SIEMENS

SINUMERIK

SINUMERIK 840D sl Easy Screen (BE2)

Programming Manual

Valid for

Control:
SINUMERIK 840D sl/840DE sl

Software:
CNC software V4.4
SINUMERIK Operate V4.4

09/2011
6FC5397-1DP40-2BA0

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
with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.

CAUTION
without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.

NOTICE
indicates that an unintended result or situation can occur if the corresponding information is not taken into account.

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 for the specific task, 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 adhered to. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of the 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
Industry Sector
Postfach 48 48
90026 NÜRNBERG
GERMANY

order number: 6FC5397-1DP40-0BA0
℗ 01/2011

Table of contents

Introduction 1

1.1 "Easy Screen" range of functions

Overview

"Easy Screen" is implemented with an interpreter and configuration files containing descriptions of the user interfaces.

"Easy Screen" 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.

The "Easy Screen" interpreter can be used to create user interfaces that display functional expansions designed by the machine manufacturer or user, or simply to implement your own layout on the HMI. Preconfigured user interfaces supplied by Siemens or the machine manufacturer can be modified or replaced.

Part programs, for example, can be edited on user interfaces created by users. Dialogs can be created directly on the control system.

Basic configuration

The "Easy Screen" function enables machine manufacturers to configure their own dialogs. Even with the basic configuration, it is possible to configure 5 screens in the operator menu tree or for customer-specific cycle dialogs.

Software option

To expand the number of dialogs, you require the following Software option:

"SINUMERIK Operate Runtime license OA Easy Screen"

Supplementary Conditions

The following conditions must be met:

- It is only possible to switch between dialogs within a single operating area.
- User, setting, and machine data are initialized on request.
- User variables may not have the same names as system or PLC variables.
- The dialogs activated by the PLC form a separate operating area (similar to measuring cycle screens).
- The cycle support (//C ...) is no longer supported by the software.

Tools

An additional graphics program is needed to produce graphics/display images.

Use

You can implement the following functions:

1. Display dialogs containing the following elements:
   - Softkeys
   - Variables
   - Texts and Help texts
   - Graphics and Help displays
2. Open dialogs by:
   - Pressing the (start) softkeys
   - Selection from the PLC
3. Restructure dialogs dynamically:
   - Edit and delete softkeys
   - Define and design variable fields
   - Insert, exchange and delete display texts (language-dependent or independent)
   - Insert, exchange and delete graphics
4. Initiate operations in response to the following actions:
   - Displaying dialogs
   - Input values (variables)
   - Select a softkey
   - Exiting dialogs
5. Data exchange between dialogs
6. Variables
   - Read (NC, PLC and user variables)
   - Write (NC, PLC and user variables)
   - Combine with mathematical, comparison or logic operators
7. Execute functions:
   - Subroutines
   - File functions
   - PI services
8. Apply protection levels according to user classes

1.2 Fundamentals of Configuration

Configuration files

The defining data for new user interfaces are 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.

Note

The description can also be explained using comments. A ";" is inserted as comment character before every explanation.

An ASCII editor (e.g. Notepad or the HMI editor) is used to create configuration files.

Note

If you create or edit the files, use an editor that supports UTF-8 coding.

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:

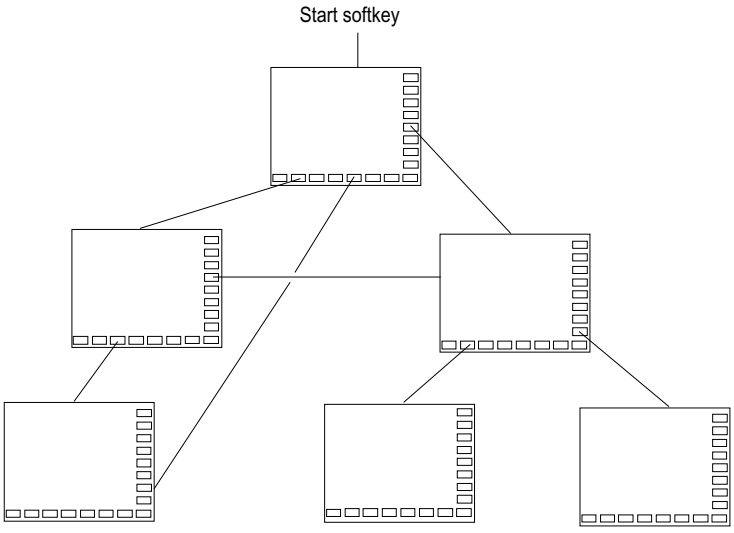


Figure 1-1 Menu tree

Start softkeys

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

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 user interfaces and return to the standard application.

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.

Creating a configuration file as ASCII file

Dialogs can contain, for example, the following elements:

- Input/output fields (variables) with
  - Short text
  - Graphic text
  - Text for units
- Screens
- Softkey menus
- Tables

1.3 Structure of configuration file

Overview

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

Other files:

In the event of "Load a screen form" (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.

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 (see above "Formatting text files"

The read-in mechanism via code page code is still supported. In order to activate this mechanism in the easyscreen.ini file, the following entry is required:

```
[Compatibility]

UseTextCodecs = true
```

Note

Constraint

In the supplied version of easyscreen.ini, this entry is not available, therefore, text files with UTF-8 coding are expected.

If the entry is supplemented, the old mechanism becomes effective again with the following restriction: It does not function correctly for Asian languages.

Storage location for configuration files

The configuration files are located on the CF card in the `/user/sinumerik/hmi/proj` directory and in the add_on and oem directories accordingly.

Structure of the configuration file

A configuration file consists of the following elements:

1. Description of the start softkeys
2. Definition of dialogs
3. Definition of variables
4. Description of the blocks
5. Definition of a softkey menu

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 variables
LOAD                                 ;   Description of the blocks
...
END_LOAD
UNLOAD
...
END_UNLOAD
...
//END
//S (...)                            ;   Definition of a softkey menu
//END
```

1.4 Troubleshooting (log book)

Overview

The log book is the easyscreen_log.txt file to which error messages generated by syntax interpretation are written.

The file easyscreen_log.txt is supplied in the following directory:

```
/user/sinumerik/hmi/log/
```

Example:

```
DEF VAR1 = (R)
DEF VAR2 = (R)
LOAD
VAR1 = VAR2 + 1           ; Error message in log book, as VAR2 has no value
```

Syntax

The system does not start to interpret 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)
//END
```

Contents of easyscreen_log.txt

If "Easy Screen" detects errors when interpreting the configuration files, these errors will be written to the easyscreen_log.txt ASCII file. The file will be deleted each time the HMI is restarted.

The file indicates:

- 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

How do I create a configuration? 2

2.1 Defining start softkeys

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=(...)                   ; defining 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 Easy Screen start softkeys are permissible in the operating areas:

Operating area	Position
Machine	HSK6
Parameter	HSK7
Program	HSK6 Measuring cycles: HSK13 and HSK14
Program manager	HSK2-8 and HSK12-16, if not assigned to drives.
Diagnostics	HSK7
Commissioning	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 for the Startup area). This does not apply to the machine operating area, where there are a number of files specific to operating modes (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 whereby 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.

Menus for standard applications are given the "easyscreenmode" menu property as part of their XML configuration. This indicates whether the menu involved permits the use of Easy Screen start softkeys (= easyscreen) or not (= off):

```
<SCREEN name="SlEasyScreenTest">
  <FORM ... >
      <PROPERTY ... > ... </PROPERTY>
  </FORM>
    <MENU name="menu_horiz" softkeybar="hu" easyscreenmode="easyscreen" />
    <MENU name="menu_vert" softkeybar="vr" easyscreenmode="off" />
</SCREEN>
```

Example

Separate start softkey menus can be defined for horizontal and vertical menus. The "MENU" attribute is used for this purpose.

If a new menu is displayed in an HMI application and this menu permits the use of start softkeys in accordance with the configuration (easyscreenmode = "easyscreen"), a search will first be performed for the "MENU" attribute in the configuration of the start softkey menu:

- If a configuration for a start softkey menu with the "MENU" attribute is found and if the "MENU" attribute contains the name of the menu that is currently being displayed (in the example: "menu_horiz"), then this start softkey menu is displayed.

  Only the horizontal softkeys are taken into account here, as the "menu_horiz" menu involves a horizontal menu bar.

- Where there is no menu-specific softkey menu for a particular menu (i.e. the "MENU" attribute is not available), the default start softkey menu will be loaded.

```
//S(Start)
  MENU="menu_horiz"
  HS2=("Contour",ac6,se3)
    PRESS(HS2)
      LS("Contour")
    END_PRESS
  ...
//END
```

Template for configurations

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

```
/card/siemens/sinumerik/hmi/cfg
```

This file is used as a template for your own configurations.

