SIEMENS

SINUMERIK

SINUMERIK 840D sl
Easy XML

Programming Manual

Valid for:

CNC software Version 4.95

07/2021
6FC5398-5EP40-0BA0

Legal information

Warning notice system

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

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

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

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

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

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

Qualified Personnel

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

Proper use of Siemens products

Note the following:

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

Trademarks

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

Disclaimer of Liability

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

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

Document order number: 6FC5398-5EP40-0BA0
Ⓟ 06/2021 Subject to change

Table of contents

Introduction 1

1.1 About SINUMERIK

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

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

1.2 About this documentation

Target group

This publication is intended for:

- Programmers
- Project engineers

Benefits

The Programming Manual enables the target group to design customer and application-specific user interfaces with XML-based script elements.

Standard scope

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

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

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

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

Websites of third-party companies

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

1.3 Documentation on the internet

1.3.1 Documentation overview SINUMERIK 840D sl

You will find extensive documentation on the functions of SINUMERIK 840D sl from version 4.8 SP4 at 840D sl documentation overview (https://support.industry.siemens.com/cs/ww/en/view/109766213).

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

The documentation is divided into the following categories:

- User: Operating
- User: Programming
- Manufacturer/Service: Functions
- Manufacturer/Service: Hardware
- Manufacturer/Service: Configuration/Setup
- Manufacturer/Service: Safety Integrated
- Manufacturer/Service: SINUMERIK Integrate/MindApp
- Information and training
- Manufacturer/Service: SINAMICS

1.3.2 Documentation overview SINUMERIK operator components

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

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

The documentation is divided into the following categories:

- Operator Panels
- Machine control panels
- Machine Pushbutton Panel
- Handheld Unit/Mini handheld devices
- Further operator components

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

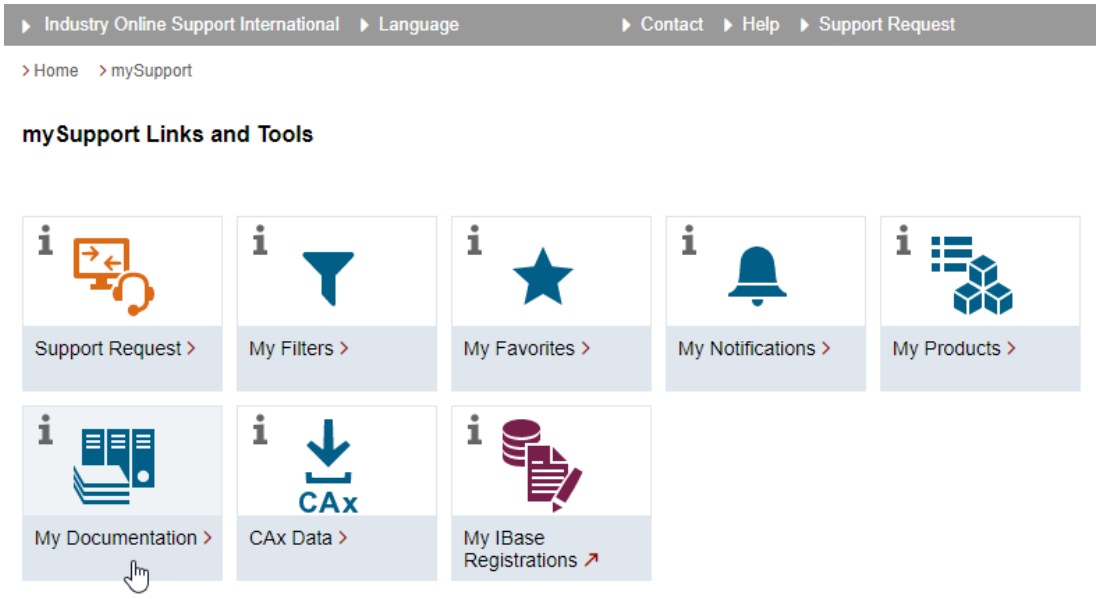
1.4 Feedback on the technical documentation

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

1.5 mySupport documentation

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

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



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

Note

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

1.6 Service and Support

Product support

You can find more information about products on the internet:

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

The following is provided at this address:

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

Technical support

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

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

Training

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

Siemens support on the go

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

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

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

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

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

Data matrix code on the nameplate

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

1.7 Important product information

Using OpenSSL

This product can contain the following software:

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

You can find more information on the internet:

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

Compliance with the General Data Protection Regulation

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

For this product, this means:

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

Fundamental security instructions 2

2.1 General safety instructions

⚠ WARNING

Electric shock and danger to life due to other energy sources

Touching live components can result in death or severe injury.

- Only work on electrical devices when you are qualified for this job.
- Always observe the country-specific safety rules.

Generally, the following steps apply when establishing safety:

1. Prepare for disconnection. Notify all those who will be affected by the procedure.
2. Isolate the drive system from the power supply and take measures to prevent it being switched back on again.
3. Wait until the discharge time specified on the warning labels has elapsed.
4. Check that there is no voltage between any of the power connections, and between any of the power connections and the protective conductor connection.
5. Check whether the existing auxiliary supply circuits are de-energized.
6. Ensure that the motors cannot move.
7. Identify all other dangerous energy sources, e.g. compressed air, hydraulic systems, or water. Switch the energy sources to a safe state.
8. Check that the correct drive system is completely locked.

After you have completed the work, restore the operational readiness in the inverse sequence.

⚠ WARNING

Electric shock due to connection to an unsuitable power supply

When equipment is connected to an unsuitable power supply, exposed components may carry a hazardous voltage. Contact with hazardous voltage can result in severe injury or death.

- Only use power supplies that provide SELV (Safety Extra Low Voltage) or PELV- (Protective Extra Low Voltage) output voltages for all connections and terminals of the electronics modules.

⚠ WARNING

Electric shock due to equipment damage

Improper handling may cause damage to equipment. For damaged devices, hazardous voltages can be present at the enclosure or at exposed components; if touched, this can result in death or severe injury.

- Ensure compliance with the limit values specified in the technical data during transport, storage and operation.
- Do not use any damaged devices.

⚠ WARNING

Electric shock due to unconnected cable shields

Hazardous touch voltages can occur through capacitive cross-coupling due to unconnected cable shields.

- As a minimum, connect cable shields and the cores of cables that are not used at one end at the grounded housing potential.

⚠ WARNING

Electric shock if there is no ground connection

For missing or incorrectly implemented protective conductor connection for devices with protection class I, high voltages can be present at open, exposed parts, which when touched, can result in death or severe injury.

- Ground the device in compliance with the applicable regulations.

NOTICE

Damage to equipment due to unsuitable tightening tools.

Unsuitable tightening tools or fastening methods can damage the screws of the equipment.

- Be sure to only use screwdrivers which exactly match the heads of the screws.
- Tighten the screws with the torque specified in the technical documentation.
- Use a torque wrench or a mechanical precision nut runner with a dynamic torque sensor and speed limitation system.

⚠ WARNING

Spread of fire from built-in devices

In the event of fire outbreak, the enclosures of built-in devices cannot prevent the escape of fire and smoke. This can result in serious personal injury or property damage.

- Install built-in units in a suitable metal cabinet in such a way that personnel are protected against fire and smoke, or take other appropriate measures to protect personnel.
- Ensure that smoke can only escape via controlled and monitored paths.

⚠ WARNING

Unexpected movement of machines caused by radio devices or mobile phones

Using radio devices or mobile telephones in the immediate vicinity of the components can result in equipment malfunction. Malfunctions may impair the functional safety of machines and can therefore put people in danger or lead to property damage.

- Therefore, if you move closer than 20 cm to the components, be sure to switch off radio devices or mobile telephones.
- Use the "SIEMENS Industry Online Support app" only on equipment that has already been switched off.

⚠ WARNING

Fire due to inadequate ventilation clearances

Inadequate ventilation clearances can cause overheating of components with subsequent fire and smoke. This can cause severe injury or even death. This can also result in increased downtime and reduced service lives for devices/systems.

- Ensure compliance with the specified minimum clearance as ventilation clearance for the respective component.

NOTICE

Overheating due to inadmissible mounting position

The device may overheat and therefore be damaged if mounted in an inadmissible position.

- Only operate the device in admissible mounting positions.

⚠ WARNING

Unexpected movement of machines caused by inactive safety functions

Inactive or non-adapted safety functions can trigger unexpected machine movements that may result in serious injury or death.

- Observe the information in the appropriate product documentation before commissioning.
- Carry out a safety inspection for functions relevant to safety on the entire system, including all safety-related components.
- Ensure that the safety functions used in your drives and automation tasks are adjusted and activated through appropriate parameterizing.
- Perform a function test.
- Only put your plant into live operation once you have guaranteed that the functions relevant to safety are running correctly.

Note

Important safety notices for Safety Integrated functions

If you want to use Safety Integrated functions, you must observe the safety notices in the Safety Integrated manuals.

⚠ WARNING

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

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

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

2.2 Equipment damage due to electric fields or electrostatic discharge

Electrostatic sensitive devices (ESD) are individual components, integrated circuits, modules or devices that may be damaged by either electric fields or electrostatic discharge.

NOTICE

Equipment damage due to electric fields or electrostatic discharge

Electric fields or electrostatic discharge can cause malfunctions through damaged individual components, integrated circuits, modules or devices.

- Only pack, store, transport and send electronic components, modules or devices in their original packaging or in other suitable materials, e.g conductive foam rubber of aluminum foil.
- Only touch components, modules and devices when you are grounded by one of the following methods:
  - Wearing an ESD wrist strap
  - Wearing ESD shoes or ESD grounding straps in ESD areas with conductive flooring
- Only place electronic components, modules or devices on conductive surfaces (table with ESD surface, conductive ESD foam, ESD packaging, ESD transport container).

2.3 Warranty and liability for application examples

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

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

2.4 Security information

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

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

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. These systems, machines and components should only be connected to the enterprise network or the Internet if and only to the extent necessary and with appropriate security measures (firewalls and/or network segmentation) in place.

You can find more information on protective measures in the area of industrial security by visiting:
https://www.siemens.com/industrialsecurity (https://www.siemens.com/industrialsecurity).

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

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

Further information is provided on the Internet:

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

⚠ WARNING

Unsafe operating states resulting from software manipulation

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

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

2.5 Residual risks of power drive systems

When assessing the machine- or system-related risk in accordance with the respective local regulations (e.g., EC Machinery Directive), the machine manufacturer or system installer must take into account the following residual risks emanating from the control and drive components of a drive system:

1. Unintentional movements of driven machine or system components during commissioning, operation, maintenance, and repairs caused by, for example,
   - Hardware and/or software errors in the sensors, control system, actuators, and cables and connections
   - Response times of the control system and of the drive
   - Operation and/or environmental conditions outside the specification
   - Condensation/conductive contamination
   - Parameterization, programming, cabling, and installation errors
   - Use of wireless devices/mobile phones in the immediate vicinity of electronic components
   - External influences/damage
   - X-ray, ionizing radiation and cosmic radiation
2. Unusually high temperatures, including open flames, as well as emissions of light, noise, particles, gases, etc., can occur inside and outside the components under fault conditions caused by, for example:
   - Component failure
   - Software errors
   - Operation and/or environmental conditions outside the specification
   - External influences/damage
3. Hazardous shock voltages caused by, for example:
   - Component failure
   - Influence during electrostatic charging
   - Induction of voltages in moving motors
   - Operation and/or environmental conditions outside the specification
   - Condensation/conductive contamination
   - External influences/damage
4. Electrical, magnetic and electromagnetic fields generated in operation that can pose a risk to people with a pacemaker, implants or metal replacement joints, etc., if they are too close
5. Release of environmental pollutants or emissions as a result of improper operation of the system and/or failure to dispose of components safely and correctly
6. Influence of network-connected communication systems, e.g. ripple-control transmitters or data communication via the network

For more information about the residual risks of the drive system components, see the relevant sections in the technical user documentation.

Generating user dialogs 3

3.1 Scope of functions

Overview

The "Generate user dialogs" function offers an open structure and enables the user to develop customer-specific and application-specific user interfaces in SINUMERIK Operate.

The control system offers an XML-based script language for generating user dialogs.

This script language makes it possible to display machine-specific menus and dialog forms in the <CUSTOM> operating area in SINUMERIK Operate.

All dialog forms can be designed on a language-neutral basis. In such cases, the system reads out the texts to be displayed from the accompanying language database.

Use

The defined XML instructions offer the following properties:

1. Display dialogs containing the following elements:
   - Softkeys
   - Variables
   - Text and help text
   - Graphics and help displays
2. Call dialogs by:
   - Pressing the (start) softkeys
3. Restructure dialogs dynamically:
   - Edit and delete softkeys
   - Define and design variable fields
   - Insert, exchange, and delete display texts (language-dependent or language-neutral)
   - Insert, exchange, and delete graphics
   - Dynamically create executable script parts and include them in the script processing
4. Initiate operations in response to the following actions:
   - Displaying dialogs
   - Inputting values (variables)
   - Selecting 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:
   - Subprograms
   - File functions
   - PI services
8. Apply protection levels according to user groups

The valid elements (tags) for the script language are described in the "XML tags" (Page 44) section.

Note

The following section "Basic principles of configuration" is not intended as a comprehensive description of XML (Extensible Markup Language). Please refer to the relevant specialist literature for additional information.

3.2 Fundamentals of Configuration

Configuration files

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

An XML editor or another form of text editor can be used to generate the configuration files.

Note

File names may only contain lowercase letters.

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.

Configured start softkeys can be used to create a further menu tree behind the start menu:

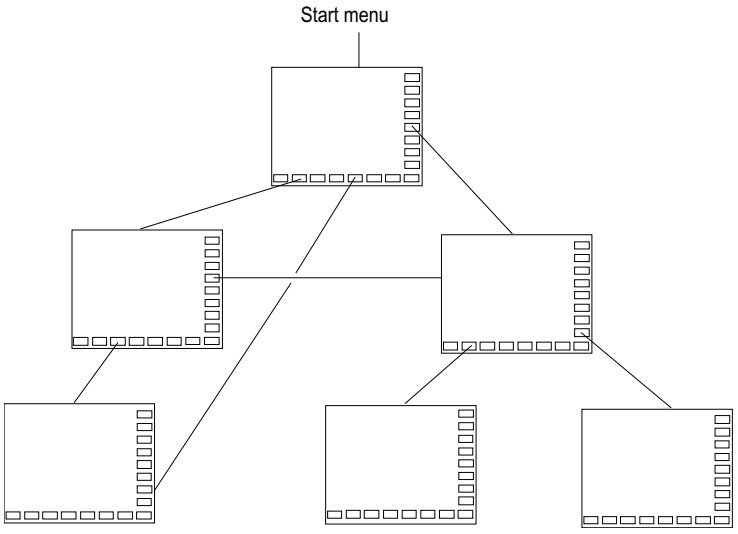


Figure 3-1 Menu tree for user dialogs

Start menu

The start menu is defined by the name "main" in the "xmldial.xml" file. The start menu is used to initiate your own operating sequences.

You can use the "main" menu to link your own dialogs or additional softkey bars so that they can be loaded and used for executing additional actions.

The figure below shows the manufacturer's folder "System CF-Card/oem/sinumerik/hmi" on the control.

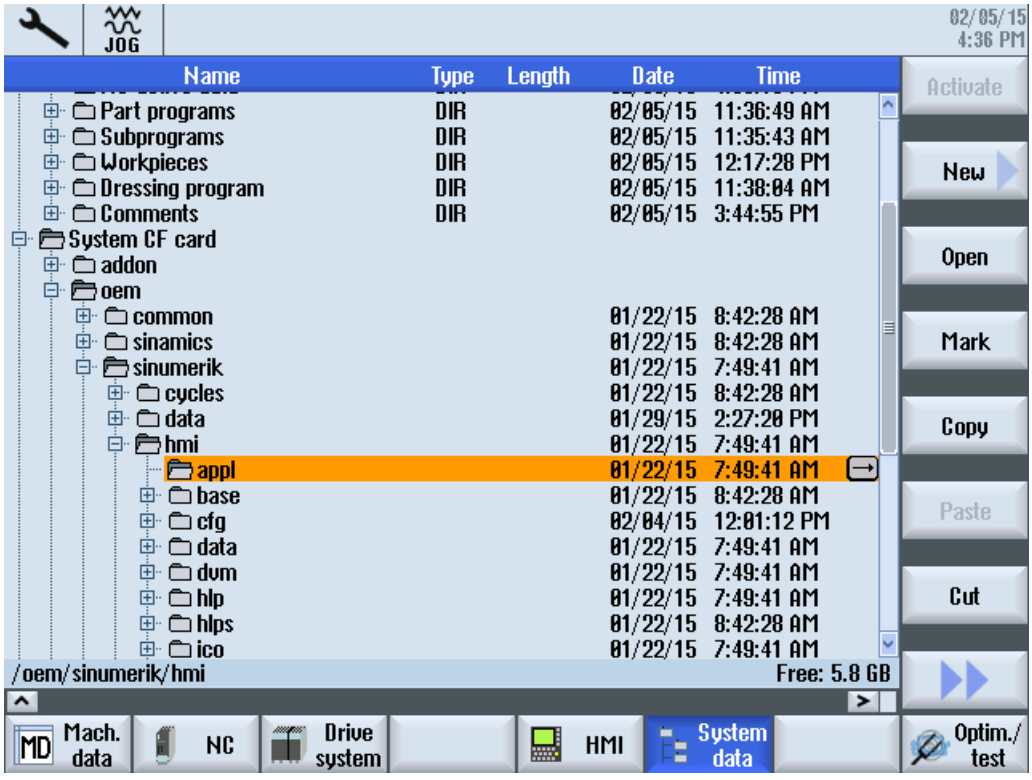


Figure 3-2 OEM directory

The following files in the OEM directory "System CF-Card/oem/sinumerik/hmi" on the control are needed to configure user dialogs:

Table 3-1 Files for configuration

File type	Name of the file	Meaning	Storage location in operating area Setup > System data
Script file	"xmldial.xml"	Standard text file This script file uses XML tags to control how the configured soft-key menus and dialog forms in SINUMERIK Operate will appear in the <CUSTOM> operating area.	OEM directory > subdirectory "appl" for the applications
Application-specific script file	"xxx.xml"	Freely selectable file name. The reference to this name is established via the corresponding INI files.	

File type	Name of the file	Meaning	Storage location in operating area Setup > System data
Text file	"oem_xml_screens_xxx.ts"	Standard text file for the customer operating area This text file contains the texts for the menus and dialog forms for the individual languages.	OEM directory > subdirectory "lng" for the desired languages
User-specific text file	"yyy_xxx.ts"	Freely selectable file name. The reference to this name is established using the corresponding attributes at tags DialogGUI, menu or form. The description is provided in Chapter "Project-specific text files (Page 254)". For Sidescreen or Display Manager applications, the text file name is derived from the page or widget name. You can find an example in Chapter "Language and text management (Page 230)".	
Bitmaps		The control system supports BMP and PNG formats.	OEM directory > subdirectory "ico"
XML files inserted in the "xmldial.xml" control file with the "INCLUDE" XML tag.	E.g. "machine_settings.xml"	These files also contain programmed instructions for displaying the dialog forms and parameters in SINUMERIK Operate.	OEM directory > subdirectory "appl" for the applications

Easy XML connection points

The following connection points are available for Easy XML scripts:

Hardkey/Softkey	Connection point
Softkey operating menu	Standard script file name "xmldial.xml" Activated in the file "slamconfig.ini" **Example:** `[CustomXML]` `TextId=SL_AM_CUSTOM` `SidescreenTextId=SL_SIDESCREEN_CUSTOM` `TextFile=slam` `TextContext=SlAmAreaMenu` `SoftkeyPosition=12` `Visible=true`
Menu User	Standard script file name "menu_user.xml"
Menu Function	Standard script file name "menu_function.xml"

Hardkey/Softkey	Connection point
PLC hardkey	The script file name relates to the key description. **Example:** `KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog, cmdline:="-conf slagmdialog.hmi -mainModule restore.xml -entry cmc2"` `KEY51.0 = area:=CustomXML, dialog:=SlEECustomDialog, cmdline:="-conf slagmdialog.hmi -mainModule activate.xml -entry cmc1"`
MMC command	The script file name relates to the NC command description. **Example:** `MMC("POPUPDLG, XML_ON, TEST.XML, cmd1","A)`

Hardkey/Softkey	Connection point
Reserved softkeys of an operating area	To integrate an OEM application into an operating area, the appropriate templates are available in directory siemens\sinumerik\hmi\template\cfg\ (sl<operating area>_oem.xml). The XML file belonging to the operating area should be copied to directory oem\sinumerik\hmi\cfg and edited as follows: Function **switchToDialog** should be programmed as softkey response. Keywords **area** with value **CustomXML** and **dialog** with value **SlEECustomDialog** should be specified as arguments. **Syntax:** `<FUNCTION args="-area CustomXML -dialog SlEECustomDialog -mainModule <name of the main module> -entry <menu name>" name="switchToDialog"/>` **Example:** Integrating an Easy XML application into the diagnostics area: "dg.xml" was used as script file name. In this file, menu with name **main** should be called. `<MENU name="DgGlobalHu">` `<ETCLEVEL id="0">` `<SOFTKEYGROUP name="GroupEtc">` `<SOFTKEY position="7">` `<PROPERTY name="textID" type="QString">DG_SK</PROPERTY>` `<PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>` `<FUNCTION args="-area CustomXML -dialog SlEECustomDialog -mainModule dg.xml -entry main" name="switchToDialog"/>` `</SOFTKEY>` `</SOFTKEYGROUP>` `</ETCLEVEL>` `</MENU>`
Sidescreen	The script file name relates to the sidescreen description of the file "slsidescreen.ini".
Easy Extend softkey	Standard script file name "agm.xml"

3.3 Configuration files

Introduction

The figure below shows the manufacturer's folder "System CF-Card/oem/sinumerik/hmi" on the control.

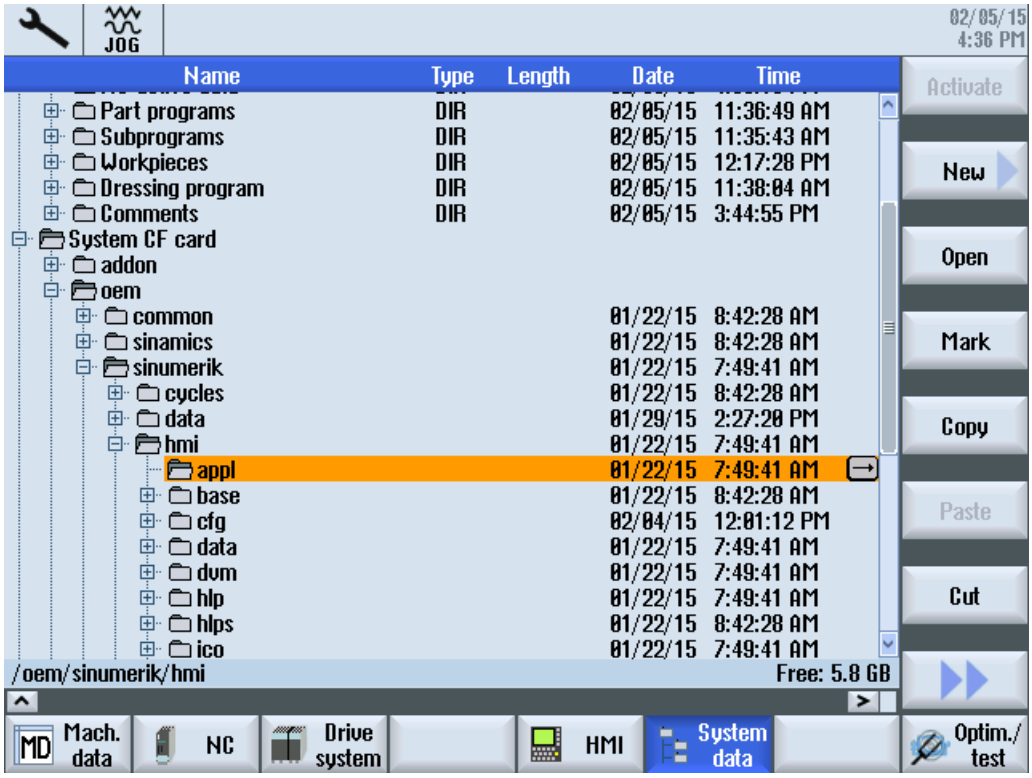


Figure 3-3 Manufacturer's folder

The following files in the manufacturer's folder "System CF-Card/oem/sinumerik/hmi" on the control are needed to configure user dialogs:

Table 3-2 Files for configuration

File type	Name of the file	Meaning	Storage location in operating area Setup > System data
Script file	"xmldial.xml"	This script file uses XML tags to control how the configured softkey menus and dialog forms in SINUMERIK Operate will appear in the <CUSTOM> operating area.	Manufacturer's folder > subdirectory "appl" for the applications
Text file	"oem_xml_screens_xxx.ts"	This text file contains the texts for the menus and dialog forms for the individual languages.	Manufacturer's folder > subdirectory "lng" for the languages

File type	Name of the file	Meaning	Storage location in operating area Setup > System data
Bitmaps		The control system supports BMP and PNG formats.	Manufacturer's folder > subdirectory "ico" The bitmaps are saved in the subdirectory for the screen resolution belonging to the control. Note: If a path to the bitmap file is specified, the files can be stored in this directory directly.
XML files inserted in the "xmldial.xml" control file with the "INCLUDE" XML tag.	E.g. "machine_settings.xml"	These files also contain programmed instructions for displaying the dialog forms and parameters in SINUMERIK Operate.	Manufacturer's folder > subdirectory "appl" for the applications

Dependencies of files for configuring user dialogs

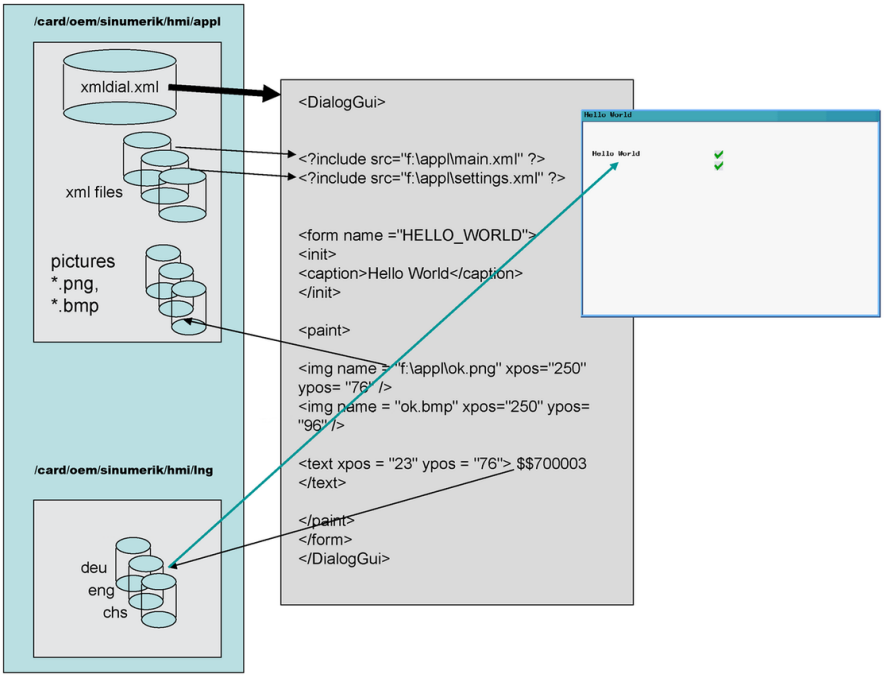


Figure 3-4 Dependencies

Load configuration

As described in the "Storage location in the operating area" column in the previous "Files for configuration" table, the generated files must be copied to the appropriate subdirectories in the manufacturer's folder.

Note

As soon as there is a script file "xmldial.xml" in the subdirectory for applications, the user can start this user dialog in the <CUSTOM> operating area.

After the initial copying process, the HMI must be reset via a "Normal power-up".

Example of a user dialog in SINUMERIK Operate

The configured softkey menus are displayed when the <CUSTOM> operating area is called. This enables the user to operate the dialog forms which have been configured.

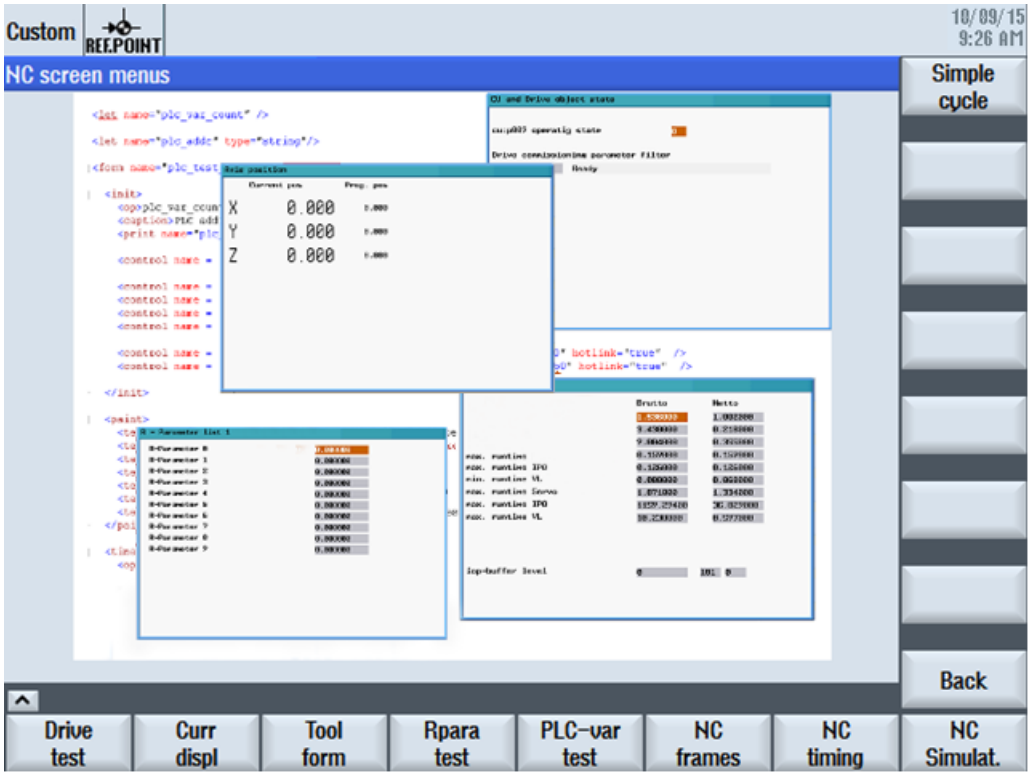


Figure 3-5 Example of a user dialog in the <CUSTOM> operating area

Further examples are provided in the toolbox.

See also

Predefined functions (Page 153)

3.4 Structure of configuration file

Overview

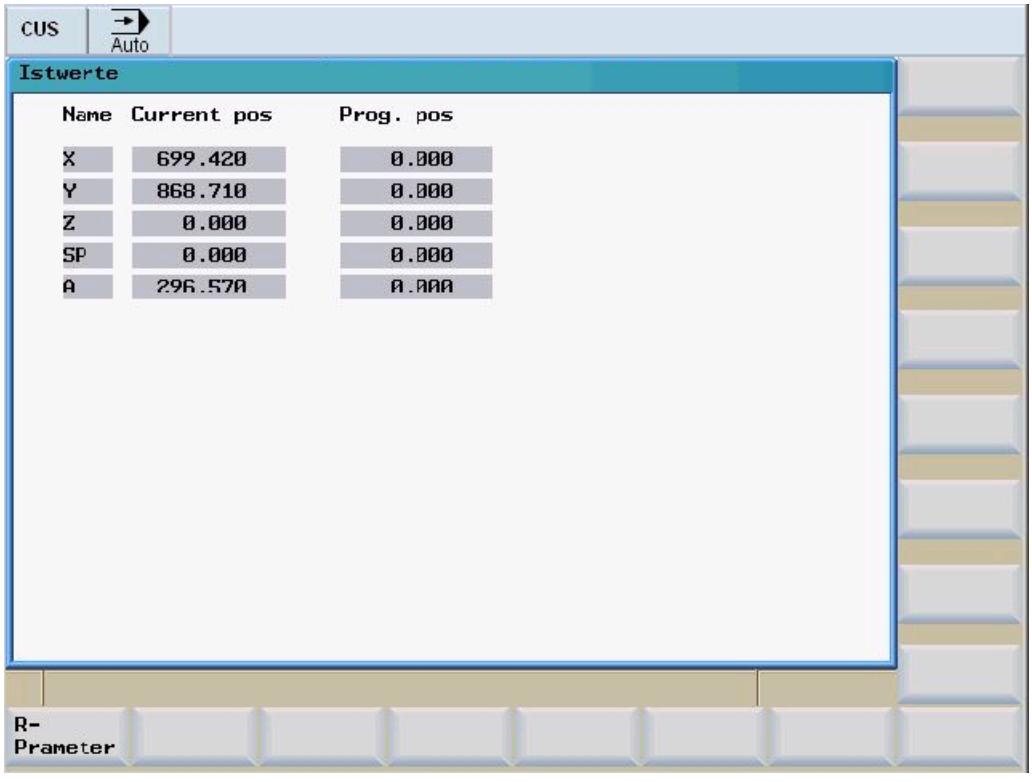
A configuration file consists of the following elements:

- Description of the "main" start menu with start softkeys
- Definition of the dialogs
- Definition of the variables
- Description of the blocks
- Definition of softkey bars

The following examples show the XML script for the "xmldial.xml" file and the corresponding screenshots.

