SIEMENS

# SINUMERIK 828D
# SINUMERIK Integrate Run
# MyScreens

**Programming Manual**

Valid for

CNC software version 4.95

**10/2020**
6FC5398-4EP40-0BA0

## 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: 6FC5398-4EP40-0BA0
Ⓟ 07/2021 Subject to change

# Table of contents

# Introduction 1

## 1.1 About SINUMERIK

From simple, standardized CNC machines to premium modular machine designs – the SINUMERIK CNCs offer the right solution for all machine concepts. Whether for individual parts or mass production, simple or complex workpieces – SINUMERIK is the highly dynamic automation solution, integrated for all areas of production. From prototype construction and tool design to mold making, all the way to large-scale series production.

Visit our website for more information SINUMERIK (https://www.siemens.com/sinumerik).

## 1.2 About this documentation

### Target group

This document addresses programmers and configuring engineers.

### Benefits

Using the Programming Manual, the target group can develop, write, test, and debug programs and software user interfaces.

### Standard scope

This documentation only describes the functionality of the standard version. This may differ from the scope of the functionality of the system that is actually supplied. Please refer to the ordering documentation only for the functionality of the supplied drive system.

It may be possible to execute other functions in the system which are not described in this documentation. This does not, however, represent an obligation to supply such functions with a new control or when servicing.

For reasons of clarity, this documentation cannot include all of the detailed information on all product types. Further, this documentation cannot take into consideration every conceivable type of installation, operation and service/maintenance.

The machine manufacturer must document any additions or modifications they make to the product themselves.

### Websites of third-party companies

This document may contain hyperlinks to third-party websites. Siemens is not responsible for and shall not be liable for these websites and their content. Siemens has no control over the information which appears on these websites and is not responsible for the content and information provided there. The user bears the risk for their use.

# 1.3 Documentation on the internet

## 1.3.1 Documentation overview SINUMERIK 828D

Comprehensive documentation about the functions provided in SINUMERIK 828D Version 4.8 SP4 and higher is provided in the 828D documentation overview (https://support.industry.siemens.com/cs/ww/en/view/109766724).

You can display documents or download them in PDF and HTML5 format.

The documentation is divided into the following categories:

- User: Operating
- User: Programming
- Manufacturer/Service: Configuring
- Manufacturer/Service: Commissioning
- Manufacturer/Service: Functions
- Manufacturer/Service: Safety Integrated
- SINUMERIK Integrate/MindApp
- Info & Training

## 1.3.2 Documentation overview SINUMERIK operator components

Comprehensive documentation about the SINUMERIK operator components is provided in the Documentation overview SINUMERIK operator components (https://support.industry.siemens.com/cs/document/109783841/technische-dokumentation-zu-sinumerik-bedienkomponenten?dti=0&lc=en-WW).

You can display documents or download them in PDF and HTML5 format.

The documentation is divided into the following categories:

- Operator Panels
- Machine control panels

- Machine Pushbutton Panel
- Handheld Unit/Mini handheld devices
- Further operator components

An overview of the most important documents, entries and links to SINUMERIK is provided at SINUMERIK Overview - Topic Page (https://support.industry.siemens.com/cs/document/109766201/sinumerik-an-overview-of-the-most-important-documents-and-links?dti=0&lc=en-WW).

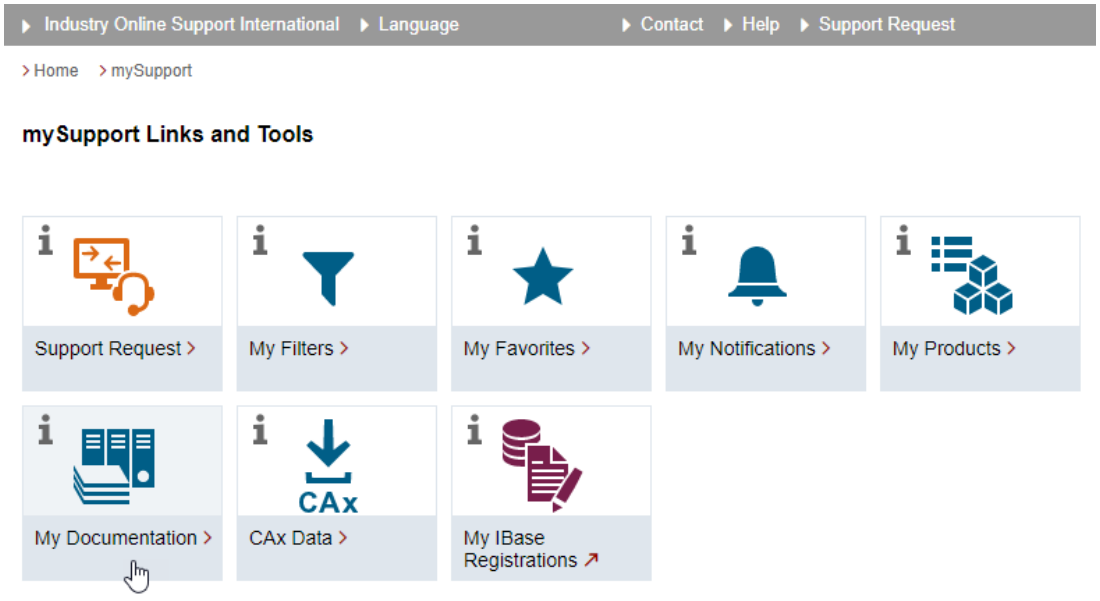
## 1.4 Feedback on the technical documentation

If you have any questions, suggestions or corrections regarding the technical documentation which is published in the Siemens Industry Online Support, use the link "Send feedback" link which appears at the end of the entry.

## 1.5 mySupport documentation

With the "mySupport documentation" web-based system you can compile your own individual documentation based on Siemens content, and adapt it for your own machine documentation.

To start the application, click on the "My Documentation" tile on the mySupport homepage (https://support.industry.siemens.com/cs/ww/en/my):



The configured manual can be exported in RTF, PDF or XML format.

**Note**

Siemens content that supports the mySupport documentation application can be identified by the presence of the "Configure" link.

## 1.6 Service and Support

### Product support

You can find more information about products on the internet:

Product support (https://support.industry.siemens.com/cs/ww/en/)

The following is provided at this address:

- Up-to-date product information (product announcements)
- FAQs (frequently asked questions)
- Manuals
- Downloads
- Newsletters with the latest information about your products
- Global forum for information and best practice sharing between users and specialists
- Local contact persons via our Contacts at Siemens database (→ "Contact")
- Information about field services, repairs, spare parts, and much more (→ "Field Service")

### Technical support

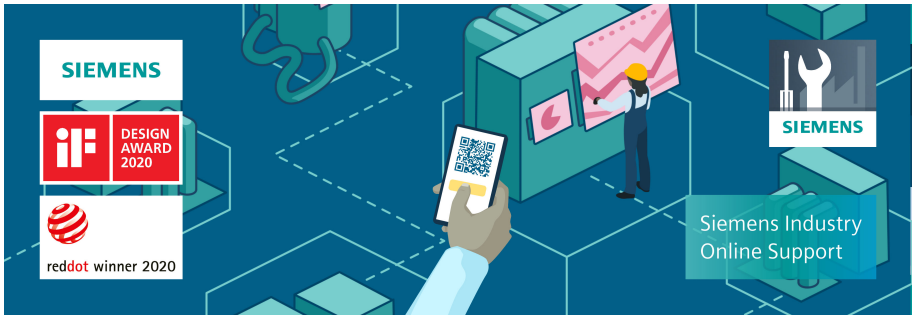
Country-specific telephone numbers for technical support are provided on the internet at address (https://support.industry.siemens.com/cs/ww/en/sc/4868) in the "Contact" area.

If you have any technical questions, please use the online form in the "Support Request" area.

### Training

You can find information on SITRAIN at the following address (https://www.siemens.com/sitrain).
SITRAIN offers training courses for automation and drives products, systems and solutions from Siemens.

### Siemens support on the go




With the award-winning "Siemens Industry Online Support" app, you can access more than 300,000 documents for Siemens Industry products – any time and from anywhere. The app can support you in areas including:

- Resolving problems when implementing a project
- Troubleshooting when faults develop
- Expanding a system or planning a new system

Furthermore, you have access to the Technical Forum and other articles from our experts:

- FAQs
- Application examples
- Manuals
- Certificates
- Product announcements and much more

The "Siemens Industry Online Support" app is available for Apple iOS and Android.

### Data matrix code on the nameplate

The data matrix code on the nameplate contains the specific device data. This code can be read with a smartphone and technical information about the device displayed via the "Industry Online Support" mobile app.

## 1.7 Important product information

### Using OpenSSL

This product can contain the following software:

- Software developed by the OpenSSL project for use in the OpenSSL toolkit
- Cryptographic software created by Eric Young.
- Software developed by Eric Young

You can find more information on the internet:

- OpenSSL (https://www.openssl.org)
- Cryptsoft (https://www.cryptsoft.com)

## Compliance with the General Data Protection Regulation

Siemens observes standard data protection principles, in particular the data minimization rules (privacy by design).

For this product, this means:

The product does not process or store any personal data, only technical function data (e.g. time stamps). If the user links this data with other data (e.g. shift plans) or if he/she stores person-related data on the same data medium (e.g. hard disk), thus personalizing this data, he/she must ensure compliance with the applicable data protection stipulations.

# Fundamental safety instructions 2

## 2.1 General safety instructions

**⚠ WARNING**

**Danger to life if the safety instructions and residual risks are not observed**

If the safety instructions and residual risks in the associated hardware documentation are not observed, accidents involving severe injuries or death can occur.

- Observe the safety instructions given in the hardware documentation.
- Consider the residual risks for the risk evaluation.

**⚠ WARNING**

**Malfunctions of the machine as a result of incorrect or changed parameter settings**

As a result of incorrect or changed parameterization, machines can malfunction, which in turn can lead to injuries or death.

- Protect the parameterization against unauthorized access.
- Handle possible malfunctions by taking suitable measures, e.g. emergency stop or emergency off.

## 2.2 Warranty and liability for application examples

Application examples are not binding and do not claim to be complete regarding configuration, equipment or any eventuality which may arise. Application examples do not represent specific customer solutions, but are only intended to provide support for typical tasks.

As the user you yourself are responsible for ensuring that the products described are operated correctly. Application examples do not relieve you of your responsibility for safe handling when using, installing, operating and maintaining the equipment.

## 2.3 Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to

an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For additional information on industrial security measures that may be implemented, please visit
https://www.siemens.com/industrialsecurity (https://www.siemens.com/industrialsecurity).

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under
https://www.siemens.com/industrialsecurity (https://new.siemens.com/global/en/products/services/cert.html#Subscriptions).

Further information is provided on the Internet:

Industrial Security Configuration Manual (https://support.industry.siemens.com/cs/ww/en/view/108862708)

**⚠ WARNING**

**Unsafe operating states resulting from software manipulation**

Software manipulations, e.g. viruses, Trojans, or worms, can cause unsafe operating states in your system that may lead to death, serious injury, and property damage.

- Keep the software up to date.
- Incorporate the automation and drive components into a holistic, state-of-the-art industrial security concept for the installation or machine.
- Make sure that you include all installed products into the holistic industrial security concept.
- Protect files stored on exchangeable storage media from malicious software by with suitable protection measures, e.g. virus scanners.
- On completion of commissioning, check all security-related settings.

# Functional scope

# 3

## 3.1 Range of functions of "Run MyScreens"

### Overview

"Run MyScreens" can be used to create user interfaces that display functional expansions designed by the machine manufacturer or user, or to implement your own layout.

Cycle calls can also be generated with user interfaces that you have created. Dialogs can be created directly on the control system.

"Run MyScreens" is implemented with an Interpreter and configuration files that describe the user interfaces.

"Run MyScreens" is configured using ASCII files: These configuration files contain the description of the user interface. The syntax that must be applied in creating these files is described in the following chapters.

### Basic configuration

The "Run MyScreens" function enables machine manufacturers to configure their own dialogs. Even with the basic configuration, it is possible to configure five dialogs in the operator menu tree or for customer-specific cycle dialogs.

**Software option**

To expand the number of dialogs, you require one of the following software options:

- SINUMERIK 828D/840D sl, SINUMERIK Integrate Run MyScreens (6FC5800-0AP64-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyHMI / 3GL (6FC5800-0AP60-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyHMI / WinCC (6FC5800-0AP61-0YB0)

### Supplementary conditions

The following conditions must be met:

- It is only possible to switch between dialogs within a single operating area.
- User variables may not have the same names as system or PLC variables (also see List Manual System Variables /PGAsl/).
- The dialogs activated by the PLC form a separate operating area (similar to measuring cycle screens).

### Tools

- UTF8-capable editor (e.g. the integrated editor of the user interface or Notepad)
- A graphics program is needed to create graphics/display images.

### File names and coding

**Note**

When using HMI Operate in the NCU, note that all file names are written in lower case letters (com, png, txt) on the CF card. This is required because of Linux.

File names are not case-sensitive on the PCU. However, it is recommended that you also use lower case letters here, in case of a possible subsequent transfer to Linux.

**Note**

When saving configuration and language files, ensure that the coding is set to UTF-8 for the editor you are using.

### Storage paths

Note the following convention when storing the configuration files, help files, etc.

| | | |
|---|---|---|
| [System siemens-directory] | | |
| | Linux: | /card/siemens/sinumerik/hmi |
| | Windows 7: | C:\ProgramData\Siemens\MotionControl\siemens |
| [System oem-directory] | | |
| | Linux: | /card/oem/sinumerik/hmi |
| | Windows 7: | C:\ProgramData\Siemens\MotionControl\oem |
| [System user-directory] | | |
| | Linux: | /card/user/sinumerik/hmi |
| | Windows 7: | C:\ProgramData\Siemens\MotionControl\user |
| [System addon-directory] | | |
| | Linux: | /card/addon/sinumerik/hmi |
| | Windows 7: | C:\ProgramData\Siemens\MotionControl\addon |
| | | |
| Storage locations | | |
| | "Run MyScreens" configuration: | [System oem-directory]/proj |
| | Configuration file: | [System oem-directory]/cfg |
| | Language files: | [System oem-directory]/lng |

| | | |
|---|---|---|
| | Image files: | [System oem-directory]/Ico/ico[resolution]<br>Example:<br>[System oem-directory]/Ico/ico640 |
| | Online help: | [System oem-directory]/hlp/[language]<br>Example:<br>[System oem-directory]/hlp/eng |

**Use**

You can implement the following functions:

- Display dialogs containing the following elements:
  - Softkeys
  - Variables
  - Texts and Help texts
  - Graphics and Help displays
- Call dialogs by:
  - Pressing the (start) softkeys
  - Selection from the PLC/NC
- Restructure dialogs dynamically:
  - Edit and delete softkeys
  - Define and design variable fields
  - Insert, exchange, and delete display texts (language-dependent or language-neutral)
  - Insert, exchange, and delete graphics
- Initiate operations in response to the following actions:
  - Displaying dialogs
  - Inputting values (variables)
  - Selecting a softkey
  - Exiting dialogs
  - Value change of a system or user variable
- Data exchange between dialogs
- Variables:
  - Read (NC, PLC and user variables)
  - Write (NC, PLC and user variables)
  - Combine with mathematical, comparison or logic operators

- Execute functions:
  - Subprograms
  - File functions
  - PI services
- Apply protection levels according to user groups

# Getting Started

# 4

## 4.1 Introduction

Using the following example, you get to know the steps necessary to insert your own dialogs into the SINUMERIK Operate user interface using Run MySreens. You also learn how to create your own dialogs, insert context-sensitive help screens and help calls, define softkeys and how you can navigate between the dialogs.

## 4.2 Project example

### 4.2.1 Task description

In the project example you create the dialogs described in the following, including a context-sensitive help.

#### Dialog 1

R parameters that can be written to (0 and 1) and geometry axis names are displayed in the first dialog (input fields). The corresponding help screens are linked in for the two R parameters. A context-sensitive help is linked in for the geometry axes. Further, the dialog includes examples for separating lines (horizontal and vertical), toggle fields, input/output fields with integrated unit selection and progress bars (with and without color change).

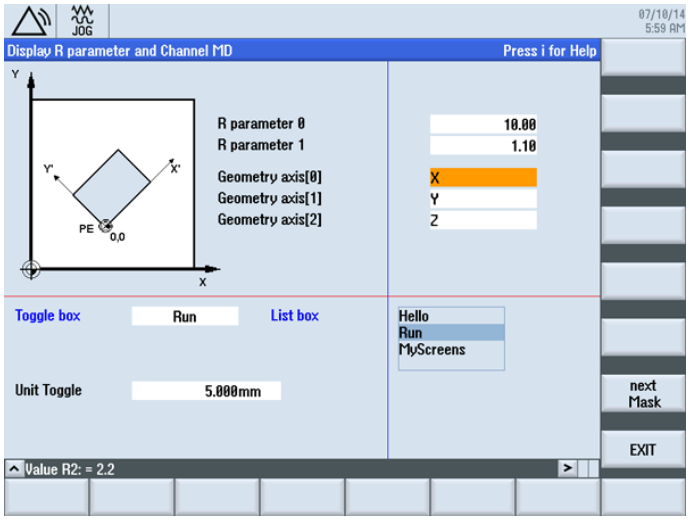



Figure 4-1 R parameters with help displays

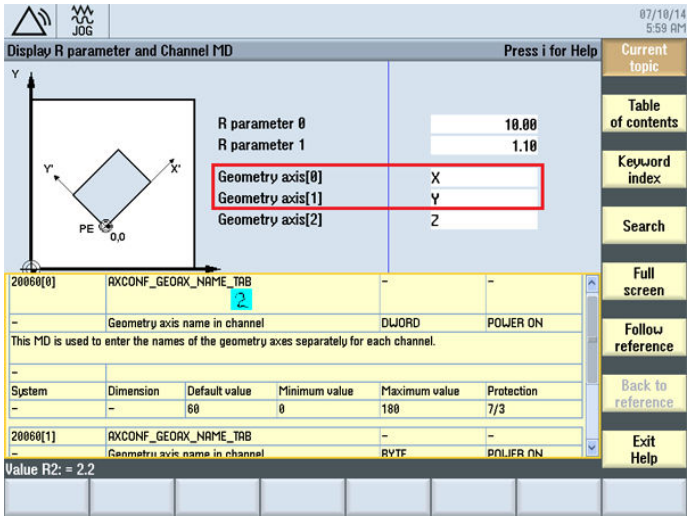



Figure 4-2 Geometry axes with context sensitive help

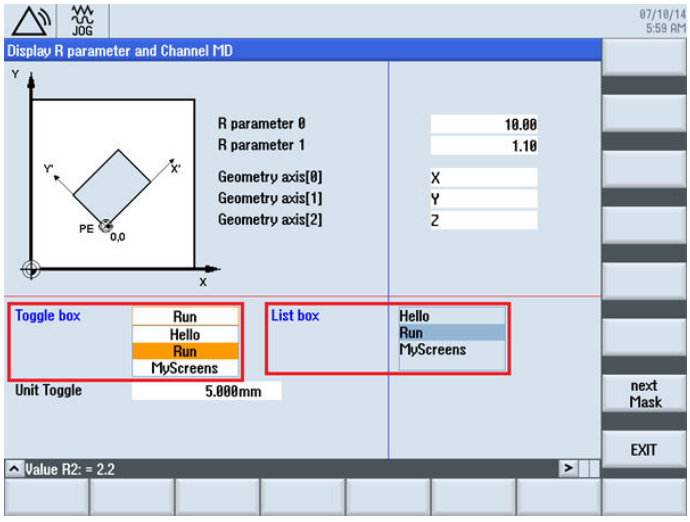



Figure 4-3 Toggle fields: List and box

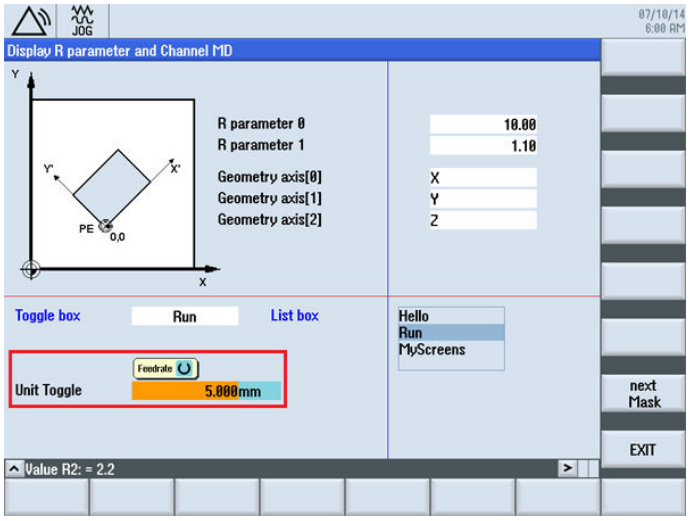



Figure 4-4 I/O field with integrated unit selection

## Dialog 2

MCS and WCS values are displayed in the second dialog. The dialog also contains examples of progress bars with and without color change.

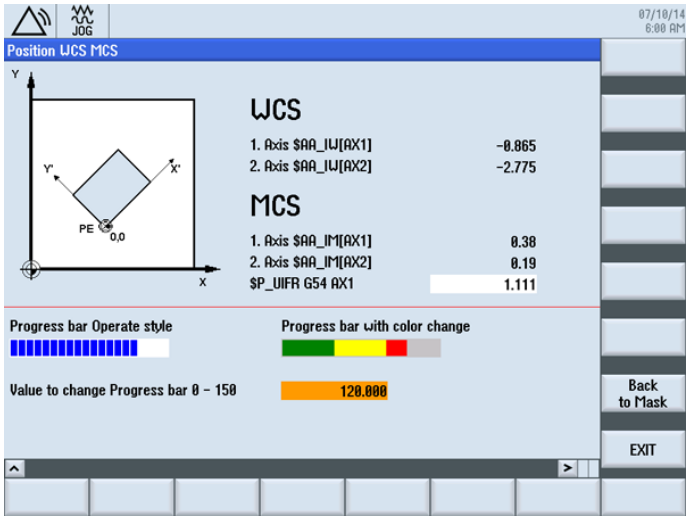



Figure 4-5 Progress bars with (right) and without (left) color change

## Navigation

The first dialog is called using the "START" softkey in the diagnostics operating area. The horizontal SK7 softkey is used.

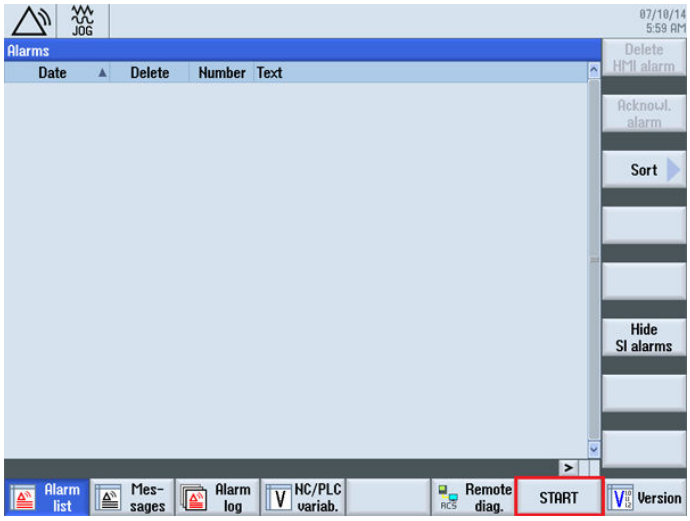



Figure 4-6 Start softkey „START" in the machine operating area, AUTO mode

You can call the second dialog from the first dialog with the "Next Mask" softkey. You can return to the root screen of the operating area by pressing the "EXIT" softkey (see the screen above).

Using the "EXIT" softkey, you can also return to the root screen of the operating area from the second dialog (see the screen above). You can return to the first dialog via the "Back to Mask" softkey.

## Procedure

The necessary steps are described in the following chapters:

1. Creating the configuration file (COM file) (Page 25)
2. Saving the configuration file in the OEM directory (Page 28)
3. Creating the online help (Page 29)
4. Integrating the online help and saving the files to the OEM directory (Page 32)
5. Copying "easyscreen.ini" into the OEM directory (Page 33)
6. Registering the COM file in "easyscreen.ini" (Page 33)
7. Testing the project (Page 33)

## 4.2.2 Creating the configuration file (example)

### Content of the configuration file

Using a UTF8-capable editor, create the configuration file "diag.com" for the two dialogs.

```
; Start identifier of start softkey
//S(START)
   ; Start softkey only text
   HS7=("START")
   ; Start softkey with language-dependent text and png
   ;HS7=([$80792,"\\sk_ok.png"])

   ; Press method
   PRESS(HS7)
      ; LM or LS function
      LM("MASK1")
      ; LM, specifying a com file
      LM("MASK1","TEST.COM")
   END_PRESS
; End identifier of start softkey
//END

; Definition of dialog 1 with header and screen
//M(MASK1/"Display R parameter and Channel MD"/"mz961_01.png")
   ; Definition of the variables
   DEF VAR1 = (R2///,"R parameter 0"///"$R[0]"/200,50,150/400,50,100,)
   ; With help screen
   DEF VAR2 = (R2///,"R parameter 1"//"mz961_02.png"/"$R[1]"/200,70,150/400,70,100)
   ; With online help
   DEF ACHS_NAM1 = (S///"Press i for Help","Geometry axis[0]"////200,100,150/400,100,100//"sinumer-
ik_md_1.html","20060[0]")
   ; With online help
   DEF ACHS_NAM2 = (S///"Press i for Help","Geometry axis[1]"////200,120,150/400,120,100//"sinumer-
ik_md_1.html","20060[1]")
   DEF ACHS_NAM3 = (S///,"Geometry axis[2]"////200,140,150/400,140,100)
   ; Definition of toggle fields and unit toggle
   DEF VAR_TGL = (S/* "Hello", "Run", "MyScreens"/"Run"/,"Toggle box"////10,230,100/120,230,100/,,6)

   DEF VAR_TGB = (S/* "Hello", "Run", "MyScreens"/"Run"/,"List box"/
dt4///250,230,100/370,230,100,60/,,"#0602ee")
   ; BC, FC, BC_ST, FC_ST, BC_GT, FC_GT, BC_UT, FC_UT, SC1, SC2
   DEF VarEdit = (R///,"Unit Toggle",,,"Feedrate"///"$R[11]"/5,300,100/120,300,100///"VarTgl"),
VarTgl = (S/*0="mm",1="inch"/0//WR2////220,300,40)
```

```
; Softkey definition in the dialog
HS1=("")
HS2=("")
HS3=("")
HS4=("")
HS5=("")
HS6=("")
HS7=("")
HS8=("")
VS1=("")
VS2=("")
VS3=("")
VS4=("")
VS5=("")
VS6=("")
VS7=("Next Mask")
VS8=("EXIT")

; Definition LOAD block
LOAD
   ; Read value with RNP
   ACHS_NAM1 = RNP("$MC_AXCONF_GEOAX_NAME_TAB[0]")
   ACHS_NAM2 = RNP("$MC_AXCONF_GEOAX_NAME_TAB[1]")
   ACHS_NAM3 = RNP("$MC_AXCONF_GEOAX_NAME_TAB[2]")
   ; Output of a dialog line
   DLGL("Value R2: = " << RNP("$R[2]"))
   ; Write WNP to a value
   WNP("$R[3]",VAR0)

  ; Vertical dividing line
  V_SEPARATOR(360,1,6,1)
  ; Horizontal dividing line
  H_SEPARATOR(220,1,7,1)
END_LOAD

; Press method
PRESS(VS7)
   ; Load an additional dialog
   LM("MASK2")
; End identifier press method
END_PRESS

; Press method
PRESS(VS8)
```

```
      ; Exit the dialog
      EXIT
   ; End identifier press method
   END_PRESS
; End identifier dialog 1
//END

; Definition of dialog 2 with header and screen
//M(MASK2/"Position WCS MCS"/"mz961_01.png")
   ; Definition of the variables
   DEF TEXT1 = (I///,"WCS"/WR0,fs2///230,30,120/,,1)
   DEF VAR1 = (R3///,"1. Axis $AA_IW[AX1]"/WR1//"$AA_IW[AX1]"/230,70,150/400,70,100)
   DEF VAR2 = (R3///,"2. Axis $AA_IW[AX2]"/WR1//"$AA_IW[AX2]"/230,90,150/400,90,100)

   DEF TEXT2 = (I///,"MCS"/WR0,fs2///230,120,120/,,1)
   DEF VAR3 = (R2///,"1. Axis $AA_IM[AX1]"/WR1//"$AA_IM[AX1]"/230,160,150/400,160,100)
   DEF VAR4 = (R2///,"2. Axis $AA_IM[AX2]"/WR1//"$AA_IM[AX2]"/230,180,150/400,180,100)
   DEF VAR5 = (R3///,"$P_UIFR G54 AX1"///"$P_UIFR[1,AX1,TR]"/230,200,150/400,200,100)

   ; Definition of the progress bar
   DEF PROGGY0 = (R/0,150//,"Progress bar Operate style"/DT2,DO0//"$R[10]"/5,240,190/5,260,150/6,10)
   DEF PROGGY4 = (R/0,150,50,100/120/,"Progress bar with color change"/DT1,DO0//"$R[10]"/
260,240,190/260,260,150/3,4,,,9,7)

   ; Definition of the variables
   DEF PROGVAL = (R/0,150//,"Value to change Progress bar 0 - 150"///"$R[10]"/5,300,230/260,300,100)

  ; Softkey definition in the dialog
  HS1=("")
  HS2=("")
  HS3=("")
  HS4=("")
  HS5=("")
  HS6=("")
  HS7=("")
  HS8=("")
  VS1=("")
  VS2=("")
  VS3=("")
  VS4=("")
  VS5=("")
  VS6=("")
  VS7=("Back to Mask")
  VS8=("EXIT")
```

```
   ; Load method
   LOAD
      H_SEPARATOR(230,1,7,1)
   END_LOAD

   ; Change method
   CHANGE(VAR5)
      ; Function PI_START
      PI_START("/NC,201,_N_SETUFR")
   ; End identifier change method
   END_CHANGE

   ; Press method
   PRESS(RECALL)
      ; Return to the calling mask
      LM("MASK1")
   ; End identifier press method
   END_PRESS

   ; Press method
   PRESS(VS7)
      ; Return to the calling mask
      LM("MASK1")
   ; End identifier press method
   END_PRESS

   ; Press method
   PRESS(VS8)
      ; Exit the dialog to the standard application
      EXIT
   ; End identifier press method
   END_PRESS
; Dialog end identifier
//END
```

### 4.2.3 Saving the configuration file in the OEM directory (example)

**Storage path**

Save the configuration file "diag.com" under the following path:

[System oem-directory]/proj

### 4.2.4 Creating the online help (example)

Create the HTML file "sinumerik_md_1.html". The example under "Content of the online help" shows the context-sensitive help call via **name="20060[0]"** and **name="20060[1]"**.

#### Notes

You can find notes on creating HTML files in Chapter Generating HTML files (Page 73).

Notes regarding the integration of online help are provided in Section Integrating the online help and saving the files to the OEM directory (example) (Page 32).

### Content of the online help

The example has no comments.

```
<html><head><meta http-equiv="Content-Type" content="text/html; charset="UTF-8"/><title></
title></head>
   <body>
      <table border="1" cellspacing="0" cellpadding="2" width="100%">
         <tr>
            <td><a name="20060[0]"><b>20060[0]</b></a></td>
            <td colspan="3">AXCONF_GEOAX_NAME_TAB</td>
            <center>
               <img src="../picture/bitmap3.png">
            </center>
            <td>-</td>
            <td>-</td>
         </tr>
         <tr>
            <td>-</td>
            <td colspan="3">Geometry axis name in channel</td>
            <td>DWORD</td>
            <td>POWER ON</td>
         </tr>
         <tr>
            <td colspan="6">This MD is used to enter the names of the geometry axes
separately for each channel.<br />
         <tr>
            <td>-</td>
            <td colspan="5"> </td>
         </tr>
         <tr>
          <td width="16*">System</td><td width="16*">Dimension</td><td width="16*">Default
value</td>
            <td width="16*">Minimum value</td><td width="16*">Maximum value</td>
            <td width="16*">Protection</td>
         </tr>
         <tr>
            <td>-</td>
            <td>-</td>
            <td>60</td>
            <td>0</td>
            <td>180</td>
            <td>7/3</td>
         </tr>
      </table>
      <p></p>
      <table border="1" cellspacing="0" cellpadding="2" width="100%">
         <tr>
            <td><a name="20060[1]"><b>20060[1]</b></a></td>
            <td colspan="3">AXCONF_GEOAX_NAME_TAB</td>
            <td>-</td>
            <td>-</td>
         </tr>
         <tr>
            <td>-</td>
            <td colspan="3">Geometry axis name in channel</td><td>BYTE</td>
            <td>POWER ON</td>
         </tr>
         <tr>
```

```
            <td colspan="6">This MD is used to enter the names of the geometry axes
separately for each channel.<br />
         <tr>
            <td>-</td>
            <td colspan="5"> </td>
         </tr>
         <tr>
            <td width="16*">System</td>
            <td width="16*">Dimension</td>
            <td width="16*">Default value</td>
            <td width="16*">Minimum value</td>
            <td width="16*">Maximum value</td>
            <td width="16*">Protection</td>
         </tr>
         <tr>
            <td>-</td>
            <td>-</td>
            <td>0</td>
            <td>0</td>
            <td>2</td>
            <td>7/3</td>
         </tr>
      </table>
      <p></p>
   </body>
</html>
```

## 4.2.5 Integrating the online help and saving the files to the OEM directory (example)

### Integrating the online help

The syntax to integrate the online help is analogous to SINUMERIK Operate.

DEF RFP=(R//1/,"RFP","RFP"///////"sinumerik_md_1.html","9006")

**Note**

As a result of LINUX, the HTML file must be written in lower case letters!

### Storage path

Save the HTML file "sinumerik_md_1.html" for the English help under the following path:

[System oem-directory]/hlp/eng

You must create a folder for additional languages as required (e.g. chs, deu, esp, fra, ita, etc.).

A list of language codes is provided in the Appendix "Reference lists".

### Further information

Detailed information about creating OEM-specific online help files is provided in Chapter Configuring the online help (Page 72).

## 4.2.6 Copying "easyscreen.ini" into the OEM directory (example)

### Storage path

Copy the file "easyscreen.ini" from the directory

[System siemens-directory]/cfg

to the directory

[System oem-directory]/cfg.

## 4.2.7 Registering the COM file in "easyscreen.ini" (example)

### Adaptation of the "easyscreen.ini"

Make the following change in the "easyscreen.ini" in the OEM directory. You have thus registered the "diag.com" configuration file.

```
;########################
;# AREA Diagnosis       #
;########################
;<=============================================================>
;< OEM Softkey on first horizontal Main Menu                    >
;< SOFTKEY position="7"                                         >
;<=============================================================>
StartFile28 = area := AreaDiagnosis, dialog:= SlDgDialog, menu := DgGlobalHu, startfile := diag.com
```

## 4.2.8 Testing the project (example)

### Testing the dialog call

Switch to the diagnostics operating area. Click the "START" horizontal softkey.

If "Run MyScreens" detects errors when interpreting the configuration files, these errors will be written to the "easyscreen_log.txt" text file. Further information can be found in Chapter Troubleshooting (log book) (Page 41).

### Testing context sensitive help screens and online help

Test the context sensitive help screens and the online help. If errors occur, then check the paths where files have been saved.

# Fundamentals 5

## 5.1 Structure of configuration file

### Introduction

The description of new user interfaces is stored in configuration files. These files are automatically interpreted and the result displayed on the screen. Configuration files are not stored in the software supplied and must be set up by the user.

A UTF-8-capable editor (e.g. the integrated editor of the user interface or Notepad) is used to create the configuration files and language files. The description can also be explained using comments. A ";" is inserted as comment character before every explanation.

**Note**

When saving configuration and language files, ensure that the coding is set to UTF-8 for the editor you are using.

However, keywords - also in an UTF-8-coded COM file - may only comprise characters that are contained in the ASCII character set. Otherwise the interpretation and therefore the display of the screens/dialogs cannot be guaranteed.

### Configuration

Each HMI operating area has permanent start softkeys which can be used to access newly generated dialogs.

In the event of a "Load a screen" (LM) or a "Load softkey menu" (LS) call in a configuration file, a new file name containing the object called can be specified. This makes it possible to structure the configuration, e.g. all functions in one operating level in a separate configuration file.

### Structure of the configuration file

A configuration file consists of the following elements:

1. Description of the start softkeys
2. Definition of the dialogs
3. Definition of the variables
4. Description of the methods
5. Definition of the softkey menus

**Note**

**Sequence**

The specified sequence in the configuration file must be maintained.

Example:

```
//S (START)                  ; Definition of the start softkey (optional)
....
//END
//M (.....)                  ; Definition of the dialog
DEF .....                    ; Definition of the variables
LOAD                         ; Description of the blocks
...
END_LOAD
UNLOAD
...
END_UNLOAD
...
//END
//S (...)                    ; Definition of a softkey menu
//END
```

### Storing the configuration files

The configuration files are stored in the directory [System user-directory]/proj and accordingly also in the directories [System addon-directory] and [System oem-directory].

### Converting texts from other HMI applications

Procedure to convert a text file with code page coding to text-coding UTF-8:

1. Open the text file on a PG/PC in a text editor.
2. When saving, set the UTF-8 coding.

The read-in mechanism via code page code is still supported.

## 5.2 Structure of the menu tree

### Menu tree principle

Several interlinked dialogs create a menu tree. A link exists if you can switch from one dialog to another. You can use the newly defined horizontal/vertical softkeys in this dialog to call the preceding or any other dialog.

A menu tree can be created behind each start softkey:

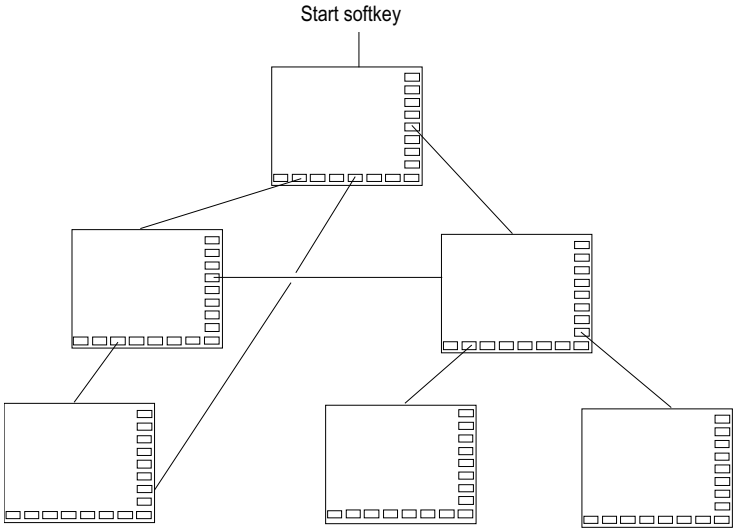



Figure 5-1 Menu tree

## Start softkeys

One or more softkeys (start softkeys), which are used to initiate your own operating sequences, are defined in the configuration files specified in "easyscreen.ini".

The loading of a dedicated dialog is associated with a softkey definition or another softkey menu. These are then used to perform the subsequent actions.

Pressing the start softkey loads the assigned dialog. This will also activate the softkeys associated with the dialog. Variables will be output to the standard positions unless specific positions have been configured.

## Reverting to the standard application

You can exit the newly created dialog and return to the standard operating area.

You can use the <RECALL> key to close new user interfaces if you have not configured this key for any other task.

**Note**

**Calling dialogs in the PLC user program**

Dialogs can be selected from the PLC as well as via softkeys: An interface signal is available in DB19.DBB10 for signal exchange between the PLC → HMI.

# 5.3 Definition and functions for start softkeys

## 5.3.1 Defining the start softkey

### Dialog-independent softkey

Start softkeys are dialog-independent softkeys which are not called from a dialog, but which have been configured **before** the first new dialog. In order to access the start dialog or a start softkey menu, the start softkey must be defined.

### Programming

The definition block for a start softkey is structured as follows:

```
//S(Start)                    ; Start identifier of start softkey
HS1=(...)                     ; Define the start softkey: horizontal SK 1
PRESS(HS1)                    ; Method
  LM...                       ; LM or LS function
END_PRESS                     ; End of method
//END                         ; End identifier of start softkey
```

### Permissible positions for start softkeys

The following positions for "Run MyScreens" start softkeys are permissible in the operating areas:

| Operating area | Position |
|---|---|
| Machine | HSK6 |
| Parameter | HSK7 |
| Program | HSK6 and HSK15<br>Measuring cycles: HSK13 and HSK14 |
| Program Manager | HSK2-8 and HSK12-16, if not assigned to drives. |
| Diagnostics | HSK7 |
| Commission | HSK7 |

Start softkeys are configured in special files. The names of these files are stated in the "easyscreen.ini" file. They usually have a name which is specific to an operating area (e.g. "startup.com" operating area for the startup area). The machine operating area represents an exception. There are several mode-specific files in the machine operating area ("ma_jog.com", "ma_auto.com").

The softkey menu with the start softkeys is called "Start". Existing configurations for start softkeys can still be used. The function where start softkeys are merged with the softkeys for the respective HMI application (operating area) in the start softkey menu is not supported.

This means that until the first dialog call is made - in other words, the time at which full functionality becomes available (e.g. execution of PRESS blocks) - menus or softkey menus can only be replaced by others in their entirety.

### Template for configuration of the start softkey

A detailed description of all permissible positions for start softkeys and their configuration is located in the "easyscreen.ini" file in the following directory:

[System siemens-directory]/cfg

This file is used as a template for your own configuration of the start softkey.

### See also

Lists of the start softkeys

## 5.3.2 Functions for start softkeys

### Functions for dialog-independent softkeys

Only certain functions can be initiated with start softkeys.

The following functions are permitted:

- The **LM function** can be used to load another dialog: **LM**("Identifier"["File"])
- The **LS function** can be used to display another softkey menu: **LS**("Identifier"[, "File"][, Merge])
- You can use the **"EXIT" function** to exit newly configured user interfaces and return to the standard application.
- You can use the **"EXITLS" function** to exit the current user interface and load a defined softkey menu.

### PRESS method

The softkey is defined within the definition block and the "LM" or "LS" function is assigned in the PRESS method.

If the start softkey definition is designated as a comment (semicolon (;) at beginning of line) or the configuration file removed, the start softkey will not function.

```
//S(Start)                          ; Start identifier
HS6=("1st screen")                  ; Label horizontal SK 6 with "1st screen"
PRESS(HS6)                          ; PRESS method for horizontal SK 6
  LM("Screen1")                     ; Load screen1 function, where Screen 1 must be
                                    defined within the same file.
```

```
END_PRESS                          ; End of PRESS method
HS7=("2nd screen")                 ; Label horizontal SK 7 with "2nd screen"
PRESS(HS7)                         ; PRESS method for horizontal SK 7
  LM("Screen2")                    ; Load Screen2 function, where Screen2 must be de-
                                   fined within the same file.
END_PRESS                          ; End of PRESS method
//END                              ; End identifier of entry block
```

## Example

```
HS1 = ("new softkey menu")
HS2 = ("no function")
PRESS (HS1)
  LS("Menu1")                        ; Load new softkey menu
END_PRESS
PRESS (HS2)                          ; Empty PRESS method
END_PRESS
```

## Various configurations of the start softkeys

Various configurations of the start softkeys are merged. In this case, initially the name of the file to be interpreted is read-out of "easyscreen.ini". A search is made for *.com files in the following directories:

- [System user-directory]/proj
- [System oem-directory]/proj
- [System addon-directory]/proj
- [System siemens-directory]/proj

The configurations included for the start softkeys are now merged to form a configuration, i.e. the individual softkeys are compared. If there are two or more configurations for a softkey, the higher order configuration is always used in the merge version.

Softkey menus or dialogs that are possibly included are ignored. If a softkey has a command without file information, e.g. LM ("test"), as the required softkey menu or dialog is contained in the same file, then the corresponding file name is supplemented in the internal merge version so that in this case, no changes are required. The merge configuration obtained is then subsequently displayed.

# 5.4 Troubleshooting (log book)

## Overview

If "Run MyScreens" detects errors when interpreting the configuration files, these errors will be written to the "easyscreen_log.txt" text file. Only errors of the currently selected dialog are entered. Error entries from previously selected dialogs are deleted.

The file contains the following information:

- The action during which an error occurred.
- The line and column number of the first faulty character.
- The entire faulty line of the configuration file.
- The entries made by the DEBUG function.

**Note**

Each entry has a prefix with the current time stamp in square brackets. This can be helpful, for example, when analyzing time-critical configurations.

## Saving "easyscreen-log.txt"

The "easyscreen_log.txt" file is saved in the following directory:

[System user-directory]/log

## Syntax

The system does not start to interpret the syntax until the start softkey has been defined and a dialog with start and end identifiers as well as a definition line has been configured.

```
//S(Start)
HS6=("1st screen")
PRESS(HS6)
  LM("Maske1")
END_PRESS
//END

//M(Maske1)
  DEF Var1=(R)
  DEF VAR2 = (R)
  LOAD
  VAR1 = VAR2 + 1             ; Error message in logbook, as VAR2 has no value
...
//END
```

```
; correct way would be, for
example:
//M(Maske1)
   DEF Var1=(R)
   DEF VAR2 = (R)

   LOAD
      VAR2 = 7
      VAR1 = VAR2 + 1 ;
...
```

## 5.5 Notes on "easyscreen.ini"

As of SINUMERIK Operate V4.7, "easycreen.ini" has been extended by the entries described in this chapter. The "easyscreen.ini" can be found in the [System siemens-directory]/cfg.

### Help screen start position

**Entry in "easyscreen.ini":**

```
[GENERAL]
HlpPicFixPos=true
```

**Note:**

The start position of help displays is positioned at the configured pixel position independent of the resolution (default=true).

### Elongation behavior, default line height and line spacing

**Entry in easyscreen.ini:**

```
[GENERAL]
SymmetricalAspectRatio=false
DefaultLineHeight=18
DefaultLineSpacing=3
```

**Notes:**

- The "SymmetricalAspectRatio" entry determines whether the same aspect ratio is used in the X and Y direction when adapting a configuration to a specific screen resolution.
  - "false" (default): Fields and graphics are shortened in widescreen resolution in the Y direction (asymmetrical elongation in relation to 640x480). For example, a square configured in 640x480 is shortened vertically for a widescreen panel and becomes a rectangle.
  - "true": The same aspect ratio is used in the X and Y direction and fields and graphics retain their originally configured proportions in relation to 640x480. For example, a square configured in 640x480 remains a square in a widescreen panel.
- The entries "DefaultLineHeight" and "DefaultLineSpacing" can be used to specify the default line height (default: 18 pixels) and the default line spacing (default: 3 pixels) in relation to 640x480. They only take effect when no Y position or height is specified in the configuration for the position of the short text or the input/output field.

## Cycles in the basic display Machine

- Entry to machine operating mode JOG via HS6 with the possibility to generate a cycle call into the Machine main screen with section [JOBSHOPINTEGRATION]:
  ```
  [JOBSHOPINTEGRATION]
  Integration = true
  ```
  or via the starting block in the configuration
  ```
  LM("Screen1",,1)
  PRESS(VS8)
     GC("MOVE_RIDE") ; cycle call is being generated
     EXIT
  END_PRESS
  ```
- Retention of the operate menus with section [Integration]:
  If the dialog only includes vertical softkeys, the horizontal standard softkey bar incl. the start softkey is retained when the dialog is displayed. If the dialog contains horizontal and vertical softkeys, the horizontal bar from the dialog replaces the horizontal bar with the start softkey.
  ```
  [Integration]
  OperateMenusEnabled = true
  ```

## Resolution-dependent screen position: Form panels

**Entry in "easyscreen.ini":**

```
[640x480]
MyPanel = x:=0, y:=220, width:=340, height:=174
[800x480]
MyPanel = x:=0, y:=220, width:=420, height:=174
...
```

**Notes:**

The screen position can be defined as follows:

- The screen position can be specified in "Pixels in relation to 640x480".
  Example:
  `//M(MyMask/"MyCaption"/"\\myhelp.png"/`**`0,219,335,174`**`)`
- The screen position can be coupled to the resolution-dependent position of a form panel from an Operate standard screen layout, e.g. "FormPanel4" ("auxiliary functions") from the screen layout "slmastandardscreenlayout.SlMaStandardScreenLayout" of the "Machine" area
  `//M(MyMask/"MyCaption"/"\`
  `\myhelp.png"/`**`"slmastandardscreenlayout.SlMaStandardScreenLayout.FormPanel4"`**
  `)`

  It is therefore easy to superimpose standard forms from Operate or position Easyscreen screens precisely at their position.
  Example:
  `//M(Mask/"Mask"//`**`"slstandardscreenlayout.SlStandardScreenLayout.LowerForm"`**`)`
  Any other screen layout can also be used.
- The screen position can be coupled to a self-defined, resolution-dependent definition of form panels in the "easyscreen.ini" file.
  Example:
  `//M(MyMask/"MyCaption"/"\\myhelp.png"/`**`"MyPanel"`**`)`

## 5.6 Notes for personnel changing over to "Run MyScreens"

**Note**

When using HMI Operate in the NCU, note that all file names are saved in lower case letters on the CF card (com, png, txt).

**Note**

Make sure that the coding is set to UTF-8 in the editor that you are using when saving the configuration and language files.

### Image files

Always save image files in the PNG format "*.png".

The data must be saved, e.g. for OEM modifications, under

[System oem-directory]/ico/[resolution].

Further information can be found in Chapter Using display images/graphics (Page 63).

### Adapting the configuration file

Check your configuration files regarding the following points:

- Compare the start softkeys with the currently permissible softkeys and adapt them, if required.
- Rename the integrated image files corresponding to the "Image files" point above.

The data is stored, e.g. for OEM modifications, in the following directory:

[System oem-directory]/proj

[System user-directory]/proj A

[System addon-directory]/proj

### Adapting the help files

All help files must be saved in UTF-8. Check your existing files, and resave them accordingly with a suitable editor.

The HTML files are saved in the following directory, e.g. for English:

[System oem-directory]/hlp/eng

[System user-directory]/hlp/eng

[System addon-directory]/hlp/eng

You must create the directories for additional languages corresponding to the language codes.

### Checking the "Run MyScreens" license

Check whether the number of dialogs that you have entered exceeds the basic scope of a maximum of five dialogs.

To expand the number of dialogs, you require one of the following software options:

- SINUMERIK 828D/840D sl, SINUMERIK Integrate Run MyScreens (6FC5800-0AP64-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyScreens + Run MyHMI (6FC5800-0AP65-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyHMI / 3GL (6FC5800-0AP60-0YB0)
- SINUMERIK 840D sl, SINUMERIK Integrate Run MyHMI / WinCC (6FC5800-0AP61-0YB0)

## 5.7 Extended configuration syntax

As of SINUMERIK Operate V4.7, a simplified syntax is available for the definition of screens, variables, softkeys and for table columns. Reading and servicing are improved with this alternative syntax. The properties and attributes can be specified in any order, empty entries are omitted. Instead of with round brackets "**(**" and "**)**" as in the previous syntax, the listing of the properties and attributes is now enclosed in curly brackets "**{**" and "**}**".

The properties and attributes are specified as follows:
`{<Name> = <Value>, <Name> = <Value>, …}`

The previous syntax remains compatible.

## Extended syntax for the definition of screens

//M {<Mask name> [,**HD**=<Title>] [,**HLP**=<Graphic>] [,**X**=<X position>] [,**Y**=<Y position>] [,**W**=<Width>] [,**H**=<Height>] [,**VAR**=<> System or user variable] [,**HLP_X**=<X position help screen] [,**HLP_Y**=<Y position help screen>] [,**CM**=<Column alignment>] [,**CB**= Response when opening the dialog] [,**XG**=<Interpret help screen as X3d graphic>] [,**PANEL**=<Name of the linked FormPanel>][,**MC**=<Mask background color:>][,**HD_AL**=<Alignment of the mask header>] [,**LANGFILELIST**=<List of the mask-specific language files>]}

**Example:**
```
//M{VariantTest, HD="My Mask"}
```

## Extended syntax for the definition of variables

DEF <Variable name> = {[**TYP**Type] [,**MIN**=<Minimum value>] [,**MAX**=<Maximum value>] [,**TGL**=<Toggle values>] [,**VAL**=<Preassignment>] [,**LT**=<Long text>] [,**ST**=<Short text>] [,**GT**=<Graphic text>] [,**UT**=<Unit text>] [,**TT**=<ToolTip text>] [,**TG**=<Toggle option>] [,**WR**=<Input mode>] [,**AC**=<Access level>] [,**AL**=<Text alignment>] [,**FS**=<Font size>] [,**LI**=<Limit value handling>] [,**UR**=<Refresh rate>] [,**CB**=<Text align when opening the dialog>] [,**HLP**=<Help screen>] [,**VAR**=<System or user variable>] >] [,**TXT_X**=<X position short text>] [,**TXT_Y**=<Y position short text>] [,**TXT_W**=<Width short text>] [,**TXT_H**=<Height short text>] [,**X**=<X position input/output field>] [,**Y**=<Y position input/output field>] [,**W**=<Width input output field>] [,**H**=<Height input/output field>] [,**UT_DX**=<Distance between input field and unit field>] [,**UT_W**=<Width unit field>] [,**BC**=<Background color input/output field>] [,**FC**=<Foreground color input/output field>] [,**BC_ST**=<Background color short text>] [,**FC_ST**=<Foreground color short text>] [,**BC_GT**=<Background color graphic text>] [,**FC_GT**=<Foreground color graphic text>] [,**BC_UT**=<Background color unit text>] [,**FC_UT**=<Foreground color unit text>] [,**SC1**=<Signal color 1 for progress bar>] [,**SC2**=<Signal color 2 for progress bar>] [,**SVAL1**=<Threshold value 1 for progress bar>] [,**SVAL2**=<Threshold value 2 for progress bar>] [,**DT**=<Display type>] [,**DO**=<Display alignment>] [,**OHLP**=<Online help>][,**LINK_TGL**=<Name of the linked toggle v]}

**Examples:**

```
DEF MyVar5={TYP="R2", ST="MyVar5", VAL=123.4567, OHLP="myhelp.html", MIN=100.1,
MAX=200.9}
DEF MyVar2={TYP="I", TGL="*1,2,3", VAL=1}
DEF MyVar3={TYP="R2", TGL="*0=""Off"", 1=$80000", VAL=1}
DEF MyVar4={TYP="R2", TGL="*MyArray",VAL=1}
DEF MyVar1={TYP="R2", TGL="%grid99", X = 0, W=300, H=200}
DEF MyVar6={TYP="R2", TGL="+$80000", VAR="$R[10]", ST="Text offset"}
```

## Extended syntax for the definition of softkeys

SK = {[**ST**=<Label>] [,**AC**=<Access level>] [,**SE**=<Status>]}

**Examples:**

```
HS1={ST="""MySk""", AC=6, SE=1}
HS3={ST="SOFTKEY_CANCEL"}
HS5={ST="[$81251,""\\sk_ok.png""]"}
HS8={ST="[""Test"",""\\sk_ok.png""]"}
```

### Extended syntax for the definition of table columns

{[**TYP**=<Type>] [,**MIN**=<Minimum value>] [,**MAX**=<Maximum value>] [,**LT**=<Long text>] [,**ST**=<Short text>] [,**WR**=<Input mode>] [,**AC**=<Access level>] [,**AL**=<Text alignment>] [,**FS**=<Font size>] [,**LI**=<Limit value handling>] [,**UR**=<Refresh rate>] [,**HLP**=<Help screen>] [,**VAR**=<System or user variable>] >] [,**W**=<Column width>] [,**OF1**=<Offset1>] [,**OF2**=<Offset2>] [,**OF3**=<Offset3>]}

**Example:**

```
DEF MyGridVar={TYP="R", TGL="%MyGrid1", X=10, W=550, H=100}
//G(MyGrid1/0/5)
   {TYP="I", ST="Index", WR=1, VAR="1", W=80, OF1=1}
   {TYP="S", LT="LongText2", ST="Text", WR=1, VAR="$80000", AL=2, W=330, OF1=1}
   {TYP="R3", LT="LongText1", ST="R9,R11,R13,R15", WR=2, VAR="$R[1]", W=110, OF1=2}
//END
```

## 5.8 SmartOperation and MultiTouch operation

You have the following setting options for specific adjustment to SmartOperation and for MultiTouch operation:

- Automatic adjustment of the field height and field width to the so-called minimum usable field size and the line spacing (mask attribute MA).
- Optimized and pixel-precise stretching of the fields for higher resolutions (mask attribute PA).
- Setting of the field height and line spacing proportional to the font (mask attribute FA).
- Allows free scrolling so that the input field is not covered by the virtual keyboard (mask attribute KM).

