdays of the calendar.

"GetTimeZone" method

Supplies the time zone of the calendar.

```
CFRESULT GetTimeZone(uint32_t* pTimeZone)
```

- `pvarRet`
  [out]: The time zone ID of Microsoft.

19.10.7.4 ISHCCategory (RT Uni)

Description

The C++ interface "ISHCCategory" specifies the methods of a time category of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCCategory" instance.

```
CFRESULT GetName(CFSTR * pvarRet)
```

- `pvarRet`
  [out]: The name

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCCategory" instance and their language code IDs.

```
CFRESULT GetDescriptions(OUT ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for description).

Example:

```
ICfMapIDToVariantPtr pDescriptions;
hr = pShcCategory->GetDescriptions(&pDescriptions);
if (pDescriptions != nullptr && CF_SUCCEEDED(hr))
{
     std::cout << "Descriptions::" << std::endl << std::endl;
     uint32_t nCount2 = 0;
     pDescriptions->Count(&nCount2);
     for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)
     {
          int32_t nLanguageID;  pDescriptions->KeyAt(nIndex2,
&nLanguageID);
          CCfVariant strDescription;
          pDescriptions->ValueAt(nLanguageID, &strDescription);
          std::cout << "LangauageID =" << nLanguageID << "
Description=" << CCfSmartString(strDescription).ToUTF8().c_str() <<
std::endl;
     }
}
```

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCCategory" instance and their language code IDs.

```
CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for display name).

Example: Similar to "GetDescriptions"

"GetColor" method

Supplies the color of the "ISHCCategory" instance.

```
CFRESULT GetColor(uint32_t* pColor)
```

- `pColor`
  [out]: Returns a 4-byte value for an RGBA color value.

"GetIsDeleted" method

Supplies the information on whether the "ISHCCategory" instance was deleted in Engineering.

```
CFRESULT GetIsDeleted(CFBOOL* p_bIsDeleted) = 0;
```

- `p_bIsDeleted`
  [out]:
  - 0: Was not deleted (default)
  - 1: Was deleted

19.10.7.5 ISHCCategoryEnumerator (RT Uni)

Description

The C++ interface "ISHCCategoryEnumerator" specifies methods for handling the enumeration of the time categories of an "ISHCCalendar" instance. The enumeration is returned by the `Read` method of an "ISHCCategoryProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCCategory** ppItem)
```

- `ppItem`
  [out]: The current "ISHCCategory" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of categories

Example

Copy code

```
void PrintCategory(const ISHCCategoryEnumeratorPtr& p_pShcCategoryEnum)
{
    std::cout << std::endl << "************PrintCategory************" << std::endl << endl;
     if (p_pShcCategoryEnum != nullptr)
     {
          uint32_t nCount = 0;
          p_pShcCategoryEnum->Count(&nCount);
          for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
          {
               if (CF_SUCCEEDED(p_pShcCategoryEnum->MoveNext()))
               {
                    ISHCCategoryPtr pShcCategory;
                    p_pShcCategoryEnum->Current(&pShcCategory);
                    if (pShcCategory != nullptr)
                    {
                         cout << endl;
                         CFRESULT hr = CF_ERROR;
                         CCfString strName;
                         hr = pShcCategory->GetName(&strName);
                         if (CF_SUCCEEDED(hr) && (!strName.IsEmpty()))
                         {
                            cout << "CategoryName= " << strName.ToUTF8().c_str() << endl;
                         }
                         uint32_t nColor;
                         hr = pShcCategory->GetColor(&nColor);
                         if (CF_SUCCEEDED(hr))
                         {
                              cout << "Color= " << nColor << endl;
                         }
                         CFBOOL bIsDeleted;
                         hr = pShcCategory->GetIsDeleted(&bIsDeleted);
                         if (CF_SUCCEEDED(hr))
                         {
                              cout << "IsDeleted= " << (uint32_t)bIsDeleted << endl;
                         }
                         ICfMapIDToVariantPtr pDisplayNames;
                         hr = pShcCategory->GetDisplayNames(&pDisplayNames);
                         if (pDisplayNames != nullptr && CF_SUCCEEDED(hr))
                         {
                              std::cout << "DisplayNames::" << std::endl << std::endl;
                              uint32_t nCount1 = 0;
                              pDisplayNames->Count(&nCount1);
                              for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
                              {
                                   int32_t nLanguageID;
                                   pDisplayNames->KeyAt(nIndex1, &nLanguageID);
                                   CCfVariant strDIsplayname;
                                   pDisplayNames->ValueAt(nLanguageID, &strDIsplayname);
                               std::cout << "LangauageID =" << nLanguageID << " DisplayName
=" << CCfSmartString(strDIsplayname).ToUTF8().c_str() << std::endl;
                              }
                         }
                         ICfMapIDToVariantPtr pDescriptions;
                         hr = pShcCategory->GetDescriptions(&pDescriptions);
```

Copy code

```
                        if (pDescriptions != nullptr && CF_SUCCEEDED(hr))
                        {
                            std::cout << "Descriptions::" << std::endl << std::endl;
                            uint32_t nCount2 = 0;
                            pDescriptions->Count(&nCount2);
                            for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)
                            {
                                int32_t nLanguageID;
                                pDescriptions->KeyAt(nIndex2, &nLanguageID);
                                CCfVariant strDescription;
                                pDescriptions->ValueAt(nLanguageID, &strDescription);
                                std::cout << "LangauageID =" << nLanguageID << "
Description=" << CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
                            }
                        }
                    }
                }
            }
        }
    }
```

19.10.7.6 ISHCCategoryProvider (RT Uni)

Description

The C++ interface "ISHCCategoryProvider" provides you with read access to an "ISHCCategoryEnumerator" instance which contains an enumeration with the time categories of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCCategoryEnumerator" instance which has access to an enumeration with the "ISHCCategory" instances of the calendar.

```
CRFESULT Browse(ISHCCategoryEnumerator** data)
```

- `data`
  [out]: The enumerator

Example:

```
ISHCCategoryEnumeratorPtr pShcCategoryEnum;
CFRESULT hr = pShcCategoryProvider->Browse(&pShcCategoryEnum);
if (CF_SUCCEEDED(hr))
    PrintCategory(pShcCategoryEnum);
```

19.10.7.7 ISHCTimeSlice (RT Uni)

Description

The C++ interface "ISHCTimeSlice" specifies the methods of a time slice.

The interface inherits from the "ICfUnknown" interface.

Members

"GetStartTime" method

Returns the start time of the "ISHCTimeSlice" instance.

```
CFRESULT GetStartTime(CFDATETIME64* pStartTime)
```

- `pStartTime`
  [out]: Time stamp with the start time of the time slice

"GetDuration" method

Supplies the duration of the "ISHCTimeSlice" instance.

```
CFRESULT GetDuration(CFTIMESPAN64* pDuration)
```

- `pDuration`
  [out]: The duration of the time slice

"GetCategory" method

Returns the time category time of the "ISHCTimeSlice" instance.

```
CFRESULT GetCategory(CFSTR* pstrCategoryName)
```

- `pstrCategoryName`
  [out]: The name of the time category

"SetCategory" method

Sets the time category of the "ISHCTimeSlice" instance.

```
CFRESULT SetCategory(CFSTR pstrCategoryName)
```

- `pstrCategoryName`
  [in]: The name of the new time category.

"SetStartTime" method

Sets the start time of the "ISHCTimeSlice" instance.

```
CFRESULT SetStartTime(CFDATETIME64 startTime)
```

- `startTime`
  [in]: The new start time of the time slice.

"SetDuration" method

Sets the duration of the "ISHCTimeSlice" instance.

```
CRFESULT SetDuration(CFTIMESPAN64 duration)
```

- `duration`
  [in]: The new duration of the time slice

19.10.7.8 ISHCTimeSliceEnumerator (RT Uni)

Description

The C++ interface "ISHCTimeSliceEnumerator" specifies methods for handling the enumeration of the time slices of an "ISHCShiftTemplate" instance or "ISHCShift" instance.

The enumeration is returned by the `Read` method of these instances.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCTimeSlice** ppItem)
```

- `ppItem`
  [out]: The current "ISHCTimeSlice" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of time slices

Example

Copy code

```
void printTimeSlice(const ISHCTimeSliceEnumeratorPtr& p_pShcTimeSliceEnum)
{
    std::cout << endl << "***********TimeSlice*********" << std::endl << endl;
    if (p_pShcTimeSliceEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCout = 0;
        p_pShcTimeSliceEnum->Count(&nCout);
        for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
        {
            if (CF_SUCCEEDED(p_pShcTimeSliceEnum->MoveNext()))
            {
                cout << endl;
                ISHCTimeSlicePtr pTimeSlice;
                p_pShcTimeSliceEnum->Current(&pTimeSlice);
                if (pTimeSlice != nullptr)
                {
                    CCfString strCategoryName;
                    hr = pTimeSlice->GetCategory(&strCategoryName);
                    if (CF_SUCCEEDED(hr) && (!strCategoryName.IsEmpty()))
                    {
                        std::cout << "Category= " << strCategoryName.ToUTF8().c_str() <<
std::endl;
                    }
                    CCfTimeSpan64 tsDuration;
                    hr = pTimeSlice->GetDuration(&tsDuration);
                    if (CF_SUCCEEDED(hr))
                    {
                        CCfString strDuration = tsDuration.GetTimeSpanString();
                        std::cout << "Durations= " << strDuration.ToUTF8().c_str() <<
std::endl;
                    }
                    CCfDateTime64 dtStartTime;
                    hr = pTimeSlice->GetStartTime(&dtStartTime);
                    if (CF_SUCCEEDED(hr))
                    {
                        CCfString strStarttime = dtStartTime.GetDateTimeString(false);
                        std::cout << "Start Time= " << strStarttime.ToUTF8().c_str() <<
std::endl;
                    }
                }
            }
        }
    }
}
```

19.10.7.9 ISHCDay (RT Uni)

Description

The C++ interface "ISHCDay" specifies the methods of a day.

The interface inherits from the "ICfUnknown" interface.

Members

"CreateShift" method

Returns a new "ISHCShift" instance.

```
CRFESULT CreateShift(ISHCShiftTemplate* pShiftTemplate, CFTIMESPAN64
startTime, ISHCShift** pShift)
```

- `pShiftTemplate`
  [in]: The "ISHCShiftTemplate" instance on which the new shift is to be based.
- `startTime`
  [in]: Time stamp for the start time of the new shift.
- `pShift`
  [out]: The new shift

"DeleteShift" method

Deletes an "ISHCShift" instance.

```
CRFESULT DeleteShift(ISHCShift* pShift)
```

- `pShift`
  [in]: Reference to the shift to be deleted.

"GetShifts" method

Supplies an "ISHCShiftEnumerator" instance via which you access the "ISHCShift" instances of the "ISHCDay" instance.

```
CRFESULT GetShifts(ISHCShiftEnumerator** ppSHCShiftEnumerator)
```

- `ppSHCShiftEnumerator`
  [out]: The enumerator with which you access the shifts of the "ISHCDay" instance. The shifts are contained in an array.

"GetComments" method

Supplies a map with the comments of the "ISHCDay" instance and their language code IDs.

```
CRFESULT GetComments(ICfMapIDToVariant** ppComments)
```

- `ppComments`
  [out]: A map with int32/string pairs (language code ID for comment).

Example:

```
ICfMapIDToVariantPtr pComments;
hr = pShcshift->GetComments(&pComments);
if (pComments != nullptr && CF_SUCCEEDED(hr))
{
     std::cout << "comments:-" << std::endl << std::endl;
     uint32_t nCount = 0;  pComments->Count(&nCount);
     for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
     {
          int32_t nLanguageID;
```

```
            pComments->KeyAt(nIndex1, &nLanguageID);
            CCfVariant strComments;
            pComments->ValueAt(nLanguageID, &strComments);
           std::cout << "LangauageID =" << nLanguageID << " Comments="
<< CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
      }
}
```

"SetComments" method

Adds a new comment in the specified language to the "ISHCDay" instance.

`CRFESULT SetComment(CFLCID languageId , CFSTR pComments)`

- `languageId`
  [in]: The language code ID
- `pComments`
  [in]: The comment text

"GetStartTime" method

Returns the start time of the "ISHCDay" instance.

`CRFESULT GetStartTime(CFDATETIME64* pStartTime)`

- `pStartTime`
  [out]: The start time

"SetStartTime" method

Sets the start time of the "ISHCDay" instance.

`CRFESULT SetStartTime(CFDATETIME64 startTime)`

- `startTime`
  [in]: The new start time

"GetIsCustomized" method

Supplies the information on whether the "ISHCDay" instance was edited by users.

`CRFESULT GetIsCustomized)(CFBOOL* pIsCustomized)`

- `pIsCustomized`
  [out]:
  - 0: Was not processed
  - 1: Was processed

"GetDayTemplate" method

Supplies the "ISHCDayTemplate" instance on which the "ISHCDay" instance is based.

`CRFESULT GetDayTemplate(CFSTR* pDayTemplate)`

- `pDayTemplate`
  [out]: The day template

"SetDayTemplate" method

Sets the "ISHCDayTemplate" instance of the "ISHCDay" instance.

```
CRFESULT SetDayTemplate(CFSTR pDayTemplate)
```

- `pDayTemplate`
  [in]: The new day template

19.10.7.10 ISHCDayEnumerator (RT Uni)

Description

The C++ interface "ISHCDayEnumerator" specifies methods for handling the enumeration of the days of an "ISHCCalendar" instance. The enumeration is returned by the "Read" method of an "ISHCDayProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCTimeSlice** ppItem)
```

- `ppItem`
  [out]: The current "ISHCDay" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of days

Example

Copy code

```
void PrintDay(const ISHCDayEnumeratorPtr& p_pShcdayEnum)
{
    cout << endl << "***************PrintDay*****************" << endl << endl;
    if (p_pShcdayEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        p_pShcdayEnum->Count(&nCount);
        for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
        {
            cout << endl;
            if (CF_SUCCEEDED(p_pShcdayEnum->MoveNext()))
            {
                ISHCDayPtr pShcday;
                p_pShcdayEnum->Current(&pShcday);
                if (pShcday != nullptr)
                {
                    CCfString strDaytemplate;
                    hr = pShcday->GetDayTemplate(&strDaytemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                      cout << "DayTemplate= " << strDaytemplate.ToUTF8().c_str() << endl;
                    }
                    ICfMapIDToVariantPtr pComments;
                    hr = pShcday->GetComments(&pComments);
                    if (CF_SUCCEEDED(hr) && pComments != nullptr)
                    {
                        std::cout << "comments:-" << std::endl << std::endl;
                        uint32_t nCount1 = 0;
                        pComments->Count(&nCount1);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pComments->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strComments;
                            pComments->ValueAt(nLanguageID, &strComments);
                          std::cout << "LangauageID = " << nLanguageID << " Comments= " <<
CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
                        }
                    }
                    CFBOOL bIsCustomized;
                    hr = pShcday->GetIsCustomized(&bIsCostomized);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "IsCustomized=" << (uint32_t)bIsCostomized << endl;
                    }
                    CCfDateTime64 dtStartTime;
                    hr = pShcday->GetStartTime(&dtStartTime);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "StartTime= " <<
dtStartTime.GetDateTimeString(false).ToUTF8().c_str() << endl;
                    }
                    ISHCShiftEnumeratorPtr pDayShifts;
```

Copy code

```
                    hr = pShcday->GetShifts(&pDayShifts);
                    if (CF_SUCCEEDED(hr))  printShift(pDayShifts);
                }
            }
        }
    }
}
```

19.10.7.11 ISHCDayProvider (RT Uni)

Description

The C++ interface "ISHCDayProvider" provides you with access to an "ISHCDayEnumerator" instance which contains an enumeration with the days of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete days. The provider is returned by the "GetDayProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCDayEnumerator" instance which has access to an enumeration with the "ISHCDay" instances of a specific time period of the calendar.

```
CRFESULT Browse(CFDATETIME64 StartTime, CFDATETIME64 endTime,
ISHCDayEnumerator** ppISHCDayEnumerator)
```

- `startTime`
  [in]: Start time
- `endTime`
  [in]: End time
- `ppISHCDayEnumerator`
  [out]: The enumerator

Example:

Copy code

```
CCfTimeSpan64 Get1Hour()
{
     CCfDateTime64 dt = CCfDateTime64::Now(false);
     CFDATETIMEST st2;
     dt.GetDateTimeStruct(st2);
     st2.cHours = st2.cHours - 1;
     CCfDateTime64 dt2;
     dt2.SetFromDateTimeStruct(&st2);
     return dt.GetDifference(dt2);
}
CCfDateTime64 GetStartoftheDay()
{
     CCfDateTime64 dt = CCfDateTime64::Now(false);
     CFDATETIMEST st;
     dt.GetDateTimeStruct(st);
     st.cHours = 0;
     st.cMinutes = 0;
     st.cSeconds = 0;
     st.sHundredNanoSeconds = 0;
     st.sMicroSeconds = 0;
     st.sMilliSeconds = 0;
     dt.SetFromDateTimeStruct(&st);
     return dt;
}
ISHCDayEnumeratorPtr pDayEnum; // Get day instances for a timespan of three days
CFRESULT hr = pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24),
GetStartoftheDay() + (Get1Hour() * 48), &pDayEnum);
```

"Create" method

Adds new days to the enumeration with the "ISHCDay" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pDays)i
```

- `pDays`
  [in]: An array with the days to be added.

Example:

Copy code

```
void CreateDayWithShift()
{
    if (nullptr != pShcDayTemplateProvider && nullptr != pCalendar && nullptr !=
pShcDayProvider && pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        CCfArrayIUnknown arrDays;
        ICfArrayIUnknownPtr pDays;
        hr = pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcDayTemplates->Count(&nCount);
            if (nCount > 0)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    CCfString strDayTemplateName;
                    pShcDayTemplate->GetName(&strDayTemplateName);
                    ICfUnknownPtr pUnk;
                    pCalendar->GetObject(ODK_SHC_DAY, &pUnk);
                    ISHCDayPtr pShcDay = (ISHCDayPtr)pUnk;
                    if (pShcDay != nullptr)
                    {
                        hr = pShcDay->SetDayTemplate(strDayTemplateName);
                        if (CF_FAILED(hr))
                        {
                            std::cout << "failed to SetDayTemplate" << std::endl;
                        }
                        CCfDateTime64 dt = GetStartoftheDay();
                        hr = pShcDay->SetStartTime(dt);
                        if (CF_FAILED(hr))
                        {
                            std::cout << "failed to SetStartTime" << std::endl;
                        }
                        hr = pShcDay->SetComment(1033, CCfString(L"DayComments"));
                        if (CF_FAILED(hr))
                        {
                            std::cout << "failed to SetComment" << std::endl;
                        }
                    }
                    arrDays.Append(pShcDay);
                    arrDays.DetachEnumerator(&pDays);
                    hr = pShcDayProvider->Create(pDays);
                    ISHCShiftTemplateEnumeratorPtr pShcShiftTemplateEnum;
                 hr = pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplateEnum);
                    if (CF_SUCCEEDED(hr) && pShcShiftTemplateEnum != nullptr)
                    {
                        if (CF_SUCCEEDED(pShcShiftTemplateEnum->MoveNext()))
                        {
                            ISHCShiftTemplatePtr pshcShiftTemplate;
                            pShcShiftTemplateEnum->Current(&pshcShiftTemaplate);
```

Copy code

```
                            ISHCShiftPtr pShcdayShift;
                            CCfTimeSpan64 starttime = Get1Hour() * 10;
                            hr = pShcDay->CreateShift(pshcShiftTemaplate, starttime, 
&pShcdayShift);
                            if (CF_FAILED(hr) || pShcdayShift == nullptr)
                            {
                                std::cout << "failed to Create Shift" << std::endl;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

"Update" method

Updates "ISHCDay" instances of the enumeration.

`CRFESULTUpdate(ICfArrayIUnknown* pDays)`

- `pDays`
  [in]: An array with the days to be updated.

Example:

Copy code

```
void UpdateDayWithShift()
{
    if (pShcDayProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayEnumeratorPtr pDayEnum;
        hr = pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24),
GetStartoftheDay() + (Get1Hour() * 24), &pDayEnum);
        if (pDayEnum != nullptr && CF_SUCCEEDED(hr))
        {
            hr = pDayEnum->MoveNext();
            if (CF_SUCCEEDED(hr))
            {
                CCfArrayIUnknown ArrayDays;
                ICfArrayIUnknownPtr pArrayDays;
                ISHCDayPtr pShcDay;
                pDayEnum->Current(&pShcDay);
                if (pShcDay != nullptr)
                {
                    CCfDateTime64 dt;
                    pShcDay->GetStartTime(&dt);
                    dt.AddTimeSpan(Get1Hour());
                    hr = pShcDay->SetStartTime(dt);//Update StartTime Not Supported
                    hr=pShcDay->SetComment(1033, CCfString(L"DayComment"));
                    if (CF_FAILED(hr))
                    {
                        std::cout << "Failed to SetComment" << std::endl;
                    }
                    ISHCShiftEnumeratorPtr pShiftEnum;
                    hr = pShcDay->GetShifts(&pShiftEnum);
                    if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                    {
                        hr = pShiftEnum->MoveNext();
                        if (CF_SUCCEEDED(hr))
                        {
                            ISHCShiftPtr pShcShift;
                            pShiftEnum->Current(&pShcShift);
                            if (pShcShift != nullptr)
                            {
                                CCfTimeSpan64 ts;
                                hr = pShcShift->GetDuration(&ts);
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to GetDuration" << std::endl;
                                }
                                hr = pShcShift->SetDuration(ts + Get1Hour());
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to SetDuration" << std::endl;
                                }
                                hr = pShcShift->SetComment(1033,
CCfString(L"UpdatedShiftComments"));
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to SetComment" << std::endl;
```

Copy code

```
                                }
                            }
                        }
                    }
                    ArrayDays.Append(pShcDay);
                    ArrayDays.DetachEnumerator(&pArrayDays);
                    hr = pShcDayProvider->Update(pArrayDays);
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to Update" << std::endl;
                    }
                }
            }
        }
    }
}
```

"Delete" method

Deletes "ISHCDay" instances of the calendar from the enumeration.

`CRFESULT Delete(ICfArrayIUnknown* pActionTemplates)`

- `pActionTemplates`
  [in]: An array with the days to be deleted.

Example:

Copy code

```
void DeleteDayWithShift()
{
    if (pShcDayProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayEnumeratorPtr pDayEnum;
        hr = pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24),
GetStartoftheDay() + (Get1Hour() * 48), &pDayEnum);
        if (pDayEnum != nullptr && CF_SUCCEEDED(hr))
        {
            uint32_t nSize = 0;
            pDayEnum->Count(&nSize);
            CCfArrayIUnknown ArrayDays;
            ICfArrayIUnknownPtr pArrayDays;
            for (uint32_t nIdnex = 0;nIdnex < nSize;nIdnex++)
            {
                hr = pDayEnum->MoveNext();
                if (CF_SUCCEEDED(hr))
                {
                    ISHCDayPtr pShcDay;
                    pDayEnum->Current(&pShcDay);
                    if (pShcDay != nullptr)
                    {
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDay->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            hr = pShiftEnum->MoveNext();
                            if (CF_SUCCEEDED(hr))
                            {
                                ISHCShiftPtr pShcShift;
                                pShiftEnum->Current(&pShcShift);
                                if (pShcShift != nullptr)
                                {
                                    hr = pShcDay->DeleteShift(pShcShift);
                                    if (CF_FAILED(hr))
                                    {
                                       std::cout << "failed to DeleteShift" << std::endl;
                                    }
                                }
                            }
                        }
                        ArrayDays.Append(pShcDay);
                    }
                    ArrayDays.DetachEnumerator(&pArrayDays);
                    hr = pShcDayProvider->Delete(pArrayDays);
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to Delete" << std::endl;
                    }
                }
            }
        }
    }
}
```

19.10.7.12 ISHCDayTemplate (RT Uni)

Description

The C++ interface "ISHCDayTemplate" specifies the methods of a day template.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCDayTemplate" instance.

```
CFRESULT GetName(CFSTR * pvarRet)
```

- `pvarRet`
  [out]: The name of the day template

"SetName" method

Sets the name of the "ISHCDayTemplate" instance.

```
CRFESULT SetName(CFSTR  value)
```

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCDayTemplate" instance and their language code IDs.

```
CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for display name).

Example:

```
ICfMapIDToVariantPtr pDisplayNames;
CFRESULT hr = pShcDayTemplate->GetDisplayNames(&pDisplayNames);
if (pDisplayNames != nullptr && CF_SUCCEEDED(hr))
{
     std::cout << "DisplayNames::" << std::endl << std::endl;
     uint32_t nCount2 = 0;
     pDisplayNames->Count(&nCount2);
     for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)
     {
          int32_t nLanguageID;
          pDisplayNames->KeyAt(nIndex2, &nLanguageID);
          CCfVariant strDIsplayname;
          pDisplayNames->ValueAt(nLanguageID, &strDIsplayname);
          std::cout << "LangauageID =" << nLanguageID << "
DisplayName=" << CCfSmartString(strDIsplayname).ToUTF8().c_str() <<
std::endl;
     }
}
```

"SetDisplayName" method

Adds a new display name in the specified language to the "ISHCDayTemplate" instance.

```
CRFESULT SetDisplayName(CFLCID languageId, CFSTR pDisplayName)
```

- `languageId`
  [in]: The language code ID
- `pDisplayName`
  [in]: The comment text

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCDayTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for description).

Example: Similar to "GetDescriptions" of "ISHCCategory".

"SetDescription" method

Adds a new description in the specified language to the "ISHCDayTemplate" instance.

```
CRFESULT SetDescription)(CFLCID languageId, CFSTR pDescriptions)
```

- `languageId`
  [in]: The language code ID
- `pDescriptions`
  [in]: The description text

"GetShifts" method

Supplies an "ISHCShiftEnumerator" instance. The enumerator provides you with access to the "ISHCShift" instances of the "ISHCDayTemplate" instance.

```
GetShifts(ISHCShiftEnumerator** ppSHCShiftEnumerator)
```

- `ppSHCShiftEnumerator`
  [out]: The enumerator

"CreateShift" method

Adds an "ISHCShift" instance to the "ISHCDayTemplate" instance.

```
CRFESULT CreateShift(ISHCShiftTemplate* pShiftTemplate, CFTIMESPAN64
pStartTime, ISHCShift** pShift)
```

- `pShiftTemplate`
  [in]: Reference to the shift template on which the shift is based.
- `pStartTime`
  [in] Time stamp with the start time of the shift
- `pShift`
  [out] reference to the new shift

"DeleteShift" method

Deletes an "ISHCShift" instance of the "ISHCDayTemplate" instance.

```
CFRESULTDeleteShift(ISHCShift* pShift)
```

- `pShift`
  [in]: Reference to the shift to be deleted

"GetIsDeleted" method

Supplies the information on whether the "ISHCDayTemplate" instance was deleted by users.

```
CFRESULT GetIsDeleted(CFBOOL* pIsDeleted)
```

- `pIsDeleted`
  [out]:
  - 0: Was not deleted
  - 1: Was deleted

19.10.7.13 ISHCDayTemplatesProvider (RT Uni)

Description

The C++ interface "ISHCDayTemplatesProvider" provides you with access to an "ISHCDayTemplateEnumerator" instance which contains an enumeration with the day templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete day templates. The provider is returned by the "GetDayTemplateProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCDayTemplateEnumerator" instance which has access to an enumeration with the "ISHCDayTemplate" instances of the calendar.

```
CRFESULT Browse(CFBOOL includeDeleted, ISHCDayTemplateEnumerator**
ppISHCDayTemplateEnumerator)
```

- `includeDeleted`
  Saves whether the enumerator also has access to the deleted day templates.
- `ppISHCDayTemplateEnumerator`
  [out]: The enumerator

Example:

```
ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
CFRESULT hr = pShcDayTemplateProvider->Browse(CF_FALSE,
&pShcDayTemplates);
```

"Create" method

Adds new "ISHCDayTemplate" instances to the enumeration with the"ISHCDayTemplate" instances of the calendar.

```
CRFESULTUpdate(ICfArrayIUnknown* pDayTemplates)
```

- `pDayTemplates`
  [in]: An array with the day templates to be added.

Example:

Copy code

```
void CreateDayTemplateWithShift()
{
    if (pCalendar != nullptr && pShcDayTemplateProvider != nullptr &&
pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ICfUnknownPtr pUnk;
        pCalendar->GetObject(ODK_SHC_DAY_TEMPLATE, &pUnk);
        CCfArrayIUnknown ArrayTemplate;
        ICfArrayIUnknownPtr pArrayTemplate;
        ISHCDayTemplatePtr pShcDayTemplate = (ISHCDayTemplatePtr)pUnk;
        if (pShcDayTemplate != nullptr)
        {
            hr = pShcDayTemplate->SetName(CCfString(L"DaytemplateName"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetName" << std::endl;
            }
            hr = pShcDayTemplate->SetDisplayName(1033,
CCfString(L"DayTemplateDisplayName"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetDisplayName" << std::endl;
            }
            hr = pShcDayTemplate->SetDescription(1033, CCfString(L"DayTemplate
Descriptions"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetDescription" << std::endl;
            }
        }
        ArrayTemplate.Append(pShcDayTemplate);
        ArrayTemplate.DetachEnumerator(&pArrayTemplate);
        hr = pShcDayTemplateProvider->Create(pArrayTemplate);
        if (CF_SUCCEEDED(hr))
        {
            ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
            hr = pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
            if (CF_SUCCEEDED(hr) && pShcShiftTemplates != nullptr)
            {
                if (CF_SUCCEEDED(pShcShiftTemplates->MoveNext()))
                {
                    ISHCShiftTemplatePtr pShcshiftTemplate;
                    pShcShiftTemplates->Current(&pShcshiftTemplate);
                    if (pShcshiftTemplate != nullptr)
                    {
                        CCfString strShiftTemplateName;
                        pShcshiftTemplate->GetName(&strShiftTemplateName);
                        ISHCShiftPtr pShift;
                      hr = pShcDayTemplate->CreateShift(pShcshiftTemplate, Get1Hour() * 1,
&pShift);
                        if (CF_FAILED(hr) || pShift == nullptr)
                        {
                            cout << "Failed to Create Shift" << endl;
                        }
```

Copy code

```
                        }
                    }
                }
            }
        }
    }
```

"Update" method

Updates "ISHCDayTemplate" instances of the enumeration.