The script contains the dialogs for displaying actual values and residual distances, as well as an R parameter list.

Dialog form section "Actual values"



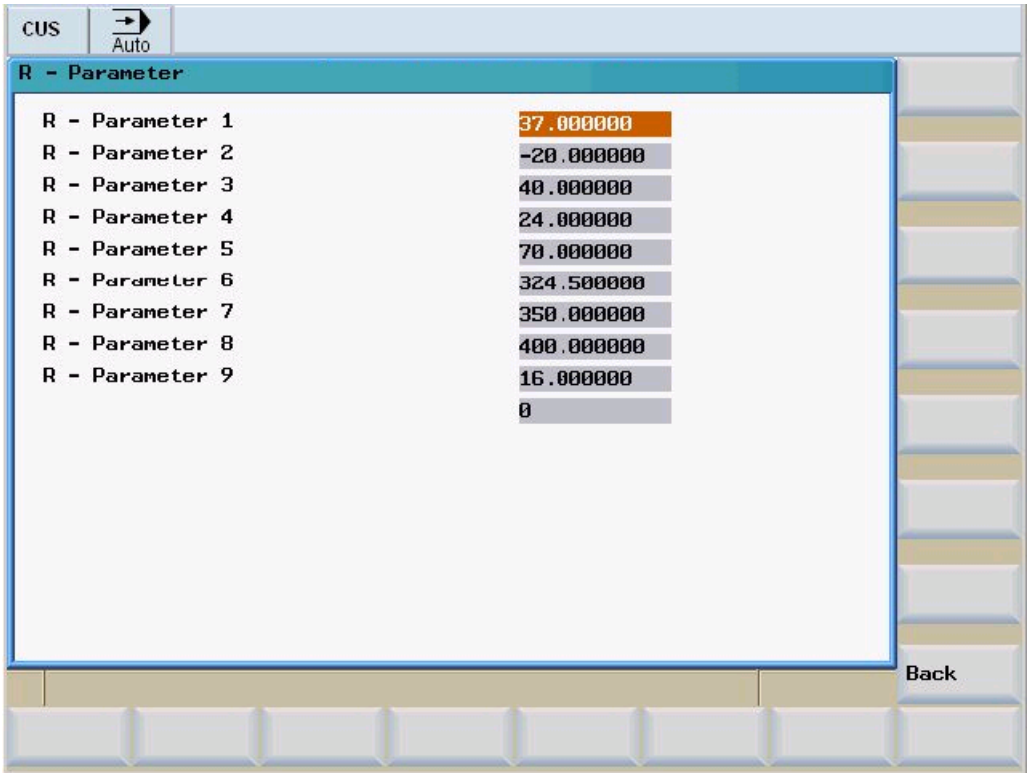
xmldial.xml

```
<DialogGui>

<!--
main menu
It is called by the system software. It starts the application.
The menu tag manages the soft key reactions. One input form can be assigned
to a menu tag.
-->
<menu name = "MAIN">
<OPEN_FORM name = "CURRENT_DISPLAY" />

<softkey POSITION="1">
<caption>R-%nPrameter</caption>
<navigation>MENU_R_PARAMETER</navigation> <!-- opens the menu R parameter --
>
</softkey>


</menu>


<form name = "CURRENT_DISPLAY">

<init>
<caption>Istwerte</caption>

<control name = "label1" xpos = "36" ypos = "56" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[0]" />
<control name = "label2" xpos = "36" ypos = "76" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[1]" />
<control name = "label3" xpos = "36" ypos = "96" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[2]" />
<control name = "label4" xpos = "36" ypos = "116" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[3]" />
<control name = "label5" xpos = "36" ypos = "136" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[4]" />

<control name = "edit1" xpos = "80" ypos = "56" refvar="nck/Channel/
GeometricAxis/actProgPos[0]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit2" xpos = "80" ypos = "76" refvar="nck/Channel/
GeometricAxis/actProgPos[1]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit3" xpos = "80" ypos = "96" refvar="nck/Channel/
GeometricAxis/actProgPos[2]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit4" xpos = "80" ypos = "116" refvar="nck/Channel/
GeometricAxis/actProgPos[3]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit5" xpos = "80" ypos = "136" refvar="nck/Channel/
GeometricAxis/actProgPos[4]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
```

xmldial.xml

```
<control name = "edit11" xpos = "210" ypos = "56" refvar="nck/Channel/
GeometricAxis/progDistToGo[0]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit12" xpos = "210" ypos = "76" refvar="nck/Channel/
GeometricAxis/progDistToGo[1]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit13" xpos = "210" ypos = "96" refvar="nck/Channel/
GeometricAxis/progDistToGo[2]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit14" xpos = "210" ypos = "116" refvar="nck/Channel/
GeometricAxis/progDistToGo[3]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit15" xpos = "210" ypos = "136" refvar="nck/Channel/
GeometricAxis/progDistToGo[4]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>

</init>

<paint>
<text xpos= "36" ypos="30">Name</text>
<text xpos= "80" ypos="30">Current pos</text>
<text xpos= "210" ypos="30">Prog. pos</text>

</paint>
</form>

.....
```

Dialog form section "R Parameters"



xmldial.xml

```
.....

<!-- ************************************************************
Menu R- Parameter
-->

<menu name ="MENU_R_PARAMETER">
<OPEN_FORM name = "R-Parameter" />

<softkey POSITION="16">
<caption>Back</caption>
<navigation>MAIN</navigation>
</softkey>

</menu>

<form name = "R-Parameter">

<init>
<DATA_ACCESS type="true" />
<caption>R - Parameter</caption>

<control name = "edit1" xpos = "322" ypos = "34" refvar="nck/Channel/
Parameter/R[1]" />
<control name = "edit2" xpos = "322" ypos = "54" refvar="nck/Channel/
Parameter/R[2]" />
<control name = "edit3" xpos = "322" ypos = "74" refvar="nck/Channel/
Parameter/R[3]" />
<control name = "edit4" xpos = "322" ypos = "94" refvar="nck/Channel/
Parameter/R[4]" />
<control name = "edit5" xpos = "322" ypos = "114" refvar="nck/Channel/
Parameter/R[5]" />
<control name = "edit6" xpos = "322" ypos = "134" refvar="nck/Channel/
Parameter/R[6]" />
<control name = "edit7" xpos = "322" ypos = "154" refvar="nck/Channel/
Parameter/R[7]" />
<control name = "edit8" xpos = "322" ypos = "174" refvar="nck/Channel/
Parameter/R[8]" />
<control name = "edit9" xpos = "322" ypos = "194" refvar="nck/Channel/
Parameter/R[9]" />
<control name = "edit10" xpos = "322" ypos = "214" refvar="nck/Channel/
Parameter/R[10]" />
<control name = "edit10" xpos = "322" ypos = "214" refvar="plc/mb170" />
</init>

<paint>
<text xpos = "23" ypos = "34">R - Parameter 1</text>
<text xpos = "23" ypos = "54">R - Parameter 2</text>
<text xpos = "23" ypos = "74">R - Parameter 3</text>
<text xpos = "23" ypos = "94">R - Parameter 4</text>
<text xpos = "23" ypos = "114">R - Parameter 5</text>
<text xpos = "23" ypos = "134">R - Parameter 6</text>
<text xpos = "23" ypos = "154">R - Parameter 7</text>
<text xpos = "23" ypos = "174">R - Parameter 8</text>
<text xpos = "23" ypos = "194">R - Parameter 9</text>
```

xmldial.xml

```
</paint>
</form>

</DialogGui>
```

3.5 Language dependency

Language-dependent texts are used for:

- Softkey labels
- Headers
- Help texts
- Any other texts

The language-dependent texts are stored in text files.

Note

You will need to perform the following steps when using these text files:

- Make them available in the required languages.
- Transfer them into the relevant language directories of the control system.

3.6 XML diagnostics

The system provides the Easy XML diagnostic function for diagnosing and finding script errors.

The use of the diagnostic function checks the XML syntax and runs through all the scripts belonging to the project. To do this, functions, menus, forms and the existence and validity of variables are also checked. The errors found are listed.

The Easy XML diagnostics function can be activated either by an attribute in the **DialogGui** tag or by Display machine data.

Activate easy XML diagnostics

Example:

```
<DialogGui diagnose="true" >
...
...
</DialogGui >
```

or

Display machine data:

MD9113 $MM_EASY_XML_DIAGNOSE		Diagnostics and correction help support for Easy XML scripts
= 0	No diagnostics active	
= 1	Syntax check active	
= 0x100	The messages are additionally stored in the error_log.txt file.	

Diagnostics using trace outputs

Trace outputs can be integrated into a script using tag **debug**. Data is only saved if attribute **debug_msg** with the value **on** is specified in tag **DialogGui**.

Softkey function overview

Dialog	Softkey	Function
Main menu "EasyXML diagnostics"	Start script	The loaded program is started directly.
	Start check	The syntax check is being started. All accrued messages can be seen. Renewed checking is available.

Dialog	Softkey	Function
"Errors" and "Warnings" dialogs	Errors	The result are displayed, sorted according to errors and warnings.
	Warnings	
	Go to error	If errors or warnings are found, the softkey is displayed. The softkey opens the marked XML file from the error list for editing. The cursor is automatically placed on the erroneous row.
	Check result	All accrued messages can be seen.
"Document" dialog	Save	Changes in the XML file are being saved.
	Cancel	The XML file is closed unchanged. The check result is displayed again.

3.7 XML identifier

3.7.1 General structure

Structure and instructions of the script file for dialog configuration

All dialog configurations should be stored in the **DialogGui** tag.

```
<DialogGui>
…
</DialogGui>
```

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<DialogGui>
…
<FORM name ="Hello_World">
<INIT>
<CAPTION>Hello World</CAPTION>
</INIT>
...
</FORM>

</DialogGui>
```

Instructions

The language offers the following instructions for executing conditional instructions and loop controls:

- For loop
- WHILE loop
- Do while loop
- Conditional processing
- Switch and Case instructions
- Operator controls in a dialog form
- Softkey descriptions
- Define variables

A detailed description of the instructions can be obtained in section "Instruction/identifier descriptions (Page 45)".

3.7.2 Instruction/identifier descriptions

The following **XML tags** are defined for generating dialogs and menus, and for executing program sequences:

Note

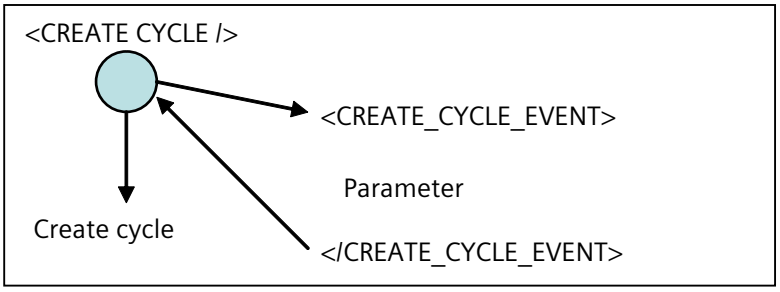
Attribute values that are in quotation marks "<...>" should be replaced by the currently used expressions.

Example:
<DATA_LIST action="read/write/append" id="<list name>">
is programmed as follows:

<DATA_LIST action="read/write/append" id="my datalist">

When copying multi-line XML tags and examples, care must be taken that the markups are not separated by unwanted line breaks.

Tag identifier	Meaning
BREAK	Conditional cancellation of a loop.
CONDITION	The tag must be programmed in one line or, in the case of multi-line notation, the conditions must be specified separately from the tag name. **Example:** `<if>` `<condition>test <= 2</condition>` `...` or `<if>` `<condition>` `test <= 2` `</condition>` `...`
CONTROL	The tag is used to generate control elements. The description is provided in Chapter "Generating softkey menus and dialog forms (Page 76)".
CONTROL_RESET	The tag enables one or more control components to be restarted. **Syntax:** `<CONTROL_RESET resetnc="TRUE" />` **Attributes:** • **RESETNC = "TRUE"** The NC component is restarted • **RESETDRIVE = "TRUE"** The drive components are restarted.

Tag identifier	Meaning
CREATE_CYCLE_EVENT	If the parser starts to process the tag CREATE_CYCLE, initially, the message <CREATE_CYCLE_EVENT> is sent to the active form. This message can be used for preparing the cycle parameters, before the parser generates the NC operation from the parameter list and the generation rule.  **Syntax:** `<CREATE_CYCLE_EVENT>` `…` `…` `</CREATE_CYCLE_EVENT>` **Example:** `<SOFTKEY_OK>` `…` `<CREATE_CYCLE />` `…` `…` `<SOFTKEY_OK>` `…` `…` `<FORM>` `<NC_INSTRUCTION>MY_CYCLE($P1, $P2)</ NC_INSTRUCTION >` `…` `…` `<CREATE_CYCLE_EVENT>` `<type_cast name="P1" type="int"/>` `<OP> P1 = P1 * 150 </OP>` `</CREATE_CYCLE_EVENT>` `…` `…` `</FORM>`

Tag identifier	Meaning
DATA	The tag allows writing to NC, PLC, GUD and drive data. Chapter "Component addressing (Page 140)" describes how addresses are formed. **Attribute:** • **name** Variable address **Tag value:** A constant, **a variable, or a term** is expected as tag value. **Syntax:** <DATA name="<variable name>"> value </DATA> <DATA name="<variable name>"> <term> </DATA> **Example:** <DATA name = "plc/mb170"> 1 </DATA> … <LET name = "tempVar"> 7 </LET> <!— the contents of the local variables "tempVar" are written to bit memory byte 170 → <DATA name = "plc/mb170">$tempVar</DATA>
DATA continued	**Example:** Calculation of a term. The result is assigned to the specified variable. <let name="a" >#f</let> <let name="b" >7</let> <data name = "b"> a & b</data> <let name="c" type="string"></let> <data name = "c"> _T" test" + _T" and test"</data>

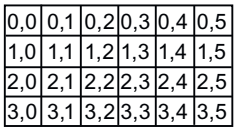
Tag identifier	Meaning
DATA_LIST	The tag enables the listed drive and machine data to be saved or restored. Addresses are listed in lines. Chapter "Component addressing (Page 140)" describes how addresses are formed. Up to 20 temporary data lists can be created. **Attributes:** • **action** *read* – the values of the listed variables are stored in a temporary memory *append* – the values of the listed variables are added to an existing list *write* – the backed up values are copied to the relevant machine data • **id** The identifier identifies the temporary memory **Syntax:** `<DATA_LIST action="<read/write/append>"  id="<list name>">` `NC/PLC address compilation` `</DATA_LIST>` **Example:** `<DATA_LIST action ="read" id="<name>">` `nck/channel/parameter/r[2]` `nck/channel/parameter/r[3]` `nck/channel/parameter/r[4]` `$MN_USER_DATA_INT[0]` … `</ DATA_LIST>` `<DATA_LIST action ="write" id="<name>" />`
DIALOGGUI	This tag represents the root tag for the Easy XML function. All of the instructions encompassed are edited using the Easy XML parser. **Attributes:** The listed attributes can be optionally specified in order to activate central functions: • **diagnose** - value **true** activates the XML diagnostics • **debug_msg** - value "on" - trace messages are saved **More information** is provided in Chapter "XML diagnostics (Page 42)" • **textfile** - name of the text file to be used • **textcontext** - text context to be used only valid in conjunction with the attribute textfile • **textfilelist** - allows a list with different text files to be loaded **More information** is provided in Chapter "Project-specific text files (Page 254)"
DIALOGGUI continued	• **restoresession** - value **true** If the Easy XML project is selected again, the parser opens the menu last selected and the last selected form. This behavior is retained until the HMI is restarted. **More information** is provided in Chapter "Restoring the session (Page 259)"
DEBUG_MSG	The attribute controls the storage of trace outputs. For a description, see Chapter "Generating softkey menus and dialog forms (Page 76)" under tag DEBUG. If the value **on** is assigned, the parser stores all outputs in one file. This can result in slower script execution. The value **off** is set by default.

Tag identifier	Meaning
DO_WHILE	Do while loop `DO` `  Instructions` `WHILE (Test)` **Syntax:** `<DO_WHILE>` `Instructions` `…` `<CONDITION>…</CONDITION>` `</DO_WHILE>` The Do While loop comprises a block of instructions and a condition. The code in the instruction block is executed first, then the condition is applied. If the condition is true, the function executes the code section again. This is continually repeated until the condition is false. **Example:** `<DO_WHILE>` `<DATA name = "PLC/qb11"> 15 </DATA>` `<CONDITIION> "plc/ib9" == 0 </CONDITION>` `</DO_WHILE>`
DYNAMIC_INCLUDE	The tag includes an XML script file. Contrary to the INCLUDE tag, read-in is first only realized when executing the corresponding operation. For large projects, the use of the tag reduces the load time of the customer area and/or the cycle support. Further, the average level of resources is reduced, as not all of the dialogs are always called during a session. **Syntax:** <DYNAMIC_INCLUDE src="path name"/> **Example:** `<SOFTKEY POSITION="3">` `  <CAPTION>MY_MENU</CAPTION>` `  <DYNAMIC_INCLUDE src="f:\appl\sk3_script.xml"/>` `  <NAVIGATION>MY_MENU</NAVIGATION>` </SOFTKEY>
ELSE	Instruction for situations where the condition has not been met (**IF, THEN, ELSE**)

Tag identifier	Meaning
FOR	For loop `for (initialization; test; continuation) instruction(s)` **Syntax:** `<FOR>` `<INIT>…</INIT>` `<CONDITION>…</CONDITION>` `<INCREMENT>…</INCREMENT>` `Instructions` `…` `</FOR>` The For loop is executed as follows: 1. Evaluation of the expression **initialization** (INIT). 2. Evaluation of the expression **test** (CONDITION) as a Boolean expression. If the value is false, the For loop is exited. 3. Execution of the following instructions. 4. Evaluation of the expression **continuation** (INCREMENT). 5. Continue with 2. All the variables within the INIT, CONDITION, and INCREMENT branches are declared and initialized outside the For loop. **Example:** `<LET name = "count">0</LET>` `<FOR>` `<INIT>` `<OP> count = 0</OP>` `</INIT>` `<CONDITION> count <= 7 </CONDITION>` `<INCREMENT>` `<OP> count = count + 1 </OP>` `</INCREMENT>` `<OP> "plc/qb10" = 1+ count </OP>` `</FOR>`
FORM	The tag contains the description of a user dialog. The description is provided in Chapter "Generating softkey menus and dialog forms (Page 76)".
HMI_RESET	The tag triggers an HMI restart. Interpretation is stopped after this instruction.

Tag identifier	Meaning
IF	Conditional instruction (**IF, THEN, ELSE**) The THEN and ELSE tags are enclosed in the IF tag. The condition that is executed in the CONDITION tag follows the IF tag. The further processing of the instructions depends upon the result of the operation. If the function result is true, then the THEN branch is executed and the ELSE branch is skipped. If the result of the function is false, the parser executes the ELSE branch. **Syntax:** `<IF>` `<CONDITION> Condition != 7 </CONDITION>` `<THEN>` `Instruction for the case: condition fulfilled` `</THEN>` `<ELSE>` `Instruction for the case: Condition not fulfilled` `</ELSE>` `</IF>` **Example:** `<IF>` `<CONDITION> "plc/mb170" != 7 </CONDITION>` `<THEN>` `<OP> "plc/mb170" = 7 </OP>` … `</THEN>` `<ELSE>` `<OP> "plc/mb170" = 2 </OP>` … `</ELSE>` `</IF>`
IF continued	If only local variables are used in the condition, then the condition can be specified as attribute in the IF tag. **Example:** `<let name="var1">1</let>` `<if condition="var == 1">` `<then>` `...` `...` `</then>` `</if>` **Note:** Controls that are not coupled with a reference variable are assigned the integer data type. Exceptions: Controls, type **imagebox** and **multiline**, are assigned data type **string**.

Tag identifier	Meaning
INCLUDE	The instruction includes an XML description. (see also DYNAMIC_INCLUDE in this table) **Attribute:** • **src** Contains the path name. **Syntax:** `<?INCLUDE src="<Path name>" ?>`

Tag identifier	Meaning
LET	The instruction creates a local variable under the specified name. **Fields:** Using the attribute **dim** (dimension) single or two-dimensional fields can be created. The field index addresses the individual field elements. For a two-dimensional field, initially the line index is specified and then the column index. • One-dimensional field: Indices 0 to 4  • Two-dimensional field: Index line 0 to 3 and index column 0 to 5  **Attributes:** • **name** Variable name • **type** The variable type can be an integer (INT), unsigned integer (UINT), double (DOUBLE), float (FLOAT), string (STRING), or STRUCT. It is also possible to use a structure created by **typedef** as a variable type (see tag identifier TYPEDEF). If there is no type instruction specified, the system creates an integer variable. `<LET name = "VAR1" type = "INT" />` • **permanent** If the attribute is set to **true**, then the variable value is saved permanently. This attribute is only effective for a global variable. • **poweroff** If the value of the attribute is **true**, the values are retained until the control is switched off. **More information** is provided in Chapter "Restoring the session (Page 259)" • **dim** The following number of field elements must be specified. For a two-dimensional field, the second dimension is specified after the first dimension separated by a comma. A field element is accessed via the field index, which is specified in square brackets after the variable name. name[index] or name[row,column] – One-dimensional field: dim="<number of elements>" – Two-dimensional field: dim="<number of lines>,<number of columns>" Non-initialized field elements are pre-assigned with "0".

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5

Tag identifier	Meaning
LET continued	**Example:** One-dimensional field: `<let name="array" dim="10"></let>` Two-dimensional field: `<let name="list_string" dim="10,3" type="string"></let>` **Pre-assignment:** A variable can be initialized with a value. `<LET name = "VAR1" type = "INT"> 10 </LET>` If values comprising NC or PLC variables are saved in a local variable, the assignment operation automatically adapts the format to that of the variables which have been loaded. • Pre-assignment for a string variable: Texts containing more than one line can be assigned to a string variable if the formatted text is transferred as a value. If a line is to end with a line feed <LF> (line feed) , the characters "\\n" should be added at the end of the line. `<LET name = "text" type = "string"> F4000 G94\\n` `G1 X20\\n` `Z50\\n` `M2\\n` `</LET>>` **`Fields (Arrays):`** `<let name="list" dim="10,3">` `{1,2,3},` `{1,20}` `</let>` `<let name="list_string" dim="10,3" type="string">` `{"text 10","text 11"},` `{"text 20","text 21"}` `</let>` **Assignment:** Values made up of the machine data or subroutines can be assigned to a variable using the assignment operation "=". A variable remains valid until the end of the higher-level XML block. Variables which are to be available globally should be created directly after the **DialogGUi** tag. The following must be observed for a dialog box: • The message processing opens the corresponding tag. • The tag is closed after the message has been executed. • All variables within the tag are deleted when closing.

Tag identifier	Meaning
LET continued	**Variable type struct**: This variable type contains a composition of variables that can be addressed using the structure name. A structure can contain all variable types and structures. Within the structure, a variable is declared with the "element" tag. The attributes of the tags and the initialization correspond to the attributes and initialization of the let instruction. `<let name="name" type="struct">` `<element name="<Name>" type="<Variable type>" />` … … … `</let>` A structure element can be assigned a default value which is used as initial value when creating a variable. **For more information**, see section "Initialization of structures" This value is overwritten if an initial value is specified when the variable is created.
LET continued	Access to a variable of the structure is via the structure name and variable name. Both names are separated by a point operator. **Syntax:** `<op>` `Structure_name.variable_name = value;` `</op>` **Example:** `<let name="info" type="struct">` `  <element name="id" type="int" />` `  <element name="name" type="string" />` `  <element name="phone" type="string" />` `</let>` `<op>` `  info.id = 1;` `  info.name = _T"my name";` `  info.phone = _T"0034 45634";` `</op>`

Tag identifier	Meaning
LET continued	**Initialization of structures:** Structures can be initialized when the variables are created by specifying an initial value for each structure element. In an array of structures, each structure must be separated from the others by braces. **Example:** `<typedef name="StructRect" type="struct" >` `  <element name="left" type="int">22</element>` `  <element name="top" type="int">122</element>` `  <element name="right" type="int">220</element>` `  <element name="bottom" type="int">56</element>` `</typedef>` `<let name="rect" type="StructRect" >` With the creation of the **rect** variables, the structure elements are initialized with the following values: `rect.left= 22` `rect.top = 122` `rect.right = 220` `rect.bottom = 56` If initial values are assigned to a structure, they overwrite the default values. `<let name="rect" type="StructRect" >` `  {11,12}` `</let>` With the creation of the rect variables, the structure elements are initialized with the following values: `rect.left= 11` `rect.top = 12` `rect.right = 220` `rect.bottom = 56`

Tag identifier	Meaning
LET continued	**Nested structures:** Structures are variables of a variable type defined by the user and can thus be used as structure elements in a structure. The elements are initialized according to the described rule. **Example:** `<typedef name="StructRectRect" type="struct" >` `  <element name="r1" type="StructRect">77, 99, 232, 23</element>` `  <element name="r2" type="StructRect">77, 88, 45, 12</element>` `</typedef>` `<let name="rect_rect_array" type="StructRectRect" >` With the creation of the **rect_rect_array** variables, the structure elements are initialized with the following values: `r1.left= 77` `r1.top = 99` `r1.right = 232` `r1.bottom = 23` `r2.left= 77` `r2.top = 88` `r2.right = 45` `r2.bottom = 12` `<let name="rect_rect_array1" type="StructRectRect" >` `  {{11,12,13,14},{111,188,199,114}}` `</let>` With the creation of the **rect_rect_array** variables, the structure elements are initialized with the following values: `r1.left= 11` `r1.top = 12` `r1.right = 13` `r1.bottom = 14` `r2.left= 111` `r2.top = 188` `r2.right = 199` `r2.bottom = 114`
LET continued	**Initialization of global string variables:** If you use a text identifier to initialize a global string variable, the identifier and the text to be replaced are expected in the standard text file **oem_xml_screens_xxx.ts** or in the text file specified in the **DialogGUI** tag. When loading the application, the text of the active language replaces the text identifier. If a language change is performed while the application is active, no new initialization of the global string variable is performed. The text can be exchanged in the **LANGUAGE_CHANGED** message.
LOCK_OPERATING_AREA	The operating area switchover is locked. The operating area switchover lock is withdrawn with the **UNLOCK_OPERATING_AREA** tag. **Syntax:** `<LOCK_OPERATING_AREA />`

Tag identifier	Meaning
MSG	The operator component shows the message which is indicated in the tag. If an alarm number is used, the dialog box displays the text which is saved for the number. **Example:** `<MSG text ="my message" />`
MSGBOX	The instruction opens a message box whose return value can be used for branching. **Syntax:** `<MSGBOX text="<Message>" caption="<caption>" retvalue="<variable name>" type="<button type>" />` **Attributes:** • **text** Text • **caption** Header • **retvalue** Name of the variables to which the return value is copied: 1 – OK 0 – CANCEL • **type** Acknowledgment options: "BTN_OK" "BTN_CANCEL" "BTN_OKCANCEL" If an alarm number is used for the **"text"** or **"caption"** attribute, the message box displays the text which is saved for the number. **Example:** `<MSGBOX text="Test message" caption="Information" retvalue="result" type="BTN_OK" />`

Tag identifier	Meaning
OP	The tag executes the specified operations. The operations listed in Chapter "Operators (Page 74)" can be executed. For the purpose of accessing the NC, PLC and drive data, the complete variable name should be placed in **quotation marks**. Chapter "Component addressing" (Page 140) describes how addresses are formed. `PLC: "PLC/MB170"` `NC: "NC/Channel/…"` **Example:** `<LET name = "tmpVar" type="INT"> </LET>` `<OP> tmpVar = "plc/mb170" </OP>` `<OP> tmpVar = tmpVar *2 </OP>` `<OP> "plc/mb170" = tmpVar </OP>` More than one equation can be used within an operation tag. A semicolon marks the end of the instruction. **Example:** `<op>` `  x = x+1;` `  y = y+1;` `</op>` **Character string processing:** The operation instruction is able to process character strings and assign the results to the string variable specified in the equation. The identifier _T should be placed at the start as a means of identifying text expressions. Formatting of variable values is also possible. The identifier _F should be placed at the start of the formatting regulation, followed by the format instruction. The address is then specified for the variable. **Example:** `<LET name="buffer" type="string"></LET>` … … … `<OP> buffer = _T"unformatted value R0= " + "nck/Channel/Parameter/R[0]" + _T" and " + _T"$$85051" + _T" formatted value R1 " + _F %9.3f"nck/Channel/Parameter/R[1]" </OP>`

Tag identifier	Meaning
OPERATION	**Operation** Instructions can be moved within an equation. **Move left "<<" operator** The << function moves bits to the left. You can specify the value and the number of move increments directly or with a variable. If the limit of the data format is reached, the bits will be moved beyond the limit without an error message being issued. **Execution rule** Execution from left to right; '+' and '-' have priority over '<<' and '>>' **Example:** `<op> idx = idx << 2 </op>` `<op> idx = 3 + idx << 2 </op>` **Move right ">>" operator** The >> function moves bits to the right. You can specify the value and the number of move increments directly or with a variable. If the limit of the data format is reached, the bits will be moved beyond the limit without an error message being issued. **Execution rule** Execution from left to right; '+' and '-' have priority over '<<' and '>>' **Example:** `<op> idx = idx >> 2 </op>` `<op> idx = 3+idx >> 2</op>`
PASSWORD	This tag is used to unlock additional units for the "EasyExtend" function. The specified character string is written to the specified reference variable and must be processed in the PLC user program. **Syntax:** `<PASSWORD refVar ="<variable name>" />` **Attributes:** • **refVar** Name of the reference variable • **text** Specifying an attribute replaces the default text (optional) **Example:** `<PASSWORD refvar="plc/mw107" />` **Example optional:** `<password refVar = "plc/MD108" text="Password" />`
POWER_OFF	A message prompts the operator to switch the machine off. The message text is permanently saved in the system.

Tag identifier	Meaning
PRINT	The tag outputs a text in the dialog line or copies the text to the variable specified. If the text contains formatting identifiers, the variable values are inserted at the appropriate places. **Syntax:** `<PRINT name="Variable name" text="text %Formatting"> Variable, … </PRINT>` `<PRINT text="text %Formatting"> Variable, … </PRINT>` **Attributes:** • **name** Name of the variable where the text is to be stored (optional) • **text** Text **Formatting:** The character "%" causes the variable specified as the value to be formatted. %[Flags] [Width] [.decimal places] type • Flags: Optional character for defining output formatting: – Right-justified or left-justified ("-" for left-justified) – Add leading zeros ("0") – Fill with blanks • Width: The argument defines the minimum output width for a non-negative number. If the value to be output has fewer places than the argument defined, the missing spaces are filled with blanks. • Decimal places: With floating-point numbers, the optional parameter defines the number of decimal places. • Type: The type character defines which data formats are transferred for the print instruction. These characters need to be specified. – d: Integer value – f: Floating-point number – s: String

Tag identifier	Meaning
PRINT continued	**Values:** Number of variables whose values are to be inserted into the text. The variable types must match the corresponding type identifier for the formatting instruction and must be separated from one another with a comma. **Example:** Output of a text in the information line `<PRINT text="Infotext" />` Output of a text with variable formatting `<LET name="trun_dir"></LET>` `<PRINT text="M%d">trun_dir</PRINT>` Output of a text in a string variable with variable formatting `<LET name="trun_dir"></LET>` `<LET name="str" type="string" ></LET>` `<print name="str" text="M%d ">trun_dir</print>`
PROGRESS_BAR	The tag opens or closes a progress bar. The bar is displayed below the application window. **Syntax:** `<PROGRESS_BAR type="<true/false>"> value </ PROGRESS_BAR>` **Attributes:** • **type =** "TRUE" - opens the progress bar • **type** = "FALSE" - closes the progress bar • **min** (optional) – minimum value • **max** (optional) – maximum value **Value:** • **Value** Percentage position of the bar **Example:** `<PROGRESS_BAR type="true" min="0" max="101" >20< /` `PROGRESS_BAR>......<PROGRESS_BAR >50< /` `PROGRESS_BAR>......<PROGRESS_BAR type="false" >100< /PROGRESS_BAR>`

Tag identifier	Meaning
SEND_MESSAGE	The tag sends a message with two parameters to the active form, which is processed in the tag message. **Syntax:** `<SEND_MESSAGE>p1, p2</SEND_MESSAGE>` **Example:** `<SOFTKEY POSITION="3">` `  <caption>Set%nParameter</caption>` `  <send_message>1, 0</send_message>` `</SOFTKEY>` `<FORM>` … … `<MESSAGE>` `<SWITCH>` `<CONDITION>$message_par1</CONDITION>` `<CASE value="1">` … … `</CASE>` `</SWITCH>` `</MESSAGE>` … … `</FORM>`
SHOW_CONTROL	The visibility of a control can be controlled using the tag. **Syntax:** `<SHOW_CONTROL name="<name>" type="<type>" />` **Attributes:** • **name** Name of the control • **type =** "TRUE" - control becomes visible • **type** = "FALSE" - control becomes invisible (hidden) **Example:** `<SHOW_CONTROL name="myEditfield" type="false" />` … … `<SHOW_CONTROL name="myEditfield" type="true" />`

Tag identifier	Meaning
SLEEP	The tag interrupts script execution for the specified period. The interruption time is obtained from the transferred value multiplied by the time base of 50 ms. **Syntax:** `<SLEEP value="Interruption time" />` **Example:** Wait time, 1.5 s `<SLEEP value="30" />`
STOP	Interpretation is canceled at this point.
SWITCH	The **SWITCH** instruction describes a multiple choice. A term is evaluated once and compared with a number of constants. If the expression matches the constants, the instructions are executed within the **CASE** instruction. The **DEFAULT** instruction is executed when none of the constants match the expression. **Syntax:** `<SWITCH>` `<CONDITION> Value </CONDITION>` `<CASE value="<Constant 1>">` `Instructions` `…` `</CASE>` `<CASE value="<Constant 2>">` `Instructions` `…` `</CASE>` `<DEFAULT>` `Instructions` `…` `</DEFAULT>` `</SWITCH>`

Tag identifier	Meaning
SWITCHTOAREA	The SWITCHTOAREA tag changes from the Customer area into the specified operating area. The parameter is specified as an attribute value. **Syntax:** `<switchToArea name="area" args="argument "/>` **Attributes:** **name** The following names are declared for the operating areas: • **AreaMachine** - machine • **AreaParameter** - parameters • **AreaProgramEdit** - editor • **AreaProgramManager** - Program Manager • **AreaDiagnosis** - diagnostics • **AreaStartup** - setup **args** (reserved) Runtime arguments can be passed to the operating area to activate dialogs. **Example:** `<switchToArea name="AreaMachine" />`
SWITCHTODYNAMICTARGET	If the previous dialog is defined as a dynamic jump destination, the SWITCHTODYNAMICTARGET tag will activate this dialog and end script execution. **Syntax:** `<switchToDynamicTarget />`
THEN	Instruction if the condition has been fulfilled (**IF, THEN, ELSE**)
TYPE_CAST	The tag converts the data type of a local variable. **Syntax:** <type_cast name="variable name" type=" new type" /> **Attributes:** • **name** Name of the variable • **type** The new data type is assigned to the variable. • **convert** The new data type is assigned to the variable. The variable value is also converted to the new data type.

Tag identifier	Meaning
TYPEDEF	A new identifier for a data type can be defined with this tag. This has the benefit for the structure definitions that the data type can be defined once and then used as a data type in a LET instruction. The identifier and type are expected as attributes. The parser supports only the specification of structure definitions. In the type definition, a variable is declared with the "element" tag. The attributes of the tag correspond to the attributes of the let instruction. `<typedef name="<identifier>" type="struct">` `<element name="<name>" type="<variable type>" />` … … … `</typedef>` After definition, the identifier can be used as a data type for the LET instruction. `<let name="<variable name>" type="<identifier>"></let>` **Example**: `<typedef name="my_struct" type="struct">` `<element name="id" type="int" />` `<element name="name" type="string" />` `<element name="phone" type="string" />` `</typedef>` `<let name="info" type="my_struct"></let>` … … `<op>` `info.id = 1;` `info.name = _T"my name";` `info.phone= _T"0034 45634";` `</op>`

Tag identifier	Meaning
TYPEDEF continued	Some predefined functions expect variables of structure type RECT, POINT, or SIZE as the call parameter. These structures are defined in the file struct_def.xml. **More information** is provided in Chapter "Creating the file struct_def.xml (Page 138)" **RECT:** `<typedef name="StructRect" type="struct" >` `  <element name="left" type="int">0</element>` `  <element name="top" type="int">0</element>` `  <element name="right" type="int">0</element>` `  <element name="bottom" type="int">0</element>` `</typedef>` **POINT:** `<typedef name="StructPoint" type="struct" >` `  <element name="x" type="int">0</element>` `  <element name="y" type="int">0</element>` `</typedef>` **SIZE:** `<typedef name="StructSize" type="struct" >` `  <element name="width" type="int">0</element>` `  <element name="height" type="int">0</element>` `</typedef>`
UNLOCK_OPERATING_AREA	Withdrawal of operating area switchover lock **Syntax:** `<UNLOCK_OPERATING_AREA />`
WAITING	The tag waits for the component to undergo a hot restart after an NC or drive reset. **Attributes:** • **WAITINGFORNC** = "TRUE" - the system waits for the NC to restart • **WAITINGFORDRIVE** = "TRUE" - the system waits for the drives to restart **Syntax:** `<WAITING WAITINGFORNC = "TRUE"/>` **Example:** ... `<CONTROL_RESET resetnc = "true" resetdrive = "true"/>` `<WAITING waitingfornc = "true" waitingfordrive = "true" />` ...

Tag identifier	Meaning
WHILE	WHILE loop `WHILE (Test)` `  Instruction` **Syntax:** `<WHILE>` `<CONDITION>…</CONDITION>` `Instructions` … `</WHILE>` The While loop executes a sequence of instructions repeatedly while a condition is met. This condition is tested before the sequence of instructions is executed. **Example:** `<WHILE>` `<CONDITION> "plc/ib9" == 0 </CONDITION>` `<DATA name = "PLC/qb11"> 15 </DATA>` `</WHILE>`
XML_PARSER	The "XML_PARSER" tag can be used to parse XML files. The parser interprets an XML file and calls defined call-back functions. Each call-back function belongs to a predefined event. The programmer can process the XML data within this function. **Predefined events:** • start document The parser opens the document and starts parsing. • end document The parser closes the document. • start element The parser has found an element and creates a list with all attributes and attribute values. These lists are forwarded to the call-back function. • end element The end of the element has been found. • characters The parser forwards all characters of an element. • error The parser has detected a syntax error. When an event occurs, the parser calls the callback function and checks the return value of the function. If the function returns the value "true", the parser continues the process.

Tag identifier	Meaning
XML_PARSER continued	**Interfaces** The value of the **name** attribute contains the path to the XML file. To assign events to the call-back functions, the following properties must be specified: **Standard** startElementHandler endElementHandler charactersHandler **Optional** errorHandler documentHandler The value of an attribute defines the name of the call-back function. **Example:** `<XML_PARSER name="f:\appl\xml_test.xml">` `<!-- standard handler -->` `<property startElementHandler="startElementHandler" />` `<property endElementHandler="endElementHandler" />` `<property charactersHandler="charactersHandler" />` `<!-- optional handler -->` `<property errorHandler="errorHandler" />` `<property documentHandler="documentHandler" />` `</XML_PARSER>`

Tag identifier	Meaning
XML_PARSER continued	The parser also supplies variables so that the call-back functions can access the event data. **startElementHandler:** **Function parameters** tag_name - tag name num - number of attributes found **System variables** $xmlAttribute String array with the number of elements indicated by num. $xmlValue String array that contains the 0-num attribute value range. **Example:** `<function_body name="startElementHandler" return="true" parameter="tag_name, num">` `<print text="attribute name %s"> $xmlAttribute[0]</print>` `<print text="attribute value %s"> $xmlValue[0]</print>` `</function_body>` **endElementHandler:** **Function parameters** tag_name - tag name **Example:** `<function_body name="endElementHandler" parameter="tag_name">` `<print text="name %s"> tag_name </print>` `</function_body>`

Tag identifier	Meaning
XML_PARSER continued	**charactersHandler:** **System variables** $xmlCharacters String with data $xmlCharactersStart Always 0 $xmlCharactersLength Number of bytes **Example:** `<function_body name="charactersHandler" return="true" >` `<print text="chars %s"> $xmlCharacters </print>` `</function_body>` **documentHandler:** **Function parameters** state 1 start document, 2 end document **errorHandler:** **System variables** $xmlErrorString Contains the invalid line (system variable) **Example:** `<function_body name="errorHandler" >` `<print text="error %s">$xmlErrorString</print>` `</function_body>` **Call-back result:** $return If 1 (true), the parser continues parsing the file

3.7.3 Color coding

The color attribute uses the color coding scheme for the HTML language.

In terms of syntax, color specifications consist of the "#" (hash) character and six digits from the hexadecimal system, with each color represented by two digits.

R – Red

G – Green

B – Blue

#RRGGBB

Example:

color= "#ff0011"

Example colors:

Red	Green	Blue	Yellow	White	Black
#FF0000	#00FF00	#0000FF	#ffff00	#FFFFFF	#000000

3.7.4 Special XML syntax

Characters with special meanings in XML syntax have to be rewritten if they are to be displayed correctly by a general XML editor.

The following characters are affected:

Character	Notation in XML
<	<
>	>
&	&
"	"
'	'

3.7.5 HTML special characters

For the representation of characters or for the use of control characters, the HTML special character evaluation is offered.

The decimal and hexadecimal encodings of the code page Latin 1 can be used.

Example

Characters	Description	Notation decimal	Notation hexadecimal
"	Quotation marks	"	"
LF	Line feed

3.7.6 Operators

The operation instruction processes the following operators:

Operator	Meaning
=	Assignment
==	Equal to
<, <	Less than
>, >	Greater than
<=, <=	Less than or equal to
>=, >=	Greater than or equal to
|	OR operation in bits
||	Logic OR operation
&, &	AND operation in bits
&&, &&	Logic AND operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
!	Not
!=	Not equal to

Operation instructions are processed from left to right. It may make sense to place terms in parentheses under certain circumstances in order to define the priority for executing subterms.

3.7.7 System variables

System variables are variables available in every script, which are used for exchanging data between the parser and the script execution.

The following table provides an overview of the tags for which variables are automatically generated.

Variable name	Meaning	Valid in tag
$actionresult	Signals to the parser, whether the parser is to process the event.	KEY_EVENT
$focus_name	Contains the name of the field which has the input focus	FOCUS_IN INDEX_CHANGED EDIT_CHANGED
$focus_item_data	Contains the numerical value item_data, that was assigned to the field	
$gestureinfo	For finger gestures, the parser provides the gesture information in the variable and executes the gesture_event tag.	GESTURE_EVENT
$return	The variable transports the return value of a subfunction	FUNCTION_BODY
$message_par1 $message_par2	Contains the call parameters of the SEND_MESSAGE function	MESSAGE
$xmlAttribute	Contains a list of the tag attributes found	XML_PARSER startElementHandler
$xmlValue	Contains a list of the tag values found	
$xmlCharacters	Contains the data flow	XML_PARSER charactersHandler
$xmlCharactersStart	Contains the start index:	
$xmlCharactersLength	Contains the number of characters stored in the data flow	
$mouse_event.type $mouse_event.x $mouse_event.y $mouse_event.id $mouse_event.buttons $mouse_event.button	Structure to transfer the parameters of the mouse event	MOUSE_EVENT

3.7.8 Generating softkey menus and dialog forms

Dialog forms can only be integrated if there is a menu tag with the name "main" in the XML description. This tag is called by the system after the <CUSTOM> operating area has been activated. Additional dialog forms can be branched and activation of a dialog box defined within the tag.