Further details can be found in Chapter Defining dialog properties (Page 51), Section "Programming".

# Dialogs

# 6

## 6.1 Structure and elements of a dialog

### 6.1.1 Defining a dialog

**Definition**

A dialog is part of a user interface consisting of a display line, dialog elements and/or graphics, an output line for messages and 8 horizontal and 8 vertical softkeys.

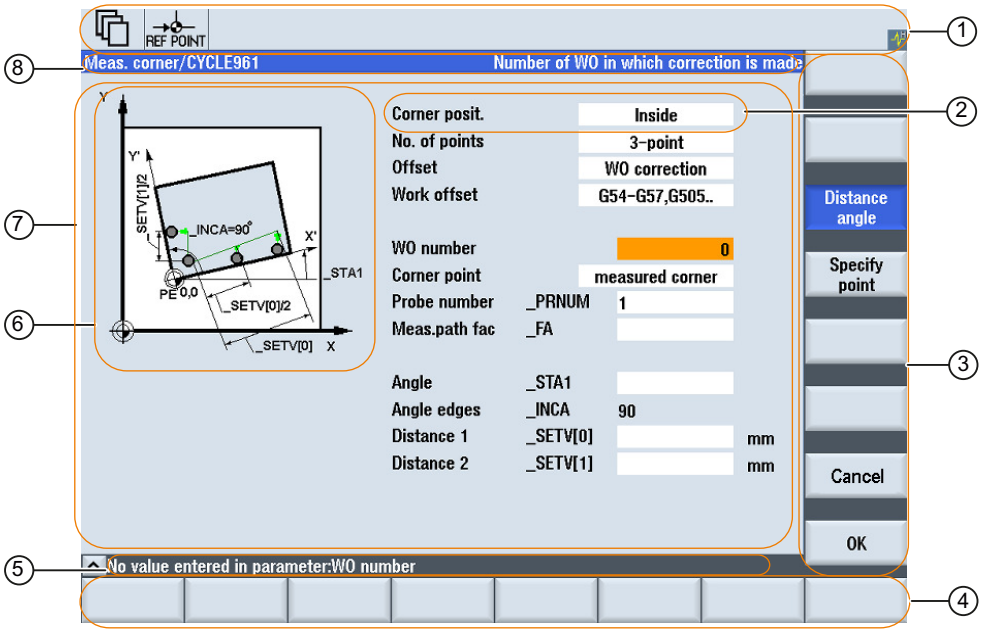
Dialog elements are:

- Variables
  - Limits/toggle field
  - Default setting of variables
- Help display
- Texts
- Attributes
- System or user variable
- Position of short text
- Position of input/output field
- Colors

Dialog properties:

- Header
- Graphic
- Dimension
- System or user variable

- Graphic position
- Attributes




① Machine status display ("header")
② Dialog element
③ 8 vertical softkeys
④ 8 horizontal softkeys
⑤ Output of diagnostic messages
⑥ Graphic
⑦ Dialog
⑧ Header line of the dialog with header and long text

Figure 6-1 Structure of the dialog

## Overview

The definition of a dialog (definition block) is basically structured as follows:

| Definition block | Comment | Chapter reference |
|---|---|---|
| `//M...` | ;Dialog start identifier | |
| `DEF Var1=...`<br>`...` | ;Variables | Variables (Page 83) |
| `HS1=(...)`<br>`...` | ;Softkeys | Defining softkey menus (Page 63) |
| `PRESS(HS1)`<br>`  LM...`<br>`END_PRESS` | ;Method start identifier<br>;Actions<br>;Method end identifier | Methods (Page 119) |
| `//END` | ;Dialog end identifier | |

Within the dialog definition block, various variables that appear as dialog elements in the dialog, as well as horizontal and vertical softkeys, are defined first. Different types of actions are then configured in methods.

**Note**

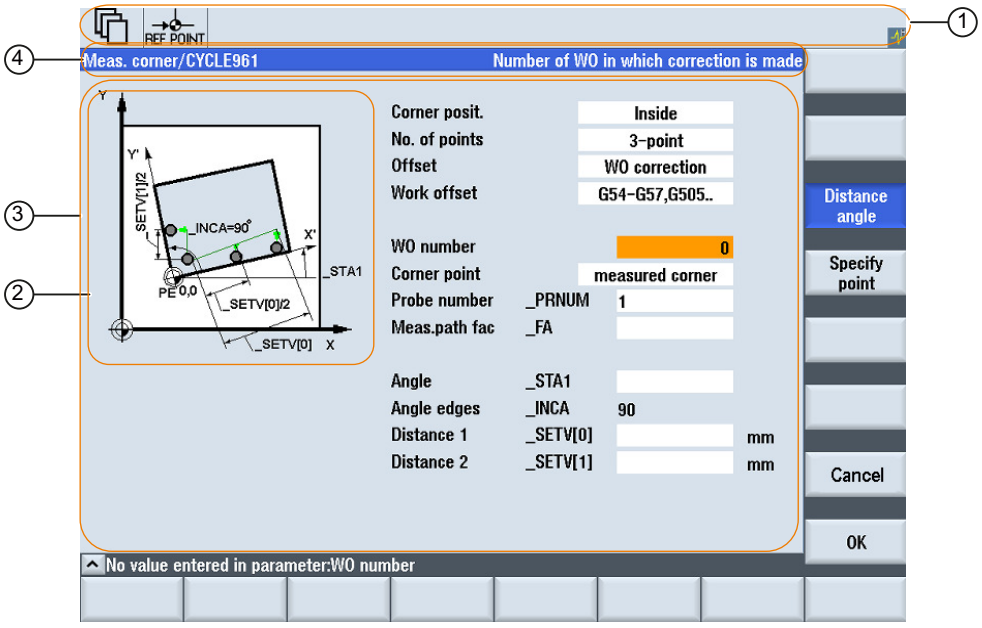
In operating area "Machine", for a dialog change, the following sequence should be observed:

Under the precondition that the dimensions of the following mask are smaller than the previous mask – or the position of the following mask is different than the previous mask, dialog change only functions if the following mask is exited before returning to the first mask, and only then is the first mask reloaded.

## 6.1.2 Defining dialog properties

### Description

The properties of the dialog are defined in the start identifier line of the dialog.




① Machine status display ("header")
② Graphic
③ Dialog
④ Header line of the dialog with header and long text

Figure 6-2 Dialog properties

## Programming

| | |
|---|---|
| Syntax: | **//M**(Identifier/[Title]/[Graphic]/[Dimension]/[System or user variable]/[Position graphic]/[Attribute]/List of the mask-specific language files)<br>See also Chapter Extended configuration syntax (Page 45). |
| Description: | Defines a dialog |

| | | | | |
|---|---|---|---|---|
| Parameter: | Identifier | | | Name of the dialog |
| | Header | | | Dialog header as text or call for text (e.g. $85011) from a language-specific text file. |
| | Graphic | | | Graphics file with path in double quotation marks. |
| | Dimension | | | Position and size of the dialog in pixels (distance from left-hand side, distance from top, width, height), in relation to the upper left-hand corner of the screen. The entries are separated by a comma. |
| | System or user variable | | | System or user variable to which the current cursor position is assigned. The NC or PLC can be provided with the cursor position via the system or user variable. The first variable has index 1. The order corresponds to the configuration order of the variables. |
| | Graphic position | | | Position of the graphic in pixels (distance from left-hand side, distance from top), in relation to the upper left-hand corner of the dialog. The entries are separated by a comma. |
| | Attributes | | | The specifications of the attributes are separated by a comma.<br>Possible attributes are: |
| | | CM | | Column mode: Column alignment |
| | | | CM0 | Default setting: The column distribution is carried out separately for each line. |
| | | | CM1 | The column distribution of the line with the most columns applies to all lines. |
| | | CB | | CHANGE block: Response when dialog is opened: cb attributes specified for a variable in a variables definition take priority over the default setting in the dialog definition.<br>See also AUTOHOTSPOT. |
| | | | CB0 | Default setting: All CHANGE blocks associated with the dialog are processed when it is opened. |
| | | | CB1 | CHANGE blocks are then only processed if the relevant value changes. |
| | | XG | | Integration of an X3D animation as help display (only in the machining step programming).<br>Example:<br>`//M(Meas/ $85605/"myx3dhelpfile.hmi,,Z_Animation,,G17"///30,10/ XG1)` |
| | | | XG0 | Default setting = 0 |
| | | | XG1 | The mask attribute XG must already be set to 1 in the mask definition, a change during runtime is not possible.<br>Note:<br>The specification in the mask property HLP must also be set accordingly. |
| | | AL | | The alignment of the mask header can be influenced with the mask attribute AL.<br>Example:<br>Central setting of the "Set password" window header:<br>`//M(MY_PWD_SET/”Set password”//"EasyPwdModalLayout"// AL2)` |
| | | | AL0 | Left-justified, default |
| | | | AL1 | Right-justified |
| | | | AL2 | Centered |

| | | |
|---|---|---|
| KM | | Freely scrolling in the mask (SlGfwScrollArea), if the virtual keyboard is to be displayed for MultiTouch, can be influenced as follows:<br>• In the mask definition line with mask attribute "KM" (KeyboardMode)<br>Example:<br>`//M(MyMTMask/"MultiTouch Mask"/////KM1)`<br>• As global setting for all Run MyScreens masks in the "easyscreen.ini"<br>Example:<br>`[GENERAL]`<br>`DefaultVirtualKeyboardMode=1`<br><br>Note:<br>If, when configuring the masks, the mask attribute KM is explicitly set, then this has a higher priority than the global setting in "easyscreen.ini".<br><br>(See also Chapter SmartOperation and MultiTouch operation (Page 47)) |
| | KM1 | Free scrolling active (default) |
| | KM0 | Free scrolling inactive |
| PG | | Alignment and appearance of the window title in conjunction with PNG images (only effective in the editor!)<br>Example:<br>`//M(MyPg1SampleMask/"My PG1 Sample Mask"/////PG1)` |
| | PG0 | (Default) |
| | PG1 | Alignment and appearance of the window title in conjunction with PNG images<br>Path specification (on the far left), mask name (on the right)<br>The display of the long text is no longer possible in this mode. Alternatively, you can work with tooltips. |
| MA | | Automatic adjustment of the field height for MultiTouch operation (Mutli-Touch Adjustment)<br>If required, the adjusment for a MultiTouch operator panel includes the necessary enlargement to the so-called minimum usable size (width, height) of the fields (exceptions: progress bar, animated GIF, CustomWidget) and the line spacing.<br>The MultiTouch adjustment can be made<br>• globally for all Run MyScreens mask in "easyscreen.ini", e.g.<br>`[GENERAL]`<br>`DefaultMultiTouchAdjustmentLevel=1`<br>• or mask-specifically in the mask definition line with the mask property "MA" in order to overwrite the global setting in "easyscreen.ini", e.g.<br>`//M(MyMTMask/"MultiTouch Mask"/////MA1)`<br>If the mask attribute MA is set explicitly in the mask configuration, then this has a higher priority than the global setting in "easyscreen.ini".<br>(See also Chapter SmartOperation and MultiTouch operation (Page 47)) |
| | MA0 | There is no adjustment |
| | MA1 | Only those fields are adjusted for which no spacing from the top and/or field height/width has been configured – i.e. the fields that use default positions and therefore for which Run MyScreens calculates the spacing from the top and/or field height/width. (Default) |

| | | |
|---|---|---|
| | MA2 | All fields are adjusted irrespective of the configured field sizes. |
| PA | | Optimized and pixel-precise stretching of the fields for higher resolutions (Pixel Fine Adjustment)<br>Previously all field positions were calculated in relation to 640x480 and then zoomed with the appropriate horizontal and vertical stretch factors. This process has the disadvantage that rounding errors can occur with "unfavorable stretch factors". For example, the field height or line spacing can "jump" by a pixel from line-to-line. The "Pixel Fine Adjustment" mode can be used to prevent this as the field positions are determined directly in the current resolution.<br>`//M(MyMask/"My Mask"////PA1)`<br>This setting can also be made globally for all Run MyScreens masks in "easyscreen.ini", e.g.<br>`[GENERAL]`<br>`DefaultPixelFineAdjustment=1`<br>If the mask attribute PA is set explicitly in the mask configuration, it has a higher priority than the global setting in "easyscreen.ini".<br>If a mask is displayed with Font Adjustment = FA1, then the Pixel Fine Adjustment mode is also automatically active for this mask.<br>(See also Chapter SmartOperation and MultiTouch operation (Page 47)) |
| | PA0 | Stretching as previously, i.e. the positions are calculated in relation to 640x480 and then stretched.<br>(For compatibility reasons: Default) |
| | PA1 | Optimized and pixel-precise stretching of the fields for higher resolutions |
| FA | | Field height and line spacing proportional to the font (Font Adjustment)<br>This setting can also be made globally for all Run MyScreens masks in "easyscreen.ini", e.g.<br>`[GENERAL]`<br>`DefaultFontAdjustment=1`<br>If the mask attribute FA is set explicitly in the mask configuration, then this has a higher priority than the global setting in "easyscreen.ini".<br>The settings for the DefaultLineHeight and DefaultLineSpacing have no significance when Font Adjustment mode is active.<br>(See also Chapter SmartOperation and MultiTouch operation (Page 47)) |
| | FA0 | Field height and line spacing are stretched as previously. (For compatibility reasons: Default) |
| | FA1 | Field height and line spacing are set proportional to the font (automatically sets mask attribute PA=1) |
| | FA2 | Same as FA1, but the X coordinate and the field width are also stretched proportionally to the font<br>(Automatically sets mask attribute PA=1)<br>(Only possible in conjunction with attribute XG=1!) |
| NT | | Navigation Type |
| | NT0 | NT0 = standard mode<br>The navigation is analogous to the masks of the operating area in which Run MyScreens mask is integrated. This means in the cycle masks in the editor, navigation is line-oriented (see NT2), in all other "normal" masks, it is a geometrical (see NT1). |

| | | | | |
|---|---|---|---|---|
| | | | NT1 | Geometrical navigation (top, bottom, left, right), until the borders of the mask are reached (standard in a normal Run MyScreens environment, e.g. area "Custom") |
| | | | NT2 | Line-oriented, i.e. within the line to the right until the end of the line is reached (standard, in the cycle masks of the editor) |
| | | NR | | Navigation direction when pressing the Enter key (Return Navigate Direction) |
| | | | NR0 | NR0 = off (default), i.e. when pressing the Enter key, navigation is realized within the mask in a tabular sequence (standard in the normal Run MyScreens environment) |
| | | | NR1 | When pressing the Enter key, navigation is the same as when pressing the up arrow key |
| | | | NR2 | As for NR1, however, downward |
| | | | NR3 | As for NR1, however, to the left |
| | | | NR4 | As for NR1, however, to the right |
| | | TA | | Tooltip alignment |
| | | | TA0 | Automatic (default) |
| | | | TA1 | Left |
| | | | TA2 | Right |
| | | | TA3 | Top |
| | | | TA4 | Bottom |
| | | | TA5 | Top left |
| | | | TA6 | Bottom left |
| | | | TA7 | Top right |
| | | | TA8 | Bottom right |
| | MC | | | Mask background color (mask color)<br>Example:<br>Set mask background color blue (= 6)<br>`//M(MyMask/"MyMask"//////6)`<br>or<br>`PRESS(HS1)`<br>`  MC=6`<br>`END_PRESS` |
| | List of the mask-specific language files | | | separated by a comma |

## Accessing the dialog properties

Read and write access to the following dialog properties is permitted within methods (e.g. PRESS block).

- HD = Header
- HLP = Help display
- VAR = System or user variable
- MC = Mask background color
- CM = Column alignment (read-only)

- CB = Behavior when opening (read-only)
- XG = Integration of X3d (read-only)
- AL = Mask header alignment (read-only)

## Example

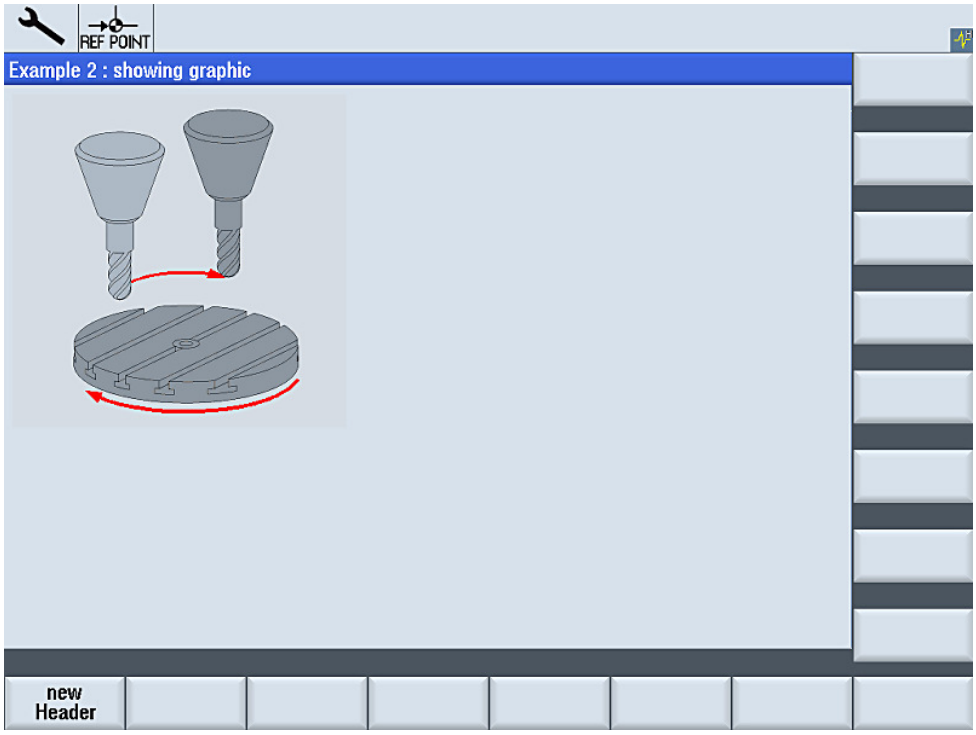



Figure 6-3 "Example 2: showing graphic"

```
//S(Start)
HS7=("Example", se1, ac7)

PRESS(HS7)
  LM("Mask2")
END_PRESS

//END
//M(Mask2/"Example 2 : showing graphic"/"example.png")
HS1=("new%nHeader")
HS2=("")
HS3=("")
HS4=("")
HS5=("")
HS6=("")
HS7=("")
```

```
HS8=("")
VS1=("")
VS2=("")
VS3=("")
VS4=("")
VS5=("")
VS6=("")
VS7=("")
VS8=("")

PRESS(HS1)
  Hd= "new Header"
END_PRESS
...
//END
```

**See also**

Programming example for the "Custom" area (Page 274)

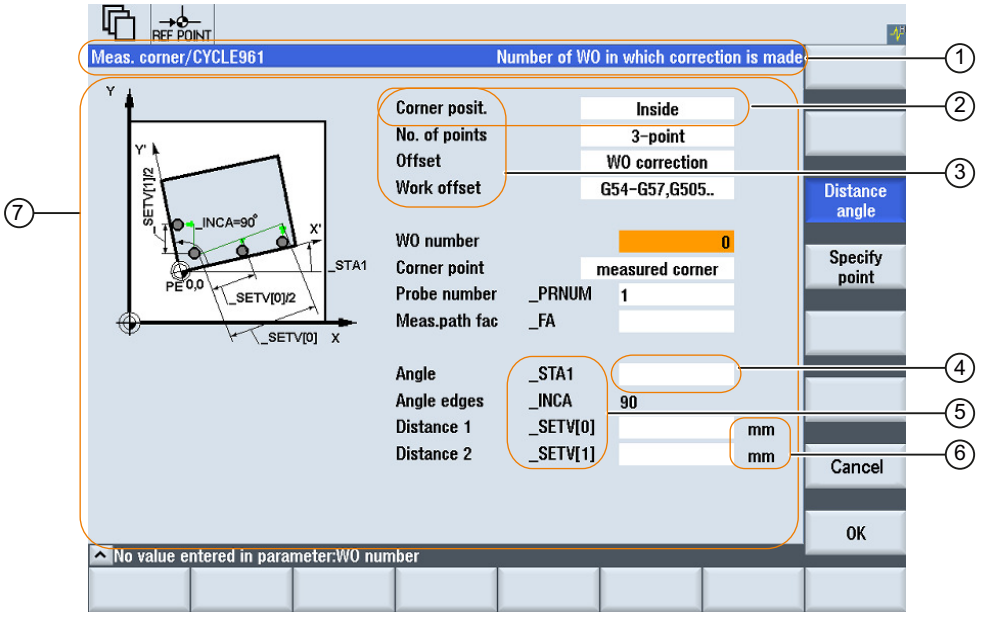
## 6.1.3 Defining dialog elements

### Dialog element

The term "dialog element" refers to the visible part of a variable, i.e. short text, graphics text, input/output field, unit text and tooltip. Dialog elements fill lines in the main body of the dialog. One or more dialog elements can be defined for each line.

## Variable properties

All variables are valid in the active dialog only. Properties are assigned to a variable when it is defined. The values of dialog properties can be accessed within methods (e.g. a PRESS method).




① Header line of the dialog with header and long text
② Dialog element
③ Short text
④ Input/output field
⑤ Graphic text
⑥ Unit text
⑦ Main body of the dialog

Figure 6-4 Elements of a dialog

## Programming - Overview

The single parameters to be separated by commas are enclosed in round brackets:

| DEF [identifier] = | **Identifier = Name of variable** |
|---|---|
| | Variable type |
| | /[Limit values or toggle field] |
| | /[Default] |
| | /[Texts (Long text, Short text\|Image, Graphic text, Unit text)] |
| | /[Attributes] |
| | /[Help display] |
| | /[System or user variable] |
| | /[Position of short text] |
| | /[Position of I/O field(Left, Top, Width, Height)] |

| | |
|---|---|
| | /[Colors] |
| | /[online help] (Page 72) |

### See also

Variable parameters (Page 91)

## 6.1.4 Defining dialogs with multiple columns

### Overview

Multiple variables can also be represented in a dialog on one line. In this case, the variables are all defined in the configuration file on a single definition line.

```
DEF VAR11 = (S///"Var11"), VAR12 = (I///"Var12")
```

To make individual variables in the configuration file more legible, the definition lines can be wrapped after every variables definition and following comma.

The key word "DEF" always indicates the beginning of a new line:

```
DEF Tnr1=(I//1/"","T ",""/wr1///,,10/20,,50),
  TOP1=(I///,"Type="/WR2//"$TC_DP1[1,1]"/80,,30/120,,50),
  TOP2=(R3///,"L1="/WR2//"$TC_DP3[1,1]"/170,,30/210,,70),
   TOP3=(R3///,"L2="/WR2//"$TC_DP4[1,1]"/280,,30/320,,70),
   TOP4=(R3///,"L3="/WR2//"$TC_DP5[1,1]"/390,,30/420,,70)
DEF Tnr2=(I//2/"","T ",""/wr1///,,10/20,,50),
   TOP21=(I///,"Typ="/WR2//"$TC_DP1[2,1]"/80,,30/120,,50),
   TOP22=(R3///,"L1="/WR2//"$TC_DP3[2,1]"/170,,30/210,,70),
   TOP23=(R3///,"L2="/WR2//"$TC_DP4[2,1]"/280,,30/320,,70),
   TOP24=(R3///,"L3="/WR2//"$TC_DP5[2,1]"/390,,30/420,,70)
```

**Note**

When configuring multi-column dialogs, please observe that a large number of columns might slow down the system!

## 6.1.5 Password dialogs

### Using the standard password dialogs

You can integrate the following predefined password dialogs in the screen configuration:

- Set password

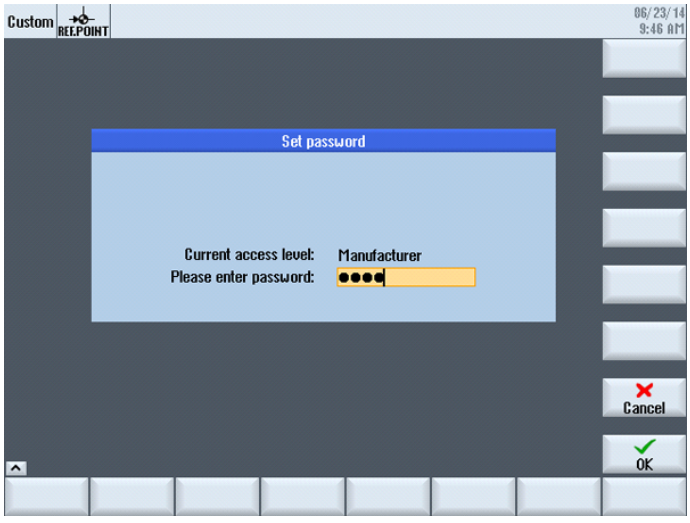



Figure 6-5 Set password dialog

- Change password

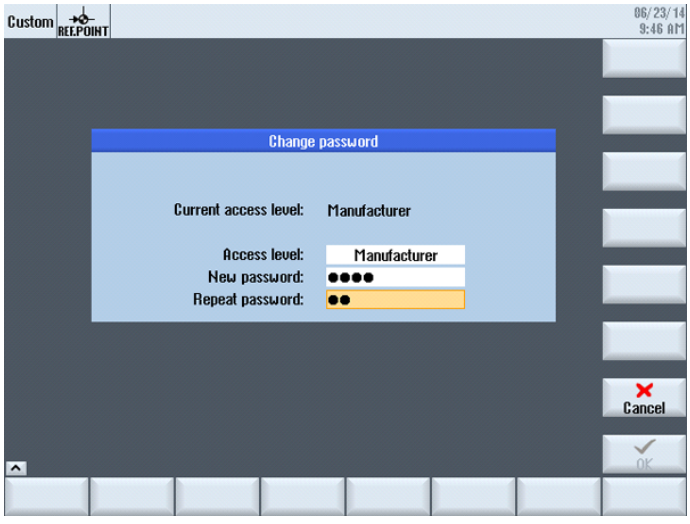



Figure 6-6 Change password dialog

- Delete password

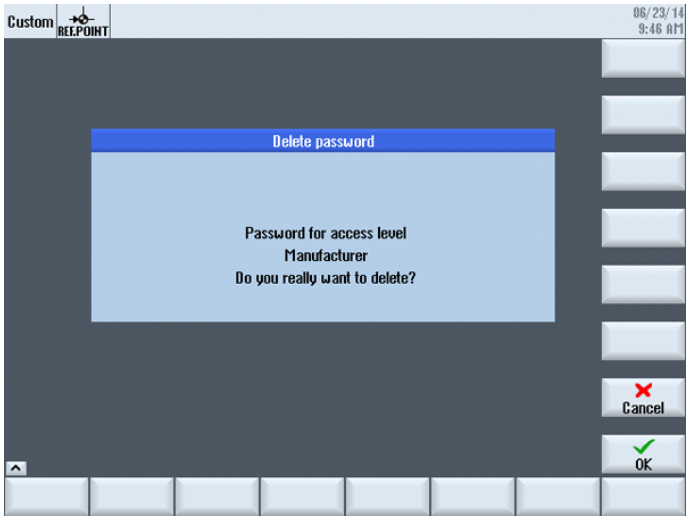



Figure 6-7 Delete password dialog

**Note**

Password functionality is provided with these dialogs. The dialogs do not correspond to the dialogs of SINUMERIK Operate.

The dialogs can be called either via the Load softkey (LS) function or the Load mask (LM) function:

1. Call via the Load softkey (LS) function:
   `LS("Passwd","slespasswd.com",1)`
   Extension of the vertical softkeys:
   - Softkey 1: Set password
   - Softkey 2: Change password
   - Softkey 3: Delete password
2. Alternatively, the three masks can be directly jumped to by calling the (LM) function:
   - Set password: `LM("PWD_SET", "slespasswd.com",1)`
   - Change password: `LM("PWD_CHG", "slespasswd.com",1)`
   - Delete password: `LM("PWD_CLEAR", "slespasswd.com",1)`

### 6.1.6 Using display images/graphics

#### Use of graphics

A distinction is made between:

- Figures/graphics in the graphic area
- Help displays illustrating, for example, individual variables, which are superimposed in the graphic area.
- More help displays can be configured instead of short text or an input/output field, which you position where you like.

#### Storage locations

The image matching the resolution of the connected monitor is searched for in the associated resolution directories in the following sequence:

[System user-directory]`/ico/ico<Resolution>`
[System oem-directory]`/ico/ico<Resolution>`
[System addon-directory]`/ico/ico<Resolution>`

If the image is not displayed or not found, copy it into one of the following directories for a resolution of 640 x 480 pixels:

[System user-directory]`/ico/ico640`
[System oem-directory]`/ico/ico640`
[System addon-directory]`/ico/ico640`

**Note**

The images are positioned proportionally for the different panel resolutions

## 6.2 Defining softkey menus

#### Definition

The term softkey menu is used to refer to all the horizontal and vertical softkeys displayed on a mask. In addition to the existing softkey menus, it is possible to define other menus, which partially or completely overwrite the existing menus.

The names of the softkeys are predefined. Not all softkeys need to be assigned.

HSx x 1 - 8, Horizontal softkeys 1 to 8

VSy y 1 - 8, Vertical softkeys 1 to 8

The definition of a softkey menu (softkey menu definition block) is basically structured as follows:

| Definition block | Comment | Section reference |
|---|---|---|
| `//S...` | ;Start identifier of softkey menu | |
| `HSx=...` | ;Define softkeys | |
| `PRESS(HSx)`<br>`  LM...`<br>`END_PRESS` | ;Method start identifier<br>;Actions<br>;Method end identifier | See Chapter Methods (Page 119) |
| `//END` | ;End identifier of softkey menu | |

## Description

Properties are assigned to softkeys during definition of the softkey menu.

## Programming

| | | |
|---|---|---|
| Syntax: | //S*(Identifier)*<br>...<br>//END | ;Start identifier of softkey menu<br><br>;End identifier of softkey menu |
| | See also Section Extended configuration syntax (Page 45). | |
| Description: | Defines the softkey menu | |
| Parameters: | Identifier | Name of the softkey menu |
| | Text or image file name | |
| Syntax: | **SK** = (Text[, access level][, Status][, Alignment of the softkey image][, text alignment referred to the softkey image]) | |
| Description: | Define softkey | |
| Parameters: | SK | Softkey, e.g. HS1 to HS8, VS1 to VS8 |

| | | | |
|---|---|---|---|
| | Text | Enter text | |
| | Image file name | "\\my_pic.png"<br>or via separate text file $85199, e.g. with the following text in the (language-specific) text file: 85100 0 0 "\ \my_pic.png".<br>The image size which can be displayed on a softkey depends on the OP used: | |
| | | OP 08: | 640 x 480 mm → 25 x 25 pixels |
| | | OP 010: | 640 x 480 mm → 25 x 25 pixels |
| | | OP 012: | 800 x 600 mm → 30 x 30 pixels |
| | | OP 015: | 1024 x 768 mm → 40 x 40 pixels |
| | | OP 019: | 1280 x 1024 mm → 72 x 72 pixels |
| | Access level | ac0 to ac7 (ac7: default) | |
| | Status | se1: visible (default)<br>se2: disabled (gray text)<br>se3: displayed (last softkey used) | |
| | Alignment of the softkey image | PA (PictureAlignment) | |
| | | Valid values:<br>0: Left<br>1: Right<br>2: Centered<br>3: Top (default)<br>4: Bottom | |
| | Text alignment in relation to the softkey image | TP (TextAlignedToPicture) | |
| | | Valid values:<br>0: Text is not aligned to the image<br>1: Text is aligned to the image (default) | |

**Note**

Enter %n in the softkey text to create a line break.

A maximum of 2 lines with 9 characters each are available.

## Assigning access level

Operators can only access information on this access level and lower access levels (see also AUTOHOTSPOT).

## Example

```
//S(Menu1)                                ; Start identifier of softkey menu
HS1=("NEW", ac6, se2)                     ; Define softkey HS1, assign the label "NEW", protec-
                                          tion level 6, and the status "disabled"
```

```
HS2=("\\image1.png")                          ; Assign a softkey to the graphic
HS3=("Exit")
HS4=(["Confirm","\\sk_ok.png"],PA0,TP1)       ; Softkey with text and graphic, Text="Confirm", Im-
                                              age="sk_ok.png", Alignment of the softkey image: Left,
                                              text is aligned to the image


VS1=("sub mask")
VS2=($85011, ac7, se2)                        ; Define softkey VS2, assign the text from the lan-
                                              guage file, protection level 1, and the status "disa-
                                              bled"
VS3=("Cancel", ac1, se3)                      ; Define softkey VS3, assign the label "Cancel", pro-
                                              tection level 1 and the status "highlighted"
VS4=("OK", ac6, se1)                          ; Define softkey VS4, assign the label "OK", protec-
                                              tion level 6 and the status "visible"
VS5=(SOFTKEY_CANCEL,,se1)                     ; Define cancel standard softkey VS5 and assign the
                                              status "visible"
VS6=(SOFTKEY_OK,,se1)                         ; Define OK standard softkey VS6 and assign the status
                                              "visible"
VS7=(["\\image1.png","OEM text"],,se1)        ; Define softkey VS7, assign an image, assign the la-
                                              bel "OEM Text" and the status "visible"
VS8=(["\\image1.png",$83533],,se1)            ; Define softkey VS8, assign an image, assign text
                                              from language file and the status "visible"


PRESS(HS1)                                    ; Method start identifier
  HS1.st=" Calculate"                         ; Assign a label text to the softkey
...
END_PRESS                                     ; Method end identifier

PRESS(RECALL)                                 ; Method start identifier
  LM("Screen21")                              ; Load dialog
END_PRESS                                     ; Method end identifier
//END                                         ; End identifier of softkey menu
```

## 6.2.1 Changing softkey properties during runtime

### Description

The softkey properties Text, Access Level and Status can be changed in the methods during runtime.

## Programming

| | | | |
|---|---|---|---|
| Syntax: | SK.st = "Text"<br>SK.ac = Access level<br>SK.se = Status<br>SK.pa = "Alignment of the softkey image"<br>SK.tp = "Text alignment in relation to the softkey image" | | ;Softkey with label<br>;Softkey with protection level<br>; Softkey with status<br>;Softkey with image<br>;Softkey with image and label |
| Description: | Assign properties | | |
| Parameter: | Text | | Label text in inverted commas |
| | Access level | | Range of values: 0 ... 7 |
| | Status | 1:<br>2:<br>3: | Visible and operator-controllable<br>Disabled (gray text)<br>Highlighted (last softkey used) |
| | Alignment of the softkey image | 0:<br>1:<br>2:<br>3:<br>4: | Left<br>Right<br>Centered<br>Top (default)<br>Bottom |
| | Text alignment in relation to the softkey image | 0:<br>1: | Text is not aligned to the image<br>Text is aligned to the image (default) |

## Example

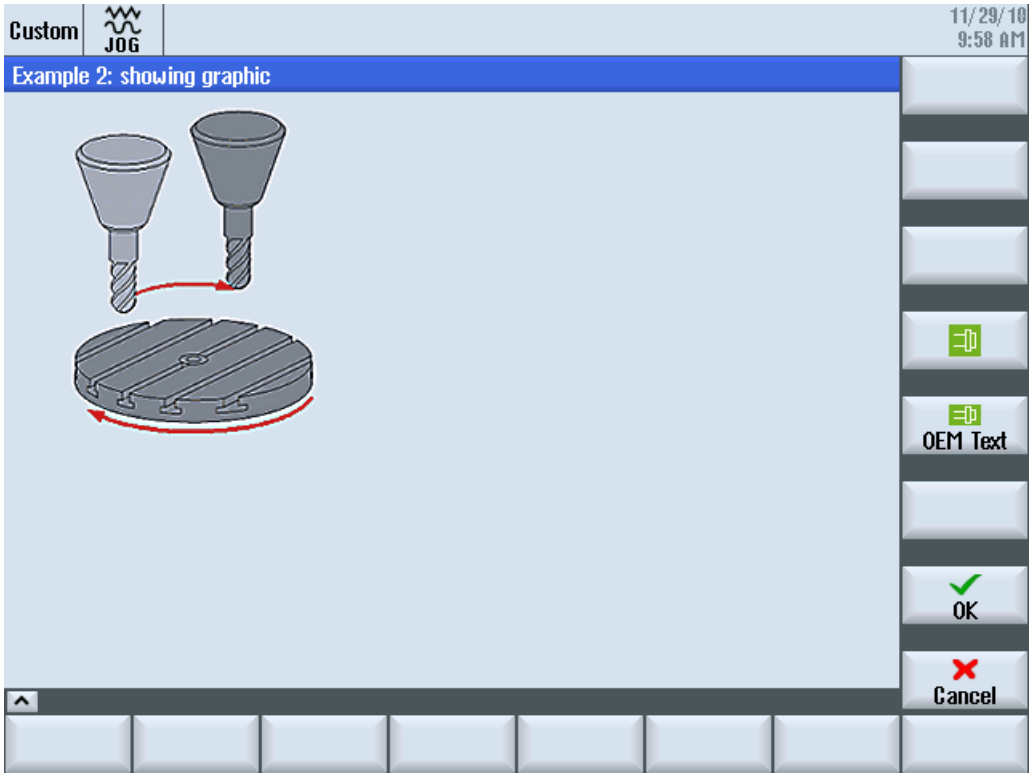



Figure 6-8 Example 3: Graphics and softkeys

```
//S(Start)
HS7=("Example", ac7, se1)

PRESS(HS7)
  LM("Maske3")
END_PRESS

//END

//M(Maske3/"Example 2: showing graphic"/"example.png")
HS1=("")
HS2=("")
HS3=("")
HS4=("")
HS5=("")
HS6=("")
HS7=("")
HS8=("")
VS1=("")
VS2=("")
```

```
VS3=("")
VS4=("\\sp_ok.png",,SE1)
VS5=(["\\sp_ok_small.png","OEM Text"],,SE1)
VS6=("")
VS7=(SOFTKEY_OK,,SE1)
VS8=(SOFTKEY_CANCEL,,SE1)
PRESS(VS4)
  EXIT
END_PRESS
PRESS(VS5)
  EXIT
END_PRESS
PRESS(VS7)
  EXIT
END_PRESS
PRESS(VS8)
  EXIT
END_PRESS
//END
```

## 6.2.2 Language-dependent text

### Overview

Language-dependent texts are used for:

- Softkey labels
- Headings
- Help texts
- Any other texts

The language-dependent texts for dialogs are stored in text files.

The text files are stored in the following directories:

- [System user-directory]/lng
- [System oem-directory]/lng
- [System addon-directory]//lng

**Note**

The text files must be saved in the same way as the project files.

For example:

[System user-directory]/lng/[Text file]

[System user-directory]/proj/[Configuration file]

| | |
|---|---|
| alsc.txt | Language-dependent texts for the Siemens standard cycles |
| almc.txt | Language-dependent texts for the Siemens measuring cycles |

The text files used during program runtime are specified in the "easyscreen.ini" file:

```
[LANGUAGEFILES]
LngFile03 = user.txt      ;->user<_xxx>.txt (e.g.: user_eng.txt)
```

In this instance, the "user.txt" file has been chosen as an example of a text file. The name can always be freely selected. Depending on the language of the texts within the file, the relevant language code must be added according to the following syntax. An underscore followed by the relevant language identifier is added after the name e.g. "user_eng.txt".

**Note**

You must not use LngFile01 and LngFile02 for user texts because they are already assigned in the standard "easyscreen.ini".

## See also

AUTOHOTSPOT

## Mask-specific language files

In order to avoid overlaps for the number ranges used in the language files, only certain language files can be used in a specific mask.

This is the reason that the language files to be used are listed in the mask definition line, separated by a comma. The file listed at the last location has the highest priority (analogous to the logic in "easyscreen.ini").

All of the language-dependent texts used in the mask are predominantly searched for in these language files. If the text is not found in these, then a search is made in text files configured in "easyscreen.ini".

If no language files have been defined in the mask, then a search is only made in the language files specified in "easyscreen.ini".

**Note**

The English language file is used (default) if a language file is not available in the currently selected language.

**Examples:**

```
//M(MyMask/$85597///// //"mytexts.txt, general.txt, mask.txt")
DEF ….
```

You can also use this procedure for the start softkeys:

```
//S(Start, "mytexts.txt, general.txt, mask.txt")
VS1=($85597)
PRESS(VS1)
   LM("MyMask","masks.com")
END_PRESS
```

## Format of text files

The text files must be saved in UTF-8 format.

**Note**

Make sure that the coding is set to UTF-8 in the editor that you are using when saving the configuration and language files.

## Format of a text entry

| | | | |
|---|---|---|---|
| Syntax: | 8xxxx 0 0 "Text" | | |
| Description: | Assignment between text number and text in the file | | |
| Parameters: | xxxx | 85,000 to 89,999<br>900,000 to 999,999 | Text identification number range reserved for users. You must assign unique numbers. |
| | "Text" | | Text that appears in the dialog |
| | %n | | Control characters in the text for creating a line break |

Parameters 2 and 3 are separated by blanks and act as control characters for alarm text output. To ensure that the text format is identical to that of the alarm texts, you must set zero.

**Examples of language-dependent texts:**

```
85000 0 0 "Retraction plane"
85001 0 0 "Drilling depth"
85002 0 0 "Pitch"
85003 0 0 "Pocket radius"
```

# 6.3 Configuring the online help

## 6.3.1 Overview

In addition to the existing extensive online help, you also have the option of generating a manufacturer-specific online help and then linking this into SINUMERIK Operate.

This online help is generated in the HTML format, i.e. it comprises HTML documents that are linked with one another. The subject being searched for is called in a separate window from a contents or index directory. Similar to a document browser (e.g. Windows Explorer), a list of possible selections is displayed in the left-hand half of the window and when you click on the required subject, the explanation is displayed in the right hand half of the window.

Context sensitive selection of online help pages is not possible.

### Procedure

1. Generating HTML files
2. Generating a help book
3. Integrating the online help in SINUMERIK Operate
4. Saving help files

### Other application cases

Online help for the following OEM-specific expansions can be created and used to supplement the SINUMERIK Operate online help system:

- Online help for **cycles and/or M functions** of the machine manufacturer which extend the programming options for SINUMERIK control systems. This online help is called in just the same way as the SINUMERIK Operate online help "Programming".
- Online help for OEM-specific **variables** of the machine manufacturer. This online help is called from the variable view of SINUMERIK Operate.

### Programming online help

You can use the "SINUMERIK HMI programming package sl" for additional options for configuring the online help. Using this programming package, it is possible to develop high-level language applications in the C++ programming language for SINUMERIK Operate on the NCU 7x0.

**Note**

The "SINUMERIK HMI programming package sl" must be ordered separately as a software option. The associated documentation is provided together with the programming package.

## 6.3.2 Generating HTML files

Generating help files in the HTML format. It is possible to save all information in a single HTML file or to distribute the information over several HTML files.

You can assign the file names yourself, however, you must observe the following:

- References within HTML files should always be specified with relative paths. Only then can it be ensured that the references function in precisely the same way on both the development computer as well as on the target system.
- If jumps are to be made to certain points within an HTML file per link, then so-called anchors must be defined for this purpose.
  Example of an HTML anchor:
  `<a name="myAnchor">This is an anchor</a>`
- The contents of HTML documents must be saved with the UTF-8 coding. Only then is it guaranteed that the HTML documents are correctly displayed in all of the country languages supported by SINUMERIK Operate.
- The following sub-sets of the HTML functional scope are supported:

### HTML tags

| Tag | Description | Comment |
|---|---|---|
| a | Anchor or link | Supported attributes: href and name |
| address | Address | |
| big | Larger font | |
| block quote | Indented paragraph | |
| body | Document body | Supported attributes: bgcolor (#RRGGBB) |
| br | Line break | |
| center | Centered paragraph | |
| cite | Inline citation | Same effect as tag i |
| code | Code | Same effect as tag tt |
| dd | Definition data | |
| dfn | Definition | Same effect as tag i |
| div | Document division | The standard block attributes are supported |

| Tag | Description | Comment |
|---|---|---|
| dl | Definition list | The standard block attributes are supported |
| dt | Definition term | The standard block attributes are supported |
| em | Emphasized | Same effect as tag i |
| font | Font size, family, color | Supported attributes: size, face, and color (#RRGGBB) |
| h1 | Level 1 heading | The standard block attributes are supported |
| h2 | Level 2 heading | The standard block attributes are supported |
| h3 | Level 3 heading | The standard block attributes are supported |
| h4 | Level 4 heading | The standard block attributes are supported |
| h5 | Level 5 heading | The standard block attributes are supported |
| h6 | Level 6 heading | The standard block attributes are supported |
| head | Document header | |
| hr | Horizontal line | Supported attributes: width (can be specified as absolute or relative value) |
| html | HTML document | |
| i | Italic | |
| img | Image | Supported attributes: src, width, height |
| kbd | User-entered text | |
| meta | Meta information | |
| li | List item | |
| nobr | Non-breakable text | |
| ol | Ordered list | The standard attributes for lists are supported |
| p | Paragraph | The standard block attributes are supported (default setting: left-aligned) |
| pre | Preformatted text | |
| s | Strikethrough | |
| samp | Sample code | Same effect as tag tt |
| small | Small font | |
| span | Grouped elements | |
| strong | Strong | Continuous text is heavily emphasized, replaces tag b |
| sub | Subscript | |
| sup | Superscript | |
| table | Table | Supported attributes: border, bgcolor (#RRGGBB), cellspacing, cellpadding, width (absolute or relative), height |
| tbody | Table body | No effect |
| td | Table data cell | The standard attributes for table cells are supported |
| tfoot | Table footer | No effect |
| th | Table header cell | The standard attributes for table cells are supported |
| thead | Table header | This is used to print tables that extend over several pages |
| title | Document title | |
| tr | Table row | Supported attributes: bgcolor (#RRGGBB) |
| tt | Typewrite font | |
| u | Underlined | |

| Tag | Description | Comment |
|---|---|---|
| ul | Unordered list | The standard attributes for lists are supported |
| var | Variable | Same effect as tag tt |

### Block attributes

The following attributes are supported by the tags div, dl, dt, h1, h2, h3, h4, h5, h6, p:

- align (left, right, center, justify)
- dir (ltr, rtl)

### Standard attributes for lists

The following attributes are supported by tags ol and ul:

- type (1, a, A, square, disc, circle)

### Standard attributes for tables

The following attributes are supported by tags td and th:

- width (absolute, relative, no-value)
- bgcolor (#RRGGBB)
- colspan
- rowspan
- align (left, right, center, justify)
- valign (top, middle, bottom)

### CSS properties

The following table includes the supported CSS functional scope:

| Property | Values | Description |
|---|---|---|
| background color | <color> | Background color for elements |
| background-image | <uri> | Background image for elements |
| color | <color> | Foreground color for text |
| text-indent | <length>px | Indent the first line of a paragraph in pixels |
| white-space | normal \| pre \| nowrap \| pre-wrap | Defines how whitespace characters are handled in HTML documents |
| margin-top | <length>px | Width of the upper edge of the paragraph in pixels |
| margin-bottom | <length>px | Width of the lower edge of the paragraph in pixels |
| margin-left | <length>px | Length of the left hand edge of the paragraph in pixels |
| margin-right | <length>px | Width of the right-hand edge of the paragraph in pixels |

| Property | Values | Description |
|---|---|---|
| vertical-align | baseline \| sub \| super \| middle \| top \| bottom | Vertical alignment for text (in tables, only the values middle, top and bottom are supported) |
| border-color | <color> | Border color for text tables |
| border-style | none \| dotted \| dashed \| dot-dash \| dot-dot-dash \| solid \| double \| groove \| ridge \| inset \| outset | Border style for text tables |
| background | [ <'background-color'> \|\| <'background-image'> ] | Short notation for background property |
| page-break-before | [ auto \| always ] | Page break before a paragraph/table |
| page-break-after | [ auto \| always ] | Page break after a paragraph/table |
| background-image | <uri> | Background image for elements |

### Supported CSS selectors

All of the CSS 2.1 selector classes are supported, with the exception of so-called pseudo-selector classes, such as :first-child, :visited and :hover.

## 6.3.3 Generating the help book

The help book is an XML file in which the structure of the online help is defined. In this file, you define:

- HTML documents
- Contents and subject index

### Syntax for the help book

| Tag | Number | Meaning | |
|---|---|---|---|
| HMI_SL_HELP | 1 | Root element of the XML document | |
| I-BOOK<br>I<br>I<br>I<br>I<br>I<br>I<br>I<br>I<br>I<br>I | + | Identifies a help book. The name can be freely selected under the constraint that no name predefined by the system is used (such as sinumerik_alarm_plc_pmc).<br>In the example, the name of the help book is "hmi_myhelp"<br>Attributes: | |
| | | ref | Identifies the HTML document that is displayed as the entry page for the help book. |
| | | title | Title of the help book that is displayed in the table of contents. |
| | | helpdir | Directory that contains the online help of the help book. |

| Tag | Number | Meaning | |
|---|---|---|---|
| I-ENTRY<br>II<br>II<br>II<br>II<br>II<br>II<br>II | * | Section of the online help<br>Attributes: | |
| | | ref | Identifies the HTML document that is displayed as entry page for the section. |
| | | title | Title of the section that is displayed in the table of contents. |
| II-INDEX_ENTRY<br>II<br>II<br>II<br>II<br>II<br>II<br>II | * | Subject (keyword) to be displayed<br>Attributes: | |
| | | ref | Identifies the HTML document that is jumped to for this subject index entry. |
| | | title | Title of the subject that is displayed in the subject index. |

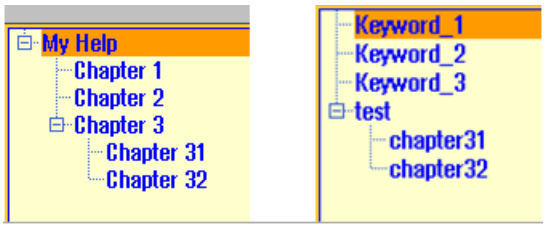
The following applies for the "Number" column:
* means 0 or more
+ means 1 or more

## Example for a help book

In the following example, the structure of a help book with the "My Help" name is described. Further, it forms the basis for the table of contents and subject index.

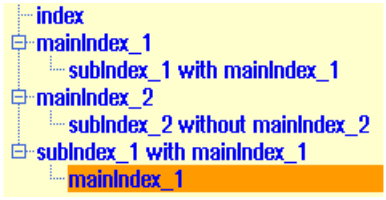
```
<?xml version="1.0" encoding="utf-8"?>
<HMI_SL_HELP language="en-US">
  <BOOK ref="index.html" title="My Help" helpdir="hmi_myhelp">
    <ENTRY ref="section_1.html" title="Section 1">
      <INDEX_ENTRY ref="section_1html#Keyword_1" title="Keyword_1"/>
      <INDEX_ENTRY ref="section_1.html#Keyword_2" title="Keyword_2"/>
    </ENTRY>
    <ENTRY ref="section_2.html" title="Section 2">
      <INDEX_ENTRY ref="section_2.html#Keyword_3" title="Keyword_3"/>
    </ENTRY>
    <ENTRY ref="section_3.html" title="Section 3">
      <ENTRY ref="section_31.html" title="Section 31">
        INDEX_ENTRY ref="section_31.html#test" title="test;section31"/>
      </ENTRY>
      <ENTRY ref="section_32.html" title="Section 32">
        INDEX_ENTRY ref="section_32.html#test" title="test;section32"/>
      </ENTRY>
    </ENTRY>
  </BOOK>
</HMI_SL_HELP>
```

The book comprises three sections, whereby the third section has two subsections. The various subject words (keywords) are defined within the section.




You have the following three options to format the subject index:

1. Single entry:
   `<INDEX_ENTRY ...title="index"/>`
2. Two two-stage entry, whereby each title has a main and a subentry. Separate the entries from one another using a comma.
   `<INDEX_ENTRY ...title="mainIndex_1,subIndex_1 with mainIndex_1"/>`
3. Two-stage entry, whereby the first title is the main entry and the second title is the subentry. Separate the entries from one another using a semicolon.
   `<INDEX_ENTRY ...title="mainIndex_2;subIndex_2 without mainIndex_1"/>`




## 6.3.4 Integrating the online help in SINUMERIK Operate

If you wish to integrate the generated help book into the online help system of SINUMERIK Operate, then you require the "slhlp.xml" file.

### Format description of the "slhlp.xml"

| Tag | Number | Meaning |
|---|---|---|
| CONFIGURATION | 1 | Root element of the XML document. Designates that this involves a configuration file. |
| I-OnlineHelpFiles | 1 | Introduces the section about the online help books. |
| II-<help_book> | * | Introduces the section of a help book. |

| Tag | Number | Meaning | |
|---|---|---|---|
| III-EntriesFile<br>III<br>III<br>III<br>III<br>III | 1 | File name of the help book with the list of contents and subject (keyword) entries.<br>Attributes: | |
| | | value | Name of the XML file |
| | | type | Data type of the value (QString) |
| III-Technology<br>III<br>III<br>III<br>III<br>III<br>III<br>III<br>III<br>III<br>IIII | 0, 1 | Specifies the technology that applies to the help book.<br>"All" applies to all technologies.<br>If the help book applies to several technologies, then the technologies are listed separated by comma.<br>Possible values:<br>All, Universal, Milling, Turning, Grinding, Stroking, Punching<br>Attributes: | |
| | | value | Technology data |
| | | type | Data type of the value (QString) |
| III -DisableSearch<br>III<br>III<br>III<br>III<br>III | 0, 1 | Disable the subject (keyword) search for the help book.<br>Attributes: | |
| | | value | true, false |
| | | type | type, data type of the value (bool) |
| III-DisableFullTextSearch<br>III<br>III<br>III<br>III | 0, 1 | Disable the full text search for the help book.<br>Attributes: | |
| | | value | true, false |
| | | type | type, data type of the value (bool) |
| III-DisableIndex<br>III<br>III<br>III<br>III | 0, 1 | Disable the subject index for the help book.<br>Attributes: | |
| | | value | true, false |
| | | type | type, data type of the value (bool) |
| III-DisableContent<br>III<br>III<br>III<br>III | 0, 1 | Disable the table of contents for the help book.<br>Attributes: | |
| | | value | true, false |
| | | type | type, data type of the value (bool) |
| III-DefaultLanguage<br>III<br>III<br>III<br>III<br>III | 0, 1 | Abbreviation for the language that should be displayed if the actual country language is available for the help book.<br>Attributes: | |
| | | value | chs, deu, eng, esp, fra, ita, ... |
| | | type | Data type of the value (QString) |

The following applies for the "Number" column:
* means 0 or more

### Example of a file "slhlp.xml"

The help book "hmi_myhelp.xml" is made known to SINUMERIK Operate in the following example.