See also

Lists of start softkeys (Page 167)

2.1.1 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 form")             ;   horizontal SK 6 labeled "1st screen form"
PRESS(HS6)                          ;   PRESS method for horizontal SK 6
  LM("Screen form1")                ;   Load screen form1 function, where screen form
                                        1 must be defined within the same file.
END_PRESS                           ;   End of PRESS method
HS7=("2nd screen form")             ;   horizontal SK 7 labeled "2nd screen form"
PRESS(HS7)                          ;   PRESS method for horizontal SK 7
  LM("Screen form2")                ;   Load screen form2 function, where screen form
                                        2 must be defined 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
```

Configuration

The names of the files containing the configuration for the start softkey menus are given in the easyscreen.ini file. The entry can be specific to the operating area, the dialog, or the screen. For example:

```
StartFile01 =      area := SlGfwTest,
                   dialog := SlGfwTestDialog,
                   screen := SlEasyScreenTest,
                   startfile := test.com
StartFile02 =      area := AreaMachine,
                   dialog := SlMachine,
                   screen := Jog,
                   startfile := ma_jog.com
StartFile03 =      area := AreaMachine,
                   dialog := SlMachine,
                   screen := Auto,
                   startfile := ma_auto.com
StartFile04 =      area := AreaProgramManager,
                   dialog := ,
                   screen := ,
                   startfile := progman.com
StartFile05 =      area := AreaProgramEdit,
                   dialog := ,
                   screen := ,
                   startfile := aeditor.com
StartFile06 =      area := AreaStartup,
                   dialog := SlSuDialog,
                   screen := ,
                   startfile := test.com
```

The names given in the systemconfiguration.ini file should be used for "area" and "dialog". The screen identifiers in the dialog configuration should be used for "screen"; "startfile" refers to the file in which the start softkey menu (default or menu-specific) is configured.

If a menu-specific start softkey menu is required, an additional name is provided by the attribute "menu", e.g.:

```
StartFile01 =      area := SlGfwTest,
                   dialog := SlGfwTestDialog,
                   screen := SlEasyScreenTest,
                   menu := menu_horiz,
                   startfile := test.com
```

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 files with the .com extension in the following directories:

- /user/sinumerik/hmi/proj/
- /oem/sinumerik/hmi/proj/
- /addon/sinumerik/hmi/proj/
- /siemens/sinumerik/hmi/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 is always transferred into 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 contained is then subsequently displayed.

"System" parameter in the easyscreen.ini file

Dialogs can be displayed on different systems.

Default setting: System = 1

If dialogs are based on a value of 0, the value can be adapted by entering the following in easyscreen.ini:

[SYSTEM]

System = 0

2.2 Structure and elements of a dialog

2.2.1 Defining a dialog

Definition

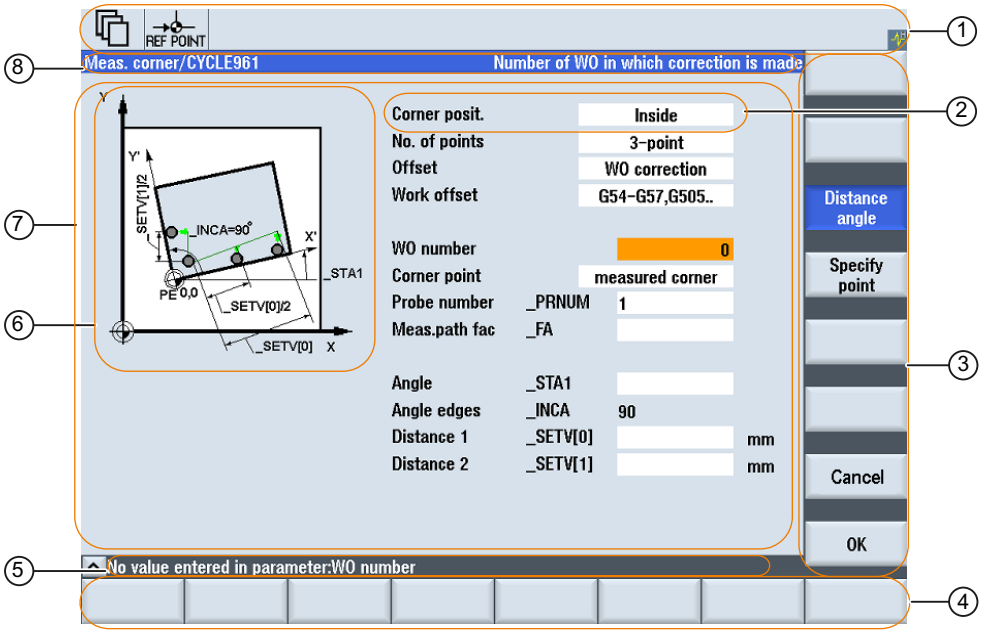
A dialog is part of a user interface consisting of a display line, dialog elements and/or graphics, an output line for messages and 8 horizontal and 8 vertical softkeys.

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 Displaying messages
⑥ Graphic
⑦ Dialog
⑧ Header line of the dialog with header and long text

Figure 2-1 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=...` `...`	;Variables	See chapter "Variables"
`HS1=(...)` `...`	;Softkeys	See chapter "Softkey menus"

`PRESS(HS1)`
`  LM...`
`END_PRESS`
	;Method start identifier
	;Actions
	;Method end identifier
		See chapter "Methods"

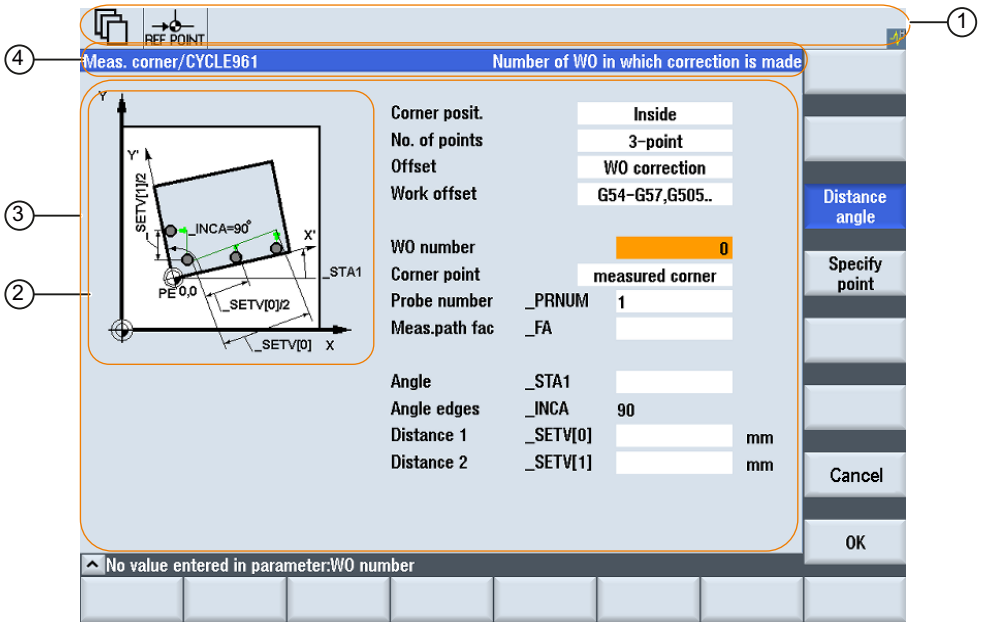
`//END`	;Dialog end identifier	

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.

2.2.2 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 2-2 Dialog properties

Programming

Syntax:	**//M**(Identifier/[Header]/[Graphic]/[Dimension]/[System or user variable]/[Graphic position]/[Attributes])	
Description:	Defines a dialog	
Parameters:	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 right-hand side, 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 the 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 right-hand side), in relation to the upper left-hand corner of the dialog. The minimum clearance from the top is 18 pixels. The entries are separated by a comma.
	Attributes	The specifications of the attributes are separated by a comma.
		Possible attributes are:
	CMx	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.
		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.

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

Example

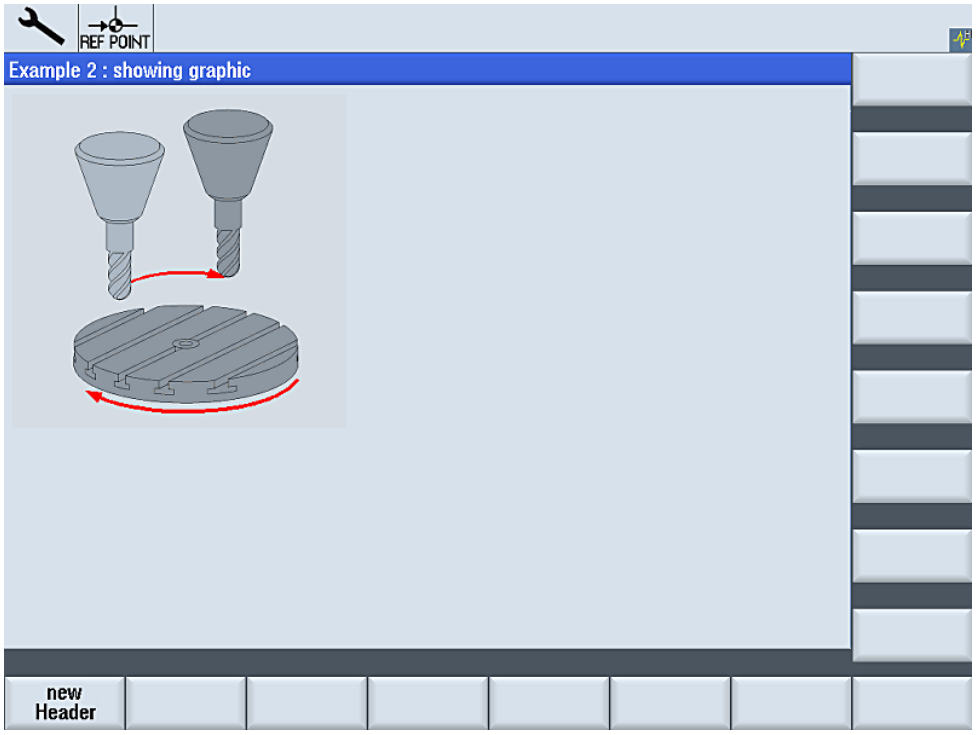


Figure 2-3 "Example 2: showing graphic"

```
//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 161)

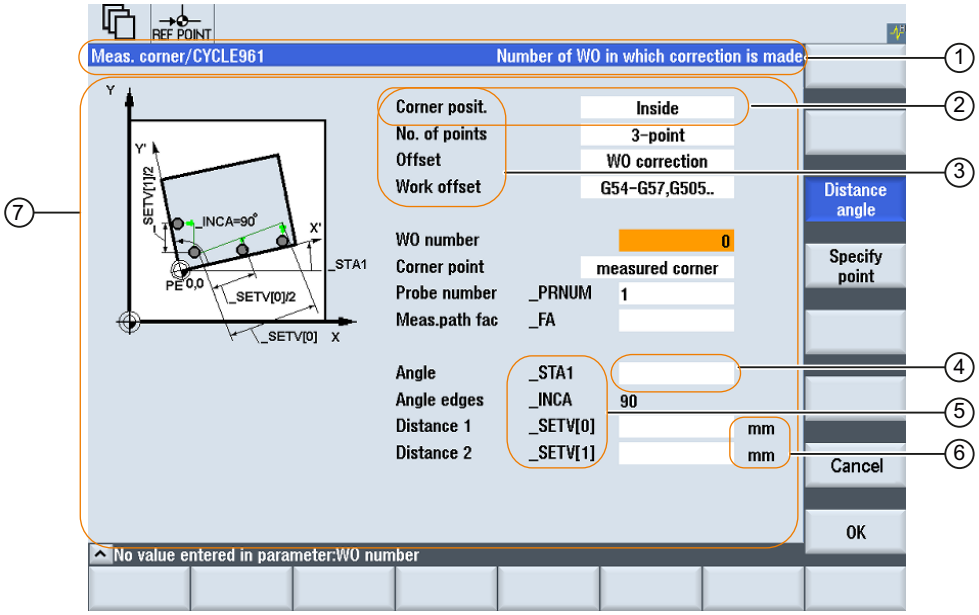
2.2.3 Defining dialog elements

Dialog element

The term "dialog element" refers to the visible part of a variable, i.e., short text, graphics text, input/output field and unit text. 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 block).