`CRFESULTUpdate(ICfArrayIUnknown* pDayTemplates)`

- `pDayTemplates`
  [in]: An array with the day templates to be updated.

Example:

Copy code

```
void UpdateDayTemplateWithShift()
{
    if (pShcDayTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        hr = pShcDayTemplateProvider->Browse(false, &pShcDayTemplates);
        CCfArrayIUnknown ArrayDayTemplate;
        ICfArrayIUnknownPtr pArrayDayTemplate;
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;  pShcDayTemplates->Count(&nCount);
            for (uint32_t i = 0; i < nCount; i++)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                       hr = pShcDayTemplate->SetName(CCfString(L"UpdatedDayTemplateName"));
                        if (CF_FAILED(hr))
                        {
                            std::cout << "SetName failed" << std::endl;
                        }
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDayTemplate->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pShiftEnum->Count(&nlength);
                            for (uint32_t nIndex = 0; nIndex < nlenght; nIndex++)
                            {
                                if (CF_SUCCEEDED(pShiftEnum->MoveNext()))
                                {
                                    ISHCShiftPtr pShcShift;
                                    hr = pShiftEnum->Current(&pShcShift);
                                    if (CF_SUCCEEDED(hr) && pShcShift != nullptr)
                                    {
                                        ISHCTimeSliceEnumeratorPtr pShcTimeSliceEnum;
                                       hr = pShcShift->GetTimeSlices(&pShcTimeSliceEnum);
                                        pShcShift->SetDuration(Get1Hour() * 8);
                                   pShcShift->SetComment(1033, CCfString("ShiftComment"));
                                    if (CF_SUCCEEDED(hr) && pShcTimeSliceEnum != nullptr)
                                        {
                                            uint32_t nCount1 = 0;
                                            hr = pShcTimeSliceEnum->Count(&nCount1);
                                      for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1+
+)
                                            {
                                         if (CF_SUCCEEDED(pShcTimeSliceEnum->MoveNext()))
                                                {
                                                    ISHCTimeSlicePtr pShctimeslice;
                                                    hr = pShcTimeSliceEnum-
>Current(&pShctimeslice);
```

Copy code

```
                                                    if (pShctimeslice != nullptr)
                                                    {
                                                        hr = pShctimeslice-
>SetCategory(CCfString(L"Maintenance"));
                                                        if (CF_FAILED(hr))
                                                        {
                                                      std::cout << "failed to SetCategory"
<< std::endl;
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                        ArrayDayTemplate.Append(pShcDayTemplate);
                    }
                }
            }
            ArrayDayTemplate.DetachEnumerator(&pArrayDayTemplate);
        }
        hr = pShcDayTemplateProvider->Update(pArrayDayTemplate);
        if (CF_FAILED(hr))
        {
            std::cout << "failed to Update" << std::endl;
        }
    }
}
```

"Delete" method

Deletes "ISHCDayTemplate" instances of the calendar from the enumeration.

`CRFESULT Delete(ICfArrayIUnknown* pDayTemplates)`

- `pDayTemplates`
  [in]: An array with the day templates to be deleted.

Example:

Copy code

```
void DeleteDaytemplateWithShift()
{
    if (pShcDayTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        hr = pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
        CCfArrayIUnknown ArrayDayTemplate;
        ICfArrayIUnknownPtr pArrayDayTemplate;
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcDayTemplates->Count(&nCount);
            for (uint32_t i = 0; i < nCount; i++)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    if (CF_SUCCEEDED(hr) && pShcDayTemplate != nullptr)
                    {
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDayTemplate->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pShiftEnum->Count(&nlength);
                            for (uint32_t nIndex = 0; nIndex < nlenght; nIndex++)
                            {
                                if (CF_SUCCEEDED(pShiftEnum->MoveNext()))
                                {
                                    ISHCShiftPtr pShcShift;
                                    hr = pShiftEnum->Current(&pShcShift);
                                    if (CF_SUCCEEDED(hr) && pShcShift != nullptr)
                                    {
                                        hr = pShcDayTemplate->DeleteShift(pShcShift);
                                        if (CF_FAILED(hr))
                                        {
                                            std::cout << "failed to Update" << std::endl;
                                        }
                                    }
                                }
                            }
                        }
                        ArrayDayTemplate.Append(pShcDayTemplate);
                    }
                }
            }
            ArrayDayTemplate.DetachEnumerator(&pArrayDayTemplate);
            hr = pShcDayTemplateProvider->Delete(pArrayDayTemplate);
            if (CF_FAILED(hr))
            {
                std::cout << "failed to Update" << std::endl;
            }
        }
```

Copy code

```
    }
}
```

19.10.7.14 ISHCShiftTemplate (RT Uni)

Description

The C++ interface "ISHCShiftTemplate" specifies the methods of a shift template.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCShiftTemplate" instance.

`GetName(CFSTR * pvarRet)`

- `pvarRet`
  [out]: Name

"SetName" method

Sets the name of the "ISHCShiftTemplate" instance.

`CRFESULT SetName(CFSTR  value)`

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

`CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)`

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for display name).

"SetDisplayName" method

Adds a new entry to a map with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

`CRFESULT SetDisplayName(CFLCID languageId, CFSTR pDisplayName)`

- `languageId`
  [in]: The language code ID
- `pDisplayName`
  [in]: The comment text

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCShiftTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for description).

"SetDescription" method

Adds a new entry to a map with the description of the "ISHCShiftTemplate" instance and their language code IDs.

```
CRFESULT SetDescription(CFLCID languageId, CFSTR pDescriptions)
```

- `languageId`
  [in]: The language code ID
- `pDescriptions`
  [in]: The description text

"GetIsDeleted" method

Supplies the information on whether the "ISHCShiftTemplate" instance was deleted by users.

```
CFRESULT GetIsDeleted(CFBOOL* pIsDeleted)
```

- `pIsDeleted`
  [out]:
  - 0: Was not deleted
  - 1: Was deleted

"GetDuration" method

Supplies the duration of the "ISHCShiftTemplate" instance.

```
CFRESULT GetDuration(CFTIMESPAN64* pDuration)
```

- `pDuration`
  [out]: The duration of the shift template

"SetDuration" method

Sets the duration of the "ISHCShiftTemplate" instance.

```
CFRESULT SetDuration(CFTIMESPAN64 duration)
```

- `duration`
  [in]: The duration of the shift template

"GetTimeSlices" method

Supplies an "ISHCTimeSliceEnumerator" instance. The enumerator provides you with access to the "ISHCTimeSlice" instances of the "ISHCShiftTemplate" instance.

```
CRFESULT GetTimeSlices(ISHCTimeSliceEnumerator**
ppSHCTimeSliceEnumerator)
```

- `ppSHCTimeSliceEnumerator`
  [out]: The enumerator

"CreateTimeSlice" method

Adds an "ISHCTimeSlice" instance to the "ISHCShiftTemplate" instance.

```
CRFESULT CreateTimeSlice(ISHCTimeSlice* pSlice)
```

- `pSlice`
  [in]: Reference to the new time slice

"DeleteTimeSlice" method

Deletes an "ISHCTimeSlice" instance of the "ISHCShiftTemplate" instance.

```
CRFESULT DeleteTimeSlice(ISHCTimeSlice* pSlice)
```

- `pSlice`
  [in]: Reference to the time slice to be deleted

19.10.7.15 ISHCShiftTemplateEnumerator (RT Uni)

Description

The C++ interface "ISHCShiftTemplateEnumerator" specifies methods for handling the enumeration of the shift templates of an "ISHCCalendar" instance. The enumeration is returned by the "Read" method of an "ISHCShiftTemplatesProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCShiftTemplate** ppItem)
```

- `ppItem`
  [out]: The current shift template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

Supplies the number of shift templates in the list.

```
CFRESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of shift templates

Example

Copy code

```
void PrintShiftTemplate(const ISHCShiftTemplateEnumeratorPtr& p_pShcShiftTemplateEnum)
{
    std::cout << std::endl << "************PrintShiftTemplate************" << std::endl <<
endl;
    if (p_pShcShiftTemplateEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCout = 0;
        p_pShcShiftTemplateEnum->Count(&nCout);
        for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
        {
            cout << endl;
            if (CF_SUCCEEDED(p_pShcShiftTemplateEnum->MoveNext()))
            {
                ISHCShiftTemplatePtr pShcShiftTemplate;
                p_pShcShiftTemplateEnum->Current(&pShcShiftTemplate);
                if (pShcShiftTemplate != nullptr)
                {
                    CCfString strName;
                    hr = pShcShiftTemplate->GetName(&strName);
                    if (CF_SUCCEEDED(hr) && (!strName.IsEmpty()))
                    {
                        std::cout << "Name=" << strName.ToUTF8().c_str() << std::endl;
                    }
                    CCfTimeSpan64 tsDuration;
                    hr = pShcShiftTemplate->GetDuration(&tsdurantion);
                    if (CF_SUCCEEDED(hr))
                    {
                        CCfString strDuration = tsdurantion.GetTimeSpanString();
                        std::cout << "Duration=" << strDuration.ToUTF8().c_str() <<
std::endl;
                    }
                    CFBOOL bIsDeleted;
                    hr = pShcShiftTemplate->GetIsDeleted(&bIsDeleted);
                    if (CF_SUCCEEDED(hr))
                    {
                        std::cout << "ISDeleted=" << (uint32_t)bIsDeleted << std::endl;
                    }
                    ICfMapIDToVariantPtr pDescriptions;
                    hr = pShcShiftTemplate->GetDescriptions(&pDescriptions);
                    if (pDescriptions != nullptr && CF_SUCCEEDED(hr))
                    {
                        std::cout << "Descriptions=" << std::endl << std::endl;  uint32_t
nCount = 0;
                        pDescriptions->Count(&nCount);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pDescriptions->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strDescription;
                            pDescriptions->ValueAt(nLanguageID, &strDescription);
                            std::cout << "LangauageID =" << nLanguageID << " Description="
<< CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
                        }
```

Copy code

```
                    }
                    ICfMapIDToVariantPtr pDisplayNames;
                    hr = pShcShiftTemplate->GetDisplayNames(&pDisplayNames);
                    if (pDisplayNames != nullptr && CF_SUCCEEDED(hr))
                    {
                        std::cout << "DisplayNames::" << std::endl << std::endl;
                        uint32_t nCount = 0;
                        pDisplayNames->Count(&nCount);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pDisplayNames->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strDIsplayname;
                            pDisplayNames->ValueAt(nLanguageID, &strDIsplayname);
                            std::cout << "LangauageID =" << nLanguageID << " DisplayName="
<< CCfSmartString(strDIsplayname).ToUTF8().c_str() << std::endl;
                        }
                    }
                    ISHCTimeSliceEnumeratorPtr pShiftTemplatetimeSlice;
                    hr = pShcShiftTemplate->GetTimeSlices(&pShiftTemplatetimeSlice);
                    if (CF_SUCCEEDED(hr))  printTimeSlice(pShiftTemplatetimeSlice);
                }
            }
        }
    }
}
```

19.10.7.16 ISHCShiftTemplatesProvider (RT Uni)

Description

The C++ interface "ISHCShiftTemplatesProvider" provides you with access to an "ISHCShiftTemplateEnumerator" instance which contains an enumeration with the shift templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete shift templates. The provider is returned by the "GetShiftTemplateProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCShiftTemplateEnumerator" instance which has access to an enumeration with the "ISHCShiftTemplate" instances of the calendar.

```
CRFESULT Browse(CFBOOL includeDeleted, ISHCShiftTemplateEnumerator**
ppISHCShiftTemplateEnumerator)
```

- `includeDeleted`
  Saves whether the enumerator also has access to the deleted shift templates.
- `ppISHCShiftTemplateEnumerator`
  [out]: The enumerator

Example:
```
ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
CFRESULT hr = pShcDayTemplateProvider->Browse(CF_FALSE,
&pShcDayTemplates);
```

"Create" method

Adds new shift templates to the enumeration with the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pShiftTemplates)
```

- `pShiftTemplates`
  [in]: An array with the shift templates to be added.

Example:

Copy code

```
void CreateShiftTemplateWithTimeslice()
{
    if (pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCShiftTemplatePtr pShcShiftTemplate;
        ICfUnknownPtr pUnk;
        pCalendar->GetObject(ODK_SHC_SHIFT_TEMPLATE, &pUnk);
        pShcShiftTemplate = (ISHCShiftTemplatePtr)pUnk;
        if (pShcShiftTemplate != nullptr)
        {
            hr = pShcShiftTemplate->SetName(CCfString(L"ShiftTemplateName"));
            if (CF_FAILED(hr))
            {
                cout << "Failed to SetName" << endl;
            }
            hr = pShcShiftTemplate->SetDisplayName(1033, CCfString(L"ShiftDisplayName"));
            if (CF_FAILED(hr))
            {
                cout << "Failed to SetDisplayName" << endl;
            }
            hr = pShcShiftTemplate->SetDescription(1033,
CCfString(L"ShiftTemplateDescription"));
            if (CF_FAILED(hr))
            {
                cout << "Failed to SetDescription" << endl;
            }
            hr = pShcShiftTemplate->SetDuration(Get1Hour() * 8);
            if (CF_FAILED(hr))
            {
                cout << "Failed to SetDuration" << endl;
            }
            ICfArrayIUnknownPtr pArrayShiftTemplate;
            CCfArrayIUnknown ArrayShiftTemplate;
            ArrayShiftTemplate.Append(pShcShiftTemplate);
            ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);
            hr = pShcShiftTemplateProvider->Create(pArrayShiftTemplate);
            if (CF_FAILED(hr))
            {
                cout << "Failed to v" << endl;
            }
            ICfUnknownPtr pUnkTimeSlice;
            pCalendar->GetObject(ODK_SHC_TIME_SLICE, &pUnkTimeSlice);
            if (pUnkTimeSlice != nullptr)
            {
                ISHCTimeSlicePtr pShcTimeSlice;
                pUnkTimeSlice->QueryInterface(IID_ISHCTimeSlice,
(ICfUnknown**)&pShcTimeSlice);
                if (pShcTimeSlice != nullptr)
                {
                    pShcTimeSlice->SetCategory(CCfString(L"Working"));
                    pShcTimeSlice->SetDuration(Get1Hour() * 1);
                    CCfDateTime64 dt = GetStartoftheDay();
                    pShcTimeSlice->SetStartTime(dt);
                    hr = pShcShiftTemplate->CreateTimeSlice(pShcTimeSlice);
```

Copy code

```
                    if (CF_FAILED(hr))
                    {
                        std::cout << "Failed to Create TimeSlcie" << std::endl;
                    }
                }
            }
        }
    }
}
```

"Update" method

Updates the enumeration with the "ISHCShiftTemplate" instances of the calendar.

`CRFESULT Update(ICfArrayIUnknown* pShiftTemplates)`

- `pShiftTemplates`
  [in]: An array with the shift templates to be updated.

Example:

Copy code

```
void UpdateShiftTemplateWithTimeSlice()
{
    if (pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
        hr = pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
        CCfArrayIUnknown ArrayShiftTemplate;
        ICfArrayIUnknownPtr pArrayShiftTemplate;
        if (CF_SUCCEEDED(hr) && pShcShiftTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcShiftTemplates->Count(&nCount);
            for (uint32_t i = 0; i < nCount; i++)
            {
                if (CF_SUCCEEDED(pShcShiftTemplates->MoveNext()))
                {
                    ISHCShiftTemplatePtr pShcShiftTemplate;
                    hr = pShcShiftTemplates->Current(&pShcShiftTemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                        hr = pShcShiftTemplate->SetDuration(Get1Hour() * 6);
                        if (CF_FAILED(hr))
                        {
                            std::cout << "SetDuration failed" << std::endl;
                        }
                        hr = pShcShiftTemplate-
>SetName(CCfString(L"UpdatedShiftTemplateName"));
                        if (CF_FAILED(hr))
                        {
                            std::cout << "SetName failed" << std::endl;
                        }
                        ISHCTimeSliceEnumeratorPtr pTimeSlilceEnum;
                        hr = pShcShiftTemplate->GetTimeSlices(&pTimeSlilceEnum);
                        if (CF_SUCCEEDED(hr) && pTimeSlilceEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pTimeSlilceEnum->Count(&nlength);
                            for (uint32_t nIndex = 0; nIndex < nlenght; nIndex++)
                            {
                                if (CF_SUCCEEDED(pTimeSlilceEnum->MoveNext()))
                                {
                                    ISHCTimeSlicePtr pShcTimeSlice;
                                    hr = pTimeSlilceEnum->Current(&pShcTimeSlice);
                                    if (CF_SUCCEEDED(hr) && pShcTimeSlice != nullptr)
                                    {
                                        pShcTimeSlice->SetDuration(Get1Hour() * 4);
                                        pShcTimeSlice->SetCategory(CCfString(L"Break"));
                                    }
                                }
                            }
                        }
                        ArrayShiftTemplate.Append(pShcShiftTemplate);
                    }
                }
```

Copy code

```
            }
            ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);
        }
        hr = pShcShiftTemplateProvider->Update(pArrayShiftTemplate);
        if (CF_FAILED(hr))
        {
            std::cout << "failed to Update" << std::endl;
        }
    }
}
```

"Delete" method

Deletes an "ISHCShiftTemplate" instance of the calendar from the enumeration.

`CRFESULT Delete(ICfArrayIUnknown* pShiftTemplates)`

- `pShiftTemplates`
  [in]: An array with the shift templates to be deleted.

Example:

Copy code

```
void DeleteShiftTemplateWithTimeSlice()
{
    if (pShcShiftTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
        hr = pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
        CCfArrayIUnknown ArrayShiftTemplate;
        ICfArrayIUnknownPtr pArrayShiftTemplate;
        if (CF_SUCCEEDED(hr) && pShcShiftTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcShiftTemplates->Count(&nCount);
            for (uint32_t i = 0; i < nCount; i++)
            {
                if (CF_SUCCEEDED(pShcShiftTemplates->MoveNext()))
                {
                    ISHCShiftTemplatePtr pShcShiftTemplate;
                    hr = pShcShiftTemplates->Current(&pShcShiftTemplate);
                    if (CF_SUCCEEDED(hr) && pShcShiftTemplate != nullptr)
                    {
                        ISHCTimeSliceEnumeratorPtr pTimeSlilceEnum;
                        hr = pShcShiftTemplate->GetTimeSlices(&pTimeSlilceEnum);
                        if (CF_SUCCEEDED(hr) && pTimeSlilceEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pTimeSlilceEnum->Count(&nlength);
                            for (uint32_t nIndex = 0; nIndex < nlenght; nIndex++)
                            {
                                if (CF_SUCCEEDED(pTimeSlilceEnum->MoveNext()))
                                {
                                    ISHCTimeSlicePtr pShcTimeSlice;
                                    hr = pTimeSlilceEnum->Current(&pShcTimeSlice);
                                    if (pShcTimeSlice != nullptr)
                                    {
                                   hr = pShcShiftTemplate->DeleteTimeSlice(pShcTimeSlice);
                                        if (CF_FAILED(hr))
                                        {
                                            std::cout << "failed to DeleteTimeSlice" <<
std::endl;
                                        }
                                    }
                                }
                            }
                        }
                        ArrayShiftTemplate.Append(pShcShiftTemplate);
                    }
                }
            }
            ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);
        }
        hr = pShcShiftTemplateProvider->Delete(pArrayShiftTemplate);
        if (CF_FAILED(hr))
        {
            std::cout << "failed to Delete" << std::endl;
```

Copy code

```
            }
        }
}
```

19.10.7.17 ISHCShift (RT Uni)

Description

The C++ interface "ISHCShift" specifies the methods of a shift.

The interface inherits from the "ICfUnknown" interface.

Members

"GetTimeSlices" method

Supplies an "ISHCTimeSliceEnumerator" instance. The enumerator provides you with access to the "ISHCTimeSlice" instances of the "ISHCShift" instance.

```
CRFESULT GetTimeSlices(ISHCTimeSliceEnumerator**
ppSHCTimeSliceEnumerator)
```

- `ppSHCTimeSliceEnumerator`
  [out]: The enumerator

"CreateTimeSlice" method

Adds an "ISHCTimeSlice" instance to the "ISHCShiftTemplate" instance.

```
CRFESULT CreateTimeSlice(ISHCTimeSlice* pSlice)
```

- `pSlice`
  [in]: Reference to the new time slice

Example

Copy code

```
void CreateTimeSliceUsingShift()
{
    if (nullptr != pShcDayTemplateProvider && nullptr != pCalendar)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        hr = pShcDayTemplateProvider->Read(CF_FALSE, &pShcDayTemplates);
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcDayTemplates->Count(&nCount);
            if (nCount > 0)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDayTemplate->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pShiftEnum->Count(&nlength);
                            if (nlength > 0)
                            {
                                if (CF_SUCCEEDED(pShiftEnum->MoveNext()))
                                {
                                    ISHCShiftPtr pShcShift;
                                    hr = pShiftEnum->Current(&pShcShift);
                                    if (CF_SUCCEEDED(hr) && pShcShift != nullptr)
                                    {
                                        ISHCTimeSlicePtr pShcTimeSlice;
                                        ICfUnknownPtr pUnk;
                                    hr = pCalendar->GetObject(ODK_SHC_TIME_SLICE, &pUnk);
                                        pShcTimeSlice = (ISHCTimeSlicePtr)pUnk;
                                        if (pShcTimeSlice != nullptr)
                                        {
                                         pShcTimeSlice->SetCategory(CCfString(L"Break"));
                                            pShcTimeSlice->SetDuration(Get1Hour() * 1);
                                            CCfDateTime64 dt = GetStartoftheDay();
                                            dt.AddTimeSpan(Get1Hour() * 3);
                                            pShcTimeSlice->SetStartTime(dt);
                                        }
                                        hr = pShcShift->CreateTimeSlice(pShcTimeSlice);
                                        if (CF_FAILED(hr))
                                        {
                                            std::cout << "failed to CreateTimeSlice" <<
std::endl;
                                        }
                                    }
                                }
                            }
                        }
```

Copy code

```
                    }
                }
            }
        }
    }
}
```

"DeleteTimeSlice" method

Deletes an "ISHCTimeSlice" instance of the "ISHCShiftTemplate" instance.

`CRFESULT DeleteTimeSlice(ISHCTimeSlice* pSlice)`

- `pSlice`
  [in]: Reference to the time slice to be deleted

Example:

Copy code

```
void DeleteTimeSliceUsingShift()
{
    if (nullptr != pShcDayTemplateProvider && nullptr != pCalendar)
    {
        CFRESULT hr = CF_ERROR;
        ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
        hr = pShcDayTemplateProvider->Read(CF_FALSE, &pShcDayTemplates);
        if (CF_SUCCEEDED(hr) && pShcDayTemplates != nullptr)
        {
            uint32_t nCount = 0;
            pShcDayTemplates->Count(&nCount);
            if (nCount > 0)
            {
                if (CF_SUCCEEDED(pShcDayTemplates->MoveNext()))
                {
                    ISHCDayTemplatePtr pShcDayTemplate;
                    hr = pShcDayTemplates->Current(&pShcDayTemplate);
                    if (CF_SUCCEEDED(hr))
                    {
                        ISHCShiftEnumeratorPtr pShiftEnum;
                        hr = pShcDayTemplate->GetShifts(&pShiftEnum);
                        if (CF_SUCCEEDED(hr) && pShiftEnum != nullptr)
                        {
                            uint32_t nlength = 0;
                            pShiftEnum->Count(&nlength);
                            if (nlength > 0)
                            {
                                if (CF_SUCCEEDED(pShiftEnum->MoveNext()))
                                {
                                    ISHCShiftPtr pShcShift;
                                    hr = pShiftEnum->Current(&pShcShift);
                                    if (CF_SUCCEEDED(hr) && pShcShift != nullptr)
                                    {
                                        ISHCTimeSliceEnumeratorPtr pShcTimeSliceEnum;
                                       hr = pShcShift->GetTimeSlices(&pShcTimeSliceEnum);
                                    if (CF_SUCCEEDED(hr) && pShcTimeSliceEnum != nullptr)
                                        {
                                            uint32_t nCountTimeSlice = 0;
                                            pShcTimeSliceEnum->Count(&nCountTimeSlice);
                                            if (nCountTimeSlice > 0)
                                            {
                                         if (CF_SUCCEEDED(pShcTimeSliceEnum->MoveNext()))
                                                {
                                                    ISHCTimeSlicePtr pShcTimeSlice;
                                                    hr = pShcTimeSliceEnum-
>Current(&pShcTimeSlice);
                                                    if (CF_SUCCEEDED(hr) && pShcTimeSlice !
= nullptr)
                                                    {
                                                        hr = pShcShift-
>DeleteTimeSlice(pShcTimeSlice);
                                                        if (CF_FAILED(hr))
                                                        {
                                                   std::cout << "failed to DeleteTimeSlice"
<< std::endl;
```

Copy code

```
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

"GetComments" method

Supplies a map with the comments of the "ISHCShift" instance and their language code IDs.

`CRFESULT GetComments(ICfMapIDToVariant** ppComments)`

- `ppComments`
  [out]: A map with int32/string pairs (language code ID for comment).

"SetComments" method

Adds a new entry to a map with the comments of the "ISHCShift" instance and their language code IDs.

`CRFESULT SetComment(CFLCID languageId , CFSTR pComments)`

- `languageId`
  [in]: The language code ID
- `pComments`
  [in]: The comment text

"GetDuration" method

Supplies the duration of the "ISHCShift" instance.

`CFRESULT GetDuartion(CFTIMESPAN64* pDuration)`

- `pDuration`
  [out]: The duration of the shift

"SetDuration" method

Sets the duration of the "ISHCShift" instance.

`CFRESULT SetDuration(CFTIMESPAN64 duration)`

- `duration`
  [in]: The duration of the shift

"GetShiftTemplate" method

Supplies the "ISHCShiftTemplate" instance on which the "ISHCShift" instance is based.

```
CRFESULT GetShiftTemplate(CFSTR* pSHCShiftTemplate)
```

- `pSHCShiftTemplate`
  [out]: The shift template

"CreateAction" method

Adds an "ISHCAction" instance to the "ISHCShift" instance.

```
CRFESULT CreateAction(ISHCActionTemplate* pActionTemplate,
CFTIMESPAN64 offset, ISHCAction** pShcAction)
```

- `pActionTemplate`
  [in]: The action template on which the new action is to be based.
- `offset`
  [in]: The offset of the action in relation to the start time of the "ISHCShift" instance. Positive and negative value allowed.
- `pShcAction`
  [out]: The new action

"DeleteAction" method

Deletes an "ISHCAction" instance of the "ISHCShift" instance.

```
CFRESULT DeleteAction(ISHCAction* pShcAction)
```

- `pShcAction`
  [in]: Reference to the action to be deleted.

"GetAction" method

Supplies an "ISHCActionEnumerator" instance via which you access the "ISHCAction" instances of the "ISHCShift" instance.

```
CRFESULT GetActions(ISHCActionEnumerator** ppSHCActionEnumerator)
```

- `ppSHCActionEnumerator`
  [out]: The enumerator

"GetIsCustomized" method

Supplies the information on whether the "ISHCShift" instance was edited by users.

```
CRFESULT GetIsCustomized)(CFBOOL* pIsCustomized)
```

- `pIsCustomized`
  [out]:
  - 0: Was not processed
  - 1: Was processed

"GetDeltaKind" method

Supplies the delta type of the "ISHCShift" instance.

```
CRFESULT GetDeltaKind(CFENUM* pDeltaKind)
```

- `pDeltaKind`
  [out]: Points to the enumeration "ShcDeltaType", which can contain the following values:
  - Added (0)
  - Modified (1)
  - Deleted (2)

"GetShiftId" method

Supplies the ID of the "ISHCShift" instance.

```
CFRESULT GetShiftId(uint32_t* pShiftId)
```

- `pShift`
  [out]: The ID

"SetShiftId" method

Sets the ID of the "ISHCShift" instance.

```
SetShiftId(uint32_t ShiftId)
```

- `ShiftId`
  [in]: The new ID

19.10.7.18 ISHCShiftEnumerator (RT Uni)

Description

The C++ interface "ISHCShiftEnumerator" specifies methods for handling the enumeration of shifts of an "ISHCDay" instance or "ISHCDayTemplate" instance.