The subject index has not been activated for the help book.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE CONFIGURATION>
<CONFIGURATION>
   <OnlineHelpFiles>
     <hmi_myHelp>
       <EntriesFile value="hmi_myhelp.xml" type="QString"/>
       <DisableIndex value="frue" type="bool"/>
     </hmi_myHelp>
   </OnlineHelpFiles>
</CONFIGURATION>
```

## 6.3.5 Saving help files

### Saving help files in the target system

1. Open the /**oem**/sinumerik/hmi/hlp directory and create a new folder for the required language. For this purpose, use the specified language code.
   It is mandatory that the folder names are written in lower-case letters.
   For instance, if you are integrating a help function for German and English, then create the "deu" and "eng" folders.
2. Place the help book, e.g. "hmi_myhelp.xml" in the "deu" and "eng" folders.
3. Copy the help files into the directories, e.g. /**oem**/sinumerik/hmi/hlp/deu/hmi_myhelp for German and /**oem**/sinumerik/hmi/hlp/eng/hmi_myhelp for English help files.
4. Place the configuration file "slhlp.xml" into the directory /**oem**/sinumerik/hmi/cfg.
5. Restart the HMI.

**Note**

When displaying the list of contents and subject index of a help book, the help files are saved in the binary format (slhlp_<Hilfe-Buch_*.hmi) under the directory /siemens/sinumerik/sys_cache/hmi/hlp for faster use. If you change the help book, you must always delete these files.

## 6.3.6 Help files in PDF format

In addition to help files in HTML format, you can also include PDF-format information in the operating software. Individual PDF helps can be opened with links from the table of contents or index, or directly from HTML files.

### Storing PDF helps

Copy the PDF helps to one of the following directories:

```
/oem/sinumerik/hmi/hlp/<lng>/<hmi_myhelp>
```

```
/user/sinumerik/hmi/hlp/<lng>/<hmi_myhelp>
```

### Including PDF helps

Bind in dialog configurations or configurations of table of contents and indexes with the extension "pdf" in the same manner as for the "html" extension:

```
<ENTRY ref="myFile.pdf" title="Help 1">
```

Create a link from HTML files to the PDF help:

```
<a href="myFile.pdf">My Help File</a>
```

**Note**

It is not possible to select context-sensitive jump labels in the PDF help, select jump labels or make jumps to other HTML or PDF files.

The search function is possible only within a PDF file. A higher-level search over multiple PDF helps is not supported.

# Variables

# 7

## 7.1 Defining variables

### Variable value

The major property of a variable is its value.

The value of variables can be assigned by means of:

- Default settings when defining variables
- Assignment to a system or user variable
- A method

### Programming

| | | |
|---|---|---|
| Syntax: | Identifier.**val** = Variable value<br>Identifier = Variable value | |
| Description: | Variable value val (value) | |
| Parameter: | Identifier: | Name of the variable |
| | Variable value: | Value of the variable |
| Example: | `VAR3 = VAR4 + SIN(VAR5)`<br>`VAR3.VAL = VAR4 + SIN(VAR5)` | |

### Variable status

The "Variable status" property scans a variable for valid content during runtime. This property can be read and written with the value FALSE = 0.

### Programming

| | | |
|---|---|---|
| Syntax: | Identifier.**vld** | |
| Description: | Variable status vld (validation) | |
| Parameter: | Identifier: | Name of the variable |
| | | The result of the scan can be: |
| | FALSE = | invalid value |
| | TRUE = | valid value |
| Example: | `IF VAR1.VLD == FALSE`<br>`  VAR1 = 84`<br>`ENDIF` | |

# 7.2 Application examples

## Help variables

Help variables are internal arithmetic variables. Arithmetic variables are defined like other variables, but have no other properties apart from variable value and status, i.e. help variables are not visible in the dialog. Help variables are of the VARIANT type.

## Programming

| | | |
|---|---|---|
| Syntax: | DEF [identifier] | |
| Description: | Internal arithmetic variables of the VARIANT type | |
| Parameters: | Identifier: | Name of the help variable |
| | | |
| Example: | `DEF OTTO ;Definition of a help variable` | |

| | | |
|---|---|---|
| Syntax: | Identifier.val = Help variable value<br>Identifier = Help variable value | |
| Description: | A value is assigned to a help variable in a method. | |
| Parameters: | Identifier: | Name of the help variable |
| | Help variable value: | Content of the help variables |

**Example:**

```
LOAD
  OTTO = "Test"             ; Assign the value "Test" to the help variable Otto
END_LOAD
LOAD
  OTTO = REG[9].VAL         ; Assign the value of the register to the help variable
END_LOAD                    Otto
```

## Calculation with variables

Variables are calculated every time you exit an I/O field (by pressing the ENTER or toggle key). The calculation is configured in a CHANGE method that is processed every time the value changes.

Whether a variable has a valid value can be queried via the variable status, e.g.:

```
IF VAR1.VLD == FALSE
   VAR1 = 84
ENDIF
```

### Addressing system variables indirectly

A system variable can also be addressed indirectly, i.e. as a function of another variable:

```
PRESS(HS1)
  ACHSE=ACHSE+1
  WEG.VAR="$AA_DTBW["<<ACHSE<<"]"       ;Address axis address via variable
END_PRESS
```

### Changing a softkey label

Example:

```
HS3.st = "New text"                     ;Change softkey label
```

## 7.3 Example 1: Assigning the variable type, texts, help display, colors, tooltips

### Example 1a

The following example defines a variable for which the properties variable type, texts, help display and colors are set.

| **DEF Var1 = (R///,"Actual value",,"mm"//"Var1.png"////8,2)** | | | |
|---|---|---|---|
| | Variable type: | | REAL |
| | Texts: | | |
| | | Short text: | Actual value |
| | | Unit text: | mm |
| | Help screen: | | Var1.png |
| | Colors: | | |
| | | Foreground color: | 8 (brown) |
| | | Background color: | 2 (orange) |

### Example 1b

The following example defines a variable for which the properties variable type, default setting, texts, tooltip, input mode and position of short text are set.

| **DEF Var2 = (I//5/"","Value","","," Tooltip text"/wr2///20,250,50)** | | | |
|---|---|---|---|
| | Variable type: | | INTEGER |
| | Default setting: | | 5 |
| | Texts: | | |
| | | Short text: | Value (possible language text ID) |
| | | Tooltip: | ToolTipText |
| | Attributes: | | |
| | | Input mode: | Reading and writing |
| | Position of short text: | | |
| | | Distance from left: | 20 |
| | | Distance from top: | 250 |
| | | Width: | 50 |

### See also

Variable parameters (Page 91)

## 7.4 Example 2: Assigning the Variable Type, Limits, Attributes, Short Text Position properties

### Example 2

The following example defines a variable for which the properties variable type, limits, input mode, alignment and position are set.

| **DEF Var2 = (I/0,10///wr1,al1///,,300)** | | | |
|---|---|---|---|
| | Variable type: | | INTEGER |
| | Limits or toggle field entries: | | MIN: 0<br>MAX: 10 |
| | Attributes: | | |
| | | Input mode: | Read-only |
| | | Alignment of short text: | Right-justified |
| | Position of short text: | | |
| | | Width: | 300 |

### See also

Variable parameters (Page 91)

## 7.5 Example 3: Assigning the Variable Type, Default, System or User Variable, Input/Output Field Position properties

### Example 3

The following example defines a variable for which the properties variable type, default setting, system or user variable and position are set.

| **DEF Var3 = (R//10////"$R[1]"//300,10,200//)** | | | |
|---|---|---|---|
| | Variable type: | | REAL |
| | Default setting: | | 10 |
| | System or user variable: | | $R[1] (R-Parameter 1) |
| | Position of short text: | | Default position in relation to input/output field |
| | Position of input/output field: | | |
| | | Distance from left: | 300 |
| | | Distance from top: | 10 |
| | | Width: | 200 |

### See also

Variable parameters (Page 91)

## 7.6 Example 4: Toggle field and list field

### Example 4a

Various entries in the toggle field:

```
DEF Var1 = (I/* 0,1,2,3)                ;Simple toggle field to toggle numeric values
DEF Var2 = (S/* "On", "Off")            ;Simple toggle field to toggle strings
DEF Var3 = (B/* 1="On", 0="Off")        ;Extended toggle field to toggle numeric values whereby a
                                         display text is assigned to each numeric value
DEF Var4 = (R/* ARR1)                   ;Toggle field, which obtains its values to be toggled from an
                                         array
```

### Example 4b

The list field corresponds to the configuration of a toggle field, but the display type for the list field (variable attribute DT = 4) must also be set.

| **DEF VAR_LISTBOX_Text = (S/*$80000,$80001,$80002,$80003,$80004/$80001//DT4////200,,340,60)** | | | |
|---|---|---|---|
| | Variable type: | | STRING |
| | Limits / toggle field: | | *$80000,$80001,$80002,$80003,$80004 (list of the language-dependent texts to be displayed) |
| | Default setting: | | $80001 |
| | Attributes: | | |
| | | Display type: | 4 (list field) |
| | Position of input/output field: | | |
| | | Distance from left: | 200 |
| | | Width: | 340 |
| | | Height: | 60 |
| | Colors: | | |
| | | Foreground color: | 6 (blue) |
| | | Background color: | 10 (white) |

## 7.7 Example 5: Image display

### Example 5

Displaying an image instead of a short text: The size and position of the image is defined under "Position of input/output field (left, top, width, height)".

| **DEF VAR6= (V///,"\\image1.png" ////160,40,50,50)** | | | |
|---|---|---|---|
| | Variable type: | | VARIANT |
| | Texts: | | |
| | | Short text: | image1.png |
| | Position of input/output field: | | |
| | | Distance from left: | 160 |
| | | Distance from the top: | 40 |
| | | Width: | 50 |
| | | Height: | 50 |

## 7.8 Example 6: Progress bar

The progress bar is a special display type of the input/output field and is designed for display without input.

Basically there are two progress bar types:

1. Progress bars with up to two color changes, e.g. for temperature or utilization display (see example 6a)
2. Progress bars to display progress (no color change) in the Operate style (see example 6b)

## Example 6a

Progress bar with two color changes:

Figure 7-1 Progress bar with two color changes

| DEF PROGGY0 = (R/0,150,50,100///DT1,DO0//"$R[10]"//,,150/3,4,,,,,,9,7) | | |
|---|---|---|
| Variable type: | | REAL |
| Limits / toggle field: | | |
| | MIN: | 0 |
| | MAX: | 150 |
| | Signal value SVAL1: | 50 |
| | Signal value SVAL2: | 100 |
| Attributes: | | |
| | Display mode DT: | 1 (progress bar) |
| | Display option DO: | 0 (from left to right (default)) |
| System or user variable: | | $R[10] |
| Position of input/output field: | | |
| | Width: | 150 |
| Colors: | | |
| | Foreground color: | 3 (dark green) |
| | Background color: | 4 (light gray) |
| | Signal color SC1: | 9 (yellow) |
| | Signal color SC2: | 7 (red) |

To use a progress bar with color change, the display mode DT (DisplayType) must be set to 1.

The orientation of the progress bar is determined via the attribute display option DO (DisplayOption):

0: From left to right (default)

1: From right to left

2: From bottom to top

3: From top to bottom

A MIN and a MAX value must be specified for the display of the progress bar (in the example MIN: 0, MAX: 150).

Depending on the current value of the variable PROGGY0, this setting already displays a progress bar with the foreground color 3 (= dark green) and the background color 4 (= light gray).

Optionally, one or two signal values SVAL1 and SVAL2 (limit parameters) can be defined (in the example SVAL1: 50 and SVAL2: 100). With these signal values, the foreground color of the progress bar changes. The appropriate signal colors are specified via parameters SC1 and SC2 (in the above example SC1: 9 (= yellow) and SC2: 7 (=red)).

The following rule applies when specifying limit values for the progress display:

MIN < SVAL1 < SVAL2 < MAX.

## Example 6b

Progress bar without color change:

Figure 7-2 Progress bar

| **DEF PROGGY0 = (R/0,150///DT2,DO0//"$R[10]"//,,150/6,10)** | | |
|---|---|---|
| Variable type: | | REAL |
| Limits / toggle field: | | |
| | MIN: | 0 |
| | MAX: | 150 |
| Attributes: | | |
| | Display type DT: | 2 (progress bar) |
| | Display option DO: | 0 (from left to right (default)) |
| System or user variable: | | $R[10] |
| Position of input/output field: | | |
| | Width: | 150 |
| Colors: | | |
| | Foreground color: | 6 (blue) |
| | Background color: | 10 (white) |

To use a progress bar without color change, the display mode DT (DisplayType) must be set to 2.

The orientation of the progress bar is determined via the attribute display option DO (DisplayOption) (see description for example 6a).

A MIN and a MAX value must be specified for the display of the progress bar (in the example MIN: 0, MAX: 150).

Depending on the current value of the variable PROGGY0, this setting displays a progress bar with the foreground color 6 (= blue) and the background color 10 (= white).

## 7.9 Example 7: Password input mode (asterisk)

### Example 7

To implement a field with hidden input, e.g. to enter a password, then display mode DT must be set to 5. Asterisks are then displayed instead of the entered characters.

Figure 7-3 Password input mode (asterisk)

| DEF VAR_SET_PWD=(S//""//DT5) | | | |
|---|---|---|---|
| | Variable type: | | STRING |
| | Default setting: | | Empty string |
| | Attributes: | | |
| | | Display mode DT: | 5 (password input mode) |

## 7.10 Variable parameters

### Parameters - overview

The following overview provides a brief explanation of the variable parameters. Subsequent chapters contain a more detailed description.

| Parameter | Description | |
|---|---|---|
| Variable type (Page 97) | The variable type must be specified. | |
| | R[x]: | REAL (+ digit for the decimal place) |
| | I: | INTEGER |
| | S[x]: | STRING (+ digit for string length) |
| | C: | CHARACTER (individual character) |
| | B: | BOOL |
| | V: | VARIANT |

| Parameter | Description | |
|---|---|---|
| Limits (Page 86) | Limit value MIN, limit value MAX<br>Default setting: Empty<br>The limit values are separated by a comma. Limits can be specified for types I, C and R in decimal formats or as characters in the form "A", "F". | |
| | Optionally, two signal colors SC1 and SC2 can be configured for the display of a progress bar with color change (variable attribute DT = 1) which are displayed as the respective foreground color of the bar when the signal value SVAL1 or SVAL2 is exceeded. The signal values are specified as integer values.<br>See application example for Progress bar (Page 88). | |
| Pre-assignment (Page 101) | If no default setting has been configured and no system or user variable has been assigned to the variable, the first element of the toggle field is assigned. If no toggle field has been defined, there is no default setting, which means the status of the variable is "not calculated".<br>Default setting: No default | |
| Toggle field (Page 100) | List with predetermined entries in the IO field: The list is initiated by a *; the entries are separated by a comma. The entries can be assigned a value.<br>For the toggle field, the entry for the limit is interpreted as a list. If only one * is entered, a variable toggle field is created.<br>Default setting: None | |
| Texts (Page 85) | The sequence is specified. Instead of a short text, an image can also be displayed.<br>Default setting: Empty | |
| | Long text: | Text in the display line |
| | Short text: | Name of the dialog element |
| | Graphic text: | Text refers to the terms in the graphics |
| | Unit text: | Unit of the dialog element |
| | Tooltip | Serves as brief information in a screen configuration for the display and toggle fields. The information is configured via plain text and language text ID. |

| Parameter | Description | |
|---|---|---|
| Attributes (Page 86) | The attributes influence the following properties:<br>• Display mode<br>• Display option<br>• Update rate<br>• Toggle symbol<br>• Tooltip<br>• Input mode<br>• Access level<br>• Alignment of short text<br>• Font size<br>• Limits<br>The attributes are separated by commas and appear in any order. A definition can be made for each component. | |
| | Display mode | dt0: Standard (input/output field or toggle field) (default)<br>dt1: Progress bar with color change<br>dt2: Progress bar without color change in the Operate style<br>dt4: List field<br>dt5: Password input mode (asterisk) |
| | Display option | Particularly, e.g. for the display modes dt1 and dt2 (progress bars), a display option may also have to be configured in combination.<br>do0: From left to right (default)<br>do1: From right to left<br>do2: From bottom to top<br>do3: From top to bottom |
| | Update rate | The attribute UR (update rate) controls the update of the display and therefore the processing of the associated, configured CHANGE block of variables or grid columns. Depending on the configuration, the CPU load can be drastically reduced and therefore shorter response times achieved on the user interface.<br>ur0: SlCap::standardUpdateRate() (currently = 200 ms, default value)<br>ur1: 50 ms<br>ur2: 100 ms<br>ur3: 200 ms<br>ur4: 500 ms<br>ur5: 1000 ms<br>ur6: 2000 ms<br>ur7: 5000 ms<br>ur8: 10000 ms |
| | Toggle symbol | tg0: Toggle symbol off (default)<br>tg1: Toggle symbol on<br>If the attribute TG is set to 1, the toggle symbol also appears in the tooltip of the input field.<br>Example:<br>`DEF OFFS = (R//123.456/,,,,"My ToolTip"/TG1)` |

| Parameter | Description | |
|---|---|---|
| | Tooltip | The tooltip text can also be changed during runtime via the variable property "TT".<br>Example:<br>`PRESS(VS1)`<br>`  MyVar.TT = "My new ToolTip"`<br>`END_PRESS`<br>or<br>`DEF MyVar=(R///,,,,"My ToolTip-text")` |
| | Input mode | wr0: IO field invisible, short text visible<br>wr1: Read (no focus possible for input)<br>wr2: Read and write (line appears in white)<br>wr3: wr1 with focus<br>wr4: All variable elements invisible, no focus possible<br>wr5: The value entered is saved immediately on every keystroke (in contrast to wr2, where it is only saved when the field is exited or RETURN is pressed).<br>Default setting: wr2 |
| | Access level | Empty: Can always be written<br>ac0...ac7: Protection levels<br>If the access level is not adequate, then the first line is displayed in gray, default setting: ac7 |
| | Alignment of short text | al0: Left-justified<br>al1: Right-justified<br>al2: Centered<br>Default setting: al0 |
| | Font size | fs1: Default font size (8 pt.)<br>fs2: Double font size<br>Default setting: fs1<br>The clearances between the lines is defined. With the default font size, 16 lines will fit into the dialog. Graphics and unit text can only be configured in the default font size. |
| | Limits | Consequently, it is possible to check whether the values of the variable are within the MIN and MAX limits specified.<br>Default setting: Determined by specified limits<br>li0: No check<br>li1: Check with respect to min.<br>li2: Check with respect to max.<br>li3: Check with respect to min. and max. |
| | Behavior when opening | cb attributes specified for a variable in a variables definition take priority over the cb default setting in the dialog definition. Multiple attributes are separated by commas (see also AUTOHOTSPOT). |
| | Calling the CHANGE method | CB0: The CHANGE method is triggered when the screen is displayed if the variable has a valid value at this time (e.g. through default setting or NC/PLC variable). |

| Parameter | Description | |
|---|---|---|
| | | CB1: The CHANGE method is not explicitly triggered when the screen is displayed. If the variable has a configured NC/PLC variable, then the CHANGE method is of course still called. |
| | Empty Input (EI) | The variable attribute "EI" (= Empty Input) can be used to specify how the input field is to respond if an empty string is entered.<br>EI0: Standard, i.e. empty inputs into the input field are accepted.<br>EI1: Empty inputs lead to an error status for the input field, and the previous value is set again (Undo). |
| Help display (Page 85) | Help display file: | Name of the png file<br>Default setting: Empty |
| | The name of the Help display file appears in double quotation marks. The display appears automatically (instead of the previous graphic) if the cursor is positioned on this variable. | |
| System or user variable (Page 87) | System or user data from the NC/PLC can be assigned to the variable. The system or user variable appears in double quotation marks.<br>Reference: List Manual System Variables, /PGAsl/ | |
| Position of short text (Page 102) | Position of short text (distance from left, distance from top, width)<br>The positions are entered in pixels and relate to the upper left-hand corner of the main body of the dialog. The entries are separated by commas. | |
| Position of input/output field (Page 102) | Position of input/output field (distance from left, distance from top, width, height)<br>The positions are entered in pixels and relate to the upper left-hand corner of the main body of the dialog. The entries are separated by commas. If this position changes, the positions of the short text, graphic text and unit text also change. | |
| Colors (Page 85) | Foreground and background colors for input/output fields, short texts, graphic texts, unit texts and signal colors for progress bars:<br>The colors are separated by a comma.<br>For specification of the color, see Chapter AUTOHOTSPOT.<br>Default setting for input/output field: Foreground color: Black, background color: White<br>The default colors of the input/output field depend on the "wr" write mode:<br>Default setting for short text, graphic text, unit text: Foreground color: Black, background color: Transparent. | |
| | Optionally, two signal colors SC1 and SC2 can be configured for the display of a progress bar with color change (variable attribute DT = 1) which are displayed as the respective foreground color of the bar when the signal value SVAL1 or SVAL2 is exceeded.<br>See application example for Progress bar (Page 88). | |
| | The colors are expected in the following order in the variables definition:<br>1. Foreground color of the input/output field: FC<br>2. Background color of the input/output field: BC<br>3. Foreground color of the short text: FC_ST<br>4. Background color of the short text: BC_ST<br>5. Foreground color of the graphic text: FC_GT<br>6. Background color of the graphic text: BC_GT<br>7. Foreground color of the unit text: FC_UT<br>8. Background color of the unit text: BC_UT<br>9. Signal color 1<br>10. Signal color 2 | |

| Parameter | Description |
|---|---|
| Online help file | The name of the online help file appears in double quotation marks. |
| I/O field with integrated unit selection | You can use the input/output fields with integrated unit selection to toggle between different units. If the cursor is moved to the input field, the integrated unit selection is highlighted (focus does not have to be explicitly placed on it). A tooltip with a toggle symbol is also displayed with the appropriate reference to this functionality.<br>Examples:<br>`DEF VarEdit=(R/////////200,,100///"VarTgl")`<br>`VarTgl=(S/*0="mm",1="inch"/0//WR2////302,,40)`<br>or<br>`DEF VarEdit_2={TYP="R", VAL=1.234, X=200, W=100, LINK_TGL="vartgl_2"},`<br>`VARTgl_2={TYP="S", TGL="* 0=""mm"", 1=""inch""",WR=2, X=302, W=40}` |

## Variable: Changing properties

A new value is assigned to the variables in the notation Identifier.Property = Value when changing. The expression to the right of the equality sign is evaluated and assigned to the variable or variable property.

| | |
|---|---|
| Identifier.**ac** = Access level | (ac: access level) |
| Identifier.**al** = Text alignment | (al: alignment) |
| Identifier.**bc** = Background color of the input/output field | (bc: back color) |
| Identifier.**bc_gt** = Background color of the graphic text | (bc: back color)<br>(gt: graphic text) |
| Identifier.**bc_st** = Background color of the short text | (bc: back color)<br>(st: short text) |
| Identifier.**bc_ut** = Background color of the unit text | (bc: back color)<br>(ut: unit text) |
| Identifier.**do** = Display option | (do: display option) |
| Identifier.**dt** = Display mode | (dt: display type) |
| Identifier.**fc** = Foreground color of the input/output field | (fc: front color) |
| Identifier.**fc_gt** = Foreground color of the graphic text | (fc: front color)<br>(gt: graphic text) |
| Identifier.**fc_st** = Foreground color of the short text | (fc: front color)<br>(st: short text) |
| Identifier.**fc_ut** = Foreground color of the unit text | (fc: front color)<br>(ut: unit text) |
| Identifier.**fs** = Font size | (fs: font size) |
| Identifier.**gt** = Graphic text | (gt: graphic text) |
| Identifier.**hlp** = Help display | (hlp: help) |
| Identifier.**li** = Limit | (li: limit) |
| Identifier.**lt** = Long text | (lt: long text) |
| Identifier.**max** = MAX limits | (max: maximum) |
| Identifier.**min** = MIN limits | (min: minimum) |

| | |
|---|---|
| Identifier.**sc** = Signal color | (sc: signal color) |
| Identifier.**st** = Short text | (st: short text) |
| Identifier.**tg** = Toggle symbol | (tg: toggle) |
| Identifier.**tt** = Tooltip | (tt: tooltip |
| Identifier.**typ** = Variable type | (typ: type) |
| Identifier.**ur** = Update rate | (ur: update rate) |
| Identifier.**ut** = Unit text | (ut: unit text) |
| Identifier.**val** = Variable value | (val: value) |
| Identifier.**var** = System or user variable | (var: variable) |
| Identifier.**vld** = Variable status | (vld: validation) |
| Identifier.**wr** = Input mode | (wr: write) |

### See also

Extended configuration syntax (Page 45)

## 7.11 Details on the variable type

### Variable type INTEGER

The following extensions for determining the display in the input/output field and the memory utilization are possible for the "INTEGER" type:

**2nd character** in the extension data type

| **Representation format** | |
|---|---|
| B | Binary |
| D | Decimal signed |
| H | Hexadecimal |
| Not specified | Decimal signed |

**3rd and/or 4th character** in the extension data type

| **Memory utilization** | |
|---|---|
| B | Byte |
| W | Word |
| D | Doubleword |
| BU | Byte, unsigned |
| WU | Word, unsigned |
| DU | Double word, unsigned |

### Sequence of characters for the INTEGER data type

1. "I" Basic INTEGER designation
2. Representation format
3. Memory utilization
4. "U" Unsigned

| **Valid INTEGER type specifications:** | |
|---|---|
| IB | Integer variable 32 bits in binary notation |
| IBD | Integer variable 32 bits in binary notation |
| IBW | Integer variable 16 bits in binary notation |
| IBB | Integer variable 8 bits in binary notation |
| I | Integer variable 32 bits in decimal notation signed |
| IDD | Integer variable 32 bits in decimal notation signed |
| IDW | Integer variable 16 bits in decimal notation signed |
| IDB | Integer variable 8 bits in decimal notation signed |
| IDDU | Integer variable 32 bits in decimal notation unsigned |
| IDWU | Integer variable 16 bits in decimal notation unsigned |
| IDBU | Integer variable 8 bits in decimal notation unsigned |
| IH | Integer variable 32 bits in hexadecimal notation |
| IHDU | Integer variable 32 bits in hexadecimal notation |
| IHWU | Integer variable 16 bits in hexadecimal notation |
| IHBU | Integer variable 8 bits in hexadecimal notation |

### Variable type VARIANT

The VARIANT variable type is determined by the data type of the last value assignment. If the assigned or entered value starts with '-', '+', '.' or a number ('0'-'9'), then the value is interpreted as numeric. In all other cases as a string.

It can be scanned using the ISNUM or ISSTR function. The VARIANT type is mainly suited to the purpose of writing either variable names or numerical values to the NC code.

### Programming

The data type of variables can be checked:

| Syntax: | **ISNUM** (VAR) | |
|---|---|---|
| Parameters: | VAR | Name of the variable whose data type is to be checked. |
| | | The result of the scan can be: |
| | FALSE = | not a numerical variable (data type = STRING) |
| | TRUE = | numerical variable (data type = REAL) |

| Syntax: | **ISSTR** (VAR) | |
|---|---|---|
| Parameters: | VAR | Name of the variable whose data type is to be checked. |
| | | The result of the scan can be: |
| | FALSE = | numerical variable (data type = REAL) |
| | TRUE = | not a numerical variable (data type = STRING) |
| Example: | `IF ISNUM(VAR1) == TRUE`<br>`IF ISSTR(REG[4]+2) == TRUE` | |

The display mode of variables can be changed:

- For INTEGER, the display type can be changed.

| | |
|---|---|
| B | Binary |
| D | Decimal signed |
| H | Hexadecimal |
| unsigned | |
| With the addition of U for Unsigned | |

- For REAL data types, only the number of places after the decimal point can be changed. Changing the type is illegal and generates an error message in the "easyscreen_log.txt" file. Example:
  ```
  Var1.typ = "IBW"
  Var2.typ = "R3"
  ```

## Number formats

Numbers can be represented in either binary, decimal, hexadecimal or exponential notation:

| | |
|---|---|
| Binary | B01110110 |
| Decimal | 123.45 |
| Hexadecimal | HF1A9 |
| Exponential | -1.23EX-3 |
| Examples: | |
| | `VAR1 = HF1A9` |
| | `REG[0]= B01110110` |
| | `DEF VAR7 = (R//-1.23EX-3)` |

**Note**

When codes are generated with the "GC" function, only numerical values in decimal or exponential notation are evaluated, but **not** those in binary or hexadecimal notation.

## See also

Variable parameters (Page 91)

# 7.12 Details on the toggle field

## Description

The toggle field extension function displays texts (entries in toggle field) as a function of NC/PLC variables. A variable, which makes use of a toggle field extension, is read-only. The list of the toggle field is opened by pressing the ENTER key.

## Programming

| | | |
|---|---|---|
| Syntax: | **DEF VAR1=(IB/+ $85000/15////"DB90.DBB5")** or<br>**DEF VAR_TGL = (S/* "Hello", "Run", "MyScreens"/"Run")** | |
| Description: | When the dialog is opened, the content of text number $85015 is displayed in the IO field. Default value 15 is entered in system variable DB90.DBB5. If the value saved in system variable DB90.DBB5 changes, the displayed text number $(85000 + <DB90.DBB5>) is recalculated in response to every change. | |
| Parameter: | Variable type | Type of variables specified in the system or user variable. |
| | Text number | Number (basis) of the language-specific text valid as the basis number. |
| | System or user variable | System or user variable (offset) via which the final text number (basis + offset) is displayed. |

## Toggle-field-dependent displays

The toggle field is overlaid with graphics, which change depending on the value of the memory byte. If the value of the memory byte is 1, "image1.png" will appear. If it is 2, "image2.png" will appear.

```
DEF VAR1=(IDB/*1="\\image1.png", 2="\\image2.png"//,$85000/wr1//"MB[130]"//160,40,50,50)
```

The size and position of the image is defined under "Position of IO field (left, top, width, height)".

## Virtual toggle key

There is no list for toggle fields without configured list, e.g. `DEF NoTglList=(R/*)`. The next element is not generated until the toggle key is actuated or the associated CHANGE() method of the corresponding variable run through.

For this case, a small virtual keyboard (comprising only a toggle key) is displayed on the right next to the variable toggle field for operation with a touch panel.

The display of the virtual toggle key can also be forced via the following entry in the configuration file "slguiconfig.ini" irrespective of a touch panel.

```
[VirtualKeyboard]
```

```
; Forces easyscreen virtual toggle keyboard function
ForceEasyscreenVirtualToggleKey = true
```

### See also

Variable parameters (Page 91)

## 7.13 Details on the default setting

### Overview

A variable can assume various states depending on whether a default value, or a system or user variable, or both, is assigned to the variable field (input/output field or toggle field), (not calculated: toggling is only possible if a valid value has been assigned to the variable).

### Scope of the default settings

| Condition | | | Reaction of field type |
|---|---|---|---|
| **Field type** | **Default setting** | **System or user variable** | |
| I/O field | Yes | Yes | Write default value to system or user variable |
| | No | Yes | Use system or user variable as default value |
| | Fault | Yes | Not calculated, system or user variable is not written/ used. |
| | Yes | No | Default setting |
| | No | No | Not calculated |
| | Fault | No | Not calculated |
| | Yes | Fault | Not calculated |
| | No | Fault | Not calculated |
| | Fault | Fault | Not calculated |
| Toggle | Yes | Yes | Write default value to system or user variable |
| | No | Yes | Use system or user variable as default value |
| | Fault | Yes | Not calculated,<br>system or user variable not written/used |
| | Yes | No | Default setting |
| | No | No | Default = first toggle field element |
| | Fault | No | Not calculated |
| | Yes | Fault | Not calculated |
| | No | Fault | Not calculated |
| | Fault | Fault | Not calculated |

**See also**

Variable parameters (Page 91)

## 7.14 Details on the position of the short text, position of the input/ output field

### Overview

The short text and graphic text, as well as the input/output field and unit text, are each treated like a unit, i.e., position settings for short text apply to the graphic text and data for the input/ output field and to unit text.

### Programming

The configured position entry overwrites the default value, i.e., only one value can be changed. If no position settings have been configured for subsequent screen form elements, then the position settings for the preceding screen form element are applied.

The default setting is used if positions are not specified for any of the dialog elements. The column width for the short text and input-output field is calculated as standard for each line from the number of columns and maximum line width, i.e. column width = maximum line width/ number of columns.

The width of the graphics and unit text is predefined and optimized to suit the requirements of programming support. If graphics or unit text has been configured, the width of the short text or I/O field is reduced accordingly.

The order of short text and I/O field can be reversed by position settings.

### Distance between the input/output field and unit field and width of the unit field

You can configure the distance between an input/output field and a unit field as well as the width of the unit field.

In the definition line, in the section for the input/output position, enter the distance from the input/output field to the unit field, separated by a comma (e.g. 7 pixels) and/or the width of the unit field (e.g. 60 pixels):

```
DEF VarDT=(R3//0.000/,"DT",,"s"////0,,24/39,,71,,7,60)
```

Or:

```
DEF VarDT={TYP="R3", VAL="0.000", ST="DT", UT="s", TXT_X=0,
TXT_W=24, X=39, W=71, UT_DX=7, UT_W=60}
```

In this case, the distance between the input/output field/unit field and/or the width of the unit field have/has been configured and the following points should be taken into account:

- The configured width of the input/output field does **not** include the fixed width of the unit field (this is fixed at 50 pixels). This means that you directly configure the width of the input/output field.
- A 50 pixel width applies as default if a width is not configured for the unit field.
- 0 pixels applies as default if a distance is not configured between the input/output field / unit field.

**Special feature regarding the associated toggle field**

Requirements:

- An associated toggle field is configured for a variable,
- For the variable, a distance is configured between the input/output field and unit field and/or a width for the unit field, and
- The variable has no unit text.

In this particular case, the associated toggle field is positioned at the configured position of the unit field of the variable.

A position possibly configured for the input/output component of the associated toggle field is ignored.

### Example

In the following example, the associated toggle field **F_Unit** is automatically set with a distance of **7 pixels** to the input/output field of variable **VarF** , and with a width of **59 pixels** .

```
DEF VarF=(R//0.0/,"F",,,////0,,24/39,,85,,7,59///"F_Unit"), F_Unit =
(I/*3="mm/min", 1="mm/U"/3// ////181,,155)
```

### See also

Variable parameters (Page 91)

## 7.15 Use of strings

### Strings

Strings can be used as part of the configuration. These allow text to be displayed dynamically or different texts to be chained for the purpose of code generation.

### Rules

The following rules must be observed with regard to string variables:

- Logic operations are processed from left to right.
- Nested expressions are solved from the inside outwards.
- No distinction is made between uppercase and lowercase type.
- String variables are generally displayed left justified.

Strings can be deleted simply by assigning a blank string.

Strings can be appended after the equality sign using the operator "<<". Quotation marks (") in the string are represented by two successive quotation mark symbols. Strings can be checked for equality in IF instructions.

### Example

Default settings for the following examples:

```
VAR1.VAL = "This is an"

VAR8.VAL = 4

VAR14.VAL = 15

VAR2.VAL = "Error"

$85001 = "This is an"

$85002 = "Alarm text"
```

Editing strings:

- Chaining of strings:
  ```
  VAR12.VAL = VAR1 << " Error."   ;Result: "This is an error"
  ```
- Deleting a variable:
  ```
  VAR10.VAL = ""   ;Result: Blank string
  ```
- Setting a variable with a text variable:
  ```
  VAR11.VAL = VAR1.VAL   ;Result: "This is an"
  ```
- Data type matching:
  ```
  VAR13.VAL ="This is the " << (VAR14 - VAR8) << ". error"
              ;Result: "This is the 11th error"
  ```
- Treatment of numerical values:
  ```
  VAR13.VAL = "Error" << VAR14.VAL << ": " << $85001 << $85002
           ;Result: "Error 15: "This is an alarm text"
  IF VAR15 == "Error"   ;Strings in IF statement
    VAR16 = 18.1234
           ;Result: VAR16 equals 18.1234,
           ;if VAR15 equals "Error".
  ENDIF
  ```

- Quotation marks within a string:
```
VAR2="Hello, this is a " Test""
   ;Result: Hello, this is a " Test"
```
- System or user-variable strings dependent on variable content:
```
VAR2.Var = "$R[" << VAR8 << "]"   ;Result: $R[4]
```

### See also

STRING functions (Page 177)

## 7.16 CURPOS variable

### Description

The CURPOS variable calls or manipulates the position of the cursor in the active input field of the current dialog. The variable shows how many characters are in front of the cursor. If the cursor is located at the start of the input field, then CURPOS assumes the value of 0. If the value of CURPOS is changed, then the cursor is positioned at the appropriate location in the input field.

In order to be able to respond to changes in the variable value, it is possible to monitor for changes using a CHANGE method. If the value of CURPOS changes, then a jump is made to the CHANGE method and the instructions contained there are executed.

## 7.17 CURVER variable

### Description

The CURVER (CURrent VERsion) property allows the programming to be adapted in order to handle different versions. The CURVER variable is read-only.

**Note**

Even if previously recompiled with an older version, the code is automatically generated with the most recent version. The "GC" command always generates the most recent version. An additional identifier indicating the generated version is inserted in the user comment of the generated code in versions > 0.

### Rules

The most recent dialog with all its variables is always displayed.

- Variables used previously may not be changed.
- New variables are inserted in the existing (cycle) programming in arbitrary order.

- It is not permissible to delete variables from a dialog from one version to the next.
- The dialog must contain all variables of all versions.

### Example

```
(IF CURVER==1 ...)                ; When the code is recompiled, CURVER is automat-
                                    ically assigned the version of the recompiled
                                    code.
```

## 7.18 ENTRY variable

### Description

The ENTRY variable can be used to check by what method a dialog has been called.

### Programming

| Syntax: | **ENTRY** | |
|---|---|---|
| Description: | The ENTRY variable is a read only variable. | |
| Return Value: | | The result of the scan can be: |
| | 0 = | No programming support |
| | 1 = | Start of a screen via softkey; no code has been generated yet (default as configured) |
| | 2 = | Start of a screen via softkey; code has been generated<br>(default from the code generated last by this screen) |
| | 3 = | Recompilation with user comment (# lines) |
| | 4 = | Code_type = 0: NC code with user comment (# lines) |
| | 5 = | Code_type = 1: NC code without user comment (# lines) |

### Example

```
IF ENTRY == 0
  DLGL ("The dialog was not called under programming")
ELSE
  DLGL ("The dialog was called under programming")
ENDIF
```

## 7.19 ERR variable

### Description

Variable ERR checks whether the preceding line has been executed correctly.

### Programming

| Syntax: | **ERR** | |
|---|---|---|
| Description: | The ERR variable is read-only. | |
| Return value: | | The result of the scan can be: |
| | FALSE = | previous line was executed error-free |
| | TRUE = | previous line was not executed error-free |

### Example

```
VAR4 = Thread[VAR1,"CDM",3]            ; Output value from array
IF ERR == TRUE                         ; Query whether the value has been found in
                                       the array
  VAR5 = "Error accessing array"
                                       ; If the value has not been found in the ar-
                                       ray, the value "Error accessing array" is as-
                                       signed to the variables.
ELSE
  VAR5 = "All OK"                      ; If the value has been found in the array,
                                       the value "All OK" is assigned to the varia-
                                       bles.
ENDIF
```

## 7.20 FILE_ERR variable

### Description

Variable FILE_ERR can be used to check whether the preceding GC or CP command has been executed correctly.

### Programming

| Syntax: | **FILE_ERR** |
|---|---|
| Description: | The FILE_ERR variable is read-only. |

| | | |
|---|---|---|
| Return value: | | Possible results are: |
| | 0 = | Operation okay |
| | 1 = | Drive/path not available |
| | 2 = | Path/file access error |
| | 3 = | Drive not ready |
| | 4 = | Incorrect file name |
| | 5 = | File is already open |
| | 6 = | Access denied |
| | 7 = | Target path not available or not permitted |
| | 8 = | Copy source same as target |
| | 10 = | Internal error: FILE_ERR = 10 means that the error cannot be classified in the other categories. |

## Example

```
CP("D:\source.mpf","E:\target.mpf")
                                        ; Copy from source.mpf to E:\tar-
                                        get.mpf
IF FILE_ERR > 0                         ; Query whether error has occurred
   IF FILE_ERR == 1                     ; Query specific error numbers and
                                        output associated error text
      VAR5 = "Drive/path not available"
   ELSE
      IF FILE_ERR == 2
         VAR5 = "Path/file access error"
      ELSE
         IF FILE_ERR == 3
            VAR5 = "incorrect file name"
         ENDIF
      ENDIF
   ENDIF
ELSE
   VAR5 = "All OK"                      ; If no errors have occurred in CP
                                        (or GC), "All OK" is output
ENDIF
```

# 7.21 FOC variable

## Description

With the variable FOC, the input focus (current active input/output field) is controlled in a dialog. The reaction of the cursor left, right, up, down as well as PGUP, PGDN are predefined.

**Note**

The FOC function must not be initiated as a result of a navigation event. The cursor position may only be changed in softkey PRESS methods, CHANGE methods, etc.

The FOC function cannot be applied to variables with input mode wr = 0 and wr = 4 or to Help variables.

## Programming

| Syntax: | **FOC** | |
|---|---|---|
| Description: | The variable can be read and written. | |
| Return value: | Read | The result is the name of the variable to which the FOC function has been applied. |
| | Write | It is possible to assign either a string or a numerical value. A string is interpreted as a variable name and a numerical value as a variable index. |

## Example

```
IF FOC == "Var1"              ; Read focus
  REG[1] = Var1
ELSE
  REG[1] = Var2
ENDIF

FOC = "Var1"                  ; The input focus is assigned to variable 1.
FOC = 3                       ; The input focus is assigned to the 3rd dialog ele-
                              ment with WR ≥ 2.
```

## See also

FOCUS (Page 123)

## 7.22 Variable S_ALEVEL

### Description

The current access level can be queried in the configuration with the screen property S_ALEVEL.

### Programming

| | |
|---|---|
| Syntax: | **S_ALEVEL** |
| Description: | Query of the current access level |
| Return value: | 0: System<br>1: Manufacturer<br>2: Service<br>3: User<br>4: Key switch 3<br>5: Key switch 2<br>6: Key switch 1<br>7: Key switch 0 |

### Example

```
REG[0] = S_ALEVEL
```

### See also

ACCESSLEVEL (Page 120)

## 7.23 S_CHAN variable

### Description

The S_CHAN variable determines the number of the current channel for display or evaluation purposes.

### Programming

| | |
|---|---|
| Syntax: | **S_CHAN** |
| Description: | Query of the current channel number |
| Return value: | Channel number |

### Example

```
REG[0] = S_CHAN
```

### See also

CHANNEL (Page 122)

## 7.24 Variable S_CONTROL

### Description

The current control name can be queried in the configuration with the screen property S_CONTROL.

### Programming

| | |
|---|---|
| Syntax: | **S_CONTROL** |
| Description: | Query of the current control name |
| Return value: | The control name is the section name in the "mmc.ini" |

### Example

```
REG[0] = S_CONTROL
```

### See also

CONTROL (Page 122)

## 7.25 Variable S_LANG

### Description

The current language can be queried in the configuration with the screen property S_LANG.

### Programming

| | |
|---|---|
| Syntax: | **S_LANG** |
| Description: | Query of the current language |
| Return value: | Language code from resources.xml, e.g. "deu", "eng", etc. |

### Example

```
REG[0] = S_LANG
```

### See also

LANGUAGE (Page 123)

## 7.26 Variable S_NCCODEREADONLY

### Description

The screen property S_NCCODEREADONLY is only relevant when a cycle is recompiled in the editor with a "Run MyScreens" dialog. S_NCCODEREADONLY determines whether a recompiled NC code from the editor can be changed or not.

The following applies for the return value:

- TRUE: The recompiled NC code from the editor can be changed
- FALSE: The recompiled NC code from the editor cannot be changed (read-only), because, for example, it is already in the preprocessing (= TRUE).

## 7.27 Variables S_RESX and S_RESY

### Description

The current resolution or its X and Y component can be queried in the configuration with the screen properties S_RESX and S_RESY.

### Example

```
REG[0] = S_RESY
```

**See also**

RESOLUTION (Page 128)

# Programming commands

# 8

## 8.1 Operators

### Overview

The following operators can be used when programming:

- Mathematical operators
- Relational operators
- Logic (Boolean) operators
- Bit operators
- Trigonometric functions

## 8.1.1 Mathematical operators

### Overview

| Mathematical operators | Designation |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| MOD | Modulo operation |
| ( ) | Parentheses |
| AND | AND operator |
| OR | OR operator |
| NOT | NOT operator |
| ROUND | Round off numbers with decimal places |

**Example:** `VAR1.VAL = 45 * (4 + 3)`

### ROUND

The ROUND operator is used to round off numbers with up to 12 decimal places during execution of a dialog configuration. The variable fields cannot accept the decimal places in the display.

### Use

ROUND is controlled by the user with two parameters:

```
VAR1 = 5.2328543
VAR2 = ROUND( VAR1, 4 )
```

**Result:** `VAR2 = 5.2339`

VAR1 contains the number to be rounded. The parameter "4" indicates the number of decimal places in the result, which is placed in VAR2.

### Trigonometric functions

| Trigonometric functions | Designation |
|---|---|
| SIN(x) | Sine of x |
| COS(x) | Cosine of x |
| TAN(x) | Tangent of x |
| ATAN(x, y) | Arc tangent of x/y |
| SQRT(x) | Square root of x |
| ABS(x) | Absolute value of x |
| SDEG(x) | Conversion to degrees |
| SRAD(x) | Conversion to radian |
| CALC_ASIN(x) | Arc sine of x |
| CALC_ACOS(x) | Arc cosine of x |

**Note**

The functions operate with radian measure. The functions SDEG() and SRAD() can be used for conversion.

**Example:** `VAR1.VAL = SQRT(2)`

### Mathematical functions

| Mathematical functions | Designation |
|---|---|
| CALC_CEIL(x) | Determines the next highest integer of x (round up) |
| CALC_FLOOR(x) | Determines the next lowest integer of x (round down) |
| CALC_LOG(x) | Determines the (natural) logarithm for the base e of x |
| CALC_LOG10(x) | Determines the logarithm for the base 10 of x |
| CALC_POW(x, y) | Determines x to the power of y |
| CALC_MIN(x, y) | Determines the lower number of x and y |
| CALC_MAX(x, y) | Determines the higher number of x and y |

## Random function: Random number

| | | |
|---|---|---|
| Syntax: | **RANDOM** (lower limit value, upper limit value) | |
| Description: | The **RANDOM** function returns a pseudo random number in a specified range. | |
| Parameters: | Lower limit value | Lower limit value >= -32767,<br>Lower limit value < upper limit value |
| | Upper limit value | Upper limit value <= 32767 |

## Example

```
REG[0] = RANDOM(-10,10)         ; Possible result = -3
```

## Constants

| **Constants** | |
|---|---|
| PI | 3.14159265358979323846 |
| FALSE | 0 |
| TRUE | 1 |

**Example:** `VAR1.VAL = PI`

## Relational operators

| **Relational operators** | |
|---|---|
| == | Equal to |
| <> | Not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

**Example:**

```
IF VAR1.VAL == 1
  VAR2.VAL = TRUE
ENDIF
```

### Conditions

The nesting depth is unlimited.

Condition with one command:

```
IF
...
ENDIF
```

Condition with two commands:

```
IF
...
ELSE
...
ENDIF
```

## 8.1.2 Bit operators

### Overview

| Bit operators | Designation |
|---|---|
| BOR | Bit-serial OR |
| BXOR | Bit-serial XOR |
| BAND | Bit-serial AND |
| BNOT | Bit-serial NOT |
| SHL | Shift bits to left |
| SHR | Shift bits to right |

### SHL operator

Bits are shifted to the left using the SHL (SHIFT LEFT) operator. You can specify both the value to be shifted and the number of shift increments directly or via a variable. If the limit of the data format is reached, the bits are shifted beyond the limit without displaying an error message.

### Use

| | | |
|---|---|---|
| Syntax: | variable = value **SHL**increment | |
| Description: | Shift Left | |
| Parameters: | value | value to be shifted |
| | increment | number of shift increments |

### Example

```
PRESS(VS1)
  VAR01 = 16 SHL 2           ; Result = 64
```

```
  VAR02 = VAR02 SHL VAR04     ; Convert content of VAR02 to 32-bit unsigned
                              , and shift content to left by number of bits specified
                              in VAR04. Then convert 32-bit value back to format of
                              variable VAR02.
END_PRESS
```

### SHR operator

Bits are shifted to the RIGHT using the SHR (SHIFT RIGHT) function. You can specify both the value to be shifted and the number of shift increments directly or via a variable. If the limit of the data format is reached, the bits are shifted beyond the limit without displaying an error message.

### Use

| | | |
|---|---|---|
| Syntax: | variable = value **SHR**increment | |
| Description: | Shift Right | |
| Parameters: | value | value to be shifted |
| | increment | number of shift increments |

### Example

```
PRESS(VS1)
  VAR01 = 16 SHR 2              ; Result = 4
  VAR02 = VAR02 SHR VAR04       ; Convert content of VAR02 to 32-bit unsigned
                                , and shift content to right by number of bits
                                specified in VAR04. Then convert 32-bit value back
                                to format of variable VAR02.
END_PRESS
```

## 8.2 Methods

### Overview

Various types of event (exit input field, actuate softkey) can initiate specific actions in dialogs and dialog-dependent softkey menus (softkey menus that are called from a newly configured dialog). These actions are configured in methods.

The following table shows the basic principle used to program a method:

| Definition block | Comment | Section reference |
|---|---|---|
| PRESS(HS1) | ;Method start identifier | |
| LM...<br>LS... | ;Functions | See Chapter Functions (Page 131) |

| Definition block | Comment | Section reference |
|---|---|---|
| `Var1.st = ...` | ;Changing properties | see Section Defining softkey menus (Page 63) and Section Defining dialog properties (Page 51). |
| `Var2 = Var3 + Var4`<br>`...`<br>`EXIT` | ;Calculation with variables | See Chapter Defining variables (Page 83) |
| `END_PRESS` | ;Method end identifier | |

## 8.2.1 ACCESSLEVEL

### Description

The ACCESSLEVEL method is run through when the current access level has changed for an opened screen.

### Programming

| | |
|---|---|
| Syntax: | ACCESSLEVEL<br><instructions><br>END_ACCESSLEVEL |
| Description: | Access level |
| Parameters: | - None - |

### See also

Variable S_ALEVEL (Page 110)

## 8.2.2 CHANGE

### Description

CHANGE methods are executed when a variable value changes. This means that variable calculations that are performed as soon as a variable value changes are configured within a CHANGE method.

There are two types of CHANGE method, i.e., element-specific and global:

- The **element-specific CHANGE method** is executed if the value of a specified variable changes. If a system or user variable is assigned to a variable, cyclic updating of the variable value can be configured in a CHANGE method.
- The **global CHANGE method** is executed if the value of any variable changes and no element-specific CHANGE method has been configured.

### "Element-specific" programming

| | | |
|---|---|---|
| Syntax: | CHANGE(identifier)<br>...<br>END_CHANGE | |
| Description: | Changes the value of a specific variable | |
| Parameters: | Identifier | Name of the variable |

### Example

```
DEF VAR1=(I//////"DB20.DBB1")        ; A system variable is assigned to Var1
CHANGE(VAR1)
  IF VAR1.Val <> 1
  VAR1.st="Tool OK!"                 ; If the value of the system variable ≠ 1, the
                                     short text of the variable states: Tool OK!
    otto=1
  ELSE
    VAR1.st="Attention: Error!"      ; If the value of the system variable = 1, the
                                     short text of the variable states: Attention:
                                     Error!
    otto=2
  ENDIF
  VAR2.Var=2
END_CHANGE
```

### "Global" programming

| | |
|---|---|
| Syntax: | CHANGE()<br>...<br>END_CHANGE |
| Description: | Changes any variable value |
| Parameters: | - None - |

### Example

```
CHANGE()
  EXIT                               ; If any of the variable values change, the di-
                                     alog will be terminated.
END_CHANGE
```

## 8.2.3 CHANNEL

### Description

The CHANNEL method is run through when the current channel has changed for an opened screen, i.e. a channel switchover has been performed.

### Programming

| | |
|---|---|
| Syntax: | CHANNEL<br><instructions><br>END_CHANNEL |
| Description: | Channel switchover |
| Parameters: | - None - |

### See also

S_CHAN variable (Page 110)

## 8.2.4 CONTROL

### Description

The CONTROL method is run through when the current control has changed for an opened screen, i.e. typically at a 1:n switchover.

### Programming

| | |
|---|---|
| Syntax: | CONTROL<br><instructions><br>END_CONTROL |
| Description: | Control switchover |
| Parameters: | - None - |

### See also

Variable S_CONTROL (Page 111)

## 8.2.5 FOCUS

### Description

The FOCUS method is executed if the focus (cursor) is positioned on another field in the dialog.

The FOCUS method must not be initiated as a result of a navigation event. The cursor position may only be changed in softkey PRESS methods, CHANGE methods, etc. The response of cursor movements is predefined.

**Note**

Within the FOCUS method, it is not possible to select a different variable, nor can a new dialog be loaded.

### Programming

| | |
|---|---|
| Syntax: | FOCUS<br>...<br>END_FOCUS |
| Description: | Positions the cursor |
| Parameters: | - None - |

### Example

```
FOCUS
  DLGL("The focus has been placed on variable << FOC << ".)
END_FOCUS
```

### See also

FOC variable (Page 109)

## 8.2.6 LANGUAGE

### Description

The LANGUAGE method is run through when the current language has changed for an opened screen.

### Programming

| | |
|---|---|
| Syntax: | LANGUAGE<br><instructions><br>END_LANGUAGE |
| Description: | Language |
| Parameters: | - None - |

### See also

Variable S_LANG (Page 111)

## 8.2.7 LOAD

### Description

The LOAD method is executed after the variable and softkey definitions (DEF Var1= ..., HS1= ...) have been interpreted. At this time, the dialog is not yet displayed.

### Programming

| | |
|---|---|
| Syntax: | LOAD<br>...<br>END_LOAD |
| Description: | Download |
| Parameters: | - None - |

### Example

```
LOAD                          ; Start identifier
  Screen form1.Hd =  $85111   ; Assign text for dialog header from language file
  VAR1.Min = 0                ; Assign MIN variable limit
  VAR1.Max = 1000             ; Assign MAX variable limit
END_LOAD                      ; End code
```

### See also

Defining graphic elements (Page 189)

## 8.2.8 UNLOAD

### Description

The UNLOAD method is executed before a dialog is unloaded.

### Programming

| | |
|---|---|
| Syntax: | UNLOAD<br>...<br>END_UNLOAD |
| Description: | Unload |
| Parameters: | - None - |

### Example

```
UNLOAD
  REG[1] = VAR1            ; Save variable in register
END_UNLOAD
```

## 8.2.9 OUTPUT

### Description

The OUTPUT method is executed if the "GC" function is called. Variables and Help variables are configured as an NC code in an OUTPUT method. The individual elements in a code line are linked by means of blanks.

**Note**

The NC code can be generated in an extra file by means of file functions and transferred to the NC.

### Programming

| | | |
|---|---|---|
| Syntax: | OUTPUT(identifier)<br>...<br>END_OUTPUT | |
| Description: | Outputs variables in the NC program. | |
| Parameters: | Identifier | Name of OUTPUT method |

### Block numbers and skip identifiers

The OUTPUT method must not contain line numbers or skip identifiers if you wish to keep the line numbers and skip identifiers directly set with active program support in the parts program in case of recompilations.

Editor changes in the parts program produce the following response:

| Condition | Response |
|---|---|
| Number of blocks remains unchanged. | Block numbers are retained. |
| Number of blocks is reduced. | The highest block numbers are canceled. |
| Number of blocks is increased. | New blocks are not numbered. |

### Example

```
OUTPUT(CODE1)
  "CYCLE82(" Var1.val "," Var2.val "," Var3.val ","Var4.val "," Var5.val "," Var6.val ")"
END_OUTPUT
```

## 8.2.10 PRESS

### Description

The PRESS method is executed when the corresponding softkey is pressed.

### Programming

| | | | |
|---|---|---|---|
| Syntax: | PRESS(softkey)<br>...<br>END_PRESS | | |
| Designation: | Pressing a softkey | | |
| Parameters: | Softkey | Name of softkey: HS1 - HS8 and VS1 - VS8 | |
| | RECALL | <RECALL> key | |
| | ENTER | For the <ENTER> key, see PRESS(ENTER) (Page 127) | |
| | TOGGLE | For the <TOGGLE> key, see PRESS(TOGGLE) (Page 127) | |
| | PU | Page Up | Screen up |
| | PD | Page Down | Screen down |
| | SL | Scroll left | Cursor left |
| | SR | Scroll right | Cursor right |
| | SU | Scroll up | Cursor up |
| | SD | Scroll down | Cursor down |

### Example

```
HS1 = ("another softkey menu")
HS2 = ("no function")
PRESS (HS1)
  LS("Menu1")                    ; Load another softkey menu
  Var2 = Var3 + Var1
END_PRESS
PRESS (HS2)
END_PRESS
PRESS(PU)
  INDEX = INDEX -7
  CALL("UP1")
END_PRESS
```

## 8.2.11 PRESS(ENTER)

### Description

The PRESS(ENTER) method is always called when the Enter key is pressed for a variable with input/output field with input mode WR3 or WR5:

- WR3: Navigation to field and pressing of the Enter key
- WR5: In the input mode, transfer of the value with the Enter key

### Programming

| | |
|---|---|
| Syntax: | PRESS(ENTER)<br><instructions><br>END_PRESS |
| Description: | Enter key pressed |
| Parameters: | - None - |

## 8.2.12 PRESS(TOGGLE)

### Description

The PRESS(TOGGLE) method is always called when the toggle key is pressed irrespective of the currently focused variable.

When required, the FOC screen property can be used to determine which variable is currently in focus.

### Programming

| | |
|---|---|
| Syntax: | PRESS(TOGGLE)<br><instructions><br>END_PRESS |
| Description: | Toggle key pressed |
| Parameter: | - None - |

### Example

```
PRESS(TOGGLE)
   DLGL("Toggle key pressed at variable " << FOC)        ; The FOC screen property determines which
                                                         variable is currently in focus
END_PRESS
```

## 8.2.13 RESOLUTION

### Description

The RESOLUTION method is run through when the current resolution has changed for an opened screen, i.e. typically at a TCU switchover.

### Programming

| | |
|---|---|
| Syntax: | RESOLUTION<br><instructions><br>END_RESOLUTION |
| Description: | Resolution |
| Parameters: | - None - |

### See also

Variables S_RESX and S_RESY (Page 112)

## 8.2.14 RESUME

### Description

The RESUME method is called when the screen was interrupted, e.g. for an area change, and is now shown again. The value changes are processed again, if required, the timers started and the RESUME block executed.

**Note**

The functions LS, LM, DLGL, EXIT and EXITLS must not be configured in the methods SUSPEND or RESUME. When using the functions in these methods, the execution of the functions is prevented.

### Programming

| | |
|---|---|
| Syntax: | RESUME<br><instruction><br>END_RESUME |
| Description: | Screen active again |
| Parameters: | - None - |

## 8.2.15 SUSPEND

### Description

The SUSPEND method is called when the screen is interrupted and not unloaded. This is the case, for example, when the screen is exited by a simple area change, but not explicitly unloaded. The screen is retained in the background, but no value change or timers are processed. The screen is paused.

**Note**

The functions LS, LM, DLGL, EXIT and EXITLS must not be configured in the methods SUSPEND or RESUME. When using the functions in these methods, the execution of the functions is prevented.

### Programming

| | |
|---|---|
| Syntax: | SUSPEND<br><instruction><br>END_SUSPEND |
| Description: | Interruption of the screen |
| Parameters: | - None - |

### Example

```
SUSPEND
   MyVar1 = MyVar1 + 1
END_SUSPEND
```

## 8.2.16 Example: Version management with OUTPUT methods

### Overview

Additional variables can be added to existing dialogs when expanding the user interface. A version identifier in parentheses is appended to the additional variables in the definition following the variable name: (0 = Original, is not written), 1 = Version 1, 2 = Version 2, etc.

**Example:**

```
DEF var100=(R//1)              ; Original, corresponds to Version 0
DEF var101(1)=(S//"Hello")     ; Extension as of Version 1
```

When writing the OUTPUT method, you can specify which variables are written, with reference to a particular version identifier.

**Example:**

```
OUTPUT(NC1)                    ; Only the variables of the original are offered in
                               the OUTPUT method.
OUTPUT(NC1,1)                  ; The variables of the original and the extensions
                               with version identifier 1 are offered in the OUTPUT
                               method.
```

The OUTPUT method for the original does not need a version identifier, however you can specify it with 0. OUTPUT(NC1) is equivalent to OUTPUT(NC1,0). Version identifier n in the OUTPUT method includes all variables of the originals 0, 1, 2, ... up to and including n.

### Programming with version identifier

```
//M(XXX)                          Version 0 (default)
DEF var100=(R//1)
DEF var101=(S//"Hello")
DEF TMP
VS8=("GC")
PRESS(VS8)
   GC("NC1")
END_PRESS

OUTPUT(NC1)
var100",,"var101
END_OUTPUT

; ************ Version 1, extended definition ***************
//M(XXX)
DEF var100=(R//1)
DEF var101=(S//"Hello")
DEF var102(1)=(V//"HUGO")
DEF TMP
VS8=("GC")
PRESS(VS8)
   GC("NC1")
END_PRESS
...