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 Text for units
⑦ Main body of the dialog

Figure 2-4 Elements of a dialog

Programming - Overview

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

DEF *Identifier* =	**Identifier = Name of variable**
	Variable type
	/[Limit values or toggle field]
	/[Default]
	/[Texts (Long text, Short text|Image, Graphic text, Units 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 41)

See also

Variable parameters (Page 52)

2.2.4 Example Opening the Dialog

Programming

The new "Example" dialog is called via the "Example" start softkey from the "Startup" operating area:

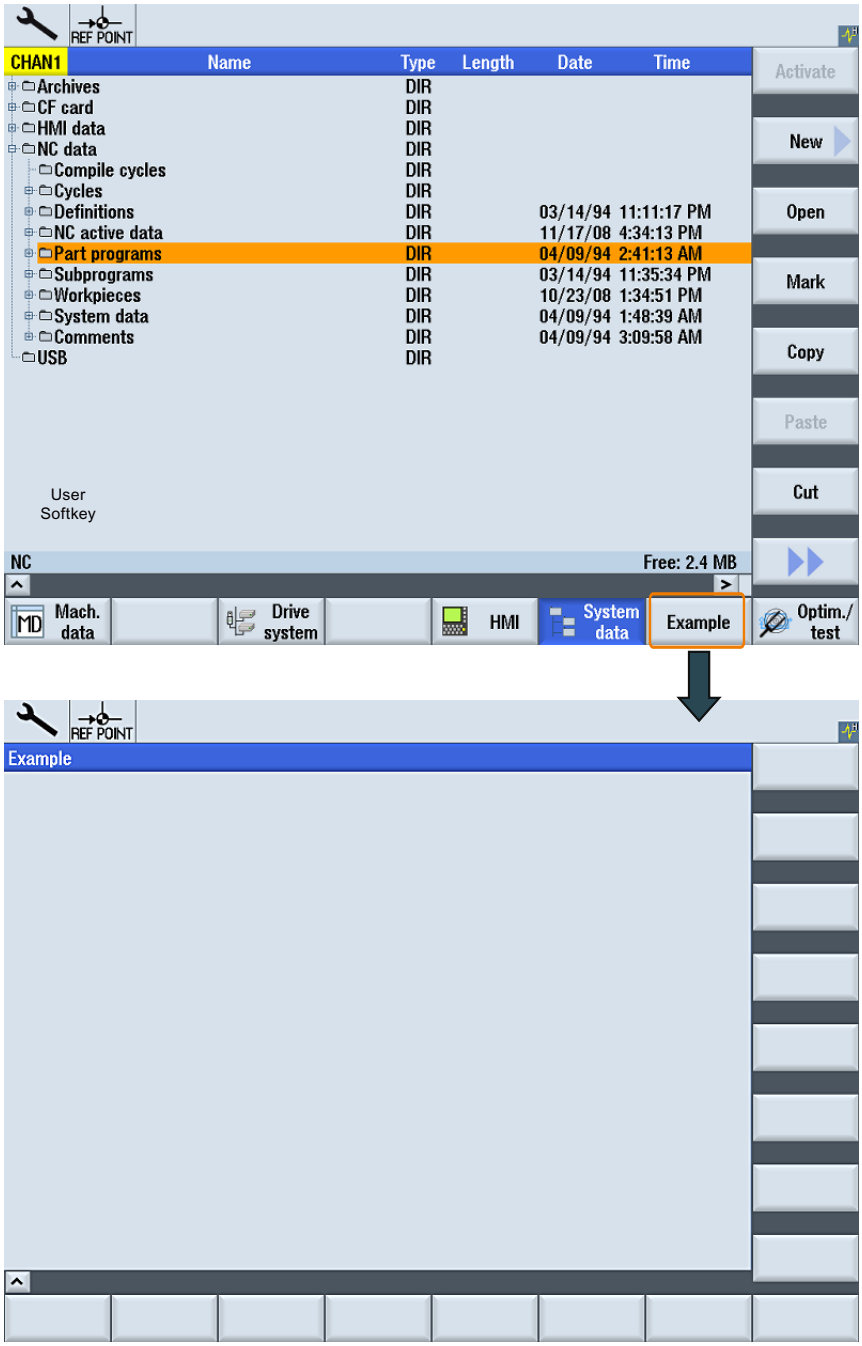


Figure 2-5 Example: Calling a new dialog

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

PRESS(HS7)
  LM("Maske1")
END_PRESS

//END
//M(Maske1/"Example")
HS1=("")
HS2=("")
HS3=("")
HS4=("")
HS5=("")
HS6=("")
HS7=("")
HS8=("")
VS1=("")
VS2=("")
VS3=("")
VS4=("")
VS5=("")
VS6=("")
VS7=("")
VS8=("")
...                                          ;  Methods
//END
```

2.2.5 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)

...
```

NOTICE
When creating dialogs with multiple columns, the options and limits of the hardware being used should be taken into consideration in terms of the number of columns and DEF instructions. A lot of columns can slow down the system.

2.2.6 Using display images/graphics

Use of graphics

There are two display categories:

- Display images/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

First, the relevant resolution directory is searched for the display image corresponding to the resolution of the connected monitor. If it is not found there, a search is performed for the display image in the next smaller resolution directory until – if the display image is not found earlier – directory ico640 is reached:

Search sequence:

```
/user/sinumerik/hmi/ico/ico<Resolution>
/oem/sinumerik/hmi/ico/ico<Resolution>
/addon/sinumerik/hmi/ico/ico<Resolution>
```

Note

Graphics are proportionally positioned for resolutions 640 x 480, 800 x 600 and 1024 x 768 pixels.

2.3 Defining softkey menus

Definition

The term softkey menu is used to refer to all the horizontal and vertical softkeys displayed on a screen form. 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	Chapter reference
`//S...`	;Start identifier of softkey menu	
`HSx=...`	;Define softkeys	

`PRESS(HSx)`
`  LM...`
`END_PRESS`
	;Method start identifier
	;Actions
	;Method end identifier
		See chapter "Methods"

`//END`	;End identifier of softkey menu	

Description

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

Programming

Syntax:	***//S****(Identifier)*	;Start identifier of softkey menu
	...	
	//END	;End identifier of softkey menu
Description:	Defines softkey menu	
Parameters:	Identifier	Name of softkey menu

Syntax:	**SK** = *(Text[, Access level][, Status])*	
Description:	Define softkey	
Parameters:	SK	Softkey, e.g. HS1 to HS8, VS1 to VS8
	Text	Enter text
	Display file name	"\\my_pic.png"
		or via separate text file $85199, e.g. with the following text in the (language-specific) text file: 85100 0 0 "\\my_pic.png".
		The size of image which can be displayed on a softkey depends on the OP used:
		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
	Access level	ac0 to ac7 (ac7: default)
	Status	se1: visible (default)
		se2: disabled (gray text)
		se3: displayed (last softkey used)

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 and lower access levels. The meanings of the different access levels are as follows: ac0 is the highest access level and ac7 the lowest.

Security level	Locked by	range
ac0	Password	Siemens
ac1	Password	Machine manufacturer
ac2	Password	Service
ac3	Password	User
ac4	Keylock switch position 3	Programmer, machine setter
ac5	Keylock switch position 2	Qualified operator
ac6	Keylock switch position 1	Trained operator
ac7	Keylock switch position 0	Semi-skilled operator

Example

```
//S(Menu1)                          ;   Start identifier of softkey menu
HS1=("NEW", ac6, se2)               ;   Define softkey HS1, assign the label "NEW",
                                        protection level 6, and the status "disabled"
HS2=("\\image1.png")                ;   Assign a graphic to the softkey
HS3=("Exit")


VS1=("sub screen form")
VS2=($85011, ac7, se2)              ;   Define softkey VS2, assign the text from the
                                        language file, protection level 1, and the
                                        status "disabled".
VS3=("Cancel", ac1, se3)            ;   Define softkey VS3, assign the label "Cancel",
                                        protection level 1 and the status
                                        "highlighted".
VS4=("OK", ac6, se1)                ;   Define softkey VS4, assign the label "OK",
                                        protection 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           ;   Define softkey VS7, assign an image, assign the
text"],,se1)                            label "OEM Text" and the status "visible"
VS8=(["\\image1.png",               ;   Define softkey VS8, assign an image, assign
$83533],,se1)                           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 delimiter

PRESS(RECALL)                   ;   Method start identifier
  LM("Screen form21")           ;   Load dialog
END_PRESS                       ;   Method end delimiter
//END                           ;   Softkey menu end identifier
```

2.3.1 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"	;Softkey with label
	SK.ac = Access level	;Softkey with security level
	SK.se = Status	; Softkey with status
Description:	Assign properties	
Parameters:	Text	Label text in inverted commas
	Access level	Range of values: 0 ... 7
	Status	1:visible and operator-controllable
		2:disabled (gray text)
		3:displayed (last softkey used)