The enumeration is returned by the "GetShifts" method of these instances.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCShift** ppItem)
```

- `ppItem`
  [out]: The current shift

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of shifts

Example

Copy code

```
void printShift(const ISHCShiftEnumeratorPtr& p_pShcShiftEnum)
{
    std::cout << endl << "***********Print Shift*********" << std::endl << endl;if
(p_pShcShiftEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCout = 0;
        p_pShcShiftEnum->Count(&nCout);
        for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
        {
            cout << endl;
            if (CF_SUCCEEDED(p_pShcShiftEnum->MoveNext()))
            {
                ISHCShiftPtr pShcshift;
                p_pShcShiftEnum->Current(&pShcshift);
                if (pShcshift != nullptr)
                {
                    CCfString strshiftTemplateName;
                    hr = pShcshift->GetShiftTemplate(&strshiftTemplateName);
                    if (CF_SUCCEEDED(hr) && (!strCategoryName.IsEmpty()))
                    {
                        std::cout << "ShiftTemplateName= " <<
strshiftTemplateName.ToUTF8().c_str() << std::endl;
                    }
                    CCfTimeSpan64 tsDuration;
                    hr = pShcshift->GetDuration(&tsDuration);
                    if (CF_SUCCEEDED(hr))
                    {
                        CCfString strDuration = tsDuration.GetTimeSpanString();
                        std::cout << "Durations= " << strDuration.ToUTF8().c_str() <<
std::endl;
                    }
                    CFBOOL bIsCustomised;
                    hr = pShcshift->GetIsCustomized(&bIsCustomised);
                    if (CF_SUCCEEDED(hr))
                    {
                        std::cout << "Is Customised= " << (uint32_t)bIsCustomised <<
std::endl;
                    }
                    CFENUM ndeltaKind;
                    hr = pShcshift->GetDeltaKind(&ndeltaKind);
                    if (CF_SUCCEEDED(hr))
                    {
                        std::cout << "deltaKind = " << ndeltaKind << std::endl;
                    }
                    uint32_t nShiftId;
                    hr = pShcshift->GetShiftId(&nShiftId);
                    if (CF_SUCCEEDED(hr))
                    {
                        std::cout << "ShiftId = " << nShiftId << std::endl;
                    }
                    ICfMapIDToVariantPtr pComments;
                    hr = pShcshift->GetComments(&pComments);
                    if (pComments != nullptr && CF_SUCCEEDED(hr))
```

Copy code

```
                    {
                        std::cout << "comments:-" << std::endl << std::endl;
                        uint32_t nCount = 0;  pComments->Count(&nCount);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pComments->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strComments;
                            pComments->ValueAt(nLanguageID, &strComments);
                            std::cout << "LangauageID =" << nLanguageID << " Comments=" << 
CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
                        }
                    }
                    ISHCTimeSliceEnumeratorPtr pShiftTimesliceEnum;
                    hr = pShcshift->GetTimeSlices(&pShiftTimesliceEnum);
                    if (CF_SUCCEEDED(hr))  printTimeSlice(pShiftTimesliceEnum);
                }
            }
        }
    }
}
```

19.10.7.19 ISHCAction (RT Uni)

Description

The C++ interface "ISHCAction" specifies the methods of an action.

The interface inherits from the "ICfUnknown" interface.

Members

"GetOffset" method

Returns the offset of the "ISHCAction" instance in relation to the starting point of its "ISHCShift" instance.

`CFRESULT GetOffset(CFTIMESPAN64* pOffsetType)`

- `pOffset`
  [out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCAction" instance in relation to the starting point of its "ISHCShift" instance.

`CFRESULT GetOffset(CFTIMESPAN64 OffsetType)`

- `OffsetType`
  [in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetActionTemplate" method

Supplies the "ISHCActionTemplate" instance of the "ISHCAction" instance.

```
GetActionTemplate(CFSTR* p_strActionTemplate)
```

- `p_strActionTemplate`
  [out]: The action template

"GetIsCustomized" method

Supplies the information on whether the "ISHCAction" instance was edited by users.

```
CRFESULT GetIsCustomized(CFBOOL* pIsCustomized)
```

- `pIsCustomized`
  [out]:
  - 0: Was not processed
  - 1: Was processed

"GetElements" method

Supplies an "ISHCActionElementEnumerator" instance via which you access the action elements of the "ISHCAction" instance.

```
GetElements(ISHCActionElementEnumerator**
ppSHCActionElementEnumerator)
```

- `ppSHCActionElementEnumerator`
  [out]: The enumerator

19.10.7.20 ISHCActionEnumerator (RT Uni)

Description

The C++ interface "ISHCActionEnumerator" specifies methods for handling the enumeration of actions of an "ISHCShift" instance or "ISHCShiftTemplate" instance.

The enumeration is returned by the "GetActions" method of these instances.

The interface inherits from the "IcfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCAction** ppItem)
```

- `ppItem`
  [out]: The current action

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of actions

Example

Copy code

```
void PrintAction(const ISHCActionEnumeratorPtr& pShcActionEnum)
{
    cout << endl << "******PrintAction*******" << endl << endl;if (pShcActionEnum !=
nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        pShcActionEnum->Count(&nCount);
        if (nCount > 0)
        {
            for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
            {
                cout << endl; if (CF_SUCCEEDED(pShcActionEnum->MoveNext()))
                {
                    ISHCActionPtr pShcAction;
                    pShcActionEnum->Current(&pShcAction);
                    if (pShcAction != nullptr)
                    {
                        CCfString strActionTemplateName;
                        hr = pShcAction->GetActionTemplate(&strActionTemplateName);
                        if (CF_SUCCEEDED(hr))
                        {
                            std::cout << "ActionTemplateName = " <<
strActionTemplateName.ToUTF8().c_str() << std::endl;
                        }
                        CFBOOL isCustomize;
                        hr = pShcAction->GetIsCustomized(&isCustomize);
                        if (CF_SUCCEEDED(hr))
                        {
                            cout << "IsCustomize= " << (uint32_t)isCustomize << endl;
                        }
                        CCfTimeSpan64 offset;
                        hr = pShcAction->GetOffset(&offset);
                        if (CF_SUCCEEDED(hr))
                        {
                          cout << "Offset=" << offset.GetTimeSpanString().ToUTF8().c_str()
<< endl;
                        }
                        ISHCActionElementEnumeratorPtr pShcActionElementEnum;
                        hr = pShcAction->GetElements(&pShcActionElementEnum);
                        if (CF_SUCCEEDED(hr))
                        {
                            PrintActionElement(pShcActionElementEnum);
                        }
                    }
                }
            }
        }
    }
}
```

19.10.7.21 ISHCActionElement (RT Uni)

Description

The C++ interface "ISHCActionElement" specifies the methods of the action elements of an "ISHCAction" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetElementType" method

Supplies the type of the "ISHCActionElement" instance.

```
CRFESULT GetElementType(CFENUM* pElementType)
```

- `pElementType`
  [out]: Points to the enumeration "ShcActionElementType", which can contain the following values:
  - `Tag` (0)
    The action element controls a tag.

"GetEnabled" method

Supplies the information on whether the "ISHCActionElement" instance is activated.

```
CRFESULT GetEnabled(CFBOOL* pEnabled)
```

- `pEnabled`
  [out]:
  - 0: Deactivated
  - 1: Activated

"SetEnabled" method

Sets whether the "ISHCActionElement" instance is activated.

```
CRFESULT SetEnabled(CFBOOL Enabled)
```

- `Enabled`
  [in]:
  - 0: Deactivated
  - 1: Activated

"GetOffset" method

Returns the offset of the "ISHCActionElement" instance in relation to the anchor point of its "ISHCAction" instance.

```
CFRESULT GetOffset(CFTIMESPAN64* pOffset)
```

- `pOffset`
  [out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCActionElement" instance in relation to the anchor point of its "ISHCAction" instance.

```
CFRESULT SetOffset(CFTIMESPAN64 offset)
```

- `offset`
  [in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetValue" method

Supplies the value of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT GetValue(CFVARIANT* pValue)
```

- `pValue`
  [out]: The tag value

"SetValue" method

Sets the value of the tag controlled by the "ISHCActionElement" instance.

```
CRFESULT SetValue(CFVARIANT value)
```

- `value`
  [in]: The new tag value

"GetElementName" method

Supplies the name of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT GetElementName(CFSTR* p_strActionElement)
```

- `p_strActionElement`
  [out]: The tag name

"SetElementName" method

Sets the name of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT SetElementName(CFSTR ActionElement)
```

- `ActionElement`
  [in]: The tag name

19.10.7.22 ISHCActionElementEnumerator (RT Uni)

Description

The C++ interface "ISHCActionElementEnumerator" specifies methods for handling the enumeration of action elements of an "ISHCAction" instance.

The enumeration is returned by the "GetElements" method of an "ISHCAction" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCActionElement** ppItem)
```

- `ppItem`
  [out]: The current action element

"Reset" method

Reset the current position in enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of action elements

Example

Copy code

```
void PrintActionElement(const ISHCActionElementEnumeratorPtr& pActionElementEnum)
{
    cout << endl << "******PrintActionElement*******" << endl << endl;
    if (pActionElementEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        pActionElementEnum->Count(&nCount);
        if (nCount > 0)
        {
            for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
            {
                cout << endl;
                if (CF_SUCCEEDED(pActionElementEnum->MoveNext()))
                {
                    ISHCActionElementPtr pShcActionElement;
                    pActionElementEnum->Current(&pShcActionElement);
                    if (pShcActionElement != nullptr)
                    {
                        CCfString strElementName;
                        hr = pShcActionElement->GetElementName(&strElementName);
                        if (CF_SUCCEEDED(hr))
                        {
                          cout << "ActionElementName= " << strElementName.ToUTF8().c_str()
<< endl;
                        }
                        CFENUM nType;
                        hr = pShcActionElement->GetElementType(&nType);
                        if (CF_SUCCEEDED(hr))
                        {
                            cout << "Element Type= " << nType << endl;
                        }
                        CFBOOL bIsEnable;
                        hr = pShcActionElement->GetEnabled(&bIsEnable);
                        if (CF_SUCCEEDED(hr))
                        {
                            cout << "Is Enable = " << (uint32_t)bIsEnable << endl;
                        }
                        CCfTimeSpan64 offset;
                        hr = pShcActionElement->GetOffset(&offset);
                        if (CF_SUCCEEDED(hr))
                        {
                        cout << "Offset = " << offset.GetTimeSpanString().ToUTF8().c_str()
<< endl;
                        }
                        CCfVariant vtValue;
                        hr = pShcActionElement->GetValue(&vtValue);
                        if (CF_SUCCEEDED(hr))
                        {
                            cout << "value = " << vtValue.uint32 << endl;
                        }
                    }
                }
            }
```

Copy code

```
        }
    }
}
```

19.10.7.23 ISHCActionTemplate (RT Uni)

Description

The C++ interface "ISHCActionTemplate" specifies the methods of an action template.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCActionTemplate" instance.

`CFRESULT GetName(CFSTR * pvarRet)`

- `pvarRet`
  [out]: The name of the action template

"SetName" method

Sets the name of the "ISHCActionTemplate" instance

`CFRESULT SetName(CFSTR  value)`

- `value`
  [in]: The name

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCActionTemplate" instance and their language code IDs.

`CFRESULT GetDisplayNames)(ICfMapIDToVariant** ppDisplayNames)`

- `ppDisplayNames`
  [out]: The map with String/String pairs (language code ID for display name).

"SetDisplayNames" method

Adds a display name and its language code ID to the map with the display name of the "ISHCActionTemplate" instance.

`CFRESULT SetDisplayNames)(CFLCID languageId, CFSTR pDisplayName)`

- `languageID`
  [in]: The language code ID of the display name
- `pDisplayName`
  [in]: The display name

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCActionTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** value)
```

- `value`
  [out]: The map with String/String pairs (language code ID for description).

"SetDescriptions" method

Adds a description and its language code ID to the map with the descriptions of the "ISHCActionTemplate" instance.

```
CFRESULT SetDescription(CFLCID languageId, CFSTR pDescriptions)
```

- `languageID`
  [in]: The language code ID of the description
- `pDescriptions`
  [in]: The description text

"GetIsDeleted" method

Supplies the information on whether the "ISHCActionTemplate" instance was deleted by users.

```
CFRESULT GetIsDeleted(CFBOOL* pIsDeleted)
```

- `pIsDeleted`
  [out]:
  - 0: Was not deleted
  - 1: Was deleted

"GetElements" method

Supplies an "ISHCActionTemplateElementEnumerator" instance. The enumerator provides you with access to the "ISHCActionTemplateElement" instances of the "ISHCActionTemplate" instance.

```
CRFESULT GetElements(ISHCActionTemplateElementEnumerator**
ppISHCActionTemplateElementEnumerator)
```

- `ppISHCActionTemplateElementEnumerator`
  [out]: The enumerator

"CreateElement" method

Adds an "ISHCActionTemplateElement" instance to the "ISHCActionTemplate" instance.

```
CRFESULT CreateElement(ISHCActionTemplateElement*
pSHCActionTemplateElement)
```

- `pSHCActionTemplateElement`
  [in]: Reference to the new action element

"DeleteElement" method

Deletes an action element of the "ISHCActionTemplate" instance.

```
CRFESULT DeleteElement(ISHCActionTemplateElement*
pSHCActionTemplateElement)
```

- `pSHCActionTemplateElement`
  [in]: Reference to the action element to be deleted

19.10.7.24 ISHCActionTemplateEnumerator (RT Uni)

Description

The C++ interface "ISHCActionTemplateEnumerator" specifies methods for handling the enumeration of the action templates of an "ISHCCalendar" instance.

The enumeration is returned by the "Read" method of an "ISHCActionTemplatesProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCActionTemplate** ppItem)
```

- `ppItem`
  [out]: The current action template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of action templates

Example

Copy code

```
void PrintActionTemplate(const ISHCActionTemplateEnumeratorPtr& p_pShcActionTemplateEnum)
{
    cout << endl << "******PrintActionTemplate*******" << endl << endl;
    if (p_pShcActionTemplateEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        p_pShcActionTemplateEnum->Count(&nCount);
        for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
        {
            cout << endl;
            if (CF_SUCCEEDED(p_pShcActionTemplateEnum->MoveNext()))
            {
                ISHCActionTemplatePtr pShcActionTemplate;
                p_pShcActionTemplateEnum->Current(&pShcActionTemplate);
                if (pShcActionTemplate != nullptr)
                {
                    CCfString strActionTemplateName;
                    hr = pShcActionTemplate->GetName(&strActionTemplateName);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "ActionTemplateName=" <<
strActionTemplateName.ToUTF8().c_str() << endl;
                    }
                    CFBOOL bIsDeleted;
                    hr = pShcActionTemplate->GetIsDeleted(&bIsDeleted);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "IsDeleted=" << (uint32_t)bIsDeleted << endl;
                    }
                    ICfMapIDToVariantPtr pDescriptions;
                    hr = pShcActionTemplate->GetDescriptions(&pDescriptions);
                    if (pDescriptions != nullptr && CF_SUCCEEDED(hr))
                    {
                        std::cout << "Descriptions=" << std::endl << std::endl;
                        uint32_t nCount1 = 0;
                        pDescriptions->Count(&nCount1);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
                        {
                            int32_t nLanguageID;
                            pDescriptions->KeyAt(nIndex1, &nLanguageID);
                            CCfVariant strDescription;
                            pDescriptions->ValueAt(nLanguageID, &strDescription);
                            std::cout << "LangauageID =" << nLanguageID << " Description="
<< CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
                        }
                    }
                    ICfMapIDToVariantPtr pDisplayNames;
                    hr = pShcActionTemplate->GetDisplayNames(&pDisplayNames);
                    if (pDisplayNames != nullptr && CF_SUCCEEDED(hr))
                    {
                        std::cout << "DisplayNames::" << std::endl << std::endl;  uint32_t
nCount1 = 0;  pDisplayNames->Count(&nCount1);
                        for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
```

Copy code

```
                          {
                              int32_t nLanguageID;
                              pDisplayNames->KeyAt(nIndex1, &nLanguageID);
                              CCfVariant strDIsplayname;
                              pDisplayNames->ValueAt(nLanguageID, &strDIsplayname);
                              std::cout << "LangauageID =" << nLanguageID << " DisplayName="
<< CCfSmartString(strDIsplayname).ToUTF8().c_str() << std::endl;
                          }
                      }
                      ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
                      hr = pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);
                      if (CF_SUCCEEDED(hr))
PrintActionTemplateElement(pShcActionTemplateElementEnum);
                }
            }
        }
    }
}
```

19.10.7.25 ISHCActionTemplatesProvider (RT Uni)

Description

The C++ interface "ISHCActionTemplatesProvider" provides you with access to an "ISHCActionTemplateEnumerator" instance which contains an enumeration with the action templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete action templates. The provider is returned by the "GetActionTemplatesProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCActionTemplateEnumerator" instance which has access to an enumeration with the "ISHCActionTemplate" instances of the calendar.

```
CRFESULT Browse(CFBOOL includeDeleted,OUT
ISHCActionTemplateEnumerator** ppISHCActionTemplateEnumerator)
```

- `includeDeleted`
  Saves whether the enumerator also has access to the deleted action templates.
- `ppISHCActionTemplateEnumerator`
  [out]: The enumerator

Example:

```
ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
CFRESULT hr = pShcActionTemplateProvider->Browse(CF_FALSE,
&pShcActionTemplateEnum);
```

"Create" method

Adds new action templates to the enumeration with the "ISHCActionTemplate" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pActionTemplates)
```

- `pActionTemplates`
  [in]: An array with the action templates to be added.

Example:

Copy code

```
void CreateActionTemplateWithActionTemplateElement()
{
    if (nullptr != pCalendar && nullptr != pShcActionTemplateProvider)
    {
        CFRESULT hr = CF_ERROR;
        ICfUnknownPtr pUnk;
        hr = pCalendar->GetObject(ODK_SHC_ACTION_TEMPLATE, &pUnk);
        CCfArrayIUnknown ArrayTemplate;
        ICfArrayIUnknownPtr pArrayTemplate;
        ISHCActionTemplatePtr pShcActionTemplate = (ISHCActionTemplatePtr)pUnk;
        if (pShcActionTemplate != nullptr)
        {
            hr = pShcActionTemplate->SetName(CCfString(L"ActionTemplateName"));
            if (CF_FAILED(hr))
            {
                cout << "set name failed" << endl;
            }
            hr = pShcActionTemplate->SetDisplayName(1033, 
CCfString(L"ActionTemplateDisplayName"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetDisplayName" << std::endl;
            }
            hr = pShcActionTemplate->SetDescription(1033, 
CCfString(L"ActionTemplateDescription"));
            if (CF_FAILED(hr))
            {
                std::cout << "failed to SetDescription" << std::endl;
            }
            ArrayTemplate.Append(pShcActionTemplate);
        }
        ArrayTemplate.DetachEnumerator(&pArrayTemplate);
        hr = pShcActionTemplateProvider->Create(pArrayTemplate);
        if (CF_SUCCEEDED(hr))
        {
            hr = pCalendar->GetObject(ODK_SHC_ACTION_TEMPLATE_ELEMENT, &pUnk);
            ISHCActionTemplateElementPtr pShcActionTemplateElement = 
(ISHCActionTemplateElementPtr)pUnk;
            if (nullptr != pShcActionTemplateElement)
            {
                hr = pShcActionTemplateElement-
>SetElementName(CCfString(L"HMI_RT_1::Unit1.Member_1"));
                if (CF_FAILED(hr))
                {
                    std::cout << "failed to SetElementName" << std::endl;
                }
                CFTIMESPAN64 Offset = Get1Hour();
                hr = pShcActionTemplateElement->SetOffset(Offset);
                if (CF_FAILED(hr))
                {
                    std::cout << "failed to SetOffset" << std::endl;
                }
                uint32_t value = 1;
                hr = pShcActionTemplateElement->SetValue(CCfVariant(value));
                if (CF_FAILED(hr))
```

Copy code

```
                {
                    std::cout << "failed to SetValue" << std::endl;
                }
            }
            hr = pShcActionTemplate->CreateElement(pShcActionTemplateElement);
            if (CF_FAILED(hr))
            {
                std::cout << "failed to CreateElement" << std::endl;
            }
        }
    }
}
```

"Update" method

Updates "ISHCActionTemplate" instances of the enumeration.

`CRFESULTUpdate(ICfArrayIUnknown* pActionTemplates)`

- `pActionTemplates`
  [in]: An array with the action templates to be updated.

Example:

Copy code

```
void UpdateActionTemplateWithActionTemplateElement()
{
    if (pShcActionTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
        hr = pShcActionTemplateProvider->Browse(CF_FALSE, &pShcActionTemplateEnum);
        if (CF_SUCCEEDED(hr) && pShcActionTemplateEnum != nullptr)
        {
            hr = pShcActionTemplateEnum->MoveNext();
            if (CF_SUCCEEDED(hr))
            {
                CCfArrayIUnknown ArrayActionTemplate;
                ICfArrayIUnknownPtr pArrayActionTemplate;
                ISHCActionTemplatePtr pShcActionTemplate;
                pShcActionTemplateEnum->Current(&pShcActionTemplate);
                if (pShcActionTemplate != nullptr)
                {
                    hr = pShcActionTemplate->SetName(CCfString(L"UpdatedActionTemplate"));
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to SetName" << std::endl;
                    }
                    hr = pShcActionTemplate->SetDisplayName(1033,
CCfString(L"UpdatedActionTemplateDisplayName"));
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to SetDisplayName" << std::endl;
                    }
                    ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
                    pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);
                    if (pShcActionTemplateElementEnum != nullptr)
                    {
                        hr = pShcActionTemplateElementEnum->MoveNext();
                        if (CF_SUCCEEDED(hr))
                        {
                            ISHCActionTemplateElementPtr pShcActionTemplateElement;
                            pShcActionTemplateElementEnum-
>Current(&pShcActionTemplateElement);
                            if (pShcActionTemplateElement != nullptr)
                            {
                                CCfTimeSpan64 offset = Get1Hour() * 2;
                                hr = pShcActionTemplateElement->SetOffset(offset);
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to SetOffset" << std::endl;
                                }
                                uint32_t nValue = 2;
                             hr = pShcActionTemplateElement->SetValue(CCfVariant(nValue));
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to SetValue" << std::endl;
                                }
                            }
                        }
```

Copy code

```
                    }
                    ArrayActionTemplate.Append(pShcActionTemplate);
                    ArrayActionTemplate.DetachEnumerator(&pArrayActionTemplate);
                    hr = pShcActionTemplateProvider->Update(pArrayActionTemplate);
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to Update" << std::endl;
                    }
                }
            }
        }
    }
}
```

"Delete" method

Deletes "ISHCActionTemplate" instances of the calendar from the enumeration.

`CRFESULT Delete(ICfArrayIUnknown* pActionTemplates)`

- `pActionTemplates`
  [in]: An array with the action templates to be deleted.

Example:

Copy code

```
void DeleteActionTemplateWithActionTemplateElement()
{
    if (pShcActionTemplateProvider != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
        hr = pShcActionTemplateProvider->Browse(CF_FALSE, &pShcActionTemplateEnum);
        if (CF_SUCCEEDED(hr) && pShcActionTemplateEnum != nullptr)
        {
            hr = pShcActionTemplateEnum->MoveNext();
            if (CF_SUCCEEDED(hr))
            {
                CCfArrayIUnknown ArrayActionTemplate;
                ICfArrayIUnknownPtr pArrayActionTemplate;
                ISHCActionTemplatePtr pShcActionTemplate;
                pShcActionTemplateEnum->Current(&pShcActionTemplate);
                if (pShcActionTemplate != nullptr)
                {
                    ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
                    pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);
                    if (pShcActionTemplateElementEnum != nullptr)
                    {
                        hr = pShcActionTemplateElementEnum->MoveNext();
                        if (CF_SUCCEEDED(hr))
                        {
                            ISHCActionTemplateElementPtr pShcActionTemplateElement;
                            pShcActionTemplateElementEnum-
>Current(&pShcActionTemplateElement);
                            if (pShcActionTemplateElement != nullptr)
                            {
                                hr = pShcActionTemplate-
>DeleteElement(pShcActionTemplateElement);
                                if (CF_FAILED(hr))
                                {
                                    std::cout << "failed to DeleteElement" << std::endl;
                                }
                            }
                        }
                    }
                    ArrayActionTemplate.Append(pShcActionTemplate);
                    ArrayActionTemplate.DetachEnumerator(&pArrayActionTemplate);
                    hr = pShcActionTemplateProvider->Delete(pArrayActionTemplate);
                    if (CF_FAILED(hr))
                    {
                        std::cout << "failed to Delete" << std::endl;
                    }
                }
            }
        }
    }
}
```

19.10.7.26 ISHCActionTemplateElement (RT Uni)

Description

The C++ interface "ISHCActionTemplateElement" specifies the methods of an action template.

The interface inherits from the "ICfUnknown" interface.

Members

"GetElementType" method

Supplies the type of the "ISHCActionTemplateElement" instance.

```
CRFESULT GetElementType(CFENUM* pElementType)
```

- `pElementType`
  [out]: Points to the enumeration "ShcActionElementType", which can contain the following values:
  - `Tag` (0)
    The action element controls a tag.

"GetOffset" method

Returns the offset of the "ISHCActionTemplateElement" instance in relation to the anchor point of its "ISHCActionTemplate" instance.

```
CFRESULT GetOffset(CFTIMESPAN64* pOffset)
```

- `pOffset`
  [out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCActionTemplateElement" instance in relation to the anchor point of its "ISHCActionTemplate" instance.

```
CFRESULT SetOffset(CFTIMESPAN64 offset)
```

- `offset`
  [in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetElementName" method

Supplies the name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT GetElementName(CFSTR* pElementName)
```

- `pElementName`
  [out]: The tag name

"SetElementName" method

Sets the name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT SetElementName(CFSTR pElementName)
```

- `pElementName`
  [in]: The tag name

"GetValue" method

Supplies the value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT GetValue(CFVARIANT* pValue)
```

- `pValue`
  [out]: The tag value

"SetValue" method

Sets the value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CRFESULT SetValue(CFVARIANT pValue)
```

- `pValue`
  [in]: The new tag value

19.10.7.27 ISHCActionTemplateElementEnumerator (RT Uni)

Description

The C++ interface "ISHCActionTemplateElementEnumerator" specifies methods for handling the enumeration of "ISHCActionTemplateElement" instances of an "ISHCActionTemplate" instance.

The enumeration is returned by the "GetElements" method of an "ISHCActionTemplate" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCActionTemplateElement** ppItem)
```

- `ppItem`
  [out]: The current action element of the action template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of "ISHCActionTemplateElement" instances

Example

Copy code

```
void PrintActionTemplateElement(const ISHCActionTemplateElementEnumeratorPtr&
p_pShcActionTemplateElementEnum)
{
    cout << endl << "****PrintActionTemplateElement***********" << endl << endl;
    if (p_pShcActionTemplateElementEnum != nullptr)
    {
        CFRESULT hr = CF_ERROR;
        uint32_t nCount = 0;
        p_pShcActionTemplateElementEnum->Count(&nCount);
        for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
        {
            cout << endl; if (CF_SUCCEEDED(p_pShcActionTemplateElementEnum->MoveNext()))
            {
                ISHCActionTemplateElementPtr pShcActionTemplateElement;
                p_pShcActionTemplateElementEnum->Current(&pShcActionTemplateElement);
                if (pShcActionTemplateElement != nullptr)
                {
                    CCfString strElementName;
                    hr = pShcActionTemplateElement->GetElementName(&strElementName);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "TemplateElementName= " << strElementName.ToUTF8().c_str()
<< endl;
                    }
                    CFENUM type;
                    hr = pShcActionTemplateElement->GetElementType(&Type);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "ElementType=" << Type << endl;
                    }
                    CCfTimeSpan64 tsOffset;
                    hr = pShcActionTemplateElement->GetOffset(&tsOffset);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "Element Offset=" <<
tsOffset.GetTimeSpanString().ToUTF8().c_str() << endl;
                    }
                    CCfVariant vtValue;
                    hr = pShcActionTemplateElement->GetValue(&vtValue);
                    if (CF_SUCCEEDED(hr))
                    {
                        cout << "Value=" << vtValue.uint32 << endl;
                    }
                }
            }
        }
    }
}
```

Working with plant objects and plant views 20

20.1 Basics

20.1.1 Introduction

Object-oriented configuration

The option of object-oriented configuration is available to you in WinCC Unified Scada RT. Define reusable plant object types and assign the associated plant object instances in hierarchical plant views.

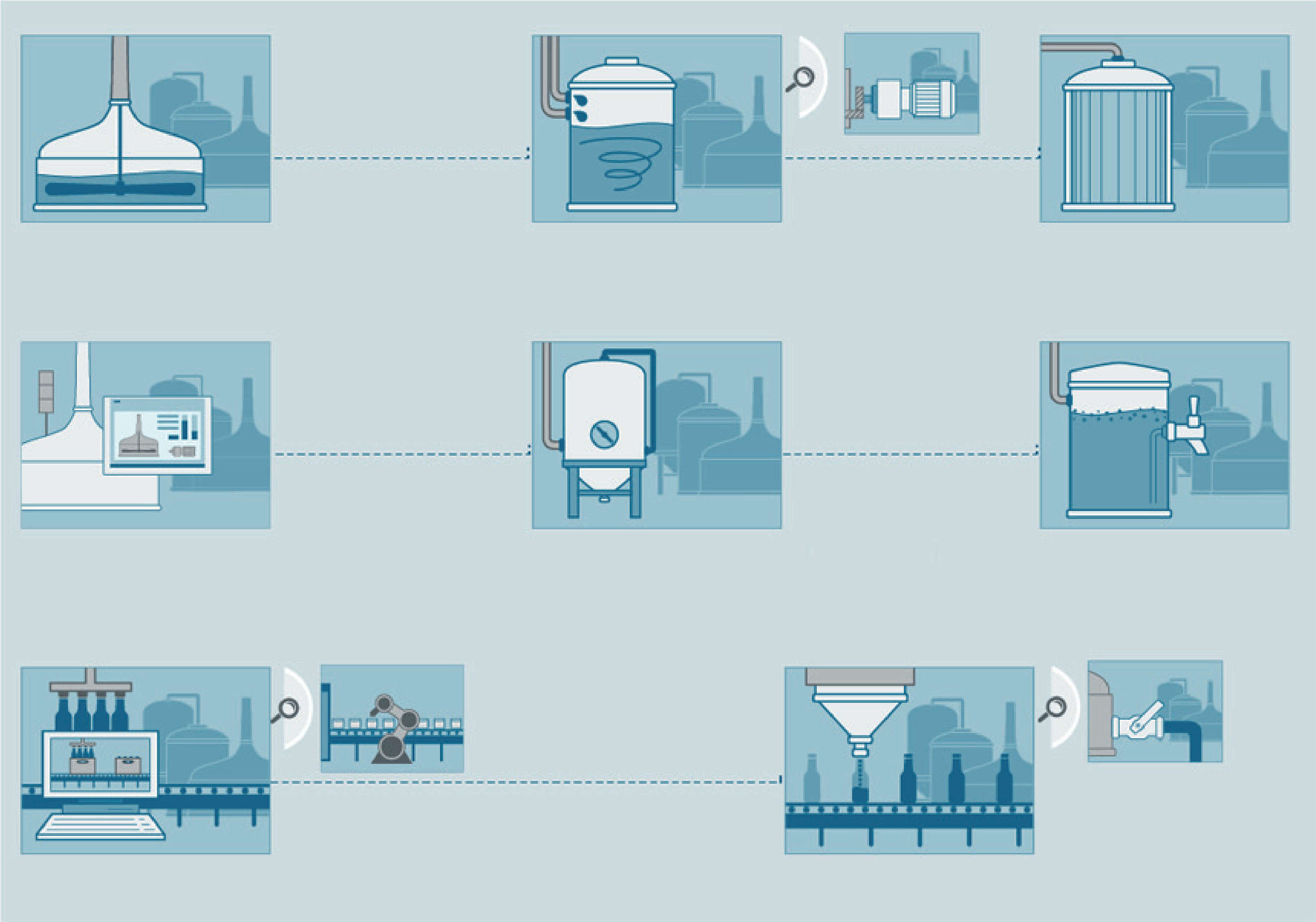
In this way, you can model the plant view of your machine or unit/plant, for example, based on user-defined or standardized technological objects.