```
<menu name= "MAIN">
<OPEN_FORM name= "main dialogue">
<softkey POSITION="1">
<caption>sub menu 1</caption>
<navigation>sub menu 1</navigation>
</softkey>
<softkey POSITION="8">
<caption>sub menu 8</caption>
<navigation>sub menu 8</navigation>
</softkey>
</menu>

<menu name= "sub menu 1">
<OPEN_FORM name= "dialogue 1">
</menu>

<menu name= "sub menu 8">
<OPEN_FORM name= "dialogue 8“>
</menu>
```

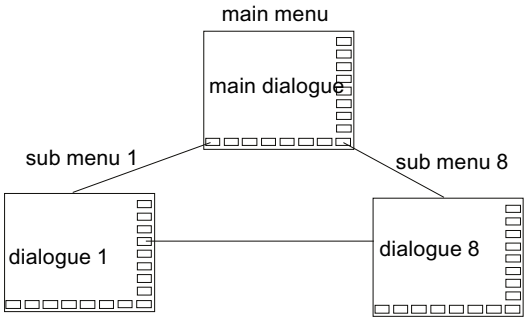
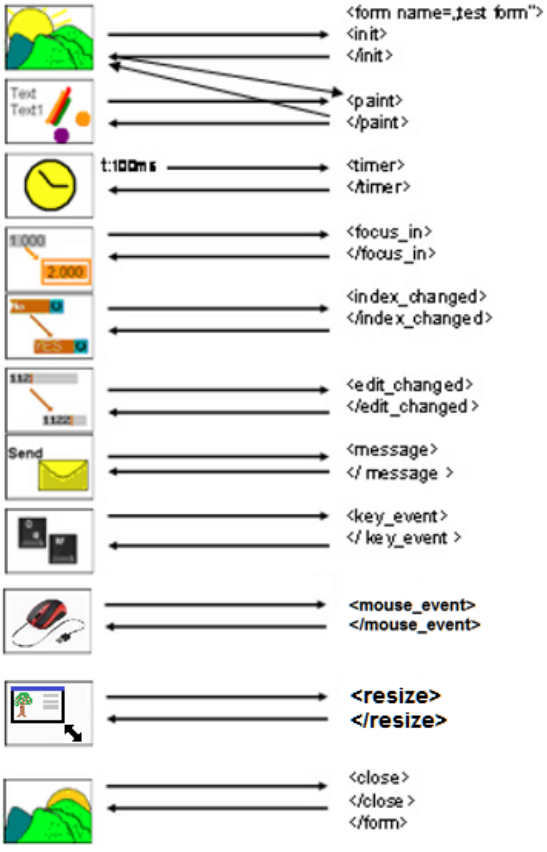


Figure 3-6 Menu structure

Tag identifier	Meaning
FORM	The tag contains the description of a user dialog. The relevant tags are described in the section on generating menus and dialog masks. **Syntax:** `<FORM name="<dialog name>" color="#ff0000">` **Attributes:** • **color** Background color of the dialog mask **More information** is provided in Chapter "Color coding (Page 72)" – Default white • **name** Identifier of the form • **type** Permissible value is *cycle* , which identifies a user cycle screen form • **xpos** X-position of the top left corner of the dialog box (optional) • **ypos** Y position of the top left corner (optional) • **width** Extension in the X direction (in pixels) (optional) • **height** Extension in the Y direction (in pixels) (optional)
FORM continued	With the AUTOSCALE_CONTENT attribute, you can define how the controls of a form will behave at different screen resolutions. By default, the controls are automatically adapted to the screen resolution already set. If you would like to control the positioning and dimensioning yourself, set value OFF for the attribute in the Form tag. **Attribute:** **autoscale_content** **Values:** • **on** The coordinates of the controls are automatically adapted to the screen resolution (default) • **off** The coordinates of the controls are used unchanged **Example:** `<form name="main_form" autoscale_content="off">` ... ... `</form>`

Tag identifier	Meaning
FORM continued	Attributes: • **textfile** The attribute allows a form-related text file to be specified. As standard text context, the system uses identifier EASY_XML. • **textcontext** The attribute allows an identifier to be specified for the text context to be used. This attribute is only valid in conjunction with attribute **textfile**.
FORM continued	In order to be able to use standard layouts or layouts that are stored in the easyscreen.ini file, use the attribute LAYOUT. The value that you enter is the name of the layout to be used. **Note:** It is only advisable to change the default settings for popup windows because for these the calling dialog box is not hidden. **Attribute:** **layout** **Syntax:** `<form name="Name of the form" layout="Name of the layout" >` … … `</form>` **Example:** `<form name="form0" layout="slstandardscreenlayout.` `SlStandardScreenLayout.LowerForm" >` … … `</form>` Besides the predefined screen form positions and dimensions, you can also store your own definitions in the easyscreen.ini file. Entry in easyscreen.ini: `[640x480]` `MyPanel = x:=0, y:=220, width:=340, height:=174` `[800x480]` `MyPanel = x:=0, y:=220, width:=420, height:=174` **Example:** `<form name="form0" layout="MyPanel" >` … … `</form>`

Tag identifier	Meaning
FORM continued	**Dialog messages:** • **INIT** • **PAINT** • **TIMER** • **CLOSE** • **FOCUS_IN** • **INDEX_CHANGED** • **EDIT_CHANGED** • **GESTURE_EVENT** • **KEY_EVENT** • **MESSAGE** • **MOUSE_EVENT** • **RESIZE** • **CHANNEL_CHANGED** • **LANGUAGE_CHANGED**
FORM continued	

Tag identifier	Meaning
FORM continued	**Syntax:** `<FORM name = "<dialog name>" color = "#ff0000">` **Example:** `<FORM name = "R-Parameter">` ` <INIT>` ` <DATA_ACCESS type = "true" />` ` <CAPTION>R - Parameter</CAPTION>` ` <CONTROL name = "edit1" xpos = "322" ypos = "34" refvar = "nck/Channel/Parameter/R[1]" />` ` <CONTROL name = "edit2" xpos = "322" ypos = "54" refvar = "nck/Channel/Parameter/R[2]" />` ` <CONTROL name = "edit3" xpos = "322" ypos = "74"` ` </INIT>` `  <PAINT>` ` <TEXT xpos = "23" ypos = "34">R - Parameter 1</TEXT>` ` <TEXT xpos = "23" ypos = "54">R - Parameter 2</TEXT>` ` <TEXT xpos = "23" ypos = "74">R - Parameter 3</TEXT>` ` </PAINT>` `</FORM>`
INIT	**Dialog box message** The tag is executed immediately after the dialog box is generated. All the input elements and "hotlinks" for the dialog form should be created here.

Tag identifier	Meaning
KEY_EVENT	Dialog message The tag KEY_EVENT can be integrated in the form to evaluate keyboard events. The system sends the MF2 keyboard code to the active form if the tag is available in a form. If the variable **$actionresult** is not set to zero, the system then subsequently processes the keyboard event. The keyboard code is provided in the variable **$keycode** as an integer value. **Example:** The character entered into the variable **exclude_key** should be filtered-out of the input stream. `<LET name="stream" type="string"/>` `<LET name="exclude_key" type="string"/>` `<FORM name = "keytest_form">` `  <INIT>` `  <CONTROL name = "p1" xpos = "120" ypos = "84" width ="200" refvar="stream" hotlink="true" />` `  <CONTROL name = "p2" xpos = "160" ypos = "104" width ="8" refvar="exclude_key" hotlink="true" />` `  </INIT>` `  <PAINT>` `  <text xpos = "8" ypos = "84">data stream</text>` `  <text xpos = "8" ypos = "104">exclude key</text>` `  </PAINT>` `  <KEY_EVENT>` `  <LET name="excl_keycode" type="string"/>` `  <OP>excl_keycode = exclude_key</OP>` `  <type_cast name="excl_keycode" type="int" />` `  <PRINT text="%d %d">$keycode, excl_keycode</PRINT>` `  <IF>` `  <CONDITION>$keycode == excl_keycode</CONDITION>` `  <THEN>` `  <op> $actionresult = 0</op>` `  </THEN>` `  </IF>` `  </KEY_EVENT>` `</FORM>`

Tag identifier	Meaning
MOUSE_EVENT	The tag can be linked into the script for processing mouse events. It is executed when the following activities have been performed with the mouse: • A button has been pressed • A button has been released • The mouse has been moved The parser provides the information in a structure and creates the structure variable **$mouse_event** with the following elements: **Structure elements:** • **type** Coding of the activity – 2 - A button has been pressed – 3 - A button has been released – 5 - The mouse has been moved • **x** X-position of the cursor in pixels; relative to the current screen resolution • **y** Y-position of the cursor in pixels; relative to the current screen resolution • **id** Identifier – -1, if the position cannot be assigned to any control – != -1, if the mouse cursor is inside a control, the content of attribute **idemdata** is returned • **button** Contains the state of the buttons at the time of the event – 0 - No button – 1 - Left button – 2 - Right button – 4 - Center button The buttons can be associated with a bit-by-bit OR operation. **Example:** `<MOUSE_EVENT>` `  <print text="button %d type %d x %d y %d ">$mouse_event.button, $mouse_event.type, $mouse_event.x, $mouse_event.y</print>` `</MOUSE_EVENT>`

Tag identifier	Meaning
MESSAGE	**Dialog message** If the Send_message operation is executed in the script, then the parser processes the tag message. Values P1 and P2 are provided in the variables $message_par1 and $message_par2 (see the "SEND_MESSAGE" tag). **Syntax:** <MESSAGE> </MESSAGE> **Example:** <LET name="user_selection" /> <SOFTKEY POSITION="3"> <CAPTION>Set%nParameter</CAPTION> <SEND_MESSAGE>1, 10</SEND_MESSAGE> </SOFTKEY> … … <FORM> … … <MESSAGE> <SWITCH> <CONDITION>$message_par1</CONDITION> <CASE value="1"> <OP> user_selection = $message_par2 </OP> … … </CASE> </SWITCH> </MESSAGE> … … </FORM>

Tag identifier	Meaning
SEND_MESSAGE	The tag sends a message with two parameters to the active form, which is processed in the tag message (see also MESSAGE). **Syntax:** `<SEND_MESSAGE>p1, p2</SEND_MESSAGE>` **Example:** `<LET name="user_selection" />` `<SOFTKEY POSITION="3">` `<CAPTION>Set%nParameter</CAPTION>` `<SEND_MESSAGE>1, 10</SEND_MESSAGE>` `</SOFTKEY>` … … `<FORM>` … … `<MESSAGE>` `<SWITCH>` `<CONDITION>$message_par1</CONDITION>` `<CASE value="1">` `<OP> user_selection = $message_par2 </OP>` … `</CASE>` `</SWITCH>` `</MESSAGE>` … … `</FORM>`

Tag identifier	Meaning
RESIZE	**Dialog box message** The tag can be linked into the script for processing a RESIZE event. This event is created by a dynamic resolution switchover. 

Tag identifier	Meaning
RESIZE continued	**Example:** `<let name="screen_size" type="StructSize" />` `<form name="menu_userscale_form" autoscale_content="off">` ` <init>` ` <caption>Use auto scaling</caption>` ` <function name="hmi.get_hmi_resolution">screen_size</function>` ` <print text="screen size: w %d h: %d">screen_size.width, screen_size.height</print>` ` <data_access type="true" />` ` <control name="s_c" xpos="8" ypos="140" fieldtype="readonly" width="500" refvar="string_var" hotlink="true"/>` ` <if>` ` <condition>screen_size.width == 800</condition>` ` <then>` ` <control name="s_1" xpos="508" ypos="140" fieldtype="readonly" width="100" refvar="string_var1" hotlink="true"/>` ` </then>` ` </if>` ` <control name="s_w" xpos="200" ypos="350" fieldtype="readonly" refvar="screen_size.width" hotlink="true"/>` ` <control name="s_h" xpos="200" ypos="370" fieldtype="readonly" refvar="screen_size.height" hotlink="true"/>` ` </init>` ` <paint>` ` <text xpos ="68" ypos="310">Text</text>` ` <text xpos ="8" ypos="350">screen width</text>` ` <text xpos ="8" ypos="370">screen height</text>` ` </paint>` ` <resize>` ` <function name="hmi.get_hmi_resolution">screen_size</function>` ` <print text="resize_event screen size: w %d h: %d">screen_size.width, screen_size.height</print>` ` <switch>` ` <condition>screen_size.width</condition>` ` <case value="640">` ` <function name="control.delete">_T"s_1"</function>` ` </case>` ` <case value="800">` ` <control name="s_1" xpos="508" ypos="140" fieldtype="readonly" width="100" refvar="string_var1" hotlink="true"/>` ` </case>` ` </switch>` ` </resize>` `</form>`

Tag identifier	Meaning
CHANNEL_CHANGED	**Dialog message** The tag enables processing of the channel change event. It is executed when the user presses the channel advance button. **Syntax:** `<channel_changed>` `</channel_changed>` **Example:** When the channel is changed, the form is to close and open again so that the data of the selected channel can be displayed. `<channel_changed>` `  <open_form name = "form1" reopen="true"/>` `</channel_changed>` or a another menu is activated when a channel is changed. `<channel_changed>` `  <navigation>menu2</navigation>` `</channel_changed>`
LANGUAGE_CHANGED	**Dialog message** The tag allows processing of a language switchover request. It is executed when the user switches language while the screen form is open. **Syntax:** `<language_changed>` `</language_changed>` **Example:** When the language is switched over, the form is to close and open again in order to be able to supply the controls with the text elements of the activated language. `<language_changed>` `  <open_form name = "form1" reopen="true"/>` `</language_changed>`
FOCUS_IN	**Dialog box message** The tag is called if the system places the focus on a control. To identify the control, the system copies the name of the control into variable **$focus_name** and the value of the attribute **item_data** into variable **$focus_item_data**. This message can be used, e.g. to output images depending on the focus position. **Example:** `<focus_in>` `<PRINT text="focus on filed:%s, %d">$focus_name,` `$focus_item_data </PRINT>` `</focus_in>`

Tag identifier	Meaning
PAINT	**Dialog box message** The tag is executed when the dialog box is displayed. All the texts and images which are to be displayed in the dialog box should be specified here. Further, the tag is executed if the system identifies that parts of the dialog box are to be redisplayed. For example, this can be initiated by closing high-level windows.
TIMER	**Dialog box message** The tag is executed cyclically. Each form is assigned a timer that initiates that the timer - tag is executed approx. every 100 ms.
CAPTION	The tag contains the title of the dialog box. This tag should be used within the INIT tag. The title bar can be divided into multiple columns. To divide the title bar, the "caption" tag must be programmed with the "define_section" attribute before the text is output. The "define_section" attribute specifies the number of columns The "property" attribute is used to define the length, starting position, and text alignment for each column. The position and length specification is a percentage value referring to the width of the form. The value of the "value" attribute indicates the column number (beginning with zero) `<caption define_sections="3">` `<property value="0" length="20" position="2" alignment="left" />` `<property value="1" length="20" position="79" alignment="right" />` `</caption>` The text can then be assigned to the column by additionally specifying the index attribute with the column index. `<caption index="0">column1</caption>` `<caption index="1">column2</caption>` **Syntax:** `<CAPTION>Titel</CAPTION>` **Example:** `<CAPTION>my first dialogue</CAPTION>`
CLOSE	**Dialog box message** This tag is executed before the dialog box is closed.

Tag identifier	Meaning
CLOSE_FORM	The tag closes the active dialog. This instruction is only necessary if the dialog is opened by the MMC command and the user is offered a softkey function to close the dialog. Generally, dialogs are automatically managed and do not have to be explicitly closed. **Syntax:** `<CLOSE_FORM/>` **Example:** `<softkey_ok>` ` <caption>OK</caption>` ` <CLOSE_FORM />` ` <navigation>main_menu</navigation>` `</softkey_ok>`

Tag identifier	Meaning
CONTROL	The tag is used to generate control elements. **Syntax:** `<CONTROL name = "<control name>" xpos = "<X position>" ypos = "<Y position>" refvar = "<NC variable>" hotlink = "true" format = "<format>" />` **Note:** Variables used to set attribute values must be declared global variables. **Attributes:** • **name** Identifier of the field. The identifier simultaneously represents a local variable, and must not be used a multiple number of times in the form. • **xpos** X position of the top left corner • **ypos** Y position of the top left corner • **fieldtype** Field type If no type is specified, the field is set as an edit field. – **edit** Data can be changed – **readonly** Data cannot be changed **combobox** The field displays the corresponding identifiers instead of numerical values. If field type **combobox** is selected, the expressions to be displayed must also be assigned to the field. The <ITEM> tag should be used for this purpose. The combo box saves the index of the currently selected text in the variable belonging to the control (see the attribute **refvar**). After the control has been created, additional elements can be inserted using the functions **addItem** or **insertItem**. – **progressbar** A progress bar with a value range of 0 to 100 appears. The valley value and peak value properties can be used to adapt the value range to the data to be displayed.

Tag identifier	Meaning
CONTROL continued	• **fieldtype edit** and **readonly** For the field type **edit** and **readonly**, the **multiline** attribute permits text to be displayed in multiple lines. The line break occurs at a word boundary or with the line feed character <LF>. **Example:** `<control name="mulitline_edit" xpos="280" ypos="60" width="200" height="100" type="string">` `  <property multiline="true" />` `</control>` `<control name="c2" xpos="280" ypos="260" width="200" height="100" type="string" fieldtype="readonly">` `  <property multiline="true" />` `</control>`

Tag identifier	Meaning
CONTROL continued	• **fieldtype** – **graphicbox** The field type generates a 2D broken line graphic control. Using the tag <ITEM> a graphical element can be inserted into the control. Parameters **width** and **height** specify the width and height of the box. After the control has been created, additional elements can be inserted using the functions **addItem** or **insertItem**. Parameter **itemdata** is not evaluated for this control. If a reference variable is assigned to the control, its values will be recorded in a 100 ms time base. Up to 10000 values can be recorded. With the property **max**, values are recorded endlessly. The oldest value is then deleted when the maximum recording time is reached. The recording time must be specified in milliseconds. Although the values are stored at discrete times, the control displays them as connected lines. The attribute **channel** permits recording of up to three further reference variables. The index starts with one. Index zero is used to set the properties of the variables assigned in the control tag. **Example:** `<control name= "c_gbox1" xpos = "250" ypos="24" width="240" height="356" fieldtype="graphicbox" refvar="val" hotlink="true" COLOR_BK="#ffffff">` `  <property max="60000" /> <!-- steps of ms -->` `  <property min="0" />` `  <property channel="0" color="#00ffff" penwidth="3"/>` `  <property ORDINATE="Y sinus" />` `  <property ABSCISSA="time *100 ms" />` `  <property channel="1" refvar="val2" color="#000000" />` `</control>` `<request name="nck/Channel/Parameter/R[2]" />` `<control name= "c_gbox" xpos = "0" ypos="5"  width="260" height="200" fieldtype="graphicbox" refvar="nck/Channel/Parameter/R[1]" hotlink="true" COLOR_BK="#ffffff">` `  <property max="400" />` `  <!-- steps of  ms  -->` `  <property min="0" />` `  <property channel="0" color="#00ff00" penwidth="1"/>` `  <property ORDINATE="Power" />` `  <property ABSCISSA="Time *100 ms" />` `  <property channel="1" refvar="nck/Channel/Parameter/R[2]" color="#FF0000" penwidth="1"/>` `  <property FIX_TIME_AREA="on" />` `</control>`

Tag identifier	Meaning
CONTROL continued	**Example graphicbox:** `<CONTROL name= "graphic" xpos = "8" ypos="23" width="300" height="352" fieldtype="graphicbox" />` • Adding elements: Elements are added using the function **additem** or **loaditem**. The following 2d elements can be used: – Line - l(inc) – Circle sector - c(ircle) – Point - p(oint) **Structure of an element:** <Element type>; coordinates • Line: l; xs; ys; xe, ye I - line marking Xs - X start position Ys - Y start position Xe - X end position Ye - Y end position • Circle: C, xs, ys, xe, ye, cc_x, cc_y, r C - circular sector marking Xs - X start position Ys - Y start position Xe - X end position Ye - Y end position Cc_x - X-coordinate, circle center point CC_y - Y-coordinate circle center point • Radius: R • Point: P, x, y P - point marking X - X position Y - Y position • Deleting the graphic: The content is deleted using the function **empty**.

Tag identifier	Meaning
CONTROL continued	The following field types are described in section "Configuring your own buttons (Page 209)". • **fieldtype** – **pushbutton** The field type is used as a pushbutton or a switch. – **radiobutton** The field type allows you to select one of several options. – **checkbox** The field type allows you to select several options. – **groupbox** The field type visually encloses a group of controls with a frame and a title. – **switch** The field type is a graphical element that signals one of two states using an icon. – **scrollarea** The field type is used to display controls within a specified area.

Tag identifier	Meaning
CONTROL continued	• **fieldtype** – **listbox** The field type generates an empty list box control. Using the tag <ITEM> a list box element can be inserted in the list box. The ITEM attribute **value** allows this element to be assigned a unique value. This can serve to identify the element, for example. Parameters **width** and **height** specify the width and height of the list box. After the control has been created, additional list box elements can be inserted using the functions **addItem**, **insertItem**, or **loadItem**. After the control has been created, additional list box elements can be inserted using the function **addItem** or **insertItem**. Parameter **itemdata** is not evaluated for this control. – **itemlist** The field type generates a static control, which displays the corresponding identifier instead of numerical values. The <ITEM> tag can be used to assign an identifier to the field. • **item_data** A user-specific integer value can be assigned to the attribute. This value is given as part of the FOCUS_IN message for identifying the focus field. • **refvar** Identifier of the reference variable that can be linked to the field (optional). • **hotlink** = "TRUE" " If the value of the reference variable changes, then the field is automatically updated (optional). • **format** The attribute defines the display format of the specified variable. Formatting data, see print-Tag (optional). The **Format** attribute also allows an integer value to be displayed in other number systems or display as a Boolean value for the field types **edit** and **readonly**. Formatting using number of characters and alignment is not supported. The following types of representation can be used: • **%o** - octal • **%x** - hexadecimal • **%n** - binary • **%b** - bool
CONTROL continued	**Example for creating columns, attribute property:** `<control name="lb1" xpos = "10" ypos = "94" width="200" height="250" fieldtype="listbox" refvar="tmp" hotlink="true">` `  <property columns="4" />` `  <property column="0" type="width">190</property>` `  <property column="1" type="width">100</property>` `  <property column="2" type="width">190</property>` `  <property column="3" type="width">16</property>` `</control>`

Tag identifier	Meaning
CONTROL continued	**Example attribute format:** `<let name="val_test">255</let>` `<form name="main_form">` `<init>` `<caption>bool hex octal bin display</caption>` `<control name="test_oVal" xpos="250" ypos="60" refvar="val_test" hotlink = "true" />` `<control name="test_bool" xpos="250" ypos="90" refvar="val_test" hotlink = "true" format="%b"/>` `<control name="test_hex" xpos="250" ypos="120" refvar="val_test" hotlink = "true" format="%x"/>` `<control name="test" xpos="250" ypos="150" refvar="val_test" hotlink = "true" format="%o"/>` `<control name="test_b" xpos="200" ypos="180" width="300" refvar="val_test" hotlink = "true" format="%n"/>` `</init>` `<paint>` `<text xpos="16" ypos="60">integer value</text>` `<text xpos="16" ypos="90">boolean display</text>` `<text xpos="16" ypos="120">hexadecimal display</text>` `<text xpos="16" ypos="150">octal display</text>` `<text xpos="16" ypos="180">binary display</text>` `</paint>` `</form>`
CONTROL continued	**Attributes:** • **color_bk** The attribute sets the background color of the control. • **color_fg** The attribute sets the foreground color of the control. **More information** is provided in Chapter "Color coding (Page 72)" • **display_format** The attribute defines the processing format of the specified variable. This attribute must be used when accessing a PLC float variable, as the access is realized by reading a double word. The following data formats are permitted: – FLOAT – INT – DOUBLE – STRING Assigning expressions (e.g. text or graphic element to be displayed) to a list box, graphics box or combo box: **Syntax:** <ITEM>Expression</ITEM> <ITEM value ="<Value>">Expression</ITEM>

Tag identifier	Meaning
CONTROL continued	**Example:** `<CONTROL name = "button1" xpos = "10" ypos = "10" fieldtype = " combobox ">` `<ITEM>text1</ITEM>` `<ITEM>text2</ITEM>` `<ITEM>text3</ITEM>` `<ITEM>text4</ITEM>` `</CONTROL>` If any integer value is to be assigned to an expression, the attribute **value = "value"** should be added to the tag. Rather than consecutive numbers, the control variable now contains the item's assigned value. **Example:** `<CONTROL name = "button1" xpos = "10" ypos = "10" fieldtype = " combobox ">` `<ITEM value = "10">text1</ITEM>` `<ITEM value = "20">text2</ITEM>` `<ITEM value = "12">text3</ITEM>` `<ITEM value = "1">text4</ITEM>` </CONTROL> **Example of a progress bar:** `<CONTROL name = "progress1" xpos = "10" ypos = "10" width = "100" fieldtype = "progressbar" hotlink = "true" refvar = "nck/Channel/GeometricAxis/actProgPos[1]">` `<PROPERTY min = "0" />` `<PROPERTY max = "1000" />` `</CONTROL>` **Example, list box:** `<let name="item_string" type="string"></let>` `<let name="item_data" ></let>` `<CONTROL name="listbox1" xpos = "360" ypos="150" width="200" height="200" fieldtype="listbox" />` • Adding elements: Elements are added using the function **additem** or **loaditem**. • Deleting the content: The content is deleted using the function **empty**. `<op> item_string = _T"text1\\n" </op>` `<function name="control.additem">_T"listbox1", item_string, item_data </function>` `<op> item_string = _T"text2\\n" </op>` `<function name="control.additem">_T"listbox1", item_string, item_data </function>`
CONTROL continued	**Example itemlist:** `<CONTROL name = "itemlist1" xpos = "10" ypos = "10" fieldtype = " itemlist">` `<ITEM value = "10">text1</ITEM>` `<ITEM value = "20">text2</ITEM>` `<ITEM value = "12">text3</ITEM>` `<ITEM value = "1">text4</ITEM>` `</CONTROL>`

Tag identifier	Meaning
CONTROL continued	The image box control manages a picture in bitmap or GIF format. If the picture is larger than the displayable area, the control will show a scroll bar. To control the visible area, the system provides the CONTROL.IMAGEBOXSET function. **More information** is provided in Chapter "Predefined functions (Page 153)" **Syntax:** `<function name="control.imageboxget"> ... </function>`
CONTROL continued	**Attributes:** • **disable** The attribute locks/permits the input in an edit control. • **tooltip** An information text is displayed if the cursor is placed on the control. • **factor** Conversion factor • **font** Definition of a font • **fontpixelsize** Character height - the value is to be specified in number of pixels and refers to a resolution of 640x480 pixels. The displayed character height is determined from the ratio of the actual resolution and the reference resolution. **Example:** `<control name="c2" xpos="180" ypos="120" fontname="arial" fontpixelsize="26" height="100" type="string" />`

Tag identifier	Meaning
CONTROL continued	**Attribute:** • **fontname** The attribute is used to specify the font to be used. The installed fonts can be determined using the HMI.GET_FONT_FAMILIES function. **Value:** Font name The following Siemens fonts are installed in the system: • Siemens AD CH Simplified • Siemens AD CH Traditional • Siemens AD JP • Siemens AD KS • Siemens AD Mono • Siemens AD Mono CH Simplified • Siemens AD Mono CH Traditional • Siemens AD Mono JP • Siemens AD Mono KS • Siemens AD Mono TH • Siemens AD Mono VN • Siemens AD Sans • Siemens AD TH • Siemens AD VN • Siemens Logo • Siemens Sans Cond Global • Siemens Sans Cond Mono • Siemens Sans Global MS Windows-based systems may contain additional fonts. **Example:** `<control name="c2" xpos="180" ypos="120" fontname="arial" fontpixelsize="26" height="100" type="string" />`

Tag identifier	Meaning
CONTROL continued	**Changing the control after creation** A control tag changes the properties of an existing control after it has been created. The tag must be specified with the name of the control to be changed and the new properties. It can be executed only within a form tag. The following properties can be changed, for example: • name • xpos • ypos • width • height • color_bk • color_fg • access level • fieldtype • itemdata • min • max • default • disable • tooltip • font • factor The reference variable cannot be modified. If a property is to be changed by triggering by a softkey event, the send message tag must transfer this request into the form context. The message tag is used to acquire the message.

Tag identifier	Meaning
CONTROL continued	**Control change in an operating instruction** Another possibility of changing properties of a control during runtime is to make the change in an operating instruction. Therefore the name of the control and the property to which a new value is to be assigned must be defined. The property is separated from the control name by a point. **Syntax:** <Name>.<Property> **Example:** `...` `...` `<let name="value" />` `<let name="w" />` `<let name="h" />` `...` `...` `<control name="c_move" xpos="$xpos" ypos="124" />` `...` `...` `<op>` `c_move.xpos = 300;` `value = c_move.xpos;` `h = c_move.height;` `w = c_move.width;` `</op>`

Tag identifier	Meaning
HELP_CONTEXT	This tag defines the help topic to be called. It should be programmed in the INIT block. **808/802D sl systems** The name specified in the attribute is supplemented by the prefix XmlUserDlg_ and is transferred to the help system. The associated structure of the help file should be taken from the topic - generating an online help. **Sequence when activating the help system:** 1. Press the "Info" key. 2. The dialog supplies the expression "my_dlg_help". 3. Parser converts the expression into "XmlUserDlg_my_dlg_help" . 4. Activating the help system. 5. Submitting the search term "XmlUserDlg_my_dlg_help". **840D sl/828D systems** **More information** regarding the layout and generation of a help book: SINUMERIK Operate Commissioning Manual **Attributes:** • **name** The name specified in the attribute is passed on to the help system. The file name used in the help book in tag **entry** must be specified. • **anchor** With the attribute you can enter a keyword. **Syntax:** `<HELP_CONTEXT name="<file name>" anchor="key word"/>` **Example:** `<form name = "form1">` ` <init>` `  <help_context name="chapter_1.html" anchor="Keyword_1" />` `...` `...` `  <control name = "cpwd" xpos = "322" ypos = "64" >` `   <property password="true" />` `   <property help_context="chapter_1.html" anchor="Keyword_2" />` `  </control>` `</form>`

Tag identifier	Meaning
DATA_ACCESS	The tag controls the behavior of the dialog forms when user inputs are being saved. The behavior should be defined within the INIT tag. If the tag is not used, inputs are buffered in each case. Exception: Controls for which the **hotlink** attribute is set to **true** are always written to and read directly. **Attribute:** • **type** = "TRUE" – the input values are not buffered. The dialog form copies the input values to the reference variables directly. • **type =** "FALSE" – the values are only copied to the reference variable with the **UPDATE_CONTROLS** type = "FALSE" tag **Example:** `<DATA_ACCESS type = "true" />`

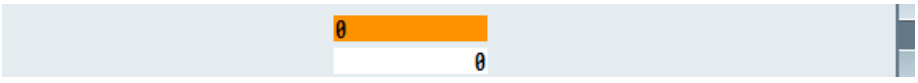
Tag identifier	Meaning