Example

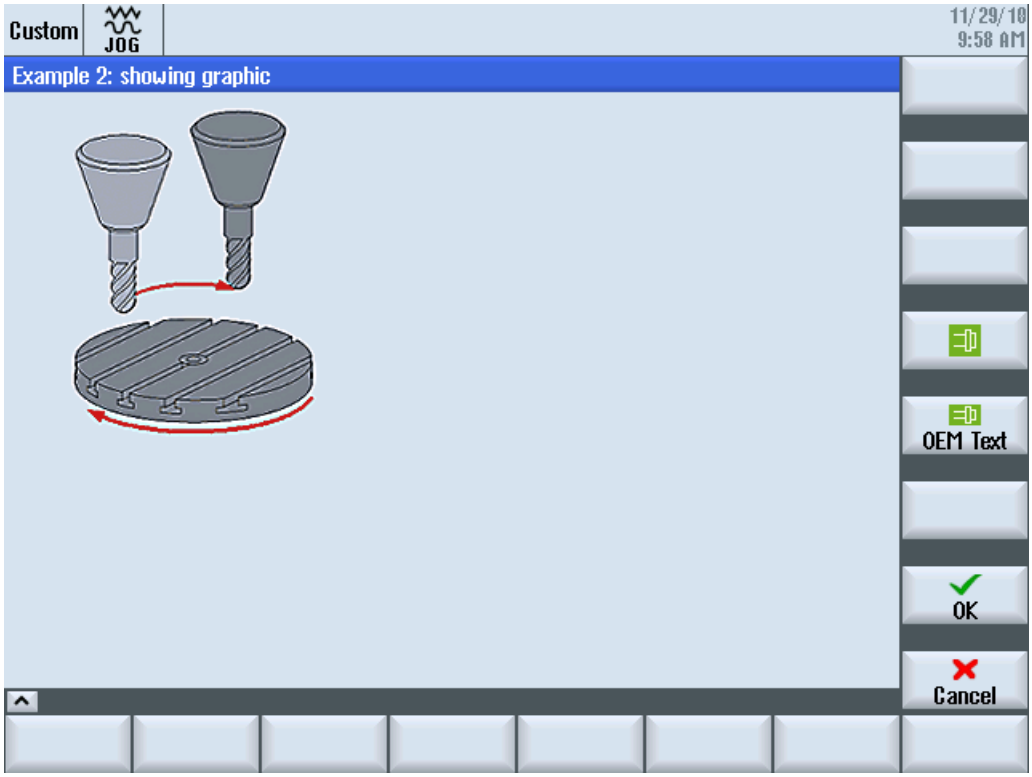


Figure 2-6 Example 3: Graphics and softkeys

```
//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
```

2.3.2 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:

- /user/sinumerik/hmi/lng/
- /oem/sinumerik/hmi/lng/
- /addon/sinumerik/hmi/lng/

alsc.txt	Contains the language-dependent texts for the Siemens standard cycles
almc.txt	Contains the language-dependent texts for the manufacturer cycles
aluc.txt	Language-dependent user texts

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

```
[LANGUAGEFILES]

LngFile01 = alsc.txt ;->alsc<_xxx>.txt (e.g. alsc_eng.txt)

LngFile02 = user.txt
```

In this instance, the user.txt file has been chosen as an example of a text file. Any name can be selected, in principle. Depending on the language of the texts within the file, the relevant language code must be added using the following syntax:

```
user.txt → user_xxx.txt (e.g. user_eng.txt)
```

An underscore followed by the relevant language identifier are added after the name.

See also

List of language codes used in file names (Page 172)

Format of text files

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

If, for example, you use Notepad to generate text files, select "File" → "Save As" and choose **UTF-8** encoding.

Format of a text entry

Syntax:	8xxxx 0 0 "Text"		
Description:	Assignment between text number and text in the file		
Parameters:	xxxx	5000 to 9899	Text identification number range reserved for users. You must assign unique numbers.
	"text"		Text that appears in 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, these two parameters must always be set to zero.

Examples of alarms:

```
85000 0 0 "Retraction plane"

85001 0 0 "Drilling depth"

85002 0 0 "Pitch"

85003 0 0 "Pocket radius"
```

2.4 Configuring the online help

Online help

An online help for the configured dialogs and elements can be created in the HTML format. The syntax and procedure for the online help is essentially the same as for SINUMERIK Operate, e.g.:

```
DEF VAR14 = (I///,"\\ein.png"/al1,cb1,wr2//"DB1.DBB0"////"sinumerik_md_1.html",
"9100")
```

References

Commissioning Manual "Base software and operating software" (IM9), Chapter "OEM-specific online help"

See also

Defining dialog elements (Page 27)

Variables

3

3.1 Defining variables

Variable value

The most basic 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	
	Identifier = Variable value	
Description:	Variable value val (value)	
Parameters:	Identifier:	Name of variable
	Variable value:	Value of variable
Example:	`VAR3 = VAR4 + SIN(VAR5)` `VAR3.VAL = VAR4 + SIN(VAR5)`	

Variable status

The "Variable status" property can be used to scan 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)	
Parameters:	Identifier:	Name of variable
		The result of the scan can be:
		FALSE =invalid value
		TRUE =valid value

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

Variable: Changing properties

The variables are assigned a new value in the notation *Identifier.Property = Value* when a change is made. The expression to the right of the equality sign is evaluated and assigned to the variable or variable property.

Examples:

Identifier.**ac** = Access level	(ac: access level)
Identifier.**al** = Text alignment	(al: alignment)
Identifier.**bc** = Background color	(bc: back color)
Identifier.**fc** = Foreground color	(fc: front color)
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.**st** = short text	(st: short text)
Identifier .**typ** = Variable type	(typ: type)
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)

3.2 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 Help variables

Example

```
DEF OTTO  ;Definition of a Help variable
```

Syntax:	Identifier.val = *Help variable value*	
	Identifier = *Help variable value*	
Description:	A value is assigned to a Help variable in a method.	
Parameters:	Identifier:	Name of Help variables
	Help variable value:	Content of the Help variables

Example:

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

Calculation with variables

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

You can scan the variable status to ascertain the validity of the value of the variable, e.g.,

```
Var1 = Var5 + SIN(Var2)

Otto = PI * Var4
```

Addressing system variables indirectly

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

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

Changing softkey labels

Example

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

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

Example 1a

Assigning the Variable type, Texts, Help display, and Colors properties

DEF Var1 = (R///,"Actual value",,"mm"//"Var1.png"////8,2)

Variable Type:	REAL
Limits or entry in the toggle field:	none
Default setting:	None
Texts:	
Long text:	None
Short text:	Actual value
Graphic text:	none
Unit text:	mm
Attributes:	None
Help display:	Var1.png
System or user variable:	None
Position of short text:	No data, i.e., default position
Position of input/output field:	No data, i.e., default position
Colors:	
Foreground color:	8
Background color:	2

Example 1b

Assigning tooltips

DEF Var2 = (I//5/"","value","","," Tooltiptext"/wr2///20,250,50)

Variable type:	INTEGER
Limits or entry in the toggle field:	None
Default setting:	5
Texts:	
Short text:	Value (possible language text ID)
Tooltip:	Tooltip text
Attributes:	
Input mode	Reading and writing
Help display:	None
Position of short text:	
Distance from left	20
Distance from top	250
Width:	50
Colors:	No data, i.e. default

See also

Variable parameters (Page 52)

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

Example 2

Assigning the Variable Type, Limits, Attributes, Short Text Position properties

DEF Var2 = (I/0,10///wr1,al1///,,300)

Variable Type:	INTEGER
Limits or toggle field entries:	MIN: 0
	MAX: 10
Default:	none
Texts:	none
Attributes:	
Input mode	read-only
Alignment of short text	Right-justified
Help display:	none
System or user variable:	none
Position of short text:	
Distance from left	None
Distance from top	None, i.e., default distance from top left
Width:	300
Position of input/output field:	No data, i.e., default position
Colors:	No data, i.e., default
Help:	none

See also

Variable parameters (Page 52)

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

Example 3

Assigning the Variable Type, Default, System or User Variable, Input/Output Field Position properties

DEF Var3 = (R//10///"$R[1]"//300,10,200//)

Variable Type:	REAL
Limits or toggle field entries:	none
Default setting:	10
Texts:	none
Attributes:	None
Help display:	none
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
Colors:	No data, i.e. default

See also

Variable parameters (Page 52)

3.6 Examples relating to toggle field and image display

Example 4

Various entries in the toggle field:

Limits or toggle field entries:

```
DEF Var1 = (I/* 0,1,2,3)
DEF Var2 = (S/* "In", "Out")
DEF Var3 = (B/* 1="In", 0="Out")  ;1 and 0 are values, "In" and "Out" are displayed.
DEF Var4 = (R/* ARR1)             ;ARR1 is the name of an array.
```

Example 5

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

DEF VAR6= (V///,"\\image1.png" ////160,40,50,50)

Variable type:	VARIANT
Limits or entries in the toggle field:	None
Default setting:	None
Texts:	
Short text:	image1.png
Attributes:	None
Help display:	none
System or user variable:	None
Position of short text:	
Distance from left:	160
Distance from the top:	40
Width:	50
Height:	50
Position of input/output field:	No details
Colors:	No data, i.e. default

3.7 Variable parameters

Parameter overview

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

Parameter	Description	
Variable type (Page 55)	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	
Limits (Page 49)	Limit value MIN, limit value MAX Default setting: Empty 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".	
Default setting (Page 61)	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". Default setting: No default	
Toggle field (Page 59)	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. 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. Default setting: None	
Texts (Page 47)	The sequence is specified. Instead of a short text, an image can also be displayed. Default setting: Empty	
	Long text: Short text: Graphic text: Unit text:	Text in the display line Name of the dialog element Text refers to the terms in the graphics Unit of the dialog element
	Tooltips (Page 47)	Serve as brief information in a screen form configuration for the display and toggle fields. The information is configured via plain text and language text ID.

Parameter	Description	
Attributes (Page 49)	The attributes influence the following properties: • Input mode • Access level • Alignment of short text • Font size • Limits • Response when dialog is opened in terms of CHANGE block The attributes are separated by commas and appear in any order. The attributes are not valid for toggle fields. A definition can be made for each component.	
	Input mode	wr0: IO field invisible, short text visible wr1: Read (no focus possible for input) wr2: Read and write (line appears in white) wr3: wr1 with focus wr4: All variable elements invisible, no focus possible 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). Default setting: wr2
	Access level	Empty: Can always be written ac0...ac7: Protection levels 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 al1: Right-justified al2: centered Default setting: al0
	Font size	fs1: Default font size (8 pt.) fs2: Double font size Default setting: fs1 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. Default setting: Determined by specified limits li0: No check li1: Check with respect to min. li2: Check with respect to max. 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.