The plant structure is created from individual objects, each of which represents a specific component or unit. You configure each object in the context of the operator control and monitoring solution.

In plant object types, you combine all required configuration elements for visualization, such as faceplates, tags, alarms, scripts, etc. Changes to the plant object type automatically affect all instances. This translates into real time savings, especially for plants with a high degree of standardization.

You can start object-oriented plant modeling based on the engineering data, if necessary, and can derive the configuration of the HMI devices and automation systems from this.

Break the machine or unit/plant up into reusable technological units and arrange them hierarchically in a technological plant view according to the plant structure.



The following options are available to you in technology-oriented and object-oriented configuration:

- Creating various hierarchical plant views: technological view, building view, independent of the HMI device that is used.
- Configuration of plant objects and plant object types with data elements for mapping the actual plant configuration
- Access to plant objects (data elements, HMI alarms, logs, screens, etc.)
- Generation of the screen hierarchy
- Expansion of configured plant objects and types using Plant Intelligence options, e.g. WinCC Unified Performance Insight or WinCC Unified Sequence Execution

See also

Applications (Page 1499)

Overview (Page 1509)

Type/instance concept in object-oriented configuration (Page 1501)

Requirements (Page 1501)

20.1.2 Applications

Overview

You use technology-oriented and object-oriented configuration for automation solutions to increase overall effectiveness.

In particular, in plants with high level of standardization, the object-oriented approach increases the configuration efficiency through the reuse of objects, the capability of changing objects centrally, and the integration of manufacturing execution system functionalities such as the calculation of individual key indicators for separate machines.

Technology-oriented and object-oriented configuration supports you in the following operating phases:

- Planning phase: Efficient plant configuration and simple plant expansion through integration of part models from other projects
- Plant maintenance: Transparency through mapping of the exact plant structure
- Quality management: Continuous optimization of your projects

Advantages

- Creation and generation of modular projects based on standardized plant objects
- Reduced engineering workloads and fewer inconsistencies with a shared model in Engineering and Runtime
- Simple plant expansion through integration of part models from other projects
- Creating the screen hierarchy
- Transparent access to all objects and their properties and methods, independent of device assignment
- Targeted corrective measures through transparent relationships of individual plant objects
- Intelligent use of information from the entire manufacturing environment in combination with Plant Intelligence options

Operation in runtime

Depending on your configuration, the following possibilities are available to you in runtime:

- Display hierarchy path of alarm source
- Filter alarm control by plant objects
- Display alarm status of a line and navigation to the alarm source
- Display the most frequently occurring alarms, filtered by plant object or plant object type
- Area-based access protection
- Screen navigation via the plant model
- Determine the energy consumption of a line and compare with another line
- Analysis based on plant objects

The plant hierarchy is also available for scripting in runtime.

Requirements on the configuration engineer

The following prior knowledge is required for using technology-oriented and object-oriented configuration:

- You have experience performing configuration in STEP 7 and WinCC.

See also

Introduction (Page 1497)

Type/instance concept in object-oriented configuration (Page 1501)

Configuration concept (Page 1504)

20.1.3 Requirements

Software requirements

You acquire the following products to use technology- and object-oriented configuration:

- TIA Portal version V15.1 and higher
- WinCC Unified Scada RT

The "Plant objects" area is visible under Project tree after the installation of WinCC Unified Scada RT.

Supported devices

The following SIMATIC S7 controllers are supported:

- SIMATIC S7-1500

See also

Creating plant objects (Page 1517)

Creating plant object types (Page 1516)

Introduction (Page 1497)

20.1.4 Type/instance concept in object-oriented configuration

Introduction

The object-oriented approach of WinCC based on the type-instance concept. In types, you create central properties for an object. The instances represent local places of use for the types.

Plant objects are instances of a plant object type.

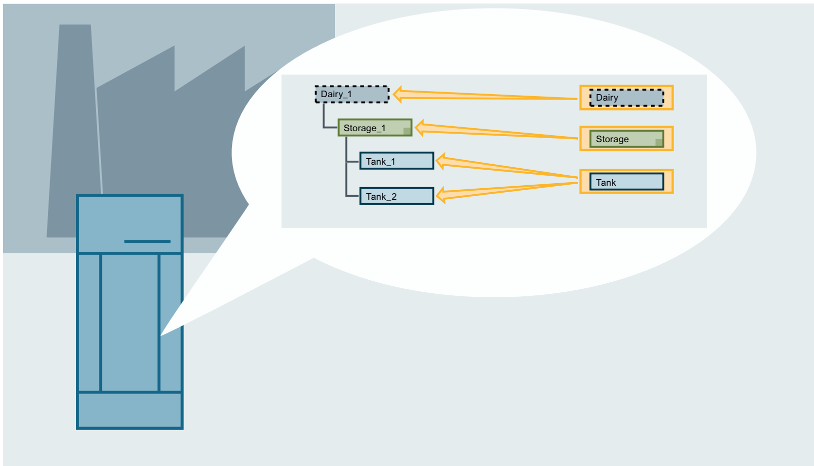
The plant object type is the central, object-oriented definition of a reusable plant component (such as conveyor robot). As instances of the plant object type, the plant objects generally map concrete, physically existing plant components (e.g. conveyor robot_A and conveyor robot_B).

If you change a property of a plant object type, the property is saved centrally and also changed in all instances.

Effect of the type instance concept on object-oriented configuration

The use of a type is called an instance. The common plant model is generated from instances. Each instance inherits all the properties of the type. The Common Plant Model with high level of standardization is characterized by the use of many instances of few types in the model.

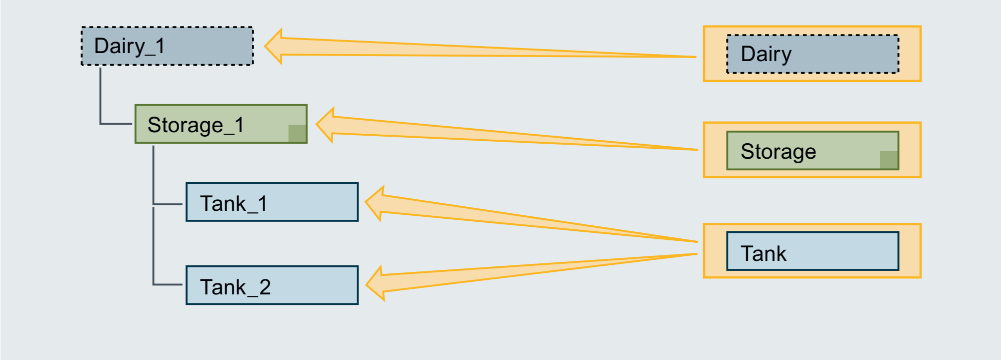
The general types of the plant units are configured and these are reused when required in the configuration and adapted to the specific plant objects. The plant structure hereby specifies the addressing of the plant objects.



In the object-oriented approach of WinCC the following correspondences apply:

- Type = Plant object type
- Instance = Plant object

The following figure shows the basic structure of a plant model:



Plant objects and plant object types

A plant object is a technological unit. In a plant object, the components are stored in a typical form which is required for modeling a plant.

A valid plant object must be created from a plant object type. The plant structure is created from plant objects.

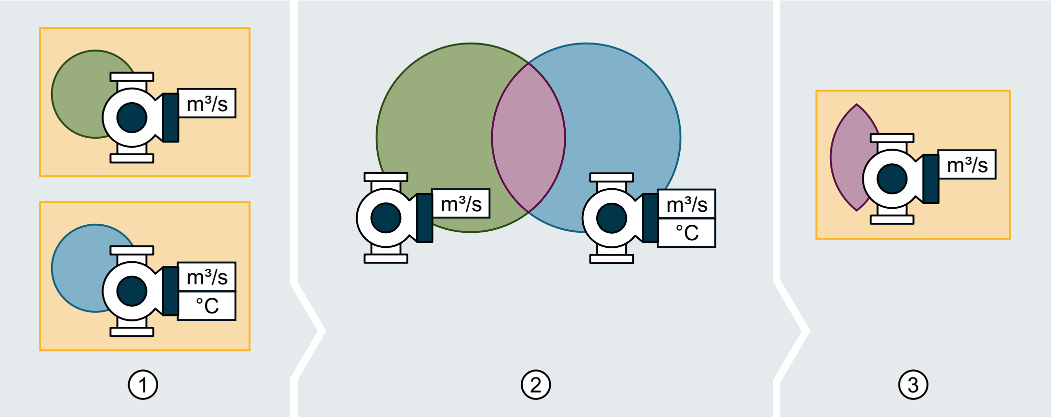
The definition of a plant object type consists of the data structure and context information:

- Alarms
- Logging
- Visualization
- Data member (internal and external)
- Facets (e.g. performance indicators)

Type definition in terms of high reuse

A plant object type is used to describe a plant object independently of its use in the Common Plant Model. Define a plant object type as generally as possible and as specifically as required. Take into account the following aspects:

- Identical data structure in PLC (function block or PLC UDT)
  Example: Pumps that have different performance ranges are installed in a plant. The data structure in the PLC is identical for each pump. Map these pumps with a common plant object type. At each instance you configure the specific value ranges for the respective performance ranges.
  A pump function block (standard FB for a pump) is available on the control side. The customer defines the plant object type "Pump" based on this function block. The data structure of the plant object type is taken over directly from the block. Only the HMI relevant properties from the function block are hereby transferred. They are automatically updated when the block changes. Simply parameterize an instance of the function block as process connection of the plant objects.
- Similarity
  When you have similar plant object types, check if it possible to map these with a common plant object type:



① Example: A pump is installed in a plant in two different variants:

- Variant 1 only measures the flow rate.
- Variant 2 measures the temperature in addition to the flow rate.

Effects on the definition of the plant object types:

- You map each of the two pumps with a single plant object type. The representation in the Common Plant Model hereby corresponds to reality.
- There is more configuration work.

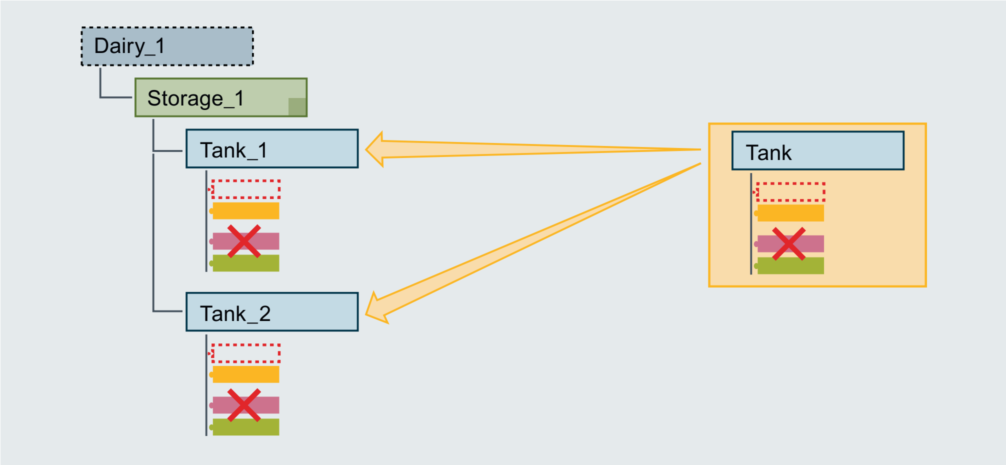
② The common intersection of the two pumps is measuring the flow.

③ If, for example, you can do without measuring the temperature for operation, define only one plant object type:

- There is less configuration work.
- The two variants of the pumps are not fully represented in the Common Plant Model.

Effects of changes on plant object types

The following figure shows how changes to the plant object type affect its instances, i.e. plant objects:



See also

Overview (Page 1509)

Options for creating plant objects (Page 1513)

Introduction (Page 1497)

Applications (Page 1499)

Configuration concept (Page 1504)

Plant model and target systems (Page 1506)

20.1.5 Configuration concept

Requirements

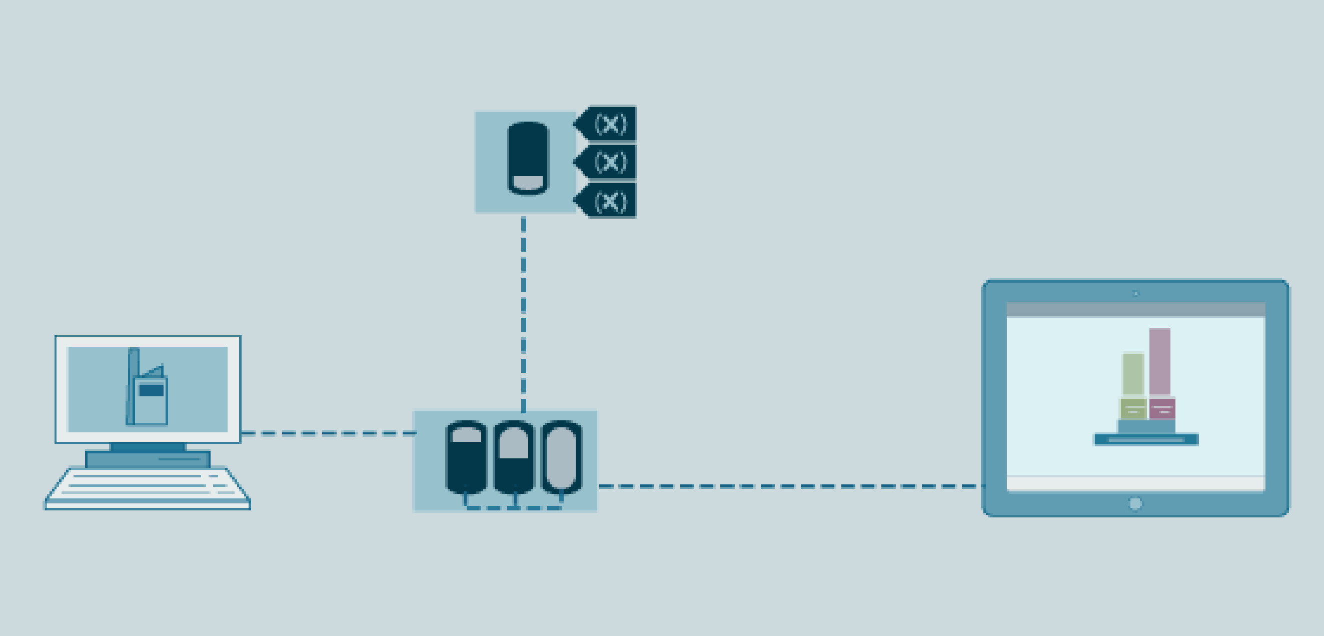
- You have experience in configuring with WinCC and STEP 7.
- The TIA Portal project has been created.
- The HMI device WinCC Unified Scada RT has been created.
- A SIMATIC S7-1500 PLC has been created.
- Data blocks are configured in the PLC.

Workflow for configuration

The starting point for the definition of a standardized object-oriented plant model in object oriented configuration is the existing plant structure.

If you want to create a plant structure, use the following sequence of steps as a guide:

- Analyze the plant structure and break it down into units and components (plant objects)
- Identify required plant object types
- Define data of the plant object types based on FBs and PLC UDTs
- Define hierarchical plant view using instances
- Create a target system
- Map the plant structure
- Position plant objects in the plant structure
- Add functional facets to object types, e.g. assign shift calendars for all machines of a line or plant



Tips for effective procedure
If you are using pre-planning and automation engineering tools, you can have your plant structure automatically created via TIA Portal Openness. Next, set up the process connection of the plant objects via TIA Portal Openess.

Differences between device-oriented and object-oriented configuration

In technology- and object-oriented configuration, you work with objects with relevant names instead of individual tags or faceplates, for example.

You have access to all objects and their properties, methods, etc. in the hierarchy, independent of HMI device assignment.

The equipment from different products and versions is integrated in the object-oriented configuration.

Using multiuser engineering

If you use multiuser engineering in object oriented configuration, you can save your changes only in the server project view. You cannot check the changes you make in the local session into the server project.

You can find more information on Multiuser Engineering in "Using Multiuser Engineering".

See also

Creating plant objects (Page 1517)

Structure of a plant model (Page 1507)

Creating plant object types (Page 1516)

Configure plant object types (Page 1520)

Creating a plant hierarchy (Page 1514)

Type/instance concept in object-oriented configuration (Page 1501)

Plant model and target systems (Page 1506)

Applications (Page 1499)

20.1.6 Plant model and target systems

Configuration of the plant model

When the configuration of a visualization solution begins, the development of the automation solution often takes place in the final phase. Initially, only the actual plant structure is relevant for mapping the plant model. Whether this involves one or multiple target systems is initially irrelevant.

There are always two views in a WinCC project:

- Device view with configured target systems
- Object-oriented view (common plant model)

You can perform configuration independently in both views.

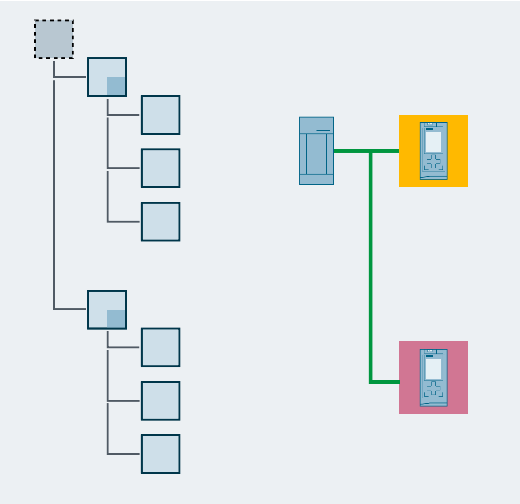
Process connection of the plant model

The target systems are the interface between the common plant model and the process. One or more connections to PLCs are configured on each target system. The plant objects communicate with the PLCs over the target systems.

Your project must meet the following conditions for productive use:

- Each plant view is assigned to a HMI device.
- Each plant object with a process connection is also connected to a PLC.

The following figure shows a schematic representation of the mapping of plant objects to the configured target systems and PLCs:



◪	Plant objects without a process connection as representation of a unit
□	Plant objects with process connection
▯	Runtime server (target system)
▯	PLC

See also

Type/instance concept in object-oriented configuration (Page 1501)

Configuration concept (Page 1504)

20.1.7 Structure of a plant model

Basic principles

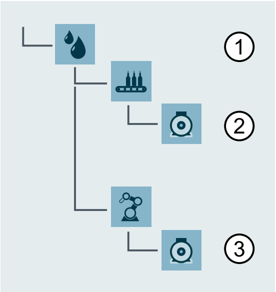
With object oriented configuration, a configured plant object corresponds to a real plant object. Basically, the number of plant objects is determined by the plant hierarchy.

Whether you need to map each plant object with a plant object type is determined by the following factors:

- Relevance of the plant object type for the process visualization
- Depth of the plant hierarchy that is to be mapped
- Degree of reuse

The specific function of a plant object is clear from its position in the plant hierarchy.

For example, the function of a "Drive" plant object is only revealed in the plant hierarchy:



① Process for filling beer into bottles

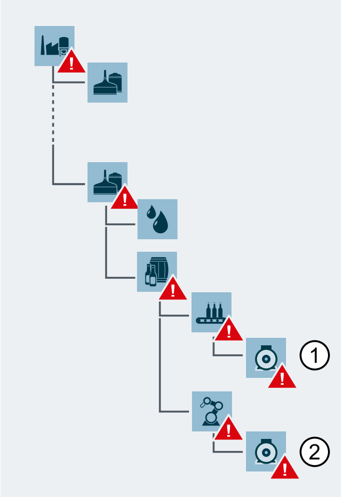
② Drive for conveyor belt

③ Drive for robot

Depth of the plant hierarchy

Define any depth of the plant hierarchy The depth of the plant hierarchy depends essentially on the number of plant objects. A deep plant hierarchy leads to a precise fault localization. You can then, for example, formulate the concise alarm text.

The context of the plant object is also taken into account in runtime, for example, in the localization of faults. The following figure uses the example of the "Temperature exceeded" alarm to show the advantage a deep hierarchy offers in runtime:



① Representation of the message in an alarm control:

"Brewery.Filling.Paletting.Robots.Drive.Temperature exceeded"

The Common Plant Model with deeper hierarchy leads to a precise fault localization. You can therefore formulate the alarm text concisely.

② Since the drive for the robot is based on the same plant object type, the context of the alarm is automatically correct when a fault occurs:

"Brewery.Filling.Paletting.Robots.Drive.Temperature exceeded"

Configuration data at the plant object type

The following configuration data are created during the definition of a plant object type:

- Properties through which data is exchanged inside and outside of WinCC Unified Scada RT.
- HMI visualization: Alarms, logs
- KPIs

See also

Configure plant object types (Page 1520)

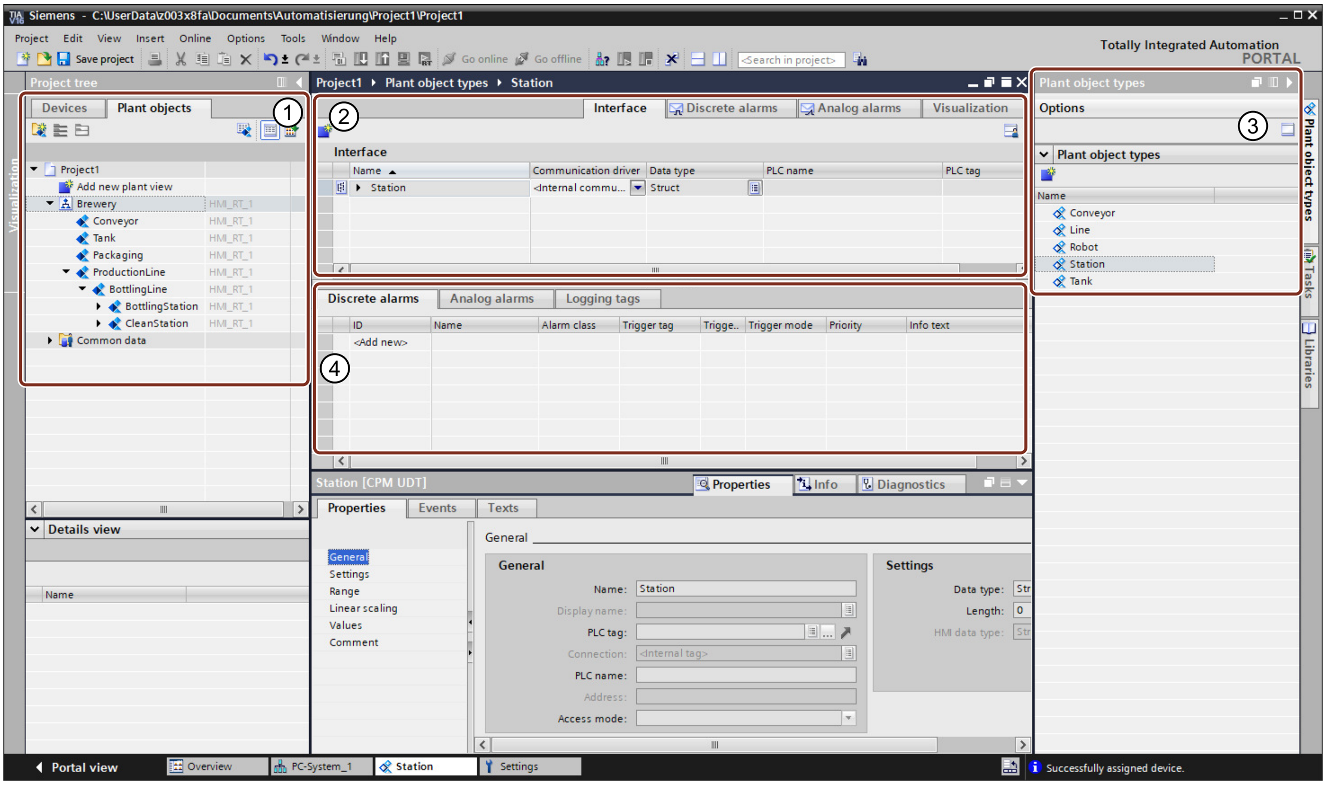
Configuration concept (Page 1504)

20.2 Elements and basic settings

20.2.1 Overview

"Plant objects" area

To access object-oriented configuration, click on "Plant objects" in "Project tree".



Project1	
Add new plant view	
Brewery	HMI_RT_1
Conveyor	HMI_RT_1
Tank	HMI_RT_1
Packaging	HMI_RT_1
ProductionLine	HMI_RT_1
BottlingLine	HMI_RT_1
BottlingStation	HMI_RT_1
CleanStation	HMI_RT_1
Common data	

Interface

Name	Communication driver	Data type	PLC name	PLC tag
Station	<Internal commu...	Struct		

Name
Conveyor
Line
Robot
Station
Tank

ID	Name	Alarm class	Trigger tag	Trigge..	Trigger mode	Priority	Info text
<Add new>							

General

General

Name: Station
Display name:
PLC tag:
Connection: <Internal tag>
PLC name:
Address:
Access mode:

Settings

Data type: Str
Length: 0
HMI data type: Str

Successfully assigned device.

① "Plant objects" area for object-oriented configuration

② Plant object specific tabs, e.g. "Interface", "Visualization", etc.

③ "Plant object types" task card

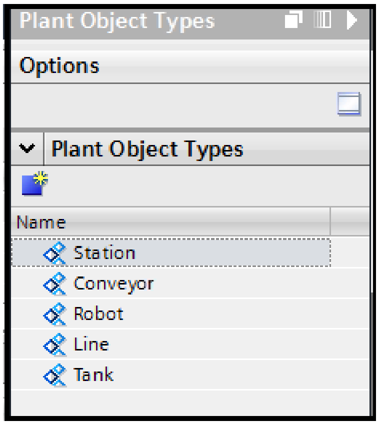
④ Tabs for the configuration of alarms and logs for plant objects

Create the plant view under "Project tree > Plant objects". You can create a plant view in a project. The plant view is filled with plant nodes and thus maps your plant. Plant nodes act as structural elements. Create plant objects based on the plant object types created in the project.

In the "Plant objects" area you assign an HMI device to the plant view.

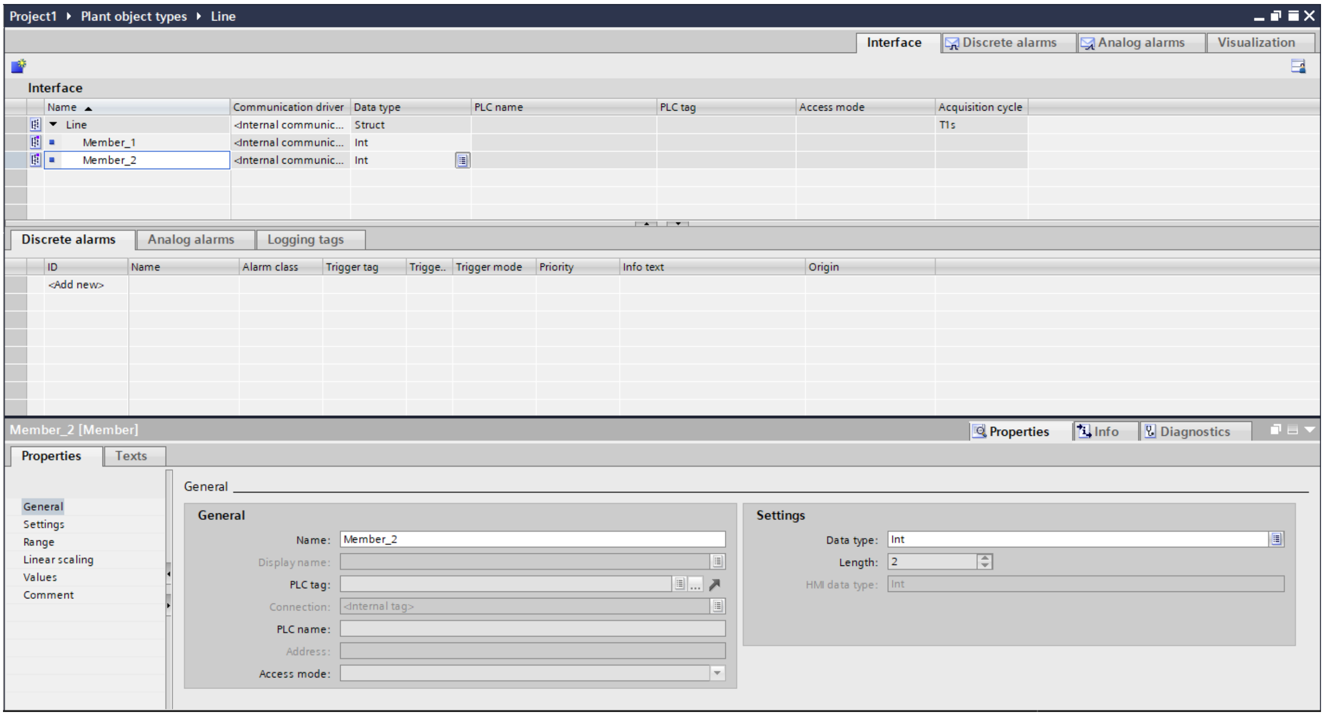
"Plant object types" task card

Under "Plant object types", create the plant object types from which you create plant objects.



"Interface" tab

Plant object types are edited in "Interface" create tags for the communication between a PLC and an HMI device, create members for plant object types, and create alarms and logging tags.