DEBUG	The tag is used to store trace outputs, which can be programmed in the script. The attribute **text** is used to write the text to be stored. This attribute and its values correspond to the **text** attribute and the values of the print instruction. By default, a debug tag is not executed as it slows down the system. To activate the debug outputs, program the attribute **debug_msg**, with value **on** in the **DialogGui** or **AGM** tag. The debug outputs are stored together with the parser error messages in the following file: card/oem/sinumerik/hmi/dvm/log/dvm.log **Syntax:** `<debug text="<Text see print instruction>">Variables see print instruction</debug>` **Example Easy XML:** `<DialogGui debug_msg="on">` `...` `...` `<control name="test_bool" xpos="250" ypos="90" refvar="val_test" hotlink = "true" format="%b"/>` `<debug text="main_form control test_bool created value:%d">test_bool</debug>` Entry in the file dvm.log: `date: 10/22/ 2018 time: 09:38:44:351` `Debug xmldial.xml 17: main_form control test_bool created value:255` **Example Easy Extend:** `<AGM debug_msg="on">` `...` `...` `<control name="test_bool" xpos="250" ypos="90" refvar="val_test" hotlink = "true" format="%b"/>` `<debug text="main_form control test_bool created value:%d">test_bool</debug>` Entry in the file dvm.log: `date: 10/22/ 2018 time: 09:38:44:351` `Debug xmldial.xml 17: main_form control test_bool created value:255`

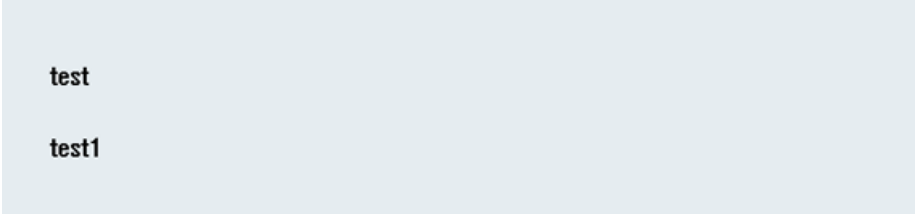
Tag identifier	Meaning
EDIT_CHANGED	**Dialog box message** This tag is called if the contents of an edit control have changed. To identify the control, the system copies the name of the control into variable **$focus_name** and the value of the attribute **item_data** into variable **$focus_item_data**. **Example:** `<EDIT_CHANGED>` `<print text="index changed filed:%s, %d"> $focus_name, $focus_item_data </print>` … … `</EDIT_CHANGED>`
GESTURE_EVENT	The tag is used to perform finger gestures for multi-touch operation. The tag is described in section "Multitouch operation (Page 201)".
INDEX_CHANGED	**Dialog box message** The tag is called, if the operator changes the selection of a combo box. To identify the control, the system copies the name of the control into the variable **$focus_name** and the value of the attribute **item_data** into the variable **$focus_item_data**. **Note:** A reference variable assigned to the control, has not been aligned to the control variable at this point in time and contains the index of the previous selection of the combo box. **Example:** `<INDEX_CHANGED>` `<print text="index changed filed:%s, %d"> $focus_name, $focus_item_data </print>` … … `</INDEX_CHANGED>`
MENU	The tag defines a menu containing the softkey description and the dialog to be opened. **Attribute:** • **name** Menu name **Syntax:** `<MENU name = "<menu name>">` … `<open_form …>` … `<SOFTKEY …>` `</SOFTKEY>` `</MENU>`

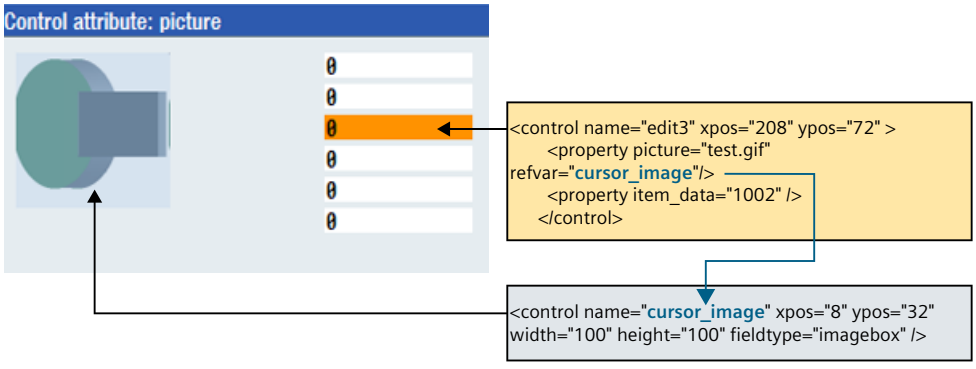
Tag identifier	Meaning
NAVIGATION	This tag defines the menu to be called. It can be used within a softkey block, a menu block, and in a form. If a variable name is assigned to the tag as its value, the parser will activate the menu stored in the variable. In a menu block, the navigation is at the position in the instruction. Subsequent instructions are no longer executed. **Note:** If the navigation target is to be determined by the content of a variable, this variable must be declared a global variable. **Syntax:** `<NAVIGATION>menu name</NAVIGATION>` **Example:** `<menu name = "main">` `  <softkey POSITION="1">` `  <caption>sec. form</caption>` `  <navigation>sec_menu</navigation>` `  </softkey>` `</menu>` `<menu name = "sec_menu">` `  <open_form name = "sec_form" />` `  <softkey_back>` `  <navigation>main</navigation>` `  </softkey_back>` `</menu>`

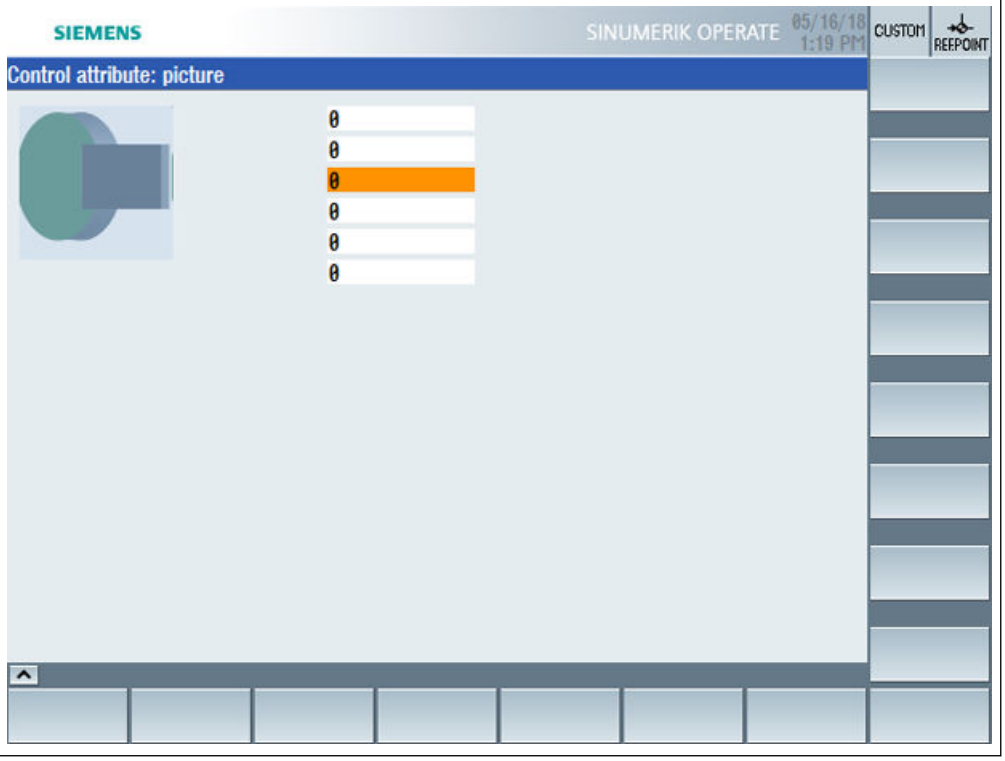
Tag identifier	Meaning
OPEN_FORM	The tag opens the dialog form given under the name. If the specified screen form is already active, this tag is not executed. You can force the screen form to open again by entering the attribute **reopen**. This may be necessary if system states change that require rearrangement of screen form contents. **Attribute:** • **name** Name of the dialog form • **reopen** If the value of the attribute is set to **true**, when the tag **open_form** is called again, a check is made to see whether the screen form is the active form. If this is the case, the parser closes the active form and opens it again. **Syntax:** `<OPEN_FORM name = "<form name>" />` **Example:** `<menu name = "main">` `  <open_form name = "main_form" />` `  <softkey POSITION="1">` `  <caption>main form</caption>` `  <navigation>main</navigation>` `  </softkey>` `</menu>` `<form name="main_form">` `  <init>` `  </init>` `  <paint>` `  </paint>` `</form>`

Tag identifier	Meaning
PROPERTY	This tag can be used to define additional properties for an operator control. The tag is embedded in the control tag. **Attributes:** • **max** = "<maximum value>" • **min** = "<minimum value>" • **default** = "<default>" • **factor** = "conversion factor" • **color_bk** = "<background color coding>" • **color_fg** = "<font color coding>" • **password** = "<true>" - entered character is displayed with "*" • **multiline** = "<true>" - permits multi-line inputs in an edit control • **disable** = "<true/false>" - blocks/permits the input in an edit control • **transparent** = "transparent color of a bitmap" **More information** is provided in Chapter "Color coding (Page 72)" • **tooltip** = Information text is displayed if the cursor is set to the control. The syntax is: <property tooltip="note text" /> • **abscissa** = "name of the first coordinate axis" (only permissible for a graphic box) • **ordinate** = "name of the second coordinate axis" (only permissible for a graphic box) • **fix_time_area** = "on" - The attributes **min** and **max** define the time period in which the signals are to be displayed. The displayed signals are shifted from the right to the left. **Example:** `<CONTROL name = "progress1" xpos = "10" ypos = "10" width = "100" fieldtype = "progressbar" hotlink = "true" refvar = "nck/Channel/GeometricAxis/actProgPos[1]">` `<PROPERTY min = "0" />` `<PROPERTY max = "1000" />` `</CONTROL>` `<CONTROL name = "edit1" xpos = "10" ypos = "10">` `<PROPERTY min = "20" />` `<PROPERTY max = "40" />` `<PROPERTY default = "25" />` `</CONTROL>`

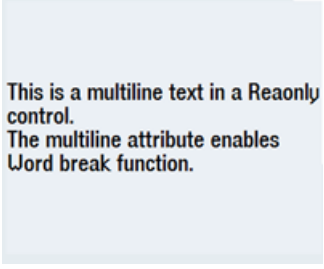
Tag identifier	Meaning
PROPERTY continued	**Attribute:** **alignment** - the property allows left or right-justified text alignment. By default, a text is left-justified. **Values:** • **left** • **right** **Syntax:** `alignment="<right/left>"` **Example:** `<control name="edit2" xpos="208" ypos="52" >` `<property alignment="right"/>` `</control>` 

Tag identifier	Meaning
PROPERTY continued	**Attribute:** **parent** - the property determines the membership of the control. By default, controls are assigned to the form. If you want to assign controls to a scroll view, you must specify it as the parent window. **Value:** Name of the parent window **Example:** `...` `<control name="area" xpos="8" ypos="72" width="554" height="320" fieldtype="scrollarea" >` `<control name="test" xpos="20" ypos="40" width="20" fieldtype="readonly" />` `</control >` `<control name="test1" xpos="20" ypos="80" width="20" fieldtype="readonly" parent="area"/>`  `<init>` `<control name="area" xpos="8" ypos="72" width="554" height="120" fieldtype="scrollarea">` `<control name="test" xpos="20" ypos="40" width="80" fieldtype="readonly" type="string"/>` `</control >` `<control name="test1" xpos="20" ypos="80" width="80" fieldtype="readonly" type="string" parent="area"/>` `<op>test = _T"test"</op>` `<op>test1 = _T"test1"</op>` `<update_controls type="true" />` `...`

Tag identifier	Meaning
PROPERTY continued	**Attribute:** **picture** - the attribute specifies a graphic to be displayed The specified graphic is displayed or the name of the graphic is stored in a variable if the input focus is set on this field. This attribute is to be used in conjunction with the **refvar** attribute to be able to define the data sink. To display a graphic or an animation, the name of an **imagebox** type control must be specified. This control assumes management of the graphic. If the name of a local variable is specified, this variable must be of a **String** data type. The graphic continues to be displayed until a new name is assigned to the reference variables. **Syntax:** `<property picture="<Name der Grafik>" refvar="<Datensenke>"/>`
PROPERTY continued	**Example of field linking:** 

Tag identifier	Meaning
PROPERTY continued	**Example:** `<let name="cursor_icon" type="string" />` ` <form name="main_form1">` `  <init>` `  <caption>Control attribute: picture</caption>` `  <control name="cursor_image" xpos="8" ypos="32" width="100" height="100" fieldtype="imagebox" />` `  <control name="edit1" xpos="208" ypos="32" >` `  <property picture="red_led_off.bmp" refvar="cursor_image"/>` `  </control>` `  <control name="edit2" xpos="208" ypos="52" >` `  <property picture="red_led_on.bmp" refvar="cursor_image"/>` `  </control>` `  <control name="edit3" xpos="208" ypos="72" >` `  <property picture="test.gif" refvar="cursor_image"/>` `  </control>` `  <control name="edit4" xpos="208" ypos="92" >` `  <property picture="red_led_off.bmp" refvar="cursor_image"/>` `  </control>` `  <control name="edit5" xpos="208" ypos="112" >` `  <property picture="red_led_off.bmp" refvar="cursor_icon"/>` `  </control>` `  <control name="edit6" xpos="208" ypos="132" >` `  <property picture="red_led_on.bmp" refvar="cursor_icon"/>` `  </control>` `  </init>` `  <timer><print text="%s">cursor_icon</print></timer>` ` </form>` 

Tag identifier	Meaning
PROPERTY continued	**Attribute:** **cursortext** - this attribute defines the cursor text to be displayed The text is issued in the title bar if the input focus is set on this field. Two further attributes can be specified in addition to this attribute which define the alignment of the text and the cursor text area. The text is left-justified as standard. The cursor text area takes up 50% of the title bar width by default. `alignment="<right/left>"` - text alignment right-justified/left-justified `length="<Width>"` - percentage of the title bar which should be required for the cursor text **Syntax:** `<property cursortext="<text>" />` or `<property cursortext="<text>" alignment="<right/left>" />` or `<property cursortext="text2" alignment="r="<text>" alignment="<right/left>" length="<Ratio>"/>` **Example:** `<form name="main_form2">` ` <init>` `  <caption>Control attribute: cursortext</caption>` `  <control name="edit1" xpos="208" ypos="32" >` `  <property cursortext="cursor text field edit1" />` `  </control>` `  <control name="edit2" xpos="208" ypos="52" >` `  <property cursortext="cursor text field edit2" alignment="right"/>` `  </control>` `  <control name="edit3" xpos="208" ypos="72" >` `  <property cursortext="cursor text field edit3" alignment="right" length="40"/>` `  </control>` `  <control name="edit4" xpos="208" ypos="92" >` `  <property cursortext="cursor text field edit4" />` `  </control>` ` </init>` `</form>` 

Tag identifier	Meaning
PROPERTY continued	**Attribute:** **multiline** - this allows the multi-line output of a text in an edit or read-only control **Syntax:** `<property multiline="true" />` **Example:** `...` `...` `<op>` `textlist[2] = _T"This is a multiline text in a Reaonly control.\` `\nThe multiline attribute enables Word break function.";` `</op>` `<control name="lstring" xpos="8" ypos="120" width="200" height="160"` `fieldtype="readonly" refvar="textlist[2]" >` `<property multiline="true" />` `</control>` 
PROPERTY continued	**Attributes:** • **help_context** The attribute makes it possible to assign a help topic to a control. The file name used in tag **entry** in the help book must be specified. • **anchor** With the attribute you can enter a keyword. This attribute is only valid in conjunction with the attribute **help_context**. **Syntax:** `<property help_context="<file name>" anchor="<key word>"/>` **Example:** `<control name = "cpwd" xpos = "322" ypos = "64" >` `<property password="true" />` `<property help_context="chapter_1.html" anchor="Keyword_2" />` `</control>`

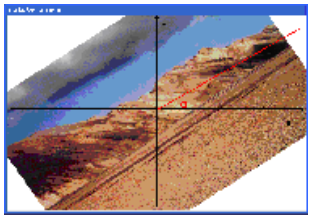
Tag identifier	Meaning
SOFTKEY	The tag defines the properties and responses of a softkey. **Attributes:** • **position** Number of the softkey. 1-8 horizontal softkeys, 9-16 vertical softkeys The following attributes become effective from: • **type** Defines the property of the softkey. user_controled - the script defines how the softkey is displayed toggle_softkey - the softkey is displayed alternating between pressed and not pressed • **refvar** Should only be used in conjunction with toggle_softkey . Reference variable, into which the actual softkey property is copied. A variable, type "String" should be specified, which includes the properties pressed, not pressed or locked (see tag state). • **picture** Using the attribute, a bitmap can be output left justified on the softkey. The complete path name should be specified. The number of text characters that can be displayed is reduced to the width of the bitmap.

Tag identifier	Meaning
SOFTKEY continued	The following additional actions can be defined within the softkey block: • **picturealignment** The image orientation is specified by this attribute. The image is aligned with the left side of the softkey by default. The following values can be specified for alignment: – **top** Top edge – **bottom** Bottom edge – **left** Left edge – **right** Right edge – **center** Centered • **caption** Softkey text • **state** Should only be used in conjunction with user_controlled . The tag assigns the required softkey display to the system. **Syntax:** <state type="<state>" /> The following strings can be specified: – **notpressed** The softkey is displayed as being not pressed. – **pressed** The softkey is displayed as being pressed. – **disabled** The softkey is locked and is displayed in gray. • **navigation** • **update_controls** • **function**

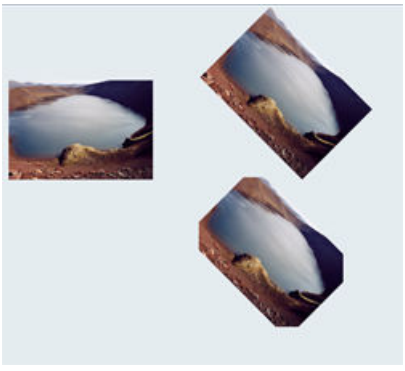
Tag identifier	Meaning
SOFTKEY continued	**Syntax:** Standard softkey: <state type="<softkey state>" /> <softkey position = "<1>"> … … </softkey> or Script-controlled softkey: <softkey position = "<1>" type="<user_defined>" > <state type="<softkey state>" /> … … </softkey> or Toggle softkey: <softkey position = "<1>" type="<toggle_softkey>" refvar="<variable name>" > … … </softkey> **Example:** <let name="define_sk_type" type="string">PRESSED</let> <let name="sk_type">1</let> <softkey POSITION="1" type="user_controled" > <caption>Toggle%nSK</caption> <if> <condition>sk_type == 0 </condition> <then> <op> sk_type = 1 </op> <op> define_sk_type = _T"PRESSED" </op> </then> <else> <op> define_sk_type = _T"NOTPRESSED" </op> <op> sk_type = 0 </op> </else> </if> <state type="$$$define_sk_type" /> … … </softkey>

Tag identifier	Meaning
SOFTKEY continued	**Example:** or <let name="curr_softkey_state" type="string">PRESSED</let> </softkey> <softkey POSITION="3" type="toggle_softkey" refvar="curr_softkey_state"> <caption>Toggle%nSK</caption> ... </softkey>  
SOFTKEY_OK	The tag defines the response of the softkey "OK".  The following additional actions can be defined within the softkey block: • **navigation** • **update_controls** • **function** **Syntax:** <SOFTKEY_OK> ... ... </SOFTKEY_OK>
SOFTKEY_CANCEL	The tag defines the response of the softkey "Cancel".  The following additional actions can be defined within the softkey block: • **navigation** • **update_controls** • **function** **Syntax:** <SOFTKEY_CANCEL> ... ... </SOFTKEY_CANCEL>

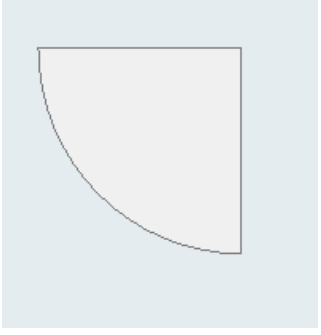
Tag identifier	Meaning
SOFTKEY_BACK	The tag defines the response of the softkey "Back".  The following additional actions can be defined within the softkey block: • **navigation** • **update_controls** • **function** **Syntax:** `<SOFTKEY_BACK>` ... ... `</SOFTKEY_BACK>`
SOFTKEY_ACCEPT	The tag defines the response of the softkey "Accept".  The following additional actions can be defined within the softkey block: • **navigation** • **update_controls** • **function** **Syntax:** `<SOFTKEY_ACCEPT>` ... ... `</SOFTKEY_ACCEPT>`
TEXT	The tag is used to display a text in the specified position. If an alarm number is used, the dialog box displays the text which is saved for the number. **Syntax:** `<TEXT xpos = "<X position>" ypos = "<Y position>"> Text </TEXT>` **Attributes:** • **xpos** X position of the top left corner • **ypos** Y position of the top left corner • **color** Text color **More information** is provided in Chapter "Color coding (Page 72)" **Value:** Text to be displayed

Tag identifier	Meaning
IMG	The tag is used to display an image in the specified position. The BMP and PNG image formats are supported. **Syntax:** `<IMG xpos = "<X position>" ypos = "<Y position>" name = "<name>" />` `<IMG … xrot="angle X axis" yrot=" angle Y axis " zrot=" angle Z axis " />` **Attributes:** • **xpos** X position of the top left corner • **ypos** Y position of the top left corner • **name** Complete path name. The path information is made in lowercase letters. • **transparent** Transparent color of the bitmap **More information** is provided in Chapter "Color coding (Page 72)"
IMG continued	**Example**: The image is rotated through 34 degrees around the Z axis: `<img xpos = "5" ypos = "23" name = "f:/appl/pic3.bmp" height="355" width="550" zrot="34" color_bk="#ffffff" >` `</img>` **Note:** The drive designations vary depending on the system.  **Optional:** If the image display is to differ from the original size, the dimensions can be defined using the attributes **width** and **height**. • **width** Width in pixels • **height** Height in pixels **Examples:** `<IMG xpos = "20" ypos = "40" name = "f:/appl/test.bmp" />` `<IMG xpos = "5" ypos = "23" name = "f:/appl/test.bmp" height = "355" width = "550"/>`

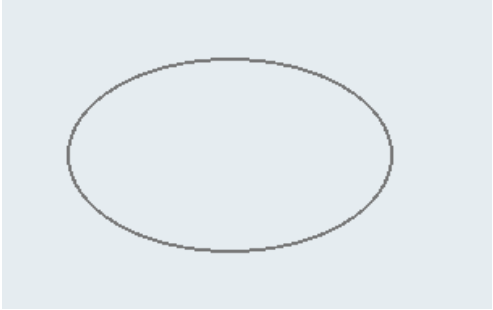
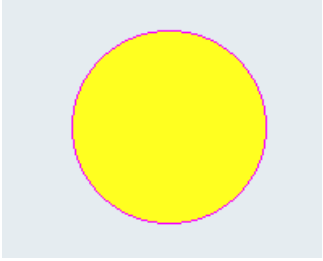
Tag identifier	Meaning
IMG continued	**Attribute:** **AspectRatioMode** This attribute allows the control of the aspect ratio for an image wherever non-proportional zooming is performed. **Values:** • **Ignore** The aspect ratio of the bitmap is adapted to the prescribed height and width. • **Keep** (default) The aspect ratio is retained and ensures that the image is scaled to a rectangle which assumes the greatest expansion within the prescribed height and width. • **KeepByExpanding** The aspect ratio is kept and ensures that the image is scaled to a rectangle which assumes the greatest expansion outside of the prescribed height and width. **Example:** `<img xpos = "8" ypos = "30" name = "test.bmp" height="155" width="150" AspectRatioMode="KeepbyExpanding">` `  <property transparent="#00ff00" />` `</img>` `<img xpos = "8" ypos = "200" name = "test.bmp" height="155" width="150" AspectRatioMode="Ignore">` `  <property transparent="#00ff00" />` `</img>` `<img xpos = "200" ypos = "200" name = "test.bmp" height="155" width="150" AspectRatioMode="Keep">` `  <property transparent="#00ff00" />` `</img>`  • Top - **KeepbyExpanding** property set • Bottom left - **Ignore** property set • Bottom right - **Keep** property set

Tag identifier	Meaning
IMG continued	**Attributes:** **ExpandingForRotation** The aspect ration is retained. The image is scaled to a rectangle which corresponds with the bounding box of the rotated and scaled image. **Example:** `<img xpos = "8" ypos = "100" name = "pic1.bmp" height="155" width="150" >` `  <property transparent="#ffffff" />` `</img>` `<op>` `  zrot = 45.0;` `</op>` `<img xpos = "205" ypos = "23" name = "pic1.bmp" height="155" width="150" zrot="$zrot" AspectRatioMode="ExpandingForRotation">` `  <property transparent="#ffffff" />` `</img>` `<img xpos = "205" ypos = "200" name = "pic1.bmp" height="155" width="150" zrot="$zrot" >` `  <property transparent="#ffffff" />` `</img>`  • Left - Image is scaled to a size of 150x155 pixels • Top right - Image is scaled to a size of 150x155 pixels and is rotated around 45 degrees with the **ExpandingForRotation** property • Bottom right - Image is scaled to a size of 150x155 pixels and is rotated around 45 degrees

Tag identifier	Meaning
ARC	The tag is used to draw a circle sector in a form. **Attributes:** Position within the form in pixels: • **xpos1**- starting point of the circle segment X-coordinate • **ypos1**- starting point of the circle segment Y-coordinate • **xpos2** - end point of the circle segment X-coordinate • **ypos2** - end point of the circle segment Y-coordinate Position within the form in pixels: • **ccx** - center of the circle segment X-coordinate • **ccy** - center of the circle segment Y-coordinate • **radius** - circle radius, value in pixels **Note:** **Center point or radius** If both parameters are specified, the circle center is used.
ARC continued	• **penwidth** The attribute specifies the line width of the line in pixels. • **color** Line color • **color_bk** Fill color of the circle segment **More information** is provided in Chapter "Color coding (Page 72)" • **style** The attribute defines the line type: – **"SolidLine"** - solid line (default) – **"DashLine"** - dashed line – **"DotLine"** - dotted line – **"DashDotLine"** - dot-dash line – **"DashDotDotLine"** - dash-dot-dot line

Tag identifier	Meaning
ARC continued	• **type** – **cw** - clockwise – **ccw** - counterclockwise • **arrow** Arrows are displayed at the segment ends: – **"P1"** - arrow at the starting point of the line – **"P2"** - arrow at the starting point of the line – **"P1P2"** - arrows at the starting point and end point of the line – **"P1_FILLED"** - filled arrow at the starting point of the line – **"P2_FILLED"** - filled arrow at the starting point of the line – **"P1P2_FILLED"** - filled arrows at the starting point and end point of the line • **arrow_size** If the Arrow attribute is used, the length of the arrow can be specified in pixels.
ARC continued	**Examples:**  `<arc xpos1="20" ypos1="200" xpos2="180" ypos2="200" ccx="100" ccy="200" type="ccw" PENWIDTH="1" color="#777777" arrow="p1p2_filled"/>` or `<arc xpos1="20" ypos1="200" xpos2="180" ypos2="200" radius="80" type="ccw" PENWIDTH="1" color="#777777" arrow="p1p2_filled"/>`  `<arc xpos1="20" ypos1="200" xpos2="100" ypos2="280" radius="80" type="ccw" PENWIDTH="1" color="#777777" color_bk="#f0f0f0"/>`

Tag identifier	Meaning
BOX	The tag draws a rectangle at the specified position, colored as indicated. **Syntax:** `<BOX xpos = "<X position>" ypos = "<Y position>" width = "<X extension>" height = "<Y extension>" color = "<Color code>" />` **Attributes:** • **xpos** X position of the top left corner • **ypos** Y position of the top left corner • **width** Extension in X direction (in pixels) • **height** Extension in Y direction (in pixels) • **color** Color coding **More information** is provided in Chapter "Color coding (Page 72)"
ELLIPSE	The tag is used to draw an ellipse in a form. **Attributes:** Position within the form in pixels: • **xpos** - starting point of the X-coordinate line • **ypos** - starting point of the Y-coordinate line • **width** - width of the surrounding rectangle (bounding box) • **height** - height of the surrounding rectangle (bounding box) • **penwidth** The attribute specifies the line width of the line in pixels. • **color** Line color • **color_bk** Fill color of the ellipse **More information** is provided in Chapter "Color coding (Page 72)" • **style** The attribute defines the line type: – **"SolidLine"** - solid line (default) – **"DashLine"** - dashed line – **"DotLine"** - dotted line – **"DashDotLine"** - dot-dash line – **"DashDotDotLine"** - dash-dot-dot line

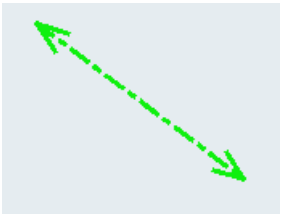
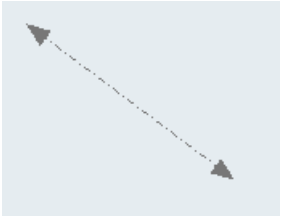
Tag identifier	Meaning
ELLIPSE continued	**Examples:**  `<ellipse xpos="302" ypos="160" width="100" height="60" PENWIDTH="2" color="#777777" />`  `<ellipse xpos="402" ypos="360" width="60" height="60" PENWIDTH="1" color="#f800ff"  color_bk="#ffff20"/>`

Tag identifier	Meaning
FUNCTION	**Function call** The tag executes the function body, which is specified under the attribute "**name**". **Attributes:** • **name** = "name of the function body" • **return** = "variable name for saving the result of the function" **Values:** List of variables to be transferred to the function body. The variables must be separated by a comma. A maximum of 10 parameters can be transferred. It is also possible to specify constants or text expressions as call parameters. The identifier _T should be placed at the start as a means of identifying a text expression. **Syntax:** `<FUNCTION name = "<function name>" />` Calling function expects a return value `<FUNCTION name = "<function name>" return = "<Variablenname>" />` Parameter transfer `<FUNCTION name = "<function name>"> var1, var2, var3 </FUNCTION>` `<FUNCTION name = "<function name>"> _T"Text", 1.0, 1 </FUNCTION>` **Examples:** See "FUNCTION_BODY"

Tag identifier	Meaning
FUNCTION_BODY	**Function body** The tag contains the function body of a subfunction. The function body needs to be programmed within the DialogGui tag. **Attributes:** • **name** = "name of the function body" • **parameter** = "parameter list" (optional) The attribute lists the transfer parameters that are required. The parameters must be separated by a comma. When the function body is called, the values of the parameters specified in the function call are copied to the transfer parameters listed. • **return** = "true" (optional) If the attribute is set to true then the local variable $return is created. The function's return value which is forwarded to the calling function on quitting the function should be copied to this variable. **Syntax:** **Function body without parameter** `<FUNCTION_BODY name = "<function name>">` … … … `</ FUNCTION_BODY>` **Function body with parameter** `<FUNCTION_BODY name = "<function_name>" parameter = "<p1, p2, p3>">` … `<LET name = "tmp></LET>` `<OP> tmp = p1 </OP>` … `</FUNCTION_BODY>` **Function body with return value** `<FUNCTION_BODY name = "<function_name>" parameter = "<p1, p2, p3>" return = "true">` … `<LET name = "tmp></LET>` `<OP> tmp = p1 </OP>` … `<OP> $return = tmp </OP>` `</FUNCTION_BODY>`

Tag identifier	Meaning
FUNCTION_BODY continued	**Example:** `<function_body name = "test" parameter = "c1,c2,c3" return = "true">` `<LET name = "tmp">0</LET>` `<OP> tmp = c1+c2+c3 </OP>` `<OP> $return = tmp </OP>` `</function_body>` … … … `<LET name = "my_var"> 4 </LET>` `<function name = "test" return = " my_var "> 2, 3,4</function>` `<print text = "result = %d"> my_var </print>` … …
LINE	The tag is used to draw a line in a form. **Attributes:** Position within the form in pixels: • **xpos1 -** starting point of the X-coordinate line • **ypos1** - starting point of the Y-coordinate line • **xpos2** - end point of the X-coordinate line • **ypos2** - end point of the Y-coordinate line • **penwidth** The attribute specifies the line width of the line in pixels. • **color** Line color **More information** is provided in Chapter "Color coding (Page 72)" • **style** The attribute defines the line type: – **"SolidLine"** - solid line (default) – **"DashLine"** - dashed line – **"DotLine"** - dotted line – **"DashDotLine"** - dot-dash line – **"DashDotDotLine"** - dash-dot-dot line

Tag identifier	Meaning
LINE continued	• **arrow** Arrows are displayed at the line ends: – **"P1"** - arrow at the starting point of the line – **"P2"** - arrow at the starting point of the line – **"P1P2"** - arrows at the starting point and end point of the line – **"P1_FILLED"** - filled arrow at the starting point of the line – **"P2_FILLED"** - filled arrow at the starting point of the line – **"P1P2_FILLED"** - filled arrows at the starting point and end point of the line • **arrow_size** If the Arrow attribute is used, the length of the arrow can be specified in pixels. **Syntax:** `<line xpos1="<X-coordinate start point>" ypos1="<Y-coordinate start point>" xpos2="<X-coordinate end point>" ypos2="<Y-coordinate start point>"` `PENWIDTH="<line width>" color="<line color>" style="<line style>" arrow="<arrow type>" arrow_size="<arrow size>"/>`

Tag identifier	Meaning
LINE continued	**Examples:** `<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="3" color="#0ff00f" style="DashDotLine" arrow="p1p2" arrow_size="12"/>`  `<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="1" color="#777777" style="DashDotLine" arrow="p1p2_filled" arrow_size="9"/>`  `<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="2" color="#777777" arrow="p2_filled" arrow_size="9"/>` 

Tag identifier	Meaning
REQUEST	The tag is used to add a variable to the cyclic reading service (Hotlink). As a consequence, the access time to variables, which are not linked to the control, is reduced. If a function is to be called automatically when a value changes, then the name of the function should be specified as an additional attribute. This tag is only processed within the INIT operation. **Attributes:** • **name** Address identifier • **function** Function name **Syntax:** `<REQUEST name = "<NC-Variable>" />` or `<REQUEST name = "<NC-Variable>" function="<function name>"/>` **Example:** `<request name ="plc/mb10" />` or `<function_body name="my_function" >` `<print text="value changed" />` `</function_body>` … … … `<request name ="plc/mb10" function="my_function"/>`

Tag identifier	Meaning
RECALL	This tag can be used in a menu if a navigation is to take place via Recall button. If the tag is programmed, the Recall icon is shown, and the parser may process the Recall button. **Syntax:** `<recall>` ... ... `</recall>`
UPDATE_CONTROLS	The tag runs a comparison between the operator controls and the reference variables. **Attribute:** • **type** The attribute defines the direction of the data comparison. **= TRUE** – data is read from the reference variables and copied to the operator controls. **= FALSE** – data is copied from the operator controls to the reference variables. **Syntax:** `<UPDATE_CONTROLS type = "<Direction>"/>` **Example:** `<SOFTKEY_OK>` `< UPDATE_CONTROLS type="false"/>` `</SOFTKEY_OK>`

3.8 Generating user menus

3.8.1 Creating processing cycle forms

The cycle support function allows the automatic creation and decompilation of a cycle call through the form dialog.

To manage this functionality, the following tags are available:

- **NC_INSTRUCTION**
- **CREATE_CYCLE**

To mark a cycle form, in the **FORM** tag, the attribute **type** should be specified with the value **cycle**. This marking allows the **NC_INSTRUCTION** to be processed.

Example