Parameter	Description	
	cb0:	The CHANGE block defined for this variable is edited when the dialog is opened (default setting). Multiple attributes are separated by commas.
	cb1:	The CHANGE block defined for this variable is then only processed if the value of the variable changes.
Help display (Page 47)	Help display file:	Name of the png file 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 50)	System or user data from the NC/PLC can be assigned to the variable. The system or user variable appears in double quotation marks. Reference: List Manual System Variables, /PGAsl/	
Position of short text (Page 62)	Position of short text (distance from left, distance from top, width) 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 62)	Position of input/output field (distance from left, distance from top, width, height) 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 47)	Foreground color, background color: The colors are separated by a comma. Color settings are only relevant to the input/output field; colors cannot be specified for the other texts. Range of values: 1...10 Default setting: Foreground color: Black, background color: white The default colors of the input/output field are determined by the Write mode: "wr" indicates write mode.	

3.8 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

Display format	
B	Binary
D	Decimal signed
H	hexadecimal
No data	Decimal signed

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

Memory utilization	
B	Byte
W	Word
D	Double Word
BU	Byte, Unsigned
WU	Word, Unsigned
DU	Double word, Unsigned

Sequence of characters in the INTEGER data type

1. "I" Basic INTEGER designation
2. Display 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

VARIANT variable type

The VARIANT variable type is determined by the data type of the last value assignment. It can be scanned using the ISNUM or ISSTR functions. 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
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"
```

Numerical 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 52)

3.9 Details on the toggle field

Description

The toggle field extension function can be used to display 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.

Programming

Syntax: **DEF identifier =(variable type /+ $text number | * value="\\image"[,value="\\image2.png"][, ...]**
/[Default]
/[Texts(Long text, Short text, Graphic text, Units text)]
/[Attributes]
/[Help display]
/[System or user variable]
/[Position of short text]
/[Position input/output field(Left, Top, Width, Height)]
/[Colors]

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.

Parameters:

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.

Example:
```
DEF VAR1=(IB/+ $85000/15////"DB90.DBB5")
```

Variable toggle field

It is possible to assign a variable toggle field to a dialog element, i.e., when the toggle key is pressed, a value configured in a CHANGE method is assigned to the variable.

An asterisk * is entered in the Limits or Toggle Field property to identify a variable toggle field when a variable is defined.

Example: `DEF VAR1=(S/*)`

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[0]"//160,40,50,50)
```

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

See also

Variable parameters (Page 52)

3.10 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, has been assigned to the variable field (I/O field or toggle field). (Not calculated: Toggling is not possible until a valid value is assigned to the variable).

Scope of the default settings

If...			Then...
Field type	Default setting	System or user variable	Reaction of field type
I/O field	yes	yes	Write default value to system or user variable
	No	yes	Use system or user variable as default value
	Error	yes	Not calculated, system or user variable is not written into/used.
	yes	No	Default setting
	No	No	Not calculated
	Error	No	Not calculated
	yes	Error	Not calculated
	No	Error	Not calculated
	Error	Error	Not calculated
Toggle	yes	yes	Write default value to system or user variable
	No	yes	Use system or user variable as default value
	Error	yes	Not calculated, system or user variable not written/used
	yes	No	Default setting
	No	No	Default = first toggle field element
	Error	No	Not calculated
	yes	Error	Not calculated
	No	Error	Not calculated
	Error	Error	Not calculated

See also

Variable parameters (Page 52)

3.11 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 settings 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.

If no positions have been specified for any dialog elements, the default setting is applied. By default, the column width for the short text and input/output field is calculated for each line based on 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.

See also

Variable parameters (Page 52)

3.12 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 130)

3.13 CURPOS variable

Description

Using the CURPOS variable, it is possible to display or manipulate the position of the cursor in the active input field of the current dialog. The variable indicates how many characters are located 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 block. If the value of CURPOS changes, then a jump is made to the CHANGE block and the instructions contained there are executed.

3.14 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
                                    automatically assigned the version of the
                                    recompiled code.
```

3.15 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 =Programming support (the dialog was called by programming support)
- 2 =Programming support + default setting from the previous dialog (sub-dialog)
- 3 =Programming support + recompilation
- 4 =Programming support + recompilation with generated comments, with # sign
- 5 =Programming support + recompilation with generated comments, without # sign

Example

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

3.16 ERR variable

Description

Variable ERR can be used to check whether the preceding lines have 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                     ;   Scan to check whether value has been found
                                       in array
  VAR5 = "Error accessing array"
                                   ;   If the value has not been found in the
                                       array, the value "Error accessing array" is
                                       assigned to the variables.
ELSE
  VAR5 = "All OK"                  ;   ;If the value has been found in the array,
                                       the value "All OK" is assigned to the
                                       variables.
ENDIF
```

3.17 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:\target.mpf
IF FILE_ERR > 0                     ;  Scan to ascertain whether error has occurred
   IF FILE_ERR == 1                 ;  Scan 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 = "Wrong file name"
         ENDIF
      ENDIF
   ENDIF
ELSE
   VAR5 = "All OK"                  ;  If no errors have occurred in CP (or GC),
                                       "All OK" is output
ENDIF
```

3.18 FOC variable

Description

The FOC variable can be used to control the input focus (the current active input/output field) in a dialog. Responses to cursor left, right, up and down movements, as well as PGUP, PGDN, are predefined and cannot be modified.

Note

The FOC function may not be initiated as a result of a navigation event. The cursor position may only be changed in softkey PRESS blocks, CHANGE blocks, 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 will be assigned to Variable 1.
FOC = 3                         ;  The input focus will be assigned to the 3rd
                                   dialog element with WR ≥ 2.
```

3.19 S_CHAN variable

Description

The S_CHAN variable can be used to determine the number of the current channel for display or evaluation purposes.

Programming commands

4

4.1 Operators

Overview

The following operators can be used when programming:

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

4.1.1 Mathematical operators

Overview

Mathematical operators	Identifier
+	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	Identifier
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

Note

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

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

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 a command:

```
IF
...
ENDIF
```

Condition with two commands:

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

4.1.2 Bit operators

Overview

Bit operators	Identifier
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 left by number of bits
                                     specified in VAR04. Then convert 32-bit value
                                     back to format of variable VAR02.

END_PRESS
```

4.2 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	Chapter reference
`PRESS(HS1)`	;Method start identifier	
`LM...` `  LS...`	;Functions	See chapter "Functions"
`Var1.st = ...`	;Changing properties	see chapter "Softkey menu" and chapter "Dialog elements"

`Var2 = Var3 + Var4`
`...`
`EXIT`
	;Calculation with variables	See chapter "Defining variables"

`END_PRESS`	;Method end identifier	

4.2.1 CHANGE

Description

CHANGE methods are executed if a variable value changes, i.e., 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)* ... 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() ... END_CHANGE
Description:	Changes any variable value
Parameters:	- None -

Example

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

4.2.2 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 may only be moved in softkey PRESS blocks, CHANGE blocks, etc. Responses to cursor movements are are predefined and cannot be modified.

Note

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

Programming

Syntax:	FOCUS ... END_FOCUS
Description:	Positions the cursor
Parameters:	- None -

Example

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

4.2.3 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 ... 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

Line and rectangle (Page 137)

4.2.4 LOAD GRID

Description

The table description can be made available dynamically within the LOAD block 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

4.2.5 UNLOAD

Description

The UNLOAD method is executed before a dialog is unloaded.

Programming

Syntax:	UNLOAD ... END_UNLOAD
Description:	Unload
Parameters:	- None -

Example

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

4.2.6 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)* ... END_OUTPUT
Description:	Outputs variables in the NC program.
Parameters:	Identifier Name of OUTPUT method

Block numbers and skip identifiers

The OUTPUT block must not contain line numbers or skip identifiers if you wish to keep the line numbers and hide markings 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
```

4.2.7 PRESS

Description

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

Programming

Syntax:	PRESS*(softkey)*		
	...		
	END_PRESS		
Identifiers:	Pressing a softkey		
Parameters:	Softkey	Name of softkey: HS1 - HS8 and VS1 - VS8	
	RECALL	<RECALL> key	
	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
```

4.2.8 Example Version management with OUTPUT blocks

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")     ;  Expansion with effect from Version 1
```

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

Example

```
OUTPUT(NC1)                    ;  Only the variables of the original version are
                                  made available in the OUTPUT block.
OUTPUT(NC1,1)                  ;  The variables of the original version and the
                                  expansions with version identifier 1 are made
                                  available in the OUTPUT block
```

The OUTPUT block for the original version 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 block 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
```

4.3 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.

Subroutines

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_SERVICE function can be used to start PI Services (Program Invocation Services) from the PLC in the NC area.

See also

Function (FCT) (Page 104)

PI services (Page 134)

4.3.1 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*) **SUB***(Identifier)* END_SUB [SUB(*Identifier*) ... END_SUB] ... //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
```

4.3.2 Subprogram call (CALL)

Description

The CALL function can be used to call 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:	Subroutine call	
Parameters:	Identifier	Name of subprogram

Example

```
//M(SCREEN FORM1)
VAR1 = ...
VAR2 = ...
LOAD
  ...
  LB("PROG1")                          ;   Load block
  ...
END_LOAD
CHANGE()
  ...
  CALL("UP1")                          ;   Call subroutine and execute
  ...
END_CHANGE
...
//END
```

4.3.3 Check Variable (CVAR)

Description

You can use the CVAR (CheckVariable) function to run a scan to ascertain whether all or only certain variables or Help variables in a screen form 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. Up to 29 variables, each separated by a comma, can be checked. A character length of 500 must not be exceeded. 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, softkey 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 error-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 invalid, "VAR1
                                   and VAR2 are not OK" appears in the dialog line
ENDIF
```

4.3.4 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 can be used to query whether the function was successful:

```
CP("\MPF.DIR\CFI.MPF","\WKS.DIR\123.WPD\CFI.MPF",VAR1)
```

Example

Application with return value:

```
CP("//NC/MPF.DIR/HOHO.MPF","//NC/MPF.DIR/ASLAN.MPF",VAR3)
CP("//NC/MPF.DIR/hoho.MPF",VAR0,VAR3)
CP(VAR4,VAR0,VAR3)
CP("CF_CARD:/mpf.dir/myprog.mdf","//NC/MPF.DIR/HOHO.MPF",VAR3)
CP("//NC/MPF.DIR/HOHO.MPF",                          ; xyz must exist
"CF_CARD:/xyz/123.pmf",VAR3)
```

Application without return value:

```
CP("//NC/MPF.DIR/HOHO.MPF","//NC/MPF.DIR/ASLAN.MPF")
CP("//NC/MPF.DIR/hoho.MPF",VAR0)
CP(VAR4,VAR0)
CP("CF_CARD:/mpf.dir/myprog.mdf","//NC/MPF.DIR/HOHO.MPF")
CP("//NC/MPF.DIR/HOHO.MPF",                          ; xyz must exist
"CF_CARD:/xyz/123.mpf")
```

See also

Support of FILE_ERR: FILE_ERR variable (Page 69)

4.3.5 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/XYZ.DIR ", VAR1)
```

  VAR1 = 0 File was deleted.
  VAR1 = 1 File was not deleted.