Interface

Name ▲	Communication driver	Data type	PLC name	PLC tag	Access mode	Acquisition cycle
▾ Line	<Internal communic...	Struct				T1s
Member_1	<Internal communic...	Int				
Member_2	<Internal communic...	Int				

ID	Name	Alarm class	Trigger tag	Trigge..	Trigger mode	Priority	Info text	Origin
<Add new>								

General

General

Name: Member_2
Display name:
PLC tag:
Connection: <Internal tag>
PLC name:
Address:
Access mode:

Settings

Data type: Int
Length: 2
HMI data type: Int

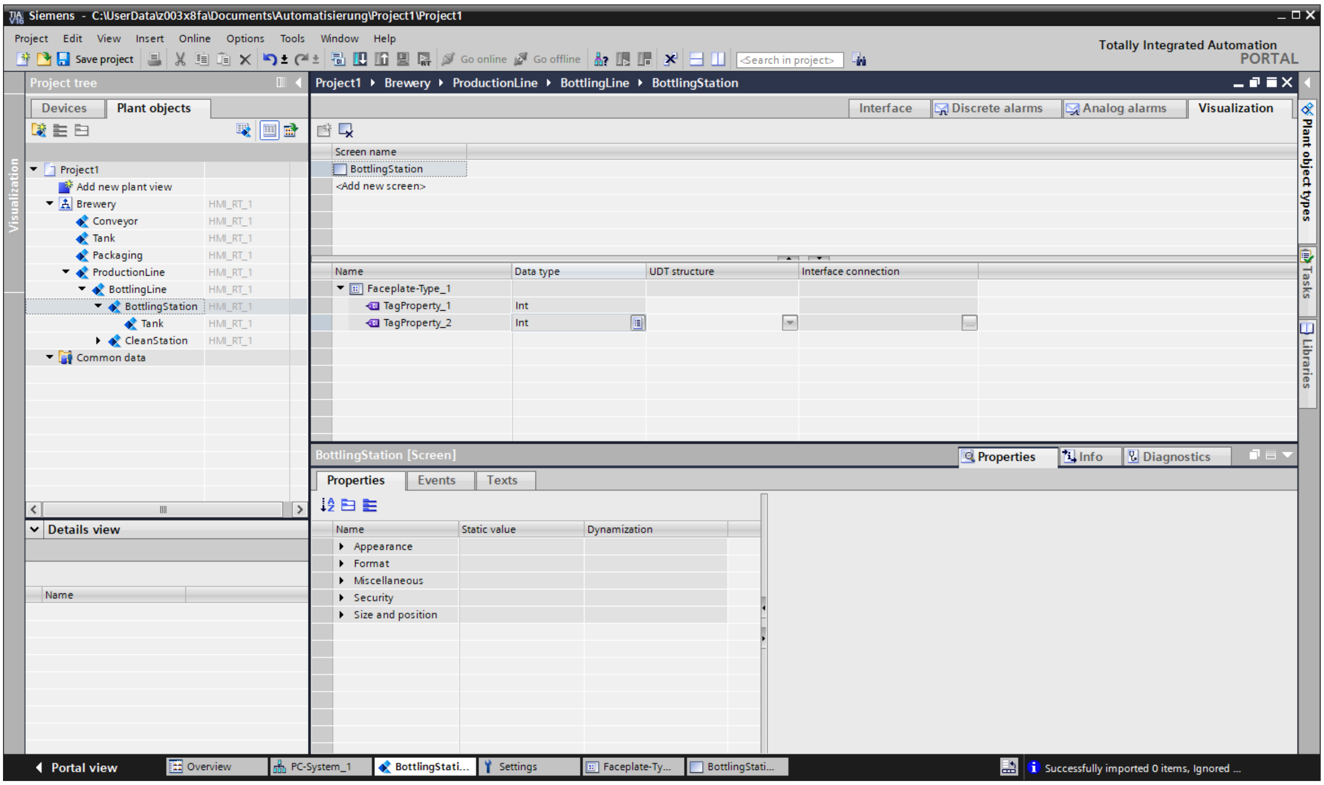
"Visualization" tab of the plant object types

In the "Visualization" tab of a plant object type, you link a faceplate type with the plant object type.

"Visualization" tab of the plant objects

Under "Visualization", you configure a screen for each plant object. In the Inspector window you edit the properties and events of the screen.

The faceplate type associated with the plant type is displayed. The configured tags of the interface are displayed, but cannot be edited.



Screen name
BottlingStation
<Add new screen>

Name	Data type	UDT structure	Interface connection
Faceplate-Type_1			
TagProperty_1	Int		
TagProperty_2	Int		

Name	Static value	Dynamization
Appearance		
Format		
Miscellaneous		
Security		
Size and position		

If you open a screen under "Visualization" by double-clicking, the view is identical to the view on an HMI device. The "Toolbox" and "Layout" task cards are also identical.

Use the "Toolbox" task card to configure in predefined objects in your screens, with which you map your plant, display process sequences and define process values.

See also

Introduction (Page 1497)

Options for creating plant objects (Page 1513)

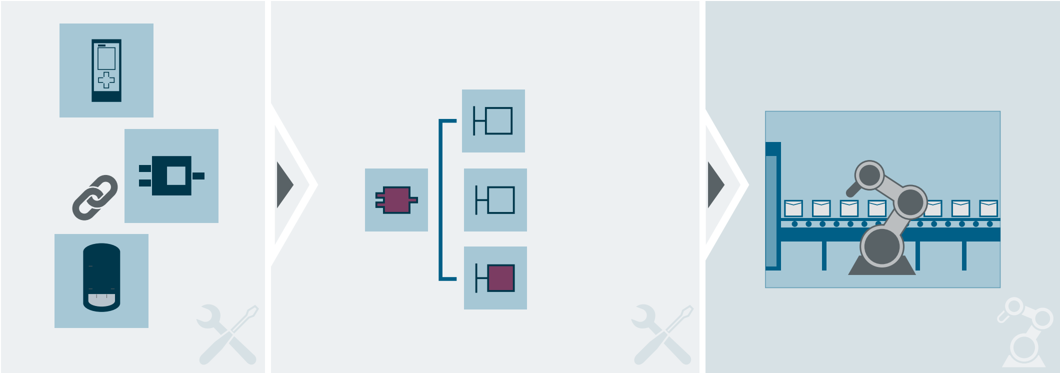
Type/instance concept in object-oriented configuration (Page 1501)

20.2.2 Options for creating plant objects

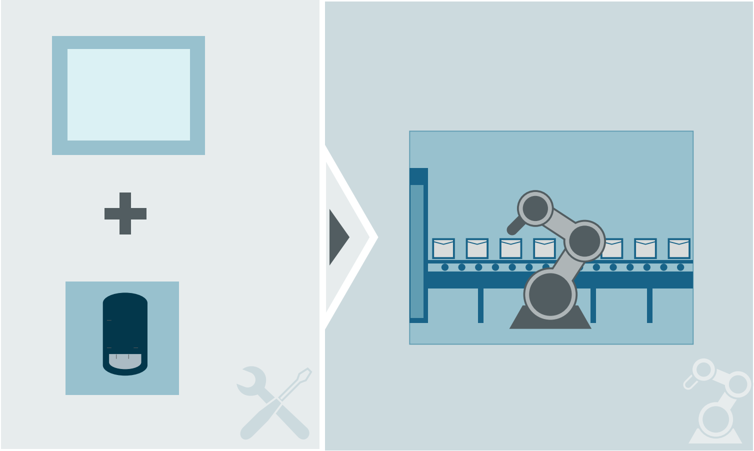
Basics

You have several options for creating plant objects on the basis of plant object types:

- Creation of plant object types from the function blocks or UDTs of an S7-1500 and creation of plant objects from the IDBs.



- Creation of plant object types within WinCC without an S7-1500



See also

Overview (Page 1509)

Type/instance concept in object-oriented configuration (Page 1501)

20.3 Object- and technology-oriented configuration

20.3.1 Creating a plant hierarchy

Introduction

Create a plant view to map the structure of your plant. You fill the plant view with plant objects and plant nodes and thus map your plant. Plant nodes act as structural elements.

Assign the plant view to a HMI device.

Requirement

- The TIA Portal project has been created.

Procedure

1. Under "Project tree > Plant objects", click on "Add new plant view".
   An empty plant view is created.

   Note

   A plant view is supported in each project.

2. Rename the plant view accordingly.

   Note

   The following options are not available for the "Plant view" object:
   - Paste
   - Cut
   - Drag-and-drop

See also

Assigning a plant hierarchy to a HMI device (Page 1515)

Creating plant objects (Page 1517)

Configure plant object types (Page 1520)

Configuration concept (Page 1504)

20.3.2 Assigning a plant hierarchy to a HMI device

Introduction

To operate the plant in runtime, always assign a plant view to an HMI device.

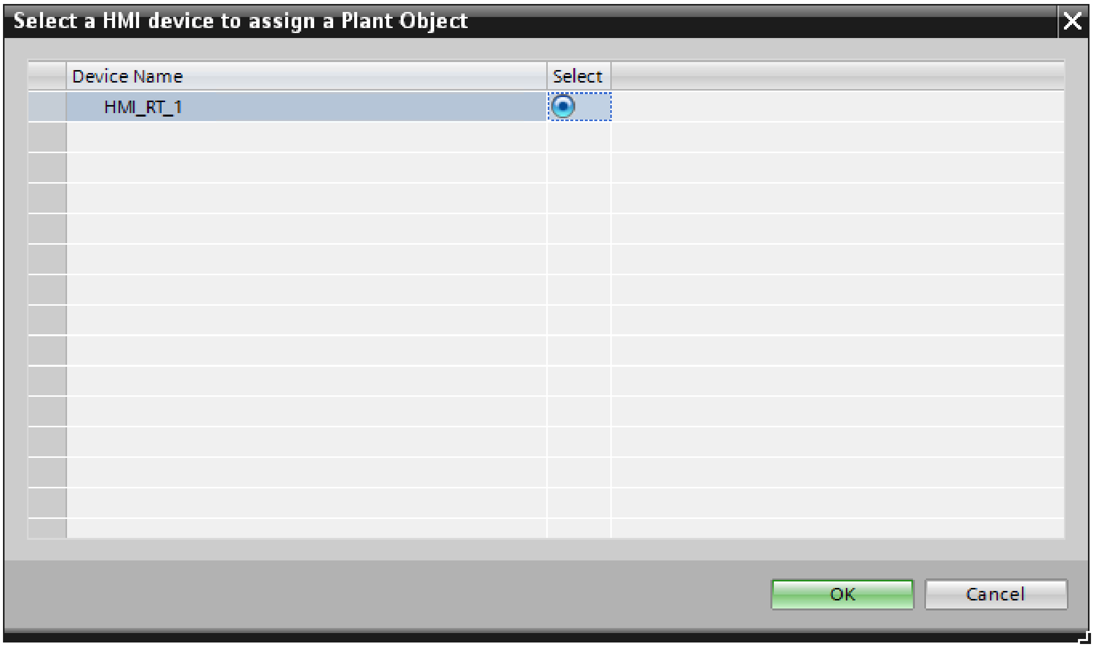
A plant view can only be assigned to a HMI device.

Requirement

- A plant view has been created.
- The HMI device WinCC Unified Scada RT has been created.

Procedure

1. Select the "Plant view" node.
2. Select the "Assign HMI device" entry from the shortcut menu.
   A "Select an HMI device for assignment" dialog appears.



Select a HMI device to assign a Plant Object

Device Name	Select
HMI_RT_1	●

3. Select the HMI device.
   The plant view and all lower-level plant objects are assigned to the HMI device.
   If a plant view was assigned to a HMI device, the assignment is visible under "Project tree > Plant objects".

See also

Creating plant objects (Page 1517)

Creating plant object types (Page 1516)

Configure plant object types (Page 1520)

Creating a plant hierarchy (Page 1514)

20.3.3 Creating plant object types

Introduction

You create plant object types.

Then define the "Communications driver" property of the interface:

- "<Internal communication>": Create data members for internal communication.
- "SIMATIC S7 1200/1500": Use either function blocks or UDTs of a S7-1500
  You can add further data members to the linked structure.

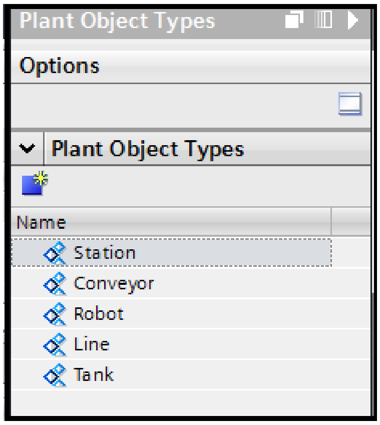
Requirement

- A project is open.
- A SIMATIC S7-1500 PLC has been created.

Procedure

Create plant object types in the "Plant object types" task card.

1. To display the "Plant object types" task card, click the "Show plant object types" button under "Project tree > Plant objects".



2. To create a plant object type, click "Add new plant object type".
   An empty plant object type is created.

3. Rename the created plant object type accordingly.
4. To edit the plant object type or create lower-level objects and members for the plant object type, double-click the plant object type in the "Plant object types" tab.
   The plant object type appears under "Interface".

   Note

   The "Communications driver" property is editable for the plant object types. The property "PLC tag" can only be edited with the communications driver "SIMATIC S7 1200/1500".

See also

Configure plant object types (Page 1520)

Example: Determine plant object type (Page 1518)

Assigning a plant hierarchy to a HMI device (Page 1515)

Requirements (Page 1501)

Configuration concept (Page 1504)

20.3.4 Creating plant objects

Introduction

You create plant objects from a plant object type using drag-and-drop operation.

Plant objects are specific versions or instances of a plant object type.

Requirement

- A project is open.
- A plant object type has been created.

Procedure

1. Open the "Plant objects" tab in the "Project tree" area.
2. Open the "Plant object types" task card.
3. Drag the plant object type from the task card to the plant view.
   An empty plant object is created.
4. Rename the plant object accordingly.

   Note

   The name of a plant object must only be assigned once within a project.

See also

Configuration concept (Page 1504)

Configure plant object types (Page 1520)

Assigning a plant hierarchy to a HMI device (Page 1515)

Requirements (Page 1501)

Creating a plant hierarchy (Page 1514)

20.3.5 Example: Determine plant object type

Scenario

For a new location of a brewery, two employees of an engineering office configure the process visualization and plant-specific parameters. The employees develop a configuration concept for this.

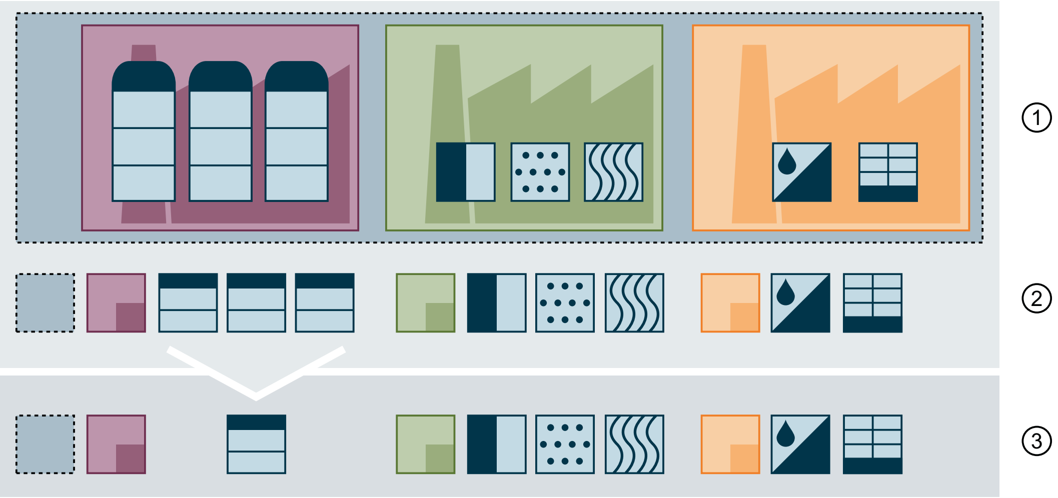
The following examples shows how process visualization and object-oriented configured mesh with each other.

Determining plant object types

How you determine plant object types by analyzing the plant structure depends on the context:

- In the WinCC Runtime Unified context, you view the plant "from the bottom" in the process view. Functional units are in the background compared to plant objects of the field and process levels. The functional units are still necessary for a complete mapping of the plant.
- In the context of plant-specific KPIs, look at the plant "from the top" in the plant view. Plant objects of the field and process levels may no longer be relevant.

The following figure is a schematic representation of the analysis sequence of the plant structure for defining the plant object types:



① Analyze the plant structure: The brewery consists of the three plant units, "Delivery/Storage", "Processing" and "Bottling". Various processes run in the plant sections.

② Determining relevant plant objects for monitoring process and productivity: These plant objects are the basis for mapping the plant hierarchy.

③ Defining plant object types: Plant objects used multiple times are mapped using a common plant object type. The data structure and context information is configured for each plant object type.

⬚ Brewery

- Plant-specific parameters: Cumulative characteristics of productivity

◪ Unit for delivery and storage of ingredients

- Plant-specific parameters: Characteristics for duration of delivery

☰ Tank for storage of ingredients

- Process visualization: Monitoring temperature and fill level

◪ Unit for processing

- Plant-specific parameters: Characteristics of productivity

◧ "Mashing"

- "Batches taken out of mash" Process visualization: Monitoring temperature and pressure
- "Pump back batches taken out of mash", check: "Iodine test": Process visualization: Monitoring temperature and pressure

⁙ "Purifying" Process visualization: Monitoring temperature and pressure

≋ "Wort boiling" Process visualization: Monitoring temperature and pressure

- Whirlpooling
- Addition of yeast and fermenting
- Storage

◪ Unit for bottling and packaging

- Plant-specific parameters: Characteristics for period of change in production, unplanned downtime and produced quantities

Bottling process

- Process visualization: Monitoring temperature / control of filling

Process for packaging the beer bottles on pallets

- Process visualization: Monitoring packaging

Note

The data structure of a plant object type is often reflected in the function blocks of the user program. In such a case you can create plant object types automatically. Consultation with the programmer is recommended especially for plant object types with strong links to the PLC.

See also

Creating plant object types (Page 1516)

20.3.6 Configure plant object types

Introduction

Configure the plant object types either from the function blocks and UDTs of an S7-1500 or create the properties and the external and internal data members for the plant object types.

In both cases you can extend the structure the created plant object types with additional internal or external data members.

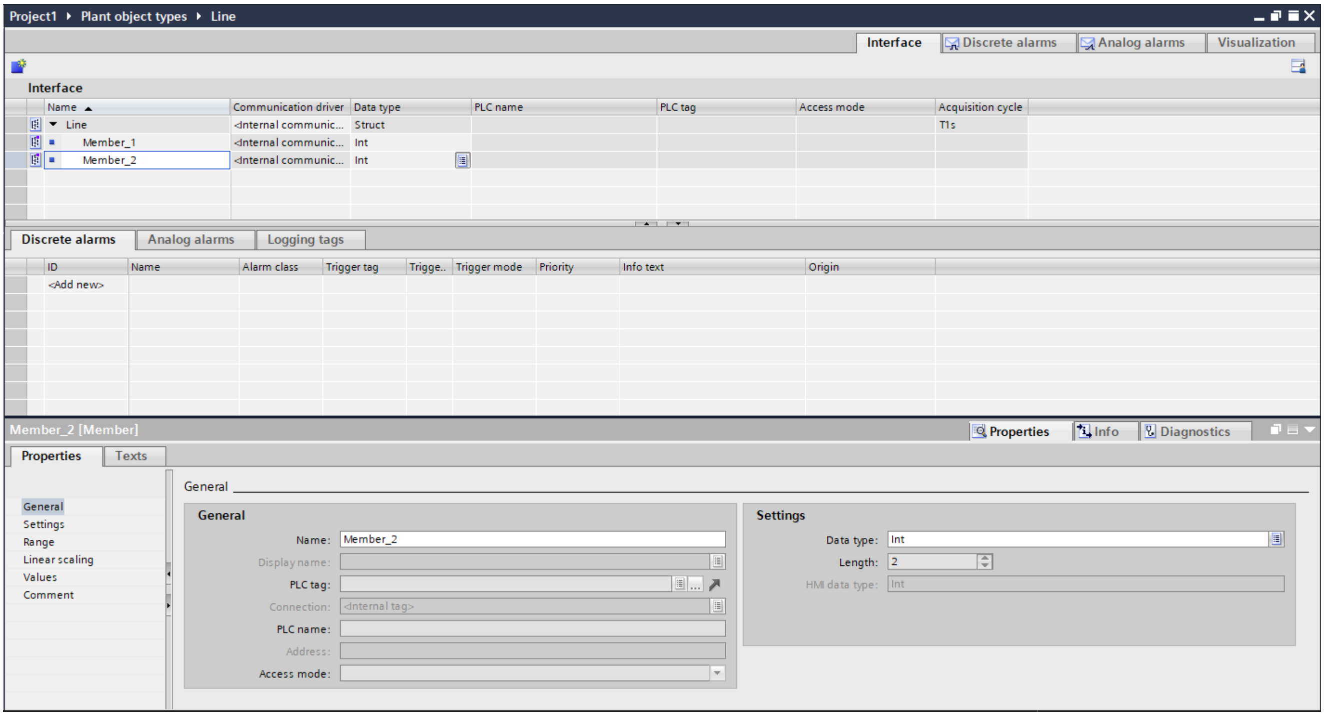
Configuration without using function blocks is described below.

Requirement

- The HMI device WinCC Unified Scada RT has been created.
- A plant view has been created and assigned to the HMI device.
- A plant object type named "Line" has been created.
- A SIMATIC S7-1500 PLC has been created.
- Tags have been configured in the S7-1500.

Procedure

1. Double-click the "Line" plant object type in the "Plant object types" editor.
   An empty plant object type with the "Struct" data type appears under "Interface".
2. To add data members to the plant object type, select the plant object type and click "Insert object".



Interface

Name ▲	Communication driver	Data type	PLC name	PLC tag	Access mode	Acquisition cycle
▾ Line	<Internal communic...	Struct				T1s
Member_1	<Internal communic...	Int				
Member_2	<Internal communic...	Int				

ID	Name	Alarm class	Trigger tag	Trigge..	Trigger mode	Priority	Info text	Origin
<Add new>								

General

General

Name: Member_2
Display name:
PLC tag:
Connection: <Internal tag>
PLC name:
Address:
Access mode:

Settings

Data type: Int
Length: 2
HMI data type: Int

The created data member inherits all properties from the higher level plant object type. "Internal communication" is selected by default in the column "Communications driver" for the newly created data members of the plant object types.

3. If you want to configure an external data member, select "SIMATIC S7-1500" in the "Communication driver" column.
4. Assign a PLC tag to the external data member in the "Tag" column.

Note

If an HMI device is assigned to the plant view, it is possible to view the data members in the "HMI tags" editor in the "Plant object tags" tab. You also have write rights for the "Comment" column.

You can also use the configured data members in screens of an assigned HMI device, e.g., for dynamization instead of tags.

Tips for effective procedure

- Differentiate between identically named plant objects and plant object types using the "Insert object" button in the "Interface" tab. If the "Insert object" button is enabled, you have selected a plant object type. If you have selected a plant object, the "Insert object" button is disabled.

See also

Creating plant object types (Page 1516)

Structure of a plant model (Page 1507)

Creating plant objects (Page 1517)

Assigning a plant hierarchy to a HMI device (Page 1515)

Configuration concept (Page 1504)

Creating a plant hierarchy (Page 1514)

Configuring plant object types from the data blocks of an S7-1500 (Page 1522)

Assigning process data to plant objects (Page 1524)

20.3.7 Configuring plant object types from the data blocks of an S7-1500

Introduction

Configure the plant object types either from the data blocks of an S7-1500 or define the properties and the external and internal data members for the plant object types without connection to a PLC.

In both cases you can extend the structure the created plant object types with additional internal or external data members.

Configuration from the data blocks of an S7-1500 PLC is described below.

Configure plant object types from the configured program blocks of an S7-1500 PLC using drag-and-drop operation.

Requirement

- The HMI device WinCC Unified Scada RT has been created.
- A plant view has been created and assigned to the WinCC Unified Scada RT.
- A plant object type named "Line" has been created.
- A SIMATIC S7-1500 PLC has been created.
- A function block "Line [FB1]" is configured in the SIMATIC S7-1500 PLC.

Procedure

1. Open the plant object type "Line" in the "Interface" tab.
2. Set the "Communications driver" parameter to the SIMATIC S7-1500 PLC that contains the function block "Line [FB1]".
3. Navigate to the function block "Line [FB1]" in the PLC.
4. Select the function block.

5. Drag the function block to the "PLC tags" field in the "Interface" tab.
   The corresponding structure with data members based on the function block "Line [FB1]" is created in the "Interface" tab.
   When you edit the blocks of the PLC, these changes are automatically transferred to the plant object types.
6. To add additional data members to the plant object type, select the plant object type and click "Insert object".

   Note

   You can create additional data member for each plant object type.

   Function blocks (FBs) or PLC UDTs act as basis for the configuration of the plant object types and their data members.

   A member structure can also be connected with a PLC type, for example, function block (FB) or PLC UDT.

   "Raw" data types and arrays are also supported.

   An assignment of the controller blocks to the external data members of the plant object types is only possible if names and data types are identical in the PLC and in the plant object type.
7. If necessary, adjust the data type of the data member.
8. To clear the connection between of a controller and the data members object, delete the block in the "PLC tag" column or select "None".

Tips for effective procedure

- Differentiate between identically named plant objects and plant object types using the "Insert object" button in the "Interface" tab. If the "Insert object" button is enabled, you have selected a plant object type. If you have selected a plant object, the "Insert object" button is disabled.

Note

If an HMI device is assigned to the plant view, it is possible to view the data members in the "HMI tags" editor in the "Plant object tags" tab. You also have write rights for the "Comment" column.

You can also use the configured data members in screens of an assigned HMI device, e.g., for dynamization instead of tags.

See also

Configure plant object types (Page 1520)

Assigning process data to plant objects (Page 1524)

20.3.8 Assigning process data to plant objects

Introduction

To establish the communication between a S7-1500 controller and a WinCC Unified Scada RT device, connect a plant object with a PLC tag or a data block of the PLC.

Requirement

- An S7-1500 PLC and a WinCC Unified Scada RT HMI device are projected and connected.
- At least one plant object type in the project contains PLC UDTs or function blocks (FBs).

Procedure

1. Drag a plant object type to the plant view.
   The plant object is created based on the plant object type.
2. Double-click the plant object.
3. In the "Interface" tab, in the "Connection" column, select the configured HMI connection for all external data members of the plant object type.
   Select only between the HMI connections that are created for the S7-1500 controllers available in the project.
4. In the "PLC tag" column, select a PLC tag or an IDB.

Note

In the "Interface" tab, similar to in the "HMI tags" editor, you can view or edit the properties in the following areas:

- "General"
- "Settings"
- "Range"
- "Linear scaling"
- "Values"
- "Comment"

See also

Configure plant object types (Page 1520)

Configuring plant object types from the data blocks of an S7-1500 (Page 1522)

20.3.9 Basic information on configuring screens

Overview

The configuration of screens for operating and monitoring is also available to you in object-oriented configuration. This means that you are working in two areas, under "Project tree> Devices" and "Project tree > Plant objects > Visualization". Here you work with both screens and faceplates that also support the type-instance concept. You can find additional information on working with faceplates in the section Configuring faceplates (Page 103).

In the area "Project tree > Devices", configure screens for HMI devices as usual. In the screens, also configure companion controls that are relevant for the display of screens of the plant objects.

In the "Project tree > Plant objects > Visualization" area, you configure screens for plant objects.

In the "Plant object types > Visualization" area, configure faceplates for plant object types.

In the areas under "Project tree > Devices" and "Project tree > Plant Objects > Visualization", the same predefined screen objects are available in the "Toolbox" task card.

When configuring faceplates, a minimized tool area is available under "Toolbox".

Configuration options

Under "Project tree > Devices", you configure a screen for the created HMI device with the "Plant overview" control and one of the companion controls, such as a screen window. In runtime, navigate the plant structure to the plant objects via the "Plant overview" control. The screen windows in the plant overview display the screens that you have previously configured for the plant project.

The companion controls are connected to one another and supplement one another in displaying the data values.

The following controls can act as companion control for the plant overview:

- Alarm control
- Screen window
- Calendar view (with use of the WinCC Unified Calendar option)

When required, configure controls as usual, e.g. screen windows, alarm view, or trend view. Select the specific plant object in the plant view as data source for the alarm control and trend control. Configure the alarm control and trend control for plant objects on the basis of the data members of the plant object types. The procedure for configuring these controls does not differ from the procedure for the device-specific configuration.

The following options are available in runtime:

- Display the hierarchy path of the alarm source
- Display the hierarchy path of the trend values
- Filter the alarm view by plant objects
- Display process values for the selected plant object

Use the "Visualization" tab for the direct visualization of plant objects and plant object types. You can create one screen for each plant object, and you can link one faceplate type for each plant object type. The type-instance concept is used for configuring the screens for plant objects and plant object types. All relevant elements are contained in the faceplate type of a plant object type. Drag a plant object to your screen using drag & drop. A faceplate container is created. For example, several faceplates can be integrated in a visualization of a higher-level plant object.