OUTPUT(NC1)                        Original and the new version in addition
var100","var101
END_OUTPUT
...

OUTPUT(NC1,1)                      Version 1
var100","var101"," var102
END_OUTPUT
```

## 8.3 Functions

### Overview

A variety of functions are available in dialogs and dialog-dependent softkey menus. These can be activated by specific events (exit input field, actuate softkey) and configured in methods.

### Subprograms

Repeatedly used configuring instructions or others, which define the process for a particular operation can be configured in subprograms. Subprograms can be loaded into the main program or other subprograms at any time and executed as often as necessary, i.e. the instructions they contain do not need to be configured repeatedly. The definition blocks of the dialogs/softkey menu constitute a main program.

### External functions

Additional, user-specific functions can be integrated by means of external functions. The external functions are stored in a DLL file and identified by an entry in the definition lines of the configuration file.

### PI services

The PI_START function starts PI Services (Program Invocation Services) from the PLC in the NC area.

### See also

Function (FCT) (Page 151)

PI services (Page 166)

## 8.3.1 Reading and writing drive parameters: RDOP, WDOP, MRDOP

### Description

You can read and write drive parameters using RDOP, WDOP and MRDOP functions.

**Note**

Do not read drive parameters faster than in a 1 second cycle, slower is better.

Reason: The communication with the drives can otherwise be extremely disturbed or even cause failures.

**Note**

If errors occur when reading and writing drive parameters, the screen property ERR is set appropriately.

### Programming

| | |
|---|---|
| Syntax: | **RDOP**("Identifier of the drive object", "Parameter number") |
| Description: | Reading a drive parameter (Drive Object Parameter) |

| | | |
|---|---|---|
| Parameters: | Identifier of the drive object | The identifier of the drive object can be found in the "DO name" column in "Drive system diagnostics" in the diagnostics operating area ( > ETC > HSK 8: drive system) (see diagram below). |
| | Parameter number | |

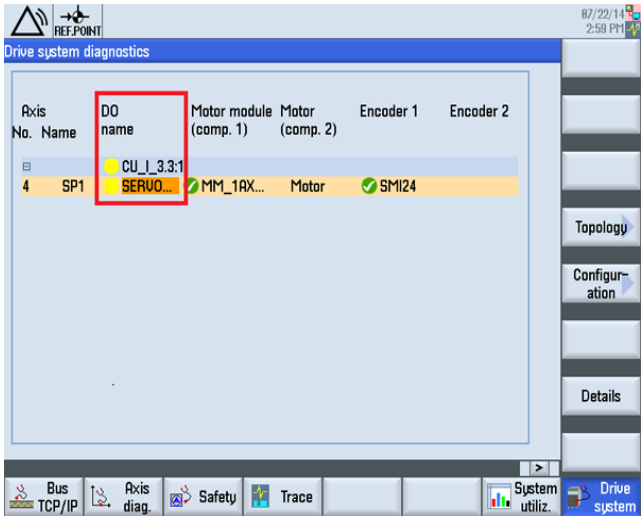



Figure 8-1 Identifier of the drive object

| | | |
|---|---|---|
| Syntax: | **MRDOP**("Identifier of the drive object", "Parameter number1"*"Parameter number"[*...], Register index) | |
| Description: | Multi-read of drive parameters.<br>The MRDOP command transfers several drive parameters or drive objects in a single register access. This access method is significantly faster than reading via individual access attempts. | |
| Parameters: | Identifier of the drive object | The identifier of the drive object can be taken, for example, from the root screen of the "Setup" operating area. |
| | Parameter number1 ... n | In the variable names, "*" is the separator. The values are transferred to register REG[Register index] in the order that the variable names appear<br>in the command. |
| | Register index | The value of the first variable is located in REG[Register index].<br>The value of the second variable is located in REG[Register index + 1] |

| Syntax: | **WDOP**("Identifier of the drive object", "Parameter number", Value) | |
|---|---|---|
| Description: | Writing a drive parameter (Drive Object Parameter) | |
| Parameters: | Identifier of the drive object | The identifier of the drive object can be taken, for example, from the root screen of the "Setup" operating area. |
| | Parameter number | Parameter number |
| | Value | Value to be written |

### Examples

Read motor temperature r0035 of drive object "SERVO_3.3:2":

```
MyVar=RDOP("SERVO_3.3:2","35") ;
```

Read motor temperature r0035 and actual torque value r0080 of drive object "SERVO_3.3:2" and store the respective results as of register index 10:

```
MRDOP("SERVO_3.3:2","35*80",10)
```

## 8.3.2 Subprogram call (CALL)

### Description

The CALL function calls a loaded subprogram from any point in a method. Subprogram nesting is supported, i.e. you can call a subprogram from another subprogram.

### Programming

| Syntax: | **CALL**("Identifier") | |
|---|---|---|
| Description: | Subprogram call | |
| Parameters: | Identifier | Name of subprogram |

### Example

```
//M(SCREEN1)
DEF VAR1 = ...
DEF VAR2 = ...
CHANGE(VAR1)
  ...
  CALL("MY_UP1")              ; Call subprogram and execute
  ...
END_CHANGE
```

```
//M(SCREEN1)
CHANGE(VAR2)
  ...
  CALL("MY_UP1")              ; Call subprogram and execute
  ...
END_CHANGE

SUB(MY_UP1)
                              ;Do something
END_SUB
//END
```

## 8.3.3 Define block (//B)

### Description

In the program file, subprograms are identified by the block identifier //B and terminated with //END. Several subprograms can be defined under each block identifier.

**Note**

The variables used in the subprogram must be defined in the dialog in which the subprogram is called.

### Programming

A block is structured in the following way:

| | | |
|---|---|---|
| Syntax: | **//B**(Block name)<br>**SUB**(Identifier)<br>END_SUB<br>[SUB(Identifier)]<br>...<br>END_SUB]<br>...<br>//END | |
| Description: | Defines a subprogram | |
| Parameters: | Block name | Name of block identifier |
| | Identifier | Name of subprogram |

### Example

```
//B(PROG1)                          ; Block start
SUB(UP1)                            ; Start of subprogram
  ...
  REG[0] = 5                        ; Assign value 5 to register 0
  ...
END_SUB                             ; End of subprogram
SUB(UP2)                            ; Start of subprogram
  IF VAR1.val=="Otto"
    VAR1.val="Hans"
    RETURN
  ENDIF
  VAR1.val="Otto"
END_SUB                             ; End of subprogram
//END                               ; Block end
```

## 8.3.4 Check Variable (CVAR)

### Description

You can use the CVAR (Check Variable) function to run a scan to ascertain whether all or only certain variables or Help variables in a screen are error-free.

It may be useful to check if variables contain a valid value before an NC code with the GC function.

A variable is error-free if the state of the variable Identifier.vld = 1.

### Programming

| Syntax: | **CVAR**(VarN) | |
|---|---|---|
| Description: | Checks variables for valid content | |
| Parameters: | VarN | List of variables to be checked.<br>Up to 29 variables, each separated by a comma, can be checked. A character length of 500 must not be exceeded.<br>The result of the scan can be: |
| | 1 = | TRUE (all variables have valid content) |
| | 0 = | FALSE (at least one variable has invalid content) |

### Example

```
IF CVAR == TRUE                          ; Check all variables
```

```
  VS8.SE = 1                                ; If all variables are error-free, soft-
                                            key VS8 is visible
ELSE
  VS8.SE = 2                                ; If a variable has an invalid value,
                                            softkey VS8 is disabled
ENDIF

IF CVAR("VAR1", "VAR2") == TRUE
                                            ; Check variables VAR1 and VAR2
  DLGL ("VAR1 and VAR2 are OK")
                                            ; If the values of VAR1 and VAR2 are er-
                                            ror-free, "VAR1 and VAR2 are OK" appears
                                            in the dialog line
ELSE
  DLGL ("VAR1 and VAR2 are not OK")
                                            ; If the values of VAR1 and VAR2 are in-
                                            valid, "VAR1 and VAR2 are not OK" appears
                                            in the dialog line
ENDIF
```

## 8.3.5 Copy Program file function (CP)

### Description

The CP (Copy Program) function copies files within the HMI file system or within the NC file system.

### Programming

| Syntax: | **CP**("Source file", "Target file") | |
|---|---|---|
| Description: | Copies a file | |
| Parameters: | Source file | Complete path to the source file |
| | Target file | Complete path data of the target file |

The return value (VAR1 defined as help variable) queries whether the function was successful:

```
CP("//NC/MPF.DIR/HOHO.MPF","//NC/MPF.DIR/ASLAN.MPF",VAR1)
```

### Example

Application with return value:

```
CP("//NC/MPF.DIR/HOHO.MPF","//NC/MPF.DIR/ASLAN.MPF",VAR1)
CP("CF_CARD:/wks.dir/MESS_BILD.WPD/MESS_BILD.MPF","//NC/WKS.DIR/AAA.WPD/HOHO2.MPF",VAR1)
```

```
CP("//NC/MPF.DIR/HOHO.MPF","CF_CARD:/wks.dir/WST1.WPD/MESS.MPF",VAR1)        ; WPD must exist
```

Application without return value:

```
CP("//NC/MPF.DIR/HOHO.MPF","//NC/MPF.DIR/ASLAN.MPF")
CP("CF_CARD:/mpf.dir/MYPROG.MPF","//NC/MPF.DIR/HOHO.MPF")
CP("//NC/MPF.DIR/HOHO.MPF","CF_CARD:/XYZ/MYPROG.MPF")                        ; XYZ must exist
```

### See also

Support of FILE_ERR: FILE_ERR variable (Page 107)

## 8.3.6 Delete Program file function (DP)

### Description

The DP (Delete Program) function deletes a file from the passive HMI or active NC file system.

### Programming

| Syntax: | **DP**("File") | |
|---|---|---|
| Description: | Delete file | |
| Parameters: | File | Complete path name of file to be deleted |

### Example

The following data management syntax is used for this function:

- With return value

  ```
  DP("//NC/MPF.DIR/MYPROG.MPF",VAR1)
  DP("//NC/WKS.DIR/TEST.WPD/MYPROG.MPF",VAR1)
  DP("//NC/CMA.DIR/MYPROG.SPF",VAR1)
  ```

  VAR1 = 0 File was deleted.
  VAR1 = 1 File was not deleted.
- Without return value:

  ```
  DP("//NC/MPF.DIR/MYPROG.MPF")
  DP("//NC/WKS.DIR/TEST.WPD/MYPROG.MPF")
  DP("//NC/CMA.DIR/MYPROG.SPF")
  ```

## 8.3.7 Exist Program file function (EP)

### Description

The EP (Exist Program) function checks whether a particular NC program is stored on the specified path in the NC or HMI file system.

### Programming

| Syntax: | **EP**("File") | |
|---|---|---|
| Description: | Checks the existence of the NC program | |
| Parameters: | File | Complete path to the file in the NC or HMI file system |
| Return value: | Name of a variable to which the result of the scan should be assigned | |

The EP function can handle the new syntax and the old logic (with adapted syntax).

The file is directly addressed using a qualifying name:

`//NC/MPF.DIR/XYZ.MPF`

or
`CF_CARD: /MPF.DIR/XYZ.MPF` (points to /user/sinumerik/data/prog)
or
`LOC:` (corresponds to CF_CARD)

**Examples of new syntax:**

```
EP("//NC/WKS.DIR/TEST.WPD/XYZ.MPF",VAR1)
EP("CF_CARD:/mpf.dir/XYZ.MPF",VAR1)
EP("LOC:/mpf.dir/XYZ.MPF",VAR1)
; With return value:
; VAR1 = 0 File exists.
; VAR1 = 1 File does not exist.
```

**Example of old syntax:**

```
EP("/MPF.DIR/CFI.MPF", VAR1)
; With return value:
; VAR1 = M File is located in the HMI file system.
; VAR1 = N File is located in the NC file system.
; VAR1 = B File is located in the HMI and the NC file system.
```

### Example

```
EP("/MPF.DIR/GROB.MPF",VAR1)                                    ; Old - path now with / instead of \

IF VAR1 == "M"
   DLGL("File is located in the HMI file system")
ELSE
   IF VAR1 == "N"
      DLGL("File is located in the NC file directory")
   ELSE
     DLGL("File is located neither in the HMI nor in the NC
file system")
   ENDIF
ENDIF
```

## 8.3.8 Move Program file function (MP)

### Description

The MP (Move Program) function copies files within the HMI file system or within the NC file system.

### Programming

| | | |
|---|---|---|
| Syntax: | **MP**("Source", "Target") | |
| | MP("CF_CARD:/MPF.DIR/MYPROG.MPF","//NC/MPF.DIR") | |
| Description: | Move file | |
| Parameters: | Source file | Complete path data |
| | Target file | Complete path data |
| Return value | Result of the query | |

### Examples

With return value:

```
MP("//NC/MPF.DIR/123.MPF","//NC/MPF.DIR/ASLAN.MPF",VAR1)        ;Within NC
MP("//NC/MPF.DIR/123.MPF",VAR0,VAR1)                            ;Target via variable
MP(VAR4,VAR0,VAR1)                                              ;Source and target via variable
MP("CF_CARD:/mpf.dir/myprog.mpf","//NC/MPF.DIR/123.MPF",VAR1)   ;From CF card to NC
MP("//NC/MPF.DIR/HOHO.MPF","CF_CARD:/XYZ/123.mpf",VAR1)         ;From NC to CF card
```

```
; With return value:
; VAR1 = 0 Executed
; VAR1 = 1 Not executed
```

## 8.3.9 Select Program file function (SP)

### Description

The SP (Select Program) function selects a file in the active NC file system for execution, i.e. the file must be loaded into the NC beforehand.

### Programming

| Syntax: | **SP**("File") | |
|---|---|---|
| Designation: | Select program | |
| Parameters: | "File" | Complete path name of NC file |

### Example

The following data management syntax is used for this function:

- With return value

  `SP("//NC/MPF.DIR/MYPROG.MPF", VAR1)`

  VAR1 = 0 File was loaded.

  VAR1 = 1 File was not loaded without return value.
- Without return value:

  `SP("//NC/MPF.DIR/MYPROG.MPF")`

## 8.3.10 File accesses: RDFILE, WRFILE, RDLINEFILE, WRLINEFILE

### Description

The RDFILE and WRFILE functions are available for read and write access to files with INI syntax.

The RDLINEFILE and WRLINEFILE functions are available for read and write access to individual lines of a file.

### Programming

| Syntax: | **RDFILE**("File name + path", "Section", "Key") |
|---|---|
| Description: | Reading from a file |

| | | |
|---|---|---|
| Parameters: | File name + path | Path and file name |
| | Section | Section in the INI file |
| | Key | Key in the INI file |

| | | |
|---|---|---|
| Syntax: | **WRFILE**(Value, "File name + path", "Section", "Key") | |
| Description: | Writing to a file | |
| Parameters: | Value | Value to be written |
| | File name + path | Path and file name |
| | Section | Section in the INI file |
| | Key | Key in the INI file |

| | | |
|---|---|---|
| Syntax: | **RDLINEFILE**("File name + path", Line number) | |
| Description: | Reading a line from a file | |
| Parameters: | File name + path | Path and file name |
| | Line number | Line number<br>Numbering begins at 0 for the first line. |

| | | |
|---|---|---|
| Syntax: | **WRLINEFILE**(Value, "File name + path") | |
| Description: | Writing of a line at the end of a file | |
| Parameters: | Value | Value to be written |
| | File name + path | Path and file name |

**Note**

- The files must not be in the file system of the NC (data management).
- If the file does not exist, or the end of the file is reached or other errors occur, the variables FILE_ERR and ERR are set accordingly. You can check whether a file exists first with the Exist Program (EP) file function.
- The processed file is generated in "UTF-8" coding (without BOM (Byte Order Mask). When reading a file, it is expected in "UTF-8" coding.
- You can delete the file explicitly when required with the Delete Program (DP) file function.

## Examples

**Reading from an INI file:**

Requirement/assumption:

Content of file "C:/tmp/myfile.ini":

```
<...>
[MyData]
MyName=Daniel
<...>
```

```
MyVar = RDFILE("C:/tmp/myfile.ini", "MyData", "MyName")
```

Result:

MyVar now contains the value "Daniel".

**Writing to an INI file:**

Requirement/assumption:

VARS=12

```
WRFILE(VARS, "C:/tmp/myfile.ini", "MySession", "NrOfSessions")
```

Result:

Content of file "C:/tmp/myfile.ini":

```
<...>
[MySession]
NrOfSessions=12
<...>
```

**Reading the 4th line of a file:**

Requirement/assumption:

Content of c:/tmp/myfile.mpf:

```
R[0]=0
F3500 G1
MYLOOPIDX1: X0 y-50 Z0
  X150 Y50 Z10
  R[0]=R[0]+1
GOTOB MYLOOPIDX1
M30
```

```
MyVar = RDLINEFILE("C:/tmp/myfile.mpf", 4)
```

Result:

MyVar now contains the value "  R[0]=R[0]+1"

**Writing at the end of a file:**

Requirement/assumption:

```
VARX=123
VARY=456

WRLINEFILE("F100 X" << VARX << " Y" << VARY, "C:/tmp/mypp.mpf")
```

Result:

Content of c:/tmp/mypp.mpf:

```
<...>
F100 X123 Y456
```

## 8.3.11 Dialog line (DLGL)

### Description

It is possible to configure short texts (messages or input tips) for output in the dialog line of the dialog in response to certain situations.

Possible number of characters in the default font size: approx. 50

**Note**

The functions LS, LM, DLGL, EXIT and EXITLS must not be configured in the methods SUSPEND or RESUME. When using the functions in these methods, the execution of the functions is prevented.

### Programming

| | | |
|---|---|---|
| Syntax: | **DLGL**("String") | |
| Description: | Outputs text in the dialog line | |
| Parameters: | String | Text, which is displayed in the dialog line |

### Example

```
IF Var1 > Var2
                              ; The text "Value too large!" appears in the dialog
                              line if variable1 > variable2.
  DLGL("Value too large!")
ENDIF
```

## 8.3.12 DEBUG

### Description

The DEBUG function provides an analysis help during the draft phase of Run MyScreen user screens. An expression in brackets transferred with the DEBUG function is evaluated during runtime. The result is attached to the "easyscreen_log.txt" logbook as a separate entry.

Each entry has the current time stamp in square brackets as prefix (see example below).

It is recommended that the DEBUG outputs are removed in time-critical parts when possible. Particularly after completing the draft phase, it is recommended that you comment out the DEBUG outputs. The write access to the constantly expanding "easyscreen_log.txt" logbook may slow down the processing.

### Programming

| | |
|---|---|
| Syntax: | **DEBUG**(expression) |
| Description: | Makes an entry in the "easyscreen_log.txt" logbook |
| Parameters: | Expression to be evaluated from which an entry is generated in the logbook |

### Example

```
IF Var1 > Var2
  DEBUG("Value of ""Var1"": " << Var1)
                                        ; Entry in the "easyscreen_log_txt:
                                        [10:22:40.445] DEBUG: Value of "Var1":
                                        123.456
ENDIF
```

## 8.3.13 Exit dialog (EXIT)

### Description

The EXIT function is used to exit a dialog and return to the master dialog. If no master dialog is found, you will exit the newly configured user interfaces and return to the standard application.

**Note**

The functions LS, LM, DLGL, EXIT and EXITLS must not be configured in the methods SUSPEND or RESUME. When using the functions in these methods, the execution of the functions is prevented.

### Programming (without parameters)

| | |
|---|---|
| Syntax: | **EXIT** |
| Description: | Exits a dialog |
| Parameters: | - None - |

### Example

```
PRESS(HS1)
  EXIT
END_PRESS
```

### Description

If the current dialog has been called with a transfer variable, the value of the variables can be changed and transferred to the output dialog.

The variable values are each assigned to the variables transferred from the output dialog to the subsequent dialog using the "LM" function. Up to 20 variable values, each separated by a comma, can be transferred.

**Note**

The sequence of variables or variable values must be the same as the sequence of transfer values programmed for the LM function to preclude assignment errors. Any unspecified variable values will not be changed when the transfer is made. The modified transfer variables are immediately valid in the output dialog on execution of the LM function.

### Programming with a transfer variable

| | | |
|---|---|---|
| Syntax: | **EXIT**[(VARx)] | |
| Description: | Exits the dialog and transfers one or more variables | |
| Parameters: | VARx | Label variables |

### Example

```
//M(Screen1)
...
PRESS(HS1)
  LM("SCREEN2","CFI.COM",1, POSX, POSY,
DIAMETER)
                                        ; Interrupt Screen1 and open Screen2. In
                                        doing this, transfer variables POSX, POSY
                                        and DIAMETER.
```

```
  DLGL("Screen2 ended")                    ; After returning from Screen2, the fol-
                                           lowing text appears in the dialog line of
                                           Screen1: Screen2 ended.
END_PRESS
...
//END

//M(Screen2)
...
PRESS(HS1)
  EXIT(5, , CALCULATED_DIAMETER)
                                           ; Exit Screen2 and return to Screen1 in
                                           the line after LM. In doing this, assign
                                           the value 5 to the variable POSX and the
                                           value of the CALCULATED_DIAMETER variable
                                           to the DIAMETER variable. The variable
                                           POSY retains its current value.
END_PRESS
...
//END
```

## 8.3.14 Dynamic manipulation of the lists of toggle fields or list box fields

### Description

The LISTADDITEM, LISTINSERTITEM, LISTDELETEITEM and LISTCLEAR functions are used for the dynamic manipulation of the lists of toggle fields or list box fields.

These functions only have an effect on variables that have their own list, such as:

- "Simple" list
  `DEF VAR_AC1 = (I/* 0,1,2,3,4,5,6,7,8)` or
- "Extended" list
  `DEF VAR_AC2 = (I/* 0="AC0", 1="AC1", 2="AC2", 3="AC3", 4="AC4", 5="AC5", 6="AC6", 7="AC7", 8="AC8").`

If the variable points to an array, e.g. `DEF VAR_AC3 = (I/* MYARRAY)`, these functions are not available as otherwise the global array would be changed.

At least one value must be defined for the variable in the DEF line. This specifies the "Simple" or "Extended" list type.

However, the list can then be completely deleted and, if required, completely rebuilt. The "Simple" or "Extended" type must be retained however, it cannot be changed dynamically.

## Programming

| | | |
|---|---|---|
| Syntax: | **LISTINSERTITEM**(Variable name, Position, ItemValue[, ItemDispValue]) | |
| Description: | Inserts an element at a specific position; reading from a file | |
| Parameters: | Variable name | |
| | Position | Position at which an element is to be inserted in the list |
| | ItemValue | Value of the list entry |
| | ItemDispValue | Value as it is to be displayed in the list |

| | | |
|---|---|---|
| Syntax: | **LISTADDITEM**(Variable name, ItemValue[, ItemDispValue]) | |
| Description: | Adds an element to the end of the list | |
| Parameters: | Variable name | |
| | ItemValue | Value of the list entry |
| | ItemDispValue | Value as it is to be displayed in the list |

| | | |
|---|---|---|
| Syntax: | **LISTDELETEITEM**(Variable name, Position) | |
| Description: | Deletes a specific element | |
| Parameters: | Variable name | |
| | Position | Position of the element to be deleted in the list |

| | | |
|---|---|---|
| Syntax: | **LISTCOUNT**(Variable name) | |
| Description: | Returns the current number of elements in the list | |
| Parameters: | Variable name | |

| | | |
|---|---|---|
| Syntax: | **LISTCLEAR**(Variable name) | |
| Description: | Deletes the complete list | |
| Parameters: | Variable name | |

## Examples

**Note**

The following examples build upon each other. The sequence of the examples is relevant in order to understand the respective results.

Requirement/assumption:

```
DEF VAR_AC = (I/* 0="Off",1="On"/1/,"Switch"/WR2)
```

**Adding an element -1="Undefined" to the end of the list:**

```
LISTADDITEM("VAR_AC", -1, """Undefined""")
```

Result: `0="Off", 1="On", -1="Undefined"`

**Inserting an element 99="Maybe" at position 2:**

```
LISTINSERTITEM("VAR_AC", 2, 99, """Maybe""")
```

Result: `0="Off", 1="On", 99="Maybe", -1="Undefined"`

**Determining the current number of list elements:**

```
REG[10]=LISTCOUNT("VAR_AC")
```

Result: REG[10] = 4

**Deleting the element at position 1:**

```
LISTDELETEITEM("VAR_AC", 1)
```

Result: `0="Off", 99="Maybe", -1="Undefined"`

**Deletion of the complete list:**

```
LISTCLEAR("VAR_AC")
```

Result: List is empty

## 8.3.15 Evaluate (EVAL)

### Description

The EVAL function evaluates a transferred expression and then executes it. With this function, expressions can be programmed during runtime. This can be useful, for example, for indexed access operations to variables.

## Programming

| Syntax: | **EVAL**(exp) | |
|---|---|---|
| Description: | Evaluates an expression | |
| Parameters: | exp | Logic expression |

## Example

```
VAR1=(S)
VAR2=(S)
VAR3=(S)
VAR4=(S)
CHANGE()
  REG[7] = EVAL("VAR"<<REG[5])  ; The expression in parentheses produces VAR3 if the
                                value of REG[5] is equal to 3. The value of VAR3 is,
                                therefore, assigned to REG[7].
  IF REG[5] == 1
    REG[7] = VAR1
  ELSE
    IF REG[5] == 2
      REG[7] = VAR2
    ELSE
      IF REG[5] == 3
        REG[7] = VAR3
      ELSE
        IF REG[5] == 4
          REG[7] = VAR4
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END_CHANGE
```

## 8.3.16 Exit Loading Softkey (EXITLS)

### Description

You can use the EXITLS function to exit the current user interface and load a defined softkey menu.

**Note**

The functions LS, LM, DLGL, EXIT and EXITLS must not be configured in the methods SUSPEND or RESUME. When using the functions in these methods, the execution of the functions is prevented.

### Programming

| Syntax: | **EXITLS**("Softkey menu"[, "Path name"]) | |
|---|---|---|
| Description: | Exits dialog and loads a softkey menu | |
| Parameters: | Softkey menu | Name of the softkey menu to be loaded |
| | Path name | Directory path of the softkey menu to be loaded |

### Example

```
PRESS(HS1)
    EXITLS( "Menu1", "AEDITOR.COM" )
END_PRESS
```

## 8.3.17 Function (FCT)

### Description

The external functions are stored in a DLL file and identified by an entry in the definition lines of the configuration file.

**Note**

The external function must have at least one return parameter.

### Programming

| Syntax: | **FCT**Function name = ("File"/Type of return/Types of permanent parameters/Types of variable parameters) |
|---|---|
| | FCT InitConnection = ("c:\tmp\xyz.dll"/I/R,I,S/I,S) |

| | | |
|---|---|---|
| Description: | An external function can, for example, be called from the LOAD method or executed in the PRESS method. | |
| Parameters: | Function name | Name of external function |
| | File | Complete path to DLL file |
| | Type of return | Data type of the return value |
| | Type of fixed parameter | Value parameter |
| | Type of variable parameter | Reference parameter |
| | The data types are separated by commas. | |

The external function can, for example, be called from the LOAD method or executed in the PRESS method.

**Example:**

```
press(vs4)
RET = InitConnection(VAR1,13,"Servus",VAR2,VAR17)
end_press
```

### Structure of the external function

The external function must take into account a certain, specific signature:

| | | |
|---|---|---|
| Syntax: | extern "C" dllexport void InitConnection (ExtFctStructPtr FctRet, ExtFctStructPtr FctPar, char cNrFctPar) | |
| Description: | DLL export, only when implemented in Windows<br>Specifiers and transfer parameters are strictly defined. The actual call parameters are transferred using the transferred structures. | |
| Parameters: | cNrFctPar | Number of call parameters = number of structure elements in FctPar |
| | FctPar | Pointer to a field of structure elements, which contain the particular call parameter with data type. |
| | FctRet | Pointer to a structure for the function value return with data type. |

### Definition of the transfer structure

```
union CFI_VARIANT
    (
    char                                b;
    short int                           i;
    double                              r;
    char*                               s;
    )
typedef struct ExtFctStructTag
```

```
    (
    char                                cTyp;
    union CFI_VARIANT                   value;
    )ExtFctStruct;
typedef struct ExtFct* ExtFctStructPtr;
```

If the external function is to be developed independently of the platform (Windows, Linux), then it is not permissible to use the keyword __declspec(dllexport). This keyword is only required under Windows. For instance, the following macro can be used under Qt:

```
#ifdef Q_WS_WIN
       #define MY_EXPORT __declspec(dllexport)
#else
       #define MY_EXPORT
#endif
```

The function is declared as follows:

```
external "C" MY_EXPORT void InitConnection
(ExtFctStructPtr FctRet, ExtFctStructPtr FctPar,char cNrFctPar)
```

If the screens, configured with "Run MyScreens", are used on the NCU and PCU/PC, then the extension of the binary file must be omitted:

```
FCT InitConnection = ("xyz"/I/R,I,S/I,S)
```

When the absolute path information is omitted, "Run MyScreens" first searches for the binary file in the configured directory.

## 8.3.18 Generate code (GC)

### Description

The GC (Generate Code) function generates NC code from the OUTPUT method.

### Programming

| | |
|---|---|
| Syntax: | **GC**("Identifier"[,"Target file"][,Opt],[Append]) |
| Description: | Generating NC code (the GC function can only be used within the NC) |

| | | |
|---|---|---|
| Parameters: | Identifier | Name of the OUTPUT method from which code is generated |
| | Target file | Path name of target file for HMI or NC file system<br>If the target file is not specified (only possible within Programming Support), the code will be written to the location of the cursor within the file that is currently open. |
| | Opt | Option for generating comments. |
| | 0: | (Default setting) Generate code with comment for the purpose of recompilability (see also Recompile (Page 171)). |
| | 1: | Do not create comments in the generated code.<br>**Note:** This code cannot be recompiled (see also Recompile (Page 171)). |
| | Append | This parameter is only relevant if a target file is specified. |
| | 0: | (Default setting) If the file already exists, the old content is deleted. |
| | 1: | If the file already exists, the new code is written at the start of the file. |
| | 2: | If the file already exists, the new code is written at the end of the file. |

## Example

```
//M(TestGC/"Code generation:")
DEF VAR1 = (R//1)
DEF VAR2 = (R//2)
DEF D_NAME
LOAD
  VAR1 = 123
  VAR2 = -6
END_LOAD
OUTPUT(CODE1)
  "Cycle123(" VAR1 "," VAR2 ")"
  "M30"
END_OUTPUT

PRESS(VS1)
  D_NAME = "\MPF.DIR\MESSEN.MPF"
  GC("CODE1",D_NAME)                ; Write NC code from the OUTPUT method to file
                                    \MPF.DIR\MESSEN.MPF:
                                    Cycle123(123, -6)
                                    M30
END_PRESS
```

## Storing the target file

The GC function can only be used within the NC. Use the function CP or MP to move the file. See the following examples:

```
; Windows directory
  GC("Code","//NC/MPF.DIR/NC1.MPF")
  MP("//NC/MPF.DIR/NC1.MPF","d:/tmp/WIN1.MPF")

; LOCAL_DRIVE
  GC("Code","//NC/MPF.DIR/NC1.MPF")
  MP("//NC/MPF.DIR/NC1.MPF","LOCAL_DRIVE:/WIN_LD1.MPF")
```

### Recompile

- **No entry for target file:**
  The GC function can only be used in the Programming Support and writes the NC code to the file currently open in the editor. Recompilation of the NC code is possible. If the GC function is configured without a target file being specified under "Run MyScreens", an error message is output when it is executed.
- **Entry for target file:**
  The code generated from the OUTPUT method is entered in the target file. If the target file does not already exist, it is set up in the NC file system. If the target file is stored in the HMI file system, it is stored on the hard disk. User comment lines (information required to recompile code) are not set up, i.e. the code cannot be recompiled.

### Special features of recompilation

The GC function cannot be called in sub-dialogs because variables originating from master dialogs can be used in sub-dialogs. These variables would not, however, be available in response to a direct call.

When generated code is processed manually with the editor, the number of characters for values created by the code generation program must not be changed. Changing these values would make it impossible to recompile the code.

Remedy:

1. Recompile
2. Make change using the configured dialog (e.g. 99 → 101)
3. GC

## 8.3.19 Password functions

### Overview

The following password functions are available:

- Set password
- Delete password
- Change password

### HMI_LOGIN function: Set new password

| Syntax: | **HMI_LOGIN**(Passwd) | |
|---|---|---|
| Description: | A password which sets the current access level is sent to the NCK with the HMI_LOGIN() function. | |
| Parameters: | Passwd | Password |

### Example

```
REG[0] = HMI_LOGIN("CUSTOMER")          ; Result = TRUE if the password has been set
                                        successfully, otherwise FALSE
```

### HMI_LOGOFF function: Delete password

| Syntax: | **HMI_LOGOFF** |
|---|---|
| Description: | The actual access level can be reset using the HMI_LOGOFF function. |
| Parameters: | - None - |

### Example

```
REG[0] = HMI_LOGOFF                     ; Result = TRUE if the password has been de-
                                        leted successfully, otherwise FALSE
```

### HMI_SETPASSWD function: Change password

| Syntax: | **HMI_SETPASSWD**(AC, Passwd) | |
|---|---|---|
| Description: | The password of the current level or a lower level can be changed with the HMI_SETPASSWD function. | |
| Parameters: | AC | Access level |
| | Passwd | Password |

### Example

```
REG[0] = HMI_SETPASSWD(4,"MYPWD")      ; Result = TRUE if the password has been
                                       changed successfully, otherwise FALSE
```

## 8.3.20 Load Array (LA)

### Description

The LA (Load Array) function can be used to load an array from another file.

### Programming

| Syntax: | **LA**(Identifier [, File]) | |
|---|---|---|
| Description: | Loads an array from a file | |
| Parameters: | Identifier | Name of array to be loaded |
| | File | File in which the array is defined |

**Note**

If an array in the current configuration file must be replaced by an array from another configuration file, then both arrays must have the same name.

### Example

```
                                       ; Extract from file  maske.com
DEF VAR2 = (S/*ARR5/"Out"/,"Toggle field")
PRESS(HS5)
  LA("ARR5","arrayext.com")            ; Load array ARR5 from file arrayext.com
  VAR2 = ARR5[0]                       ; "Above"/"Below"/"Right"/"Left" appears
                                       in the VAR2 toggle field
                                       instead of "Out/In"
END_PRESS
//A(ARR5)
("Out"/"In")
//END
                                       ; Extract from file arrayext.com
//A(ARR5)
("Above"/"Below"/"Right"/"Left")
//END
```

**Note**

Please note that a valid value must be assigned to a variable after the LA function has been used to assign another array to the toggle field of the variable.

## 8.3.21 Load Block (LB)

### Description

The LB (Load Block) function can be used to load blocks containing subprograms during runtime. LB should be configured in a LOAD method so that the loaded subprograms can be called at any time.

**Note**

Subprograms can also be defined directly in a dialog so that they do not have to be loaded.

### Programming

| | | |
|---|---|---|
| Syntax: | **LB**("Block name"[,"File"]) | |
| Description: | Loads subprogram during runtime | |
| Parameters: | Block name | Name of block identifier |
| | File | Path name of configuration file<br>Default setting = Current configuration file |

### Example

```
LOAD
  LB("PROG1")              ; Block "PROG1" is searched for in the current configura-
                           tion file and then loaded.
  LB("PROG2","XY.COM")     ; Block "PROG2" is searched for in the configuration file
                           XY.COM and then loaded.
END_LOAD
```

## 8.3.22 Load Mask (LM)

### Description

The LM function loads a new dialog. Depending on the mode of the dialog change (see table below), a message dialog can also be implemented with this function.

**Note**

The functions LS, LM, DLGL, EXIT and EXITLS must not be configured in the methods SUSPEND or RESUME. When using the functions in these methods, the execution of the functions is prevented.

### Master dialog / sub-dialog

A dialog, which calls another dialog, but is not ended itself, is referred to as a master dialog. A dialog that is called by a master dialog is referred to as a sub-dialog.

### Programming

| | | |
|---|---|---|
| Syntax: | **LM**("Identifier"[,"File"] [,MSx [, VARx] ] ) | |
| Description: | Loads a dialog | |
| Parameters: | Identifier | Name of the dialog to be loaded |
| | File | Path name (HMI file system or NC file system) of the configuration file; default setting: Current configuration file |
| | MSx | Mode of dialog change |
| | 0: | (Default setting) The current dialog is rejected, the new dialog is loaded and displayed. EXIT will send you back to the standard application. You can use the MSx parameter to determine whether or not the current dialog should be terminated when changing dialogs. If the current dialog is retained, variables can be transferred to the new dialog.<br>The advantage of the MSx parameter is that the dialogs do not always need to be reinitialized when they are changed. Instead, the data and layout of the current dialog are retained and data transfer is made easier. |
| | 1: | The current master dialog is interrupted when the LM function is initiated. The new sub-dialog is loaded and displayed (e.g. for the implementation of a message dialog). EXIT will end the sub-dialog and return to the point at which the master dialog was interrupted.<br>In the master dialog, the UNLOAD block is not processed during the interruption. |
| | VARx | Precondition: MS1<br>List of variables which can be transferred from the master dialog to the sub-dialog. Up to 20 variables, each separated by a comma, can be transferred. |

**Note**

Parameter VARx transfers only the value of the variable in each case, i.e. variables can be read and written in the sub-dialog, but are not visible in it. Variables can be returned from the sub-dialog to the master dialog by means of the EXIT function.

### Example

**SCREEN1 (master dialog): Jump to the sub-dialog SCREEN2 with variable transfer**

```
PRESS(HS1)

  LM("SCREEN2","CFI.COM",1, POSX, POSY, DIAMETER)
                                    ; Interrupt Screen1 and open Screen2: In doing
                                    this, variables POSX, POSY and DIAMETER are
                                    transferred.
  DLGL("Screen2 ended")             ; After returning from Screen2, the following
                                    text appears in the dialog line of Screen1:
                                    Screen2 ended.
END_PRESS
```

**SCREEN2 (sub-dialog): Return to master dialog SCREEN1 with transfer of the variable contents**

```
PRESS(VS8)

  EXIT(POSX, POSY, DIAMETER)
                                    ; Hide Screen2 and continue Screen1: In doing
                                    this, the changed variables POSX, POSY and DI-
                                    AMETER are transferred.
END_PRESS
```

## 8.3.23 Load Softkey (LS)

### Description

The LS function can be used to display another softkey menu.

**Note**

The functions LS, LM, DLGL, EXIT and EXITLS must not be configured in the methods SUSPEND or RESUME. When using the functions in these methods, the execution of the functions is prevented.

### Programming

| | | |
|---|---|---|
| Syntax: | **LS**("Identifier"[, "File"][, Merge]) | |
| Description: | Displays the softkey menu | |
| Parameters: | Identifier | Name of the softkey menu |
| | File | Path (HMI file system or NC file system) to the configuration file<br>Default setting: Current configuration file |
| | Merge | |
| | 0: | All existing softkeys are deleted; the newly configured softkeys are entered. |
| | 1: | Default setting<br>Only the newly configured softkeys overwrite the available softkeys. The other softkeys (= softkeys of the HMI application) are kept with their functionality and text. |

### Example

```
PRESS(HS4)
  LS("Menu2",,0)      ; Menu2 overwrites the existing softkey menu, the softkeys
                      that are displayed are deleted.
END_PRESS
```

**Note**

As long as the Interpreter has not displayed a dialog, i.e. no LM function has yet been processed, only one LS or one LM command, but no other action, can be configured in the PRESS method of the definition block for the start softkey and the softkey menu.

The LS and LM functions may only be called within a softkey PRESS method and will not react if navigation keys are pressed (PU, PD, SL, SR, SU, SD).

## 8.3.24 Load Grid (LG)

### Description

The table description (grid) can be dynamically provided within methods (e.g. LOAD) using the LG method.

In order to assign a table using the LG method, the variable must have already been defined as a grid variable and cross-referenced to an existing, valid table.

### Programming

| | |
|---|---|
| Syntax: | **LG** (Grid name, Variable name [,File name]) |
| Description: | Loads a table |

| | | |
|---|---|---|
| Parameters: | Grid name | Name of the table (grid) in inverted commas |
| | Variable name | Name of the variable to which the table is to be assigned, in inverted commas |
| | File name | Name of the file in which the table (grid) is defined, in inverted commas. Only needs to be specified if the table is not defined within the file that also contains the definition of the variable. |

### Example

```
//M(MyGridSample/"My Grid Sample")
DEF MyGridVar=(R/% MyGrid1///WR2////100,,351,100)
LOAD
   LG("MyGrid1","MyGridVar","mygrids.com")
END_LOAD
```

**Content of mygrids.com:**

```
//G(MyGrid1/0/5)
(I///,"MyGrid1"/wr1//"1"/80/1)
(R3///"LongText1","R1-R4"/wr2//"$R[1]"/80/1)
(IBB///"LongText2","M2.2-M2.5"/wr2//"M2.2"/80/,1)
(R3///"LongText3","R9,R11,R13,R15"/wr2//"$R[9]"/110/2)
//END
```

## 8.3.25 Multiple selection SWITCH

### Description

You can check a variable for different values with a SWITCH command.

The expression configured in the SWITCH statement is compared in succession to all the values configured in the following CASE statements.

If not the same, a comparison is made with the following CASE statement. If a DEFAULT statement follows and up to now no CASE statement was the same as the expression configured in the SWITCH statement, the statements following the DEFAULT statement are executed.

If the same, the following statements are executed until a CASE, DEFAULT or END_SWITCH statement occurs.

### Example

```
FOCUS
   SWITCH (FOC)
      CASE "VarF"
         DLGL("Variable ""VarF"" has the input focus.")
      CASE "VarZ"
         DLGL("Variable ""VarZ"" has the input focus.")
      DEFAULT
         DLGL("Any other variable has the input focus.")
   END_SWITCH
END_FOCUS
```

## 8.3.26 Multiple Read NC PLC (MRNP)

### Description

This MRNP command transfers several NC/PLC variables in a single register access. This access method is significantly faster than reading via individual access attempts. The NC/PLC variables must be included within an MRNP command of the same area.

The areas of the NC/PLC variables are organized as follows:

- General NC data ($MN..., $SN.., /nck/...)
- Channel-specific NC data ($MC..., $SC.., /channel/...)
- PLC data (DB..., MB.., /plc/...)
- Axis-specific NC data on the same axis ($MA..., $SA..)

### Programming

| | |
|---|---|
| Syntax: | **MRNP**(Variable name 1*Variable name 2[* ...], Register index) |
| Description: | Reads several variables |
| Parameters: | In the variable names, "*" is the separator. The values are transferred to register REG[Register index] and those following in the order that the variable names appear in the command.<br>The following therefore applies:<br>The value of the first variable is located in REG[Register index].<br>The value of the second variable is located in REG[Register index + 1], etc. |

**Note**

It should be noted that the number of registers is restricted and the list of variables cannot exceed 500 characters.

## Example

```
MRNP("$R[0]*$R[1]*$R[2]*$R[3]",1)      ;The values of variables $R[0] to $R[3] are
                                        written to REG[1] to REG[4].
```

## NC variable

All machine and setting data and R-parameters are available, but only certain system variables (see also: AUTOHOTSPOT).

All global and channel-specific user variables (GUDs) can be accessed. However, local and program-global user variables cannot be processed.

| **Machine data** | |
|---|---|
| Global machine data | $MN_... |
| Axis-specific machine data | $MA_... |
| Channel-specific machine data | $MC_... |

| **Setting data** | |
|---|---|
| Global setting data | $SN_... |
| Axis-specific setting data | $SA_... |
| Channel-specific setting data | $SC_... |

| **System variables** | |
|---|---|
| R parameter 1 | $R[1] |

## PLC variable

All PLC data are available.

| **PLC data** | |
|---|---|
| Byte y bit z of data block x | DBx.DBXy.z |
| Byte y of data block x | DBx.DBBy |
| Word y of data block x | DBx.DBWy |
| Double word y v. of data block x | DBx.DBDy |
| Real y of data block x | DBx.DBRy |
| Flag byte x bit y | Mx.y |

| PLC data | |
|---|---|
| Flag byte x | MBx |
| Flag word x | MWx |
| Flag double word x | MDx |
| Input byte x bit y | Ix.y or Ex.y |
| Input byte x | IBx or EBx |
| Input word x | IWx or EWx |
| Input double word x | IDx or EDx |
| Output byte x bit y | Qx.y or Ax.y |
| Output byte x | QBx or ABx |
| Output word x | QWx or AWx |
| Output double word x | QDx or ADx |
| String y with length z from data block x | DBx.DBSy.z |

## 8.3.27 P_LOGICAL_FILEPATH

### Description

The P_LOGICAL_FILEPATH function returns the name and logical path of the currently edited subprogram in the editor.

**Note**

The function is only available in the context of cycle masks.

### Programming

| Syntax: | P_LOGICAL_FILEPATH() | |
|---|---|---|
| Description: | Determines the name and logical path of the currently edited subprogram in the editor<br>e.g. "//NC/MPF.DIR/HUGO.MPF" | |
| Parameters: | - None - | |

### See also

P_REAL_FILEPATH (Page 166)

## 8.3.28 P_REAL_FILEPATH

### Description

The P_REAL_FILEPATH function returns the name and real path of the currently edited subprogram in the editor.

**Note**

The function is only available in the context of cycle masks.

### Programming

| | | |
|---|---|---|
| Syntax: | P_REAL_FILEPATH() | |
| Description: | Determines the name and real path of the currently edited subprogram in the editor<br>e.g. "/_N_MPF_DIR/_N_HUGO_MPF" | |
| Parameters: | - None - | |

### See also

P_LOGICAL_FILEPATH (Page 165)

## 8.3.29 PI services

### Description

The PI_START function starts PI Services (Program Invocation Services) from the PLC in the NC area.

**Note**

A list of the available PI services can be found in the Basic Functions Function Manual.

### Programming

| | | |
|---|---|---|
| Syntax: | **PI_START**("Transfer string") | |
| Description: | Executes the PI service | |
| Parameters: | "Transfer string" | Unlike the OEM documentation, the transfer string should be entered in inverted commas. |

### Example

```
PI_START("/NC,001,_N_LOGOUT")
```

**Note**

Channel-dependent PI Services always refer to the current channel.

PI services of the tool functions (TO area) always refer to the TO area that is assigned to the current channel.

## 8.3.30 Reading (RNP) and writing (WNP) system or user variables

### Description

The RNP (Read NC PLC) command can be used to read NC or PLC variables or machine data.

### Programming

| Syntax: | **RNP** ("System or user variable", value) | |
|---|---|---|
| Description: | Reads NC or PLC variable or machine data | |
| Parameters: | System or user variable | Name of NC or PLC variable |
| | Value | Value that is to be written to the system or user variable.<br>If the value is a String type, it must be written in double quotation marks. |

### Example

```
VAR2=RNP("$AA_IN[2]")         ; Read NC variable
```

### Description

The WNP (Write NC PLC) command can be used to write NC or PLC variables or machine data.

NC/PLC variables are accessed anew every time the WNP function is executed, i.e., NC/PLC access is always executed in a CHANGE method. It is advisable to use this option in cases where a system or user variable changes value frequently. If an NC/PLC variable is to be accessed only once, then it must be configured in a LOAD or UNLOAD method.

### Programming

| Syntax: | **WNP**("System or user variable", value) | |
|---|---|---|
| Description: | Writes NC or PLC variable or machine data | |
| Parameters: | System or user variable | Name of NC or PLC variable |
| | Value | Value that is to be written to the system or user variable.<br>If the value is a String type, it must be written in double quotation marks. |

### Example

```
WNP("DB20.DBB1",1)         ; Write PLC variable
```

## 8.3.31 RESIZE_VAR_IO and RESIZE_VAR_TXT

### Description

The `RESIZE_VAR_IO()` and `RESIZE_VAR_TXT()` commands changes the geometry of the input/output component or text component of a variable.

After setting the new geometry, all fields of the screen are positioned as if the screen had been configured with these positions from the start. In this way, all fields are correctly aligned, depending on the configuration.

### Programming

| Syntax: | **RESIZE_VAR_IO**(Variable name, [X], [Y], [Width], [Height])<br>**RESIZE_VAR_TXT**(Variable name, [X], [Y], [Width], [Height]) | |
|---|---|---|
| Description: | Changes the geometry of the input/output component or text component of a variable. | |
| Parameters: | Variable name | Name of the variable whose geometry of the input/output component or text component of a variable is to be changed. |
| | X | X-coordinate, top left |
| | Y | Y-coordinate, top left |
| | Width | Width |
| | Height | Height |

**Note**

The previous value is retained for each value of X, Y, Width or Height that is not specified. If -1 is specified for one of these values, "Run MyScreens" sets the specified component of the default position.

### Example

```
RESIZE_VAR_IO("MyVar1", 200, , 100)    ; The I/O component of the "MyVar1" variable
                                       is moved to the X position 200 pixels, the
                                       width is set to 100 pixels. The Y position and
                                       the height of the previous value are kept.
```

## 8.3.32 Register (REG)

### Register description

Registers are needed in order to exchange data between different dialogs. Registers are assigned to each dialog. These are created when the first dialog is loaded and assigned the value 0 or an empty string.

**Note**

Registers must not be used directly in an OUTPUT method for generating NC code.

### Programming

| | | |
|---|---|---|
| Syntax: | **REG**[x] | |
| Description: | Defines a register | |
| Parameters: | x | Register index with x = 0...19;<br>Type: REAL or STRING = VARIANT<br>Registers with x ≥ 20 have already been assigned by Siemens. |

### Description of register value

The assignment of values to registers is configured in a method.

**Note**

If a new dialog is generated from an existing dialog by means of the LM function, register content is automatically transferred to the new dialog at the same time and is available for further calculations in the second dialog.

### Programming

| | |
|---|---|
| Syntax: | Identifier.val = Register value<br>or<br>Identifier = Register value |
| Description: | |

| | | |
|---|---|---|
| Parameters: | Identifier | Name of the register |
| | Register value | Value of register |

## Example

```
UNLOAD
  REG[0] = VAR1          ; Assign value of variable 1 to register 0
END_UNLOAD

UNLOAD
  REG[9].VAL = 84        ; Assign value 84 to register 9
END_UNLOAD
                         ; These registers can then be assigned to local variables
                         again in a method in the next dialog.
LOAD
  VAR2 = REG[0]
END_LOAD
```

## Description of register status

The Status property can be used to scan a register for valid content.

One possible use for the register scan function is to ensure that a value is written to a register only if the relevant dialog is a "master dialog".

## Programming

| | | |
|---|---|---|
| Syntax: | Identifier.vld | |
| Description: | The property can only be read. | |
| Parameters: | Identifier | Name of the register |
| Return value: | | The result of the scan can be: |
| | FALSE = | invalid value |
| | TRUE = | valid value |

## Example

```
IF REG[15].VLD == FALSE    ; Scan validity of register value
  REG[15] = 84
ENDIF
VAR1 = REG[9].VLD          ; Assign the value of the REG[9] status request to Var1.
```

## 8.3.33 RETURN

### Description

The RETURN function can be used to prematurely terminate execution of the current subprogram and to return to the branch point of the last CALL command.

If no RETURN command is configured in the subprogram, the subprogram will run to the end before returning to the branch point.

### Programming

| Syntax: | **RETURN** | |
|---|---|---|
| Description: | Returns to the branch point | |
| Parameters: | - None - | |

### Example