```
<FORM name = "cycle100_form" type= "CYCLE">
…
…
</FORM>
```

The **NC_INSTRUCTION** tag contains the cycle call to be generated. All cycle parameters should be reserved using space retainers.

Example

```
<FORM name = "cycle100_form" type= "CYCLE">
<NC_INSTRUCTION refvar= "cyc_string" >Cycle100 ($p1, $p2, $p3)</
NC_INSTRUCTION>
…
…
…
</FORM
```

The **CREATE_CYCLE** tag prepares the values saved in the space retainer variables and generates the NC instruction.

This is then copied to the specified variable.

Tag identifier	Description
NC_INSTRUCTION	This tag is used to define the NC instruction to be generated. All listed cycle parameters are automatically created as string variables of the **FORM** and are available to the **FORM**. Precondition: The FORMattribute **type** is set to the value **CYCLE**. **Attribute:** • **refvar** If the tag is assigned a reference variable, all parameters are pre-assigned with the values from the NC block saved in the reference variable. **Syntax:** `<NC_INSTRUCTION> NC instruction with placeholders </NC_INSTRUCTION>` **Example:** `<let name="cyc_string" type="string"> Cycle100(0, 1000, 5)</let>` ... ... `<FORM name = "cycle100_form" type= "CYCLE">` `<NC_INSTRUCTION refvar= "cyc_string" >Cycle100($p1, $p2, $p3)</ NC_INSTRUCTION>` ... ... `</FORM>`

Tag identifier	Description
CREATE_CYCLE	The tag generates an NC block, whose syntax is defined by the value of the **NC_INSTRUCTION** tag. Before generating the NC instruction, the parser calls the CYCLE_CREATE_EVENT tag of the FORM. This tag can be used to calculate the cycle parameters. **Syntax:** `<CREATE_CYCLE/>` **Option:** If a reference variable is specified, the instruction copies the generated call into this variable. `<create_cycle refvar="name" />` **Attribute:** • **refvar** If the tag is assigned a reference variable, the NC instruction is copied to this variable. **Example:** `<LET name="cyc_string" type="string"> Cycle100(0, 1000, 5)</LET>` `<SOFTKEY_OK>` `<caption>OK</caption>` `<CREATE_CYCLE />` `<close_form />` `<navigation>main_menu</navigation>` `</SOFTKEY_OK>` `or` `<SOFTKEY_OK>` `<caption>OK</caption>` `<CREATE_CYCLE refvar= "cyc_string" />` `<close_form />` `<navigation>main_menu</navigation>` `</SOFTKEY_OK>`

3.8.2 Substitution characters

The system offers the option of defining control properties (attribute values) for the runtime. In order to use this function, the desired property must be set in a local variable and the variable name must be transferred to the tag as an attribute value preceded by the **character $**.

If the tag expects a string as attribute value or value, the $$$ characters must be placed in front of the variable name.

Example:

```
<let name="my_ypos">100</let>
<let name="field_name" type="string"></let>

<control name = "edit1" xpos = "322" ypos = "$my_ypos" refvar="nck/
Channel/Parameter/R[1]" />

<op>my_ypos = my_ypos +20 </op>

<control name = "edit2" xpos = "322" ypos = "$my_ypos" refvar="nck/
Channel/Parameter/R[2]" />

<print name ="field_name" text="edit%d">3</print>
<op>my_ypos = my_ypos +20 </op>

<control name = "$field_name" xpos = "322" ypos = "$my_ypos"
refvar="nck/Channel/Parameter/R3]" />

<caption>$$$field_name</caption>
```

3.9 Creating the file struct_def.xml

Procedure

In some functions, the file **struct_def.xml** is used for type definition. If you are missing this XML file, create a file copy with the following markup and integrate it into the function project.

Note

Create the XML file in UTF8 format.

Script

struct_def.xml

```
<!--
Copyright 2015 by Siemens AG and Wolfram Kuhnert
(wolfram.kuhnert@siemens.com)
Permission to use, copy, modify, distribute, and sell this softwareand its
documentation for any purpose is hereby granted without fee,provided that
the above copyright notice appear in all copies and that both that copyright
notice and this permission notice appearin supporting documentation, and
that the names of Siemens AG and Wolfram Kuhnertnot be used in advertising
or publicity pertaining todistribution of the software without specific,
written priorpermission.
Siemens AG and Wolfram Kuhnert make no representations about the
suitability of this software for any purpose. It isprovided "as is" without
express or implied warranty.
Required software version: Operate 4.7 Sp1 HF1
-->

<!--
    typedef struct tagRECT {
     LONG left;
     LONG top;
     LONG right;
     LONG bottom;
    } RECT;
-->

<typedef name="StructRect" type="struct" >
  <element name="left" type="int">0</element>
  <element name="top" type="int">0</element>
  <element name="right" type="int">0</element>
  <element name="bottom" type="int">0</element>
</typedef>

<typedef name="StructPoint" type="struct" >
  <element name="x" type="int">0</element>
  <element name="y" type="int">0</element>
</typedef>

<typedef name="StructSize" type="struct" >
  <element name="width" type="int">0</element>
  <element name="height" type="int">0</element>
</typedef>

<typedef name="StructMDLimits" type="struct" >
  <element name="lowerLimit" type="double">0</element>
  <element name="upperLimit" type="double">0</element>
  <element name="arrayDim1" >0</element>
  <element name="arrayDim2" >0</element>
</typedef>
```

3.10 Addressing components

Address identifiers for the desired data must be created to address NC variables, PLC blocks or drive data. An address consists of the subpaths **component name** and **variable address**. A slash should be used as a separating character.

3.10.1 PLC addressing

Addressing the PLC starts with the path section **plc**.

Table 3-3 The following addresses are permissible:

DBx.DB(f)	Data block
I(f)x	Input
Q(f)x	Output
M(f)x	Bit memory
V(f)x	Variable

DBx.DBXx.b	Data block
Ix.b	Input
Qx.b	Output
Mx.b	Bit memory
Vx.b	Variable

Table 3-4 Data format **f**:

B	Byte
W	Word
D	Double word

Data format identification is not applicable to bit addressing.

Address **x**:

Valid S7-200 address identifier

Bit addressing:

b – Bit number

Examples:

```
<data name = "plc/mb170">1</data>
<data name = "i0.1"> 1 </data>
<op> "m19.2" = 1 </op>
refvar="plc/db9062.dbb0:string"
```

3.10.2 Addressing NC variables

Addressing the NC variables starts with the path section **nck**.

This section is followed by the data address; its structure should be taken from the List Manual NC Variables and Interface Signals.

Example:

```
<LET name = "tempStatus"></LET>
<OP> tempStatus ="nck/channel/state/chanstatus" </OP>
```

3.10.3 Channel-specific addressing

If no channel number is defined in the address token, access is always to Channel 1 of the operating software.

If it is necessary to read data from a specific channel, the identifier **u** (Unit) with the desired channel number is added to the address.

Example:

```
nck/Channel/MachineAxis/actFeedRate[3]
nck/Channel/MachineAxis/actFeedRate[u1, 3]
```

3.10.4 Generating NC/PLC addresses during the runtime

It is possible to generate an address identifier during runtime.

In this case, the content of a string variable is used as address in an operation statement as well as in the nc.cap.read and nc.cap.write functions.

Observe the following for this type of addressing mode:

- Write the variable names in quotation marks.
- Use three ‚$' characters as prefix for variable names.

Syntax:

```
"$$$variable name"
```

Example:

```
<PRINT name="var_adr" text="DB9000.DBW%d"> 2000</PRINT>
<OP> "$$$var_adr" = 1 </OP>
```

3.10.5 Addressing drive components

Addressing the drive components starts with the path section **drive**.

Then the drive device is specified:

CU – Control Unit

DC – Drive Control (Motor Module)

CULNK – Expansion Modules (HUBs)

TM – Terminal Modules

LM – Line Modules

The parameter to be set is added to this section.

Example:

```
<LET name="r0002_content"></LET>
<LET name="p107_content"></LET>

<!— Reading of value r0002 on the CU ->

<OP> r0002_content = "drive/cu/r0002" </OP>
<OP> r0002_content = "drive/cu/r0002[CU1]" </OP>

<!— Reading of value r0002 on NX1 ->
<OP> r0002_content = "drive/cu/r0002[CU2]" </OP>

<!— Reading of value p107[0] on the CU ->
<OP> p107_content = "drive/cu/p107[0]" </OP>
<PRINT text="%d"> p107_content </PRINT>

<!— Reading of value p107[0] on the CU ->

<OP> p107_content = "drive/cu/p107[0, CU1]" </OP>
<PRINT text="%d"> p107_content </PRINT>

<!— Reading of value p107[0] on the NX1 ->

<OP> p107_content = "drive/cu/p107[0, CU2]" </OP>
<PRINT text="%d"> p107_content </PRINT>
```

Addressing the drive objects:

To address individual objects, the desired object should be entered in square brackets after the parameter.

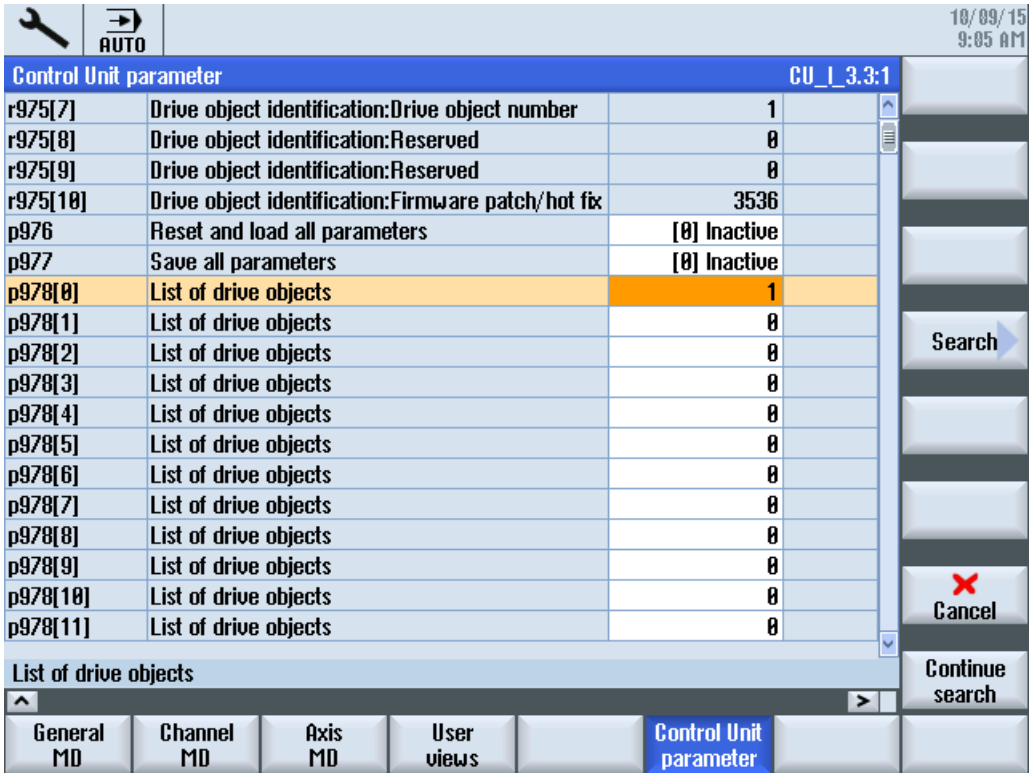


Figure 3-7 Drive parameter p0978 [...] at the control

```
Parameter number[do<DO-index>]
```

Example:
```
p0092[do1]
```

Alternatively, the drive index can be read from a local variable using **$<variable name>** "substitution characters".
```
z.B. DO$local variable
```

Example:
```
<DATA name ="drive/cu/p0092">1</DATA>
<DATA name ="drive/dc/p0092[do1] ">1</DATA>
```

Indirect addressing:
```
<LET name = "driveIndex> 0 </LET>
<OP> driveIndex = $ctrlout_module_nr[0, AX1] </OP>
<DATA name ="drive/dc[do$driveIndex]/p0092">1</DATA>
```

3.10.6 Example: Determine the DO number for the Motor Module

The DO number of a type 11 (servo) Motor Module can be determined as follows:

All connected drive objects are listed with their slot number in field p978 of the relevant CU. The component type numbers are listed in the field p101 and the component types are listed in the field p107 simultaneously.

A separate indexing is to be used for each of the following component types:

CU – Control Units

DC – Drive Controls (Motor Module)

CULNK – Expansion Modules (HUBs)

TM – Terminal Modules

LM – Line Modules

The addressing index can be determined by running through the field p107 in ascending order for each connected CU, and the type index is incremented by one each time the desired type occurs. The basic value is one. If NX components are found in this field, the counting on an NX component is not continued until the current array has been run through completely. The NX and CU components are executed in the order found.

Index determination: CUI with NX

CUI		NX1		Addressing index	
				DO	CU
p107[0]	[3]SINAMICS				1
p107[2]	[11]SERVO			1	
p107[3]	[11]SERVO			2	
p107[4]	[11]SERVO			3	
p107[5]	[254]CU-LINK				2
p107[6]	[11]SERVO			4	
		p107[1]	[11]SERVO	5	
		p107[2]	[11]SERVO	6	
		p107[3]	[11]SERVO	7	

This topology contains seven Motor Modules. Indices one to four address the Motor Modules assigned to the CU. Indices five to seven address the Motor Modules of the NX. Index one is to be used to access the CU. The NX is addressed twice by the index.

Sample scripts: xmldial.xml and drv_sys_hlpfunct.xml

xmldial.xml

```
<DialogGui>

 <?include src="f:\appl\drv_sys_hlpfunct.xml" ?>

 <menu name = "main">
  <open_form name = "main_form" />
  <softkey POSITION="1">
   <caption></caption>
   <navigation>main</navigation>
  </softkey>
 </menu>

 <form name="main_form">
  <init>
   <caption>Component arrangement</caption>
    <let name="count" />
    <let name="str" type="string" />
    <let name="do_name" type="string" />
    <let name="cu_name" type="string" />
    <let name="cui_idx" />

    <op>
     cui_idx = 0;
    </op>

    <function name="load_component" />
    <print text="%d CUs found">num_cus</print>

    <function name="calculate_do_index" />

    <control name="list_comp_no_do_idx" xpos="8" ypos="80"
fieldtype="listbox" width="360" height="500" >
     <property item_data="100" />
    </control>

    <op>
     count = 0;
    </op>

    <while>
    <condition>count &lt; 32 && address_idx_map[$count].comp_no != 0</
condition>

    <op>
     do_name= _T"";
    </op>

    <function name="read_do_name_fast" return="do_name">count</function>
    <function name="read_cu_name_fast"
return="cu_name">address_idx_map[$count].cu_idx</function>

    <print name="str" text="%3d %3d %3d %s
%s">address_idx_map[$count].cu_idx, address_idx_map[$count].comp_no,
address_idx_map[$count].do_idx, do_name, cu_name</print>
```

xmldial.xml

```
    <function name="control.additem">_T"list_comp_no_do_idx", str, count</
function>

    <op>
     count = count +1;
    </op>
    </while>

   <paint>
    <text xpos="8" ypos="30">Motor moduls</text>
    <text xpos="8" ypos="60">CU</text>
    <text xpos="40" ypos="60">Comp.</text>
    <text xpos="92" ypos="60">DO Index</text>
    <text xpos="172" ypos="60">Name</text>
  </paint>

 </form>
</DialogGui>
```

drv_sys_hlpfunct.xml

```
<typedef name="components" type="struct">
 <element name="cu_p978" dim="25" />
 <element name="do_p0101" dim="25" />
 <element name="do_p0107" dim="25" />
</typedef>

<typedef name="comp_no_do_idx_map" type="struct">
 <!-- cu index -->
 <element name="cu_idx" />
 <!-- component number -->
 <element name="comp_no" />
 <!-- address index -->
 <element name="do_idx" />
</typedef>

<let name="componentsList" dim="5" type="components" />
<let name="address_idx_map" dim="32" type="comp_no_do_idx_map" />
<let name="num_cus" />
<let name="_drv_sys_comp_array_size">23</let>

<!-- ------------------------------------------------------------

 function: load_components_description
 This function loads the parameter arrays p978, p101 and p107 into a local
memory

 input:
 cuno: CU index 0 based (index 0 == CUI)

 output:
 componentsList structure

 ------------------------------------------------------------------- -->

<function_body name="load_components_description" parameter="cuno">
 <let name="count" />
 <let name="next_nx" >0</let>
 <let name="cuidx"></let>
 <let name="error" />

 <op>
  count = _drv_sys_comp_array_size;
  next_nx = cuno;
  cuidx = cuno+1;
 </op>

 <print text="gather data" />

 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].cu_p978, "drive/cu/p0978[0, cu
$cuidx]"</function>
 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].do_p0101, "drive/cu/p0101[0, cu
$cuidx]"</function>
```

drv_sys_hlpfunct.xml

```
 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].do_p0107, "drive/cu/p0107[0, cu
$cuidx]"</function>

 <print text="gather data finished" />
 <sleep value="20" />

 <op>
  count = 0;
 </op>

 <while>
  <condition>count < _drv_sys_comp_array_size</condition>
  <if>
   <condition>componentsList[$cuno].cu_p978[$count] == 60</condition>
  <then>
   <op>
    next_nx= next_nx +1;
   </op>
  <print text="next nx %d">next_nx</print>
  <function name="load_components_description">next_nx</function>
  </then>
  </if>

 <op>
  count = count+1;
 </op>
 </while>
 <op>
  num_cus = next_nx+1;
 </op>
</function_body>

<!-- ------------------------------------------------------------

 function: calculate_do_index
 This function is looking for components of type 11 (SERVO) and lists these
in the componentsList array.

 input:
 -

 output:
 componentsList array filled

 ------------------------------------------------------------------- -->

<function_body name="calculate_do_index">
 <let name="cuno" />
 <let name="count" />
 <let name="do_index" >1</let>
 <let name="map_index" />
 <while>
  <condition>
   cuno < num_cus</condition>
   <op>
```

drv_sys_hlpfunct.xml

```
    count = 0;
   </op>
  <while>
   <condition>count < _drv_sys_comp_array_size</condition>
   <if>
    <condition> componentsList[$cuno].do_p0107[$count] == 11</condition>
    <then>
     <op>
      address_idx_map[$map_index].cu_idx = cuno+1;
      address_idx_map[$map_index].comp_no =
componentsList[$cuno].do_p0101[$count];
      address_idx_map[$map_index].do_idx = do_index;
      do_index = do_index+1;
      map_index = map_index +1;
     </op>
    </then>
   </if>
   <op>
    count = count +1;
   </op>
  </while>
  <op>
   cuno = cuno +1;
  </op>
 </while>
</function_body>
```

3.10.7 Addressing machine and setting data

Drive and setting data is identified by the character $ followed by the name of the data.

Machine data:

$Mx_<name[**index, AX<axis_number>**]>

Setting data:

$Sx_<name[index, AX<axis_number>]>

x:

N – General machine or setting data

C – Channel-specific machine or setting data

A – Axis-specific machine or setting data

Index:

For a field, the parameter indicates the index of the data.

AX<axis_number>:

The required axis (**<axis_number>**) has to be specified for axis-specific data.

Alternatively, the axis index can be read from a local variable using **$<variable name>** "substitution characters".

e.g. AX$localvariable

Example:

```
<DATA name ="$MN_AXCONF_MACHAX_NAME_TAB[0] ">X1</DATA>
```

Direct addressing of the axis:

```
<DATA name ="$MA_CTRLOUT_MODULE_NR[0, AX1] ">1</DATA>
...
...
```

Indirect addressing of the axis:

```
<LET name ="axisIndex"> 1 </LET>
<DATA name ="$MA_CTRLOUT_MODULE_NR[0, AX$axisIndex] ">1</DATA>
```

3.10.8 Channel-specific machine data

If no channel number is defined in the address token, access is always to the currently set channel of the operating software.

If it is necessary to read data from a specific channel, the identifier **u** (Unit) with the desired channel number in square brackets is added to the address. In an array-addressing, the channel definition is the last argument in the square brackets.

Example:

```
$MC_RESET_MODE_MASK
```

or

```
$MC_AXCONF_GEOAX_ASSIGN_TAB[0]
```

```
$MC_RESET_MODE_MASK[u1]
```

or

```
$MC_AXCONF_GEOAX_ASSIGN_TAB[0, u1]
$MC_RESET_MODE_MASK
```

3.10.9 Addressing user data

Addressing the user data starts with the path part **gud**, followed by the marking indicating whether the GUDs are global or channel-specific. This address part is followed by the GUD area and the GUD name.

For a field, after the name, the required field index should be specified in square brackets.

Example:
```
<DATA name ="gud/nck/mgud/syg_rm[0]" />
<OP>"gud/nck/mgud /syg_rm[0]" = 10 </op>
```

Addressing the global user data

Addressing starts with the path section gud, followed by the specification of the area NCK. This address section is followed by the specification of the GUD areas:

GUD areas	Assignment
sgud	Siemens GUD
mgud	Machine manufacturer GUD
ugud	User GUD

Addressing the channel-specific user data

Addressing starts with the path section gud, followed by the specification of the area CHANNEL. This address section is followed by the specification of the GUD areas.

Then enter the GUD name. If an array is to be addressed, the name is followed by the array subscript in square brackets.

Example:
```
<data name ="gud/channel/mgud/syg_rm[0]">1</data>
<op>"gud/channel/mgud/syg_rm[0]" = 5*2 </op>
```

Addressing multi-dimensional arrays

In the addressing of multi-dimensional arrays, the line index is expected to be followed by the column index. The indices are to be entered separated from one another by a period.

The following applies to channel-specific GUDs:

- The channel number must be noted with the marking **u** (unit) followed by the number before or after the lines/columns specification.
- The sequences must be separated by a comma.
- If no channel number is entered, the first channel is accessed.

Example:
```
"gud/Channel/sgud/_WP[u2, 2.0]"
```

or
```
"gud/Channel/sgud/_WP[2.0, u2]"
```

3.11 Predefined functions

The script language offers various string processing and standard mathematical functions. The function names listed below are reserved and cannot be overloaded.

Function name	Meaning
Ncfunc cap read	The function copies a value from the specified address into a local variable. If the read operation was error-free, then the return variable contains the value zero. Contrary to the operation instruction, in the event of a fault, this function does not interrupt the processing of the script operations. **Attributes:** • **return** - execution status – Value = 0 - fault-free – Value = 1 - The variables could not be read • **rows** - number of additional lines of an array to be read (optional) If an array index is defined for the reference variables, as from this index, the function copies the values read into the target variable. **Syntax:** `<function name="ncfunc.cap.read" return="error"> lokale variable, "address"</function>` **Example:** `<let name="error"></let>` `<function name="ncfunc.cap.read" return="error"> 3, "drive/cu/p0009"</function>` `<if>` `<condition>error != 0</condition>` `<then>` `<break />` `</then>` `</if>` or `<let name="cu_p978" dim="25" ></let>` `...` `...` `<function name="ncfunc.cap.read" return="error" rows="23">cu_p978, "drive/cu/p0978[0, 1]"` `</function>`

Function name	Meaning
Ncfunc cap write	The function writes a value into the specified variable. If the write operation was error-free, then the return variable contains the value zero. Contrary to the operation instruction, in the event of a fault, this function does not interrupt the processing of the script operations. **Attributes:** • **return** - execution status – Value = 0 - fault-free – Value = 1 - The variables could not be read • **rows** - number of additional lines of an array to be written (optional) If an array index is defined for the reference variables, as from this index, the function copies the values into the target variable. **Syntax:** `<function name="ncfunc.cap.read" return="error"> local variable or constant, "address"</function>` **Example:** `<let name="error"></let>` `<function name="ncfunc.cap.write" return="error"> 0, "drive/cu/p0009"</function>` `<if>` `<condition>error != 0</condition>` `<then>` `<break />` `</then>` `</if>` or `<let name="cu_p978" dim="25" ></let>` `...` `...` `<function name="ncfunc.cap.write" return="error" rows="23">cu_p978, "drive/cu/p0978[0, 1]"` `</function>`

Function name	Meaning
Ncfunc PI-Service	Jobs can be transferred to the NCK using the program invocation (PI) service. If the service has been executed error-free, the function returns the value 1 in the return variable. **Manipulation of the tool list:** _N_CREATO - Create tool _N_DELETO - Delete tool _N_CREACE - Create tool cutting edge _N_DELECE - Delete tool cutting edge **Activate work offsets**: _N_SETUFR - Activates the actual user frame _N_SETUDT - Activates the actual user data **Block search**: _N_FINDBL - Activate block search _N_FINDAB - Cancel block search **Reconfigure machine data:** _N_CONFIG - Machine data with activation level NEW_CONF, which the operator entered one after the other, are simultaneously activated **Syntax:** `<function name="ncfunc.pi_service" return="return var"> pi name, var1, var2, var3, var4, var5` `</function>` **Attributes:** name - function name return- Name of the variable in which the execution result is saved: • Value == 1 – job executed successfully • Value == 0 – faulty job **Tag values:** pi name - Name of the PI service (string) var1 to var5 - PI specific arguments

Function name	Meaning
Ncfunc PI-Service continued	**Arguments:** • _N_CREATO var1 - tool number • _N_DELETO var1 - tool number • _N_CREACE var1 - Tool number var2 - Cutting edge number • _N_DELECE var1 - tool number var2 - Cutting edge number • _N_SETUFR No arguments • _N_SETUDT var1- User data area to be activated – 1 - Tool offset data – 2 - Active basic frame – 3 - Active adjustable frame • _N_FINDBL var1 - Search mode – 2 - Search with contour calculation – 4 - Search for the block end point – 1 - Block search without calculation • _N_FINDAB No arguments • _N_CONFIG No arguments **Example:** Creating a tool – tool number 3 `<function name="ncfunc.pi_service">_T"_N_CREATO", 3</function>` Delete cutting edge 1 of tool 5 `<function name="ncfunc.pi_service">_T"_N_DELECE", 5, 1` `</function>`

Function name	Meaning
ncfunc chan PI-Service	The function executes a PI service in a channel-related manner. The channel number is passed after the PI service name. This is followed by all other call parameters. **Parameter:** **channel** - Channel number **Syntax:** `<function name="ncfunc.chan_pi_service" return="error">` `_T"_N_SETUFR", channel, ...` `</function>` **Example:** `<let name="chan" >1</let>` `<function name="ncfunc.chan_pi_service" return="error">` `_T"_N_SETUFR", chan</function>` `<function name="ncfunc.chan_pi_service" return="error">` `_T"_N_SETUDT", chan, _T"016", _T"00000", _T"00000"</function>`
Ncfunc display resolution	This function supplies the conversion rule for floating point numbers defined in the control. A string variable must be provided as variable. See also display machine data MD203 DISPLAY_RESOLUTION and MD204 DISPLAY_RESOLUTION_INCH **Syntax:** `<function name="ncfunc.displayresolution" return="dislay_res" />` **Example:** `<let name="dislay_res" type="string"></let>` ... `<function name="ncfunc.displayresolution" return="dislay_res" />` `  <control name = "cdistToGo" xpos = "210" ypos = "156"` `refvar="nck/Channel/GeometricAxis/progDistToGo[2]" hotlink="true"` `height="34" fieldtype="readonly" format="$$$dislay_res"` `time="superfast" color_bk="#ffffff"/>`

Function name	Meaning
Ncfunc Get drive by axis name	This function returns the addressing index of a motor module by specifying the associated axis name. The function returns the value -1 if the assignment cannot be determined. This can happen, for example, if the axis/drive assignment has not been made. **Parameter:** **str** - Axis name **Return value:** Index of the motor module - Name on a variable in which the determined index is written. **Syntax:** `<function name="NCFUNC.GETDRVBYAX_NAME" return="<index>"> <axis name></function>` **Example:** `<let name="drv_idx" /> <function name="NCFUNC.GETDRVBYAX_NAME" return="drv_idx">_T"X1" </function> <control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]" hotlink="true" color_bk="#f1f1f1"> <item value="0">Ready</item> <item value="1">Quick commissioning</item> <item value="2">Power unit commissioning</item> <item value="3">Motor commissioning</item> <item value="4">Encoder commissioning</item> <item value="5">Technological application/units </item> <item value="15">Data sets</item> <item value="17">Basic positioning commissioning </item> <item value="25">Commissioning the position control</item> <item value="29">Download</item> <item value="30">Parameter reset</item> <item value="95">Safety Integrated commissioning </item> </control> </init>`

Function name	Meaning
Ncfunc Get drive by axis index	This function returns the addressing index of a motor module by specifying the associated axis index. The function returns the value -1 if the assignment cannot be determined. This can happen, for example, if the axis/drive assignment has not been made. **Parameter:** **index** - Axis index **Return value:** Index of the motor module - Name on a variable in which the determined index is written. **Syntax:** `<function name="NCFUNC.GETDRVBYAX_IDX" return="<index>"> <axis index></function>` **Example:** `<let name="drv_idx" /> <function name="NCFUNC.GETDRVBYAX_IDX" return="drv_idx">1</function> <control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]" hotlink="true" color_bk="#f1f1f1"> <item value="0">Ready</item> <item value="1">Quick commissioning</item> <item value="2">Power unit commissioning</item> <item value="3">Motor commissioning</item> <item value="4">Encoder commissioning</item> <item value="5">Technological application/units</item> <item value="15">Data sets</item> <item value="17">Basic positioning commissioning</item> <item value="25">Commissioning the position control</item> <item value="29">Download</item> <item value="30">Parameter reset</item> <item value="95">Safety Integrated commissioning</item> </control> </init>`

Function name	Meaning
Ncfunc Get drive by drive name	This function returns the addressing index of a motor module by specifying the motor module name. The function returns the value -1 if the assignment cannot be determined. This can happen, for example, if the axis/drive assignment has not been made. **Parameter:** **string** - Name of motor module **Return value:** Index of the motor module - Name on a variable in which the determined index is written. **Syntax:** `<function name="NCFUNC.GETDRVBYDRV_NAME" return="<index>"> <drive name></function>` **Example:** `<let name="drv_idx" /> <function name="NCFUNC.GETDRVBYDRV_NAME" return="drv_idx">_T"SERVO_3.13:4"</function> <control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]" hotlink="true" color_bk="#f1f1f1"> <item value="0">Ready</item> <item value="1">Quick commissioning</item> <item value="2">Power unit commissioning</item> <item value="3">Motor commissioning</item> <item value="4">Encoder commissioning</item> <item value="5">Technological application/units </item> <item value="15">Data sets</item> <item value="17">Basic positioning commissioning </item> <item value="25">Commissioning the position control</item> <item value="29">Download</item> <item value="30">Parameter reset</item> <item value="95">Safety Integrated commissioning </item> </control> </init>`

Function name	Meaning
Ncfunc Get drive by bud address	This function returns the addressing index of a motor module by specifying the bus address of the motor module. The function returns the value -1 if the assignment cannot be determined. This can happen, for example, if the axis/drive assignment has not been made. **Parameter:** **bus** - Bus number **slave** - Slave number **component** - component number **Return value:** Index of the motor module - Name on a variable in which the determined index is written. **Syntax:** `<function name="NCFUNC.GETDRVBYBUS_ADDR" return="<index>"><bus>,<slave>,<component></function>` **Example:** `<let name="drv_idx" />` `<function name="NCFUNC.GETDRVBYBUS_ADDR" return="drv_idx"> 3,13,4 </function>` `<control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]" hotlink="true" color_bk="#f1f1f1">` `<item value="0">Ready</item>` `<item value="1">Quick commissioning</item>` `<item value="2">Power unit commissioning</item>` `<item value="3">Motor commissioning</item>` `<item value="4">Encoder commissioning</item>` `<item value="5">Technological application/units </item>` `<item value="15">Data sets</item>` `<item value="17">Basic positioning commissioning </item>` `<item value="25">Commissioning the position control</item>` `<item value="29">Download</item>` `<item value="30">Parameter reset</item>` `<item value="95">Safety Integrated commissioning </item>` `</control>` `</init>`

Function name	Meaning
Ncfunc Get MD limits	The function copies the limits of a machine data to the specified variable, data type **StructMDLimits**. The structure definition is contained in file struct_def.xml. **More information** is provided in Chapter "Creating the file struct_def.xml (Page 138)" **Parameter:** **range** - A string should be transferred that specifies the machine data area: • **displayMD** - Display machine data • **generalMD** - NC global machine data • **channelMD** - NC channel-specific machine data • **axisMD** - NC axis-specific machine data • **generalSettingMD** - NC global setting data • **channelSettingMD** - NC channel-specific setting data • **axisSettingMD** - NC axis-specific setting data • **channelGUD** - NC channel-specific user data • **globalGUD** - NC global user data **MD-number** - Machine data number of the area **variable name** - After the function call, this variable contains the value of the limits **range/unit** - Optional (e.g. channel number) **Syntax:** `<function name="NCFUNC.GETMD_LIMITS"><range>, <MD-number, <variabel name>, <range/unit>></function>` **Example:** `<let name="limits" type="StructMDLimits" />` `<function name="NCFUNC.GETMD_LIMITS">_T"generalMD", 19220, _T"limits"</function>` `<op>` `upper_BAGS = limits.upperLimit;` `</op>`
Ncfunc bico to int	The function converts a string specified in BICO format into an integer value. (see SINAMICS). **Syntax:** `<function name="ncfunc.bicotoint" return="integer variable">bico-string</function>` **Example:** `<let name="s_np0480_0" type="string"></let>` `<let name="i_p0480_0">0</let>` `<function name="ncfunc.bicotoint" return="i_p0480_0">s_np0480_0</function>`

Function name	Meaning
Ncfunc int to bico	The function converts an integer value into a BICO format string. (see SINAMICS). **Syntax:** `<function name="ncfunc.inttobico"` `return="string variable">integer variable</function>` **Example:** `<function name="ncfunc.inttobico"` `return="s_p0480_0">"drive/dc/p0480[0, DO2]"</function>`
Ncfunc is bico str valid	This function returns the value zero for a string specified in BICO format. (see SINAMICS) **Syntax:** `<function name="ncfunc.isbicostrvalid"` `return="integer variable">string varaible</function>` **Example:** `…` `<let name="s_np0480_0" type="string"></let>` `…` `…` `<control name = "cp0480_0" xpos = "402"` `ypos = "76" hotlink="true" refvar="s_np0480_0" >` `<property item_data="4001" />` `</control>` `…` `…` `<function name="ncfunc.isbicostrvalid"` `return="valid">cp0480_0</function>`
Ncfunc password	This function sets or deletes a password level from the NC. • Set password: The password should be specified for the required password level as parameter. • Delete password: A blank string deletes the password level. **Syntax:** `<function name="ncfunc.password">password` `</function>` **Example:** `<let name="password" type="string"></let>` `<function name="ncfunc.password" > password` `</function>` `<function name="ncfunc.password" > _T"CUSTOMER"` `</function>` Delete password: `<function name="ncfunc.password" > _T"" </function>`

Function name	Meaning
Control form color	The function returns the text or background color of the dialog box as a string. **More information** is provided in Chapter "Color coding (Page 72)" **Range:** • BACKGROUND – request color value of the background • TEXT – request color value of the text (foreground) **Syntax:** `<function name="control.formcolor" return="variable">_T"Area"</function>` **Example:** `<let name="bk_color" type="string"></let>` `<function name="control.formcolor" return="bk_color">_T"BACKGROUND"</function>`
Control local time	The function copies the local time in a field with 7 array elements. The name of the variable is expected as call parameter. The following is stored in an array element: • Index 0 - year • Index 1 - month • Index 2 - weekday • Index 3 - day • Index 4 - hour • Index 5 - minute • Index 6 - second **Syntax:** `<function name ="control.localtime">_T"time_array" </function>` **Example:** `<!-- index` `0 = Year` `1 = Month` `2 = Day of week` `3 = Day` `4 = Hour` `5 = Minute` `6 = Second` `-->` `<let name="time_array" dim="7" />` `<function name ="control.localtime">_T"time_array" </function>`

Function name	Meaning
Control exist	The function returns the value 1 if the specified control exists. **Syntax:** `<function name="CONTROL.EXIST"` `return="<return variable>"><control name></function>` **Example:** `<let name="ret" /> <function name="CONTROL.EXIST"` `return="ret">_T"edit"</function>`
Control is modified	The function returns the value 1 if the content of the specified edit control was changed. **Syntax:** `<function name="CONTROL.ISMODIFIED"` `return="<return variable>"><control name></function>` **Example:** `<let name="ret" /> <function name="CONTROL.ISMODIFIED"` `return="ret">_T"edit"</function>`
Control is visible	The function returns the value 1 if the specified control is visible. **Syntax:** `<function name="CONTROL.ISVISIBLE"` `return="<return variable>"><control name></function>` **Example:** `<let name="ret" /><function name="CONTROL.ISVISIBLE"` `return="ret">_T"edit"</function>`
Control set modified	The function changes the modified flag of the specified edit control. **Parameter:** **control name** • Value = 1 - mark field as changed • Value = 0 - mark field as unchanged **Syntax:** `<function name="CONTROL.SETMODIFIED"` `return="<return variable>"><control name>,` `<Flag></function>` **Example:** `<let name="b" >1</let>` `<let name="ret" />` `<control name="edit" xpos="10" ypos="80" width="30" refvar="b"` `hotlink = "true" />` `<function name="CONTROL.SETMODIFIED"` `return="ret">_T"edit", 1</function>`

Function name	Meaning
Control set working area	If you change the contents of a **scroll view**, e.g. by deleting or adding controls, the visible area of the scroll view must be recalculated. The calculation is performed by calling this function. **Value:** Name of the **scrollview** **Example:** `...` `<control name="area" xpos="8" ypos="72" width="554" height="320" fieldtype="scrollarea" >` `  <control name="test" xpos="20" ypos="40"  width="20" fieldtype="readonly" />` `</control >` `<control name="test1" xpos="20" ypos="80"  width="20" fieldtype="readonly" parent="area"/>` `<control name="test2" xpos="20" ypos="100"  width="20" fieldtype="readonly" parent="area"/>` `<function name="CONTROL.SETWORKINGAREA">_T"area"</function>` `...`
String to compare	Two strings are compared with one another from a lexicographical perspective. The function gives a return value of zero if the strings are the same, a value less than zero if the first string is smaller than the second string or a value greater than zero if the second string is smaller then the first string. **Parameter:** **str1** - string **str2** - comparison string **Syntax:** `<function name="string.cmp" return ="<int var>" > str1, str2 </function>` **Example:** `<let name="rval">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string">A brown bear hunts a brown dog.</let>` `<function name="string.cmp" return="rval"> str1, str2 </function>` **Result:** rval= 0

Function name	Meaning
String to compare without making a distinction between uppercase/ lowercase	Two strings are compared from a lexicographical perspective (the comparison is not case-sensitive). The function gives a return value of zero if the strings are the same, a value less than zero if the first string is smaller than the second string or a value greater than zero if the second string is smaller then the first string. **Parameter:** **str1** - string **str2** - comparison string **Syntax:** `<function name="string.icmp" return ="<int var>" > str1, str2 </function>` **Example:** `<let name="rval">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string">A brown Bear hunts a brown Dog.</let>` `<function name="string.icmp" return="rval"> str1, str2 </function>` **Result:** rval= 0
String left	The function extracts the first nCount character from string 1 and copies this to the return variable. **Parameter:** **str1** - string **nCount** - number of characters **Syntax:** `<function name="string.left" return="<result string>"> str1,  nCount </function>` **Example:** `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string"></let>` `<function name="string.left" return="str2"> str1, 12 </function>` **Result:** str2="A brown bear"

Function name	Meaning
String right	The function extracts the last nCount character from string 1 and copies this to the return variable. **Parameter:** **str1** - string **nCount** - number of characters **Syntax:** `<function name="string.right" return="<result string>"> str1, nCount </function>` **Example:** `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string"></let>` `<function name="string.right" return="str2"> str1, 10 </function>` **Result:** str2="brown dog."
String middle	The function extracts the specified number of characters from string 1, starting from the iFirst index, and copies these to the return variable. **Parameter:** **str1** - string **iFirst** - start index **nCount** - number of characters **Syntax:** `<function name="string.middle"` `return="<result string>"> str1, iFirst, nCount </function>` **Example:** `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string"></let>` `<function name="string.middle" return="str2"> str1, 2, 5 </function>` **Result:** str2="brown"

Function name	Meaning
String length	The function gives the number of characters in a string. **Parameter:** **str1** - string **Syntax:** `<function name="string.length" return="<int var>"> str1 </function>` **Example:** `<let name="length">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<function name="string.length" return="length"> str1 </function>` **Result:** length = 31
Strings to replace	The function replaces all the substrings found with the new string. **Parameter:** **string** - string variable **find string** - string to be replaced **new string** - new string **Syntax:** `<function name="string.replace"> string, find string, new string </function>` **Example:** `<let name="str1" type="string">A brown bear hunts a brown dog. </let>` `<function name="string.replace" > str1, _T"a brown dog" , _T"a big salmon"</function>` **Result:** str1 = "A brown bear hunts a big salmon!"

Function name	Meaning
Strings to remove	The function removes all the substrings found. **Parameter:** **string** - string variable **remove string** - substring to be deleted **Syntax:** `<function name="string.remove"> string, remove string </function>` **Example:** `<let name="index">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog. </let>` `<function name="string.remove" > str1, _T"a brown dog" </function>` **Result:** str1 = "A brown bear hunts"
Strings to insert	The function inserts a string at the index specified. **Parameter:** **string** - string variable **index** - index (zero based) **insert string** - string to be inserted **Syntax:** `<function name="string.insert"> string, index, insert string` `</function>` **Example:** `<let name="str1" type="string">A brown bear hunts. </let>` `<let name="str2" type="string">a brown dog</let>` `<function name="string.insert" > str1, 19, str2` `</function>` **Result:** str1 = "A brown bear hunts a brown dog"

Function name	Meaning
String delete	The function deletes the defined number of characters starting from the start position specified. **Parameter:** **string** - string variable **start index** - start index (zero based) **nCount** - number of characters to be removed **Syntax:** `<function name="string.delete"> string, start index , nCount` `</function>` **Example:** `<let name="str1" type="string">A brown bear hunts. </let>` `<function name="string.delete" > str1, 2, 5` `</function>` **Result:** str1 = "A bear hunts"
String find	The function searches the transferred string for the first match with the substring. If the substring is found, the function provides the index to the first character (starting with zero) or, failing this, -1. **Parameter:** **string** - string variable **findstring** - string to be found **startindex** – start index (optional) **Syntax:** `<function name="string.find" return="<int val>"> str1, find string` `</function>` **Example:** `<let name="index">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog. </let>` `<function name="string.find" return="index"> str1, _T"brown"` `</function>` **Result:** Index = 2 **or** `<function name="string.find" return="index"> str1, _T"brown", 1` `</function>`

Function name	Meaning
String reverse find	The function searches the transferred string for the last match with the substring. If the substring is found, the function provides the index to the first character (starting with zero) or, failing this, -1. **Parameter:** **string** - string variable **find string** - string to be found **startindex** - start index (optional) **Syntax:** `<function name="string.reversefind"` `return="<int val>"> str1, find string </function>` **Example:** `<let name="index">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog. </let>` `<function name="string.reversefind" return="index"> str1, _T"brown"` `</function>` **Result:** Index = 21 **or** `<function name="string.reversefind"` `return="index"> str1, _T"brown" 10 </function>` **Result:** Index = 2
String GetAt	This function reads a character from the specified position. **Parameter:** **str** - String **index** - Zero-based index for the character being read **Return value:** The name of a string variable must be specified in which the character is to be stored. **Syntax:** `<function name="string.getat" return="<result string>"><string>,` `<index></function>` **Example:** `<let name="resStr" type="string" />` `<function name="string.getat"` `return="resStr">_T"brown", 2</function>`

Function name	Meaning
String pixel height	Calculation of the height of a string in pixels. The calculation is performed using the current font of the specified control or form. The name of the control must be transferred as a string. If an empty string is specified, the calculation refers to the form. **Parameter:** **control name** **Character string** The string can be specified directly as text or a string variable must be specified. **Return:** The name of an integer variable is expected, in which the height is written. **Syntax:** `<function name="STRING.PIXELHEIGHT"` `return="<return variable>"><control name>,` `<varaible or text></function>` **Example:** Calculation of the width in relation to the font of a form `<let name="pixelsh" />` `...` `<function name="STRING.PIXELHEIGHT"` `return="pixelsh">_T"", cedit1</function>` Calculation of the width in relation to the font of a control `<function name="STRING. PIXELHEIGHT"` `return="pixelsh"> cedit1, cedit1</function>`

Function name	Meaning
String pixel width	Calculation of the width of a string in pixels. The calculation is performed using the current font of the specified control or form. The name of the control must be transferred as a string. If an empty string is specified, the calculation refers to the form. **Parameter:** **control name** **Character string** The string can be specified directly as text or a string variable must be specified. **Return:** The name of an integer variable is expected, in which the text width is written. **Syntax:** `<function name="STRING.PIXELWIDTH"` `return="<return variable>"><control name>,` `<varaible or text></function>` **Example:** Calculation of the width in relation to the font of a form `<let name="pixels" />` `...` `<function name="STRING. PIXELWIDTH"` `return="pixels">_T"", cedit1</function>` Calculation of the width in relation to the font of a control `<function name="STRING.PIXELWIDTH"` `return="pixels"> cedit1, cedit1</function>`
String SetAt	This function writes a character to the specified position. The index must be smaller than the maximum text length. **Parameter:** **str** - String **char** - Character which is to be written to the specified position **index** - Zero-based index for the character string of the target variable **Syntax:** `<function name="string.setat" ><destination string>,` `<character string>, <index></function>` **Example:** `<let name="str" type="string" >br_wn</string>` `<function name="string.setat">str, ">_T"o", 2` `</function>`

Function name	Meaning
String Split	This function deconstructs a string into a number of substrings and copies them into the string array specified. Separation is implemented at the specified separator. The separator is not saved in the substring. This function expands the array automatically if the number of separators located is greater than the defined array size. **Parameter:** **str** - String **char** - Name on a variable which includes the separator **number** - Name on a variable which includes the number of generated substrings after the function has been performed. **Return value:** The name of a string array must be specified which includes the substrings after the function has been performed. **Syntax:** `<function name=" string.split"` `return="<result string array>"><string>, <char>,` `<number></function>` **Example:** `<let name="strlist" type="string" dim="2"/>` `<let name="str_num" />` `<function name="string.split" return="strlist">` `_T"brown;green;blue;red", _T";", str_num </function>`
String trim left	The function trims the starting characters from a string. **Parameter:** **str1** - string variable **Syntax:** `<function name="string.trimleft" > str1 </function>` **Example:** `<let name="str1" type="string">test trim left</let>` `<function name="string.trimleft" > str1 </function>` **Result:** str1 = "test trim left"

Function name	Meaning
String trim right	The function trims the closing characters from a string. **Parameter:** **str1** - string variable **Syntax:** `<function name="string.trimright" > str1 </function>` **Example:** `<let name="str1" type="string"> test trim right </let>` `<function name="string.trimright" > str1` `</function>` **Result:** str1 = "test trim right"
Sine	The function calculates the sine of the value transferred in degrees. **Parameter:** **double** - angle **Syntax:** `<function name="sin" return="<double val>"> double </function>` **Example:** `<let name= "sin_val" type="double"></let>` `<function name="sin" return="sin_val"> 20.0` `</function>`
Cosine	The function calculates the cosine of the value transferred in degrees. **Parameter:** **double** - angle **Syntax:** `<function name="cos" return="<double val>"> double </function>` **Example:** `<let name= "cos_val" type="double"></let>` `<function name="cos" return="cos_val"> 20.0` `</function>`

Function name	Meaning
Tangent	The function calculates the tangent of the value transferred in degrees. **Parameter:** **double** - angle **Syntax:** `<function name="tan" return="<double val>"> double </function>` **Example:** `<let name= "tan_val" type="double"></let>` `<function name="tan" return="tan_val"> 20.0` `</function>`
ARCSIN	The function calculates the arcsine of the value transferred in degrees. **Parameter:** **double** - x in the range from -PI/2 to +PI/2 **Syntax:** `<function name="arcsin" return="<double val>"> double </function>` **Example:** `<let name= "arcsin_val" type="double"></let>` `<function name="arcsin" return="arcsin_val"> 20.0` `</function>`
ARCOS	The function calculates the arccosine of the value transferred in degrees. **Parameter:** **double -** x in the range from -PI/2 to +PI/2 **Syntax:** `<function name="arcos" return="<double val>"> double </function>` **Example:** `<let name= "arccos_val" type="double"></let>` `<function name="arccos" return="arccos_val"> 20.0` `</function>`
ARCTAN	The function calculates the arctan of the value transferred in degrees. **Parameter:** **double** - arctan of y/x **Syntax:** `<function name="arctan" return="<double val>"> double </function>` **Example:** `<let name= "arctan_val" type="double"></let>` `<function name="arctan" return="arctan_val"> 20.0` `</function>`

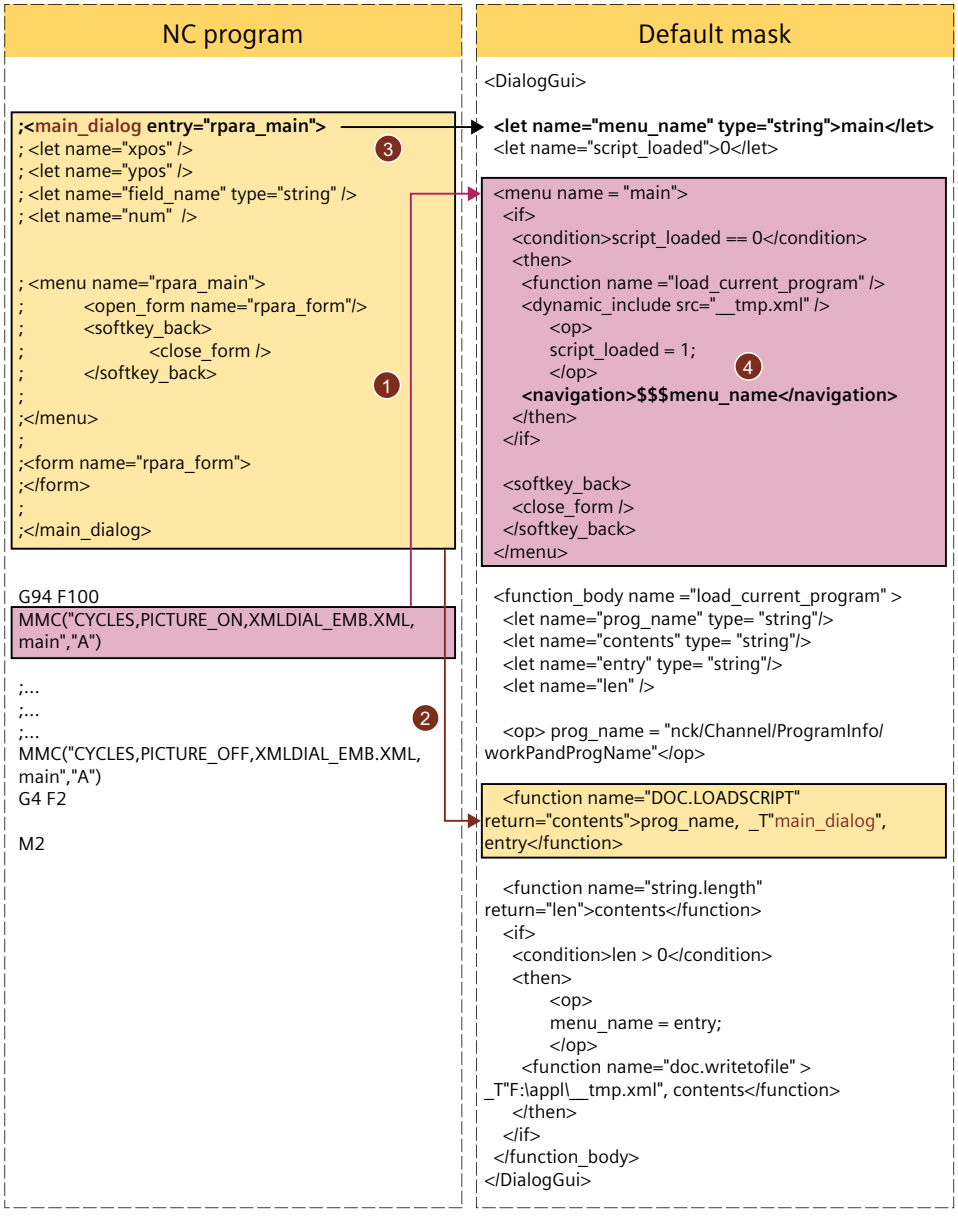
Function name	Meaning
File processing	
Reading a file	The function reads the contents of the specified file into a string variable. The number of characters to be read can optionally be specified as a second parameter. **Attribute:** **name -** The file name should be written in lowercase letters. Files in other directories are accessed via a relative path that uses the appl or dvm directory as a starting point. **return** - name of the local variable **Parameter:** **progname** - file name **number of characters** - number of characters to be read in bytes (optional) **Syntax:** `<function name="doc.readfromfile"` `return="<string var>"> progname, number of characters` `</function>` **Example:** `<let name = "my_var" type="string" ></let>` NC file system `<function name="doc.readfromfile"` `return="my_var"> _T"n:\mpf\test.mpf"` `</function>` CompactFlash card `<function name="doc.readfromfile"` `return="my_var"> _T"f:\appl\test.mpf"` `</function>` or `<function name="doc.readfromfile"` `return="my_var"> _T".\test.mpf"` `</function>`

Function name	Meaning
Writing to a file	The function writes the contents of a string variable to the file specified. The file name should be written in lowercase letters. Files in other directories are accessed via a relative path that uses the appl or dvm directory as a starting point. **Parameter:** **progname** - file name **str1** - string **Syntax:** `<function name="doc.writetofile" > progname, str1 </function>` **Example:** `<let name = "my_var" type="string" > file content </let>` NC file system `<function name="doc.writetofile">_T"n:\mpf\test.mpf", my_var` `</function>` CompactFlash card `<function name="doc.writetofile">_T"f:\appl\test.mpf", my_var` `</function>` or `<function name="doc.writetofile">_T".\test.mpf", my_var </function>`

Function name	Meaning
Deleting a file	The function removes the file specified from the directory. The file name should be written in lowercase letters. Files in other directories are accessed via a relative path that uses the appl or dvm directory as a starting point. **Parameter:** **progname** - file name **Syntax:** `<function name="doc.remove" > progname </function>` **Example:** NC file system `<function name="doc.remove">_T"n:\mpf\test.mpf"` `</function>` CompactFlash card `<function name="doc.remove">_T"f:\appl\test.mpf"` `</function>` or `<function name="doc.remove">_T".\test.mpf"` `</function>`

Function name	Meaning
Extracting script parts	The function copies a dialog description embedded in a part program into the specified local variable. The call parameters to be specified are the program name, the dialog name, and a variable for storing the main menu name. If the name of the dialog description was found in the part program, the return variable contains this description. If the content of the variable is stored in a file, the script can be executed with an indirect call. The system provides a script that extracts the dialog description from the active part program and activates the dialog. This script can be called in an MMC command to activate the screen associated with the part program. **Syntax:** `<function name="doc.loadscript"` `return="<name of script variable>">progname, _T"dialog part name",` `main menu </function>` **Attribute**: **return** - variable in which the extracted script is stored **Parameter:** **progname** - full path to the program. (The path name can be passed to the function in DOS notation.) **main menu** - the menu name found is copied into this variable **dialog part name** - tag name in which the dialog description is embedded **Example:** `<function name="doc.loadscript" return="contents">prog_name,` `_T"main_dialog", entry</function>`

Function name	Meaning
Extracting script parts continued	**Example diagram:**

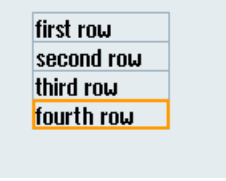


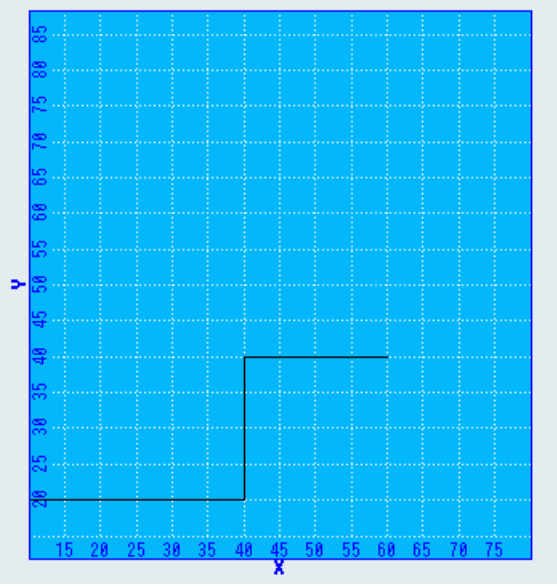
Function name	Meaning
Exist	If the file exists, the function returns the value 1. The file name should be written in lowercase letters. Files in other directories are accessed via a relative path that uses the appl or dvm directory as a starting point. **Parameter:** **progname** - file name **Syntax:** `<function name="doc.exist" return="<int_var>" > progname </function>` **Example:** `<let name ="exist">0</let>` NC file system `<function name="doc.exist" return="exist">_T"n:\mpf\test.mpf" </function>` CompactFlash card `<function name="doc.exist" return="exist">_T"f:\appl\test.mpf" </function>` or `<function name="doc.exist" return="exist">_T".\test.mpf" </function>`
NC program selection	The function selects the program specified for execution. The program must be stored in the NC file system. **Parameter:** **progname** - file name **Syntax:** `<function name="ncfunc.select"> progname </function>` **Example:** NC file system `<function name="ncfunc.select"> _T"n:\mpf\test.mpf" </function>`

Function name	Meaning
Setting an individual bit	The function is used to manipulate individual bits of the specified variables. The bits can either be set or reset. **Syntax:** `<function name="ncfunc.bitset" refvar="address" value="set/reset" > bit0, bit1, … bit9 </function>` **Attributes:** **refvar** - specifies the name of the variable, in which the bit combination should be written **value** – bit value, value range 0 and 1 **Values:** The bit numbers starting with zero should be transferred as function values. A maximum of 10 bits per call can be modified. **Example:** `<function name="ncfunc.bitset" refvar="nck/Channel/Parameter/R[1]" value="1" > 0, 2, 3, 7 </function>` `<function name="ncfunc.bitset" refvar="nck/Channel/Parameter/R[1]" value="0" > 1, 4 </function>`
Delete control	The function deletes the specified picture control. **Syntax:** `<function name="control.delete"> control name </function>` **Attribute:** **name** – function name **Value:** **control name** – name of the control **Example:** `<function name="control.delete"> _T"my_editfield" </function>`

Function name	Meaning
Add Item	The function inserts a new element at the end of the list. Note: The function is only available for the control types "listbox" and "graphicbox". **Syntax:** `<function name="control.additem"> control name, item </function>` **Attribute:** **name** – function name **Values:** **control name** – control name **item** - expression to be inserted **itemdata** - integer value; defined by the user **Example:** `<let name ="itemdata">1</let>` … … … `<op> item_string = _T"text1" </op>` `<function name="control.additem">_T"listbox1", item_string, itemdata </function>`

Function name	Meaning
Insert Item	The function inserts a new element at the specified position. Note: The function is only available for the control type "listbox". **Syntax:** `<function name="control.insertitem"> control name, index, item, itemdata </function>` **Attribute:** **name** – function name **Values:** **control name** - Control name **index** - Position starting with zero **item** - expression to be inserted **itemdata** - integer value; defined by the user **Example:** `<let name ="itemdata">1</let>` … … … `<op> item_string = _T"text2" </op>` `<function name="control.insertitem">_T"listbox1", 1, item_string, itemdata </function>`
Delete item	The function deletes an element at the specified position. Note: The function is only available for the control type "listbox". **Syntax:** `<function name="control.deleteitem"> control name, index </function>` **Attribute:** **name** - function name **Values:** **control name** - Control name **index**– index starting at 0 **Example:** `<function name="control.deleteitem">_T"listbox1", 1</function>`

Function name	Meaning
Load Item	The function inserts a list of expressions into the control. The function is only available for the control types "listbox" and "graphicbox". **Syntax:** `<function name="control.loaditem"> control name, list </function>` **Attribute:** **name** – function name **Values:** **control name** – control name **list**- string variable The string contained in the variable must conform to the structure described below. **Structure of the list:** The list contains a number of expressions, which must be separated from one another using a \\n. **Example:** In the example, the content is assigned to the list box with the function control.loaditem. The control character \\n separates expressions belonging to the line. `<CONTROL name = "list" xpos = "160" ypos = "32"` `width ="100" height="200" fieldtype="listbox"/>` `<let name="lb_list" type="string" />` `<op>` `lb_list = _T"first row\\n";` `lb_list = lb_list + _T"second row\\n";` `lb_list = lb_list + _T"third row\\n";` `lb_list = lb_list +_T"fourth row\\n";` `</op>` `<function name="control.loaditem">_T"list",` `lb_list</function>` 

Function name	Meaning
Load Item continued	**Example:** In the example, the content is assigned to the Graphicbox with the function control.loaditem. The control character \\n separates the graphical elements. `<control name= "c_gbox" xpos = "6" ypos="32" width="328" height="356" fieldtype="graphicbox" />` `<let name="plot_list" type="string" />` `<op>` `plot_list = _T"l;10;20;40;20\\n";` `plot_list = plot_list + _T"l;40;20;40;40\\n";` `plot_list = plot_list + _T"l;40;40;60;40\\n";` `plot_list = plot_list +_T"l;80;0;80;100\\n";` `</op>` `<function name="control.loaditem">_T"c_gbox", plot_list</function>` 
Empty	The function deletes the contents of the specified list box or graphic box controls. **Syntax:** `<function name="control.empty"> control name, </function>` **Attribute:** **name** – function name **Values:** **control name** – control name **Example:** `<function name="control.empty">_T"listbox1" </function>`

Function name	Meaning
Get focus	The function supplies the name of the control, which has the input focus. **Syntax:** `<function name="control.getfocus" return="focus_name" />` **Attributes:** **name** – function name **return** – a string variable should be specified, into which the control name is copied. **Example:** `<let name>="focus_field" type="string"></let>` `<function name="control.getfocus" return="focus_field"/>`
Set focus	The function sets the input focus to the specified control. The Controlname should be transferred as text expression of the function. **Syntax:** `<function name="control.setfocus"> control name` `</function>` **Attribute:** **name** - function name **Value:** **control name** - name of the control **Example:** `<function name="control.setfocus" ">_T"listbox1"` `</function>`
Get cursor selection	For a list box, the function supplies the cursor index. The Controlname should be transferred as text expression of the function. **Syntax:** `<function name="control.getcurssel" return="var">control name` `</function>` **Example:** `<let name>="index"></let>` `<function name="control.getcurssel" return="index">_T"listbox1"` `</function>`

Function name	Meaning
Set cursor selection	For a list box, the function sets the cursor to the appropriate line. The Controlname should be transferred as text expression of the function. **Syntax:** `<function name="control.setcurssel" > control name, index </function>` **Example:** `<let name>="index">2</let>` `<function name="control.setcurssel" ">_T"listbox1",index </function>`
Get Item	For a list box, the function copies the contents of the selected line to the specified variable. A string variable should be specified as reference variable. The Controlname should be transferred as text expression of the function. **Syntax:** `<function name="control.getitem" return="var"> control name, index </function>` **Example:** `<let name>="index">2</let>` `<let name>="item" type="string"></let>` `<function name="control.getitem" return="item" ">_T"listbox1",index </function>`
Get Item Data	For a list box, the function copies the user-specific allocated value of an element to the specified variable. For an edit control, the function copies the user-specific allocated value (item_data) to the specified variable. An integer variable should be specified as reference variable. The Controlname should be transferred as text expression of the function. **Syntax:** `<function name="control.getitemdata" return="var"> control name, index </function>` **Example:** `<let name>="index">2</let>` `<let name>="itemdata"></let>` `<function name="control.getitemdata" return="itemdata" ">_T"listbox1",index</function>`

Function name	Meaning
Image Box Set	This function is used to control the visible area of the controls **Imagebox** and **Graphicbox**. The call parameters to be specified are the control name, the control command, and the associated values. **Syntax:** `<function name="control.imageboxset">control name, command, command parameter</function>` **Call parameters:** • **x_offs** The function shifts the image portion to the specified X-position. A further parameter to be specified is the coordinate of the left corner. • **y_offs** The function shifts the image portion to the specified Y position. A further parameter to be specified is the coordinate of the upper corner. • **xy_offs** The function shifts the image portion to the specified X/Y-positions. Further parameters to be specified are the coordinates of the left and upper corners. • **followCursor** The image portion follows the set cursor position. • **zoomplus** The image is enlarged by a factor of 0.12. • **zoomminus** The image is reduced by a factor of 0.12. • **autozoom** The image will automatically be scaled to fit the display area. • **Update** The control is redrawn. • **SetBkColor** The command sets the specified background color. A further parameter to be specified is the color coding. • **SetCursorRect** The portion specified as a rectangle is shifted into the visible area. • **SetAnimationState** (only applies to the Imagebox control) – **start** - The command starts an animation. – **stop** - The command stops an animation.

Function name	Meaning
Image Box Set continued	**Example:** `...` `...` `<softkey POSITION="1">` `<caption>zoom%nin</caption>` `<function name="control.imageboxset">_T"c_gbox",` `_T"zoomplus"</function>` `</softkey>` `...` `...` `<form name = "main_form">` `<init>` `<caption> graphicbox</caption>` `<control name= "c_gbox" xpos = "6" ypos="32"` `width="328" height="356" fieldtype="graphicbox" />` `<let name="plot_list" type="string" />` `<op>` `plot_list = _T"1;10;20;40;20\\n";` `plot_list = plot_list + _T"1;40;20;40;40\\n";` `plot_list = plot_list + _T"1;40;40;60;40\\n";` `plot_list = plot_list +_T"1;80;0;80;100\\n";` `</op>` `<function name="control.loaditem">_T"c_gbox",` `plot_list</function>` `</init>` `</form>`
Image Box Get	This function is for querying the control properties image box. The call parameters to be specified are the control command and the associated values. **Syntax:** `<function name="control.imageboxget">control name, command,` `command parameter</function>` **Call parameters:** **GetViewSize** Returns the size of the display area A further parameter to be specified is the variable with structure type SIZE. **Example:** `<let name="view_size" type="size" />` `<function name="control.imageboxget">_T"topoview", _T"`**`GetViewSize`**`",` `view_size</function>`

Function name	Meaning
Bitmap Dim	The function copies the dimension of a bitmap back into a variable with structure type SIZE. To define the type, file struct_def.xml must be included in the project. **More information** is provided in Chapter "Creating the file struct_def.xml (Page 138)" **Syntax:** `<function name="bitmap.dim" >name, variable type` `</function>` **Parameter:** **name** - file path **variable type** - variable name of a variable of type SIZE **Example:** `<let name="bmp_size" type="size" />` `<function name="bitmap.dim" >_T"test.bmp", bmp_size` `</function>`
Screen Resolution	The function copies the absolute screen resolution of the system back into a variable with structure type SIZE. To define the type, file struct_def.xml must be included in the project. **More information** is provided in Chapter "Creating the file struct_def.xml (Page 138)" **Syntax:** `<function name="hmi.screen_resolution" >varialble type </function>`
Get HMI Resolution	The function copies the screen resolution used by SINUMERIK Operate back into a variable with structure type SIZE. To define the type, file struct_def.xml must be included in the project. **More information** is provided in Chapter "Creating the file struct_def.xml (Page 138)" **Syntax:** `<function name="hmi.get_hmi_resolution" >variable type </function>`
Get Caption Height	The function returns the title bar height in pixels. **Syntax:** `<function name="hmi.get_caption_height" return="<return var>" />` **Attributes:** **return** - integer variable

Function name	Meaning
Get Font Families	The function provides a list of the installed fonts. The font names are listed separated by an LF character. The name of a string variable is to be passed on as the return value. **Syntax:** `<function name="HMI.GET_FONT_FAMILIES"` `return="< String-Variable>" />` **Example:** `<init>` `...` `<let name="families" type="string" />` `<function name="HMI.GET_FONT_FAMILIES" return="families" />` `<control name="lb" xpos="8" ypos="40" width="260" height="140" fieldtype="listbox">` `</control>` `<function name="control.loaditem">_T"lb", families</function>` `...` `<update_controls type="true" />` `</init>`
Abs	This function returns the absolute value of the specified number. **Syntax:** `<function name="abs" return="var"> value` `</function>`
SDEG	The function converts the specified value into degrees. **Syntax:** `<function name="sdeg" return="var"> value` `</function>`
SRAD	The function converts the specified value into RADian. **Syntax:** `<function name="srad" return="var"> value` `</function>`
SQRT	The function calculates the square root of the specified value. **Syntax:** `<function name="sqrt" return="var"> value` `</function>`
ROUND	The function rounds of the transferred number to the specified number of decimal places. If the number of decimal places is not specified, then the function rounds off the number, taking into account the first decimal place. **Syntax:** `<function name="round" return="var"> value, nDecimalPlaces` `</function>`

Function name	Meaning
FLOOR	The function supplies the largest possible integer value, which is less than or equal to the transferred value. **Syntax:** `<function name="floor" return="var"> value </function>`
CEIL	The function supplies the smallest possible integer value, which is greater than or equal to the transferred value. **Syntax:** `<function name="ceil" return="var"> value </function>`
LOG	The function calculates the logarithm of the specified value. **Syntax:** `<function name="log" return="var"> value </function>`
LOG10	The function calculates the common (decadic) logarithm of the specified value. **Syntax:** `<function name="log10" return="var"> value </function>`
POW	The function calculates the value "a^b". **Syntax:** `<function name="pow" return="var"> a, b </function>`
MIN	The function compares the transferred value and returns the lower of the values. **Syntax:** `<function name="min" return="var"> value1, value2 </function>`
MAX	The function compares the transferred value and returns the higher of the values. **Syntax:** `<function name="max" return="var"> value1, value2 </function>`
RANDOM	The function returns a pseudo random number. **Syntax:** `<function name="random" return="var" </function>`

Function name	Meaning
MMC	**Function**: You can use the MMC command to display user-defined dialog forms from the part program in SINUMERIK Operate. The appearance of the dialog forms is defined in a pure text configuration (XML file in the manufacturer directory). The system software remains unchanged. Due to changes in the operate base system, the parameters are as follows: XML → CYCLES or POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF **Syntax:** MMC ("operating area, command, XML script file, menu name, reserved, reserved, display time or acknowledgment variable, reserved", "acknowledgment mode") **Description:** • MMC Calling the dialog form interactively from the part program in SINUMERIK Operate. • CYCLES or POPUPDLG Identification for user dialogs that are to be started from a part program. • PICTURE_ON or PICTURE_OFF Command: Selection or deselection of a dialog • XML file Name of the XML script file • Menu Name of the menu tag that manages the dialog to be displayed • reserved reserved for SINUMERIK Operate • reserved reserved for SINUMERIK Operate • Time Display time of the dialog for acknowledgment mode "N" • reserved reserved for SINUMERIK Operate • Acknowledgment mode "S" synchronous, acknowledgment via the "OK" softkey "N" asynchronous, dialog closes after the set time

Function name	Meaning
MMC continued	**Example of synchronous call:** Due to changes in the operate base system, the parameters are as follows: XML → CYCLES or POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF NC instruction `MMC("POPUPDLG or CYCLES",PICTURE_ON,mmc_cmd.xml,cmd1,,,,,","S")` `File: mmc_cmd.xml` `<menu name = "cmd1">` `<open_form name = "cmd1_form" />` `<softkey_ok>` `<close_form />` `</softkey_ok>` `</menu>` `<form name = "cmd1_form" xpos ="12" ypos="100" width="500" height="240">` `<init>` `</init>` `<paint>` `</paint>` `</form>` **Example of asynchronous call (acknowledgment not expected):** Due to changes in the operate base system, the parameters are as follows: XML → CYCLES or POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF NC instruction `MMC("POPUPDLG or CYCLES",PICTURE_ON,mmc_cmd.xml,cmd1,,,10,,","N")` `File: mmc_cmd.xml <menu name = "cmd1">` `<open_form name = "cmd1_form" />  <softkey_ok>  <close_form />` `</softkey_ok>` `</menu>` `<form name = "cmd1_form" xpos ="12" ypos="100" width="500" height="240">` `<init>` `</init>` `<paint>` `</paint>` `</form>`

Function name	Meaning
MMC continued	**Example of extraction of script parts from a part program:** Due to changes in the operate base system, the parameters are as follows: XML → CYCLES or POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF NC instruction `MMC("POPUPDLG, PICTURE_ON, XMLDIAL_EMB.XML,main","S")` `File: xmldial_emb.xml` `<DialogGui>` ` <let name="menu_name" type="string">main</let>` ` <let name="script_loaded">0</let>` ` <menu name = "main">` ` <if>` ` <condition>script_loaded == 0</condition>` `  <then>` ` <function name ="load_current_program" />` ` <dynamic_include src="__tmp.xml" />` `  <op>` `  script_loaded = 1;` `  </op>` ` <navigation>$$$menu_name</navigation>` ` </then>` ` </if>` ` <softkey_back>` ` <close_form />` ` </softkey_back>` ` </menu>` ` <function_body name ="load_current_program" >` ` <let name="prog_name" type= "string"/>` ` <let name="contents" type= "string"/>` ` <let name="entry" type= "string"/>` ` <let name="len" />` ` <op> prog_name = "nck/Channel/ProgramInfo/workPandProgName"</op>` ` <function name="DOC.LOADSCRIPT"` `return="contents">prog_name, _T"main_dialog", entry` `</function>` ` <function name="string.length" return="len">contents</function>` ` <if>` ` <condition>len > 0</condition>` ` <then>` `  <op>` `  menu_name = entry;` `  </op>` ` <function name="doc.writetofile" >_T"__tmp.xml", contents` `</function>` ` </then>` ` </if>` ` </function_body>` `</DialogGui>`

Function name	Meaning
MMC continued	Programming example ;<main_dialog entry="rpara_main"> ; <let name="xpos" /> ; <let name="ypos" /> ; <let name="field_name" type="string" /> ; <let name="num" /> ; <menu name="rpara_main"> ; <open_form name="rpara_form"/> ; <softkey_back> ; <close_form /> ; </softkey_back> ;</menu> ; ;<form name="rpara_form"> ; <init> ; <caption>test mask</caption> ; <let name="count" >0</let> ; <op> ; xpos = 120; ; ypos = 34; ; ; "nck/Channel/Parameter/R[10]" = 10; ; </op> ; <!-- load the number of controls --> ; <op> ; num = "nck/Channel/Parameter/R[10]"; ; </op> ; ; <while> ; <condition>count < num</condition> ; <print name="field_name" text="edit%d">count</print> ; ; <control name = "$field_name" xpos = "$xpos" ypos = "$ypos" refvar="nck/Channel/Parameter/R[$count]" hotlink="true" /> ; <op> ; ypos = ypos +24; ; count = count +1; ; </op> ; </while> ; ; </init> ; ; <paint> ; <op> ; xpos = 8; ; ypos = 36; ; count = 0; ; </op> ; <while> ; <condition>count < num</condition> ; <print name="field_name" text="R-Parameter%d">count </print> ; ; <text xpos = "$xpos" ypos = "$ypos" >$$$field_name</text> ; <op>

Function name	Meaning
	; ypos = ypos +24; ; count = count +1; ; </op> ; </while> ; ; </paint> ; ;</form> ; ;</main_dialog> … G94 F100 MMC("POPUPDLG, PICTURE_ON, XMLDIAL_EMB.XML,main","A") … … … MMC("POPUPDLG, PICTURE_OFF, XMLDIAL_EMB.XML,main","A") G4 F2 M2
ISNUMERIC	The function checks the content of a string as to whether this can be evaluated as number. Spaces and tabs are ignored. The sign and the decimal point are still considered as valid characters. The function returns a value of 1 if the conditions are satisfied. **Parameter:** **string** - Character string **Syntax:** <function name="isnumeric" return="result"><string> </function>
ROOT	The function extracts the nth root from a number. **Parameter:** **value** - Number **n** - Exponent of a root **Syntax:** <function name="ROOT" return="result"><value>, <n></function>

3.12 Multitouch operation

3.12.1 Multitouch function

Overview

When multitouch displays are introduced, it is necessary to adjust operation to the extended functionality of the display. The rigid positioning of softkeys, for example, is eliminated and replaced or supplemented by free positioning of buttons. Due to the diversity of design options for buttons, it no longer makes sense to use just one control for programming whose appearance is based exclusively on the design specifications of the operating software. Toggle controls, which only know two states, can be replaced by switches, for example, that show the respective state by means of an icon and react to tap or flick gestures.

Displays graphics can be given the capability of reacting to pan gestures. The **imagebox** control has been extended for this purpose.

Note

Graphics that are output in the paint tag do not react to gestures.

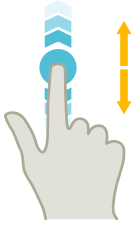
Outside of the controls provided by the parser, finger gestures can be processed in the script, thus providing the option, for example, of offering new navigation strategies in the dialogs. Finger gestures could be used for zooming in to/out of dialog contents.

The Easy XML parser supports the following finger gestures:

Finger gestures

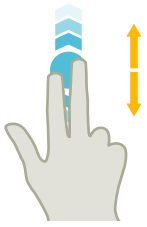
Tap

- Select window
- Select object (e.g. NC set)
- Activate entry field
  - Enter or overwrite value
  - Tap again to change the value



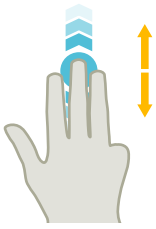
Flick vertically with one finger

- Scroll in lists (e.g. programs, tools, zero points)
- Scroll in files (e.g. NC program)



Flick vertically with two fingers

- Page-scroll in lists (e.g. ZO)
- Page-scroll in files (e.g. NC programs)



Flick vertically with three fingers

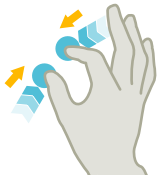
- Scroll to the start or end of lists
- Scroll to the start or end of files

Flick horizontally with one finger

- Scroll in lists with many columns

Spread

- Zoom in to graphic contents (e.g. simulation, mold making view)



Pinch

- Zoom out from graphic contents (e.g. simulation, mold making view)

Pan with one finger

- Move graphic contents (e.g. simulation, mold making view)
- Move list contents

The **gesture_event** tag and the **$gestureinfo** variable are used for handling finger gestures. The enable program actions for decoded finger gestures.

3.12.2 Programming finger gestures

gesture_event tag

Note

This tag is executed only if the operator panel supports multitouch operation.

If the operating software recognizes a finger gesture, the parser provides the gesture information in the **$gestureinfo** variable and executes the **gesture_event** tag. The variable possesses the **GestureInfoStruct** data type and contains the following attributes:

Attribute	Value	Meaning
type	3	Pan gesture
	4	Zoom out gesture
	5	Tap gesture
flag	1	Scaling factor changed
	2	Angle of rotation changed
state	1	Started
	2	Updated
	3	Completed
item_data		Identification of the active control
	-1	The gesture is not assigned to a control
	!= -1	The gesture was executed in a control to which this value was assigned during creation
point		Position of the gesture
	point.x	X position of the gesture
	point.y	Y position of the gesture
delta		Difference between the start of the gesture and the current event
	<Scaling difference>	When scaling factor is changed
	<Angular difference>	When angle of rotation is changed

These attributes are predefined in the **struct_def.xml** file.

Further information is provided in Chapter "Creating the file struct_def.xml (Page 138)"

3.12.3 Gesture control for graphics

Imagebox control variable

The following three extensions are available for the gesture control of graphics when using the **Imagebox** control variable:

Attribute	Meaning/behavior
rotationangle	This attribute value indicates the angle at which the graphic is to be displayed.
setrotationmode	The **true** attribute value allows processing of the pinch gesture
setzoommode	The **true** attribute value allows you to zoom in to/out from or move a graphic in the display area. This default setting of this attribute is **false**.

Syntax