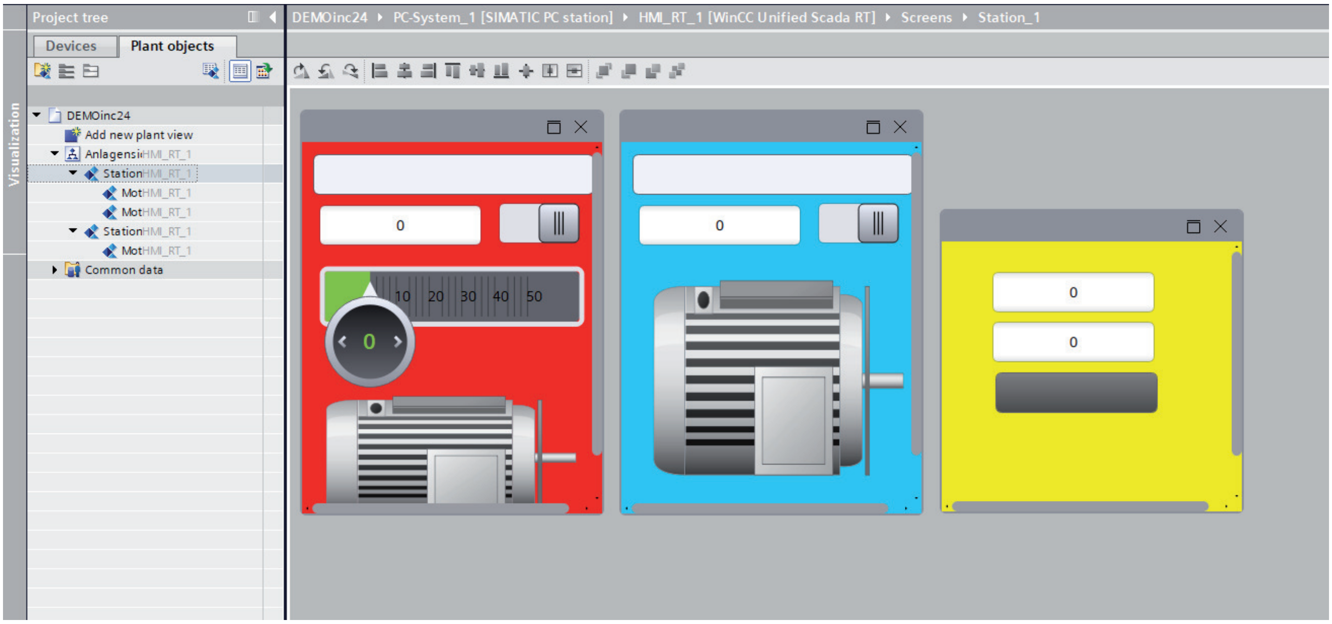


Figure 20-1 Screen of a plant object with several faceplate containers

Configuration steps

In general, proceed in the following order when configuring the screens for your plant:

1. Configure plant object types.
2. Configure faceplates for plant object types.
3. Create plant objects from plant object types.
4. Create screens for plant objects.
5. For the display in runtime, configure the "Plant view" and "Screen window" controls in a screen of the HMI device.

Displaying plant objects and plant object types

In general, you do not have to create a screen for each plant object. Create an overview for the higher-level plant object. Then create faceplates for the plant object types and drag them to the overview screen of the higher-level plant object using drag-and-drop.

If the screens of the plant objects need to differ from the screens used for the plant object types, you can use the faceplates of the plant object types as a basis and configure additional elements.

In runtime, select a plant object in the "Plant overview" control to display its screen.

See also

Configuring screens for plant objects (Page 1527)

Example: Configuring screens for brewery production lines (Page 1529)

20.3.10 Configuring screens for plant objects

Introduction

You configure an overview screen for the higher-level plant object with multiple faceplate containers for lower-level plant objects, for example, for a station that has lower-level objects motor and conveyor belt.

For each plant object you can configure a screen in which all lower-level plant objects are visible. To do this, use the faceplate containers of the plant object types.

If necessary, you also configure basic objects, elements and controls in the screen. For example, you use I/O fields to display process values such as status, temperature and rate.

In the following, you will obtain the data to be processed, such as temperature measurements or speed values from the data blocks of a controller.

You represent lower-level plant objects using faceplates in the overview screen of the higher-level plant hierarchy.

In runtime, the screen window technology assists you in switching between plant objects and representing multiple plant objects in a screen.

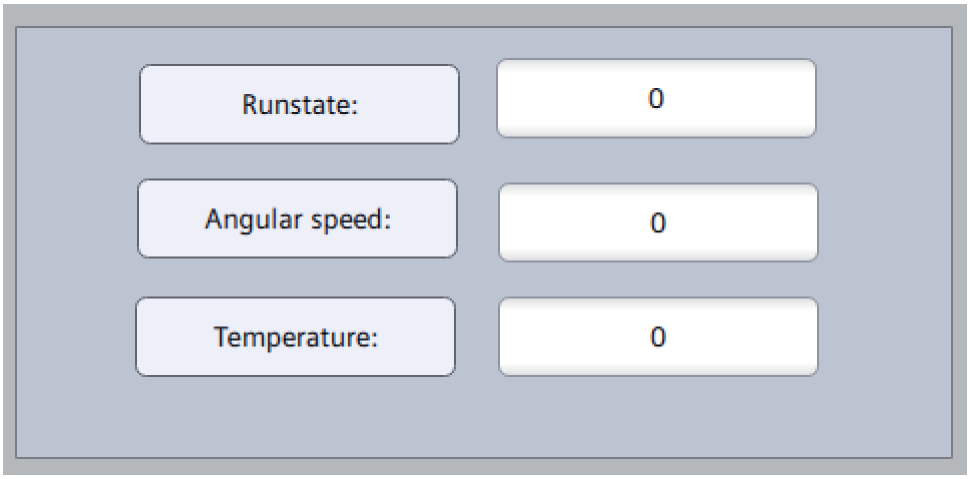
You can also use the "Plant overview" control to set up screen navigation via the plant. In runtime, you monitor the plant in this manner and see the overall progress at a glance.

Requirement

- A SIMATIC S7-1500 has been configured in the project.
- WinCC Unified Scada RT is configured in the project.
- A plant object type named "Station" has been created.
- A plant object type named "Conveyor" has been created.
- A plant object type named "Robot" has been created.
- The plant view with the "Station_1" and "Conveyor_1" plant objects has been created and assigned to the WinCC Unified Scada RT.
- The interface tags of the plant object types "Robot" and "Conveyor" are linked to the S7-1500.

Procedure

1. Open the editor of the higher-level "Station_1" plant object.
2. In the "Visualization" tab, click "Add screen".
   A screen named "Station" is created.
3. If necessary, edit the width and height of the screen under "Properties" in the inspector window.
4. If necessary, configure the required elements and controls for the plant object "Station_1", such as I/O fields and text fields.



Runstate: 0

Angular speed: 0

Temperature: 0

5. In the device view, switch to "Project > Shared Data > Unified Faceplate Types".
6. Create a "Robot" faceplate type.
7. Configure the required basic objects, elements and controls in the faceplate type.
8. Open the editor of the "Robot" plant object type.
9. In the "Visualization" tab, drag the "Robot" faceplate from "Devices > Project > Common data > Unified faceplate types" to the "Store faceplates here" button.
10. Connect the faceplate tags to the interface tags of the plant object type.
11. Create the plant object "Robot_1" from the plant object type "Robot" using drag-and-drop.
12. Open the editor of the "Robot_1" plant object type.
13. Assign the respective data block to the plant object under "Interface" in the "PLC tag" column.
14. To display the faceplate container of the "Robot" plant object type in the screen of the higher-level plant object "Station", drag the plant object "Robot" from the plant view to the configured screen "Station".
15. To create the faceplate container for the plant object "Conveyor_1", repeat steps 5 to 11 for the plant object type "Conveyor".

Tips for effective procedure

- Adjust the position of the faceplate container in the overview screen using the mouse or the corresponding icons on the toolbar.
- You can zoom in and out of the faceplate container in the overview screen.
- You can at any time delete and reconfigure the overview screen which contains the faceplate container for lower-level plant objects. You can reuse the faceplate types at any time.

Note

If the screen area is not sufficient for all faceplate containers, the faceplate containers are superimposed on each other in runtime.

Note

If additional basic objects, elements and controls are required specifically for a plant object, you can use the faceplates of the plant object types as a basis and configure additional objects.

Result

Using the faceplate containers of the "Robot" and "Conveyor" plant object types, you have created visualizations for the "Robot_1" and "Conveyor_1" plant objects.

See also

Operating "Plant overview" in runtime (Page 1548)

Basic information on configuring screens (Page 1525)

Example: Configuring screens for brewery production lines (Page 1529)

Configuring an alarm control for plant objects (Page 1534)

20.3.11 Example: Configuring screens for brewery production lines

Example scenario: Brewery

The production lines "Bottling" and "Packaging" exist alongside other lines in a brewery and are connected to one another in the production chain. The production lines consist of multiple units.

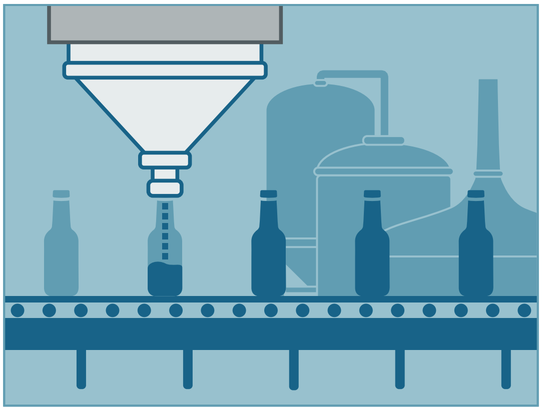
In the plant view, the plant objects are already configured from the plant object types based on the data blocks.

Screens must be configured for all plant objects relevant for the monitoring. In addition, a screen navigation should be set up so that the user can navigate from one plant object screen to another in runtime using the "Plant overview" control. The production value should also be monitored, for example the temperature in the filling tank or the weight of the product after the filling. Make sure to notify the operator in case of deviations.

The relevant production values must be logged for quality assurance purposes and for Food Authority audits.

The "Bottling" production line consists of the following units:

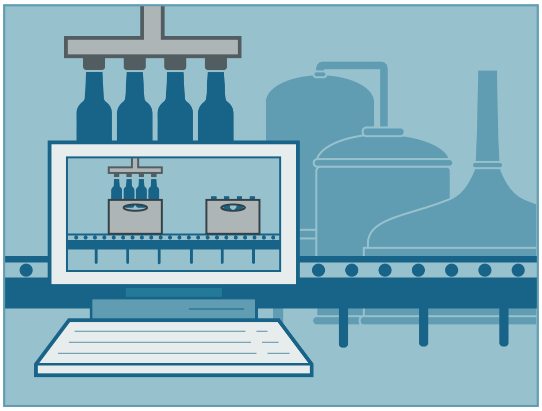
- Conveyor belt 1
- Robot 1: Places the bottles on the conveyor
- Filling tank: Fills the bottles, the temperature in the filling tank is monitored.
- Robot 2: Closes the bottles
- Robot 3: Performs quality checks (weight and light barrier)



The bottles are sorted from the conveyor belt into beverage crates on the "Packaging" production line.

The "Packaging" production line consists of the following units:

- Conveyor belt 2: Makes the filled bottles available
- Conveyor belt 3: Conveys filled crates
- Robot 4: Places the crates on the conveyor belt
- Robot 5: Puts bottles in the crates
- Robot 6: Places the crates on the pallet
- Robot 7: Performs quality checks (weight and light barrier)



Implementation concept

You create faceplate types for the "Conveyor" and "Robot" plant object types.

The faceplate types of the plant object types are instantiated in the screens of the plant objects. The "Robot" faceplate is reused three times in the production line "Bottling" according to the structure of the production line, and four times in the production line "Packaging".

You want to monitor the temperature the filling container and to notify the operator changes occur. To monitor the temperature, configure a trend control for the filling tank. Since a specific temperature must not be exceeded during the bottling of beverages, you have configured analog alarms for the filling containers. To display the alarms in runtime, configure the alarm control.

You want to notify the operator when the empty containers are running out or the liquid level in the bottling containers falls below a specific limit. You have configured analog alarms for the plant objects "Robot 2" and "Filling container" for these purposes.

The production values are logged for the Food Authority inspections. It must be verified that the temperature was complied with and that quality checks were regularly performed. You have configured logging tags for the relevant plant objects for this purpose.

Procedure

1. Under "Project tree > Plant objects", configure the overview screens "Bottling" and "Packaging" lines.
2. Under "Project tree > Devices > Project > Common data > Unified faceplate types", create a faceplate type for the "Robot" plant object type.
3. Under "Project tree > Plant objects", open the plant object type "Robot" and drag the created faceplate type to the "Save faceplates here" button in the "Visualization" tab.
4. Create faceplate instances for all the robots you need for the two production lines:
   - Open the "Bottling" overview screen.
   - then drag the plant object onto the screen.
   - Repeat this procedure for all plant objects that are based on the "Robot" plant object type.
   - Repeat the procedure for the "Packaging" overview screen.
5. Under "Project tree > Devices > Project > Common data > Unified faceplate types", create a faceplate type for the "Conveyor" plant object type.
6. Create faceplate instances for all the conveyors you need for the two production lines:
   - Open the "Bottling" overview screen.
   - then drag the plant object onto the screen.
   - Repeat this procedure for all plant objects that are based on the "Conveyor" plant object type.
   - Repeat the procedure for the "Packaging" overview screen.
7. Under "Project tree > Devices > Project > Common data > Unified faceplate types", create a faceplate for the "Filling tank" plant object type and drag the created faceplate type to the "Place faceplates here" button in the "Visualization" tab.

8. From the "Filling tank" plant object, create a faceplate instance for the filling container:
   – Open the "Bottling" overview screen.
   – Then drag the plant object onto the screen.
9. Create an "Overview" screen.
10. Under "Project tree > Devices", configure the controls "Trend control", "Alarm control", "Screen window" and "Plant overview" in the "Overview" screen..
    Configure the "Alarm view" and "Screen window" controls as companion controls of the "System overview" control.

Result

You have successfully configured screens and faceplates for the plant objects of the brewery and can display the plant objects in the runtime.

See also

Configuring screens for plant objects (Page 1527)

Basic information on configuring screens (Page 1525)

20.3.12 Configuring "Plant overview" control and companion controls

Introduction

You require the control "Plant overview" when you want to navigate through the plant.

The companion controls are connected to one another and supplement one another in displaying the data values.

You require the companion controls for the following displays:

- Display plant object screens and screen windows using the navigation option throughout the entire plant (plant overview and screen windows)
- Display alarms for plant objects using the navigation option throughout the entire plant (plant overview and alarm view)

The following controls can act as companion control for the plant overview:

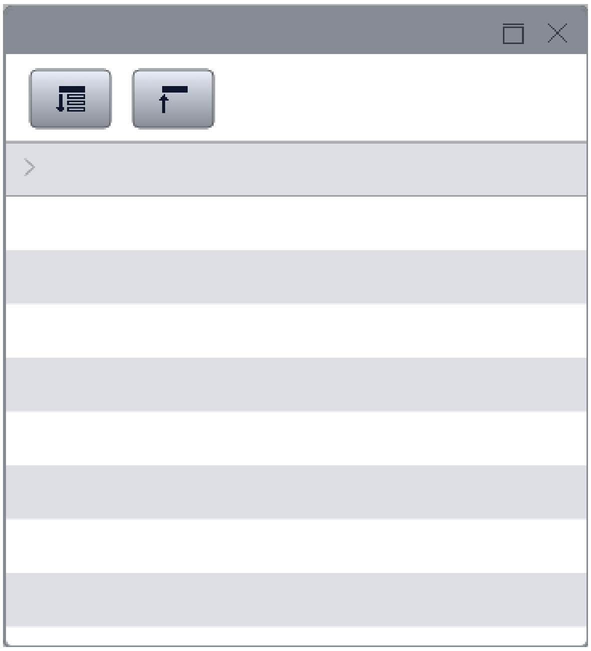
- Alarm control
- Screen window
- Calendar view (when using the WinCC Unified Calendaroption)

Requirement

- A screen is open.
- The "Toolbox" task card is open.

Procedure

1. Insert the "Plant overview" control from the "Toolbox > My controls" task card into the screen.



2. Add a companion control.
   Select from the following controls:
   - Alarm control
   - Screen window
   - Calendar control

   Note

   As companion controls, you can only select controls already configured in the screen.

3. Select the "Plant overview" control.
4. Open the Inspector window under "Properties > Properties > General > Interface > Companion controls".
5. Click on the selection button in the "Static value" column.
   The dialog for selecting companion controls opens.
6. Click "Add".
   A new index entry (beginning with 0) is created.
7. Create index entries according to the number of ancillary controls to be linked with the "Plant overview" control.
8. Close the dialog.
   The "Companion control" editor opens on the right-hand side of the Inspector window.
   The configured elements are displayed in the "Companion control" editor.

9. Specify the control type for each element:
   - Alarm control
   - Display window (for screen windows)
   - Calendar control (for calendar control)
10. Define the respective companion control as control reference for each element.

Result

In runtime you see the screen with the "Plant overview" control and the companion controls. When you navigate to the respective plant object in the "Plant overview" control, the content of this plant object is displayed in the companion controls.

If you have configured the screen window as companion control, navigate in runtime in the "Plant overview" control through the plant and display the screens you have configured for the respective plant object.

If you have configured the alarm control as companion control, navigate in runtime in the "Alarm overview" control through the plant and have the alarms for the plant objects displayed in the alarm view.

20.3.13 Configuring an alarm control for plant objects

Overview

Configure an alarm control, as in the device-specific configuration in a HMI device screen. In order that the alarm control can display the alarms of the plant objects, assign the plant hierarchy to your HMI device.

To directly jump to the alarms of the plant objects in runtime, configure the alarm control is companion control to the "Plant overview" control.

To filter by plant object alarms in the alarm control, configure a filter with the criterion "Area" with one of the following two conditions:

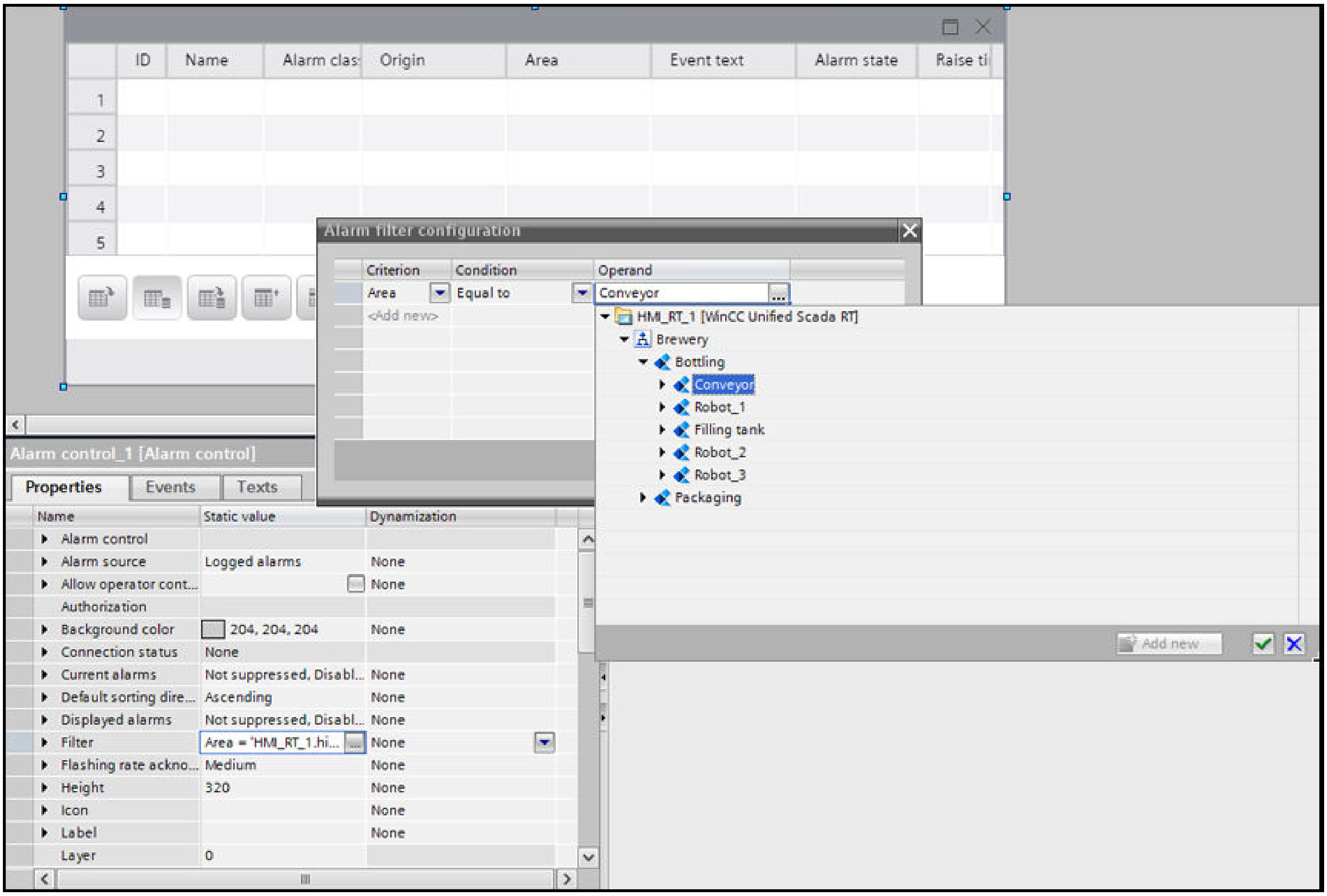
- "Equal" - only shows the alarms of the selected plant object in runtime.
- "Begins with" - shows the alarms of the underlying objects of the selected plant object in runtime.

Configuring a filter for plant objects

To filter by plant object alarms in the alarm control, configure a filter as follows:

1. In the Inspector window under "Properties > Filter", click in the "Static value" column.
   The "Alarm filter configuration" dialog opens.
2. Select the "Area" criteria.
3. Select the condition "Equal"

4. Click on the selection list in the "Operand" column.
5. Select the plant object whose alarms you want to display in runtime.



	ID	Name	Alarm clas	Origin	Area	Event text	Alarm state	Raise ti
1								
2								
3								
4								
5								

Alarm filter configuration

Criterion	Condition	Operand
Area	Equal to	Conveyor
<Add new>		

- HMI_RT_1 [WinCC Unified Scada RT]
  - Brewery
    - Bottling
      - Conveyor
      - Robot_1
      - Filling tank
      - Robot_2
      - Robot_3
    - Packaging

Alarm control_1 [Alarm control]

Name	Static value	Dynamization
Alarm control		
Alarm source	Logged alarms	None
Allow operator cont...	□	None
Authorization		
Background color	204, 204, 204	None
Connection status	None	
Current alarms	Not suppressed, Disabl...	None
Default sorting dire...	Ascending	None
Displayed alarms	Not suppressed, Disabl...	None
Filter	Area = 'HMI_RT_1.hi...	None
Flashing rate ackno...	Medium	None
Height	320	None
Icon		None
Label		None
Layer	0	

Note

You can also create filter criteria directly in runtime and use them as filters.

Result

Alarms for the selected plant object are displayed in runtime.

See also

Configuring screens for plant objects (Page 1527)

Configuring an alarm control (Page 253)

Displaying alarms for plant objects in runtime (Page 1555)

20.3.14 Configuring trend control for plant objects

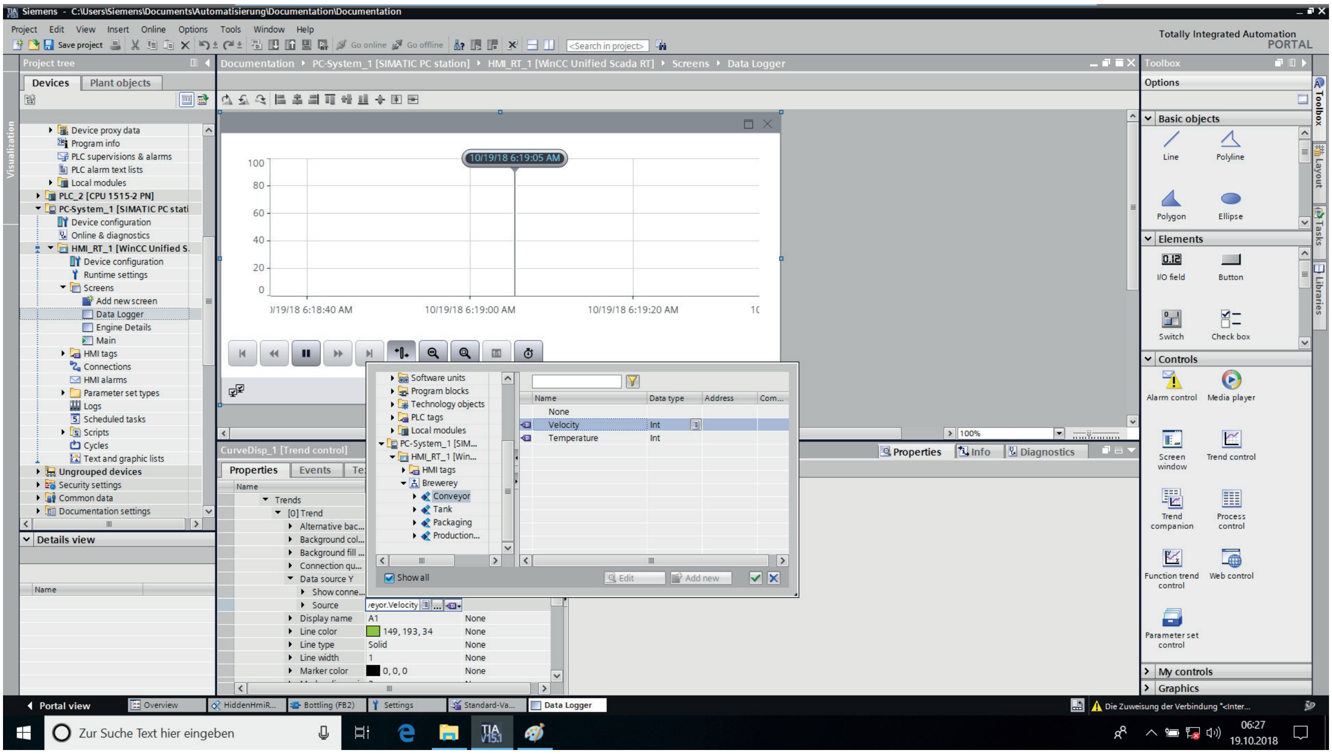
Overview

A trend control, as in the device specific configuration in a HMI device screen. Assign the plant view to your HMI device in order that the trend control can graphically represent the values of the data members of the individual plant objects in runtime.

The trend control allows you to display current and logged values for a specific time window, for example.

As with device-specific configuration, when you configure the trend control for the display of the data values you define the sources from which the values are obtained on the HMI device in runtime. The following sources are available:

- Current process values from data members of the plant object types
- Archived values from logging tags



Name	Data type	Address	Com...
None			
Velocity	Int		
Temperature	Int		

Name		
Trends		
[0] Trend		
Alternative bac...		
Background col...		
Background fill ...		
Connection qu...		
Data source Y		
Show conne...		
Source	/eyor.Velocity	
Display name	A1	None
Line color	149, 193, 34	None
Line type	Solid	None
Line width	1	None
Marker color	0, 0, 0	None

The path of the plant object is shown in the trend control when displayed in runtime.

See also

Configuring the logging of plant object types (Page 1545)

Configuring a trend control (Page 193)

Display process data of the plant objects in a trend control (Page 1550)

20.3.15 Basic information on configuring alarms

Overview

In object-oriented configuration, as in device-specific configuration via the alarming, events from the monitoring function in WinCC are displayed in form of alarms. The alarms can be acknowledged by the operator and, if necessary, logged. To do this, configure alarms that are separated into alarm classes.

For plant objects you can configure the following alarms that are used to monitor the plant:

- Discrete alarms: Display status changes
- Analog alarms: Display limit value violations (value changes),

Configure bit or analog alarms for plant object types on the basis of internal or external data members. From these plant object types you create plant objects.

If you have configured an alarm system for plant objects you can display the hierarchy path of the alarm source and the alarm status of a line or a machine in runtime, filter the alarm control by plant objects and navigate to the alarm source.

You can also filter the most frequently occurring alarms by plant object and only permit the alarm of the respective object and all lower-level objects to be displayed.

Configuration steps

In general, proceed in the following order when configuring the alarms for the plant objects:

- Configure plant object types
- Configuring bit or analog alarms for plant object types on the basis of data members.
- Creating plant objects from plant object types
- Configuring alarm control in a screen
- Configuring "Plant view" control as companion control for the alarm control.

Note

An alarm is linked to the respective plant object type. If you delete the plant object type the alarm will also be deleted.

Configuring an alarm view

The alarm view is configured for a screen. Current or logged alarms are displayed in the alarm view in runtime. More than one alarm can be displayed simultaneously, depending on the configured size. Configure the criteria for alarm filtering.

You can also configure multiple alarm views with different contents and in different screens.

See also

Configure discrete alarms for plant objects (Page 1538)

Configuring analog alarms for plant objects (Page 1541)

20.3.16 Configure discrete alarms for plant objects

Introduction

If you have configured an alarm system for plant objects you can display the hierarchy path of the alarm source and the alarm status of a line or a machine in runtime, filter the alarm control by plant objects and navigate to the alarm source.

You can also filter the most frequently occurring alarms by plant object and only permit the alarm of the respective object and all lower-level objects to be displayed.

An alarm is linked to the respective plant object type. If you delete the plant object type the alarm will also be deleted.

Requirement

- A plant object type with associated external or internal data members (with elementary data types) has been created.
- The plant structure has been assigned to a device.

Procedure

1. Select the respective data member of the plant object type on the basis of which you want to configure an alarm.
2. To create a new discrete alarm, double-click on "<Add>" under "Discrete alarms" in the table.
   A new discrete alarm is created.
3. Assign a name for the alarm.

   Note

   The name of a discrete alarm can contain up to 128 characters.

4. To configure the alarm, select "Properties > General" in the Inspector window:
   - Enter the alarm text.
   - Change the name of the alarm as required.
   - Select the alarm class.
   - Configure the priority of the alarm (a value of between "0" and "16").

Note

The alarm text must be unique in the context of the plant object type. Hierarchical information is not permitted in the alarm text.

Note

You can use the priority to sort or filter the alarms in the alarm control. With sorting by priority, you can ensure that the most important alarm (high priority) is shown in the display area in a single-line alarm control.