```
//B(PROG1)                ; Block start
SUB(UP2)                  ; Start of subprogram
  IF VAR1.val=="Otto"
    VAR1.val="Hans"
    RETURN                ; If the variable value = Otto, the value "Hans" is as-
                          signed to the variable, and the subprogram ends at this
                          point.
  ENDIF
  VAR1.val="Otto"         ; If the variable value ≠ Otto, the value "Otto" is as-
                          signed to the variable.
END_SUB                   ; End of subprogram
//END                     ; Block end
```

## 8.3.34 Recompile

### Description

In the programming support system, it is possible to **recompile** NC code that has been generated with the GC function and to display the variable values in the input/output field of the associated entry dialog again.

## Programming

Variables from the NC code are transferred to the dialog. At the same time, the variable values from the NC code are compared with the calculated variable values from the configuration file. If the values do not coincide, an error message is written to the log book because values have been changed during NC code generation.

If the NC code contains the same variable several times, it is evaluated at the point where it last occurs during recompilation. A warning is also written to the log book.

Variables not utilized in NC code during code generation are stored as user comment. The term "user comment" refers to all information required to recompile codes. User comment must not be altered.

**Note**

The block consisting of NC code and user comment can be recompiled only if it starts at the beginning of a line.

**Examples:**

The programm contains the following NC code:

```
DEF VAR1=(I//101)
OUTPUT(CODE1)
  "X" VAR1 " Y200"
  "X" VAR1 " Y0"
END_OUTPUT
```

The following code is then stored in the parts program:

```
;NCG#TestGC#\cus.dir\aeditor.com#CODE1#1#3#
X101 Y200
X101 Y0
;#END#
```

The Editor reads the following during recompilation:

```
X101 Y200
X222 Y0          ; The value for X has been changed in the part program (X101 → X222)
```

The following value is displayed for VAR1 in the input dialog: `VAR1 = 222`

## See also

Generate code (GC) (Page 153)

## 8.3.35 Recompile without user comment

### Description

In the Programming Support, it is possible to **recompile without user comments** the NC code that has been generated with the GC function and to display the variable values in the input/output field of the associated entry dialog again.

### Programming

The GC command can be executed in the following way in order to suppress comment lines that are generated for standard code generation:

```
GC("CODE1",D_NAME,1)
```

Normally, the resulting code cannot be recompiled. The following steps are required in order to be able to recompile the cycle calls generated in this way:

- **Extend the file "easyscreen.ini"**
  The section `[RECOMPILE_INFO_FILES]` is introduced into the "easyscreen.ini" file. In this section, all ini files are listed that contain descriptions for cycles recompiled without user comment:
  ```
  [RECOMPILE_INFO_FILES]
  IniFile01 = cycles1.ini
  IniFile02 = cycles2.ini
  ```
  Several ini files can be specified, whose names can be freely selected.
- **Creating an ini file for a cycle description**
  Save the ini file with the cycle descriptions under the following path:
  [System user-directory]/cfg
  [System oem-directory]/cfg
  [System addon-directory]/cfg
  A separate section is required for each cycle. The section name corresponds to the name of the cycle:
  ```
  [Cycle123]
  Mname = TestGC
  Dname = testgc.com
  OUTPUT = Code1
  Anzp = 3
  Version = 0
  Code_type = 1
  Icon = cycle123.png
  Desc_Text = This is describing text
  ```

| | |
|---|---|
| Mname | Screen name |
| Dname | Name of the file in which the screen is defined |
| OUTPUT | Name of the relevant OUTPUT method |
| Anzp | Number of parameters of the screen to be recompiled (all with DEF-created variables, also help variables) |
| Version | (optional) version specification for cycle |

| | |
|---|---|
| Icon | (optional) icon for display in the machining step program, format *.png<br>Screen size for corresponding resolution:<br>640 x 480 mm → 16 x 16 pixels<br>800 x 600 mm → 20 x 20 pixels<br>1024 x 768 mm → 26 x 26 pixels<br>1280 x 1024 mm → 26 x 26 pixels<br>1280 x 768 mm → 26 x 26 pixels<br>Storage: [System user-directory]/ico/ico<resolution><br>**Notes:**<br>• For resolutions of 1280, the folder for 1024 x 768 mm is used (only suitable for machining step programs).<br>• The screen size is not just dependent on the resolution, but also on the font size set in the editor. |
| Desc_Text | (optional) Explanation text for display in the machining step program, max. length of string is 17 characters (only suitable for machining step programs) |
| Param_Text | (optional) Text for the parameter list<br>T=%1 D=%2 |

**Note**

**Notes on Icon, Desc_Text and Param_Text:**

1. The user icon is always displayed in the G code editor. The G code icon is always displayed in the ShopMill/ShopTurn editor. This helps to better distinguish between ShopMill/ShopTurn steps and NON-ShopMill/ShopTurn steps.
2. The Run MyScreens step is dependent on the editor setting "Display cycles as machining step". This applies to both the G code and the JobShop editor.
   This means that when "Display cycles as machining step" = "YES", the "Desc_Text" of the Run MyScreens configuration is taken; when "NO", the cycle is displayed.
3. Desc_Text and Param_Text
   – can be specified according to language via the $xxxxx notation.
   – The parameters of the cycle call generated can be accessed via %1, %2, etc. This involves the placeholder being replaced by the parameter that is specified after the '%' in the parameter list generated.

## Example

```
//M(TestGC/"Code generation:")
DEF VAR1 = (R//1)
DEF VAR2 = (R//2)
DEF D_NAME
LOAD
  VAR1 = 123
```

```
  VAR2 = -6
END_LOAD
OUTPUT(CODE1)
  "Cycle123(" VAR1 "," VAR2 ")"
  "M30"
END_OUTPUT

PRESS(VS1)
  D_NAME = "\MPF.DIR\MESSEN.MPF"
  GC("CODE1",D_NAME)                    ;Write NC code from the OUTPUT method to
                                        file \MPF.DIR\MESSEN.MPF:
                                        Cycle123(123, -6)
                                        M30
END_PRESS
```

## Supplementary conditions

- The "Recompile without user comment" functionality does not have the full range of functions as "Recompile with user comment".
  Typical cycle calls, such as MYCYCLE(PAR1, PAR2, PAR3, ...), are supported during "Recompile without user comment". However, there must not be a user comment in the line of the function call. Optional parameters that are not transferred during the function call and are of the string S type must however be at least specified with empty quotation marks, e.g. "". Otherwise, "Run MyScreens" attempts to fill these parameters using commas so that the "filled cycle call" can then be recompiled.
- Parameters of the string type must not have any commas or semicolons in the string to be transferred.
- During "Recompile without user comment", all variables contained in the OUTPUT method must always be within the brackets so that the "Fill in missing cycle parameters with commas" functionality can take effect.

Example:

Permitted:

```
OUTPUT
    "MYCYCLE(" MYPAR1 "," MYPAR2 "," MYPAR3 ")"
END_OUTPUT
```

Not permitted (variable MYCOMMENT is positioned after the closing bracket.):

```
OUTPUT
    "MYCYCLE(" MYPAR1 "," MYPAR2 "," MYPAR3 ")" MYCOMMENT
END_OUTPUT
```

## 8.3.36 Search Forward, Search Backward (SF, SB)

### Description

The **SF, SB (Search Forward, Search Backward)** function is used to search for a string from the current cursor position in the NC program currently selected in the Editor and to output its value.

### Programming

| | | |
|---|---|---|
| Syntax: | **SF**("String") | |
| Designation: | **S**earch **F**orward: Search forward from the current cursor position | |
| Syntax: | **SB**("String") | |
| Designation: | **S**earch **B**ackward: Search backward from the current cursor position | |
| Parameters: | String | Text to be found |

### Rules governing text search

- A blank must be inserted before and after the search concept unit, consisting of search string and its value, in the currently selected NC program.
- The system does not search for concepts within comment text or other strings.
- The value to be output must be a numerical expression. Expressions in the form of "X1=4+5" are not recognized.
- The system recognizes hexadecimal constants in the form of X1='HFFFF', binary constants in the form of X1='B10010' and exponential components in the form of X1='-.5EX-4'.
- The value of a string can be output if it contains the following between string and value:
  - Nothing
  - Blanks
  - Equality sign

### Example

The following notations are possible:

```
X100 Y200                ; The variable Abc is assigned the value 200
Abc = SB("Y")
X100 Y 200               ; The variable Abc is assigned the value 200
Abc = SB("Y")
X100 Y=200               ; The variable Abc is assigned the value 200
Abc = SB("Y")
```

## 8.3.37 SL_HMI_INSTALL_DIR

### Description

The SL_HMI_INSTALL_DIR function returns the path of the installation directory of SINUMERIK Operate.

### Programming

| | | |
|---|---|---|
| Syntax: | SL_HMI_INSTALL_DIR() | |
| Description: | Determines the installation directory of SINUMERIK Operate | |
| Parameters: | - None - | |

## 8.3.38 SL_TEMP_DIR

### Description

The SL_TEMP_DIR function returns a path for temporary files in SINUMERIK Operate.

**Note**

The contents of this directory are not persistent, i.e. they are deleted each time SINUMERIK Operate is started.

### Programming

| | | |
|---|---|---|
| Syntax: | SL_TEMP_DIR() | |
| Description: | Determines the path of the SINUMERIK Operate non-persistent (volatile) directory | |
| Parameters: | - None - | |

## 8.3.39 STRING functions

### Overview

The following functions enable strings to be processed:

- Determine length of string
- Find a character in a string
- Extract substring from left
- Extract substring from right
- Extract substring from mid-string

- Replace substring
- Compare strings
- Insert a string in another string
- Remove a string from a string
- Remove spaces (from left or from the right)
- Insert values or strings with formatting identifiers

### LEN function: Length of a string

| Syntax: | **LEN**(string \| varname) | |
|---|---|---|
| Description: | Determines the number of characters in a string | |
| Parameters: | string | Every valid string expression. NULL is output if string is empty. |
| | varname | Any valid declared variable name |
| | Only one of the two parameters is allowed. | |

### Example

```
DEF VAR01
DEF VAR02

LOAD
  VAR01="HALLO"
  VAR02=LEN(VAR01)          ; Result = 5
END_LOAD
```

### INSTR function: Search for character in string

| Syntax: | **INSTR**(Start, String1, String2 [,Direction]) | |
|---|---|---|
| Description: | Searches for characters | |
| Parameters: | Start | Starting position for searching for string1 in string2. Enter 0 to start searching at the beginning of string2. |
| | String1 | Character that is being searched for. |
| | String2 | Character string in which the search is being made. |
| | Direction (optional) | Direction in which the search is being made.<br>0: From left to right (default setting)<br>1: From right to left |
| | 0 is returned if string1 does not occur in string2. | |

### Example

```
DEF VAR01
DEF VAR02

LOAD
  VAR01="HALLO/WELT"
  VAR02=INST(1,"/",VAR01)        ; Result = 6
END_LOAD
```

### LEFT Function: String from left

| Syntax: | **LEFT**(string, length) | |
|---|---|---|
| Description: | LEFT returns a string containing the specified number of characters starting from the left-hand side of a string. | |
| Parameters: | string | Character string or variable with the character string to be processed |
| | length | Number of characters that are to be read out |

### Example

```
DEF VAR01
DEF VAR02
LOAD
  VAR01="HALLO/WELT"
  VAR02=LEFT(VAR01,5)           ; Result = "HELLO"
END_LOAD
```

### RIGHT function: String from right

| Syntax: | **RIGHT**(string, length) | |
|---|---|---|
| Description: | RIGHT returns a string containing the specified number of characters starting from the right-hand side of a string. | |
| Parameters: | string | Character string or variable with the character string to be processed |
| | length | Number of characters that are to be read out |

### Example

```
DEF VAR01
DEF VAR02
```

```
LOAD
  VAR01="HALLO/WELT"
  VAR02=LEFT(VAR01,4)         ; Result = "WORLD"
END_LOAD
```

## MIDS function: String from mid-string

| Syntax: | **MIDS**(string, start [, length]) | |
|---|---|---|
| Description: | MIDS returns a string containing the specified number of characters starting at the specified position in the string. | |
| Parameters: | string | Character string or variable with the character string to be processed |
| | start | Start from where characters are to be read in the string |
| | length | Number of characters that are to be read out |

## Example

```
DEF VAR01
DEF VAR02
LOAD
  VAR01="HALLO/WELT"
  VAR02=LEFT(VAR01,4,4)            ; Result = "LO/W"
END_LOAD
```

## REPLACE Function: Replacing characters

| Syntax: | **REPLACE**( string, FindString, ReplaceString [, start [, count ] ] ) | |
|---|---|---|
| Description: | REPLACE replaces a character/string in a string with another character/string. | |
| Parameters: | string | String in which FindString is to be replaced with ReplaceString. |
| | FindString | String to be replaced |
| | ReplaceString | Replacement string (is used instead of the FindString) |
| | start | Starting position for search and replace operations |
| | count | Number of characters that are to be searched from the starting position after FindString |
| Return value: | | |
| | string = Empty string | Copy of string |
| | FindString = Empty string | Copy of string |
| | ReplaceString = Empty string | Copy of string, in which all occurrences of FindString are deleted |
| | start > Len(String) | Empty string |
| | count = 0 | Copy of string |

### STRCMP function: Compare strings

| Syntax: | **STRCMP**(string1, string2 [, CaseInsensitive]) | |
|---|---|---|
| Description: | STRCMP compares a character string with another character string. | |
| Parameters: | string1 | Character string that is to be compared with a second character string |
| | string2 | Character string that is to be compared with the first character string |
| | CaseInsensitive | Comparison with/without consideration of upper/lower case:<br>Not specified or FALSE = upper/lower case is considered<br>TRUE = upper/lower case is not considered |

### Example

```
REG[0]=STRCMP("Hugo", "HUGO")              ; Result = 32
                                           ; (<> 0, strings are not identical)

REG[0]=STRCMP("Hugo", "HUGO", TRUE)        ; Result = 0
                                           ; (== 0, stings are identical)
```

### STRINSERT function: Insert string

| Syntax: | **STRINSERT**(string1, string2 , insert position) | |
|---|---|---|
| Description: | STRINSERT inserts a character string at a specific position in a character string. | |
| Parameters: | string1 | Character string in which a character string is to be inserted |
| | string2 | Character string that is to be inserted |
| | insert position | Position at which the character string is to be inserted |

### Example

```
REG[0]=STRINSERT("Hello!", " world", 5)    ; Result = "Hello world!")
```

### STRREMOVE function: Remove string

| Syntax: | **STRREMOVE**(string1, remove position , count) | |
|---|---|---|
| Description: | STRREMOVE removes a specific number of characters at a specific position within a character string. | |
| Parameters: | string1 | Character string from which characters are to be removed |
| | remove position | Position at which characters are to be removed from the character string |
| | count | Number of characters to be removed |

### Example

```
REG[0]=STRREMOVE("Hello world!", 5, 6)     ; Result = "Hello!")
```

## TRIMLEFT function: Remove blanks from the left of the string

| Syntax: | **TRIMLEFT**(string1) | |
|---|---|---|
| Description: | TRIMLEFT removes blanks on the left from a character string. | |
| Parameters: | string1 | Character string from which blanks are to be removed from the left of the character string |

### Example

```
REG[0]=TRIMLEFT("    Hello!")              ; Result = "Hello!")
```

## TRIMRIGHT function: Remove blanks from the right of the string

| Syntax: | **TRIMRIGHT**(string1) | |
|---|---|---|
| Description: | TRIMLEFT removes blanks on the right from a character string. | |
| Parameters: | string1 | Character string from which blanks are to be removed from the right of the character string |

### Example

```
REG[0]=TRIMRIGHT("Hello!    ")             ; Result = "Hello!")
```

## FORMAT function: Insert values or string with formatting identifier

| Syntax: | **FORMAT**(Text with formatting identifier [, Value1] ... [, Value28] ) |
|---|---|
| Description: | Through the use of formatting identifiers, the FORMAT function enables up to 28 values or strings to be inserted at specified positions in a specific text. |

| | | |
|---|---|---|
| Formatting identifiers: | Syntax | %[Flags] [Width] [.decimal places] type |
| | Flags | Optional character for defining output formatting:<br>• Right-justified or left-justified ("-") for left-justified<br>• Add leading zeros ("0")<br>• default: Fill in with blanks if the value to be output has fewer positions than specified with [Width]. |
| | Width | The argument defines the minimum output width for a non-negative number. If the value has fewer positions than specified by the argument, the missing spaces are filled with blanks. |
| | Decimal places | With floating-point numbers, the optional parameter defines the number of decimal places. |
| | Type | The type character defines which data formats are transferred to the print statement.<br>These characters need to be specified.<br>• d: Integer value<br>• f: Floating-point number<br>• s: String<br>• o: Octal<br>• x: Hexadecimal<br>• b: Binary |

### Example

```
DEF VAR1
DEF VAR2
LOAD
VAR1 = 123
VAR2 = FORMAT("Hello %08b %.2f %s!", VAR1 + 1, 987.654321, "world")
                                    ; Result = "Hello 01111100 987.65 world!"
END_LOAD
```

### See also

Use of strings (Page 103)

## 8.3.40 WHILE/UNTIL loops

### Description

A loop can be implemented with the DO LOOP commands. Depending on the configuration, it is run through as long as a condition is satisfied (WHILE) or until a condition occurs (UNTIL).

As loops can impair the system performance, depending on the configuration, they should be used carefully and without time-intensive actions in the loops.

It is recommended that a register (REG[]) be used as run variable, because normal display variables (particularly those with system or user variables connection) also impair the system performance due to the frequent updates or write operations.

The runtime of "Run MyScreens" methods can be determined with the DEBUG function (see Section DEBUG (Page 145)). Problems caused by loops (high CPU load, reduced response capability) may be identified in this way.

**Note**

As each FOR loop can be replaced by a WHILE loop, the syntax to formulate a FOR loop is not supported in EasyScreen.

## Programming

```
DO
   <instructions>
LOOP_WHILE <Condition to continue the loop>

DO
   <instructions>
LOOP_UNTIL <Condition to terminate the loop>

DO_WHILE <Condition to continue the loop>
   <instructions>
LOOP

DO_UNTIL <Condition to terminate the loop>
   <instructions>
LOOP
```

## Example

```
  REG[0] = 5
DO
   DEBUG("OUTER: " << REG[0])
   REG[0] = REG[0] + 1
     REG[1] = -5

   DO
      DEBUG("INNER: " << REG[1])
```

```
      REG[1] = REG[1] + 1
   LOOP_WHILE REG[1] < 0

LOOP_WHILE REG[0] < 10
```

```
  REG[0] = 5
DO
   DEBUG("OUTER: " << REG[0])
   REG[0] = REG[0] + 1
     REG[1] = -5

   DO
      DEBUG("INNER: " << REG[1])
      REG[1] = REG[1] + 1
   LOOP_UNTIL 0 <= REG[1]

LOOP_WHILE 10 > REG[0]
```

```
  REG[0] = 5
DO_WHILE 10 > REG[0]
   DEBUG("OUTER: " << REG[0])
   REG[0] = REG[0] + 1
     REG[1] = -5

   DO_UNTIL 0 <= REG[1]
     DEBUG("INNER: " << REG[1])
     REG[1] = REG[1] + 1
   LOOP

LOOP
```

```
  REG[0] = 5
DO_WHILE 10 > REG[0]
   DEBUG("OUTER: " << REG[0])
   REG[0] = REG[0] + 1
     REG[1] = -5

   DO
     DEBUG("INNER: " << REG[1])
```

```
     REG[1] = REG[1] + 1
   LOOP_UNTIL 0 <= REG[1]

LOOP
```

## 8.3.41 Cyclic execution of scripts: START_TIMER, STOP_TIMER

### Description

SUB methods can be called cyclically with the aid of timers. The START_TIMER() and STOP_TIMER() functions are available for this purpose.

**Note**

Only one timer can be configured for each SUB method.

### Programming

| Syntax: | **START_TIMER**("SUB-Name", Interval) | |
|---|---|---|
| Description: | Starts cyclic processing of a SUB method | |
| Parameters: | SUB name: | Name of the SUB method to be called cyclically |
| | Interval: | Interval in milliseconds |

| Syntax: | **STOP_TIMER**("SUB name", Interval) | |
|---|---|---|
| Description: | Stops cyclic processing of a SUB method | |
| Parameters: | SUB name: | Name of the SUB method whose timer is to be stopped. |

### Example

```
//M(TimerSample/"My timer")
DEF MyVariable=(I//0/,"Number of cyclic calls:"/WR1)
VS1=("Start%ntimer")
VS2=("Stop%ntimer")

SUB(MyTimerSub)
   MyVariable = MyVariable + 1
END_SUB
```

```
//M(TimerSample/"My timer")
PRESS(VS1)
   ;Calls SUB "MyTimerSub" every 1000 milliseconds
   START_TIMER("MyTimerSub", 1000)
END_PRESS

PRESS(VS2)
   STOP_TIMER("MyTimerSub")
END_PRESS
```

If START_TIMER is called again for a timer already assigned to a SUB method, then the new interval is taken over if it is different. Otherwise, the second call is ignored.

The smallest interval for the system is100 milliseconds.

If STOP_TIMER is called for a SUB method for which no timer is currently running, the call is ignored.

# Graphic and logic elements

# 9

## 9.1 Line, dividing line, rectangle, circle and ellipse

### 9.1.1 Defining graphic elements

#### Description

Lines, dividing lines, rectangles and ellipses/circles are configured in the LOAD method:

- Transparent rectangles are created by setting the fill color to the system background color.
- With the ELLIPSE element, a circle is created by setting the height and the width to the same value.
- Horizontal and vertical dividing lines always have the exact window width or window height. No scrollbars are produced by dividing lines.
  If you previously used the RECT element to display dividing lines, e.g. RECT(305,0,1,370,0,0,1), this is detected by the Programming Support and automatically converted to V_SEPARATOR(305,1,"#87a5cd", 1). This applies to both vertical and horizontal dividing lines.

#### LINE element

Programming:

| Syntax: | LINE (x1, y1, x2, y2, color, style, tag, visible) | |
|---|---|---|
| Description: | Defines a line | |
| Parameter: | x1 | Start point x-coordinate |
| | y1 | Start point y-coordinate |
| | x2 | End point x-coordinate |
| | y2 | End point y-coordinate |
| | color | Color of the line |
| | style | Line style:<br>1 = solid<br>2 = dashed<br>3 = dotted<br>4 = dashed and dotted |
| | tag | Use this tag to show, hide or delete a graphic element or group of graphic elements with the functions SHOW_GRAPHIC, HIDE_GRAPHIC and DELETE_GRAPHIC. |
| | visible | Graphic element visible (TRUE/FALSE) |

### RECT element

Programming:

| Syntax: | RECT (x, y, w, h, frame color, fill color, style, tag, visible) | |
|---|---|---|
| Description: | Defines a rectangle | |
| Parameter: | x | x-coordinate, top left |
| | y | y-coordinate, top left |
| | w | Width |
| | h | Height |
| | frame color | Color of the border |
| | fill color | Fill color |
| | style | Border style:<br>1 = solid<br>2 = dashed<br>3 = dotted<br>4 = dashed and dotted |
| | tag | Use this tag to show, hide or delete a graphic element or group of graphic elements with the functions SHOW_GRAPHIC, HIDE_GRAPHIC and DELETE_GRAPHIC. |
| | visible | Graphic element visible (TRUE/FALSE) |

### ELLIPSE element

Programming:

| Syntax: | ELLIPSE(x, y, w, h, frame color, fill color, style, tag, visible) | |
|---|---|---|
| Description: | Defines an ellipse or circle | |
| Parameter: | x | x-coordinate, top left |
| | y | y-coordinate, top left |
| | w | Width |
| | h | Height |
| | frame color | Color of the border |
| | fill color | Fill color |
| | style | Border style:<br>1 = solid<br>2 = dashed<br>3 = dotted<br>4 = dashed and dotted |
| | tag | Use this tag to show, hide or delete a graphic element or group of graphic elements with the functions SHOW_GRAPHIC, HIDE_GRAPHIC and DELETE_GRAPHIC. |
| | visible | Graphic element visible (TRUE/FALSE) |

A circle is created when the values for height and width are the same.

## V_SEPARATOR element

Programming:

| | | |
|---|---|---|
| Syntax: | V_SEPARATOR (x,w,color,pen) | |
| Description: | Defines a vertical dividing line | |
| Parameter: | x | x position |
| | pen width | Line weight |
| | color | Color |
| | style | Line style:<br>1 = solid<br>2 = dashed<br>3 = dotted<br>4 = dashed and dotted |
| | tag | Use this tag to show, hide or delete a graphic element or group of graphic elements with the functions SHOW_GRAPHIC, HIDE_GRAPHIC and DELETE_GRAPHIC. |
| | visible | Graphic element visible (TRUE/FALSE) |

## H_SEPARATOR element

Programming:

| | | |
|---|---|---|
| Syntax: | H_SEPARATOR (y,h,color,pen) | |
| Description: | Defines a horizontal dividing line | |
| Parameter: | y | y position |
| | pen width | Line weight |
| | color | Color |
| | style | Line style:<br>1 = solid<br>2 = dashed<br>3 = dotted<br>4 = dashed and dotted |
| | tag | Use this tag to show, hide or delete a graphic element or group of graphic elements with the functions SHOW_GRAPHIC, HIDE_GRAPHIC and DELETE_GRAPHIC. |
| | visible | Graphic element visible (TRUE/FALSE) |

## More information

The Graphic functions (Page 192) section describes the features SHOW_GRAPHIC, HIDE_GRAPHIC, CLEAR_BACKGROUND, and DELETE_GRAPHIC. These functions allow you to control the display of the graphic elements.

## 9.1.2 Graphic functions

### Overview

The following graphic functions are available:

- CLEAR_BACKGROUND
- DELETE_GRAPHIC
- HIDE_GRAPHIC
- SHOW_GRAPHIC

You can apply these functions to the LINE, RECT, ELLIPSE, V_SEPARATOR and H_SEPARATOR (Page 189) graphic elements.

### CLEAR_BACKGROUND function: Deletion of graphic elements

The CLEAR_BACKGROUND function allows you to delete graphic elements.

| | | |
|---|---|---|
| Syntax: | CLEAR_BACKGROUND() | |
| Description: | Deletion of graphic elements. The memory space taken up by the graphic objects is freed. | |
| Parameter(s): | - None - | |

### DELETE_GRAPHIC function: Deletion of a graphic element or group of graphic elements

The DELETE_GRAPHIC function allows you to delete a specific graphic element or group of graphic elements. A value in the "tag" parameter must be defined for the corresponding graphic elements.

- In contrast to the the CLEAR_BACKGROUND function, your can delete specific graphic elements with DELETE_GRAPHIC.
- In contrast to the the HIDE_GRAPHIC function, the memory space taken up by the graphic objects can be freed with DELETE_GRAPHIC.

| | | |
|---|---|---|
| Syntax: | **DELETE_GRAPHIC**(tag) | |
| Description: | Deletion of a graphic element or group of graphic elements whose value is the same as that of the "tag" parameter. | |
| Parameter(s): | tag | The "tag" parameter can be defined in the graphic elements. You can also define multiple graphic elements with the same value to form a group of graphic elements. |

**Example**

```
LINE(0, 10, 100, 10, "black", "dotted", 1, true)
...
LINE(0, 10, 100, 10, "red", "dotted", 5, true)
LINE(0, 20, 100, 20, "red", "dotted", 5, true)
```

```
LINE(0, 30, 100, 30, "red", "dotted", 5, true)
...
DELETE_GRAPHIC(5)                                    ;the 3 graphic elements
                                                     of type LINE with tag=5
                                                     are deleted
```

## HIDE_GRAPHIC function: Hiding of a graphic element or group of graphic elements

The HIDE_GRAPHIC function allows you to hide a specific graphic element or group of graphic elements. A value in the "tag" parameter must be defined for the corresponding graphic elements.

| Syntax: | HIDE_GRAPHIC(tag) | |
|---|---|---|
| Description: | Hiding of a graphic element or group of graphic elements whose value is the same as that of the "tag" parameter. | |
| Parameter(s): | tag | The "tag" parameter can be defined in the graphic elements. You can also define multiple graphic elements with the same value to form a group of graphic elements. |

**Example**

```
LINE(0, 10, 100, 10, "black", "dotted", 1, true)
...
LINE(0, 10, 100, 10, "red", "dotted", 5, true)
LINE(0, 20, 100, 20, "red", "dotted", 5, true)
LINE(0, 30, 100, 30, "red", "dotted", 5, true)
...
HIDE_GRAPHIC(5)                                      ;the 3 graphic elements
                                                     of type LINE with tag=5
                                                     are hidden
```

## SHOW_GRAPHIC function: Showing of a graphic element or group of graphic elements

The SHOW_GRAPHIC function allows you to show a specific graphic element or group of graphic elements. A value in the "tag" parameter must be defined for the corresponding graphic elements.

| Syntax: | SHOW_GRAPHIC(tag) | |
|---|---|---|
| Description: | Showing of a graphic element or group of graphic elements whose value is the same as that of the "tag" parameter. | |
| Parameter(s): | tag | The "tag" parameter can be defined in the graphic elements. You can also define multiple graphic elements with the same value to form a group of graphic elements. |

**Example**

```
LINE(0, 10, 100, 10, "black", "dotted", 1, true)
...
LINE(0, 10, 100, 10, "red", "dotted", 5, true)
LINE(0, 20, 100, 20, "red", "dotted", 5, true)
LINE(0, 30, 100, 30, "red", "dotted", 5, true)
...
SHOW_GRAPHIC(5)                                          ;the 3 graphic elements
                                                         of type LINE with tag=5
                                                         are shown
```

## 9.2 Defining an array

### Definition

An array can be used to organize data of the same data type stored in the memory in such a way that it is possible to access the data via an index.

### Description

Arrays can be one- or two-dimensional. A one-dimensional array is treated like a two-dimensional array with just one line or column.

Arrays have start identifier //A and end identifier //END. The number of lines and columns is optional. An array is structured in the following way:

### Programming

| | | |
|---|---|---|
| Syntax: | **//A**(Identifier)<br>(a/b...)<br>(c/d...)<br>...<br>//END | |
| Description: | Defines array | |
| Parameters: | Identifier | Name of array |
| | a, b, c, d | Values of array<br>Values of the STRING type must be enclosed in double quotation marks. |

### Example

```
//A(Thread)                    ; Size/lead/core diameter
```

```
(0.3 / 0.075 / 0.202)
(0.4 / 0.1   / 0.270)
(0.5 / 0.125 / 0.338)
(0.6 / 0.15  / 0.406)
(0.8 / 0.2   / 0.540)
(1.0 / 0.25  / 0.676)
(1.2 / 0.25  / 0.676)
(1.4 / 0.3   / 1.010)
(1.7 / 0.35  / 1.246)
//END
```

### 9.2.1 Accessing the value of an array element

#### Description

The value of an array access operation can be transferred with property Value (identifier.val).

The line index (line number of the array) and the column index (column number of the array) each begin at 0. If a line index or column index is outside the array, the value 0 or a blank string is output and the ERR variable is set to TRUE. Variable ERR is also set to TRUE if a search concept cannot be found.

#### Programming

| | |
|---|---|
| Syntax: | Identifier [Z,[M[,C]]].val or<br>Identifier [Z,[M[,C]]] |
| Description: | Access to one-dimensional array with only one column |
| Syntax: | Identifier [S,[M[,C]]].val] or<br>Identifier [S,[M[,C]]] or |
| Description: | Access to one-dimensional array with only one line |
| Syntax: | Identifier [Z,S,[M[,C]]].val or<br>Identifier [Z,S,[M[,C]]] |
| Description: | Access to two-dimensional array |

| | | | |
|---|---|---|---|
| Parameters: | Identifier: | Name of array | |
| | Z: | Line value (line index or search concept) | |
| | S: | Column value (column index or search concept) | |
| | M: | Access mode | |
| | | 0 | Direct |
| | | 1 | Searches the line, column directly |
| | | 2 | Searches the column, line directly |
| | | 3 | Searches |
| | | 4 | Searches line index |
| | | 5 | Searches column index |
| | C: | Compare mode | |
| | | 0 | Search concept must be located in the range of values of the line or column. |
| | | 1 | Search concept must be located exactly. |
| Example: | `VAR1 = MET_G[REG[3],1,0].VAL` | | `;Assign Var1 a value from array MET_G` |

## Access mode

- **"Direct" access mode**
  With "Direct" access mode (M = 0), the array is accessed with the line index in Z and the column index in S. Compare mode C is not evaluated.
- **"Search" access mode**
  In the case of access mode M = 1, 2 or 3, the search always commences in line 0 or column 0.

| Mode M | Line value Z | Column value S | Output value |
|---|---|---|---|
| 0 | Line index | Column index | Value from line Z and column S |
| 1 | Search concept:<br>Search in column 0 | Column index of column from which value is read | Value from line found and column S |
| 2 | Line index of line from which return value is read | Search concept:<br>Search in line 0 | Value from line Z and column found |
| 3 | Search concept:<br>Search in column 0 | Search concept:<br>Search in line 0 | Value from line and column found |
| 4 | Search concept:<br>Search in column S | Column index of search column | Line index |
| 5 | Line index of search line. | Search concept:<br>Search in line Z | Column index |

### Compare mode

When compare mode C = 0 is used, the content of the search line or search column must be sorted in ascending order. If the search concept is smaller than the first element or greater than the last, the value 0 or a blank string is output and the error variable ERR is set to TRUE.

When compare mode C = 1 is used, the search concept must be present in the search line or search column. If the search concept cannot be found, the value 0 or a blank string is output and the error variable ERR is set to TRUE.

## 9.2.2 Example Access to an array element

### Prerequisite

Two arrays are defined below. These are the basis for the following examples:

```
//A(Thread)
                (0.3 / 0.075 / 0.202)
                (0.4 / 0.1   / 0.270)
                (0.5 / 0.125 / 0.338)
                (0.6 / 0.15  / 0.406)
                (0.8 / 0.2   / 0.540)
                (1.0 / 0.25  / 0.676)
                (1.2 / 0.25  / 0.676)
                (1.4 / 0.3   / 1.010)
                (1.7 / 0.35  / 1.246)
//END


//A(Array2)
                ("DES" /     "PTCH" /    "CDM" )
                (0.3 /       0.075 /     0.202 )
                (0.4 /       0.1 /       0.270 )
                (0.5 /       0.125 /     0.338 )
                (0.6 /       0.15 /      0.406 )
                (0.8 /       0.2 /       0.540 )
                (1.0 /       0.25 /      0.676 )
                (1.2 /       0.25 /      0.676 )
                (1.4 /       0.3 /       1.010 )
                (1.7 /       0.35 /      1.246 )
//END
```

### Examples

- **Access mode example 1:**
  The search concept is in Z. This key is always sought in column 0. The value from column S is output with the line index of the concept found.
  ```
  VAR1 = Thread[0.5,1,1]   ;VAR1 has the value 0.125
  ```
  Explanation:
  Search for value 0.5 in column 0 of "Thread" array and output the value found in column 1 of the same line.
- **Access mode example 2:**
  The search concept is in S. This concept is always searched for in line 0. The value from line Z is output with the column index of the concept found:
  ```
  VAR1 = ARRAY2[3,"PTCH",2]   ;VAR1 has the value 0.125
  ```
  Explanation:
  Search for column containing "PTCH" in line 0 of array "Array2". Output the value from the column found and the line with index 3.
- **Access mode example 3:**
  A search concept is in each of Z and S. The line index is searched for in column 0 with the concept in Z and the column index in line 0 with the concept in S. The value from the array is output with the line index and column index found:
  ```
  VAR1 = ARRAY2[0.6,"PTCH",3]   ;VAR1 has the value 0.15
  ```
  Explanation:
  Search for the line with the content 0.6 in column 0 of array "Array2", search for the column with the content "STG" in line 0 of Array2. Transfer the value from the line and column found to VAR1.
- **Access mode example 4:**
  The search concept is in Z. S contains the column index of the column in which concept is being searched for. The line index of the concept found is output:
  ```
  VAR1 = Thread[0.125,1,4]   ;VAR1 has the value 2
  ```
  Explanation:
  Search for value 0.125 in column 1 of array "Thread" and transfer the line index of the value found to VAR1.
- **Access mode example 5:**
  Z contains the line index of line in which concept is being searched for. The search concept is in S. The column index of the concept found is output:
  ```
  VAR1 = Thread[4,0.2,5,1]  ;VAR1 has the value 1
  ```
  Explanation:
  Search in line 4 of the "Thread" array for the value 0.2 and transfer the column index of the value found to VAR1. Comparison mode 1 was selected because the values of line 4 are not sorted in ascending order.

## 9.2.3 Scanning the status of an array element

### Description

The Status property can be used to run a scan to find out whether an array access operation is supplying a valid value.

### Programming

| | |
|---|---|
| Syntax: | Identifier [Z, S, [M[,C]]]**.vld** |
| Description: | The property can only be read. |
| Parameters: | Identifier Name of array |
| Return Value: | FALSE = invalid value |
| | TRUE = valid value |

### Example

```
DEF MPIT = (R///"MPIT",,"MPIT",""/wr3)
DEF PIT  = (R///"PIT",,"PIT",""/wr3)
PRESS(VS1)
  MPIT = 0.6
  IF MET_G[MPIT,0,4,1].VLD == TRUE
    PIT    = MET_G[MPIT,1,0].VAL
    REG[4] = PIT
    REG[1] = "OK"
  ELSE
    REG[1] = "ERROR"
  ENDIF
END_PRESS
```

## 9.3 Table description (grid)

### Definition

In contrast to the array, the values of a table (grid) are continually updated. It is a tabular display of values that can come from the NC or PLC. The table is organized in columns and always has variables of the same type in one column.

### Assignment

The reference to a table description is performed in the variables definition:

- The variables definition determines the values to be displayed, and the definition of table elements determines the appearance and arrangement on the screen window. The properties of the input/output fields from the definition line of the variables are taken over in the table.
- The visible area of the table is determined by the width and height of the input/output field. Any lines or columns than cannot be seen can be displayed by scrolling horizontally and vertically.

### Table identifiers

Identifiers of a table containing NCK or PLC values of the same type, which can be addressed via a channel block. The table identifier is differentiated from limits or toggle fields by the addition of a % sign in front of it. The file containing the table description can be specified by adding a comma after the identifier and then inserting the name of the file.

Identifier of a table for values of the same type, e.g. from the NCK or PLC. The table identifier is specified at the position at which the limit values or the toggle field are configured. The table identifier starts with a leading % character. This can be followed by a file name with a comma as separator. It specifies the file in which the table description is defined. Per default, the table is sought in the configuration file in which the screen is configured.

A table definition can also be loaded dynamically, e.g. in the LOAD method via the Load Grid (LG) function.

### System or user variable

This parameter remains empty for tables because the column definition lines contain detailed information about the variables to be displayed. The table description can be provided dynamically.

### Example

Definition of a MyGridVar variable which displays a "MyGrid1" grid in its input/output field (distance from left: 100, distance from top: Standard, width: 350, height: 100).

```
DEF MyGridVar=(V/% MyGrid1///////100,,350,100)
```

### See also

Parameters of variables (Page 91)

Load Grid (LG) (Page 161)

## 9.3.1 Defining a table (grid)

### Description

The table block comprises:

- Header
- 1 to n column descriptions

### Programming

| | |
|---|---|
| Syntax: | **//G**(table identifier/table type/number of rows/<br>[attribute fixed row],[attribute fixed column]) |
| Description: | Defines a table (grid) |

| | | | |
|---|---|---|---|
| Parameters: | Table identifier | The table identifier is used without a leading % sign. It can only be used once in a dialog. | |
| | Table type | 0 (default) | Table for PLC or user data (NCK- and channel-specific data) |
| | | 1 | and others, reserved |
| | Number of lines | Number of lines including header | |
| | | The fixed line or fixed column is not scrolled. The number of columns is the number of columns configured. | |
| | Display of column header line | -1: | The column header line is not displayed |
| | | 0: | The column header line is displayed (default) |
| | Number of fixed columns | 0: | No fixed column |
| | | 1 ... n: | Number of columns that are to be permanently visible, i.e. not scrolled horizontally |

### Examples

| | |
|---|---|
| `//G(grid1/0/5/-1,2)` | Column header line is hidden and two fixed columns |
| `//G(grid1/0/5/,1)` | Column header line is displayed and one fixed column |
| `//G(grid1/0/5)` | Column header line is displayed and no fixed columns |

## 9.3.2 Defining columns

### Description

For tables (grids), it is advisable to use variables with an index. For PLC or NC variables, the index number with one or more indices is of significance.

The values displayed in a table can be modified directly within the restrictions of the rights granted by the attributes and within any defined limits.

### Programming

| | |
|---|---|
| Syntax: | (Type/Limits/Empty/Long text,column header/Attributes/Help display/ System or user variable/Column width/Offset1, Offset2, Offset3)<br>See also Extended configuration syntax (Page 45). |
| Description: | Defining columns |

| | | |
|---|---|---|
| Parameters: | Similar to variables | |
| | Type | Data type |
| | Limits | Limit value MIN, limit value MAX |
| | Long text, column header | |
| | Attributes | |
| | Help display | |
| | System or user variable | As variable, PLC or NC variables should be entered in double quotation marks. |
| | Column width | Entry in pixels. |
| | Offset | The increment sizes to increment each index in order to fill the column are specified in the assigned offset parameter:<br>• Offset1: Step width for the 1st index<br>• Offset2: Step width for the 2nd index<br>• Offset3: Step width for the 3rd index |

### Column header from text file

The column header can be entered as text or text number ($8xxxx) and is not scrolled.

### Modifying column properties

The column properties which can be modified dynamically (written) are:

- Limits (min,max),
- Column header (st),
- Attributes (wr, ac and li),
- Help screens (hlp)

Column properties are modified via the variable identifier in the definition line and the column index (starting at 1).

Example: `VAR1[1].st="Column 1"`

The wr, ac and li attributes can be specified for column definitions.

### See also

Load Grid (LG) (Page 161)

### 9.3.3 Focus control in the table (cell)

#### Description

The Row and Col properties can be used to set and calculate the focus within a table:

- Identifier.**Row**
- Identifier.**Col**

#### Programming

Each cell in a table has the Val and Vld properties.

In order to read and write cell properties, a line and column index must be specified in addition to the variable identifiers from the definition list.

| | |
|---|---|
| Syntax: | Identifier[Line index, column index].val or<br>Identifier[Line index, column index] |
| Description: | Val properties |
| Syntax: | Identifier[Line index, column index].vld |
| Description: | Vld properties |

#### Example

```
Var1[2,3].val=1.203
```

If the line and column indices are not specified, the indices of the focused cell apply. This corresponds to:

```
Var1.Row =2
Var1.Col=3
Var1.val=1.203
```

## 9.4 Custom widgets

### 9.4.1 Defining custom widgets

#### Description

User-specific display elements are configured in the dialog using a custom widget.

**Software option**

In order to use custom widgets in dialog boxes, you require the following additional software option:

"SINUMERIK Integrate Run MyHMI /3GL" (6FC5800-0AP60-0YB0)

### Programming

| | | |
|---|---|---|
| Definition: | **DEF** (name) | |
| Syntax: | **(W///"","**(library name)**.**(class name)**"////**a**,**b**,**c**,**d**);** | |
| Description: | W | Defining custom widgets |
| Parameters: | Name | Custom widget name, freely selectable |
| | Library name | Can be freely selected, name of the dll (Windows) or (Linux) library file |
| | Class name | Freely selectable, name of the class function from the previously named library |
| | a, b, c, d | Position and size of the configuration |

### Example

A custom widget is defined in the dialog configuration in the following way:

```
DEF Cus = (W///"","slestestcustomwidget.SlEsTestCustomWidget"/////
20,20,250,100);
```

## 9.4.2 Structure of the custom widget library

### Description

Essentially, the custom widget library contains a defined class. The name of this class must be specified in the dialog configuration in addition to the library names. Starting from library names, "Run MyScreens" accesses a dll file with the same name, e.g.:

```
slestestcustomwidget.dll
```

### Programming

The class definition of the dll file should look like this:

```
#define SLESTESTCUSTOMWIDGET_EXPORT Q_DECL_EXPORT


class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
....
public slots:
   bool serialize(const QString& szFilePath, bool bIsStoring);
....
}
```

## 9.4.3 Structure of the custom widget interface

### Description

The library is supplemented by an interface in order to display the custom widget in the dialog. This contains macro definitions with which "Run MyScreens" initiates the custom widget. The interface is available in the form of a cpp file. The file name can be freely selected, e.g.: `sleswidgetfactory.cpp`

### Programming

The interface is defined as follows:

```
#include "slestestcustomwidget.h"      ; The header file for the relevant custom widg-
                                       ets is inserted at the beginning of the file
....
//Makros                               ; Macro definitions are not changed
....
WIDGET_CLASS_EXPORT(SlEsTestCustom-    ; The relevant custom widget is declared at
Widget)                                the end of the file
```

### Example

Content of the file sleswidgetfactory.cpp for a custom widget with the class name SlEsTestCustomWidget":

```
#include <Qt/qglobal.h>
#include "slestestcustomwidget.h"

////////////////////////////////////////////////////////////////////////
// MAKROS FOR PLUGIN DLL-EXPORT - DO NOT CHANGE
////////////////////////////////////////////////////////////////////////

#ifndef Q_EXTERN_C
#ifdef __cplusplus
#define Q_EXTERN_C    extern "C"
#else
#define Q_EXTERN_C    extern
#endif
#endif

#define SL_ES_FCT_NAME(PLUGIN) sl_es_create_ ##PLUGIN
#define SL_ES_CUSTOM_WIDGET_PLUGIN_INSTANTIATE( IMPLEMENTATION , PARAM) \
    { \
    IMPLEMENTATION *i = new PARAM; \
```

```
    return i; \
    }

#ifdef Q_WS_WIN
#   ifdef Q_CC_BOR
#       define EXPORT_SL_ES_CUSTOM_WIDGET_PLUGIN(PLUGIN,PARAM) \
    Q_EXTERN_C __declspec(dllexport) void* \
    __stdcall SL_ES_FCT_NAME(PLUGIN) (QWidget* pParent) \
    SL_ES_CUSTOM_WIDGET_PLUGIN_INSTANTIATE( PLUGIN,PARAM )
#   else
#       define EXPORT_SL_ES_CUSTOM_WIDGET_PLUGIN(PLUGIN,PARAM) \
    Q_EXTERN_C __declspec(dllexport) void* SL_ES_FCT_NAME(PLUGIN) \
    (QWidget* pParent) \
    SL_ES_CUSTOM_WIDGET_PLUGIN_INSTANTIATE( PLUGIN,PARAM )
#   endif
#else
#   define EXPORT_SL_ES_CUSTOM_WIDGET_PLUGIN(PLUGIN,PARAM) \
    Q_EXTERN_C void* SL_ES_FCT_NAME(PLUGIN) (QWidget* pParent) \
    SL_ES_CUSTOM_WIDGET_PLUGIN_INSTANTIATE( PLUGIN,PARAM )
#endif

#define WIDGET_CLASS_EXPORT(CLASSNAME)  \
    EXPORT_SL_ES_CUSTOM_WIDGET_PLUGIN(CLASSNAME,CLASSNAME(pParent))

////////////////////////////////////////////////////////////////////////
// FOR OEM USER - please declare here your widget classes for export
////////////////////////////////////////////////////////////////////////

WIDGET_CLASS_EXPORT(SlEsTestCustomWidget)
```

### 9.4.4 Interaction between custom widget and dialog box - Automatic data exchange

Custom widgets interact with dialog boxes and can display values or manipulate them.

#### Conditions

Automatic data exchange takes place under the following conditions:

| Condition | Direction |
|---|---|
| When starting or recompiling a dialog | Dialog → custom widget |
| When executing the GC command for generating cycle calls | Custom widget → Dialog |

## Programming

The following definitions are necessary for the interaction:

**Expansion of the dialog configuration**

| | | |
|---|---|---|
| Definition: | **DEF** (variable) | |
| Syntax: | **(**(type)**//5/"","**(variable)**","/wr2/)** | |
| Variable type: | Type | Standard input field (no grid or toggle) with any data type (no W) |
| Parameters: | Variable | Any designation of a variable for data exchange |
| Input mode: | wr2 | Reading and writing |

**Example**

```
DEF CUSVAR1 = (R//5/"","CUSVAR1",""/wr2/)
```

**Expansion of the class definition**

In the class definition of the custom widgets, a QProperty must be created whose name is identical to the selected variable of the dialog configuration, e.g.:

```
Q_PROPERTY(double CUSVAR1 READ cusVar1 WRITE setCusVar1);
```

**Example**

The class definition of the dll file should look like this:

```
#define SLESTESTCUSTOMWIDGET_EXPORT Q_DECL_EXPORT


class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
 Q_OBJECT
 Q_PROPERTY(double CUSVAR1  READ cusVar1    WRITE setCusVar1);
....
....
}
```

## 9.4.5 Interaction between custom widget and dialog box - Manual data exchange

Besides automatic data exchange, manual data exchange is also possible. Data exchange is dynamic, i.e. takes place while the dialog box is running. The following actions are possible:

- Properties of the custom widget can be read and written.
- Methods of the custom widget can be called from the Run MyScreens configuration.
- A response to a particular signal of the custom widget can be implemented in order to call subprograms (SUB) in the Run MyScreens configuration.

### 9.4.5.1 Reading and writing properties

## Description

The functions ReadCWProperties and WriteCWProperties are provided in the Run MyScreens configuration for reading and writing properties of the custom widgets.

## Programming

| Syntax: | **ReadCWProperty**("Variablename", "Propertyname") | |
|---|---|---|
| Description: | Reading a property of a custom widget | |
| Parameters: | Variablename | Name of a dialog box variable to which a custom widget is assigned |
| | Propertyname | Name of the custom widget property to be read |
| Return value: | Current value of the custom widget property | |

| Syntax: | **WriteCWProperty**("Variablename", "Propertyname", "Value") | |
|---|---|---|
| Description: | Writing the property of a custom widget | |
| Parameters: | Variable name | Name of a dialog box variable to which a custom widget is assigned |
| | Propertyname | Name of the custom widget property to be written |
| | Value | Value to be written to the property of the custom widget |

## Examples

**Example 1:**

Read property "MyStringVar" of the custom widget that is linked to dialog box variable "MyCWVar1" and assign the value to register 7.

**Custom widget class declaration:**

```
class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
   Q_PROPERTY(QString MyStringVar
              READ myStringVar
              WRITE setMyStringVar);
   ...
```

**Dialog box configuration:**

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
PRESS(VS1)
   REG[7]=ReadCWProperty("MyCWVar1", "MyStringVar")
END_PRESS
```

**Example 2:**

Write the result of the calculation "3 + sin(123.456)" into property "MyRealVar" of the custom widget that is linked to dialog box variable "MyCWVar1."

**Custom widget class declaration:**

```
class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
   Q_PROPERTY(double MyRealVar
              READ myRealVar
              WRITE setMyRealVar);
   ...
```

**Dialog box configuration:**

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
PRESS(VS1)
   WriteCWProperty("MyCWVar1", "MyRealVar", 3 + sin(123.456))
END_PRESS
```

### 9.4.5.2 Executing a method of the custom widget

## Description

The function CallCWMethod is available in the Run MyScreens configuration for executing methods of the custom widget.

The custom widget method to be called must have no more than 10 transfer parameters.

The following transfer parameter data formats are supported:

- bool
- uint
- int
- double
- QString
- QByteArray

## Programming

| | | |
|---|---|---|
| Syntax: | **CallCWMethod**("Variablename", "Methodname[, Argument 0][, Argument 1 ... [,Argument 9]") | |
| Description: | Calling a custom widget method | |
| Parameters: | Variable name | Name of a dialog box variable to which a custom widget is assigned |
| | Method name | Name of the custom widget name to be called |
| | Argument 0 - 9 | Transfer parameter for the custom widget method<br>Supported data formats: see above<br>**Note:** The transfer parameters are always transferred with "ByVal," i.e. only the value is transferred and not, for example, the reference to a variable. |
| Return value: | Return value of the custom widget method<br>The following transfer parameter data formats are supported:<br>• void<br>• bool<br>• uint<br>• int<br>• double<br>• QString<br>• QByteArray<br>**Note**: Even if the data format of the return value of the custom widget method is "void," it must be formally assigned to a variable, for example. | |

## Example

### Custom widget class declaration:

```
class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
   public slots:
       void myFunc1(int nValue, const QString& szString, double dValue);
...
```

### Dialog box configuration:

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
DEF MyStringVar1 = (S)
DEF MyRealVar = (R)

PRESS(VS3)
   REG[9] = CallCWMethod("MyCWVar1", "myFunc1", 1+7, MyStringVar1, sin(MyRealVar) -
8)
END_PRESS
```

## Note

The custom widget must implement the "serialize" method. Here, you have the option of writing the internal data of a custom widget
to a specified file, or restoring it again. This is especially necessary, if, with the "Run MyScreens" screen open, you change to another operating area and then return again. Otherwise, internal data are lost when redisplaying.

| | | |
|---|---|---|
| Syntax: | public slots:<br>bool **serialize**(const QString& szFilePath, bool bIsStoring); | |
| Description: | Reading and writing internal data and states to and from a file | |
| Parameters: | szFilePath | Name of the file with complete path data, in which the internal data and states of the custom widget are written to – or from which they are to be read.<br>If necessary, the file must create the custom widget itself. |
| | bIsStoring | TRUE = write<br>FALSE = read |

### Example

```
bool SlEsTestCustomWidget::serialize(const QString& szFilePath, bool bIsStoring)
{
   QFileInfo fi(szFilePath);
   bool bReturn = false;
   QDir dir;
   if (dir.mkpath(fi.canonicalPath()))
   {
      QFile fileData(szFilePath);
      QIODevice::OpenMode mode;

      if (bIsStoring)
      {
         mode = QIODevice::WriteOnly;
      }
      else
      {
         mode = QIODevice::ReadOnly;
      }

      if (fileData.open(mode))
      {
         QDataStream streamData;
         streamData.setDevice(&fileData);

         if (bIsStoring)
         {
            streamData << m_nDataCount << m_dValueX;
         }
```

```
bool SlEsTestCustomWidget::serialize(const QString& szFilePath, bool bIsStoring)
          else
          {
             streamData >> m_nDataCount >> m_dValueX;
          }

          streamData.setDevice(0);
             fileData.flush();
             fileData.close();
          bReturn = true;
       }
    }
    return bReturn;
}
```

### 9.4.5.3 Response to a custom widget signal

#### Description

In Run MyScreens it is possible to respond to a particular signal (invokeSub()) of the custom widget and call up a subprogram (SUB).

10 global variables, the so-called SIGARG, are available for passing values (custom widget signal -> SUB), which are comparable to registers (REG) in configuration. This is where the values transferred with the custom widget signal are stored.