- Without return value:

```
DP("//NC/MPF.DIR/XYZ.DIR ")
```

```
DP("\MPF.DIR\CFI.MPF")
```

4.3.6 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 result of the scan can be:	
	• M = File is stored on HMI	
	• N = file is stored on NC	
	• Blank string = The file neither exists on the HMI nor on the NC	

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.DIR
```

or

```
CF_CARD: /MPF.DIR/XYZ.DIR
```

or

```
LOC: /MPF.DIR/XYZ.DIR
```

New syntax:

```
EP("//NC/MPF.DIR/XYZ.DIR ", VAR1)
EP("CF_CARD:/MPF.DIR/XYZ.DIR ", VAR1)
EP("LOC:/MPF.DIR/XYZ.DIR ", VAR1)
;with return value:
; VAR1 = 0          File exists.
; VAR1 = 1          File does not exist.
```

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 NC file system.
```

Example

```
EP("\MPF.DIR\CFI.MPF", VAR1)          ;    Check whether file CFI.MPF exists in the
                                           HMI file system.
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
directory")
  ENDIF
ENDIF
```

4.3.7 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

Examples

```
MP("//NC/MPF.DIR/123.MPF","//NC/MPF.DIR/ASLAN.MPF",VAR3)              // full paths
MP("//NC/MPF.DIR/123.MPF","//NC/MPF.DIR",VAR3)                        // target without file names
MP("//NC/MPF.DIR/123.MPF",VAR0,VAR3)                                  // target via variable
MP(VAR4,VAR0,VAR3)                                  // source and target via variable
MP("CF_CARD:/mpf.dir/myprog.mdf","//NC/MPF.DIR/123.MPF",VAR3)         // from CF card in NC
MP("//NC/MPF.DIR/HOHO.MPF","CF_CARD:/xyz/123.mpf",VAR3)               // from NC in CF card
MP("USB:/mpf.dir/myprog.mdf","//NC/MPF.DIR",VAR3)//                   // from USB to NC
```

4.3.8 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")*	
Identifiers:	Selecting a 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")
```

```
//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 = "CF_CARD:/MPF.DIR/MESSEN.MPF"
  GC("CODE1",D_NAME)            ;Write code from the OUTPUT method to file
                                CF_CARD:/MPF.DIR/MESSEN.MPF
END_PRESS
PRESS(HS8)
  MP("CF_CARD:/MPF.DIR/MESSEN.MPF","//NC/MPF.DIR")      ;Load file into NC
  SP("\MPF.DIR\MESSEN.MPF")                             ;Select file
END_PRESS
```

4.3.9 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

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

4.3.10 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
```

4.3.11 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.

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 dialog and transfers one or more variables	
Parameters:	VARx	Label variables

Example

```
//M(Screen form1)
...
PRESS(HS1)
  LM("SCREEN FORM2","CFI.COM",1, POSX, POSY, DIAMETER)
                                ; Interrupt screen form1 and open screen form2.
                                  Transfer variables POSX, POSY and DIAMETER in
                                  doing this.
  DLGL("Screen form2 ended")    ; On returning from screen form2, the following
                                  text appears in the dialog line of screen form
                                  1: Screen form2 ended.
END_PRESS
...
//END

//M(Screen form2)
...
PRESS(HS1)
 EXIT(5, , CALCULATED_DIAMETER)
                                ; Exit screen form2 and return to screen form1 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
```

4.3.12 Exit Loading Softkey (EXITLS)

Description

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

Programming

Syntax:	**EXITLS***("Softkey menu"[, "Path*"])	
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
```

4.3.13 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 e.g. be called in the LOAD block or in the PRESS block.	
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 e.g. be called in the LOAD block or in the PRESS block.

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:	external "C" dllexport void InitConnection (ExtFctStructPtr FctRet, ExtFctStructPtr FctPar, char cNrFctPar)	
Description:	DLL export, only when implemented in Windows	
	Specified 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:

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

If the screens, configured with Easy Screen, 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, Easy Screen first searches for the binary file in the proj directory.

4.3.14 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:	Generate an NC code	
Parameters:	Identifier	Name of OUTPUT block from which code is generated
	Target file	Path name of target file for HMI or NC file system
		If the target file is not specified (only possible within programming support system), 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.
		1:Do not create comments in the generated code.
		Note: This code cannot be recompiled (see also Recompile without comment (Page 126)).
	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 code from OUTPUT method to file
                                              \MPF.DIR\MESSEN.MPF:
                                              Cycle123(123, -6)
                                              M30

END_PRESS
```

Recompile

- **No entry for target file:**

  The GC function can only be used in the Programming Support system 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 "Easy Screen", an error message is output when it is executed.

- **Entry for target file:**

  The code generated from the OUTPUT block is transferred to 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 considerations for target file specification

In principle, there are two different ways of specifying a target file:

- **NC notation:** `/_N_MPF_DIR/_N_MY_FILE_MPF`

  The file is created in the MPF directory on the NC.

- **DOS notation:** `d:\abc\my_file.txt` or `\\RemoteRechner\files\my_file.txt`

  The file is written to the specified directory on the hard disk or on the specified PC, provided that the directory is available on the hard disk or on a remote PC.

Note

Invalid variables generate a blank string in generated NC code and an error message in the log book when they are read.

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

See also

Recompile (Page 124)

4.3.15 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 array from 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.

4.3.16 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
		Default setting = Current configuration file

Example

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

4.3.17 Load Mask (LM)

Description

The LM function can be used to load a new dialog.

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 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 disappears; 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.
		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. EXIT will end the sub-dialog and return to the point at which the master dialog was interrupted.
		In the master dialog, the UNLOAD block is not processed during the interruption.
	VARx	Requirement: MS1
		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

```
PRESS(HS1)
  LM("SCREEN FORM2","CFI.COM",1, POSX, POSY, DIAMETER)
                               ;  Interrupt screen form1 and open screen form2:
                                  Variables POSX, POSY and DIAMETER are transferred
                                  in doing this.
  DLGL("Screen form2 ended")   ;  On returning from screen form2, the following text
                                  appears in the dialog line of screen form 1:
                                  Screen form2 ended.
END_PRESS
```

4.3.18 Load Softkey (LS)

Description

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

Programming

Syntax:	**LS***("Identifier"[, "File"][, Merge])*	
Description:	Displays softkey menu	
Parameters:	Identifier	Name of softkey menu
	File	Path (HMI file system or NC file system) to the configuration file Default: Current configuration file
	Merge	
		0:All existing softkeys are deleted; the newly configured softkeys are entered.
		1:Default 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
```

NOTICE

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 block and will not react if navigation keys are pressed (PU, PD, SL, SR, SU, SD).

4.3.19 Read NC/PLC (RNP), Write NC/PLC (WNP)

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. 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. If the value is a String type, it must be written in double quotation marks.

Example

```
WNP("DB20.DBB1",1)                    ;     Write PLC variable
```

4.3.20 Multiple Read NC PLC (MRNP)

Description

This MRNP command can be used to transfer several system or OPI variables in a single register access. This access method is significantly faster than reading via individual access attempts. The system or OPI variables must be included within an MRNP command of the same area.

The areas of the system or OPI 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. The following therefore applies: The value of the first variable is located in REG[Register index]. The value of the second variable is located in REG[Register index + 1], etc.

NOTICE
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].
```

Reading display machine data:

Display machine data can be read with RNP ($MM...) within the LOAD block.

General read/write access to display machine data is not possible using the "Easy Screen" function.

Note

User variables may not have the same names as system or PLC variables.

NC variable

All machine data, setting data and R parameters are available, but only certain system variables (see also: List of accessible system variables (Page 173)).

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
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

4.3.21 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 a blank string.

Note

Registers may not be used directly in OUTPUT blocks for generating NC code.

Programming

Syntax:	**REG***[x]*	
Description:	Defines register	
Parameters:	x	Register index with x = 0...19; Type: REAL or STRING = VARIANT 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* or *Identifier* = *Register value*	
Description:		
Parameters:	Identifier	Name of 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:	Status is a read-only property.	
Parameters:	Identifier	Name of 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.
```

4.3.22 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 assigned to the variable, and the subprogram
                                     ends at this point.
  ENDIF
  VAR1.val="Otto"                 ;  If the variable value ≠ Otto, the value "Otto"
                                     is assigned to the variable.
END_SUB                           ;  End of subroutine
//END                             ;  Block end
```

4.3.23 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 parts program
                           (X101 → X222)
```

The following value is displayed for VAR1 in the input dialog: `VAR1 = 222`

See also

Generate code (GC) (Page 107)

4.3.24 Recompile without comment

Description

In the programming support system, it is possible to **recompile without 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:

- **Expanding the easyscreen.ini**

  Section `[RECOMPILE_INFO_FILES]` will be introduced into the easyscreen.ini file. In this section, all ini files are listed that contain descriptions for cycles recompiled without 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**

  The ini file with the cycle descriptions is stored under /user or /oem in the directory /sinumerik/hmi/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 form name
Dname	Name of the file in which the screen is defined
OUTPUT	Name of the respective output block
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 Screen size for corresponding resolution: 640 X 480 mm → 16 x 16 pixels 800 X 600 mm → 20 x 20 pixels 1024 X 768 mm → 26 x 26 pixels 1280 X 1024 mm → 26 x 26 pixels 1280 X 768 mm → 26 x 26 pixels File loc.: /sinumerik/hmi/ico/ico<resolution> Note:For resolutions of 1280, the folder for 1024 x 768 mm used (only suitable for machining step programs).
Desc_Text	(optional) Explanation text for display in the machining step program, max. length of 17 character string (only suitable for machining step programs)

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 code from OUTPUT method to file
                                        \MPF.DIR\MESSEN.MPF:
                                        Cycle123(123, -6)
                                        M30

END_PRESS
```