```
<property rotationangle="angle" />
<property setrotationmode="true/false" />
<property setzoommode="true/false" />
```

Example of bitmap rotation

Rotate bitmap by 30.5 degrees clockwise:

```
<let name="image_box_pict_name" type="string" >pic1.bmp</let>
…
…
…
 <control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property rotationangle="30.5" />
 <property setrotationmode ="true" />
 </control>
```

Example of bitmap zooming

Allowing zooming for a bitmap:

```
<control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property setzoommode="true" />
 </control>
```

imageboxset control function

The properties can also be set using the **control.imageboxset** function.

The following commands are available:

Command	Meaning/behavior
SetRotationAngle	The graphic is rotated around the angle indicated in **lparam1**.
SetRotationMode	The value of **lparam1** defines the evaluation of the pinch gesture with respect to the rotation. Value equal to 1 - the gesture is processed by the control
SetZoomMode	If the value of **lparam1** is equal to 1, the pinch gesture for zooming in to/out from or moving the graphic is evaluated.

3.12.4 Gesture processing

Tag message

If the scaling factor is greater than 1.0, the image in the display area will be moved. If the factor is 1.0, this finger gesture can be used for browsing through a list of images, for example. In this case, the parser sends a message to the form, which can be evaluated in the **message** tag.

The **$message_par1** message parameter contains the Item_Data value of the control. The **$message_par2** message parameter signals the gesture's direction of movement.

The **$message_par2** can accept the following values:

- -1 roll left
- 1 roll right
- -2 roll up
- 2 roll down

Example

```
…
…
<let name="image_box_pict_name" type="string" ></let>
<let name="pictindex" />
<let name="image_box_list" type="string" dim="4">
 "pic1.bmp",
 "pic2.bmp",
 "pic3.bmp",
 "pic4.bmp",
</let>

…
…
<control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property setzoommode="false" />
</control>
…
…
…
<!--
 -1 roll left
1 roll right
-2 roll up
2 roll down
 -->
<message>
 <if condition="$message_par1 == 1000">
  <then>
```

```
    <switch>
     <condition>$message_par2</condition>
     <case value="1">
      <op>
       pictindex = pictindex+1;
      </op>
      <if condition="pictindex > 3">
       <then>
        <op>
         pictindex = 0;
        </op>
       </then>
      </if>
     </case>
     <case value="-1">
      <op>
       pictindex = pictindex-1;
      </op>
      <if condition="pictindex < 0">
       <then>
        <op>
         pictindex = 3;
        </op>
       </then>
      </if>
     </case>
    </switch>
     <op>
      image_box_pict_name = image_box_list[$pictindex];
     </op>
    </then>
   </if>
  </message>
…
…
```

3.13 Configuring your own buttons

Buttons can be integrated into a form as action buttons or selection buttons.

3.13.1 Pushbutton

Tag property

The pushbutton is a control element that can be used as a button or a switch (button with latching function). The button can be designed by means of attribute definitions in the "softkey look & feel" style as well as in a customized style. The relevant attributes must be defined with the **property** tag.

The foreground or background color can be changed using the **color_fg**, **color_bk**, **color_fg_pressed** and **color_bk_pressed** attributes.

The **pressed/locked** or **released** states are represented by the values one or zero. A handler function must be indicated to evaluate a button's change of state. The handler function is indicated with the **function** attribute in the control tag.

The state of a switch can be determined by reading out the control variable or an assigned reference variable.

With a touch operation, the finger gestures are "pressed" and "released" only when the finger gesture is "let go".

Syntax

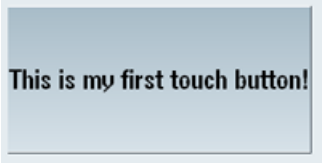
```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption></caption>
</control>
```

Or with handler function

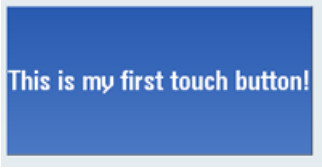
```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" function="button_handler" hotlink="true" >
 <caption></caption>
</control>
…
…
…
 <function_body name="button_handler">
…
…
…
 </function_body>
```

or

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" refvar="button_state" hotlink="true" >
 <caption></caption>

 <property checkable="true" />
</control>
```



Deactivated



Activated, latched

Example

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" refvar="button_state" hotlink="true"
color_fg="#ff00ff" color_bk="#000eee">
 <property checkable="true" />
  <caption></caption>
 <property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
</control>
```

3.13.2 Functions of the pushbutton

3.13.2.1 Sub-tags for the pushbutton

Tag caption

The programmer uses the **caption** tag to specify the button text to be displayed for the "not pressed" state. In the default setting, the text is centered. The text alignment can be changed with the **alignment** attribute. The following attribute values can be specified:

Attribute value	Alignment
left	Left-aligned
right	Right-aligned
top	Top
bottom	Bottom
center	Centered

If a different text is to be displayed for the "pressed" state, a second caption instruction must be programmed with the **pressed** attribute and the **true** value.

Syntax

Centered display

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption>Button text</caption>
</control>
```

Left-aligned display

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption alignment= "left">Button text</caption>
</control>
```

Pressed display

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption pressed="true">Button text pressed</caption>

</control>
```

3.13.2.2 Properties for the pushbutton

Additional properties are assigned to the control with the **property** tag.

Determining button properties

The **checkable** attribute allows the button to be used as a switch. To do so, the value of the attribute must be set to **true**.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <property checkable="true/false" />
</control>
```

Disabling a switch

The **disabled** attribute controls whether or not the button can be operated. If the value is **true**, control actions are not processed.

Changes in state caused by an assigned reference variable result in updating of the button.

Syntax

```
<property disabled="true/false" />
```

Example

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption alignment="left">Button text</caption>
 <property disabled="true " />
</control>
```

Assigning icons

The predefined design can be covered over by specifying your own icons or button colors. An icon can be assigned to the button for each of the states "not pressed", "pressed" and "disabled". If there is no assignment for the states "pressed" or "disabled", the control displays the icon for the "not pressed" state.

Other attributes control the alignment, scaling and transparency of each icon.

Attribute	Meaning/behavior
Picture	Icon in foreground Name of the icon for the "not pressed" state
PicturePressed	Icon in foreground Name of the icon for the "pressed" state
PictureDisabled	Icon in foreground Name of the icon for the "disabled" state

Attribute	Meaning/behavior
BackgroundPicture	Icon in background Name of the icon for the "not pressed" state
BackgroundPicturePressed	Icon in background Name of the icon for the "pressed" state
BackgroundPictureDisabled	Icon in background Name of the icon for the "disabled" state

Alignment attribute

Further attributes can be specified in the tag whose values refer to the **alignment** of the listed icon assignments:

Attribute value	Alignment
left	Left-aligned
right	Right-aligned
top	Top
bottom	Bottom
center	Centered
stretch	Background icon: If the **stretch** value is used, the parser scales the icon to the rectangular area of the button. Any desired outline can be generated for the button by declaring the transparent color with the **transparent** attribute.

3.13.2.3 Control variables for the pushbutton

Button properties can be subsequently changed by assigning new values to the attribute variables listed below. The values are assigned in an operation instruction by specifying the control name followed by the control variable name. The two names must separated by a period.