The following transfer parameter data formats are supported:

- bool
- uint
- int
- double
- QString
- QByteArray

#### Programming

**Calling the subroutine:**

| | |
|---|---|
| Syntax: | `void invokeSub(const QString& rszSignalName, const QVariantList& rvntList);` |
| Description: | Custom widget signal with which a Run MyScreens subprogram is called. |

| | | |
|---|---|---|
| Parameters: | rszSignalName | Name of the Run MyScreens subprogram to be called |
| | rvntList | QVariantList array for transferring parameters that are stored in the global parameter SIGARG and which are available in the configuration.<br>Maximum size: 10 elements<br>Data formats supported: see above<br>**Note:** The transfer parameters are always passed "ByVal," i.e. only the value and not, for example, the reference to a variable is passed. |

**Subroutine to be called:**

| | | |
|---|---|---|
| Syntax: | `SUB(on_<Variablename>_<Signalname>)`<br>`...`<br>`END_SUB` | |
| Description: | Response to a custom widget signal | |
| Parameters: | Variable name | Name of a dialog box variable to which a custom widget is assigned. |
| | Signal name | Name of the custom widget signal |
| | SIGARG 0 - 9 | Transfer parameters for the custom widget method.<br>Supported data formats: See above<br>**Note**: The transfer parameters are always passed "ByVal," i.e. only the value and not, for example, the reference to a variable is passed. |

## Example

**Customer widget class declaration:**

```
class SLESTESTCUSTOMWIDGET_EXPORT SlEsTestCustomWidget  : public QWidget
{
   Q_OBJECT
signals:
   void invokeSub(const QString& szSubName, const QVariantList& vntList);
…
}
```

**Custom widget class:**

```

QVariantList vntList;
vntList << 123.456;
emit invokeSub("MySub", vntList);
```

**Dialog box configuration:**

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
SUB(on_MyCWVar1_MySub)
```

```
DEF MyCWVar1 = (W///,"slestestcustomwidget.SlEsTestCustomWidget")
   DEBUG("SUB(on_MyCWVar1_MySub) was called with parameter: """ << SIGARG[0] <<
""""")
END_SUB
```

**Result "easyscreen_log.txt":**

```
[10:22:40.445] DEBUG: SUB(on_MyCWVar1_MySub) was called with parameter: "123.456"
```

# 9.5 SIEsGraphCustomWidget

## 9.5.1 SIEsGraphCustomWidget

### General

Using SlEsGraphCustomWidget, you can display geometrical objects (point, line, square, square with rounded corners, ellipse, arc, text) and curves from interpolation points (e.g. measured values, characteristic).

The objects are organized in one or several contours. These can then be displayed, either individually or combined; they can also be entered, selected and deleted.

Using functions, the objects are added to the currently selected contour, and these are then drawn in this particular sequence. If you wish to draw the object in a specific color, then you must set the appropriate pen color before adding. All objects subsequently added will be drawn in this pen color. In addition to the pen color, you can also influence the pen width and pen style. For closed objects such as square, square with rounded edges and ellipse, before adding the objects you can set a fill color.

Every contour can be brought into various modes:

- Standard display of all object types.
- Points can be connected to create a polyline, and can therefore be displayed as curve/graph.
- The area between points, lines or arcs up to the X axis can be filled in with the currently selected fill color. For instance, you can use this option to visualize residual material when turning.
  If, in this mode, points are displayed in addition as polyline, then the complete area below the curve up to the X axis is filled. The points, lines and arcs can be located in all four quadrants.

You can move along one or several contours using a special cursor mode. To do this, the cursor is set to the first or last point object of a contour or, starting from the actual cursor position is moved either forward or backward within the contour of a point object.

When required, a second Y axis (right) can be displayed with its own scaling. This is coupled to the first Y axis (left) through an offset and a factor.

Based on a specified X coordinate, using a search function, a point previously added to the contour can be found, and if required, the cursor can be set to this point.

You can also configure contours as ring buffer with an adjustable size.

As a result of the serialization, the actual state of the SIEsGraphCustomWidget can be saved in a file in binary form - and also restored.

The SIEsGraphCustomWidget can be operated using the "Pan" gesture (where the view is shifted) and the "Pinch"/"Spread" gestures (zooming in and out of the view).

Using the mouse, you can shift the view (left mouse button + Move), and you can zoom in and out (mouse wheel).

For performance reasons, the display is not automatically refreshed. Depending on the particular application, you can initiate the refresh by calling the appropriate function.

## Example

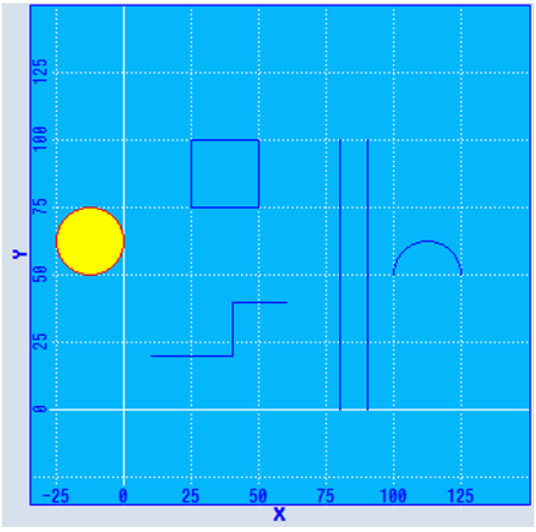



Figure 9-1 SIEsGraphCustomWidget example

```
//M(MyGraphSampleMask/"SlEsGraphCustomWidget Sample")
DEF MyGraphVar = (W///,"slesgraphcustomwidget.SlEsGraphCustomWidget"/////10,10,340,340/0,0,0,0)
VS1=("Add objects",,se1)
LOAD
   WRITECWPROPERTY("MyGraphVar", "AxisNameX", "X")
   WRITECWPROPERTY("MyGraphVar", "AxisNameY", "Y")
   WRITECWPROPERTY("MyGraphVar", "ScaleTextOrientationYAxis", 2)
   WRITECWPROPERTY("MyGraphVar", "KeepAspectRatio", TRUE)

   REG[0]= CALLCWMETHOD("MyGraphVar", "addContour", "MyContour", TRUE)
   REG[0]= CALLCWMETHOD("MyGraphVar", "showContour", "MyContour")
   REG[0]= CALLCWMETHOD("MyGraphVar", "setView", -35, -35, 150, 150)
END_LOAD

PRESS(VS1)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 10, 20, 40, 20)
```

```
//M(MyGraphSampleMask/"SlEsGraphCustomWidget Sample")
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 40, 20, 40, 40)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 40, 40, 60, 40)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 80, 0, 80, 100)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addLine", 90, 0, 90, 100)
   REG[0]= CALLCWMETHOD("MyGraphVar", "addRect", 25, 100, 50, 75)
   REG[0]= CALLCWMETHOD("MyGraphVar", "setPenColor", "#ff0000");red
   REG[0]= CALLCWMETHOD("MyGraphVar", "setFillColor", "#ffff00");yellow
   REG[0]= CALLCWMETHOD("MyGraphVar", "addCircle", -12.5, 62.5, 12.5)
   REG[0]= CALLCWMETHOD("MyGraphVar", "setFillColor");off/default
   REG[0]= CALLCWMETHOD("MyGraphVar", "setPenColor");off/default
   REG[0]= CALLCWMETHOD("MyGraphVar", "addArc", 100, 37.5, 125, 62.5, 0, 180)
   REG[0]= CALLCWMETHOD("MyGraphVar", "update")
END_PRESS
```

## 9.5.2 Notes regarding performance

Depending on the hardware used and the basic utilization of the system, when using SlEsGraphCustomWidgets you must observe the following guide values. In addition to the hardware being used and the basic utilization, the values vary depending on the SIEsGraphCustomWidgets configuration.

Observe these guide values so that the response and stability of the overall system are not diminished.

- Number of SlEsGraphCustomWidgets configured simultaneously (e.g. a maximum of 1)
- Number of configured contours (e.g. a maximum of 6)
- Number of configured graphic objects per contour (e.g. maximum of 1000)
- Frequency of the requested refresh operations (e.g. max. 500 ms)

**Note**

The display is not real-time capable.

## 9.5.3 Reading and writing properties

### Description

The properties listed in the following section are read with ReadCWProperty() and written with WriteCWProperty().

### Examples

Reading the "CursorX" property of the SIEsGraphCustomWidget linked using display variable "MyGraphVar". The result is written to register 0.

```
REG[0] = ReadCWProperty("MyGraphVar", "CursorX")
```

Writes value "MyFirstContour" to the "SelectedContour" property of the SIEsGraphCustomWidget linked using display variable "MyGraphVar".

```
WriteCWProperty("MyGraphVar ", "SelectedContour", "MyFirstContour")
```

## 9.5.4 Properties

### Overview

| Property | Description |
|---|---|
| AxisNameX | Axis identifiers, X axis |
| AxisNameY | Axis identifiers, Y axis |
| AxisNameY2 | Axis identifiers, second Y axis (right) |
| AxisY2Visible | Display/hide second Y axis (right) |
| AxisY2Offset | Offset second Y axis (right) |
| AxisY2Factor | Factor second Y axis (right) |
| ScaleTextEmbedded | Position of the scaling texts |
| ScaleTextOrientationYAxis | Alignment of the scaling texts of the Y axis |
| KeepAspectRatio | Keep aspect ratios |
| SelectedContour | Name of the currently selected contour |
| BackColor | Background color |
| ChartBackColor | Background color of the drawing area |
| ForeColor | Foreground color for text and default pen color |
| ForeColorY2 | Foreground color for the texts of the second Y axis (right) |
| GridColor | Color of the grid lines |
| GridColorY2 | Color of the horizontal grid lines of the second Y axis (right) |
| CursorColor | Color of the cursor crosshairs |
| ShowCursor | Display/hide cursor |
| CursorX | Cursor X position |
| CursorY | Cursor Y position |
| CursorY2 | Cursor Y position referred to the second Y axis (right) |
| CursorStyle | Cursor display type |
| ViewMoveZoomMode | Response when zooming and shifting using gestures |

### AxisNameX – axis identifiers, X axis

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**AxisNameX**")<br>WriteCWProperty(GraphVarName, "**AxisNameX**", Value) | |
| Description: | Reads or sets the identifier of the X axis. If an identifier is not specified, then the space available for the drawing area is fully utilized. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (QString) |
| | Value | Value to be set (QString) |

### AxisNameX – axis identifiers, Y axis

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**AxisNameY**")<br>WriteCWProperty(GraphVarName, "**AxisNameY**", Value) | |
| Description: | Reads or sets the identifier of the Y axis. If an identifier is not specified, then the space available for the drawing area is fully utilized. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (QString) |
| | Value | Value to be set (QString) |

### AxisY2Visible – display/hide second Y axis (right)

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**AxisY2Visible**")<br>WriteCWProperty(GraphVarName, "**AxisY2Visible**", Value) | |
| Description: | Displays/hides second Y axis (right) | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE or FALSE |

### AxisY2Offset – offset second Y axis (right)

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**AxisY2Offset**")<br>WriteCWProperty(GraphVarName, "**AxisY2Offset**", Value) | |
| Description: | Offset of the second Y axis (right) referred to the first Y axis (left).<br>See example for AxisY2Factor property. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (double) |
| | Value | Value to be set (double). |

## AxisY2Factor – factor second Y axis (right)

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**AxisY2Factor**") WriteCWProperty(GraphVarName, "**AxisY2Factor**", Value) | |
| Description: | Factor of the second Y axis (right) referred to the first Y axis (left). | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraphCustomWidget |
| | Return Value | Read value of the property (double) |
| | Value | Value to be set (double). |

Together with the "AxisY2Offset" property, it is possible to display a second Y axis (right) with its own scaling. The scale is coupled to the first Y axis (left) through an offset and a factor.

Formula for converting Y2 to Y:

Y = Y2 / factor - offset

Formula for converting Y to Y2:

Y2 = (Y + offset) * factor

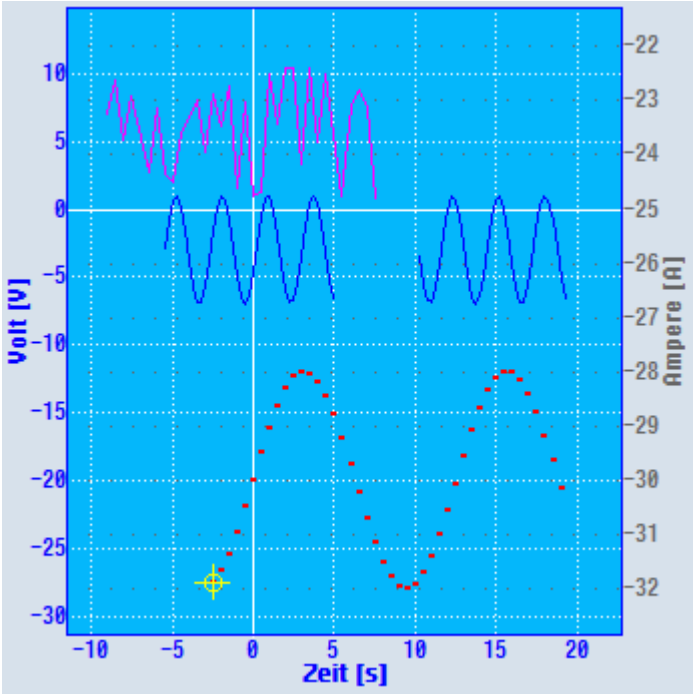



Figure 9-2 Example with second Y axis with own scaling

## Example

Y: 0 to 200 should correspond to Y2: -25 to 25.

- Offset: -100
- Factor: 0.25

**Calculation example:**

Y2 = -30

Y = -30 / 0.25 – (-100) = -20

**Calculation example:**

Y = 0

Y2 = (0 + (-100)) * 0.25 = -25

The X axis is the same for both coordinate systems.

You can set the colors using setForeColorY2() and setGridColorY2() (see colors).

The axes are labeled with setAxisNameY2() (see axis identifiers).

## ScaleTextEmbedded - position of the scaling texts

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**ScaleTextEmbedded**")<br>WriteCWProperty(GraphVarName, "**ScaleTextEmbedded**", Value) | |
| Description: | Using this property you define whether the scaling is positioned inside or outside the drawing area. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE or FALSE |

## Example

Scaling text outside the drawing area:

ScaleTextEmbedded = FALSE

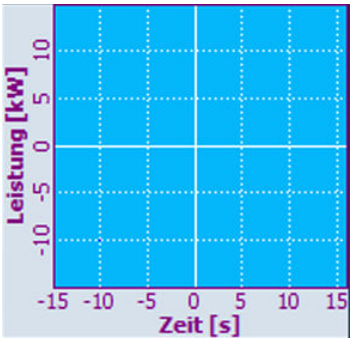



Figure 9-3 Scaling text outside the drawing area

Scaling text within the drawing area:

ScaleTextEmbedded = TRUE

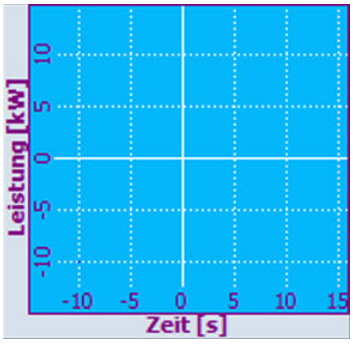



Figure 9-4 Scaling text within the drawing area

## ScaleTextOrientationYAxis - alignment of the scaling texts of the Y axis

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**ScaleTextOrientationYAxis**")<br>WriteCWProperty(GraphVarName, "**ScaleTextOrientationYAxis**", Value) | |
| Description: | Using this function, the scaling texts of the Y axis are aligned vertically. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (int) |
| | Value | Value to be set (int): 1 (= horizontal) or 2 (= vertical) |

## Example

Scaling text within the drawing area:

- ScaleTextEmbedded = TRUE

Horizontal text alignment:

- ScaleTextOrientationYAxis = 1

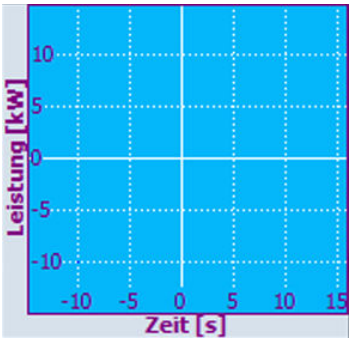



Figure 9-5 Scaling text within the drawing area, horizontal text alignment

Vertical text alignment:

- ScaleTextOrientationYAxis = 2

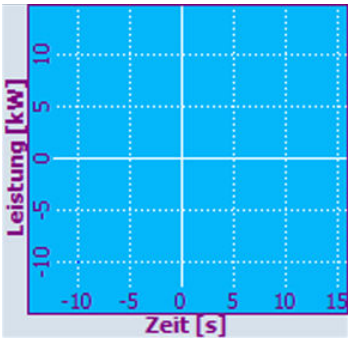



Figure 9-6 Scaling text within the drawing area, vertical text alignment

Scaling text outside the drawing area:

- ScaleTextEmbedded = FALSE

Horizontal text alignment:

- ScaleTextOrientationYAxis = 1

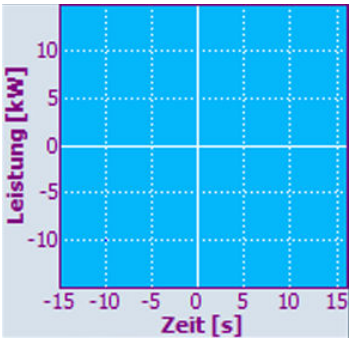



Figure 9-7 Scaling text outside the drawing area, horizontal text alignment

Vertical text alignment:

- ScaleTextOrientationYAxis = 2

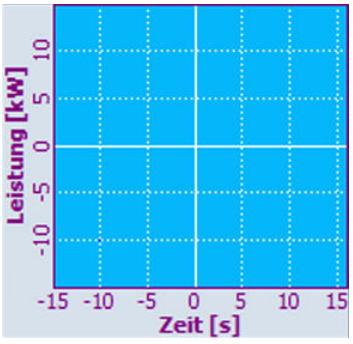



Figure 9-8 Scaling text outside the drawing area, vertical text alignment

## KeepAspectRatio – keep aspect ratio

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**KeepAspectRatio**")<br>WriteCWProperty(GraphVarName, "**KeepAspectRatio**", Value) | |
| Description: | With this property, you define as to whether the view defined using setView() should be automatically aligned, adapted or extended, so that the aspect ratio from X to Y axis always remains the same. This property is especially relevant when displaying geometrical figures. For example, a circle or square is displayed as such, and does not become an ellipse or rectangle. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (bool) |
| | Return Value | Value to be set (bool): TRUE or FALSE |

## Example

```
setView(-15, -15, 15, 15)
setFillColor("#A0A0A4")
addCircle(0, 0, 5)
KeepAspectRatio = TRUE
```

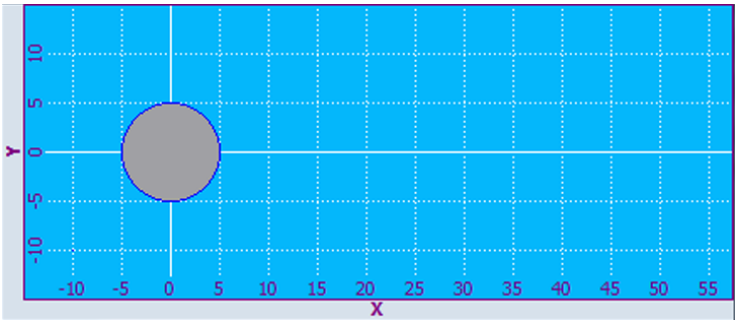



Figure 9-9 Aspect ratio from X to Y axis remain the same

```
KeepAspectRatio = FALSE
```

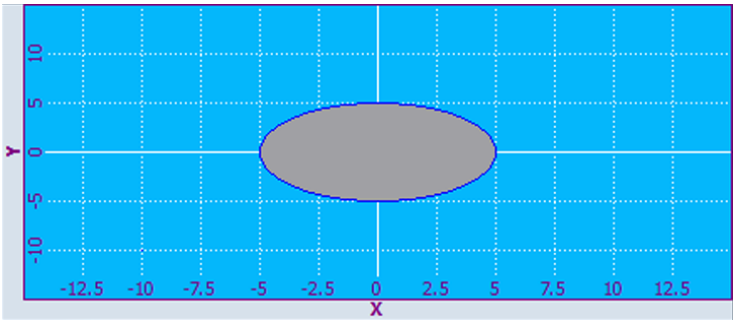



Figure 9-10 Aspect ratio from X to Y axis distorted

### SelectedContour – name of the currently selected contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, " **SelectedContour**")<br>WriteCWProperty(GraphVarName, " **SelectedContour**", Value) | |
| Description: | Reads or sets the selection of the actual contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (QString) |
| | Value | Value to be set (QString) |

**Note**

In order to be able to execute operations on a contour, e.g. to add a graphic object, you must first select this.

### BackColor – background color

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, " **BackColor**")<br>WriteCWProperty(GraphVarName, " **BackColor**", Value) | |
| Description: | The background color for labeling axes depends on the ScaleTextInsideChartArea property - also the scaling texts. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read color of the property (QString) |
| | Value | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### ChartBackColor – background color of the drawing area

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**ChartBackColor**")<br>WriteCWProperty(GraphVarName, "**ChartBackColor**", Value) | |
| Description: | Background color for the drawing area. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read color of the property (QString) |
| | Value | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### ForeColor - foreground color for labeling and default pen color

| | |
|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**ForeColor**")<br>WriteCWProperty(GraphVarName, "**ForeColor**", Value) |
| Description: | Foreground color for text and default pen color. |

| | | |
|---|---|---|
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read color of the property (QString) |
| | Value | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### ForeColorY2 - foreground color for labeling the second Y axis (right)

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**ForeColorY2**")<br>WriteCWProperty(GraphVarName, "**ForeColorY2**", Value) | |
| Description: | Foreground color for the texts of the second Y axis (right). | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read color of the property (QString) |
| | Value | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### GridColor - color of the grid lines

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**GridColor**")<br>WriteCWProperty(GraphVarName, "**GridColor**", Value) | |
| Description: | Color of the grid lines. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read color of the property (QString) |
| | Value | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### GridColorY2 - color of the horizontal grid lines of the second Y axis (right)

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**GridColorY2**")<br>WriteCWProperty(GraphVarName, "**GridColorY2**", Value) | |
| Description: | Color of the horizontal grid lines of the second Y axis (right). | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read color of the property (QString) |
| | Value | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### CursorColor - color of the cursor

| | |
|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**CursorColor**")<br>WriteCWProperty(GraphVarName, "**CursorColor**", Value) |
| Description: | Color of the cursor. |

| | | |
|---|---|---|
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read color of the property (QString) |
| | Value | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### ShowCursor - Display/hide cursor

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, " **ShowCursor**")<br>WriteCWProperty(GraphVarName, " **ShowCursor**", Value) | |
| Description: | Displays/hides cursor. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE or FALSE |

**Note**

This function automatically refreshes the display.

### CursorX – cursor X position

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**CursorX**")<br>WriteCWProperty(GraphVarName, "**CursorX**", Value) | |
| Description: | X position of the cursor. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (double) |
| | Value | Value to be set (double). |

**Note**

This function automatically refreshes the display.

### CursorY – cursor Y position

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**CursorY**")<br>WriteCWProperty(GraphVarName, "**CursorY**", Value) | |
| Description: | Y position of the cursor. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (double) |
| | Value | Value to be set (double). |

**Note**

This function automatically refreshes the display.

### CursorY2 - cursor Y position referred to the second Y axis (right)

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**CursorY2**")<br>WriteCWProperty(GraphVarName, "**CursorY2**", Value) | |
| Description: | Y position of the cursor referred to the second Y axis (right). | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (double) |
| | Value | Value to be set (double). |

**Note**

This function automatically refreshes the display.

### CursorStyle – cursor display type

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**CursorStyle**")<br>WriteCWProperty(GraphVarName, "**CursorStyle**", Value) | |
| Description: | Cursor display type. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (int) |
| | Value | Value to be set (int):<br>0 (= cross)<br>1 (= crosshairs)<br>2 (= vertical line)<br>3 (= horizontal line)<br>4 (= horizontal and vertical line)) |

### ViewMoveZoomMode – response when zooming and shifting using gestures

| | |
|---|---|
| Syntax: | ReturnValue = ReadCWProperty(GraphVarName, "**ViewMoveZoomMode**")<br>WriteCWProperty(GraphVarName, "**ViewMoveZoomMode**", Value) |
| Description: | The displayed view of the SIEsGraphCustomWidgets can be changed using gestures. This property allows you to define which type should be permitted. For example, in certain applications it may make sense that the view is only horizontally shifted and zoomed, e.g. when displaying measured value characteristics. |

| | | |
|---|---|---|
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Read value of the property (int) |
| | Value | Value to be set (int):<br>0 (= off)<br>1 (= only horizontal)<br>2 (= only vertical)<br>3 (= horizontal and vertical) |

## 9.5.5 Functions

You can call the subsequently listed functions using the CallCWMethod() function.

### Example

Setting the view of the SIEsGraphCustomWidget link using display variable "MyGraphVar".

The result is written to register 0.

```
REG[0] = CallCWMethod("MyGraphVar", "setView", -8, -5, 11- 20)
```

### Overview

| Function | Description |
|---|---|
| setView | Sets coordinate system |
| setMaxContourObjects | Sets maximum number of objects for a contour (ring buffer) |
| addContour | Adds a contour |
| showContour | Shows a contour |
| hideContour | Hides a contour |
| hideAllContours | Hides all contours |
| removeContour | Removes a contour |
| clearContour | Deletes graphic object of a contour |
| fitViewToContours | Automatic adaptation of the view |
| fitViewToContour | Automatic adaptation of the view |
| findX | Searches in a contour |
| setPolylineMode | Displays points of a contour as polyline/curve |
| setIntegralFillMode | Displays points of a contour as polyline/curve |
| repaint, update | Refreshes view |
| addPoint | Adds a point to a contour |
| addLine | Adds a line to a contour |
| addRect | Adds a rectangle to a contour |
| addRoundedRect | Adds a rectangle with rounded corners to a contour |
| addEllipse | Adds an ellipse to a contour |
| addCircle | Adds a circle to a contour |
| addArc | Adds an arc to a contour |

| Function | Description |
|---|---|
| addText | Adds text to a contour |
| setPenWidth | Sets the pen width |
| setPenStyle | Sets the pen style |
| setPenColor | Sets the pen color |
| setFillColor | Sets fill color |
| setCursorPosition | Positions the cursor |
| setCursorPositionY2 | Positions the cursor referred to the second Y axis (right) |
| setCursorOnContour | Positions the cursor on contour |
| moveCursorOnContourBegin | Positions the cursor to the first graphic object of a contour |
| moveCursorOnContourEnd | Positions the cursor to the last graphic object of a contour |
| moveCursorOnContourNext | Positions the cursor to the next graphic object of a contour |
| serialize | Saves/restores the actual state |

## setView – sets the coordinate system

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setView**", x1, y1, x2, y2) | |
| Description: | Using this function, you define the size of the coordinate system that should be displayed within the SIEsGraphCustomWidget. See also KeepAspectRatio property. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraphCustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x1 | left-hand edge (double) |
| | y1 | upper edge (double) |
| | x2 | right-hand edge (double) |
| | y2 | lower edge (double) |

**Note**

The function automatically refreshes the display.

## Example

```
setView(-8, -5, 11, 20)
AxisNameX = "X"
AxisNameY = "Y"
ScaleTextOrientationYAxis = 1
ScaleTextEmbedded = false
KeepAspectRatio = false
update
```

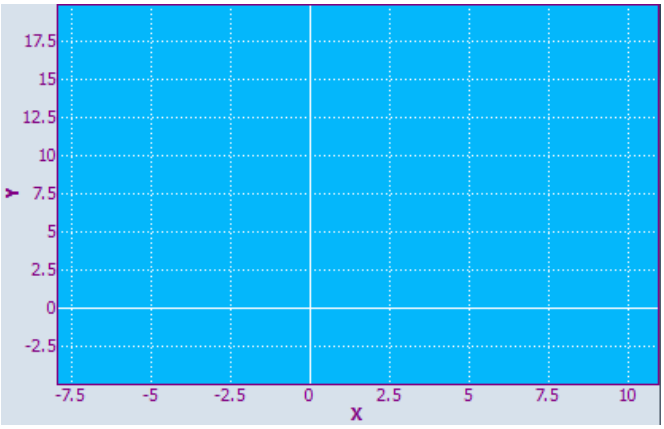



Figure 9-11 Example - setView

## setMaxContourObjects – sets the maximum number of objects of a contour (ring buffer)

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setMaxContourObjects**", Value)<br>ReturnValue = CallCWMethod(GraphVarName, "**setMaxContourObjects**", Value, ContourName) | |
| Description: | Using this function, you can define the size of a contour ring buffer. If the number of objects added to the contour exceeds this limit, then the oldest object is deleted. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | Value | Maximum number of graphic objects (uint) |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

**Note**

The function automatically refreshes the display.

## addContour – add contour

| | |
|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addContour**", ContourName)<br>ReturnValue = CallCWMethod(GraphVarName, "**addContour**", ContourName, Selected)<br>ReturnValue = CallCWMethod(GraphVarName, "**addContour**", ContourName, Selected, AssignedY2) |
| Description: | Using this function, you can define a new contour.<br>Optionally, you can assign the contour to the coordinate system of the second Y axis (right). |

| | | |
|---|---|---|
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |
| | Selected | Contour should be selected (bool): TRUE or FALSE |
| | AssignedY2 | Contour should be assigned to the second Y axis (right) (bool): TRUE or FALSE |

### showContour – shows a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**showContour**")<br>ReturnValue = CallCWMethod(GraphVarName, "**showContour**", ContourName) | |
| Description: | This function makes a contour visible. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

### hideContour – hide contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**hideContour**")<br>ReturnValue = CallCWMethod(GraphVarName, "**hideContour**", ContourName) | |
| Description: | This function hides a contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

### showAllContours – shows all contours

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**showAllContours**") | |
| Description: | This function makes all contours visible. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |

### hideAllContours – hide all contours

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**hideAllContours**") | |
| Description: | This function hides all contours. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |

### removeContour – hides a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**removeContour**")<br>ReturnValue = CallCWMethod(GraphVarName, "**removeContour**", ContourName) | |
| Description: | Removes a contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

### clearContour – delete graphic object of a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**clearContour**")<br>ReturnValue = CallCWMethod(GraphVarName, "**clearContour**", ContourName) | |
| Description: | Deletes all graphic objects on a contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

### fitViewToContours – automatic adaptation of the view

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**fitViewToContours**")<br>ReturnValue = CallCWMethod(GraphVarName, "**fitViewToContours**", OnlyVisible) | |
| Description: | Using this function, the view is automatically adapted to the contours. Depending on the transfer parameter, you define whether only all visible contours can be seen or all contours can be seen. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | OnlyVisible | Value to be set (bool): TRUE or FALSE |

### fitViewToContour – automatic adaptation of the view

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**fitViewToContour**", ContourName) | |
| Description: | Using this function, the view is automatically adapted so that only a specific contours visible. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraphCustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString) |

### findX – search in a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**findX**", x)<br>ReturnValue = CallCWMethod(GraphVarName, "**findX**", x, ContourName)<br>ReturnValue = CallCWMethod(GraphVarName, "**findX**", x, ContourName, SetCursor) | |
| Description: | Using this function, you can search for a previously defined point with a specific X coordinate in a contour. You obtain the Y coordinate as result. Optionally, you can also simultaneously position the cursor to this point. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraphCustomWidget |
| | Return Value | Error code (QString): If successful, the associated Y value is supplied |
| | x | X value to be searched for (double) |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour |
| | SetCursor | Set cursor (bool): TRUE or FALSE |

**Note**

The display is automatically updated if you set the cursor using this function.

### setPolylineMode – displays points on the contour as polyline/curve

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setPolylineMode**", SetPoly) | |
| Description: | All of the points of the actual contour added after this function call are connected to create a polyline/curve. This function is contour-specific, and can be activated and deactivated on a segment-for-segment basis. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraphCustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | SetPoly | PolylineMode (bool): TRUE = on |

## Example

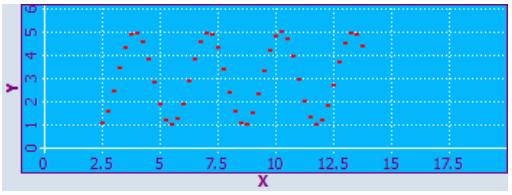



Figure 9-12 Example - PolylineMode: Off

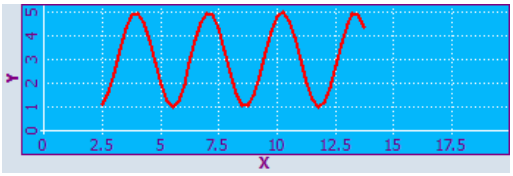



Figure 9-13 Example - PolylineMode: To

## setIntegralFillMode – fills the areas between points, lines and arcs and the X axis

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setIntegralFillMode**", SetIntFill) | |
| Description: | Areas between points, lines and arcs of the X axis are filled with the current fill color (independent of whether the points are +Y or –Y).<br>This function is contour-specific, and can be activated and deactivated on a segment-for-segment basis. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraphCustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | SetIntFill | IntegralFillMode (bool): TRUE = on |

## Example

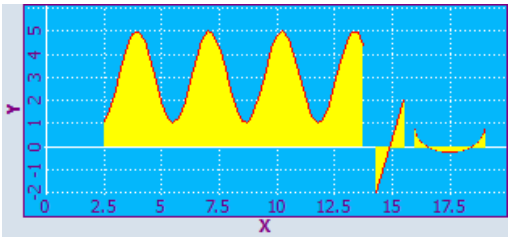



Figure 9-14 Example - setIntegralFillMode: on

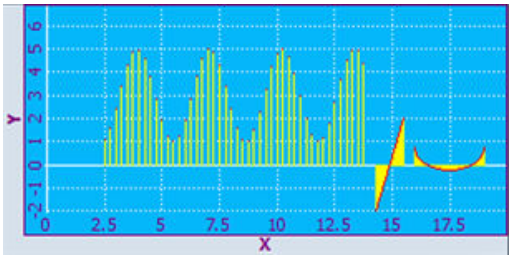



Figure 9-15 Example - setIntegralFillMode: Off

## repaint, update – update view

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**repaint**")<br>ReturnValue = CallCWMethod(GraphVarName, "**update**") | |
| Description: | Using this function, you can manually refresh the display. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |

- update()
  The function refreshes the view, assuming that the Widget refresh is not deactivated, and the Widget is not hidden. The function is optimized so that the application performance is maintained, and the view does not flicker as a result of excessively high refresh rates.
- repaint()
  The function refreshes the view directly after the function call. Therefore, you should only use the repaint function if an immediate refresh is absolutely necessary, e.g. for animation purposes

It is recommended that you always work with the update() function. Internally, the SIEsGraphCustomWidget always works with the function update() e.g. for setView().

## addPoint – add point to a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addPoint**", x, y)<br>ReturnValue = CallCWMethod(GraphVarName, "**addPoint**", x, y, DummyPoint) | |
| Description: | Adds a point to the currently selected contour Further, you can set a dummy point; although it does not show up in the drawing, it can be positioned with the cursor. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x | x coordinate (double) |
| | y | y coordinate (double) |
| | DummyPoint | Point is treated as invisible (bool): TRUE = yes |

### addLine – add line to a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addLine**", x1, y1, x2, y2) | |
| Description: | Adds a line to the currently selected contour | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x1 | x coordinate of the starting point (double) |
| | y1 | y coordinate of the starting point (double) |
| | x2 | x coordinate of the end point (double) |
| | y2 | y coordinate of the end point (double) |

### addRect – add rectangle to a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addRect**", x1, y1, x2, y2) | |
| Description: | Adds a rectangle to the currently selected contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x1 | left-hand edge (double) |
| | y1 | left-hand edge (double) |
| | x2 | right-hand edge (double) |
| | y2 | lower edge (double) |

### addRoundedRect – add rectangle with rounded corners to a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addRoundedRect**", x1, y1, x2, y2, r) | |
| Description: | Adds a rectangle with rounded off corners to the currently selected contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x1 | left-hand edge (double) |
| | y1 | upper edge (double) |
| | x2 | upper edge (double) |
| | y2 | upper edge (double) |
| | r | Radius (double) |

### addElipse – add ellipse to a contour

| | |
|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addEllipse**", x1, y1, x2, y2, radius) |
| Description: | Adds an ellipse to the currently selected contour. |

| | | |
|---|---|---|
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x1 | left-hand edge (double) |
| | y1 | upper edge (double) |
| | x2 | right-hand edge (double) |
| | y2 | lower edge (double) |
| | Radius | Radius (double) |

### addCircle – add circle to a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addCircle**", x, y, radius) | |
| Description: | Adds a circle to the currently selected contour | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x | left-hand edge (double) |
| | y | upper edge (double) |
| | Radius | Radius (double) |

### addArc – add arc to a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addArc**", x1, y1, x2, y2, StartAngle, SpanAngle | |
| Description: | Adds an arc to the currently selected contour | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x1 | left-hand edge (double) |
| | y1 | upper edge (double) |
| | x2 | right-hand edge (double) |
| | y2 | lower edge (double) |
| | StartAngle | Starting angle in degrees (double) |
| | SpanAngle | Opening angle in degrees (double) |

### addText – add text to a contour

| | |
|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**addText**", x, y, Text) |
| Description: | Adds a text to the currently selected contour |

| | | |
|---|---|---|
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x | left-hand edge (double) |
| | y | upper edge (double) |
| | Text | Text (QString) |

### setPenWidth – sets the pen width

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setPenWidth**") ReturnValue = CallCWMethod(GraphVarName, "**setPenWidth**", width) | |
| Description: | Defines the pen width from the instant that the function was called for the following objects that were added to the actual contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | width | Pen width (double).<br>If a pen width is not specified, then the default pen width is set. |

### setPenStyle – sets the pen style

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setPenStyle**")<br>ReturnValue = CallCWMethod(GraphVarName, "**setPenStyle**", style) | |
| Description: | Defines pen style from the instant that the function is called for the following objects that are added to the actual contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | style | 1 : Solid line (___)<br>2 : Interrupted line (_ _)<br>3 : Dotted line (...)<br>4 : Line-point-line (_._)<br>5 : Line-point-point (_..) |

### setPenColor – sets the pen color

| | |
|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setPenColor**")<br>ReturnValue = CallCWMethod(GraphVarName, "**setPenColor**", Color) |
| Description: | Defines the pen color from the instant that the function was called for the following objects that were added to the actual contour. |

| | | |
|---|---|---|
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | Color | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### setFillColor – set fill color

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setFillColor**")<br>ReturnValue = CallCWMethod(GraphVarName, "**setFillColor**", Color) | |
| Description: | Defines the fill color from the instant that the function was called for the following objects that were added to the actual contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | Color | Value to be set (QString) as RGB value in the form "#RRGGBB", for example, "#04B7FB" |

### setCursorPosition – position cursor

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setCursorPosition**", x, y) | |
| Description: | Sets the cursor to the specified position. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x | x coordinate (double) |
| | y | y coordinate (double) |

**Note**

This function automatically refreshes the display.

### setCursorPositionY2Cursor – position cursor referred to the second Y axis (right)

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setCursorPositionY2**", x, y2) | |
| Description: | Sets the cursor to the specified position referred to the second Y axis (right). | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | x | x coordinate (double) |
| | y2 | Y coordinate referred to the second Y axis (right) (double) |

**Note**

This function automatically refreshes the display.

### setCursorOnContour – position cursor on contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**setCursorOnContour**")<br>ReturnValue = CallCWMethod(GraphVarName, "**setCursorOnContour**", Contour-Name) | |
| Description: | Sets the cursor to the last cursor position within a contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

**Note**

This function automatically refreshes the display.

### Example

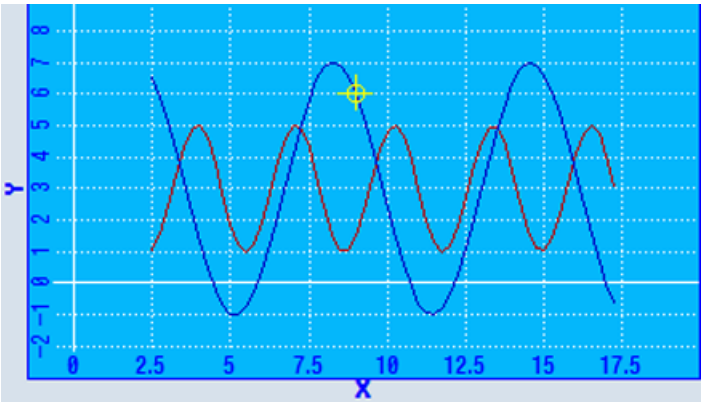



Figure 9-16 Example - setCursorOnContour

### moveCursorOnContourBegin – position the cursor to the first graphic object of a contour

| | |
|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**moveCursorOnContourBegin**")<br>ReturnValue = CallCWMethod(GraphVarName, "**moveCursorOnContourBegin**", ContourName) |
| Description: | Starting from the actual cursor position within a contour, using this function you can navigate the cursor to the first point object of a contour. |

| | | |
|---|---|---|
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

**Note**

This function automatically refreshes the display.

### moveCursorOnContourEnd – position the cursor to the last graphic object of a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**moveCursorOnContourEnd**")<br>ReturnValue = CallCWMethod(GraphVarName, "**moveCursorOnContourEnd**", ContourName) | |
| Description: | Starting from the actual cursor position within a contour, using this function you can navigate the cursor to the last point object of a contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

**Note**

This function automatically refreshes the display.

### moveCursorOnContourNext – position the cursor to the next graphic object of a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**moveCursorOnContourNext**")<br>ReturnValue = CallCWMethod(GraphVarName, "**moveCursorOnContourNext**", ContourName) | |
| Description: | Starting from the actual cursor position within a contour, using this function you can navigate the cursor to the next point object of a contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

**Note**

This function automatically refreshes the display.

### moveCursorOnContourBack – position the cursor to the previous graphic object of a contour

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**moveCursorOnContourBack**")<br>ReturnValue = CallCWMethod(GraphVarName, "**moveCursorOnContourBack**", ContourName) | |
| Description: | Starting from the actual cursor position within a contour, using this function you can navigate the cursor to the previous point object of a contour. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | ContourName | Name of the contour (QString).<br>If a contour is not specified, then the call automatically refers to the currently selected contour. |

**Note**

This function automatically refreshes the display.

### serialize – save/restore the actual state

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(GraphVarName, "**serialize**", FilePath, IsStoring) | |
| Description: | Using this function, when required, you can write the actual state and content of the SIEsGraphWidget in binary form to a file or datastream and also restore it. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SIEsGraph-CustomWidget |
| | Return Value | Error code (bool): TRUE = successful |
| | FilePath | Complete path with file name (QString) |
| | IsStoring | Save/restore (bool): TRUE = save |

**Note**

This function automatically refreshes the display.

## 9.5.6 Signals

You can capture the subsequently described ViewChanged signal in the configuration and respond appropriately.

### Overview

| Function | Description |
|---|---|
| ViewChanged | Changing the view |

### ViewChanged – changing the view

| | | |
|---|---|---|
| Syntax: | SUB(on_<GraphVarName>**_ViewChanged**)<br>...<br>END_SUB | |
| Description: | The "ViewChanged" signal is sent if the view changes. You can respond to this in the configuration in a specific SUB method. The SIGARG parameters are appropriately set. | |
| Parameters: | GraphVarName | Name of the display variable which contains a SlEsGraphCustomWidget |
| | SIGARG[0] | left-hand edge (double) |
| | SIGARG[1] | upper edge (double) |
| | SIGARG[2] | right-hand edge (double) |
| | SIGARG[3] | lower edge (double) |

### Example

```
DEF MyGraphVar = (W///,"slesgraphcustomwidget.SlEsGraphCustomWidget"/////
10,10,340,340/0,0,0,0)
```

```
SUB(on_MyGraphVar_ViewChanged
     DLGL("Current view: " << SIGARG[0] << ", " << SIGARG[1] << ", " <<   SIGARG[2]
<< ", " << SIGARG[3]
END_SUB
```

## 9.6 SlEsTouchButton

### 9.6.1 SlEsTouchButton

#### General information

You can create even functionally demanding applications simply by using Run MyScreens. For touch operation you can configure buttons that can be freely located (TouchButtons). The following attributes apply when configuring TouchButtons:

- The two possible state changes of the "clicked" and "checked" buttons can be queried in the configuration and the appropriate actions initiated.
- TouchButtons can be operated with single or multitouch, mouse and keyboard.

- The TouchButton is displayed corresponding to the actual resolution. This also applies to the font and displayed graphics.
- The TouchButton has two styles "Softkey look&feel" and "Application-specific". In the "Softkey look&feel" style, the TouchButton is displayed corresponding to the Operate softkeys. Both display styles have functions, for example, to scale graphics.
- TouchButtons are implemented and made available as custom widgets.

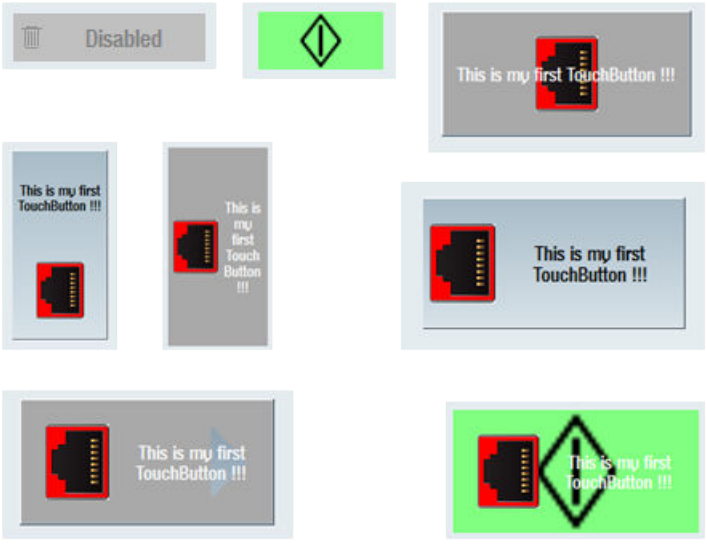



Figure 9-17 Examples of TouchButtons

## Example

```
//M(MyTBMask/"My CustomWidget TouchButton ...")
DEF MyTB1 = (W///,"slesstdcw.SlEsTouchButton"/////70,20,200,100/0,0,0,0)
LOAD
   WRITECWPROPERTY("MyTB1", "text", "This is my first TouchButton !!!")
   WRITECWPROPERTY("MyTB1", "textPressed", "This is my first TouchButton (pressed)!!!")
   WRITECWPROPERTY("MyTB1", "picture", "dsm_remove_trashcan_red.png") WRITECWPROPERTY("MyTB1", "pic-
tureAlignment", "left")
   WRITECWPROPERTY("MyTB1", "scalePicture", FALSE)
   WRITECWPROPERTY("MyTB1", "picturePressed", "slsu_topology_empty_round_slot.png") WRITECWPROPER-
TY("MyTB1", "picture", "slsu_topology_empty_slot_left_error.png")
END_LOAD
```

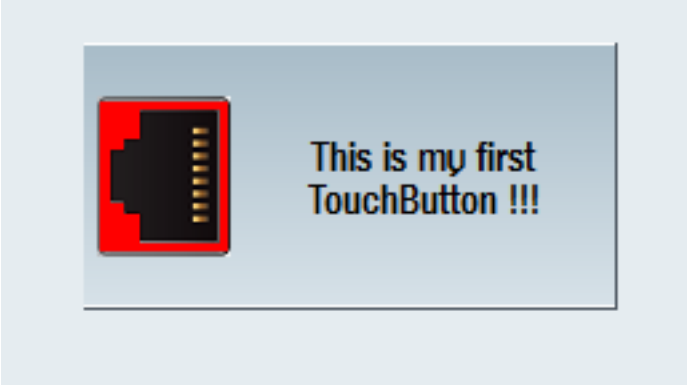



Figure 9-18 Example, "This is my first TouchButton !!!"

## 9.6.2 Reading and writing properties

### Description

The properties listed in the following section are read with ReadCWProperty() and written with WriteCWProperty().

### Examples

Reading the "Text" property of the SlEsTouchButton linked using display variable "MyTouchButton". The result is written to register 0.

```
REG[0] = ReadCWProperty("MyTouchButton", "Text")
```

Writing value "sk_ok.png" in the "Picture" property of the SlEsTouchButton, linked using display variable "MyTouchButton".