See also

Generate code (GC) (Page 107)

4.3.25 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")*	
Identifiers:	**S**earch **F**orward: Search forward from the current cursor position	
Syntax:	**SB***("String")*	
Identifiers:	**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")
```

4.3.26 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

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 blank.
	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 from string1 into string2. Enter 0 to start searching at the beginning of string2.
	String1	Character that is being searched for.
	String2	Chain of characters in which the search is being made
	Direction (optional)	Direction in which the search is being made 0: From left to right (default setting) 1: From right to left
	0 is returned if string1 does not occur in string2.	

Example

```
DEF VAR01
DEF VAR02

LOAD
  VAR01="HELLO/WORLD"
  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 string to be processed
	length	Number of characters that are to be read out

Example

```
DEF VAR01
DEF VAR02
LOAD
 VAR01="HELLO/WORLD"
  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 string to be processed
	length	Number of characters that are to be read out

Example

```
DEF VAR01
DEF VAR02
LOAD
 VAR01="HELLO/WORLD"
  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 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="HELLO/WORLD"
  VAR02=LEFT(VAR01,4,4)                    ;  Result = "LO/W"
END_LOAD
```

REPLACE Function: Replacing characters

Syntax:	**REPLACE***(string, FindString, ReplaceString [, start [, count]])*	
Description:	The REPLACE function 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 the FindString.
Return Value:		
	string = Blank string	Copy of string
	FindString = Blank string	Copy of string
	ReplaceString = Blank string	Copy of string, in which all occurrences of FindString are deleted
	start > Len(String)	Blank string
	count = 0	Copy of string

See also

Use of strings (Page 63)

4.3.27 PI services

Description

The PI_SERVICE function can be used to start PI Services (Program Invocation Services) from the PLC in the NC area.

General programming

Syntax:	**PI_SERVICE** (*service, n parameters*)	
Description:	Executes PI service	
Parameters:	Service	PI service identifier
	n parameters	List of n parameters of PI Service.
		Individual parameters are separated by commas.

Example

```
PRESS (HS2)
  PI_SERVICE("_N_CREATO",55)
END_PRESS
PRESS(VS4)
  PI_SERVICE("_N_CRCEDN",17,3)
END_PRESS
```

Starting OEM services

The PI_START command executes a PI service based on OEM documentation.

Programming

Syntax:	PI_START("*Transfer string*")	
Description:	Executes 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.

Graphic and logic elements

5

5.1 Line and rectangle

Description

Lines and rectangles are configured in the LOAD block:

- Lines are first drawn, then the rectangles and finally the configured control fields or graphics.
- Transparent rectangles are created by setting the fill color to the system background color.

LINE element

Programming:

Syntax:	LINE (x1,y1,x2,y2,f,s)	
Description:	Defining a line	
Parameters:	x1	Start point x-coordinate
	y1	Start point y-coordinate
	x2	End point x-coordinate
	y2	End point y-coordinate
	f	Color of the line
	s	Line style:
		1 = solid
		2 = dashed
		3 = dotted
		4 = dashed and dotted

RECT element

Programming:

Syntax:	RECT (x,y,w,h,f1,f2,s)	
Description:	Defining a rectangle	
Parameters:	x	x-coordinate, top left
	y	y-coordinate, top left
	w	Width
	h	Height
	f1	Color of the border
	f2	Fill color
	s	Border style:
		1 = solid
		2 = dashed
		3 = dotted
		4 = dashed and dotted

See also

LOAD (Page 81)

5.2 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)*	
	(a/b...)	
	(c/d...)	
	...	
	//END	
Description:	Defines array	
Parameters:	Identifier	Name of array
	a, b, c, d	Values of array
		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
```

5.2.1 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. The ERR variable is also set to TRUE if a search concept cannot be found.

Programming

Syntax:	Identifier [Z,[M[,C]]].val or	
	Identifier [Z,[M[,C]]]	
Description:	Access to one-dimensional array with only one column	
Syntax:	Identifier [S,[M[,C]]].val] or	
	Identifier [S,[M[,C]]] or	
Description:	Access to one-dimensional array with only one line	
Syntax:	Identifier [Z,S,[M[,C]]].val or	
	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: 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: Search in line 0	Value from line Z and column found
3	Search concept: Search in column 0	Search concept: Search in line 0	Value from line and column found
4	Search concept: Search in column S	Column index of search column	Line index
5	Line index of search line.	Search concept: 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 larger 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 found in the search line or search column. If the search concept cannot be found, the value 0 or an empty string is output and the error variable ERR is set to TRUE.

5.2.2 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.

5.2.3 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:	Status is a read-only property.
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
```

5.3 Table grid (grid)

Definition

In contrast to the array, the values of a table grid (grid) are continually updated. This involves a tabular representation of the values of system variables that can be addressed using one block in one channel.

Assignment

A variables definition is assigned to the table-elements definition via a table identifier:

- 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 table grid takes the properties of the IO fields from the variables definition line.
- The visible area of the grid is determined by the width and height of the I/O 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/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.

System or user variable

This parameter remains empty for table grids, because the column definition lines contain detailed information about the variables to be displayed. The table description can be provided in a dynamic format.

Description

The variables definition will contain a reference to a table description:

DEF *Identifier* =	**Identifier = Name of variable**
	Variable type
	/[Limits or toggle field or table identifier]
	/[Default]
	/[Texts (Long text, Short text|Image, Graphic text, Units text)]
	/[Attributes]
	/[Help display]
	/[System or user variable]
	/[Position of short text]
	/[Position input/output field(Left, Top, Width, Height)]
	/[Colors]

See also

Variable parameters (Page 52)

5.3.1 Defining table grids

Description

The table block comprises:

- Header
- 1 to n column descriptions

Programming

Syntax:	*//***G***(Table identifier/Table type/Number of lines/ [Fixed line attribute],[Fixed column attribute])*	
Description:	Defines table grids	
Parameters:	Table identifiers	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) 1and others, reserved
	No. 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.
	Fixed line attribute	1:Active 0:Not active
	Fixed column attribute	1:Active 0:Not active

5.3.2 Defining columns

Description

For table 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 grid can be modified directly by the end user 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)*	
Description:	Defines 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: • Offset1: Step width for the 1st index • Offset2: Step width for the 2nd index • Offset3: Step width for the 3rd index

Variable of type STRING

If the variable is a STRING type, then the length must be specified in the type, e.g.:

```
DEF CHAN STRING [16] TEXT[41]
```

The column definition for the CHAN variable, therefore, starts, e.g. (S16/...).

Column header from text file

The column header can be entered as text or text numbers ($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 display (hlp) and
- OPI-Variable (var).

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"`

Column properties cannot be read in the LOAD block.

The wr, ac and li attributes can be specified for column definitions.

5.3.3 Focus control in the table grid

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
	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
```

5.4 Custom widgets

5.4.1 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 software options:

"SINUMERIK HMI sl Runtime OA programming"

"SINUMERIK HMI sl Runtime OA Easy Screen"

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);
```

5.4.2 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, Easy Screen accesses a dll file 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
....
....
}
```

5.4.3 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 Easy Screen 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 widgets is inserted at the
                                      beginning of the file
....
//Makros                            ; Macro definitions are not changed
....
WIDGET_CLASS_EXPORT(SlEsTestCustom  ; The relevant custom widget is
Widget)                               declared at 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)
```

5.4.4 Interaction between custom widget and dialog

Description

Custom widgets interact with dialog boxes and can display values or manipulate them. Data is therefore exchanged for 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);
....
....
}
```

"Custom" operating area

6

6.1 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 /siemens/sinumerik/hmi/templates directory into the /siemens/sinumerik/hmi/cfg directory.
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.

6.2 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: /oem/sinumerik/hmi/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 can be used to make 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.

6.3 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 /siemens/sinumerik/hmi/templates directory.

1. First copy the files to the /oem/sinumerik/hmi/cfg directory and make your changes from there.
2. File easyscreen.ini 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 is used to define that the entry will refer to the custom.com project file when the "Custom" operating area is selected.
3. You create the **custom.com project file** in the /oem/sinumerik/hmi/proj directory. 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: /oem/sinumerik/hmi/ico/ …

6.4 Programming example for the "Custom" area

File overview

The following files are required:

- custom.ini
- easyscreen.ini

Programming

Content of the custom.com file:

```
//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
```