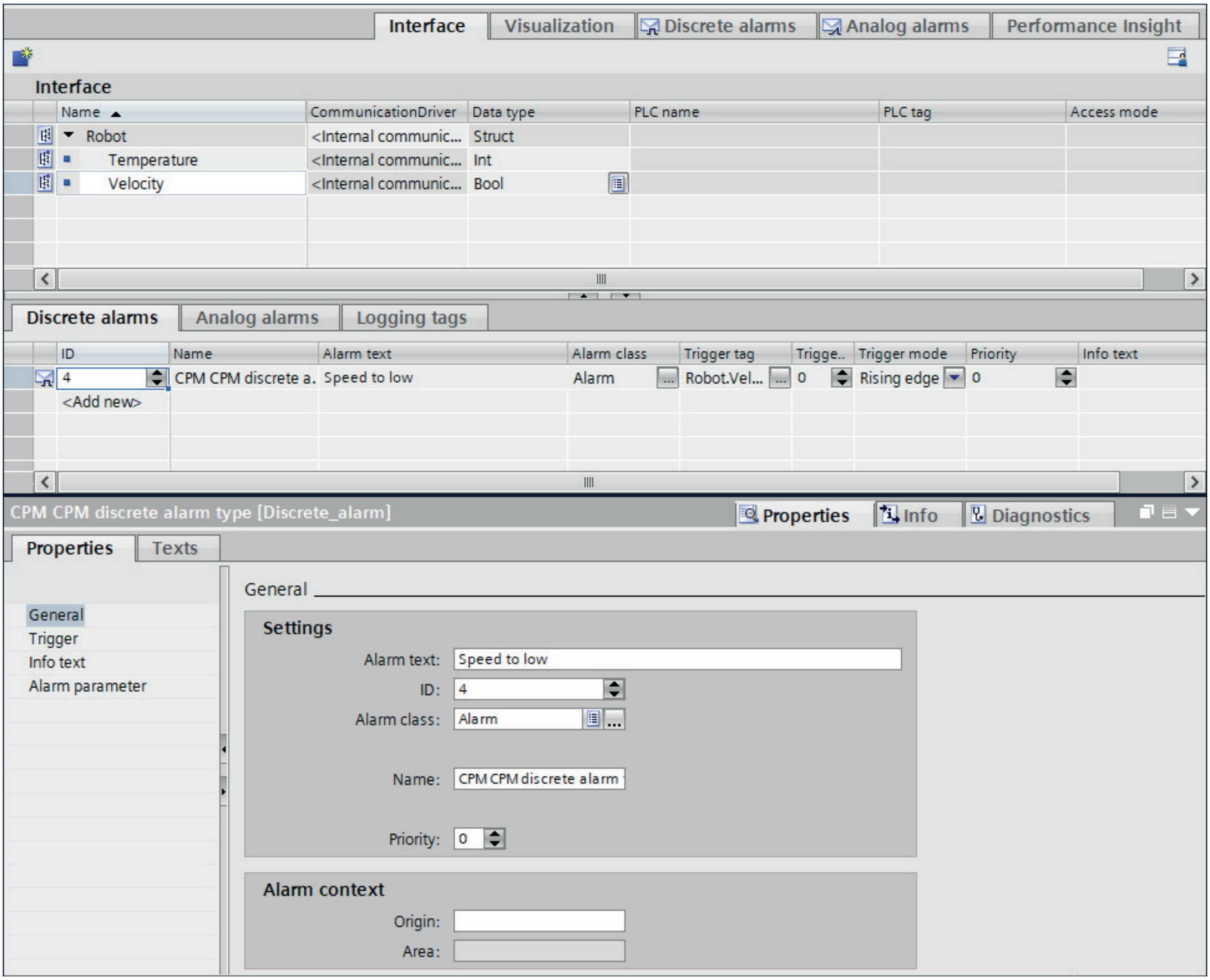
If you filter the alarm control by priority "16", only the alarms with priority "16" will appear.

5. Select "Properties > Trigger" in the Inspector window to select the tag and the bit that triggers the alarm.

Note

Only the data member of the plant object type is permitted as trigger tag.

6. Select "Trigger mode" to specify whether to trigger the alarm at a rising or falling edge.
7. To configure the alarm text, select "Properties > General > Alarm text".
   – Enter the text for the alarm under "Alarm text".



Interface

Name	CommunicationDriver	Data type	PLC name	PLC tag	Access mode
Robot	<Internal communic...	Struct			
Temperature	<Internal communic...	Int			
Velocity	<Internal communic...	Bool			

ID	Name	Alarm text	Alarm class	Trigger tag	Trigge..	Trigger mode	Priority	Info text
4	CPM CPM discrete a.	Speed to low	Alarm	Robot.Vel...	0	Rising edge	0	
<Add new>								

CPM CPM discrete alarm type [Discrete_alarm]

General

Settings

Alarm text: Speed to low
ID: 4
Alarm class: Alarm
Name: CPM CPM discrete alarm
Priority: 0

Alarm context

Origin:
Area:

See also

Configuring analog alarms for plant objects (Page 1541)

Basic information on configuring alarms (Page 1537)

Displaying alarms for plant objects in runtime (Page 1555)

20.3.17 Configuring analog alarms for plant objects

Introduction

If you have configured an alarm system for plant objects you can display the hierarchy path of the alarm source and the alarm status of a line or a machine in runtime, filter the alarm control by plant objects and navigate to the alarm source.

You can also filter the most frequently occurring alarms by plant object and only permit the alarm of the respective object and all lower-level objects to be displayed.

An alarm is linked to the respective plant object type. If you delete the plant object type the alarm will also be deleted.

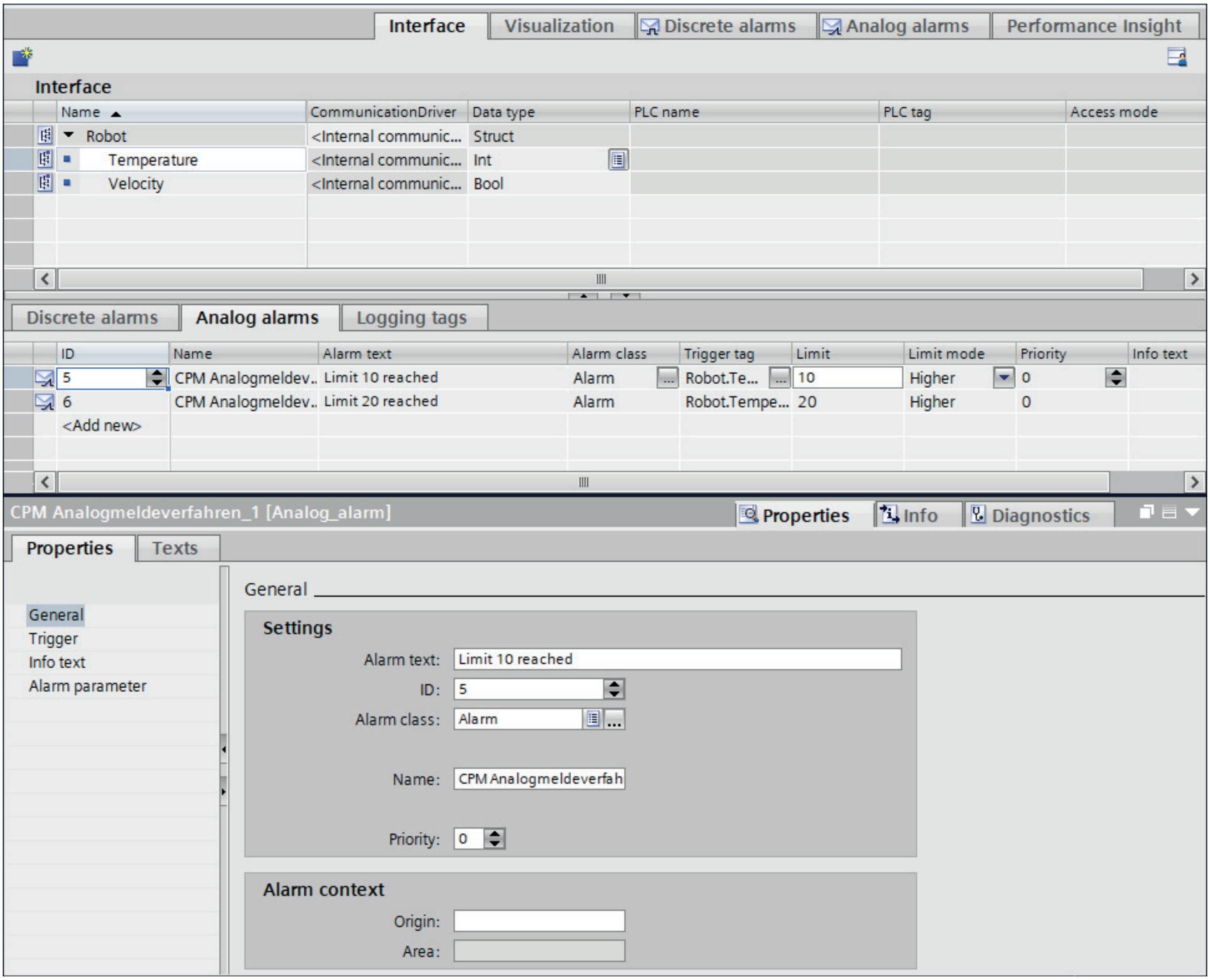
Requirement

- A plant object type with associated external or internal data members (with elementary data types) has been created.
- The plant structure has been assigned to a device.

Procedure

1. Select the respective data member of the plant object type on the basis of which you want to configure an alarm.
2. Enter the alarm text under "Properties > General".
3. To create a new analog alarm, double-click in the table on "<Add>" under "Analog alarms" in the table.
   A new alarm is displayed.

4. To configure the alarm, select "Properties > General" in the Inspector window:
   - Enter the alarm text.
   - Change the name of the alarm as required.
   - Select the alarm class.
   - Configure the priority of the alarm (a value of between "0" and "16").



Interface

Name	CommunicationDriver	Data type	PLC name	PLC tag	Access mode
Robot	<Internal communic...	Struct			
Temperature	<Internal communic...	Int			
Velocity	<Internal communic...	Bool			

ID	Name	Alarm text	Alarm class	Trigger tag	Limit	Limit mode	Priority	Info text
5	CPM Analogmeldev..	Limit 10 reached	Alarm	Robot.Te...	10	Higher	0	
6	CPM Analogmeldev..	Limit 20 reached	Alarm	Robot.Tempe...	20	Higher	0	
<Add new>								

CPM Analogmeldeverfahren_1 [Analog_alarm]

General

Settings

Alarm text: Limit 10 reached
ID: 5
Alarm class: Alarm
Name: CPM Analogmeldeverfah
Priority: 0

Alarm context

Origin:
Area:

Note

The alarm text must be unique in the context of the plant object type. Hierarchical information is not permitted in the alarm text.

Note

You can use the priority to sort or filter the alarms in the alarm control. With sorting by priority, you can ensure that the most important alarm (high priority) is shown in the display area in a single-line alarm control.

If you filter the alarm control by priority "16", only the alarms with priority "16" will appear.

5. In the Inspector window, select the tag that triggers the alarm, e.g. a data member, under "Properties > Trigger".

   Note

   Only the data member of the plant object type is permitted as trigger tag.

6. Enter a limit value in the "Value" field under "Properties > Trigger" in the Inspector window.
7. Select the trigger mode in the "Mode" field:
   – "Lower": The alarm is triggered if the limit is undershot.
   – "Upper": The alarm is triggered if the limit is exceeded.
   – "Equal": The alarm is triggered when the limit is reached.
   – "Not equal": The alarm is triggered if the limit is not reached.
   – "Lower or equal": The alarm is triggered if the limit is undershot or reached.
   – "Greater or equal": The alarm is triggered if the limit is exceeded or reached.
8. You can create additional limits for the alarm, if necessary. Note the following:
   – A tag is monitored using only one alarm type. You should therefore create either analog alarms **or** discrete alarms for a tag.
   – If the object included in the selection does not yet exist, create it in the object list and change its properties later.
9. Select the analog alarm to which you want to assign the limits.

See also

Configure discrete alarms for plant objects (Page 1538)

Temperature monitoring example Configuring analog alarms for a plant object type (Page 1544)

Basic information on configuring alarms (Page 1537)

Displaying alarms for plant objects in runtime (Page 1555)

20.3.18 Temperature monitoring example Configuring analog alarms for a plant object type

Sample scenario

The temperature of the beer brewing ingredients in a brewery must be strictly maintained. One of your tasks consists in configuring the temperature monitoring of the beer ingredient cooling.

The following requirements apply to temperature monitoring for the beer ingredients:

- The setpoint for the temperature of ingredients is 5 °C and is permitted to fluctuate by ±1 °C.
- If the temperature rises above 6 °C for longer than five minutes, the operator of the plant is notified.
- If the temperature reaches 7 °C, the operator is notified immediately. The operator has to confirm the notification.

Objective

Temperature deviations of ingredients cooling should be output on the HMI device. You plan several escalation levels for the alarms to be output according to the requirements:

- Temperature is between 4 °C and 6 °C: To prevent alarms from being triggered too frequently, no alarm is output for permitted fluctuations in temperature.
- Temperature is above 6 °C: An alarm that does not require acknowledgment is output.
- Temperature exceeds the critical temperature of 7 °C: An alarm that requires acknowledgment is output.

The temperature sensor of the beer ingredients cooling delivers analog values. Use these values to specify the triggers. The triggers determine when an alarm is triggered.

Requirement

- A trigger tag is configured for temperature monitoring, for example "TemperatureCoolingUnit".
- "Alarms" editor is open.

Create alarm for exiting the tolerance range

1. Create the alarm "Temperature is approaching the critical range".
2. Select the trigger tag.
3. Select the alarm class "Notification".
   Alarms of this alarm class do not require acknowledgment.

4. Define the trigger with a limit "5".
   This corresponds to the setpoint of 5 °C. Limits are always without units. The physical unit depends on the plant component which delivers the values.
5. Configure an absolute deadband of "1".
   This corresponds to the permitted fluctuation of ±1 °C. If the value of the trigger tag is between "4" and "6", no alarm is output.

Create alarm for when critical temperature is exceeded

1. Create the alarm "Temperature has reached the critical range".
2. Select "Alarm" as the alarm class.
   Alarms of this alarm class are displayed flashing in red on the HMI device and require acknowledgment.
3. Define the trigger with limit "7".
   This corresponds to a critical temperature of 7 °C.
4. Make sure that the value for the deadband is equal to "0".

See also

Configuring analog alarms for plant objects (Page 1541)

20.3.19 Configuring the logging of plant object types

Introduction

Save the values of the data members of the plant object types in logs for later evaluation. Alarm logging can be used to analyze error states, to optimize maintenance cycles, and to document the process.

Create a logging tag for each data member of the plant object type. These logging tags are saved in the data log of the assigned device.

You can analyze the logged tag values directly in your project, such as in a trend view, or in another user program, such as Excel.

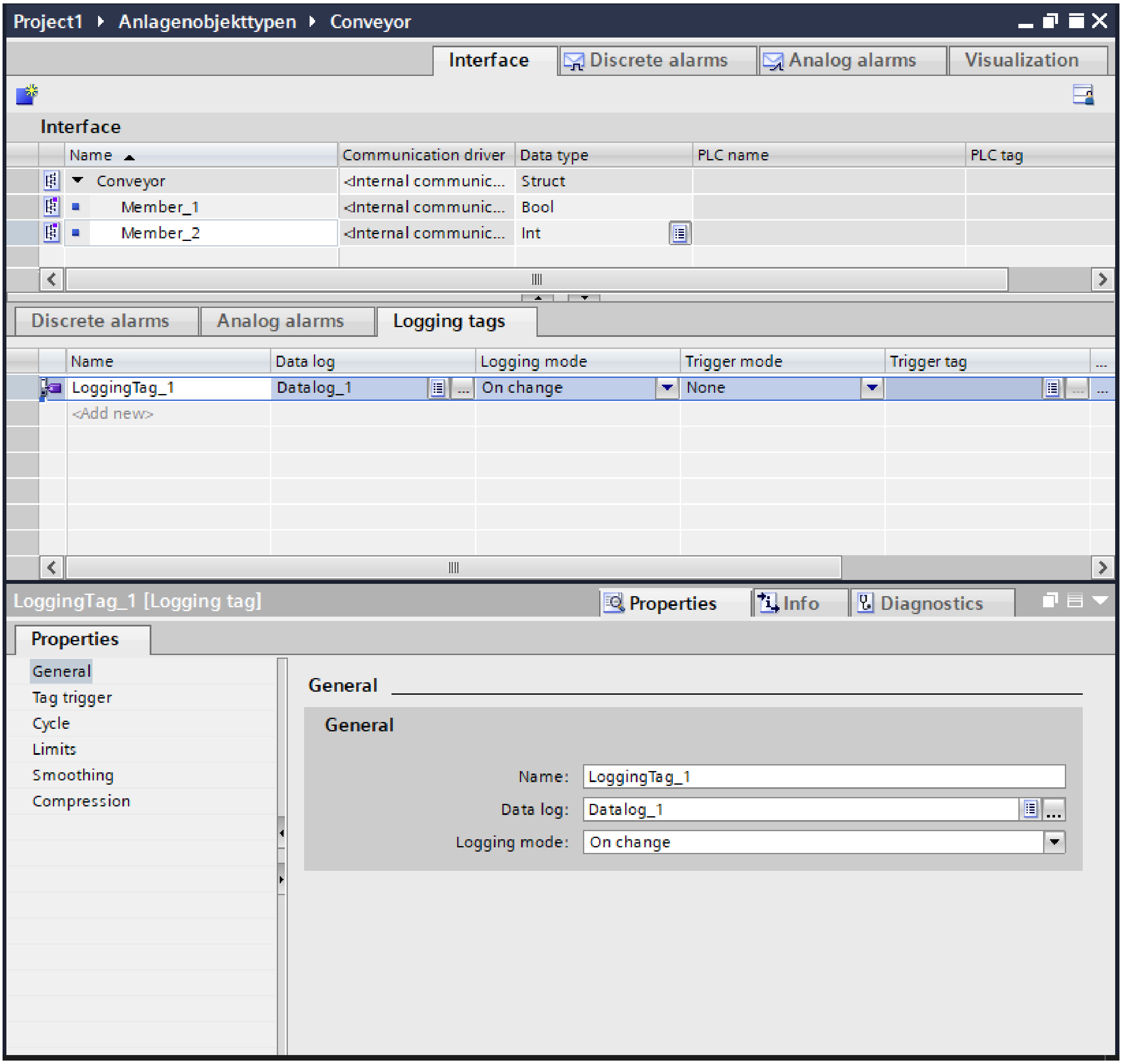
The logging tags are created for the plant object types. This means that the plant objects are automatically supplied with the logging tags of the plant object types.

Requirement

- The plant hierarchy has been created and assigned to a device.
- A plant object type with associated external or internal data members (with elementary data types) has been created.

Procedure

1. Under "Interface", jump to the "Logging tags" tab in the middle part of the work area.
2. Under "Interface", select a data member of a plant object type that you want to log.
3. Click on "Add" under "Logging tags".



Interface

Name	Communication driver	Data type	PLC name	PLC tag
▼ Conveyor	<Internal communic...	Struct		
Member_1	<Internal communic...	Bool		
Member_2	<Internal communic...	Int		

Name	Data log	Logging mode	Trigger mode	Trigger tag
LoggingTag_1	Datalog_1	On change	None	
<Add new>				

LoggingTag_1 [Logging tag]

General

Name: LoggingTag_1

Data log: Datalog_1

Logging mode: On change

A logging tag is created.

Note

A logging tag is automatically assigned to a data log. This assignment cannot be changed.

The assignment is only possible if the plant hierarchy is assigned to a HMI device.

4. If you want to reduce the number of logged values using smoothing, select the desired smoothing mode.
5. Set the limit scope and the required limits.

Note

Process values that are outside the set limit range will not be logged.

Additional information on logging tags and logging is available under "Visualizing processes with Runtime Unified".

See also

Configuring trend control for plant objects (Page 1536)

20.4 Visualizing plant objects in runtime

20.4.1 Displaying plant objects in runtime

Overview

Depending on your configuration, the following possibilities are available to you in runtime:

- Screen navigation via the plant model
- Analysis based on plant objects
- Filter alarm control by plant objects
- Display alarm status of a line and navigation to the alarm source
- Display the most frequently occurring alarms, filtered by plant object or plant object type
- Area-based access protection
- Determine the energy consumption of a line and compare with another line

You can display the configured plant hierarchy in runtime using the "Plant overview" control. If a screen window was configured as companion control for the "Plant overview", you can navigate between the screens of the plant objects and show them alternately in the screen window.

Display process data of the plant objects in a trend control. Switch between the following display modes directly in runtime:

- Device view and plant hierarchy
- Online values and log values

You can view alarms on plant objects in an alarm control.

See also

Operating "Plant overview" in runtime (Page 1548)

Display process data of the plant objects in a trend control (Page 1550)

Displaying alarms for plant objects in runtime (Page 1555)

20.4.2 Operating "Plant overview" in runtime

Introduction

Display the configured plant view in runtime using the "Plant overview" controls.

You use it to navigate to the plant objects within the plant structure and get an overview of your plant at one glance.

If you have configured screens or alarms for the lower-level plant objects and have linked them to the "Plant overview" control, display these screens and alarms in runtime.

If you have configured events for the "Plant overview" control and linked these to scripts, the scripts are called when the events occur.

An event can, for example, be linked to the operation of the buttons in the control.

Requirement

- The plant view has been created and assigned to a device.
- The "Plant overview" control and the corresponding accompanying controls are configured in the screen by the assigned device.
- Runtime is active.

Procedure

The plant view is displayed in the "Plant overview" control.

1. To display all lower-level plant objects, click [[ICON]] "Expand all".
2. To collapse all plant objects, click "Collapse all".

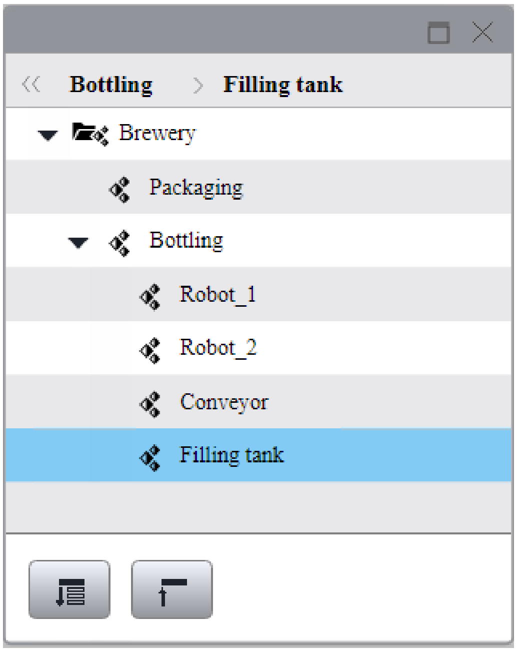
Note

The plant object that has just been selected appears in the menu bar of the "Plant overview" control: Click on the plant object to jump to the lower-level plant objects of this plant object.

3. The display the alarm control with the associated alarms for a specific plant object, double-click the alarm icon.
   The connected alarm control with the alarms for the corresponding unit opens. The plant path of the alarm source is displayed in the "Range" column.

   Note

   The alarm icon only appears an alarm has actually occurred at the respective plant object. The alarm icon disappears again when the alarm is no longer present.



4. To display the configured screen or screen window for a selected plant object, click on the respective plant object in the "Plant overview" control.

See also

Configuring screens for plant objects (Page 1527)

Displaying alarms for plant objects in runtime (Page 1555)

Display process data of the plant objects in a trend control (Page 1550)

Displaying plant objects in runtime (Page 1547)

20.4.3 Display process data of the plant objects in a trend control

Introduction

The process data or the logging data of the plant objects are displayed graphically in a trend control in runtime.

Switch directly between the following display modes in runtime.

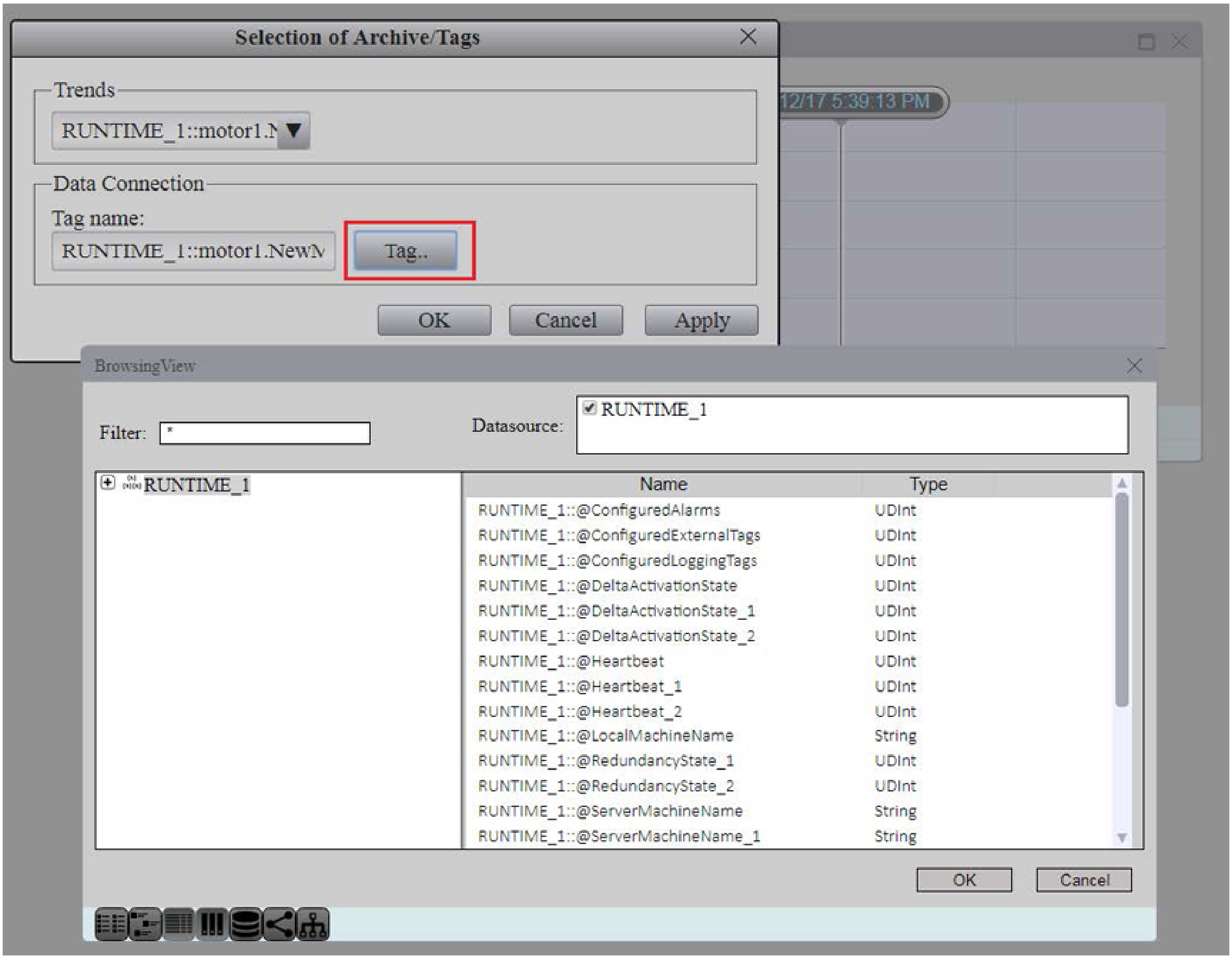
- Device view and plant hierarchy
- Online values and log values

Requirement

- The plant hierarchy has been created and assigned to a device.
- A trend control is configured in the screen by the assigned device.
- Runtime is active.

Display process data of the plant objects

1. Click "Select tags" in the trend control.
   The "Select Log/tags" dialog opens.
2. To open the list of available tags, click "Tag".
   The "Browsing view" dialog box opens.



Selection of Archive/Tags

Trends: RUNTIME_1::motor1.N

Data Connection

Tag name: RUNTIME_1::motor1.NewN

Filter: *

Datasource: ☑ RUNTIME_1

RUNTIME_1

Name	Type
RUNTIME_1::@ConfiguredAlarms	UDInt
RUNTIME_1::@ConfiguredExternalTags	UDInt
RUNTIME_1::@ConfiguredLoggingTags	UDInt
RUNTIME_1::@DeltaActivationState	UDInt
RUNTIME_1::@DeltaActivationState_1	UDInt
RUNTIME_1::@DeltaActivationState_2	UDInt
RUNTIME_1::@Heartbeat	UDInt
RUNTIME_1::@Heartbeat_1	UDInt
RUNTIME_1::@Heartbeat_2	UDInt
RUNTIME_1::@LocalMachineName	String
RUNTIME_1::@RedundancyState_1	UDInt
RUNTIME_1::@RedundancyState_2	UDInt
RUNTIME_1::@ServerMachineName	String
RUNTIME_1::@ServerMachineName_1	String

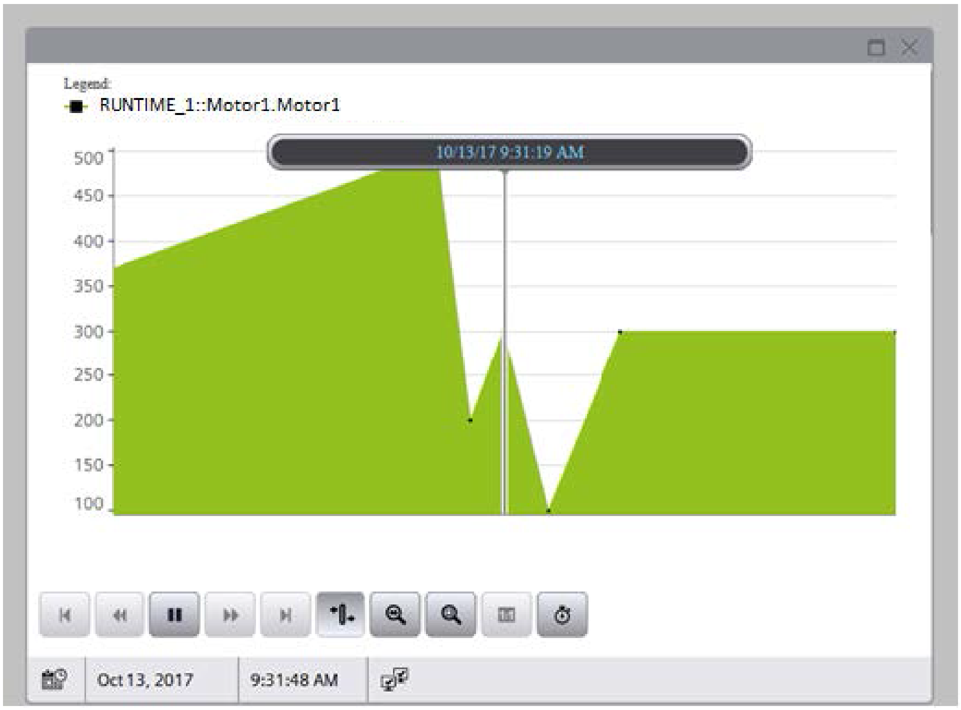
3. To jump to the plant hierarchy dialog, click on the "Plant hierarchy" icon in the toolbar.
   The plant objects and the available data members for the plant objects are displayed.