```
WriteCWProperty("MyTouchButton", "Picture", "sk_ok.png")
```

## 9.6.3 Properties

### Overview

| Property | Description |
|---|---|
| ButtonStyle | TouchButton display style |
| Flat | Displayed flat or in 3D |
| Enabled | Activate/deactivate operability |
| Checkable | Activate/deactivate toggle function |
| Checked | TouchButton is presently checked |
| ShowFocusRect | Display focus square if the TouchButton has the input focus |

| Property | Description |
|---|---|
| Picture | Picture (image) that should be displayed if the TouchButton is in the normal, non-actuated state |
| PicturePressed | Picture (image) to be displayed if the TouchButton is pressed or checked |
| PictureAlignment<br>PictureAlignmentString | Alignment of the picture |
| ScalePicture | Picture is scaled (stretched or compressed) |
| PictureKeepAspectRatio | Picture aspect ratio |
| Text | Normal display text |
| TextPressed | Text that is displayed if the TouchButton is pressed |
| textAlignment<br>textAlignmentString | Alignment of the text |
| TextAlignedToPicture | Text is aligned relative to the picture - activate/deactivate |
| BackgroundPicture | Background screen to be displayed |
| BackgroundPictureAlignment<br>BackgroundPictureAlignmentString | Alignment of the background picture |
| ScaleBackgroundPicture | Background picture is scaled (stretched or compressed) |
| BackgroundPictureKeepAspectRatio | Picture aspect ratio |
| BackColor | Background color if the TouchButton is in the normal and non-actuated state |
| BackColorChecked | Background color if the TouchButton is checked |
| BackColorPressed | Background color if the TouchButton is presently pressed |
| BackColorDisabled | Background color if the TouchButton cannot be operated |
| TextColor | Text color if the TouchButton is in the normal and non-actuated state |
| TextColorChecked | Text color if the TouchButton is checked |
| TextColorPressed | Text color if the TouchButton is presently pressed |
| TextColorDisabled | Text color if the TouchButton cannot be operated |

## Button style

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**ButtonStyle**")<br>WriteCWProperty(TouchButtonVarName, "**ButtonStyle**", Value) | |
| Description: | Reads/sets the TouchButton style | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (int) |
| | Value | value to be set (int)<br>0 = Softkey Look&Feel (default)<br>1 = user-specific |

With the setting "0 = Softkey Look&Feel", the TouchButton is displayed and responds just like a softkey. The currently set skin is taken into account – see display machine data 9112 = $MM_HMI_SKIN.

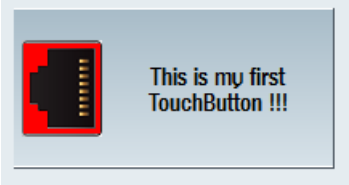



Figure 9-19 ButtonStyle - Softkey Look&Feel

With setting "1 = user-specific", the text and background color can be defined according to the TouchButton state. Further, there is a 3D effect, which allows pictures to be automatically scaled and influences the distances from the edges.

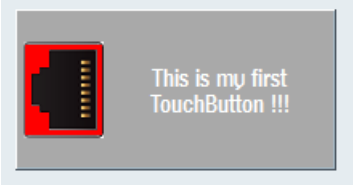



Figure 9-20 ButtonStyle - user-specific

## Flat – display type

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**Flat**")<br>WriteCWProperty(TouchButtonVarName, "**Flat**", Value) | |
| Description: | Reads/sets as to whether the TouchButton is shown flat (default) or in 3D<br>**Note:** This property is only available with active style (ButtonStyle) "1= user-specific". | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEs-TouchButton |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE (default) or FALSE |

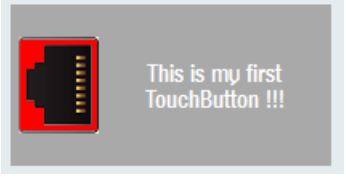



Figure 9-21 Flat display style

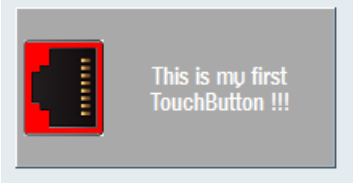



Figure 9-22 3D display style

## Enabled – operability of the TouchButton

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**Enabled**")<br>WriteCWProperty(TouchButtonVarName, "**Enabled**", Value) | |
| Description: | Read/sets as to whether the TouchButton can be operated (= TRUE) - or cannot be operated (FALSE). | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE (default) or FALSE |

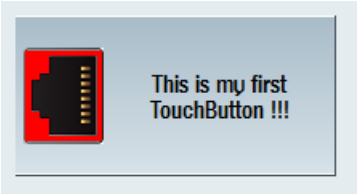



Figure 9-23 TouchButton can be operated

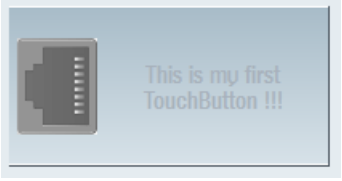



Figure 9-24 TouchButton cannot be operated

**Note**

If a TouchButton that cannot be operated is clicked, then signal "clickedDisabled" is sent. The signal can be evaluated, for example an appropriate message issued as to why the TouchButton and the associated function presently cannot be operated. In the deactivated state, the picture or background picture is automatically displayed in shades of gray.

## Checkable – activate/deactivate toggle function

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**Checkable**")<br>WriteCWProperty(TouchButtonVarName, "**Checkable**", Value) | |
| Description: | Reads/sets the TouchButton toggle function. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE or FALSE (default) |

**Note**

In the default setting, after it has been released, the TouchButton returns to its normal state (the same as a pushbutton) However, if the TouchButton toggle functionality is activated (TRUE), then after it is press, it initially remains in the pressed position. If it is actuated again, then it changes back into its normal state (same as a switch).

See also the "Checked" property.

## Checked – actual toggle state

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**Checked**")<br>WriteCWProperty(TouchButtonVarName, "**Checked**", Value) | |
| Description: | Reads/sets the actual toggle state of the TouchButton | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE or FALSE (default) |

If the toggle function is activated as the property "Checkable" is "TRUE", then the actual toggle state can be read or set using the "Checked" property.

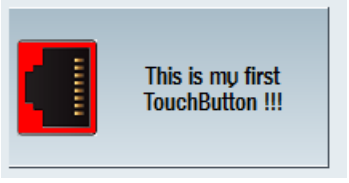



Figure 9-25 Unchecked

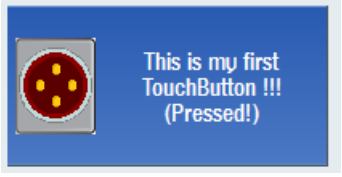



Figure 9-26 Checked

For both states, text and picture can be specifically displayed.

**Note**

See also the "Checkable" property.

### ShowFocusRect – focus rectangle display

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**ShowFocusRect**")<br>WriteCWProperty(TouchButtonVarName, "**ShowFocusRect**", Value) | |
| Description: | Reads/sets as to whether the TouchButton should display a focus rectangle as soon as it is assigned the input focus. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE or FALSE (default) |

The focus frame color is automatically set with the Operate color setting, in the following example "orange".

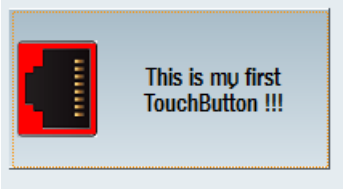



Figure 9-27 ShowFocusRect

### Picture – displayed picture

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**Picture**")<br>WriteCWProperty(TouchButtonVarName, "**Picture**", Value) | |
| Description: | Reads/sets the picture to be displayed if the TouchButton is in its quiescent state (not actuated). | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | read value of the property (string) |
| | Value | value to be set (string) |

The file name is specified as value, for example "sk_ok.png". The TouchButton automatically determines the correct picture file from the current resolution directory (see Chapter Using display images/graphics (Page 63)).

In the deactivated state, the picture is automatically displayed in shades of gray.

**Note**

See also properties - "Text", "TextAlignedToPicture", "PictureAlignment", "ScalePicture", "PictureKeepAspectRatio".

### PicturePressed – picture in the pressed state

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**PicturePressed**")<br>WriteCWProperty(TouchButtonVarName, "**PicturePressed**", Value) | |
| Description: | Reads/sets the picture to be displayed if the TouchButton is pressed or checked. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | read value of the property (string) |
| | Value | value to be set (string) |

If a value is not specified (empty string " "), then in this state, the TouchButton shows the picture specified with the "Picture" property.

The file name is specified as value, for example "sk_ok.png". The TouchButton automatically determines the correct picture file from the current resolution directory (see Chapter Using display images/graphics (Page 63)).

In the deactivated state, the picture is automatically displayed in shades of gray.

**Note**

See also properties - "Text", "TextAlignedToPicture", "PictureAlignment", "ScalePicture", "PictureKeepAspectRatio".

### PictureAlignment – picture alignment

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**PictureAlignment**")<br>WriteCWProperty(TouchButtonVarName, "**PictureAlignment**", Value) | |
| Description: | Reads/sets the alignment of the picture on the TouchButton. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (int) |
| | Value | Value to be set (int):<br>1 = left (default)<br>2 = right<br>32 = top<br>64 = bottom<br>128 = centered |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).
Also see properties - "Text", "TextAlignedToPicture", "PictureAlignmentString", "ScalePicture", "PictureKeepAspectRatio".

### PictureAlignmentString – picture alignment

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**PictureAlignmentString**")<br>WriteCWProperty(TouchButtonVarName, "**PictureAlignmentString**", Value) | |
| Description: | Reads/sets the alignment of the picture. This property is analogous to the "PictureAlignment" property. However, the value is not transfigured here as number (int), but as string. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | read value of the property (string) |
| | Value | value to be set (string):<br>"Left" = left (default)<br>"Right" = right<br>"Top" = top<br>"Bottom" = bottom<br>"Center" = centered |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

See also properties - "Text", "TextAlignedToPicture", "PictureAlignment", "ScalePicture", "PictureKeepAspectRatio".

### ScalePicture – scale picture

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**ScalePicture**")<br>WriteCWProperty(TouchButtonVarName, "**ScalePicture**", Value) | |
| Description: | Reads/sets whether the TouchButton (depending on the alignment) should scale the picture to be displayed. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE or FALSE (default) |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

### PictureKeepAspectRatio – scale picture - keeping the aspect ratio

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**PictureKeepAspectRatio**")<br>WriteCWProperty(TouchButtonVarName, "**PictureKeepAspectRatio**", Value) | |
| Description: | Reads/sets, whether, when scaling (Property "ScalePicture" = "TRUE"), the aspect ratio of the picture should be kept or not. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE (default) or FALSE |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

### Text – display text

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**Text**")<br>WriteCWProperty(TouchButtonVarName, "**Text**", Value) | |
| Description: | Reads/sets the displayed text if the TouchButton is in its quiescent state (not actuated). | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | read value of the property (string) |
| | Value | value to be set (string) |

The texts are automatically wrapped around at the word limits in the rectangle provided.

Line breaks are only forced as a result of the system if the text to be displayed comes from a language file.

**Note**

See Chapter Language-dependent texts (Page 268).

### TextPressed – displayed text in the pressed state

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**TextPressed**")<br>WriteCWProperty(TouchButtonVarName, "**TextPressed**", Value) | |
| Description: | Reads/sets the displayed text if the TouchButton is pressed or checked. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | read value of the property (string) |
| | Value | value to be set (string) |

The texts are automatically wrapped around at the word limits in the rectangle provided.

Line breaks are only forced as a result of the system if the text to be displayed comes from a language file.

**Note**

See Chapter Language-dependent texts (Page 268).

### TextAlignment – alignment of the text

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**TextAlignment**")<br>WriteCWProperty(TouchButtonVarName, "**TextAlignment**", Value) | |
| Description: | Reads/sets the alignment of the text on the TouchButton | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (int) |
| | Value | Value to be set (int):<br>1 = left<br>2 = right<br>32 = top<br>64 = bottom<br>128 = centered (default)<br>(see examples below) |

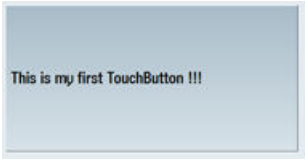



Figure 9-28 TextAlignment - left


Figure 9-29 TextAlignment - right

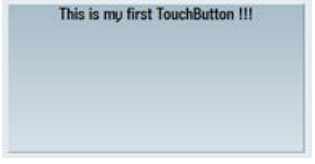



Figure 9-30 TextAlignment - top

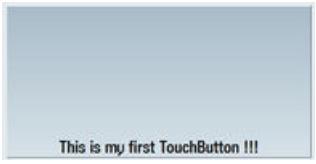



Figure 9-31 TextAlignment - bottom

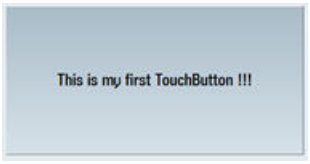



Figure 9-32 TextAlignment - centered

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

See also properties "Text", "TextAlignmentString", "TextAlignedToPicture".

## TextAlignmentString – alignment of the text

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**TextAlignment-String**")<br>WriteCWProperty(TouchButtonVarName, "**TextAlignmentString**", Value) | |
| Description: | Reads/sets the alignment of the text. This property is analogous to the "TextAlignment" property. The value is not transfigured here as number (int), but as string. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | read value of the property (string) |
| | Value | value to be set (string):<br>"Left" = left<br>"Right" = right<br>"Top" = top<br>"Bottom" = bottom<br>"Center" = centered (default) |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

See also properties - "Text", "TextAlignment", "TextAlignedToPicture".

### TextAlignedToPicture – text aligned relative to the picture

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "TextAlignedToPicture")<br>WriteCWProperty(TouchButtonVarName, "**TextAlignedToPicture**", Value) | |
| Description: | Reads/sets whether the displayed text should be positioned relative to the picture. If FALSE is set here, then the text is shown centered on the TouchButton. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE (default) or FALSE |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).
See also properties "Text", "TextPressed", "Picture", "PicturePressed".

### BackColor – background color

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, " **BackColor**")<br>WriteCWProperty(TouchButtonVarName, " **BackColor**", Value) | |
| Description: | Reads/sets the background color if the TouchButton is in its quiescent state (not actuated). | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | Read color of the property (string) |
| | Value | Value to be set (string) as RGB value in the form "#RRGGBB",<br>for example, "#04B7FB" |

**Note**

This property is only available with active style "1= user-specific".

### BackColorChecked – background color in the checked state

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**BackColorChecked**")<br>WriteCWProperty(TouchButtonVarName, " **BackColorChecked**", Value) | |
| Description: | Reads/sets the background color if the TouchButton is in the clicked state (toggle function). See properties "Checked", "Checkable" | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | Read color of the property (string) |
| | Value | Value to be set (string) as RGB value in the form "#RRGGBB",<br>for example, "#04B7FB" |

**Note**

This property is only available with active style "1= user-specific".

**Note**

See also properties "Checked", "Checkable".

## BackColorPressed – background color in the pressed state

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**BackColorPressed**")<br>WriteCWProperty(TouchButtonVarName, " **BackColorPressed**", Value) | |
| Description: | Reads/sets the background color if the TouchButton is in the pressed state | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read color of the property (string) |
| | Value | Value to be set (string) as RGB value in the form "#RRGGBB"<br>e.g. "#04B7FB" |

**Note**

This property is only available with active style "1= user-specific".

## BackColorDisabled – background color in the deactivated state

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**BackColorDisabled**")<br>WriteCWProperty(TouchButtonVarName, " **BackColorDisabled**", Value) | |
| Description: | Reads/sets the background color if the TouchButton is in the deactivated state. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read color of the property (string) |
| | Value | Value to be set (string) as RGB value in the form "#RRGGBB",<br>for example, "#04B7FB" |

**Note**

This property is only available with active style "1= user-specific".

**Note**

See also the " Enabled" property.

### TextColor – text color

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**TextColor**")<br>WriteCWProperty(TouchButtonVarName, " **TextColor**", Value) | |
| Description: | Reads/sets the text color if the TouchButton is in its quiescent state (not actuated). | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read color of the property (string) |
| | Value | Value to be set (string) as RGB value in the form "#RRGGBB",<br>for example, "#04B7FB" |

**Note**

This property is only available with active style "1= user-specific".

### TextColorChecked – text color in the checked state

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**TextColorChecked**")<br>WriteCWProperty(TouchButtonVarName, " **TextColorChecked**", Value) | |
| Description: | Reads/sets the text color if the TouchButton is in the checked state (toggle function). | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read color of the property (string) |
| | Value | Value to be set (string) as RGB value in the form "#RRGGBB",<br>for example, "#04B7FB" |

**Note**

This property is only available with active style "1= user-specific".

**Note**

See also properties "Checked", "Checkable".

### TextColorPressed – text color in the pressed state

| | |
|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**TextColorPressed**")<br>WriteCWProperty(TouchButtonVarName, "**TextColorPressed**", Value) |
| Description: | Reads/sets the text color if the TouchButton is in the pressed state |

| | | |
|---|---|---|
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read color of the property (string) |
| | Value | Value to be set (string) as RGB value in the form "#RRGGBB",<br>for example, "#04B7FB" |

**Note**

This property is only available with active style "1= user-specific".

### TextColorDisabled – text color in the deactivated state

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**TextColorDisabled**")<br>WriteCWProperty(TouchButtonVarName, "**TextColorDisabled**", Value) | |
| Description: | Reads/sets the text color if the TouchButton is in the deactivated state | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read color of the property (string) |
| | Value | Value to be set (string) as RGB value in the form "#RRGGBB",<br>for example, "#04B7FB" |

**Note**

This property is only available with active style "1= user-specific".

**Note**

See also the " Enabled" property.

### BackgroundPicture – background picture

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**BackgroundPicture**")<br>WriteCWProperty(TouchButtonVarName, "**BackgroundPicture**", Value) | |
| Description: | Reads/sets the permanent background picture to be displayed, i.e. independent of the state. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | read value of the property (string) |
| | Value | value to be set (string) |

The file name is specified as value, for example "sk_ok.png". The TouchButton automatically determines the correct picture file from the current resolution directory (see Chapter Using display images/graphics (Page 63)).

In the deactivated state, the background picture is automatically shown in shades of gray.

**Note**

See Chapter Using display images/graphics (Page 63).

See also properties - "BackgroundPictureAlignment", "ScaleBackgroundPicture", "BackgroundPictureKeepAspectRatio".

### BackgroundPictureAlignment – alignment of the background picture

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**BackgroundPictureAlignment**")<br>WriteCWProperty(TouchButtonVarName, "**BackgroundPictureAlignment**", Value) | |
| Description: | Reads/sets the alignment of the background picture. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouchButton |
| | Return Value | Read value of the property (int) |
| | Value | Value to be set (int):<br>1 = left<br>2 = right<br>32 = top<br>64 = bottom<br>128 = centered (default) |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

See also properties - "BackgroundPictureAlignment", "ScaleBackgroundPicture", "BackgroundPictureKeepAspectRatio".

### BackgroundPictureAlignmentString – alignment of the background picture

| | |
|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**BackgroundPictureAlignmentString**")<br>WriteCWProperty(TouchButtonVarName, "**BackgroundPictureAlignmentString**", Value) |
| Description: | Reads/sets the alignment of the background picture. This property is analogous to the property "BackgroundPictureAlignment". However, the value is not transfigured here as number (int), but as string. |

| | | |
|---|---|---|
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (int) |
| | Value | Value to be set (int):<br>"Left" = left<br>"Right" = right<br>"Top" = top<br>"Bottom" = bottom<br>"Center" = centered (default) |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

See also properties - "BackgroundPictureAlignment", "ScaleBackgroundPicture", "BackgroundPictureKeepAspectRatio".

### ScaleBackgroundPicture – scale background picture

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**ScalePicture**")<br>WriteCWProperty(TouchButtonVarName, "**ScalePicture**", Value) | |
| Description: | Reads/sets whether the TouchButton (depending on the alignment) should scale the picture to be displayed. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE or FALSE (default) |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

See also properties - "BackgroundPictureAlignment", "ScaleBackgroundPicture", "BackgroundPictureKeepAspectRatio".

### BackgroundPictureKeepAspectRatio – scale background picture and keep aspect ratio

| | | |
|---|---|---|
| Syntax: | ReturnValue = ReadCWProperty(TouchButtonVarName, "**BackgroundPicture-KeepAspectRatio**")<br>WriteCWProperty(TouchButtonVarName, "**BackgroundPictureKeepAspectRatio**", Value) | |
| Description: | Reads/sets, whether, when scaling (property "ScaleBackgroundPicture" = "TRUE") the aspect ratio of the background picture should be kept or not. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Read value of the property (bool) |
| | Value | Value to be set (bool): TRUE (default) or FALSE |

**Note**

See Chapter Positioning and aligning picture and text (Page 266).

## 9.6.4 Functions

You can call the subsequently listed functions using the CallCWMethod() function.

### Example

Setting the margins of the SlEsTouchButton linked using display variable "MyTouchButton".

The result is written to register 0.

```
REG[0] = CallCWMethod("MyTouchButton", "setMargins", 20, 20, 20, 20,
20)
```

### Overview

| Function | Description |
|---|---|
| setMargins | Setting the margins |
| serialize | Saves/restores the actual state |

### setMargins – setting the margins

| | |
|---|---|
| Syntax: | ReturnValue = CallCWMethod(TouchButtonVarName, "**setMargins**", left, top, right, bottom, center)<br>ReturnValue = CallCWMethod(TouchButtonVarName, "**setMargins**", left, top, right, bottom, center, marginAlignment) |
| Description: | Using this function, the margin clearances and the distance between picture and text are defined.<br>If a value of "-1" is specified, then this value is applicable for the system default clearance. If the value is greater or equal to "0", then the value is set as edge clearance. |

| | | |
|---|---|---|
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Error code (bool): TRUE = successful |
| | left | lefthand edge (int) |
| | top | top edge (int) |
| | right | right-hand edge (int) |
| | bottom | bottom edge (int) |
| | center | Clearance between picture and text (int), if the alignment is not "center" |
| | marginAlignment | Optional: Defines whether the margin next to the displayed picture should also apply when positioning the display text (bool), TRUE (default) |

### serialize – save/restore the actual state

| | | |
|---|---|---|
| Syntax: | ReturnValue = CallCWMethod(TouchButtonVarName, "**serialize**", FilePath, IsStoring) | |
| Description: | With this function, when required, you can write and restore the actual state and content of the SlEsTouchButton binary into a file or Data Stream. This function automatically refreshes the display. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | Return Value | Error code (bool): TRUE = successful |
| | FilePath | Complete path with file name (QString) |
| | IsStoring | Save/restore (bool): TRUE = save |

**Note**

Programming engineers do not directly use this function!

## 9.6.5 Signals

You can capture the subsequently described signals in the configuration and respond appropriately.

### Overview

| Function | Description |
|---|---|
| clickedDisabled | Press a deactivated TouchButton |
| clicked | A click or tap is executed |
| checked | It was toggled |

- If a TouchButton can be operated - but it cannot be toggled - then "clicked" signal is sent when the TouchButton is pressed.
- If a TouchButton can be operated and toggled - then the following sequence of signals is sent when the TouchButton is pressed.
  "checked" → "clicked"
- If a TouchButton cannot be operated - then the "clickedDisabled" signal is sent when the TouchButton is pressed.

All of the signals mentioned above are only sent after an operator action, i.e. only once the mouse or the space bar has been released or after a tap for multitouch.

As a consequence, the SlEsTouchButton has a pure switch function, i.e. a pushbutton function is not possible.

**Note**

For multitouch operation, note that a click can only be sent if a tap gesture has been completely identified (pressing and releasing within approximately 0.7 seconds). If an action was already carried out when simply pressing, it would not be possible for example to scroll/shift the ScrollArea behind, or it could lead to operator input errors.

### clicked – the TouchButton was clicked

| | | |
|---|---|---|
| Syntax: | SUB(on_<TouchButtonVarName>_clicked)<br>...<br>END_SUB | |
| Description: | A complete sequence comprising pressing and releasing a TouchButton (that can be operated) results in a "clicked" signal. We recommend working predominantly with this signal. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | SIGARG[0] | Supplies the toggle state of the TouchButton (bool) |

### Example

```
DEF MyTouchButton = (W///,"slesstdcw.SlEsTouchButton"/////70,20,200,100/0,0,0,0)
SUB(on_MyTouchButton_clicked)
   DLGL("checked: " << SIGARG[0])
END_SUB
```

### checked – a TouchButton was toggled

| | | |
|---|---|---|
| Syntax: | SUB(on_<TouchButtonVarName>_checked)<br>...<br>END_SUB | |
| Description: | A TouchButton that can be toggled was pressed, which changed its state. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | SIGARG[0] | Supplies the toggle state of the TouchButton (bool) |

### Example

```
DEF MyTouchButton = (W///,"slesstdcw.SlEsTouchButton"/////70,20,200,100/0,0,0,0)
SUB(on_MyTouchButton_toggled)
   DLGL("toggled: " << SIGARG[0])
END_SUB
```

### clickedDisabled – a TouchButton that cannot be operated was clicked

| | | |
|---|---|---|
| Syntax: | SUB(on_<TouchButtonVarName>_clickedDisabled)<br>...<br>END_SUB | |
| Description: | A complete sequence comprising pressing and releasing a TouchButton (that cannot be operated) results in a "clickedDisabled" signal. | |
| Parameters: | TouchButtonVarName | Name of the display variable that contains a SlEsTouch-Button |
| | SIGARG[0] | Supplies the toggle state of the TouchButton (bool) |

### Example

```
DEF MyTouchButton = (W///,"slesstdcw.SlEsTouchButton"/////70,20,200,100/0,0,0,0)
SUB(on_MyTouchButton_clickedDisabled)
   DLGL("checkedDisabled: " << SIGARG[0])
END_SUB
```

## 9.6.6 Positioning and aligning picture and text

### Positioning pictures

A picture is positioned as follows with the specified alignment:

- In a first step, the picture matching the current resolution is determined from the associated resolution directory.
- The rectangle of the area (ClientArea) is then reduced by the specified clearances to the edges (MarginArea).
  The MarginArea can be influenced using the "setMargins" function:
  - setMargins(-1, -1, -1, -1, -1)

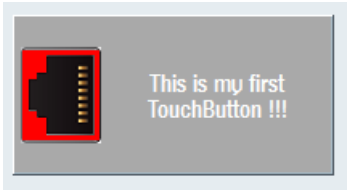



Figure 9-33 setMargins(-1, -1, -1, -1, -1)

  - setMargins(0, 0, 0, 0, 0)

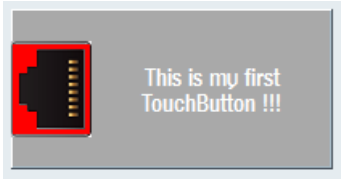



Figure 9-34 setMargins(0, 0, 0, 0, 0)

  - setMargins(20, 20, 20, 20, 20)

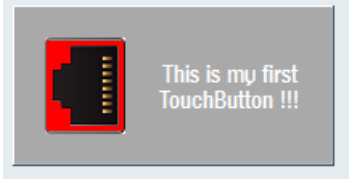



Figure 9-35 setMargins(20, 20, 20, 20, 20)

#### Example for alignment "left"

The area is horizontally halved, so that the picture to be displayed can assume half (maximum) of the area.

The picture is then drawn as follows:

- Property "scalePicture": FALSE
  The picture is drawn at this location without any scaling. It must be ensured that the basic picture is not too large, and can be completely displayed within the TouchButton.

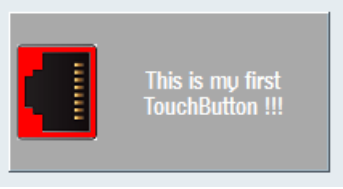



Figure 9-36 scalePicture FALSE

- Property "scalePicture": TRUE
  The picture is scaled (stretched or compressed), so that it fits into the left-hand half of the TouchButton. The "pictureKeepAspectRatio" property is taken into account as follows:
  - Property "pictureKeepAspectRatio": FALSE
    The picture is scaled horizontally and vertically so that it precisely fits into the left-hand half of the TouchButton. In so doing, the aspect ratio of the original picture is no longer taken into consideration.

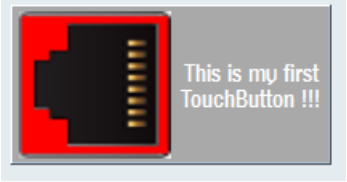



Figure 9-37 scalePicture - pictureKeepAspectRatio FALSE

  - Property "pictureKeepAspectRatio": TRUE
    Taking into consideration the aspect ratio of the original picture, the picture is scaled large enough so that it just fits into the left-hand half of the TouchButton.

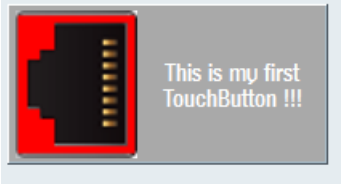



Figure 9-38 scalePicture - pictureKeepAspectRatio TRUE

**Note**

The same principle applies for "right", "top" and "bottom" alignments.

For the "centered" alignment, an attempt is made to fit the picture - depending on the "scalePicture" and "pictureKeepAspectRatio" properties - in the complete MarginArea.

### Positioning the text

The text is positioned as follows depending on the "textAlignedToPicture" property:

- Property "textAlignedToPicture": FALSE
  The text is shown in the MarginArea centered vertically and horizontally.

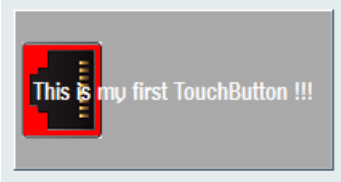



Figure 9-39 textAlignedToPicture FALSE

- Property "textAlignedToPicture": TRUE
  The text is output in the remaining area of the MarginArea minus the area of the drawn picture, centered horizontally and vertically.

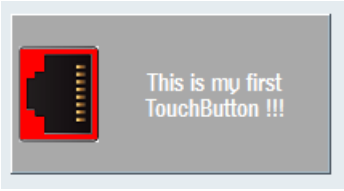



Figure 9-40 textAlignedToPicture TRUE

## 9.6.7 Language-dependent texts

SlEsTouchButton does not support foreign languages by itself. However, the following example shows how you can implement language dependency:

### Example

**easyscreen.ini:**

```
[LANGUAGEFILES]LngFile03 = user.txt
```

**user_eng.txt:**

```
85000 0 0 "This is my first TouchButton !!!"
```

**user_deu.txt:**

```
85000 0 0 "Das ist mein erster TouchButton !!!"
```

**Configuration file:**

```
//M(MyTBMask/"My CustomWidget TouchButton ...")
DEF MyTB1 = (W///,"slesstdcw.SlEsTouchButton"/////70,20,200,100/0,0,0,0)
LOAD
```

```
//M(MyTBMask/"My CustomWidget TouchButton ...")
DEF MyTB1 = (W///,"slesstdcw.SlEsTouchButton"/////70,20,200,100/0,0,0,0)
   WRITECWPROPERTY("MyTB1", "text", $85000)
END_LOAD
LANGUAGE
   WRITECWPROPERTY("MyTB1", "text", $85000)
END_LANGUAGE
```

The texts are automatically wrapped around at the word limits in the rectangle provided. The forced line break is only possible if the text to be displayed is sourced from a language file A "%n" must be inserted at the appropriate location in the text (see the following example).

**user_eng.txt:**

```
85001 0 0 "This is%nmy%nfirst%nTouchButton !!!"
```

**Configuration file:**

```
WRITECWPROPERTY("MyTB1", "text", $85001)
```

## Result


Figure 9-41 Example of a line break in a language-dependent text

# "Custom" operating area

# 10

## 10.1 How to activate the "Custom" operating area

### Activating the "Custom" operating area

The "Custom" operating area is not activated on delivery.

1. First copy the "slamconfig.ini". file from the directory [System siemens-directory]/cfg into the directory [System oem-directory]/cfg
   or correspondingly into [System addon-directory]/cfg or [System user-directory]/cfg.
2. To activate the "Custom" operating area, the following must be entered:
   ```
   [Custom]
   Visible=True
   ```

### Result

After activation is complete, the softkey for the "Custom" operating area can be found in the main menu (F10) on the menu continuation bar on the HSK4 (= default).

The "Custom" operating area displays an empty window covering the entire operating area, with a configurable header. All horizontal and vertical softkeys can be configured.

## 10.2 How to configure the "Custom" softkey

### Configuring the softkey for the "Custom" operating area

The labeling and position of the softkey for the "Custom" operating area are configured in the "slamconfig.ini" file.

The following options are available for configuring the start softkey:

1. To replace a softkey label with a **language-dependent text**, the following must be entered in the [Custom] section:
   ```
   TextId=MY_TEXT_ID
   TextFile=mytextfile
   TextContext=mycontext
   ```
   In this example, the softkey shows the language-dependent text which was saved with the text ID "MY_TEXT_ID" in text file mytextfile_xxx.qm under "MyContext" (xxx stands for language code).
2. To replace a softkey label with a **language-neutral text**, the following must be entered in the [Custom] section:
   ```
   TextId=HELLO
   TextFile=<empty>
   TextContext=<empty>
   ```
   In this example, the softkey for the "Custom" operating area displays the text "HELLO" for every language.
3. **An icon** can also be displayed on the softkey in addition to the text.
   To do this, the following must be entered in the [Custom] section:
   ```
   Picture=mypicture.png
   ```
   The softkey then displays the icon from the file mypicture.png. Graphics and bitmaps are stored at the following path: [System oem directory]/ico/ico<resolution>. The directory that corresponds to the display resolution must be used.
4. **The position** of the softkey can also be set. The following entry in the [Custom] section makes this setting:
   ```
   SoftkeyPosition=12
   ```
   The default is position 12. This corresponds to the HSK4 on the menu continuation bar of the operating area's menu. Positions 1 - 8 correspond to HSK1 to HSK8 on the menu bar, positions 9 - 16 to HSK1 to HSK8 on the menu continuation bar.

## 10.3 How to configure the "Custom" operating area

### Configuring the softkey for the "Custom" operating area

You need the "easyscreen.ini" and "custom.ini" files to configure the operating area. Templates for both these files are available in the directory [System siemens-directory]/templates/cfg.

1. First copy the files to the directory [System oem directory]/cfg and make your changes there.
2. The "easyscreen.ini" file already contains a definition line for the "Custom" operating area:
   ```
   ;StartFile02 = area := Custom, dialog := SlEsCustomDialog,
   startfile := custom.com
   ```
   The ";" at the start of the line represents the comment character. This means the line is commented out and, as such, not active. To change this, the ";" must be deleted.
   The "startfile" attribute in this line defines that the entry will refer to the "custom.com" project file when the "Custom" operating area is selected.

3. You create the **project file "custom.com"** in the directory [System oem-directory]/proj. This contains the relevant configuration which is created in the same way as the "aeditor.com" file of the "Program" operating area. The configured start softkeys are then displayed in the "Custom" operating area.
4. You configure the **language-neutral text** for the title bar of the dialog in the "custom.ini" file. The following entry is available in the template for this purpose:
   ```
   [Header]
   Text=Custom
   ```
   You can replace this text with a customized one.
5. The template contains the following entry for configuring a **start screen** for the "Custom" operating area:
   ```
   [Picture]
   Picture=logo.png
   ```
   Logo.png is the name of the start screen which appears on the "Custom" operating area's start dialog. Here you can display a company logo, for example, or another image. The file should be saved in the directory for the corresponding resolution under: [System oem-directory]/ico/ ...
6. In order to display a specific "Run MyScreens" screen when the "Custom" operating area is shown the first time, the following entry is available in the template:
   ```
   ; Mask shown with startup of area "custom"
   [Startup]
   ;Startup = Mask:=MyCustomStartupMask, File:=mycustommasks.com
   ```
   Typically, SINUMERIK Operate is started directly with a specific "Run MyScreens" screen. The "startuparea" key must be set as follows in the "[miscellaneous]" section of the "systemconfiguration.ini" file:
   ```
   [miscellaneous]
   ;name of the area to be shown at system startup
   startuparea = Custom
   ```

## 10.4 Programming example for the "Custom" area

### Example

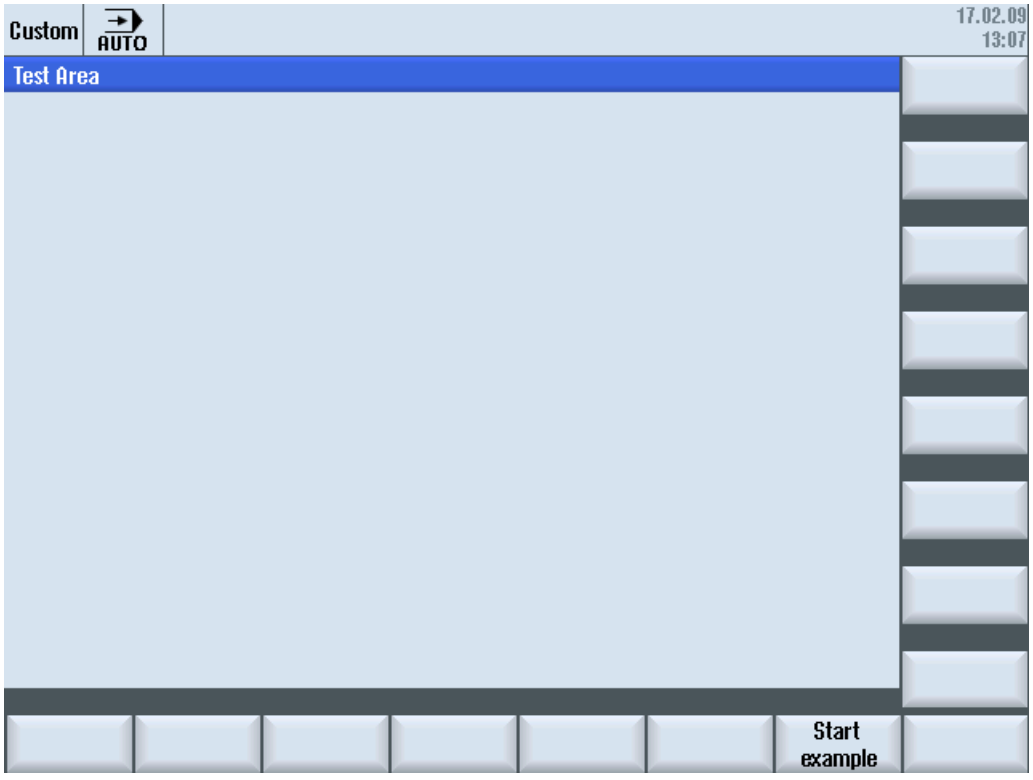



Figure 10-1 Example with "Start example" softkey

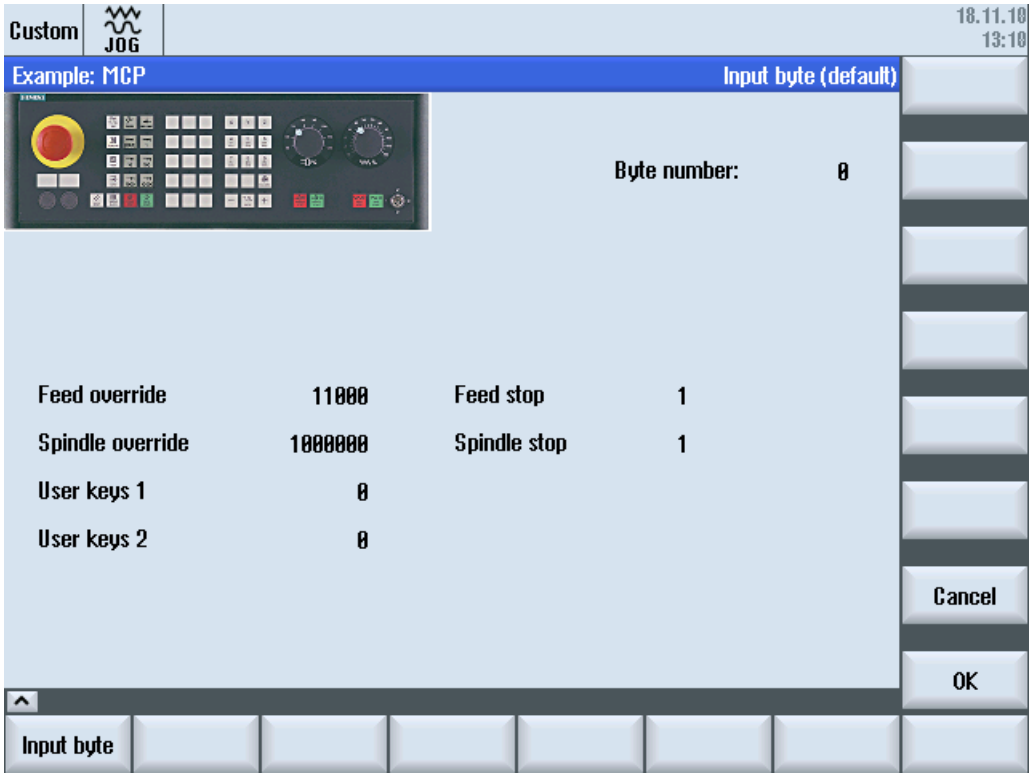



Figure 10-2 Example with bitmap and text fields

## File overview

The following files are required:

- "custom.com"
- "easyscreen.ini"

## Programming

Content of file "custom.com":

**Note**

The graphic file "mcp.png" integrated in the example is also only a sample file. If you wish to use this programming example in your application, you must replace the graphic by one of your own graphics.

```
//S(Start)
   HS7=("Start example", se1, ac7)
   PRESS(HS7)
      LM("Maske4")
```

```
   END_PRESS
//END
//M(Maske4/"Example: MCP"/"mcp.png")
   DEF byte=(I/0/0/"Input byte=0 (default)","Byte number:",""/wr1,li1///380,40,100/480,40,50)
   DEF Feed=(IBB//0/"","Feed override",""/wr1//"EB3"/20,180,100/130,180,100), Axistop=(B//0/"","Feed
stop",""/wr1//"E2.2"/280,180,100/380,180,50/100)
   DEF Spin=(IBB//0/"","Spindle override",""/wr1//"EB0"/20,210,100/130,210,100), spinstop=(B//
0/"","Spindle stop",""/wr1//"E2.4"/280,210,100/380,210,50/100)
   DEF custom1=(IBB//0/"," User keys 1",""/wr1//"EB7.7"/20,240,100/130,240,100)
   DEF custom2=(IBB//0/"","User keys 2",""/wr1//"EB7.5"/20,270,100/130,270,100)
   DEF By1
   DEF By2
   DEF By3
   DEF By6
   DEF By7

   HS1=("Input byte", SE1, AC4)
   HS2=("")
   HS3=("")
   HS4=("")
   HS5=("")
   HS6=("")
   HS7=("")
   HS8=("")
   VS1=("")
   VS2=("")
   VS3=("")
   VS4=("")
   VS5=("")
   VS6=("")
   VS7=("Cancel", SE1, AC7)
   VS8=("OK", SE1, AC7)

   PRESS(VS7)
      EXIT
   END_PRESS

   PRESS(VS8)
      EXIT
   END_PRESS

   LOAD
      By1=1
      By2=2
      By3=3
```

```
      By6=6
      By7=7
   END_LOAD

   PRESS(HS1)
      Byte.wr=2
   END_PRESS

   CHANGE(Byte)
      By1=byte+1
      By2=byte+2
      By3=byte+3
      By6=byte+6
      By7=byte+7
      Feed.VAR="EB"<<By3
      Spin.VAR="EB"<<Byte
      Custom1.VAR="EB"<<By6
      Custom2.VAR="EB"<<By7
      Axisstop.VAR="E"<<By2<<".2"
      Spinstop.VAR="E"<<By2<<".4"
      Byte.wr=1
   END_CHANGE

   CHANGE(Axis stop)
      IF Axistop==0
         Axistop.BC=9
      ELSE
         Axistop.BC=11
      ENDIF
   END_CHANGE

   CHANGE(Spin stop)
      IF Spinstop==0
         Spinstop.BC=9
      ELSE
         Spinstop.BC=11
      ENDIF
   END_CHANGE
//END
```

# Dialog selection 11

## 11.1 Dialog selection using PLC softkeys

### Configuration

Description of the procedure:

- The "systemconfiguration.ini" contains a section [keyconfiguration]. The entry specifies an action for a special PLC softkey.
- A number is given as an action. A "Run MyScreens" call is involved if the number is greater than or equal to 100.
- A section for defining the action to be performed must be created in the "easyscreen.ini" file. The name of the section is based on the name of the operating area and the dialog name (see entry under [keyconfiguration] → Area:=..., Dialog:=...) → [<Area>_<Dialog>]
  → e.g. [AreaParameter_SlPaDialog]
- The action numbers (which were given in the "systemconfiguration.ini" → see Action:=...) are defined in this section. There are two commands involved:
  1. LS("Softkey menu1","param.com") ... Loading a softkey menu
  2. LM("Screen form1","param.com") ... Loading a screen form

### Selecting softkey menus via PLC softkeys

"Run MyScreens" makes it possible to select "Run MyScreens" softkey menus and "Run MyScreens" dialogs via PLC softkeys. This can only be done if the "action" attribute to be specified when configuring the relevant PLC softkeys has a value greater than or equal to 100.

PLC softkeys are configured in the file "systemconfiguration.ini" in the section [keyconfiguration]:

```
[keyconfiguration]
KEY75.1 = Area:=area, Dialog:=dialog, Screen:=screen, Action:= 100,
```

The LM and LS commands to be executed upon activation of the relevant PLC softkeys are configured in the "easyscreen.ini" file. The names of the sections that are used for the purpose of configuration are structured as follows:

| | |
|---|---|
| `[areaname_dialogname]` | The first part of the name "areaname" refers to the operating area and the second part "dialogname" designates the dialog to which the commands configured in this section apply. |
| `[AreaParameter_SlPaDialog]`<br>`100.screen1 = LS("Softkey1","param.com")`<br>`101.screen3 = LM("Screen form1","param.com")` | The names given in the "systemconfiguration.ini" file for the operating area and dialog should be used. The dialog does not have to be specified.<br>This is particularly true for operating areas which are only implemented by means of a single dialog. Please refer to the example on the left.<br>If "screen1" is displayed in the AreaParameter operating area implemented by the SlPaDialog dialog, the "LS("Softkey1","param.com")" command will be executed when the "action" with the value 100 occurs. |
| `action.screen=Command` | Both the "action" and "screen" attributes clearly indicate when the specified command will be executed.<br>The "screen" information is optional.<br>The following commands are permissible:<br>`LM (LoadMask)`<br>`LS (LoadSoftkeys)` |

### 11.1.1 Calling Run MyScreens-cycle screens in the program editor

#### Application

PLC keys provide the possibility to open Run MyScreens cycle screens directly in the program editor.

#### Boundary condition

A technology cycle can only be selected when the program editor is open. The technology cycle parameterized via the screen will be inserted at the current cursor position in the program editor.

### Procedure

1. Create the Run MyScreens cycle call via PLC hard keys (Page 282).
2. In order for the Run MyScreens screen to be properly integrated into the program editor, you must extend the PLC Key configuration as described in the following example:
   **Example of a complete key configuration:**
   systemconfiguration.ini
   ```
   [keyconfiguration]
   KEY101.0 = action:=209, runmyscreenmode:=HandledByDialog
   ```
   easyscreen.ini
   ```
   [AreaProgramEdit_SlProgramEdit]
   209 = LM("MASK_E_DR_O1_MAIN","e_dr_o1.com")
   ```
3. You can check the feedback from the open Run MyScreens cycle screen via the PLC interface "DB19.DBW24".
   **Example:**
   In this example, the value "9876" is written to the PLC interface when the Run MyScreens screen is opened:
   Configuration with PLC-ID=9876:
   //M(MASK_E_DR_O1_MAIN/$85305/////XG1,FA2,TA7//"drilling.txt"/**9876**)
   or
   //M{ MASK_E_DR_O1_MAIN, HD=$85305, LANGFILELIST=" drilling.txt", **PLC_ID=9876**}
   See also: Function Manual SINUMERIK 840D sl Basic Functions

### Status of the program editor on the OA interface

You can query the status of the program editor via the CAP local variable "/Hmi/StepEditorInfo".

| Name of variable: | /Hmi/StepEditorInfo |
|---|---|
| Description of variable: | Information from the editor about the program that has the focus. |
| The string contains the following information | |
| [fileName] | Program path definition |
| [channel] | Channel number |
| [technology] | Technology |
| [appearance] | Program identifier (ShopMill/ShopTurn/G code) |
| [state] | Step editor status:<br>• EditorClosed: Editor is closed (e.g. HMI not in program operating area)<br>• EditEnabled: Editing allowed<br>• EditDisabled: Editing not allowed (e.g. read only or invalid cursor position) |

# 11.2 Dialog selection using PLC hard keys

## Application

The following functions can be initiated in the operating software by the PLC:

- Select an operating area
- Select certain scenarios within operating areas
- Execute functions configured at softkeys

## Hardkeys

All keys - also the PLC keys - are subsequently referred to as hardkeys. A maximum of 254 hardkeys can be defined. The following allocation applies:

| Key number | Use |
|---|---|
| Key 1 – key 9 | Keys on the operator panel front |
| Key 10 – key 49 | Reserved |
| Key 50 – key 254<br>Key 50 – key 81 | PLC keys:<br>Reserved for OEMs |
| Key 255 | Pre-assigned by control information. |

Hardkeys 1 - 9 are pre-assigned as follows:

| Key designation | | Action / effect |
|---|---|---|
| HK1 | MACHINE | Selects "Machine" operating area, last dialog |
| HK2 | PROGRAM | Selects "Program" operating area, last dialog or last program |
| HK3 | OFFSET | Selects "Parameter" operating area, last dialog |
| HK4 | PROGRAM MANAGER | Selects "Program" operating area, "Program Manager" basic screen |
| HK5 | ALARM | Selects "Diagnostics" operating area, "Alarm list" dialog |
| HK6 | CUSTOM | Selects "Custom" operating area |
| HK7 1) | MENU SELECT | Selects "Main menu" |
| HK8 1) | MENU FUNCTION | Selects "Function" operating area |
| HK9 1) | MENU USER | Selects "User" operating area |

1) For 828D only

## Configuration

The configuring is realized in the "systemconfiguration.ini" configuration file in the section [keyconfiguration]. Each line defines what is known as a hardkey event. A hardkey event is the n-th actuation of a specific hardkey. For example, the second and third actuation of a specific hardkey can result in different responses.

The entries in the "systemconfiguration.ini" configuration file can be overwritten with user-specific settings. The directories [System user-directory]/cfg and [System oem-directory]/cfg are available for this purpose.

The lines for configuring the hardkey events have the following structure:

```
KEYx.n = area:=area, dialog:=dialog, screen:=screen, forms:=form,
menus:=menu,
action:=menu.action, cmdline:=cmdline
KEYx.n = area:=area, dialog:=dialog, cmdline:=cmdline, action:=
action
```

x: Number of the hardkey, range of values: 1 – 254

n: Event number – corresponds to the nth actuation of the hardkey, range of values: 0 – 9

### Requirement

The PLC user program must fulfill the following requirement:

Only one hardkey is processed. As a consequence, a new request can only be set if the operating software has acknowledged the previous request. If the PLC user program derives the hardkey from an MCP key, it must provide sufficient buffer storage of the key(s) to ensure that no fast keystrokes are lost.

### PLC interface

An area to select a hardkey is provided in the PLC interface. The area is in DB19.DBB10. Here, the PLC can directly specify a key value of between 50 and 254.

Acknowledgment by the operating software takes place in two steps. This procedure is necessary so that the operating software can correctly identify two separate events if the same key code is entered twice consecutively. In the first step, control information 255 is written to byte DB19.DBB10. This defined virtual key stroke enables the HMI to identify every PLC key sequence uniquely. The control information is of no significance to the PLC user program and must not be changed. In the second step, the actual acknowledgment takes place with respect to the PLC by clearing DB19.DBB10. From this point in time, the PLC user program can specify a new hardkey. In parallel, the actual hardkey request is processed in the operating software.

### Example

Configuration file:

```
; PLC hardkeys (KEY50-KEY254)
[keyconfiguration]
KEY50.0 = name := AreaMachine, dialog := SlMachine
KEY51.0 = name := AreaParameter, dialog := SlParameter
KEY52.0 = name := AreaProgramEdit, dialog := SlProgramEdit
KEY53.0 = name := AreaProgramManager, dialog := SlPmDialog
KEY54.0 = name := AreaDiagnosis, dialog := SlDgDialog
KEY55.0 = name := AreaStartup, dialog := SlSuDialog
KEY56.0 = name := Custom, dialog := SlEsCustomDialog
```

The area and dialog identifiers can be found in the "systemconfiguration.ini" from [system siemens-directory]/cfg.

## 11.3 Dialog selection via NC

### MMC command in HMI Operate

You can use MMC commands as described in the following:

1. **Definition of MMC commands**
   The following combinations are available in the standard "systemconfiguration.ini" file:
   address:=MCYCLES --> command:=LM
   address:=CYCLES --> command:=PICTURE_ON
   This distinction is necessary, in order to make a differentiation between measuring cycles and other cycles. This means that:
   - LM is always applicable for measuring cycles, PICTURE_ON always for non-measuring cycles
   - It is not permissible that newly defined MMC commands are designated PICTURE_ON and LM
2. **"Run MyScreens" license**
   All of the dialogs opened by "Run MyScreen" – with the exception of the measuring cycles – fall under the "Run MyScreens" license, and therefore without license, only a limited number of dialogs can be used.
   Example call with "test.com" (Run MyScreens):

```
g0 f50
MMC("CYCLES,PICTURE_ON,test.com,screen form1","A")
m0
MMC("CYCLES,PICTURE_OFF","N")
M30
```

# Examples for cycle masks 12

## 12.1 Examples for cycle masks

When installing SINUMERIK Operate, examples for cycle masks created with Run MyScreens are also saved to the system.

The Run MyScreens examples are available under the following archive paths:

| [System siemens-directory] | | |
|---|---|---|
| ...\siemens\sinumerik\hmi\template\easyscreen\ | | |
| ... | | |
| | standard\ | Help screens, PNG |
| | x3d\ | Help screens / animations |
| ... | | |
| | Milling\ | Fräsbearbeitung |
| | Turning\ | Drehbearbeitung |
| ... | | |
| | deu\ | German description |
| | eng\ | English description |

Examples in the commissioning operating area can be found on the HMI user interface:

Horizontal softkey system data → HMI data → Templates → Examples → Easy Screen → ... (see above).

### Brief description of the examples

- Milling
  - Drilling example: This is where a position can be added
  - Measure workpiece example: This is where a cycle call can be generated and processed via NC start while the dialog is open
  - Workpiece counter example: This shows how GUDs should be handled
- Turning
  - Fixtures: this contains a tailstock, bar loader, part catcher
  - Measure workpiece example: This is where a cycle call can be generated and processed via NC start while the dialog is open
  - Drilling example: This is where a position can be added

# Configuration in the side screen

# 13

The following options are available to you for displaying Run MyScreens configurations in the side screen:

1. Display of a Run MyScreens configuration in a (separate) side screen page:
   The Run MyScreens configuration takes up all of the space of the side screen page.
2. Display of several Run MyScreens configurations in a (separate) side screen page:
   The individual Run MyScreens configurations are displayed above each other within the side screen page, in so-called side screen elements. The side screen elements, and thus also the Run MyScreens configurations, can be scrolled vertically.
3. Adding one or more Run MyScreens configurations to an existing side screen page:
   I.e. the Run MyScreens configuration is added to a side screen page contained in the SIEMENS delivery kit. This case matches point 2 with the exception of the affected side screen page.
4. Adding one or more Run MyScreens configurations to a side screen element:
   The individual Run MyScreens configurations are displayed next to each other in so-called side screen widgets and can be scrolled horizontally.

## 13.1 Display of a Run MyScreens configuration in a side screen page

To display a Run MyScreens configuration in a (separate) side screen page, you must add a new side screen page to the existing side screen configuration ("slsidescreen.ini"). The Run MyScreens configuration is then displayed or executed in this side screen page.

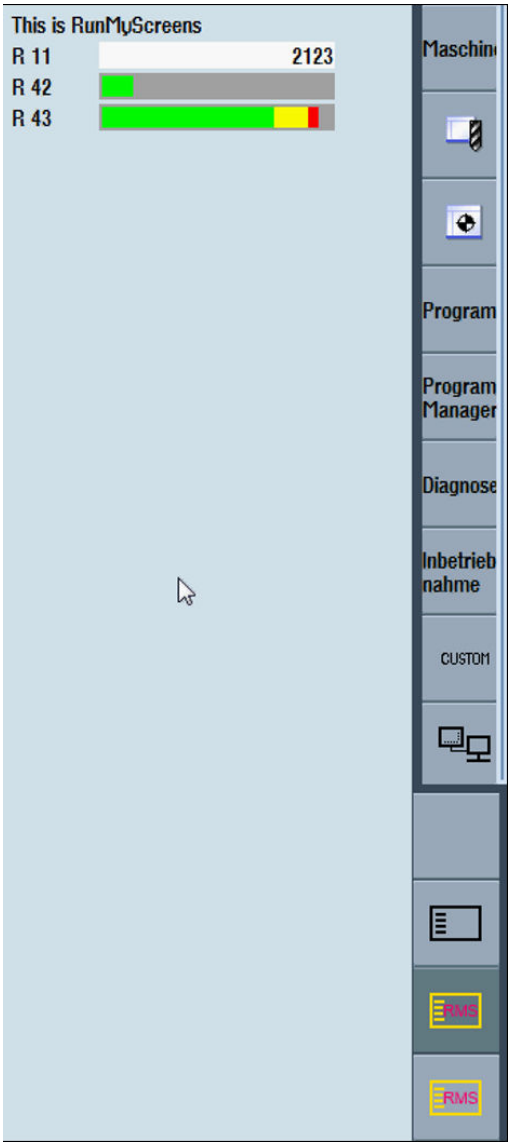



Figure 13-1 **Display of a Run MyScreens configuration in a side screen page**

### Procedure

Expand the file "slsidescreen.ini" by the following entry:

```
[Sidescreen]
PAGE100= name:=RMSPage,
implementation:=sleseasyscreen.SlEsSideScreenPage
```

- The number of the page, "100" in this case, must be counted up corresponding to existing side screen pages. The number of the page must be ≥ 100. This avoids conflicts with the SIEMENS pages.
- You can freely select the name of the page, "RMSPage" in this case.
- You cannot change the specification for implementing the page, "sleseasyscreen.SlEsSideScreenPage" in this case.

Configure the Run MyScreens configuration that is to be carried out in the next step. To do this, create a new section with a name that contains the name of the newly created page:

```
[Page_RMSPage]
Icon=rmspage.png
PROPERTY001= name:=maskPath, type:=QString,
value:="sidescreenmask.com"
PROPERTY002= name:=maskName, type:=QString, value:="Mask"
PROPERTY003= name:=focusable, type:=bool, value:="true"
```

- The name of the section is formed by appending the name of the page to the prefix "Page_", in this case "RMSPage".
- Under Icon, specify the file name, "rmspage.png" in this case. The file contains the icon for the button for selecting the page. Store the file in the directory /user/sinumerik/hmi/ico/ico640 or /oem/sinumerik/hmi/ico/ico640.
- In the property maskPath, specify the name of the file that contains the RunMyScreens configuration, "sidescreenmask.com" in this case.
- In the property maskName, specify the name of the RunMyScreens mask that is to be displayed, "sidescreenmask.com" in this case.
- Specify the input focus in the Property focusable. The page can be assigned the input focus only if this attribute has the value "true". This attribute is required for pages with input elements.

Store the "slsidescreen.ini" in the directory /oem/sinumerik/hmi/cfg or /user/sinumerik/hmi/cfg.

The Run MyScreens configuration is available after the next HMI power-up.

## Example

Example for a complete configuration in "slsidescreen.ini":

```
[Sidescreen]
PAGE005= name:=RMSPage,
implementation:=sleseasyscreen.SlEsSideScreenPage
[Page_RMSPage]
PROPERTY001= name:=maskPath, type:=QString,
value:="sidescreenmask.com"
PROPERTY002= name:=maskName, type:=QString, value:="Mask"
```

## 13.2 Display of several Run MyScreens configurations in a side screen page

For the display of several Run MyScreens configurations in a (separate) side screen page, you must also configure so-called side screen elements, in addition to the side screen page. The side screen elements are used to display the Run MyScreens configurations.

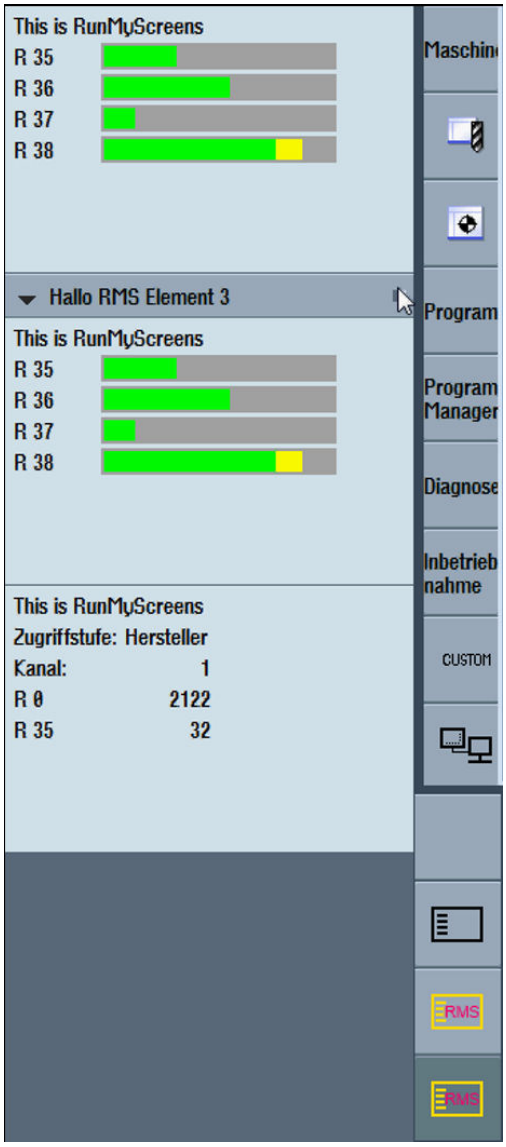



Figure 13-2 **Display of several Run MyScreens configurations in a side screen page**

### Procedure

Expand the "slsidescreen.ini" by the following entries:

```
[Sidescreen]
PAGE100= name:=RMSPage, implementation:=SlSideScreenPage
```

- The number of the page, "100" in this case, must be counted up corresponding to existing side screen pages. The number of the page must be ≥ 100. This avoids conflicts with the SIEMENS pages.
- You can freely select the name of the page, "RMSPage" in this case.
- You cannot change the specification for implementing the page, "SlSideScreenPage" in this case.

```
[Page_RMSPage]
ELEMENT001= name:=RMSElement001,
implementation:=sleseasyscreen.SlEsSideScreenElement
ELEMENT002= name:=RMSElement002,
implementation:=sleseasyscreen.SlEsSideScreenElement
```

- Two side screen elements are assigned to the page – RMSElement001 and RMSElement002. You can freely select the names of the elements.
- You cannot change the specification for implementing the elements, "sleseasyscreen.SlEsSideScreenElement" in this case.

Finally, you must assign Run MyScreens configurations that are still to be displayed in these elements to the side screen elements:

```
[Element_RMSElement001]
PROPERTY001= name:=maskPath, type:=QString,
value:="sidescreenmask.com"
PROPERTY002= name:=maskName, type:=QString, value:="Mask001"
…
[Element_RMSElement002]
PROPERTY001= name:=maskPath, type:=QString,
value:="sidescreenmask.com"
PROPERTY002= name:=maskName, type:=QString, value:="Mask002"
```

Store the "slsidescreen.ini" in the directory /oem/sinumerik/hmi/cfg or /user/sinumerik/hmi/cfg.

The Run MyScreens configuration is available after the next HMI power-up.

## 13.3 Adding Run MyScreens configurations to a side screen page

The procedure for integrating one or several Run MyScreens configurations for an existing side screen page is identical to the procedure described in Chapter Display of several Run MyScreens configurations in a side screen page (Page 290). The only difference is that the side screen elements for displaying the Run MyScreens configurations are added to an existing side screen page.

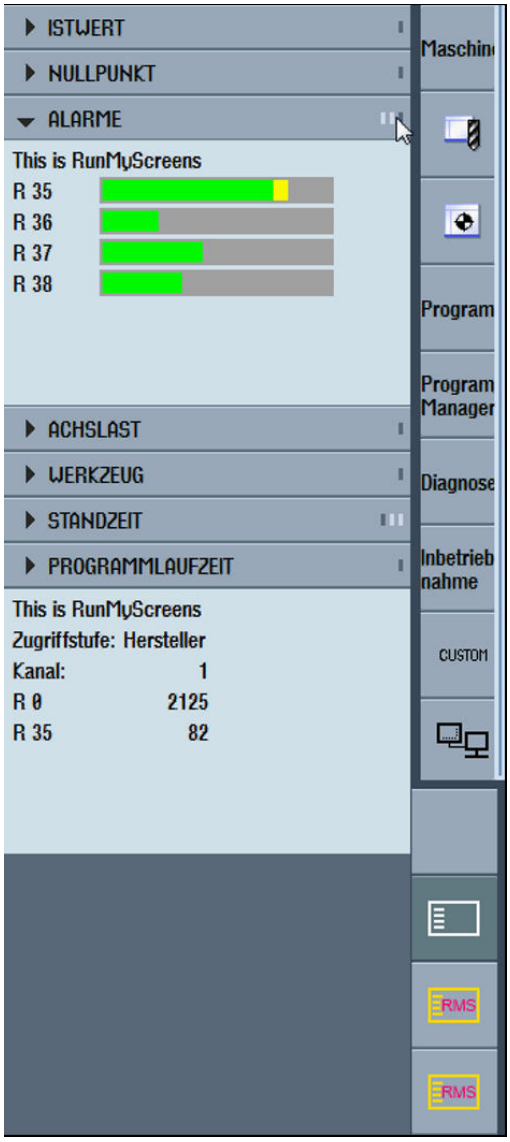



Figure 13-3 **Adding Run MyScreens configurations to a side screen page**

**Note**

Of the side screen pages contained in the SIEMENS delivery kit, you can only expand the WidgetsPage (see file "slsidescreen.ini") with Run MyScreens configurations.

### Procedure

Expand the "slsidescreen.ini" by the following entries:

In the [Page_WidgetsPage] section, add a corresponding element for receiving the Run MyScreens configuration.