Result

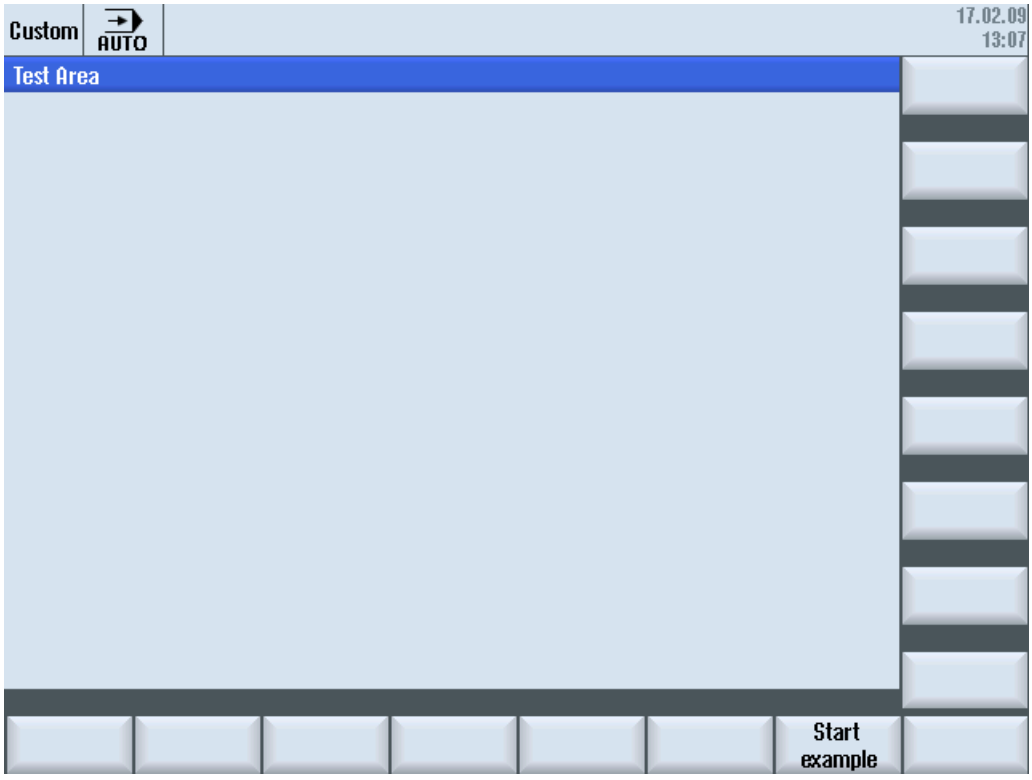


Figure 6-1 Example with "Start example" softkey

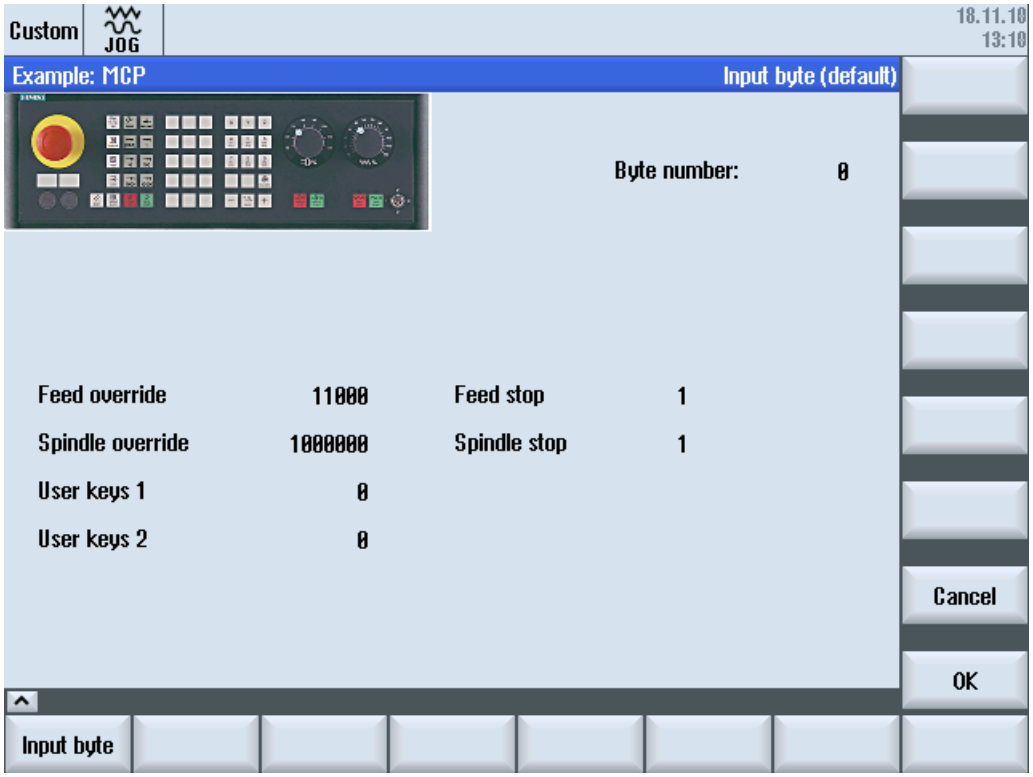


Figure 6-2 Example with bitmap and text fields

PLC softkeys

7

7.1 Introduction

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. An Easy Screen 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

Easy Screen makes it possible to select Easy Screen softkey menus and Easy Screen 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,

Cmdline:=cmdline
```

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]`
`100.screen1 = LS("Softkey1","param.com")`
`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.
	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.
	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.
	The "screen" information is optional.
	The following commands are permissible:
	`LM (LoadMask)`
	`LS (LoadSoftkeys)`

Reference lists

A

A.1 Lists of start softkeys

A.1.1 List of start softkeys for turning

Program operating area for turning

Edit	Drilling	Turning	Contour turning	Milling	Miscellaneous	Simulation	NC select
HSK1	HSK2	HSK3	HSK4	HSK5	HSK6	HSK7	HSK8
--	--	--	--	Measure turning	Measure milling	OEM	--
--	HSK10	--	--	HSK13	HSK14	HSK15	--

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 Easy Screen.

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	Counterspindle	--	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	<<	>>	<<	--

See also

Defining start softkeys (Page 15)

A.1.2 List of start softkeys for milling

Program operating area when milling

Edit	Drilling	Milling	Contour milling	Turning	Miscellaneous	Simulation	NC select
HSK1	HSK2	HSK3	HSK4	HSK5	HSK6	HSK7	HSK8
--	--	--	--	Measure turning	Measure milling	**OEM**	--
--	HSK10	--	--	HSK13	HSK14	HSK15	--

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 Easy Screen.

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**		**Turning**		**Miscellaneous**		**Straight line circle**
	HSK2	**HSK3**	**HSK4**		**HSK5**		**HSK6**		**HSK10**
VSK1	Centering	Face milling	New contour	--	Stock removal	New contour	Settings	--	Tool
VSK2	Drilling reaming	Pocket	Path	--	Groove	Stock removal	Swivel plane	Parallel axes	Straight line
VSK3	Deep-hole drilling	Multi-edge spigot	Predrilling	--	Undercut	Stock removal residual material	Swivel tool	Repeat progr.	Circle center point
VSK4	Boring	Groove	Pocket	--	Thread	Grooving	High speed settings	--	Circle radius
VSK5	Thread	Thread milling	Pocket res. mat.	--	Parting	Grooving residual material	Transformations	--	Helix
VSK6	**OEM**	Engraving	Spigot	--	--	Plunge-turning	Subprogram	--	Polar
VSK7	Positions	**OEM**	Spigot res. mat.	**OEM**	**OEM**	Plunge turning residual material	--	**OEM**	--
VSK8	Repeat position.	--	>>	<<	Contour turning	<<	>>	<<	--

A.2 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 128).

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	
128		orange	System color active field
129		Light gray	Background color
130		Blue	Header color (active)
131		black	Header font color (active)

A.3 List of language codes used in file names

Supported languages

Standard languages:

Language	Abbreviation in file name
Chinese simplified	chs
German	deu
English	eng
Spanish	esp
French	fra
Italian	ita

Other languages:

Language	Abbreviation in file name
Chinese traditional	cht
Korean	kor
Portuguese (Brazil)	ptb

Language	Abbreviation in file name
Czech	csy
Hungarian	hun
Japanese	jpn
Polish	plk
Russian	rus
Swedish	sve

Language	Abbreviation in file name
Danish	dan
Finnish	fin
Dutch	nld
Romanian	rom
Slovakian	sky
Turkish	trk

A.4 List of accessible system variables

References

List Manual System Variables/PGAsl/

See also

Multiple Read NC PLC (MRNP) (Page 118)

Glossary

Access level

Graduated system of authorization, which makes the accessibility and utilization of functions on the operator interface dependent on the authorization rights of the user.

Array

An array can be used to organize data of a standard data type stored in the memory in such a way that it is possible to access the data via an index.

Attribute

Characteristic that assigns specific → Properties to an object (→ Dialog or → Variable).

Column index

Column number of an array

Configuration file

File, which contains definitions and instructions that determine the appearance of → Dialogs and their → Functions.

Definition lines

Program section in which → Variables and softkeys are defined

Dialog

Display of the → User interface

- **Dialog-dependent softkey menu**

  Softkey menu, which is called from a newly configured dialog.

- **Dialog-independent softkeys**

  Softkeys, which are not called from a dialog, i.e., start softkey and softkey menus, which the user configures before the first, new dialog.

Editor

ASCII Editor with which characters can be entered in a file and edited.

Event

Any action, which initiates execution of a → Method: Input of characters, actuation of softkeys, etc.

Group

Reload unit for → Configuration file

Help variable

Internal arithmetic variable to which no → Properties can be assigned and is not, therefore, visible in the → Dialog.

Hotkeys

6 keys on OP 010, OP 010C and SINUMERIK keyboards with hotkey blocks. Pressing the keys selects an operating area directly. As an option, 2 additional keys can be configured as hotkeys.

Input/output field

Also I/O field: for inputting or outputting variable values.

Interpreter

The interpreter automatically converts the defined code from the → Configuration file into a → Dialog and controls its use.

Line index

Row number of an array

Menu tree

A group of interlinked → Dialogs

Method

Programmed sequence of operations executed when a corresponding → Event occurs.

Parameter

Parameters are variable elements of the programming syntax and are replaced by other words/symbols in the → Configuration file.

PI service

Function which, on an NC, executes a clearly defined operation. PI services can be called from the PLC and the HMI system.

PLC hard key

PLC hard keys are provided via the PLC interface of the HMI software, just like hotkeys. The functions triggered by them in the HMI can be configured.

They take the form of MCP keys or evaluations of PLC signal logic operations in the PLC user program. For this reason, they are referred to as "virtual keys".

Programming support

Provision of → Dialogs to assist programmers in writing → Parts programs with "higher-level" components

Properties

Characteristics of an object (e.g of a → Variable)

Recompile

NC code sections can be generated in a → Part program from input fields in → Dialogs in the → Programming support system. Recompilation is the reverse operation. The input fields used to generate a selected section of NC code are retrieved from the NC code and displayed in the original dialog.

Selecting

A program formulated in the NC language, which specifies motion sequences for axes and various special actions.

Simulation

Simulation of a → Parts program run without movement of the actual machine axes.

Softkey labels

Text/image on the screen, which is assigned to a softkey.

Softkey menu

All horizontal or all vertical softkeys

Start softkey

Softkey with which the first newly created → Dialog is started.

Toggle field

A list of values in the → Input/output field; check with toggle field: The value input in a field must be the same as one of the listed values.

User variable

Variables defined by the user in the → Parts program or data block.

Variable

Designation of a memory location, which can be displayed in a → Dialog by assigning → Properties and in which input data and the results of arithmetic operations can be entered.

Index

G

H

I

L

M

N

O

P

R

S

T

U

V

W