   Note

   If you have assigned a descriptive display name for the trend when you configured the trend control, only the display name is shown in runtime. All plant objects are visible at a glance in the selection list.

4. Select the respective plant object whose process data you want to display in the trend control.

5. Select the data members that you want to display as trend in the trend control.
6. Confirm with "OK".
   The process values for the selected plant object are displayed in the trend control.



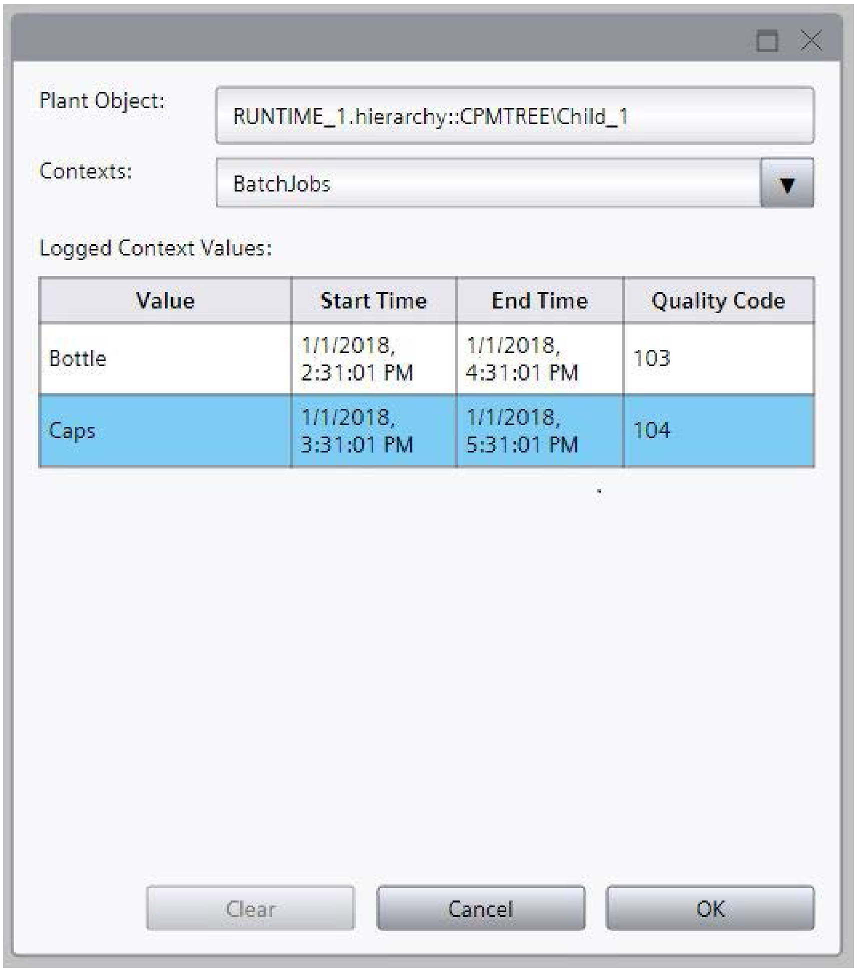
Display context data of the plant objects in a trend control

For analysis purposes, display the value range of the resulting data using the context data.

The evaluation is relevant, for example, in connection with the WinCC Unified Performance Insight to analyze the effectiveness or the fault rate of the plant.

1. In the trend control, click "Select context".
   The "Trend context" dialog opens.
2. In the "Plant objects" selection list, select the respective plant object whose data you want to display in the trend control.
3. In the "Context" selection list, select the data assigned to the plant object.
   The list of the data appears under "Logged context values".

4. Select the value that you want to display.
5. Confirm with "OK".



Plant Object: RUNTIME_1.hierarchy::CPMTREE\Child_1

Contexts: BatchJobs

Logged Context Values:

Value	Start Time	End Time	Quality Code
Bottle	1/1/2018, 2:31:01 PM	1/1/2018, 4:31:01 PM	103
Caps	1/1/2018, 3:31:01 PM	1/1/2018, 5:31:01 PM	104

The trend control displays the trends for the selected data.

See also

Operating "Plant overview" in runtime (Page 1548)

Displaying alarms for plant objects in runtime (Page 1555)

Configuring trend control for plant objects (Page 1536)

Displaying plant objects in runtime (Page 1547)

20.4.4 Displaying alarms for plant objects in runtime

Introduction

The "Alarm control" object displays alarms that occur during the production process in a plant.

Depending on your configuration, the following possibilities are available to you in runtime:

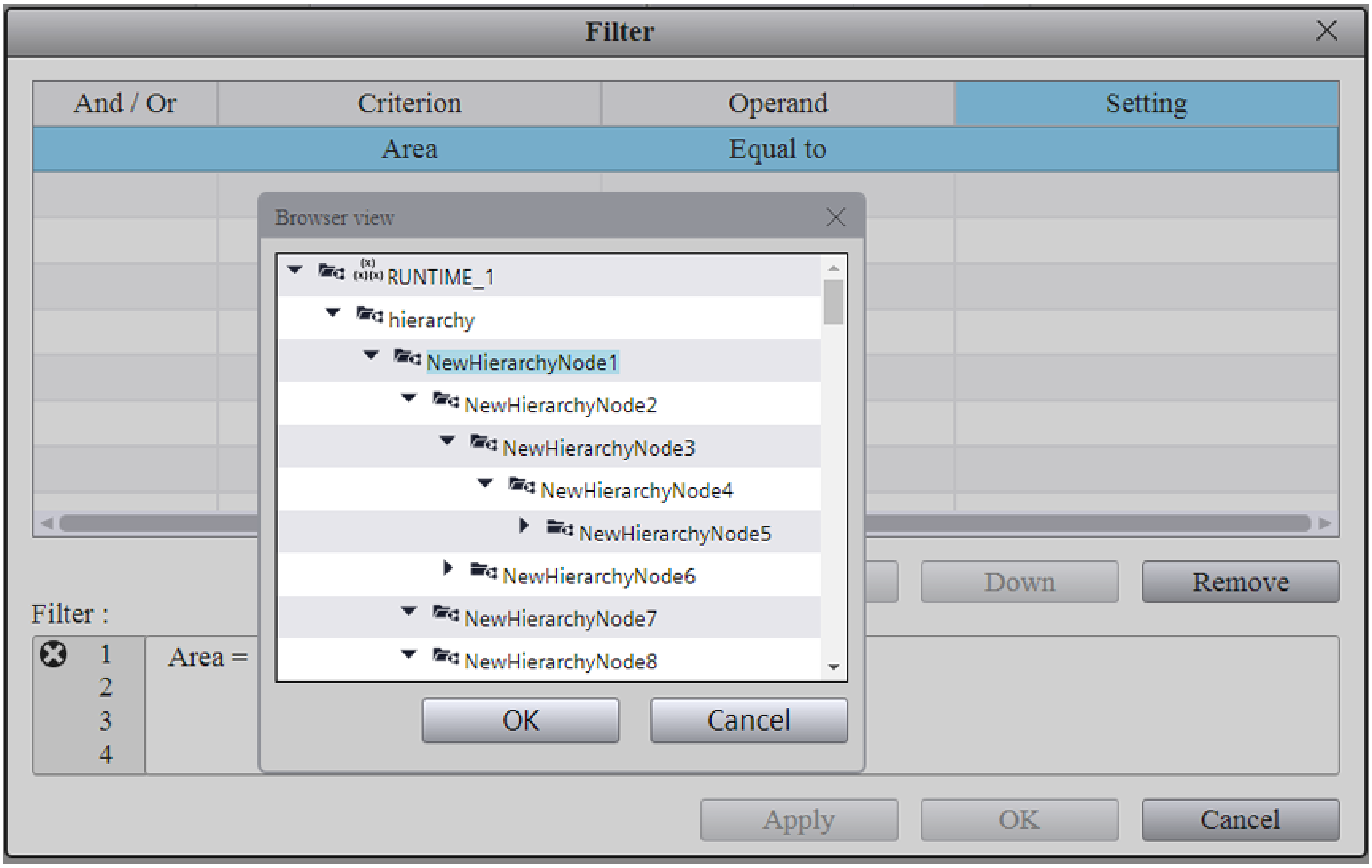
- Display hierarchy path of the alarm source
- Filter alarm control by plant objects
- Display alarm status of a line
- Navigate to the alarm source
- Display the most frequently occurring alarms, filtered by plant object or plant object type

Requirement

- The plant hierarchy has been created and assigned to a device.
- An alarm control with the filter "Range" is configured in the screen by the assigned device.
- Runtime is active.
- The alarm view has been configured.
- Runtime is active.

Filter alarms by plant objects

1. If you want to filter alarms by plant objects, click "Selection display" in runtime.
   The "Selection" dialog opens.
2. Under "Criterion", select the criterion "Range".
   All plant objects of the plant hierarchy are displayed.
3. Select the respective plant object to which you want to display the alarms.



And / Or	Criterion	Operand	Setting
	Area	Equal to	

Only alarms for the selected plant object are displayed in the alarm control.

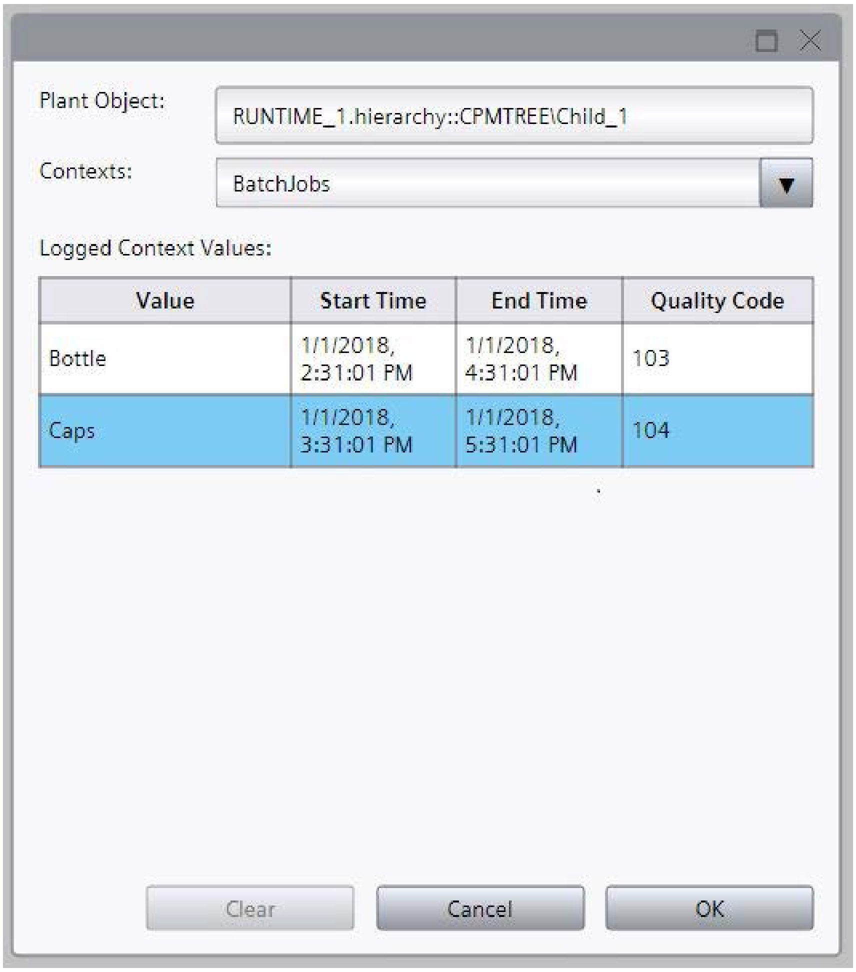
Display alarm context of the plant objects

For analysis purposes, display the value range of the resulting data using the context data.

The evaluation is relevant, for example, in connection with the WinCC Unified Performance Insight to analyze the effectiveness or the fault rate of the plant.

1. In the alarm control, click "Select context".
   The "Alarm context" dialog box opens.
2. In the "Plant objects" selection list, select the respective plant object whose data you want to display in the alarm control.
3. In the "Context" selection list, select the data assigned to the plant object.
   The list of the logging data appears under "Logged context values".

4. Select the value that you want to display.



Plant Object: RUNTIME_1.hierarchy::CPMTREE\Child_1

Contexts: BatchJobs

Logged Context Values:

Value	Start Time	End Time	Quality Code
Bottle	1/1/2018, 2:31:01 PM	1/1/2018, 4:31:01 PM	103
Caps	1/1/2018, 3:31:01 PM	1/1/2018, 5:31:01 PM	104

5. Confirm with "OK".
   The alarm control displays the alarms that match your selection.

Note

Make sure that the filter settings match the setting of the alarm context.

If no alarms appear in the alarm control, check your settings by clicking "Selection display".

See also

Operating "Plant overview" in runtime (Page 1548)

Display process data of the plant objects in a trend control (Page 1550)

Configuring an alarm control for plant objects (Page 1534)

Configuring analog alarms for plant objects (Page 1541)

Configure discrete alarms for plant objects (Page 1538)

Displaying plant objects in runtime (Page 1547)

20.5 Options

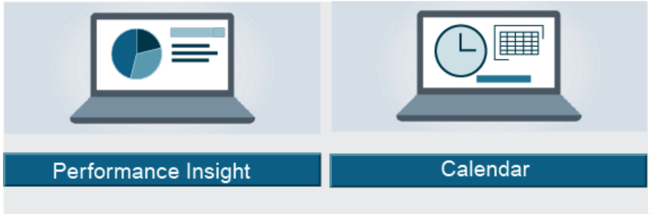
20.5.1 Plant Intelligence Options

Overview

Plant Intelligence Options offer optional enhancements to the WinCC Unified Basic System. These can be combined freely in line with your requirements.

The options allow you to plan production processes and analyze and optimize the overall effectiveness of your plant. In addition, you can design flexible production processes and coordinate complex and interlinked production processes.

Plant Intelligence options



- WinCC Unified Performance Insight
  Define, calculate and analyze plant-specific key performance indicators (KPIs) for individual aggregates, machines or entire production lines in machine-oriented or line-oriented manufacturing plants.
- WinCC Unified Calendar
  Plan, configure and manage events and actions together in a shared calendar in WinCC and combine these with WinCC tags or scripts.

Plant Intelligence Options include other options, such as coordinating and monitoring complex operations of linked machines in the production line, controlling and monitoring recipe-driven procedures, and step- and sequence-based operations.

Note

Plant Intelligence Options are successively released as add-on packages. To use the Plant Intelligence options, you require the relevant software packages and licenses.

Requirements

Please note the following requirements for using the options:

- SIMATIC WinCC Runtime Unified V16 is installed.
- STEP7 Professional V16 is installed.
- Plant Intelligence option, incl. license, is installed.
- The plant hierarchy is configured.
- License for the respective option is available.
- The configuration engineer has WinCC experience.

Unified Collaboration 21

21.1 Basics

21.1.1 Introduction

Unified Collaboration enables you to access Unified Runtime objects, such as screens of another HMI device. You can display and operate these screens.

You configure the HMI devices in the same or in different projects. The participating devices must be located in the same network and be uniquely identifiable.

You use Unified Collaboration together with SIMATIC Unified PC.

See also

Restrictions (Page 1562)

21.1.2 Requirements

To use Unified Collaboration among several PCs, the following requirements must be fulfilled:

- The following software is installed on each participating device:
  SIMATIC WinCC Unified Runtime V16 (basic package, license required)
- All required certificates are installed on each participating device, especially the collaboration certificate.

  Note

  You create and distribute the collaboration certificate using the Certificate Managers. For more information, refer to the corresponding manual.

- All devices connected via Unified Collaboration must be on the same network and have access to each other.
- Firewall settings: During the setup of TIA Portal, the components of the Unified Collaboration are released for the Windows firewall. The security release for the local subnet and can be adapted manually if required. Note the firewall rules "WinCC RTIL dist" and "WinCC RTIL proxy" in this regard.

See also

Languages in runtime (Page 772)

21.1.3 Restrictions

Note the following restrictions when using Unified Collaboration and associated Runtimes:

Screen objects

Unified Collaboration does not support all screen objects. The following screen objects cannot be used in screens that are displayed in another Runtime via Unified Collaboration:

- **Controls:**
  - Parameter set view
- **My Controls**
  - Reports
  - Plant overview

System functions and scripts

"Screen name" parameter

Screens of the "Collaboration Devices" editor cannot be used as values of the "Screen name" parameter or other parameters in system functions (e.g. ChangeScreen).

Parameters that refer to objects in the environment

System functions and scripts are executed correctly if the parameters refer to objects of the WinCC Runtime, for example:

- Screens
- Screen objects
- Tags
- Alarms
- Logs

System functions and scripts are always executed on the local HMI device if the parameters refer to objects of the environment, e.g.:

- IP addresses
- Screen brightness
- Language switching
- Logging off the user
- File operations

Example:

You have configured a script to an event of a button that processes a file stored on the local memory. The button is located in a screen that is displayed in the Runtime of another device via Unified Collaboration. If you trigger the event in runtime, the file is executed on the local HMI device and not, as intended, on the HMI device in which the button is configured.

Subsequent changes to collaboration settings

The following data must not be changed after a connection has been established:

- Collaboration name
- System ID
- IP address / Host name

Synchronicity of the system times

The system times of the HMI devices interconnected via Unified Collaboration are not allowed to differ by more than 180 seconds. If the difference is greater, no connection is established between the devices.

Configured languages

The configured and activated languages for runtime must be the same for all participating devices.

See also

Defining collaboration settings (Page 1564)

21.2 Using Unified Collaboration

21.2.1 Configuration concept

Introduction

You can use Unified Collaboration to exchange certain Runtime objects between several SIMATIC Unified Runtimes. You can display and operate a split screen of another device in a screen window.

The HMI devices can be used as follows:

- All HMI devices in the same project
- HMI devices in different projects

The configuration steps are the same for both variants.

Configuration steps

1. Create one or more projects.
2. Add several SIMATIC Unified PCs.
3. Create screens for the HMI devices.

4. Define collaboration settings.
5. Export screen references for Unified Collaboration.
6. Import screen references for Unified Collaboration.
7. Assign the external screens to the screen windows.
8. Compile and download all HMI devices.

See also

Defining collaboration settings (Page 1564)

Export screen references for Unified Collaboration (Page 1566)

Import screen references for Unified Collaboration (Page 1567)

Configuring the screen window (Page 1569)

21.2.2 Defining collaboration settings

For a device to be able to participate in Unified Collaboration, it must be uniquely identifiable. Define the following parameters in the "Collaboration" area of the Runtime settings:

- System ID
- Collaboration name
- IP address / Host name

System ID

The *system ID* must be unique for each device participating in Unified Collaboration, since this ID is used for communication between the devices.

The system ID is defined for each device in the Runtime settings in the "Collaboration" area.

The value of the system ID can be between 1 and 2046 and must be unique for the configured devices imported into the TIA Portal project.

Procedure

To determine the settings for the Unified Collaboration, follow these steps:

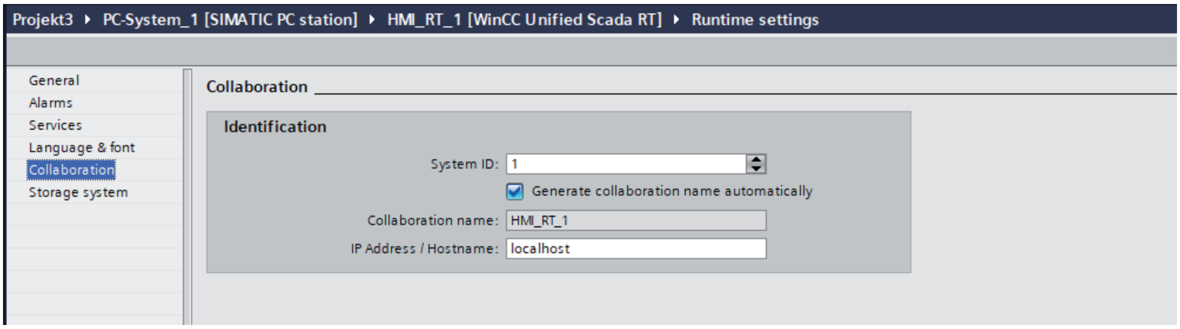
1. Open the "Devices" tab in the project navigation.
2. Open the "Runtime settings" editor of the respective HMI device.
3. Switch to the "Collaboration" area.

4. Assign a system ID for the HMI device.
   The system assigns this ID by default. If you change the ID, note that the selected ID must be unique for all devices involved in the collaboration.
   Use a number between 1 and 2046.

   Note

   Note that the system ID is also incremented separately for separate projects, and if needed, you should change the ID manually to ensure unique assignment.

5. Specify the collaboration name:
   – If "Generate collaboration name automatically" is enabled, the collaboration name corresponds to the device name.
     Changes to the device name are transferred automatically.
   – If "Generate collaboration name automatically" is disabled, assign the collaboration name manually.
     The assigned name must be unique across the system.
     The name can consist of up to 128 characters.
6. Enter the IP address or the hostname of the device. The address must correspond to the IPv4 format. All devices connected via Unified Collaboration must be in the same network and the IP address or hostname must be unique.


Projekt3 ▸ PC-System_1 [SIMATIC PC station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Runtime settings
General
Alarms
Services
Language & font
Collaboration
Storage system
Collaboration
Identification

System ID: 1

☑ Generate collaboration name automatically

Collaboration name: HMI_RT_1

IP Address / Hostname: localhost

Figure 21-1 Runtime settings for collaboration interface

Note

The following information must be different for all devices participating in Unified Collaboration:

- System ID
- Collaboration name
- IP address / Host name

See also

Restrictions (Page 1562)

21.2.3 Export screen references for Unified Collaboration

Introduction

To use Unified Collaboration, the screen references of the screens that are to be available in runtime of another HMI device must be exported.

Once this file has been imported into a project, you can use the screens for Unified Collaboration.

Requirement

- The IP address or hostname of the source device is specified under "Collaboration" in the runtime settings.
- At least one screen is configured.
- The collaboration certificate is installed on each participating device.

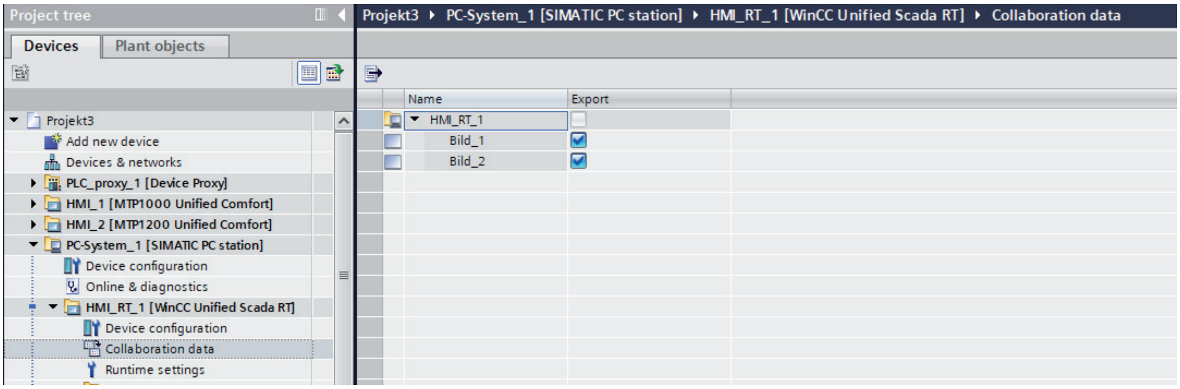
Procedure

1. Expand the folder of the device in the "Devices" tab of the project navigation.
2. Open the "Collaboration data" editor.
3. Open the device by clicking on the arrow.
   All available screens of the device are listed.
4. Select the screens you want to export. To do this, put a check mark in the "Export" column of the corresponding screen.

   Note

   When a screen is selected for export, the Unified Collaboration is activated for this device. The configuration for Unified Collaboration, including the certificates on the runtime systems, must be completed.

5. Click the export icon ➡.
   A new window opens.
6. Enter a name under which the file is to be saved.
   A message for a successful export appears.


Project tree
Devices
Plant objects
Projekt3
Add new device
Devices & networks
PLC_proxy_1 [Device Proxy]
HMI_1 [MTP1000 Unified Comfort]
HMI_2 [MTP1200 Unified Comfort]
PC-System_1 [SIMATIC PC station]
Device configuration
Online & diagnostics
HMI_RT_1 [WinCC Unified Scada RT]
Device configuration
Collaboration data
Runtime settings
Projekt3 ▸ PC-System_1 [SIMATIC PC station] ▸ HMI_RT_1 [WinCC Unified Scada RT] ▸ Collaboration data

Name	Export
HMI_RT_1	□
Bild_1	☑
Bild_2	☑

Figure 21-2 Collaboration data interface with screens

Note

Only screen references from a single device can be exported with each export operation. If you want to export the screen references of another device, switch to the device and repeat the operation.

Note

The exported xml file must not be modified manually to ensure error-free import and use of the screens in runtime.

Result

You have exported the screen references of the device. You can import the screens into any project.

21.2.4 Import screen references for Unified Collaboration

Introduction

To use Unified Collaboration, import screen references from unified HMI devices. These devices can be either HMI devices of the same project or of another project.

Requirement

- The screen references of the Unified device are exported.
- The collaboration certificate is installed on each participating device.

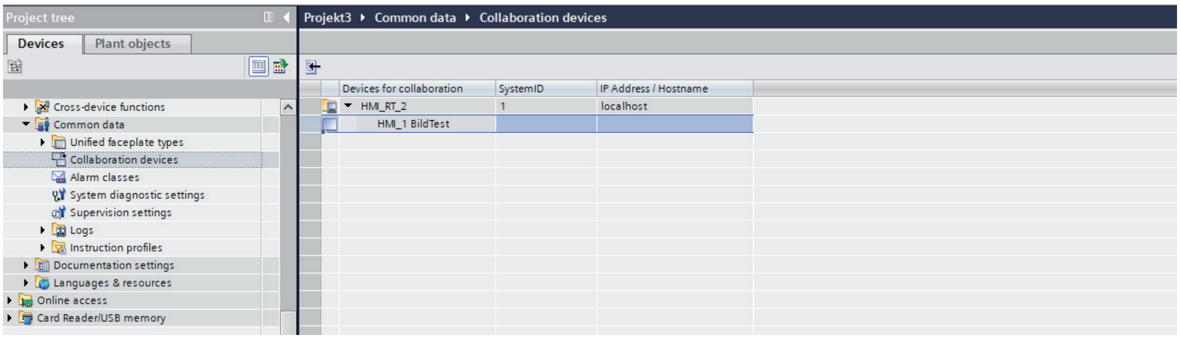
Initial import procedure

1. In the "Devices" tab of the project navigation, expand the "Shared data" folder > "Collaboration devices".
2. Open the "Collaboration Devices" editor.
3. Click the import icon.
   A new window opens.
4. Select the xml file you want to import. The file was previously exported.

   Note

   If screen references are imported, Unified Collaboration is activated for all devices of this project. The configuration for Unified Collaboration, including the certificates on the runtime systems, must be completed.

5. Confirm all dialogs.
   A message for successful import appears.
   The HMI device whose screen references you have imported now appears in the list.
6. Expand the list of the newly imported HMI device to see the screens.



Devices for collaboration	SystemID	IP Address / Hostname
HMI_RT_2	1	localhost
HMI_1 BildTest		

Figure 21-3 Collaboration device interface with imported screen

Note

Only screen references from a single device can be imported with each import process. If you want to import the screen references of another device, repeat the procedure.

Note

The exported xml file must not be modified manually to ensure error-free import and use of the screens in runtime.

Procedure for repeated import

If you want to once again import screen references that have already been imported, the procedure is the same as for the initial import. The following data must not differ from the previously imported device and must be unique across the system:

- Collaboration name
- System ID
- IP address / Host name

You see and edit this data in the "Runtime settings" of the device under "Collaboration".

If a value is used more than once, the data import cannot be performed. The import is aborted with an error message in this case.

After the successful import, the previous data is overwritten.

Deleting an imported device

To delete imported Unified devices from the list, click ✕ or use the "Delete" command on the shortcut menu. Individual screen references cannot be deleted. Attempting to delete a single screen reference will delete the entire device from the list.

Result

You have made the screens of another device available for the Runtime of an HMI device of the project and can now use them.

Note

The imported screen references of an HMI are visible to all HMI devices of the current project.

21.2.5 Configuring the screen window

Requirement

- Screens have already been configured for the Runtimes of the Unified Collaboration HMI devices.
- Export and import of the screen references are completed.
- All Runtime settings are correctly configured and the devices are connected.

Note

Note the restrictions regarding the screen objects that can be used in a screen of a Unified Collaboration device.

Procedure

1. Open the configured screen on the device on which you want to display Runtime screens of a connected Unified Collaboration device.
2. Add a screen window to the screen.
3. Open the Inspector window under "Properties > Properties > General > Screen".
4. Click the selection button in the "Static value" column.
   A new dialog opens.
5. Select the Unified Collaboration device in the left area of the window.
6. Select the screen to be displayed in the right area of the window.
7. Confirm your selection by clicking on the green check mark.

Using scripts

Alternatively, you can also configure screen windows via a script:

1. Perform steps 1 and 2 as described above.
2. Open the Inspector window.
3. Configure a script:
   - For an event
   - For an object property
4. Enter the following line:
   ```
   Screen.Windows("Screen window_1").Screen = "HMI_RT_1::Screen_1";
   ```
   Alternatively, use the snippet "Change Screen in Screen window of current screen".
5. Adapt the name of the screen window, the HMI device and the screen.
   In the example, "Screen window_1" is the screen window of the current device and "HMI_RT_1::Screen_1" is the screen of the device connected via Unified Collaboration.

Result

After compiling and downloading, Runtime displays the screen of the connected device in the screen window via Unified Collaboration.

See also

Export screen references for Unified Collaboration (Page 1566)

Restrictions (Page 1562)

Index