```
[Page_WidgetsPage]
ELEMENT100= name:=RMSElement,
implementation:=sleseasyscreen.SlEsSideScreenElement
```

- Use a number range ≥ 100 for new elements. This avoids conflicts with the SIEMENS display elements. The RMSElement is inserted in the WidgetsPage under the SIEMENS elements according to this numbering.

```
[Element_RMSElement]
PROPERTY001= name:=maskPath, type:=QString,
value:="sidescreenmask.com"
PROPERTY002= name:=maskName, type:=QString, value:="Mask001"
```

## 13.4 Adding Run MyScreens configurations to a side screen element

The basis of the procedure that is described in the following as example is a side screen page with a side screen element. Two Run MyScreens configurations are displayed next to each other in the side screen element.

Configure a side screen page with a side screen element as described in the following. Assign two side screen widgets with the respective Run MyScreens configurations to the side screen element.

### Procedure

Expand the "slsidescreen.ini" by the following entries:

```
[Sidescreen]
PAGE100= name:=RMSPage, implementation:=SlSideScreenPage
```

- You cannot change the specification for the implementation, "SlSideScreenPage" in this case.

```
[Page_RMSPage]
ELEMENT001= name:=RMSElement, implementation:= SlSideScreenElement
```

- You cannot change the specification for the implementation, "SlSideScreenElement" in this case.

```
[Element_RMSElement]
TextId=RMS_ELEMENT_TITLE
TextFile=rmstexts
TextContext=rmstexts
TextContext=rmstexts
WIDGET001= name:=RMSWidget001,
implementation:=sleseasyscreen.SlEsSideScreenWidget
WIDGET002= name:=RMSWidget002,
implementation:=sleseasyscreen.SlEsSideScreenWidget
```

- When the SlSideScreenElement implementation is used for the side screen element (see Section [Page_RMSPage]), the side screen element has a title bar in which a text can be displayed.
  This text is configured with the entries TextId, TextFile and TextContext:
  - TextId: Text ID of the text to be displayed
  - TextFile: File in which the text is saved
  - TextContext: Text context, in which the text is located within this file

  A separate file must be provided for each language.
  The name of these files is formed according to the following diagram:
  <TextFile>_<Sprachsuffix>.ts, e.g. rmstexts_deu.ts for the example above.
  If you do not specify a file or a context (TextFile=<empty> and TextContext=<empty>), the text ID is displayed as text. You can specify any string as the text ID.
  The text file (e.g. rmstexts_deu.ts) is structured as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE TS>
<TS>
 <context>
 <name>rmstexts</name>
 <message>
 <source>My application</source>
 <translation>My Application</translation>
 </message>
 </context>
</TS>
```

  Store the file in the directory /oem/sinumerik/hmi/lng or /user/sinumerik/hmi/lng. During the next HMI power-up, the file is converted to the file format that is needed during runtime.
- You can freely select the names of the widgets assigned to the element, "RMSWidget001" and "RMSWidget002" in this case.
- You must not change the widget implementation "sleseasysscreen.SlEsSideScreenWidget".

```
[Widget_RMSWidget001]
PROPERTY001= name:=maskPath, type:=QString, value:="mymask001.com"
PROPERTY002= name:=maskName, type:=QString, value:="MyMask001"

[Widget_RMSWidget002]
PROPERTY001= name:=maskPath, type:=QString, value:="mymask002.com"
PROPERTY002= name:=maskName, type:=QString, value:="MyMask002"
```

- The names of the sections are formed for configuring the side screen widgets by appending the name of the widget that is to be configured in this section to the prefix "Widget_".
- In the Property maskPath, specify the name of the file that contains the RunMyScreens configuration, "mymask001.com" or "mymask002.com" in this case.
- In the Property maskName, specify the name of the RunMyScreens mask that is to be displayed, "MyMask001" or "MyMask002" in this case.

## 13.5 Notes on carrying out Run MyScreens configurations in the side screen

Note the following:

- The functions LS, LM, DLGL, EXIT, EXITLS are not available when carrying out Run MyScreens configurations in the side screen.
- It is not possible to query the mask size in the LOAD block.
- Texts are always displayed using the font that is used in SINUMERIK Operate with a screen resolution of 640x480 pixels.
- The configured positions of operator controls are implemented without changes (e.g. extension).
- The dialog size is adapted automatically.
- If Run MyScreens configurations are carried out in a side screen page, the entire area of the page is available for the configuration. When carrying out in side screen elements or widgets, the area available for display is specified by the system depending on the context.
- The header and softkeys cannot be configured.
- All of the debugging and error outputs are sequentially written to the text file "easyscreen_log.txt". There is no specific identification indicating which configuration a message comes from.
- The status of a Run MyScreens configuration displayed in the side screen is rejected when the configuration is faded out.

**Note**

The integration of Run MyScreens configurations into the side screen can lead to more than one Run MyScreens configuration being carried out simultaneously. To maintain the functional capability of the overall system, take note of the additional system load caused as a result of the hardware used.

# Configuration in the Display Manager 14

## 14.1 Displaying Run MyScreens configurations in the Display Manager

The SISideScreenDialog dialog is available to you for displaying Run MyScreens configurations in the Display Manager. Depending on the space available, this dialog can display one or more side screen pages (see IM9 Chapter 3.13). Several pages are displayed next to each other in column format. The available space (width) is evenly divided among the individual columns. The number of pages to be displayed and their content are configured. Each page has its own configuration file. The names of these configuration files are specified during the dialog configuration in the "systemconfiguration.ini" file, in the command line parameter "cmdline". Behind the parameter identifier "–sidescreen1" is the name of the configuration file for the first column; behind the parameter identifier "–sidescreen2" is the name of the configuration file for the second column, etc.

**Example**

```
[dialogs]
DLG500= name:=RMSApp,
implementation:=sldmsidescreenapp.SlSideScreenDialog,
process:=SlHmiHost1, preload:=false, cmdline:="-sidescreen1
rmspage1.ini –sidescreen2 rmspage2.ini –spacing 3”
```

The instance of the SlSideScreenDialog dialog created in this example has the name RMSApp. It has 2 pages/columns. The two pages/columns are configured in the files "rmspage1.ini" and "rmspage2.ini". Between the two columns, there is a distance of 3 pixels.

---

**Note**

You can freely select the names of the files with the page configurations, "rmspage1.ini" and "rmspage2.ini" in this case.

---

The use of Run MyScreens configurations in the Display Manager is done as described in Chapter Configuration in the side screen (Page 287). The only difference is that the integration of the configurations is not done in the file "slsidescreen.ini", but in the respective files provided for the configuration of the existing pages.

2 variants are available to you for the integration of Run MyScreens configurations into the Display Manager of SINUMERIK Operate. These are described in the following sections.

## 14.2 Integration in SISideScreenDialog

For the integration of the Run MyScreens configuration, you must create the dialog instance of SlSideScreenDialog in the file "systemconfiguration.ini" and create the file for integrating the Run MyScreens configuration.

The following example shows the integration of a Run MyScreens configuration. The Run MyScreens configuration takes up the entire window of the SlSideScreenDialog dialog, i.e. only one page/column exists.

**Procedure**

First, create an instance of the SlSideScreenDialog in the "systemconfiguration.ini" file in the `[dialogs]` section. The instance is given the name "RMSApp".

```
[dialogs]
DLG500= name:=RMSApp,
implementation:=sldmsidescreenapp.SlSideScreenDialog,
process:=SlHmiHost1, preload:=false,
cmdline:=”-sidescreen1 rmsapp.ini”
```

In the next step, you will configure or integrate the RunMyScreens configuration in the "rmsapp.ini" file:

```
[Sidescreen]
PAGE001= name:=RMSPage, implementation:=SlSideScreenPage
```

- Create a page with the name "RMSPage".

```
[Page_RMSPage]
ELEMENT001= name:=RMSElement,
implementation:=sleseasyscreen.SlEsSideScreenElement
```

- You create an element with the name RMSElement on the RMSPage. This element is used for displaying the Run MyScreens configuration.

```
[Element_RMSElement]
PROPERTY001= name:=maskPath, type:=QString,
value:="sidescreenmask.com"
PROPERTY002= name:=maskName, type:=QString, value:="Mask"
```

- In the property maskPath, specify the name of the file that contains the RunMyScreens configuration, "sidescreenmask.com" in this case.
- In the property maskName, specify the name of the RunMyScreens mask that is to be displayed, "sidescreenmask.com" in this case.

Store the files "rmsapp.ini" and "systemconfiguration.ini" (see above) in the directory /oem/sinumerik/hmi/cfg or /user/sinumerik/hmi/cfg.

The Run MyScreens configuration is available after the next HMI power-up.

## 14.3 Integration into SIWidgetsApp

The integration of a Run MyScreens configuration into the SlWidgetsApp application, which is part of the SIEMENS delivery kit, is described in the following.

Depending on which page/column of the SlWidgetsApp application the Run MyScreens configuration is to be integrated into, you must expand either the "sldmwidgets1.ini" file or the "sldmwidgets2.ini" file accordingly.

The following example shows the integration of a Run MyScreens configuration into the left page/column of the SIWidgetsApp application.

### Procedure

Expand the "sldmwidgets1.ini" file as follows:

```
[Sidescreen]
PAGE001= name:=WidgetsPage, implementation:=SlSideScreenPage
[Page_WidgetsPage]
ELEMENT100= name:=RMSElement,
implementation:=sleseasyscreen.SlEsSideScreenElement
```

- Create an element with the name "RMSElement" for displaying the Run MyScreens configuration.

**Note**

Use a number range ≥ 100 for new elements. This avoids conflicts with the SIEMENS display elements. The RMSElement is inserted in the WidgetsPage under the SIEMENS elements according to this numbering.

```
[Element_RMSElement]
PROPERTY001= name:=maskPath, type:=QString,
value:="sidescreenmask.com"
PROPERTY002= name:=maskName, type:=QString, value:="Mask"
```

- In the property maskPath, specify the name of the file that contains the RunMyScreens configuration, "sidescreenmask.com" in this case.
- In the Property maskName, specify the name of the RunMyScreens mask that is to be displayed, "Mask" in this case.

Save the file "sldmwidgets1.ini" in the directory /oem/sinumerik/hmi/cfg or /user/sinumerik/hmi/cfg .

The Run MyScreens configuration is available after the next HMI power-up.

## 14.4 Notes on carrying out Run MyScreens configurations in the Display Manager

Note the following:

- The functions LS, LM, DLGL, EXIT, EXITLS are not available when carrying out Run MyScreens configurations in the Display Manager.
- It is not possible to query the mask size in the LOAD block.
- Texts are always displayed using the font that is used in SINUMERIK Operate with a screen resolution of 640x480 pixels.
- The configured positions of operator controls are implemented without changes (e.g. extension).

- If Run MyScreens configurations are carried out in a side screen page, the entire area of the page is available for the configuration. When carrying out in side screen elements or widgets, the area available for display is specified by the system depending on the context.
- All of the debugging and error outputs are sequentially written to the text file easyscreen_log.txt. There is no specific identification indicating which configuration a message comes from.
- The status of a Run MyScreens configuration displayed in the Display Manager is rejected when the configuration is faded out.

**Note**

The integration of Run MyScreens configurations into the Display Manager can lead to more than one Run MyScreens configuration being carried out simultaneously. To maintain the functional capability of the overall system, take note of the additional system load caused as a result of the hardware used.

# Reference lists

# A

## A.1 Lists of start softkeys

### A.1.1 List of start softkeys for turning

#### Program operating area for turning

| HSK1 | HSK2 | HSK3 | HSK4 | HSK5 | HSK6 | HSK7 | HSK8 |
|---|---|---|---|---|---|---|---|
| **Edit** | **Drilling** | **Turning** | **Contour turning** | **Milling** | **Miscellaneous** | **Simulation** | **NC select** |
| HSK9 | HSK10 | HSK11 | HSK12 | HSK13 | HSK14 | HSK15 | HSK16 |
| -- | Straight-line circle (only for ShopTurn) | -- | -- | Measure workpiece | Measure tool | **OEM** | -- |

#### Turning

The following tables list the possible start softkeys for turning technology. Assignments of individual start softkeys can differ depending on the particular system. The specified OEM softkeys are permitted for "Run MyScreens".

programGUIDE (G-Code) start softkeys:

| | **Drilling** | **Turning** | **Contour turning** | | **Milling** | | **Miscellaneous** | |
|---|---|---|---|---|---|---|---|---|
| | **HSK2** | **HSK3** | **HSK4** | | **HSK5** | | **HSK6** | |
| **VSK1** | Centering | Stock removal | Contour | -- | Face milling | Contour | Settings | High speed settings |
| **VSK2** | Drilling reaming | Groove | Stock removal | -- | Pocket | Path | Swivel plane | Parallel axes |
| **VSK3** | Deep-hole drilling | Undercut | Stock removal residual material | -- | Multi-edge spigot | Predrilling | Swivel tool | -- |
| **VSK4** | Boring | Thread | Grooving | -- | Groove | Pocket | -- | -- |
| **VSK5** | Thread | Parting | Grooving residual material | -- | Thread milling | Pocket res. mat. | -- | -- |
| **VSK6** | **OEM** | -- | Plunge-turning | -- | Engraving | Spigot | Subprogram | -- |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **VSK7** | Positions | **OEM** | Plunge turning residual material | **OEM** | **OEM** | Spigot res. mat. | -- | **OEM** |
| **VSK8** | Repeat position. | -- | >> | << | Contour milling | << | >> | << |

ShopTurn start softkeys:

| | **Drilling** | **Turning** | **Contour turning** | | **Milling** | | **Miscellaneous** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **HSK2** | **HSK3** | **HSK4** | | **HSK5** | | **HSK6** | | **HSK10** |
| **VSK1** | Drilling centered | Stock removal | New contour | -- | Face milling | New contour | Settings | High speed settings | Tool |
| **VSK2** | Centering | Groove | Stock removal | -- | Pocket | Path | Swivel plane | Parallel axes | Straight line |
| **VSK3** | Drilling reaming | Undercut | Stock removal residual material | -- | Multi-edge spigot | Predrilling | Swivel tool | Repeat progr. | Circle center point |
| **VSK4** | Deep-hole drilling | Thread | Grooving | -- | Groove | Pocket | Counter-spindle | -- | Circle radius |
| **VSK5** | Thread | Parting | Grooving residual material | -- | Thread milling | Pocket res. mat. | Transformations | -- | Polar |
| **VSK6** | **OEM** | -- | Plunge-turning | -- | Engraving | Spigot | Subprogram | -- | Approach/ retract |
| **VSK7** | Positions | **OEM** | Plunge turning residual material | **OEM** | **OEM** | Spigot res. mat. | -- | **OEM** | -- |
| **VSK8** | Repeat position. | -- | >> | << | Contour milling | << | >> | << | -- |

## A.1.2 List of start softkeys for milling

### Program operating area when milling

| HSK1 | HSK2 | HSK3 | HSK4 | HSK5 | HSK6 | HSK7 | HSK8 |
|---|---|---|---|---|---|---|---|
| **Edit** | **Drilling** | **Milling** | **Contour milling** | **Turning** (only for G code) | **Miscellaneous** | **Simulation** | **NC select** |
| HSK9 | HSK10 | HSK11 | HSK12 | HSK13 | HSK14 | HSK15 | HSK16 |
| -- | Straight-line circle (only for ShopMill) | -- | -- | Measure workpiece | Measure tool | **OEM** | -- |

## Milling

The following tables list the possible start softkeys for milling technology. Assignments of individual start softkeys can differ depending on the particular system. The specified OEM softkeys are permitted for "Run MyScreens".

programGUIDE (G-Code) start softkeys:

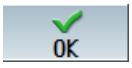
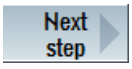
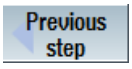
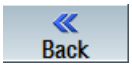
| | **Drilling** | **Milling** | **Contour milling** | | **Turning** | | **Miscellaneous** | |
|---|---|---|---|---|---|---|---|---|
| | **HSK2** | **HSK3** | **HSK4** | | **HSK5** | | **HSK6** | |
| **VSK1** | Centering | Face milling | Contour | -- | Stock removal | Contour | Settings | -- |
| **VSK2** | Drilling reaming | Pocket | Path | -- | Groove | Stock removal | Swivel plane | Parallel axes |
| **VSK3** | Deep-hole drilling | Multi-edge spigot | Predrilling | -- | Undercut | Stock removal residual material | Swivel tool | -- |
| **VSK4** | Boring | Groove | Pocket | -- | Thread | Grooving | High speed settings | -- |
| **VSK5** | Thread | Thread milling | Pocket res. mat. | -- | Parting | Grooving residual material | -- | -- |
| **VSK6** | **OEM** | Engraving | Spigot | -- | -- | Plunge-turning | Subprogram | -- |
| **VSK7** | Positions | **OEM** | Spigot res. mat. | **OEM** | **OEM** | Plunge turning residual material | -- | **OEM** |
| **VSK8** | Repeat position. | -- | >> | << | Contour turning | << | >> | << |

ShopMill start softkeys:

| | **Drilling** | **Milling** | **Contour milling** | | **Miscellaneous** | | **Straight line circle** |
|---|---|---|---|---|---|---|---|
| | **HSK2** | **HSK3** | **HSK4** | | **HSK6** | | **HSK10** |
| **VSK1** | Centering | Face milling | New contour | -- | Settings | -- | Tool |
| **VSK2** | Drilling reaming | Pocket | Path | -- | Swivel plane | Parallel axes | Straight line |
| **VSK3** | Deep-hole drilling | Multi-edge spigot | Predrilling | -- | Swivel tool | Repeat progr. | Circle center point |
| **VSK4** | Boring | Groove | Pocket | -- | High speed settings | -- | Circle radius |
| **VSK5** | Thread | Thread milling | Pocket res. mat. | -- | Transformations | -- | Helix |
| **VSK6** | **OEM** | Engraving | Spigot | -- | Subprogram | -- | Polar |
| **VSK7** | Positions | **OEM** | Spigot res. mat. | **OEM** | -- | **OEM** | -- |
| **VSK8** | Repeat position. | -- | >> | << | >> | << | -- |

## A.2 List of predefined softkeys

Table A-1 Predefined softkeys

| Name | Softkey |
|---|---|
| SOFTKEY_OK |  |
| SOFTKEY_CANCEL |  |
| SOFTKEY_APPLY |  |
| SOFTKEY_MORE |  |
| SOFTKEY_BACK |  |
| SOFTKEY_ASSISTANT_NEXT |  |
| SOFTKEY_ASSISTANT_PREVIOUS |  |
| SOFTKEY_NAV_BACK |  |

## A.3 List of access levels

The meanings of the different access levels are as follows:

| Protection level | Locked by | Area |
|---|---|---|
| ac0 | Reserved for Siemens | |
| ac1 | Password | Machine manufacturer |
| ac2 | Password | Service |
| ac3 | Password | User |
| ac4 | Keyswitch position 3 | Programmer, machine setter |
| ac5 | Keyswitch position 2 | Qualified operator |
| ac6 | Keyswitch position 1 | Trained operator |
| ac7 | Keyswitch position 0 | Semi-skilled operator |

# A.4 List of colors

## System colors

A uniform color table is available for configuring dialogs (subset of the respective standard colors). The color of an element (text, input field, background, etc.) can be selected from the following options (between 0 and 133).

As an alternative to the predefined colors, you can also specify colors as RGB values ("#RRGGBB").

| Index | Pictogram | Color | Color description |
|---|---|---|---|
| 1 | | Black | |
| 2 | | Orange | |
| 3 | | Dark green | |
| 4 | | Light gray | |
| 5 | | Dark gray | |
| 6 | | Blue | |
| 7 | | Red | |
| 8 | | Brown | |
| 9 | | Yellow | |
| 10 | | White | |
| 126 | | Black | Font color of an input/output field that is currently in focus |
| 127 | | Light orange | Background color of an input/output field that is currently in focus |
| 128 | | Orange | System color focus |
| 129 | | Light gray | Background color |
| 130 | | Blue | Header color (active) |
| 131 | | Black | Header font color (active) |
| 132 | | Turquoise | Background color of a toggle field |
| 133 | | Light blue | Background color of a list box |

## A.5 List of language codes used in file names

### Supported languages

Standard languages:

| Language | Abbreviation in file name |
|---|---|
| Chinese simplified | chs |
| German | deu |
| English | eng |
| French | fra |
| Italian | ita |
| Spanish | esp |

Other languages:

| Language | Abbreviation in file name |
|---|---|
| Chinese traditional | cht |
| Danish | dan |
| Finnish | fin |
| Indonesian | ind |
| Japanese | jpn |
| Korean | kor |
| Malay | msl |
| Dutch | nld |
| Polish | plk |
| Portuguese (Brazil) | ptb |
| Romanian | rom |
| Russian | rus |
| Swedish | sve |
| Slovakian | sky |
| Slovenian | slv |
| Thai | tha |
| Czech | csy |
| Turkish | trk |
| Hungarian | hun |
| Vietnamese | vit |

## A.6 List of accessible system variables

### References

List Manual System Variables/PGAsl/

## A.7 Behavior when opening the dialog (attribute CB)

The following table provides an overview of the conditions when the CHANGE method is called.

The following applies to attribute CB:

**CB0**

The CHANGE method is triggered when the screen is displayed if the variable has a valid value at this time (e.g. through default setting or NC/PLC variable).

**CB1 (default)**

The CHANGE method is not explicitly triggered when the screen is displayed. If the variable has a configured NC/PLC variable, then the CHANGE method is of course still called.

| Condition | | | | Response |
|---|---|---|---|---|
| **Type** | **System or user variable** | **Default setting** | **Execute the CHANGE method** | |
| I/O field | Yes | Yes | Yes | Due to the configured NC/PLC variable, there is always at least one automatic call of the CHANGE method with the current value of the NC/PLC variable. The pre-assignment of CB does not have an effect. |
| | | | No | |
| | | No | Yes | |
| | | | No | |
| | No | Yes | Yes | The CHANGE method is called with the pre-assigned value. |
| | | | No | The CHANGE method is not called. |
| | | No | Yes | No call because there is no valid value present to call the CHANGE method. |
| | | | No | |
| Toggle | Yes | Yes | Yes | Due to the configured NC/PLC variable, there is always at least one automatic call of the CHANGE method with the current value of the NC/PLC variable. The pre-assignment of CB does not have an effect. |
| | | | No | |
| | | No | Yes | |
| | | | No | |
| | No | Yes | Yes | The CHANGE method is called with the pre-assigned value. |
| | | | No | The CHANGE method is not called. |
| | | No<br>(per default the first value from the toggle is assigned) | Yes | The CHANGE method is called with the first pre-assigned value of the toggle list. |
| | | | No | The CHANGE method is not called. |

# Tips and tricks

# B

## B.1 General tips

- If possible, use the OPI version for system variables ($ variables).
  Reason:
  Avoids complex name resolution.
  Example:
  Instead of "$R[10]", it is better to use "/Channel/Parameter/R[u1,10]".
- Avoids cyclic (similar) setting, e.g. of the screen and variable HLP property. Compare the value to be set with the current value first, and only set it when it is different.
  Reason:
  Avoid a time-consuming search of the resolution-dependent help screens.
  Example:

```
DEF MyVar=(R3///,"X1",,"mm"/WR1//"$AA_IM[0]")
CHANGE(MyVar)
   IF MyVar.VAL < 100
      HLP="mypic1.png"
   ELSE
      HLP="mypic2.png"
   ENDIF
END_CHANGE
```

**Recommendation:**

```
CHANGE(MyVar)
   IF MyVar.VAL < 100
      IF HLP <> "mypic1.png"
         HLP="mypic1.png"
      ENDIF
   ELSE
      IF HLP <> "smypic2.png"
         HLP="mypic2.png"
      ENDIF
   ENDIF
END_CHANGE
```

- Replace several RNP() functions in succession with one MRNP() function.
  Replace several RDOP() functions in succession with one MRDOP() function.
  Reason:
  Reduces the communication load and increases the performance.
- Do not read drive parameters faster than in a 1 second cycle, slower is better.
  Reason: The communication with the drives can otherwise be extremely disturbed or even cause failures.
- Avoid successive arithmetic operations with system or user variable-connected dialog variables. Use, for example, register (REG[x]) or (invisible) help variables for this.
  Reason: Each assignment of a value results in a write operation in the connected system or user variable.
- Similar code used in different blocks should be combined in a SUB block for reasons of transparency, serviceability and performance (when displaying the screen). It can then be called at the appropriate positions with the CALL() function.
- By monitoring the CPU load in the dialog line (setting in slguiconfig.xml), it is possible to investigate the effects of changes in the screen on the performance.

## B.2 Tips for debugging

- Use the DEBUG() function for diagnostic purposes. When the DEBUG() function is called, the transferred string is written to the "easyscreen_log.txt" file. The output of information in the dialog line by the DLGL() function can also be helpful.
  However, after completion of the screen development, this function should be removed again for performance reasons or commented out.
- The "easyscreen_log.txt" file should have no entries after the development of a screen is completed.

## B.3 Tips for the CHANGE method

- Always keep CHANGE methods very short and small, particularly for those whose variables are connected to a system or user variable and change very frequently.
  Reason:
  Increases the screen performance.
- If possible, do not configure any RNP() functions in CHANGE methods. Instead, it is better to configure an invisible variable parallel to the system or user variable to be read and then use this.
  Reason:
  An RNP() function would be issued with each call. Otherwise, the current value which is already available would be simply accessed.

Example:

A name resolution is performed via RNP() for each change of the axis motion in order to read channel-specific machine data:

```
DEF AXIS_POSITION_X =  (R///,""///"$AA_IM[X]")

CHANGE(AXIS_POSITION_X)
      DLGL("Axis """ << RNP("$MC_AXCONF_GEOAX_NAME_TAB[0]") << """
has moved: "
<< AXIS_POSITION_X)
END_CHANGE
```

Keep the channel-specific machine data up-to-date with the aid of an invisible variable, copy each value change to a temporary variable, e.g. register.

This temporary variable can then be used in the CHANGE method of the value change for the axis position without making a name resolution of the machine data each time and the subsequent read access:

```
DEF AXIS_POSITION_X =  (R///,""///"$AA_IM[X]")
DEF AXIS_NAME_X = (S///,""/WR0//"$MC_AXCONF_GEOAX_NAME_TAB[0]")

CHANGE(AXIS_NAME_X)
      REG[0] = AXIS_NAME_X
END_CHANGE

CHANGE(AXIS_POSITION_X1)
   DLGL("Axis """ << REG[0] << """ has moved " << AXIS_POSITION_X)
END_CHANGE
```

- The update rate and therefore the execution of the associated CHANGE method of variables that are connected to system or user variables with very frequent value changes, can be reduced by using the UR variable property, e.g. variable that is coupled to the actual axis values.
  Reason:
  In this way, the associated CHANGE method is executed in a fixed specified grid for a value change.

## B.4 Tips for DO LOOP loops

As loops can impair the performance of SINUMERIK Operate, depending on the configuration, they should be used carefully and if possible without time-intensive actions.

A register (REG[]) should also be used as run variable, because normal display variables (particularly those with OPI connection) also impair the system performance due to the frequent updates or write operations.

Note that the number of script lines executed "in one operation", is currently limited to 10,000. When this number is reached, the script execution is aborted with an appropriate entry in the "easyscreen_log.txt" file.

Reason:

- Prevention of endless loops
- "Run MyScreens" must remain operable

# Animated elements

C

## C.1 Introduction

### Introduction

The manual describes working with the X3D Viewer with which you can integrate animated and non-animated graphical scenes (animated elements) – called help screens in the following – into the graphical user interface of SINUMERIK Operate as of Version V4.7 SP1 .

You can show motion sequences and parameters in context-sensitive help screens. This helps you to improve the operation of applications and make the user interface more appealing.

Implementation of your initial idea to the final help screen is described in the steps below:

- Creation of graphical elements and 3D models for the help screens that will eventually appear in the X3D Viewer (see Chapter Modeling (Page 314)).
- Creation of the scene description file in which the model data of the graphics file will be assigned to the scenes and animations to be displayed at particular points (see Chapter Structure of the scene description file (Page 320)).
- Conversion of the X3D files and XML files into HMI files (see Chapter Conversion to hmi file (Page 324)).
- C++ integration of the X3D Viewer into your own application (see Chapter Implementation example (Page 326)).
- Application in Run MyScreens (see Chapter Display in Run MyScreens (Page 327)).

### Overview of the file formats

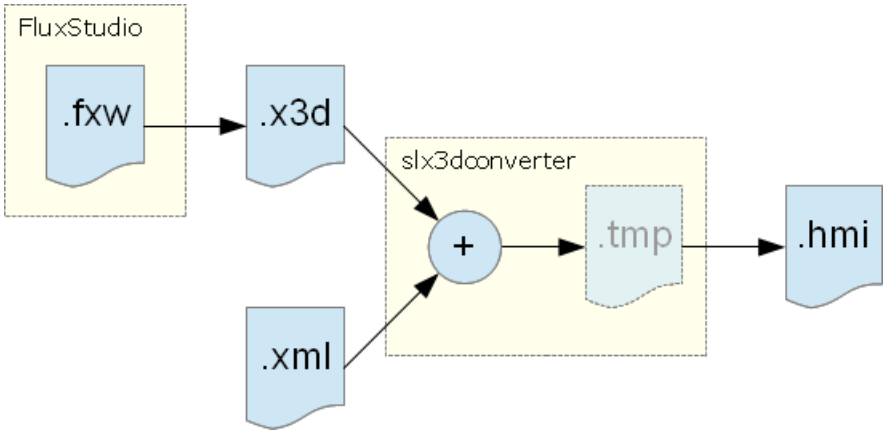



Figure C-1 File formats

| File format | Description |
|---|---|
| .fxw | Graphic file from Vivaty Studio |
| .x3d | Model file in exchange format |
| .xml | Definition file for animations and scenes |
| .hmi | Output file for the X3D Viewer |

# C.2 Modeling

## C.2.1 Requirements

The aim of the modeling is to create a file with the generated 3D models in .x3d file format, an official standard for 3D contents.

The modeling is done in **Vivaty Studio**. Vivaty Studio is a freely available 3D modelling tool with versatile export and import options.

### Requirements

- You installed Vivaty Studio, Version 1.0
- You are familiar with modeling and working with Vivaty Studio.
- File format .x3d is not described in any detail in this document, adequate knowledge of the subject is assumed.

**Note**

You can download Vivaty Studio on the Internet via the link (https://www.web3d.org/projects/vivaty-studio).

## C.2.2 Rules for modeling

Observe the following rules when modeling. You will only be able to process your help screen file if you have followed these rules correctly.

### Rules for modeling

1. The time sensor must have the name "Animation".
2. All related elements must be assigned to a group.
3. All groups must be set to the following position:
   x = 0, y = 0, z = 0

4. To make the groups invisible, the translation values are set as follows:
   x = 1000000, y = 1000000 and z = 1000000
5. The following elements must have the same names in the Vivaty Studio graphic models and in the XML scene definitions (cf. Chapter Overview (Page 319)):
   - Tool
   - Cameras
6. Before you can create a .x3d file you must make the following settings in dialog box "Export Options":

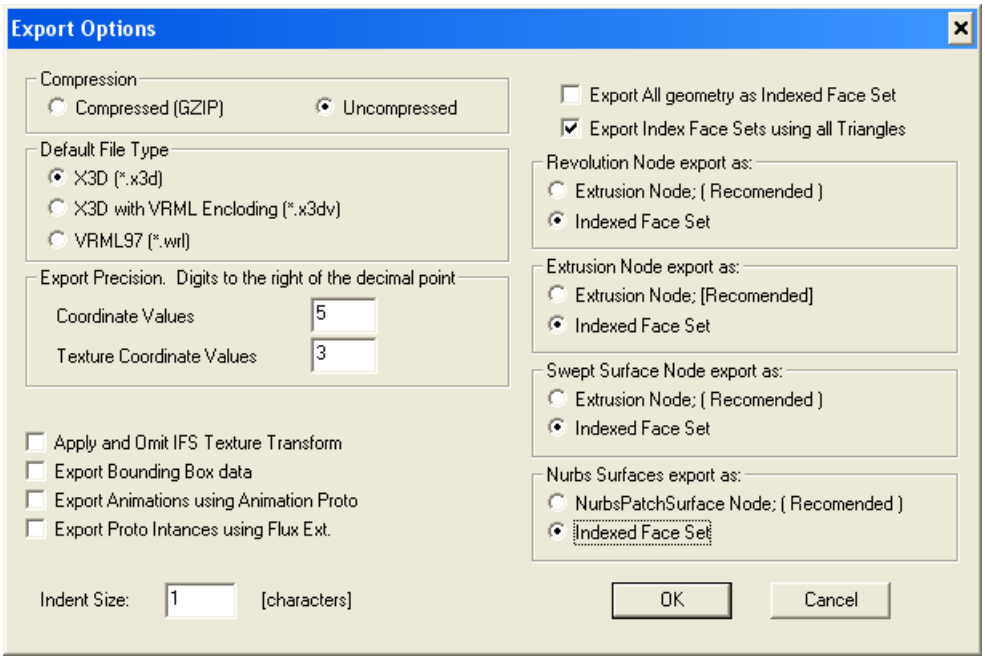



Figure C-2 Settings in the Export Options dialog box

7. We recommend the following settings for texts (see also the following dialog box):
   - Texts must always be centered.
   - The size of the text should be 0.2. This size allows you to position the text easily. The text output by the X3D Viewer is in the font size of the user interface, irrespective of this value.

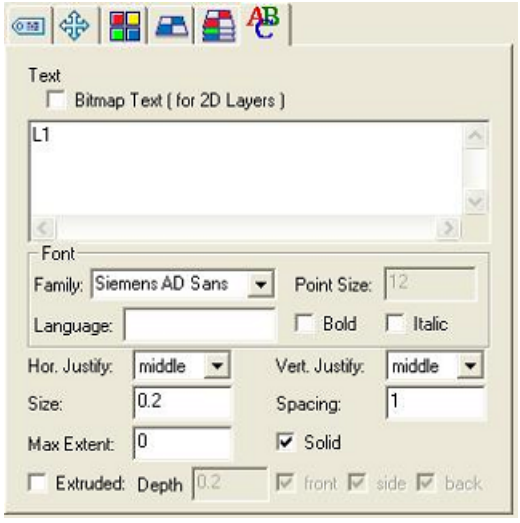



Figure C-3 Settings for text

8. To create hatching, use file schnitt.png as the texture. The lines always go from the bottom left to the top right at an angle of 45°. You should set the scaling to 3 in both dimensions. The rotation depends on the construction type and must be adapted manually.

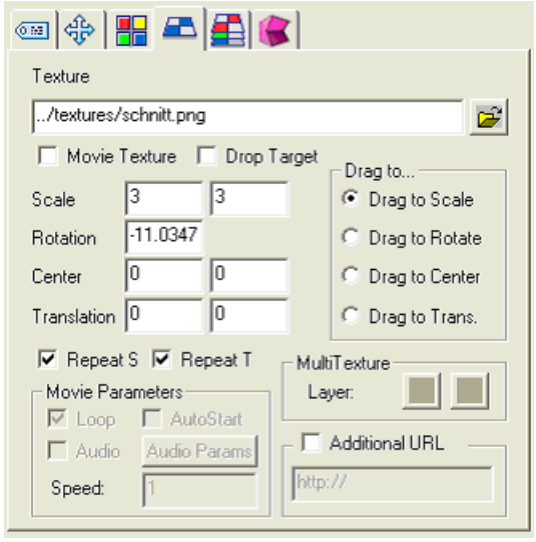



Figure C-4 Settings for hatching

9. We recommend the following sizes/dimensions for the blanks:
   - Cylinder for turning operation:
     Length = 5.5 ; Radius = 1.4 (cf. template turning_blank.fxw)
   - Cube for milling operation
     x = 4.75 ; y = 3 ; z = 3 (cf. template milling_blank.fxw)

### See also

XML commands (Page 319)

## C.2.3 Importing graphics (models)

In Flux Studio, you can import external graphics (models). You can store frequently used models, such as tools, as .x3d or .hmi files centrally for further use as finished elements in other help screens. You will find a list of supplied modeling templates in Chapter Modeling templates (Page 318).

Complex objects can also be created with other modeling tools, such as Cinema4D, and then imported.

A description of how you can reuse these models is given below.

### Import as inline

Flux Studio uses the object "Inline" for importing .x3d files. It allows you to link in external elements without multiplying the 3D data of these models.

Insert the object with menu item **Create** -> **Create Inline**. Then enter a file name in the object properties.

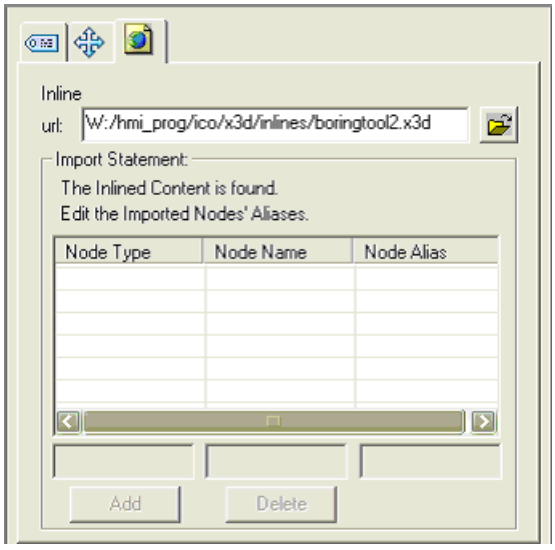



Figure C-5 Importing with object "Inline"

### Importing 3D model data

Proceed as follows to import model data from a file.

1. Open dialog box "Flux Studio Accutrans3D Translation Utility" via menu item **File** -> **Import Other Format** .
2. Select a format from the "File Types" list box. For example, to import a Cinema4D file, select file type "3D Studio."
3. Then select a file with **Select Files** and start import.

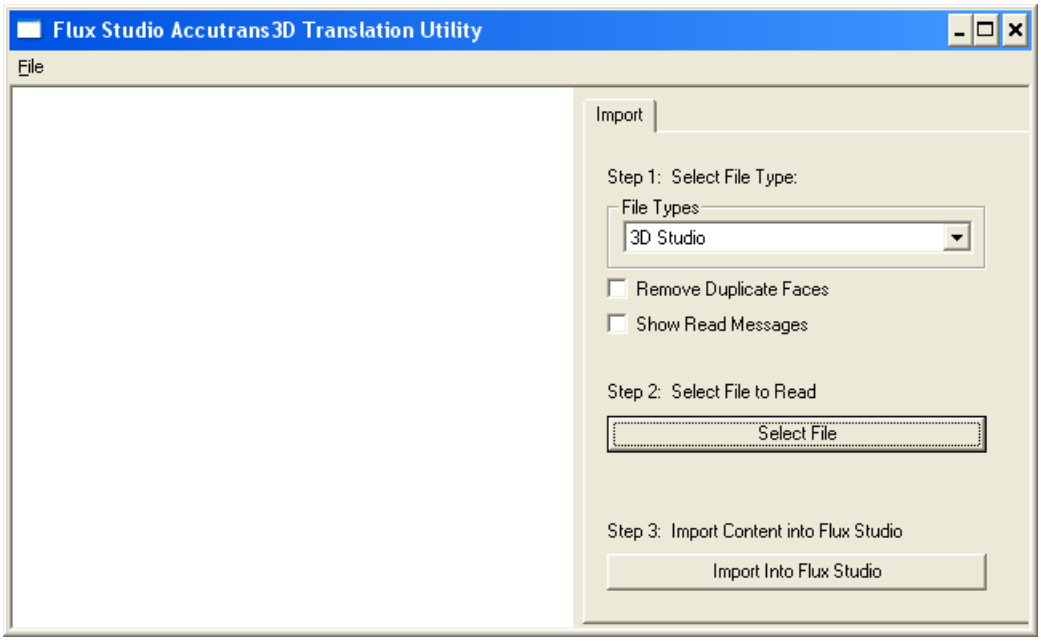



Figure C-6 Importing 3D model data

The model data from the Cinema4D file is inserted as a complete group. All cameras and any other information that is additionally imported should be deleted. Only the graphics elements of the Cinema4D file are required.

## C.2.4 Modeling templates

In this chapter, you will find a list of graphics elements on which you can base your design, colors, and sizes. To maintain a uniform "look and feel" in the user interface, we recommend using the style of the templates or the templates themselves for creating your own graphics.

### General

| Template | Description |
|---|---|
| intersection_texture.png | Texture for an sectioned area |
| rapid_traverse_line_hori.fxw | Horizontal line to illustrate the tool traversing path during rapid traverse |
| rapid_traverse_line_vert.fxw | Vertical line to illustrate the tool traversing path during rapid traverse |

| Template | Description |
|---|---|
| feed_traverse_line_hori.fxw | Horizontal line to illustrate the tool path during feed |
| feed_traverse_line_vert.fxw | Vertical line to illustrate the tool path during feed |
| dimensioning_lines_hori.fxw | Horizontal projection lines |
| dimensioning_lines_vert.fxw | Vertical projection lines |
| z1_inc.fxw | Horizontal dimension line with the identifier Z1 |
| x1_inc.fxw | Vertical dimension line with the identifier X1 |
| 3d_coordinate_origin.fxw | 3D coordinate origin |
| 3d_zero_point.fxw | 3D zero point |

### Turning

| Template | Description |
|---|---|
| turning_blank.fxw | The blank for turning technology: A simple cylinder |
| turning_centerline.fxw | Center line |
| turning_centerpoint.fxw | Coordinate origin to display the zero point in the WCS |
| turning_refpoint.fxw | Reference point, for example, for a machining operation |
| turning_machining_area.fxw | Boundary lines for the machining area |

### Milling

| Template | Description |
|---|---|
| milling_blank.fxw | The blank for milling technology: A simple cube |
| milling_centerline.fxw | Center line |
| milling_refpoint.fxw | Reference point, for example, for a machining operation |

## C.3 XML commands

### C.3.1 Overview

The scene description file (*.xml) is required so that the X3D Viewer can access the graphics. In the scene description file, you assign the scene names that are called from the configuration to the times in the *.x3d file (time sensor) at which the graphics are positioned.

## C.3.2 Structure of the scene description file

The structure of the scene description file is shown below together with an explanation of the relevant XML commands:

### Design and description

**<!-- Treatment of the plane-dependent texts for turning -->**

```
<TextPlane plane="G18_ZXY" />
```

**Description**

| | |
|---|---|
| <TextPlane plane="G18_ZXY" /> | Treatment of the plane-dependent texts (axis names) specifically for turning. If, for example, a text with the identifier "Z" is used (for G18 first geometry axis), the corresponding geometry axis identifier of the control will be shown in the help screen (for example, "Z"). |

**<!--Text alignment -->**

```
<TextPosition center='true' />
```

**Description**

| | |
|---|---|
| <TextPosition center='true' /> | Marking that the texts are centered. This is relevant to the display of the texts in rotated screens. We recommend using this setting. |

**<!--Adapting the tool speed -->**

```
<ToolSpeedFactors planeSpeedFactor="0.4" rapidSpeedFactor="0.7"
reducedSpeedFac¬tor="1.0"/>
```

**Description**

| | |
|---|---|
| <ToolSpeedFactors | This entry can be added if the tool speeds are to be modified. |
| planeSpeedFactor="0.4" | Factor for the feedrate ( 0.4 = 40% ) |
| rapidSpeedFactor="0.7" | Factor for rapid traverse ( 0.7 = 70% ) |
| reducedSpeedFactor="0.1" /> | Factor for a third speed ( 0.1 = 10% ) |

**<!--Definition of an animation-->**

```
<SceneKey name='BoringAnimation' masterRotationSpeed="-64.0"
maxRotAngle="45.0" begin-Time='0.110 endTime='0.118 view='camiso'
speedMaster='boringtool' Type="VIEW_3D_TURN_CYL">
```

**Description**

| | |
|---|---|
| <SceneKey name='BoringAnimation' | Name of the animation |
| masterRotationSpeed="-64.0" | Direction of rotation and rotation speed of the tool.<br>The setting is optional. |
| maxRotAngle="45.0" | To avoid aliasing effects (tool seems to be turning in the wrong direction). The value is usually a fourth of the symmetry of the tool.<br>The setting is optional. |
| beginTime='0.110 | Start time of the animation on the time sensor |
| endTime='0.118' | End time of the animation on the time sensor |
| view='camiso' | Camera to be used for the animation |
| speedMaster='boringtool' | Name of the tool for the animation |
| type="VIEW_3D_TURN_CYL"> | View type defines the view (see Section View type (Page 323)) |

**<!-- Elements -->**

```
<Element name='1' time='0.110' feed='rapid' dwell='2'/>
<Element name='2' time='0.112' feed='rapid'/>
<Element name='3' time='0.114' feed='rapid'/>
<Element name='4' time='0.116' feed='plane'/>
<Element name='5' time='0.118' feed='plane'/>
```

**Description**

| | |
|---|---|
| <Element name='1' | Specifies that it is the first element of the animation. |
| time='0.110' | Time of the first element on the time sensor |
| feed='rapid' | Traversing speed of the element<br>plane = machining feedrate<br>rapid = rapid traverse<br>reduced = reduced speed, slower than "plane" |
| dwell='2'/> | Pause in the traversing motion of the animation. A rotating tool continues to rotate. |

```
<!-- End of animation definition -->
</SceneKey>
```

**Description**

| | |
|---|---|
| </SceneKey> | End of the animation definition |

```
<!-- Existing animation as source for a new animation -->
<SceneKey source=''BoringAnimation' name=''BoringAnimationRear' mirror='
MirrorScreenX ' />
```

**Description**

| | |
|---|---|
| <SceneKey source="BoringAnimation' | Name of an animation that is to use another animation as its source. |
| name="BoringAnimationRear' | Name of the new animation based on the source |
| mirror=' MirrorScreenX ' /> | Mirroring in the direction of the X axis, output is in the new animation. |

```
<!-- Definition of a static scene (screen), (four examples) -->
<SceneKey name='Default' time='0.026' view='camiso'
type="VIEW_3D_DRILL_CUT"/>
<SceneKey name='Cut' time='0.046' view='camside' type="VIEW_SIDE"/>
<SceneKey name='Z1ink' time='0.056' view='camside' type="VIEW_SIDE"
 highLightedGroup='dad_Z1ink'/>
<SceneKey name='Z1abs' time='0.066' view='camside' type="VIEW_SIDE"
 highLightedGroup='dad_Z1abs'/>
```

**Description**

| | |
|---|---|
| <SceneKeyname='Z1abs' | Screen name |
| time='0.066' | Time of the screen on the time sensor |
| view='camside' | Camera to be used for the screen |
| type="VIEW_SIDE" | View type defines the view (see Section View type (Page 323)) |
| highLightedGroup='dad_Z1abs'/> | Name of the group to be highlighted. The group has been called "Z1abs" in Flux Studio. The prefix "dad_" is automatically added by Flux. |

```
<!-- End of xml file –!>
```

## C.3.3 Mirroring and rotations

Mirroring or rotation of the image or model is defined with the **mirror** command.

### Description

| | |
|---|---|
| mirror="RotateScreenX" | The screen is rotated around the X axis |
| mirror="RotateScreenY" | The screen is rotated around the Y axis |
| mirror="RotateScreenZ" | The screen is rotated around the Z axis |
| mirror="MirrorScreenX" | The screen is mirrored along the X axis |
| mirror="MirrorScreenY" | The screen is mirrored along the Y axis |
| mirror="MirrorScreenZ" | The screen is mirrored along the Z axis |
| mirror="RotatePieceX" | The model is rotated around the X axis |
| mirror="RotatePieceY" | The model is rotated around the Y axis |
| mirror="RotatePieceZ" | The model is rotated around the Z axis |
| mirror="MirrorPieceX" | The model is mirrored along the X axis |
| mirror="MirrorPieceY" | The model is mirrored along the Y axis |
| mirror="MirrorPieceZ" | The model is mirrored along the Z axis |

All rotate and mirror options can be combined. A rotation angle must be defined for rotation.

### Examples

mirror="RotatePieceZ=180"

mirror="RotatePieceZ=-90 MirrorScreenX"

mirror="RotateScreenY=90"

mirror="MirrorPieceX MirrorPieceY"

mirror="MirrorPieceZ RotatePieceX=-90"

## C.3.4 View type

You define the views with the view type. The views are based on empirical values and rules that produce a sensible representation of the objects.
This ensures that the representation of the objects matches the coordinate system settings made in the user interface (MD 52000 $MCS_DISP_COORDINATE_SYSTEM or MD 52001 $MCS_DISP_COORDINATE_SYSTEM_2).

### Description

| | |
|---|---|
| "VIEW_STATIC" | Direct view without conversion |
| "VIEW_3D_TURN_CYL" | 3D view for turning operations (cylinder) |
| "VIEW_3D_MILL_CUBE" | 3D view for milling operations (cube) |
| "VIEW_3D_DRILL_CUT" | 3D view for drilling operations (sectional view) |

| | |
|---|---|
| "VIEW_SIDE" | 2D view for drilling operations (sectional view) |
| "VIEW_TOP_GEO_AX_1" | 2D view from the direction of the 1st geometry axis |
| "VIEW_TOP_GEO_AX_2" | 2D view from the direction of the 2nd geometry axis |
| "VIEW_TOP_GEO_AX_3" | 2D view from the direction of the 3rd geometry axis |

## C.4 Conversion to hmi file

**Note**

The X3D files and their associated XML files are converted to HMI files when the HMI is powered up.

For each X3D file, a corresponding XML file of the same name must be created. To do this, you store the X3D files and XML files in directory `search path of HMI\ico\x3d\turning` or `milling`. You proceed in the same way with the .ts files.

## C.5 Display in Create MyHMI /3GL

### C.5.1 X3D Viewer

If you want to display particular help screens in a separate OA application you must integrate the X3D Viewer widget into its own OA application.

The X3D Viewer widget provides interfaces, which enable X3D contents to be presented in the HMI.

Class SlX3dViewerWidget is provided for displaying graphical scenes. The definition of the class can be found in the corresponding slx3dviewerwidget.h header file in the global GUI include directory `\hmi_prog\gui\include`.

### C.5.2 Class SlX3dViewerWidget

The class provides a flexible widget that autonomously displays the contents of a model file specified during runtime and, if required, runs the animation.

The interface of the class comprises the Constructor, Destructor, and two methods for controlling the graphical output.

As a direct derivative of the Qt class QWidget, a much more expansive interface is available, which will not be described in any further detail here, (for example, show(), hide() and resize(...). For more information, please refer to the Qt documentation).

## C.5.3 Public methods

### SlX3dViewerWidget ( QWidget* pParent = 0 )

Constructor of the X3D Viewer widget.

| Parameters | Meaning |
|---|---|
| pParent | The parameter is passed on to the constructor of the Qwidget. |

### ~SlX3dViewerWidget ( )

Destructor of the X3D Viewer widget.

| Parameters | Meaning |
|---|---|
| - | - |

## C.5.4 Public slots

### void viewSceneSlot ( const QString& rsFileName, const QString& rsScene, const QString& rsAnimationScene, int nChannel, int nPlane, SlStepTechnology nTechnology )

With the viewSceneSlot method, the X3D Viewer is instructed to load the rsScene static scene and the rsAnimationSzene animated scene from the rsFileName file and display them alternately.

The static scene and then the animated scene are repeatedly alternately displayed for a fixed time.

If no static scene is specified, the animation will be shown immediately; an animated scene may also not have been specified.

Channel number, plane and technology are used to rotate the scenes to the correct position (depending on the set machine coordinate system).

| Parameters | Meaning |
|---|---|
| rsFileName | Name of the file that contains the scenes to be displayed |
| rsScene | Name of the static scene. |
| rsAnimationScene | Name of the animated scene. |
| nChannel | Channel number |

| Parameters | Meaning |
|---|---|
| nPlane | Plane |
| nTechnology | Technology<br>The following constants are defined for the SlStep-Technology enumeration type:<br>• SL_STEP_NO_TECHNOLOGY<br>• SL_STEP_MILLING<br>• SL_STEP_TURNING<br>• SL_STEP_SURFACE_GRINDING<br>• SL_STEP_CIRCULAR_GRINDING |

**void viewSceneSlot ( const QString& rsFileName, const QString& rsScene, const QString& rsAnimationScene )**

A simplified form of the viewSceneSlot method with which the X3D Viewer can be instructed to load and display the rsScene static scene and/or the rsAnimationSzene animated scene from the rsFileName file.

| Parameters | Meaning |
|---|---|
| rsFileName | Name of the file that contains the scenes to be displayed. |
| rsScene | Name of the static scene. |
| rsAnimationScene | Name of the animated scene. |

## C.5.5 Libraries

In order to be able to apply the X3D Viewer in your own projects, the list of library dependencies must be extended by the 'slx3dviewer.lib' entry.

## C.5.6 Implementation example

You will find an implementation example in the package Create MyHMI/3GL under **\examples \GUIFrameWork\SlExGuiX3D**.

## C.5.7 Machine data

Display MD 9104 $MM_ANIMATION_TIME_DELAY

Time delay in seconds until the animation is started for help screens. The setting does not apply to help screens that are only animated.

The setting applies globally to all animations of SINUMERIK Operate.

### C.5.8 Notes about use

- The animation is interrupted if the X3D Viewer widget is hidden. If this happens, there is no need to choose a scene selection without animation.
- You should avoid frequent or multiple instantiation of the X3D Viewer widget as this reduces performance and memory capacity. The use (implementation) of an X3D Viewer Singleton is recommended in such application scenarios.
- The X3D Viewer can also be used in signal slot concepts.
- If an error occurs (for example, a file not found or unknown scene name), the X3D Viewer widget opens a message area. This area is automatically hidden again when another help screen is activated.

## C.6 Display in Run MyScreens

This section is intended for experienced "Run MyScreens" developers. Necessary background knowledge is provided in the associated documentation.

In addition to using screens in .bmp or .png format, animated help screens can also be displayed with the X3D Viewer.

To integrate help screens, use the Run MyScreens interface in the usual way.

To output images in .bmp or .png format, XG0 is entered in the definition of a dialog box in the Attributes section or the parameter is left empty. To integrate X3D help screens in a dialog, you must enter **XG1** in the attributes.

```
//M(MASK_F_DR_O1_MAIN/$85407////52,80/XG1)
```

The following parameters are required to activate the individual help screens; their meaning is described in Section 5.3 Public slots:

1. File name
2. StaticScene (optional)
3. AnimationScene (optional)
4. Technology (optional)
5. Plane (optional)

The parameters are combined into a character string in this order, separated by a comma.

```
Hlp = "File name,StaticScene,AnimationScene,Technology,Plane"
```

## Examples

- The default help screen can be set with the specified variable Hlp.
  In the following example, the "MyAnimation" animation is output from the "MyDlgHelp.hmi" file. No StaticScene is specified, i.e. no static scene is output. No specifications are made for the Technology and Plane, i. e. default values are used.
  ```
  Hlp = "MyDlgHelp.hmi,,MyAnimation"
  ```
- The hlp property can be set on a specific help screen for the individual parameters of the input screen.
  In the following example, the static "MyParam" scene and the "MyAnimation" animation are output from the "MyDlgHelp.hmi" file in the G17 plane. No specification is made for the Technology, i.e. a default value is used.
  ```
  VarMyParam.hlp = "MyDlgHelp.hmi,MyParam,MyAnimation,,G17"
  ```

# Index

## G



## H



## I



## L



## M

## N



## O



## P



## R



## S

## T

## U



## V



## W