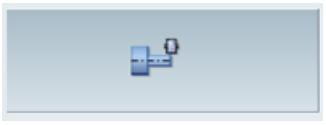
Syntax

<Control-Name>.<Control-Variable>

Control variable	Meaning/behavior
backgroundpicture	Variable type: String Name of background icon
backgroundpicturedisabled	Variable type: String Name of background icon for the disabled state
backgroundpicturepressed	Variable type: String Name of background icon for the pressed state
picture	Variable type: String Name of foreground icon
picturedisabled	Variable type: String Name of foreground icon for the disabled state
picturepressed	Variable type: String Name of foreground icon for the pressed state
picture_textalignedtopicture	Variable type: Bool Value: 1 = true 0 = false Button text is aligned on the icon
picturedisabled_textalignedtopicture	Variable type: Bool Value: 1 = true 0 = false Button text is aligned on the "disabled state" icon
picturepressed_textalignedtopicture	Variable type: Bool Value: 1 = true 0 = false Button text is aligned on the "pressed state" icon
backgroundpicture_keepaspectratio	Variable type: Bool Value: 1 = true 0 = false The aspect ratio of the background icon is retained or ignored

Control variable	Meaning/behavior
backgroundpicturedisabled_keepaspectratio	Variable type: Bool Value: 1 = true 0 = false The aspect ratio of the "disabled state" background icon is retained or ignored
backgroundpicturepressed_keepaspectratio	Variable type: Bool Value: 1 = true 0 = false The aspect ratio of the "pressed state" background icon is retained or ignored
picture_keepaspectratio	Variable type: Bool Value: 1 = true 0 = false The aspect ratio of the icon is retained or ignored
picturedisabled_keepaspectratio	Variable type: Bool Value: 1 = true 0 = false The aspect ratio of the "disabled state" icon is retained or ignored
picturepressed_keepaspectratio	Variable type: Bool Value: 1 = true 0 = false The aspect ratio of the "pressed state" icon is retained or ignored
backgroundpicture_scaled	Variable type: Bool Value: 1 = true 0 = false The background icon is adjusted to the dimensions of the button
backgroundpicturedisabled_scaled	Variable type: Bool Value: 1 = true 0 = false The "disabled state" background icon is adjusted to the dimensions of the button

Control variable	Meaning/behavior
backgroundpicturepressed_scaled	Variable type: Bool Value: 1 = true 0 = false The "pressed state" background icon is adjusted to the dimensions of the button
picture_scaled	Variable type: Bool Value: 1 = true 0 = false The icon is adjusted to the dimensions of the button
picturedisabled_scaled	Variable type: Bool Value: 1 = true 0 = false The "disabled state" icon is adjusted to the dimensions of the button
picturepressed_scaled	Variable type: Bool The "pressed state" icon is adjusted to the dimensions of the button
backpicture_alignment	Variable type: String See attribute alignment
backgroundpicturedisabled_alignment	
backgroundpicturepressed_alignment	
picture_alignment	
picturedisabled_alignment	
picturepressed_alignment	
caption	Variable type: String "Not pressed state" button text
captionpressed	Variable type: String "Pressed state" button text
caption_alignment	Variable type: String See attribute alignment
captionpressed_alignment	
captiondisabled_alignment	
disabled	Variable type: Bool Value: 1 = true - pushbutton disabled 0 = false - pushbutton can be operated



Picture

PicturePressed

PictureDisabled

BackgroundPicture + Picture

TextAlignedToPicture attribute

If the value of the attribute is **true**, the text is aligned relative to the icons.

```
<property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" />
```

Scaled attribute

If the value of the attribute is **true**, the dimensions of the image are adjusted to the dimensions of the button in such a way that 50% of the height or width of the button at most is made available for the graphic.

```
<property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
```

Stretch attribute (only effective for background icons)

If the value of the attribute is **true**, the dimensions of the image are adjusted to the dimensions of the button.

```
<property backgroundpicture="f:\appl\pbutton.png"
alignment="stretch" />
```

```
<property backgroundpicture="f:\appl\pbutton.png"
alignment="stretch" transparent="#ffffff"/>
```

```
<caption alignment="left" >This is my first touch button!</caption>
 <property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
 <property backgroundpicture="pbutton.png" alignment="stretch"
transparent="#ffffff"/>
```

3.13.3 Switch on/off

A switch control is a graphic element that signals one of two states by means of an icon. This control can be operated by touch or by using a mouse.

The state adopted can be stored in a reference variable or the change of state can be determined in a function assigned to the control. The function is assigned with the **function** attribute in the control tag.

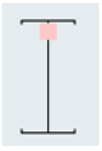
The **alignment** attribute enables the vertical or horizontal alignment of the button.

This control has no standard design. If no icons are assigned to the control, only the bounding box and switch position (shown by a dot) are displayed.

Syntax

```
<control name="name" xpos = "x position" ypos="y position"
width="width" height="height" fieldtype="switch" alignment="hr/vr">
 <property left_position="value" />
 <property right_position="value" />
</control>
```

Alignment attribute



Vertical switch: Value **vr**

Horizontal switch: Value **hr**

3.13.4 Functions of the switch

3.13.4.1 Properties for the switch

Tag property

The following switch positions can be assigned to the control with the **property** tag:

Attribute	Meaning/behavior
left_position	The value of the attribute is assigned to the control variable if the switch was moved to the left.
right_position	The value of the attribute is assigned to the control variable if the switch was moved to the right.
upper_position	The value of the attribute is assigned to the control variable if the switch was moved up.
lower_position	The value of the attribute is assigned to the control variable if the switch was moved down.
disabled	The attribute controls whether or not the switch can be operated. If the value is **true**, a control action is not evaluated. Changes in state caused by an assigned reference variable result in updating of the switching state.

The final switch design must be determined by specifying further attributes. These attributes must be defined together with the attribute **left_position**, **right_position**, **upper_position** or **lower_position**.

Any desired outline can be generated for the switch by declaring one color as a transparent color with the **transparent** attribute.

Attribute	Meaning/behavior
picture	Icon in foreground Name of the icon for the "not pressed" state
pictureDisabled	Icon in foreground Name of the icon for the "disabled" state
transparent	The attribute value includes the color value of the transparent color. This color value is used for all icons assigned to the switch.
caption	The attribute value includes the text associated with the switch position. This text is displayed on the element and limited by the dimensions of the switch.

3.13.4.2 Control variables for the switch

Switch properties can be subsequently changed by assigning new values to the control variables listed below. The values are assigned in an operation instruction by specifying the control name followed by the control variable name. The two names must separated by a period.

Syntax

<Control-Name>.<Control-Variable>

Control variable	Meaning/behavior
Left_position	Variable type: Int The value of the attribute is assigned to the control variable if the switch was moved to the left.
Right_position	Variable type: Int The value of the attribute is assigned to the control variable if the switch was moved to the right.
Lower_position	Variable type: Int The value of the attribute is assigned to the control variable if the switch was moved down.
Upper_position	Variable type: Int The value of the attribute is assigned to the control variable if the switch was moved up.
Picture_left_position	Variable type: String Name of icon for the left-hand position
Picture_right_position	Variable type: String Name of icon for the right-hand position
Picture_upper_position	Variable type: String Name of icon for the upper position
Picture_lower_position	Variable type: String Name of icon for the lower position
Picturedisabled_left_position	Variable type: String Name of icon for the left-hand position "disabled state"
Picturedisabled_right_position	Variable type: String name of icon for the right-hand position "disabled state"
Picturedisabled_upper_position	Variable type: String Name of icon for the upper position "disabled state"
Picturedisabled_lower_position	Variable type: String Name of icon for the lower position "disabled state"
Caption_left_position	Variable type: String Text in the left-hand position
Caption_right_position	Variable type: String Text in the right-hand position
Caption_upper_position	Variable type: String Text in the upper position

Control variable	Meaning/behavior
Caption_lower_position	Variable type: String Text in the lower position
disabled	Variable type: Bool Value: 1 = true - switch disabled 0 = false - switch can be operated

Horizontal display

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="switch" alignment="hr">
 <property left_position="value" picture="name" />
 <property right_position="value" picture="name" />
</control>
```

Vertical display

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="switch" alignment ="vr">
 <property upper_position="value" picture="name" />
 <property lower_position="value" picture="name" />
</control>
```

Or assign handler function

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="switch"
function="switch_handler" hotlink="true" alignment ="vr">
</control>
```

Example

icon_left.bmp

icon_right.bmp

```
<let name="switch_state" />
<control name="main_switch" xpos="100" ypos="53" width="40"
height="30" fieldtype="switch" refvar="switch_state" hotlink="true"
alignment="hr">
 <property left_position="10" picture="icon_left.bmp" />
 <property right_position="11" picture="icon_right.bmp" />
</control>
```

Assigning properties by means of control variables

```
<control name="main_switch" xpos = "450" ypos="340" width="20"
height="46" fieldtype="switch" refvar="switch_statebf"
hotlink="true" alignment="vr">
 <property upper_position="1" />
 <property lower_position="0" />
</control>
…
…
…
<op>
  main_switch.transparent = #ffffff;
  main_switch.picture_upper_position = _T"switch_up.png";
  main_switch.picture_lower_position = _T"switch_bottom.png";
  main_switch.disable= 1;
</op>
```

3.13.5 Radio button

Radio buttons enable the selection of one of a number of options. A button must be created for each option. All radio buttons belonging to one form, groupbox or scroll area interact with one another in such a way that only one button is marked as selected. The button states are transferred to the control variables.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="radiobutton"  >
 <caption>button text</caption>
</control>
```

3.13.6 Checkbox

Checkboxes enable the selection of a number of options. A button must be created for each option. The state of the checkbox is transferred to the control variable.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="checkbox"  >
 <caption>button text</caption>
</control>
```

3.13.7 Groupbox

A groupbox optically encloses a group of controls with a frame and and a title. It groups logically related controls that are only intended to interact within the group. The coordinates of the embedded controls refer to the upper left-hand corner below the title line.

All of the controls belonging to the group are embedded into the control tag as sub-controls.

When assigning the embedded control names and the **Item_Data** value, it is important to note that they must be unique with respect to all controls belonging to the form.

The title of the groupbox is specified with the **caption** tag.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="groupbox">
 <caption>caption text</caption>
  <control>…</control>
…
…
…
</control>
```

3.13.8 Scroll area

A scroll area is used to display controls within a specified area. The coordinates of the embedded controls refer to the upper left-hand corner of the scroll area. If controls are created outside of the visible area, moving the visible area is enabled. All of the controls belonging to the area must be embedded into the control tag as sub-controls.

When assigning the embedded control names and the **Item_Data** value, it is important to note that they must be unique with respect to all controls belonging to the form.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="scrollarea">
  <control>…</control>
…
…
…
</control>
```

Touch operation

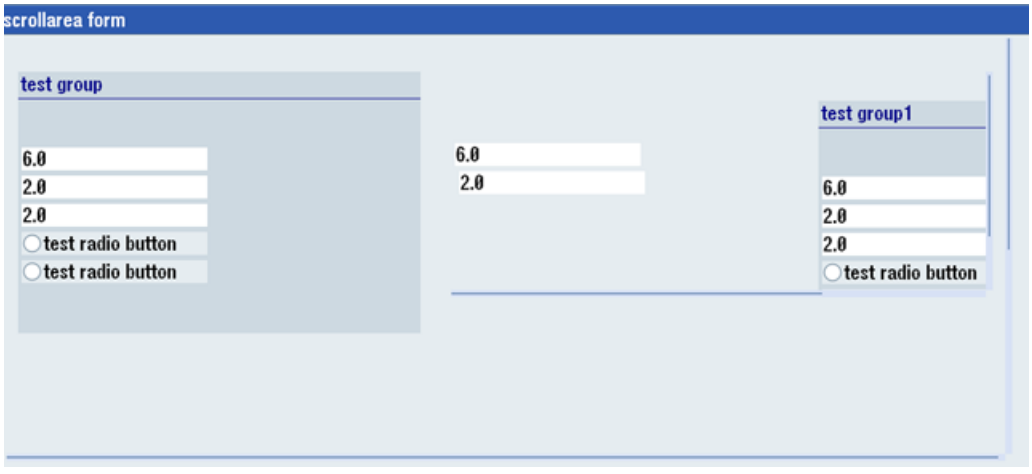
If focus is placed on a control that is currently not completely visible, the visible area is moved until the control can be displayed in full.

Tab gesture

This gesture is forwarded to the script if the gesture cannot be assigned to an embedded control.

Example

Nested scroll areas




```
<let name="CB1" >1</let>
<let name="RB1" />
```

```
<form name="scrollarea_form">
 <init>
  <caption>scrollarea form</caption>
   <data_access type="true" />
    <control name= "c_scroll" xpos = "2" ypos="24" width="520"
height="300" fieldtype="scrollarea" >
    <control name= "c_group" xpos = "6" ypos="24" width="208"
height="186" fieldtype="groupbox" >
    <caption>test group</caption>
    <control name = "c18" xpos = "2" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
    <control name = "c19" xpos = "2" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
    <control name = "c20" xpos = "2" ypos = "94" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
    <control name="radiobg" xpos = "2" ypos="114"
fieldtype="radiobutton" >
    <caption>test radio button</caption>
    <property backgroundpicture="f:\appl\cycle_my1.png" />
   </control>
   <control name="radiobg1" xpos = "2" ypos="134"
fieldtype="radiobutton" >
    <caption>test radio button</caption>
    <property backgroundpicture="f:\appl\cycle_my1.png" />
   </control>
  </control>
 <control name= "c_scroll_emb" xpos = "230" ypos="24" width="280"
height="160" fieldtype="scrollarea" >

 <control name = "c18emb" xpos = "2" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
 <control name = "c19emb" xpos = "4" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
 <control name = "c20emb" xpos = "3" ypos = "194" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
 <control name= "c_group1" xpos = "190" ypos="24" width="208"
height="186" fieldtype="groupbox" >
 <caption>test group1</caption>
 <control name = "c18gemb" xpos = "2" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
 <control name = "c19gemb" xpos = "2" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
 <control name = "c20gemb" xpos = "2" ypos = "94" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
 <control name="radiobggemb" xpos = "2" ypos="114"
fieldtype="radiobutton" >
  <caption>test radio button</caption>
  <property backgroundpicture="f:\appl\cycle_my1.png" />
</control>
<control name="radiobg1emb" xpos = "2" ypos="134"
fieldtype="radiobutton" >
```

```
 <caption>test radio button</caption>
  <property backgroundpicture="f:\appl\cycle_my1.png" />
 </control>
 </control>
 </control>
<control name = "c22" xpos = "2" ypos = "400" refvar="nck/Channel/
Parameter/R[12]" hotlink="true" />
 <control name="ChB1" xpos="208" ypos="430" width="50" height="30"
fieldtype="checkbox" refvar="PLC/M100.7" hotlink = "true"
color_bk="#00ffff">
  <caption>Check1</caption>
 </control>
  <control name="ChB2" xpos="208" ypos="460" width="58" height="30"
fieldtype="checkbox" refvar="PLC/M100.6" hotlink = "true"
color_bk="#00ffff">
  <caption>Check2</caption>
 </control>
  <control name="Radio1" xpos="208" ypos="490" width="40"
height="30" fieldtype="radiobutton" refvar="PLC/M100.0" hotlink =
"true" color_bk="#00ffff">
   <caption>Radio1</caption>
  </control>
   <control name="Radio2" xpos="208" ypos="520" width="45"
height="30" fieldtype="radiobutton" refvar="PLC/M100.1" hotlink =
"true" color_bk="#00ffff">
    <caption>Radio2</caption>
  </control>
   <control name="Radio3" xpos="208" ypos="550" width="50"
height="30" fieldtype="radiobutton" refvar="PLC/M100.2" hotlink =
"true" >
    <caption>Radio3</caption>
   </control>
    <control name="VChB1" xpos="8" ypos="430" width="50" height="30"
refvar="PLC/MB100" hotlink = "true" color_bk="#00ffff"/>
    <control name="VChB2" xpos="8" ypos="465" width="53" height="30"
refvar="PLC/MB100" hotlink = "true" color_bk="#00ffff"/>
   </control>
   <control name="ChB3" xpos="208" ypos="340" width="60" height="30"
fieldtype="checkbox" refvar="CB1" >
    <caption>Check3</caption>
   </control>
   </init>
  <timer>
  </timer>
</form>
```

3.14 Sidescreen application

3.14.1 Easy XML in the Sidescreen

The Easy XML scripting language can also be used to create dialogs in the Sidescreen area. The Sidescreen components **Page** and **Widget** are available for displaying the dialogs. The connection is made via the **slsidescreen.ini** file. Sidescreen components are automatically started when the operating software starts up and are only terminated when the system shuts down. This means that the parser loads the assigned scripts once when the Sidescreen component is activated for the first time.

The dimension of the Sidescreen area is determined by the layout parameters from the **slsidescreen_<screen resolution>.ini** file. To enable independent configuration, the usable width for Easy XML scripts is 276 pixels. The usable height of an Easy XML dialog depends on the Sidescreen component used. For a page, 768 pixels are available. For a widget, the height is determined by the Sidescreen management and can be queried with the **GETHMIResolution** function.

The script parser expects a menu to activate a shape or branch to other shapes. The main menu must be named **main**. Sidescreen does not support softkey tags, so that you need to navigate to other shapes using an operator control for the shape.

The number of Sidescreen pages or Sidescreen widgets is not limited by the parser.

3.14.2 Integrating Sidescreen dialogs

The pages or widgets are integrated using the **slsidescreen.ini** file.

Pages are entered in the **[Sidescreen]** section. Each page must be specified with the keyword **PAGE** followed by the sequential page number and the required attributes. The **name** attribute defines the object name of the page. The **implementation** attribute specifies the implementation library and the class name. Both attributes must be separated by a comma. **SlSideScreenPage** type pages can include Sidescreen elements. This is done via the "**ELEMENT**" entries. The rule for creating an **ELEMENTxxx** element line corresponds to the rule for creating a page. If a page should only contain dialogs that were configured with Easy XML, **slagmforms.SlEESideScreenPage** should be specified as value for the **implementation** attribute.

Widgets can be added to a Sidescreen element in the section **[Element_<name>]**.

The rule for creating a **WIDGETxxx** widget line corresponds to the rule for creating a page.

If an Easy XML Sidescreen widget is to be activated, **slagmforms.SlEESideScreenWidget** should be specified as the value for the **implementation** attribute.

Syntax

```
ELEMENT002= name:=<name>, implementation:=SlSideScreenElement

...
...

[Element_<name>]
WIDGET001= name:=<Widget Name>, implementation:=
slagmforms.SlEESideScreenWidget
```

The widget identifier is used by default to form the main module name.

Example

```
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage
```

Main module name: sidescreen_proginfluence.xml

Additional properties

Additional properties can be assigned to the pages or widgets by entering a section named **PAGE_<pagename>**, **ELEMENT_<elmentname>** or **WIDGET_<widgetname>** in the file **slsidescreen.ini**.

The following properties can be assigned to a page, an element or a widget:

Attribute	Meaning/behavior
TextId	Language-independent identifiers of the text
TextFile	Text file name
TextContext	Text context EASY_XML
Icon	Name of the icon

Property	Meaning/behavior
maskPath	The property specifies the main module name to use.
maskName	The property specifies the main menu name to use.
focusable	The property determines whether the input focus can be set to the element.

Example

```
[Page_sidescreen_proginfluence]
PROPERTY001= name:=focusable, type:=bool, value:="true"
Icon=sidescreen_proginfluence.png

[PAGE_sidescreen_proginfluence]
Icon= sidescreen_proginfluence.png
PROPERTY001= name:=focusable, type:=bool, value:="true"
PROPERTY002= name:=maskPath, type:=QString, value:="
sidescreen_proginfluence.xml"
PROPERTY003= name:=maskName, type:=QString, value:="main"
```

3.14.3 Language and text management

To manage the language-independent texts, each component can be assigned a text file. The text file name is formed from the name of the component, the language version and the extension ".ts".

EASY_XML should be used as the text context.

Syntax

<name>_<language>.ts

Example

```
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage
```

Text file name: sidescreen_proginfluence_**eng.ts**

If no text file is specified, the parser searches for the text identifier in the **oem_xml_screens_xxx.ts** default text file.

3.14.4 Sidescreen components

3.14.4.1 Sidescreen element

If a page is linked to the default implementation, elements can be assigned to this page.
A **[Page_<page_name>]** section is to be created for this and all elements associated with the page are listed.

Example

```
[Sidescreen]
PAGE001= name:=<page_name>, implementation:=SlSideScreenPage

[Page_<page_name>]

ELEMENT<number>= name:=<Element name>,
implementation:=SlSideScreenElement
```

Each element requires an **[Element_<element name>]** section in which the properties and associated widgets are listed.

```
[Element_<element_name>]
WIDGET001= name:=<widget_name>, implementation:= <implementation>
```

3.14.4.2 Sidescreen widget

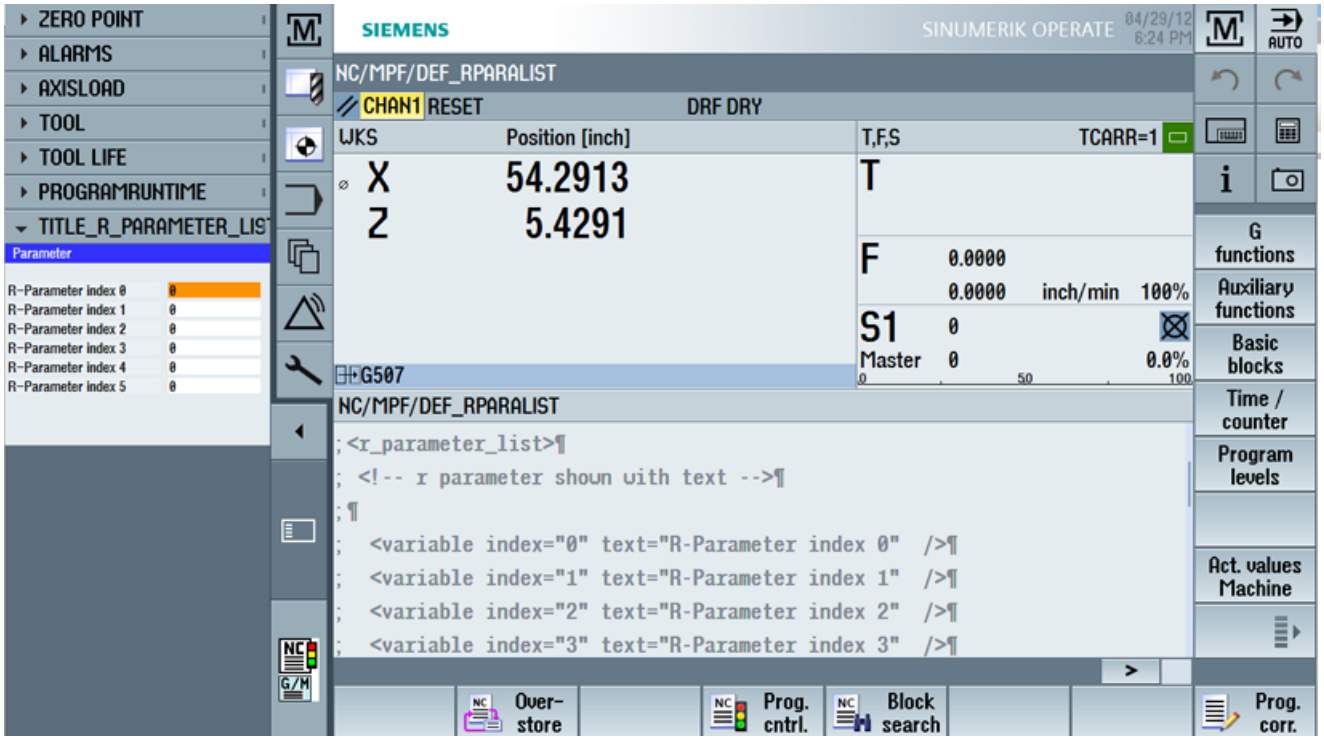
A widget is assigned an area by the Sidescreen management in which the configured shapes can be displayed. By default, a widget does not obtain an input focus so that fields cannot be edited. This behavior can be changed by specifying the **focusable** attribute. When the widget opens, the parser opens the form associated with the main menu. Within the form, the user can branch out to other menus with a navigation instruction.

Example

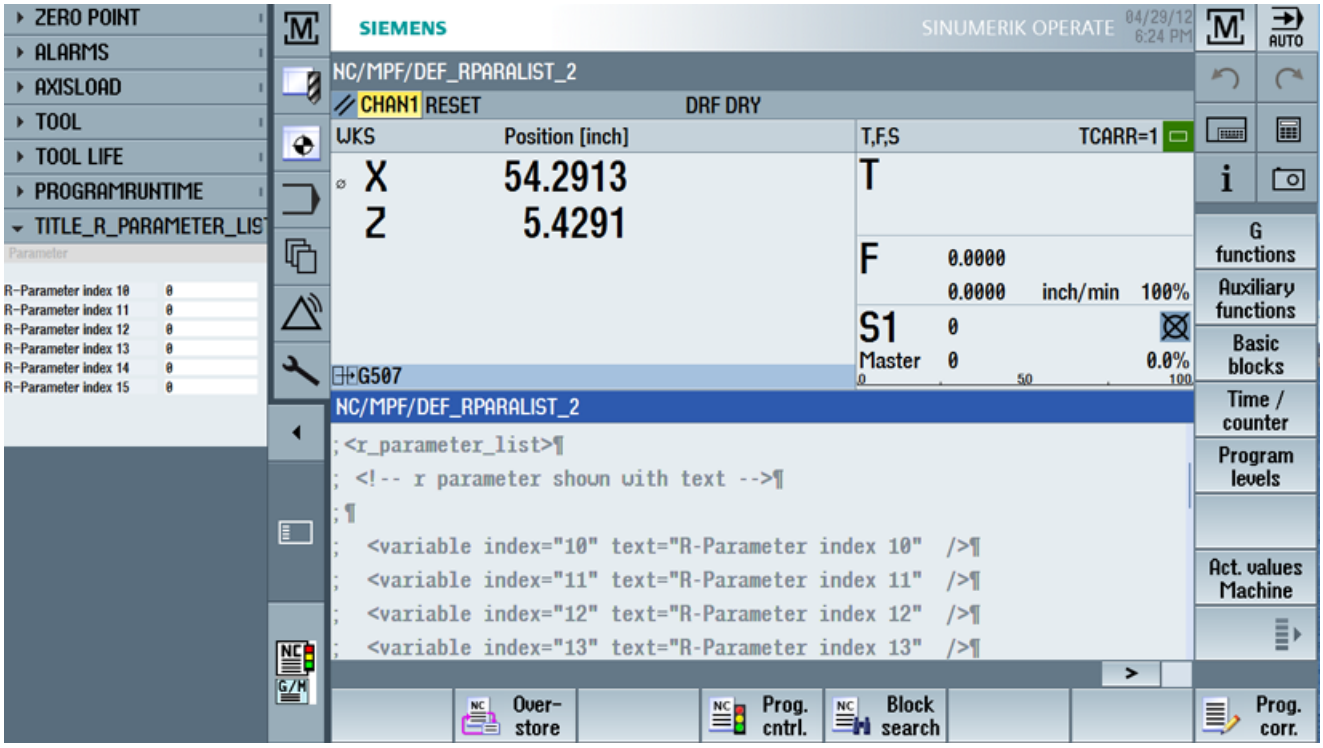
The Easy XML script searches for the description for the data content of the widget in the active part program. In this specific example, the description of a list with R parameters is expected that are to be displayed. Each parameter can be assigned a description text.

Toolbox example

Custom Screen Sample\
side_screen_examples\sidescreenwidget\displayRparameter\rparameter_widget.xml



If another part program is selected, the widget is automatically reworked.


ZERO POINT
ALARMS
AXISLOAD
TOOL
TOOL LIFE
PROGRAMRUNTIME
TITLE_R_PARAMETER_LIST
Parameter
R-Parameter index 10 0
R-Parameter index 11 0
R-Parameter index 12 0
R-Parameter index 13 0
R-Parameter index 14 0
R-Parameter index 15 0
SIEMENS
SINUMERIK OPERATE
04/29/12
6:24 PM
NC/MPF/DEF_RPARALIST_2
CHAN1 RESET
DRF DRY
WKS
Position [inch]
X 54.2913
Z 5.4291
T,F,S
TCARR=1
T
F 0.0000
0.0000 inch/min 100%
S1 0
Master 0 0.0%
G507
NC/MPF/DEF_RPARALIST_2
;<r_parameter_list>¶
; <!-- r parameter shoun uith text -->¶
;¶
; <variable index="10" text="R-Parameter index 10" />¶
; <variable index="11" text="R-Parameter index 11" />¶
; <variable index="12" text="R-Parameter index 12" />¶
; <variable index="13" text="R-Parameter index 13" />¶
AUTO
G functions
Auxiliary functions
Basic blocks
Time / counter
Program levels
Act. values Machine
Over-store
Prog. cntrl.
Block search
Prog. corr.

3.14.4.3 Sidescreen page

A page, which is started via the **slagmforms.SIEESideScreenPage** implementation, makes available the entire Sidescreen area of a form. It has no title line so that a heading cannot be configured with the Caption statement. By default, a page does not obtain an input focus so that fields cannot be edited. This behavior can be changed by specifying the **focusable** attribute. When the page opens, the parser opens the form associated with the main menu. Within the form, the user can branch out to other menus with a navigation instruction.

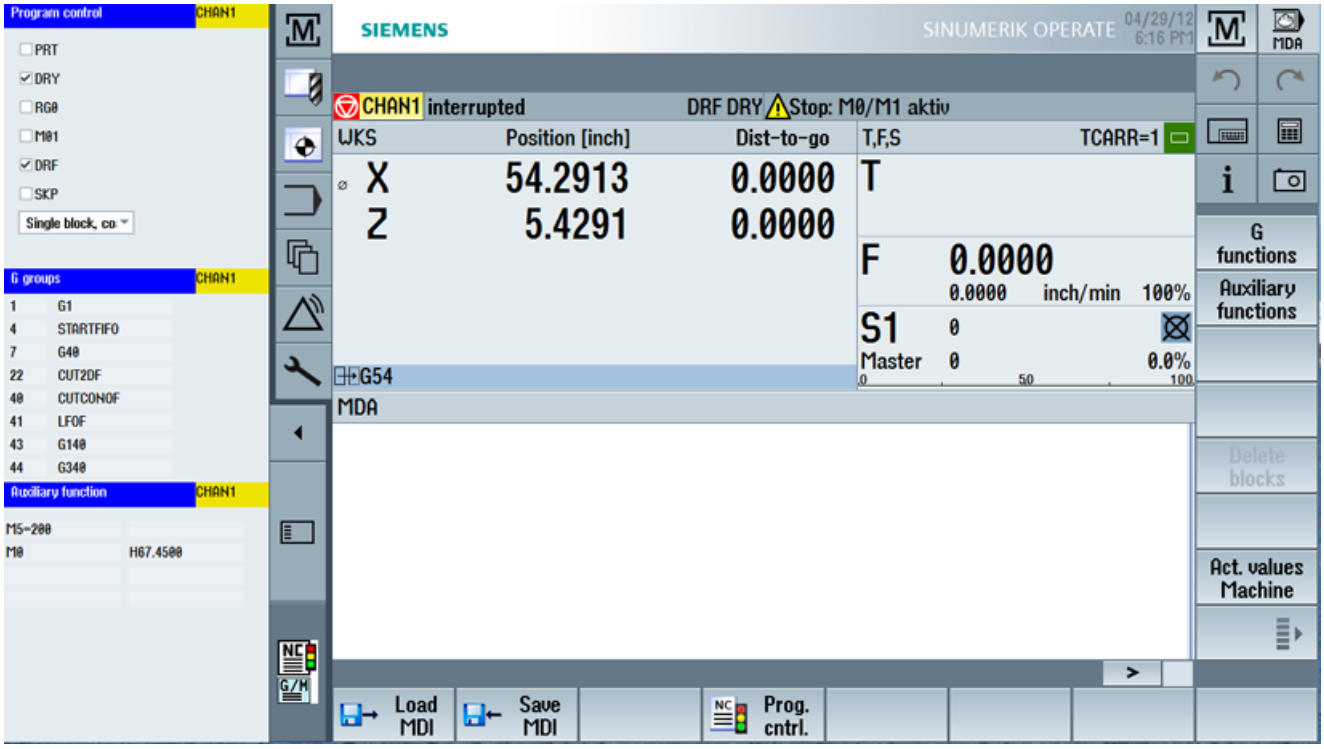
Example

The Sidescreen page is displayed as page which only shows an Easy XML form.

In this example, additional information is displayed in the Sidescreen area.

Toolbox example

Custom Screen Sample\side_screen_examples\sidecreenpage\sidescreen_proginfluence.xml




```
[Sidescreen]
PROPERTY001= name:=minimizable, type:=bool, value:="true"
PROPERTY002= name:=buttonBarVisible, type:=bool, value:="true"
PAGE001= name:=WidgetsPage, implementation:=SlSideScreenPage
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage
```

```
[Page_sidescreen_proginfluence]
PROPERTY001= name:=focusable, type:=bool, value:="true"
Icon=sidescreen_proginfluence.png
```

Text file: sidescreen_proginfluence_deu.ts

3.15 Display Manager application

3.15.1 Easy XML in the Display Manager

The Easy XML function can be included in the Display Manager configuration based on dialog **SlSideScreenPage**. Depending on the space available, this dialog can display one or more side screen pages. Several pages are displayed next to each other in the column format. The available space (width) is evenly divided among the individual columns. The number of pages to be displayed and their content are configured. Each page has its own configuration file. The names of these configuration files are specified when configuring the dialog in the **systemconfiguration.ini** file, in command line parameter **cmdline**.

Behind the parameter identifier **–sidescreen1**, is the name of the configuration file for the first column; behind the parameter identifier **–sidescreen2**, is the name of the configuration file for the second column, etc.

All applications are automatically started by the Display Manager at system start and closed again when the system is shut down. This means that script changes only become active after the HMI has been restarted.

3.15.2 Integrating the Customer Area in the Display Manager menu

The precondition for calling the Easy XML parser in the Customer Area is that an area has been linked with the Easy XML dialog. This is the default scenario with the area definition **AREA111** in file **systemconfiguration.ini**.

Syntax

```
AREA0111= name:=CustomXML, dialog:=SlEECustomDialog,
panel:=SlHdStdHeaderPanel
```

This area is invisible when the system is delivered. Activation is realized using file **slamconfig.ini**, in which visibility flag **Visible** should be set from **false** to **true**.

```
[CustomXML]
TextId=SL_AM_CUSTOM
SidescreenTextId=SL_SIDESCREEN_CUSTOM
TextFile=slam
TextContext=SlAmAreaMenu
SoftkeyPosition=12
Visible=false -> true
```

Examples

The Customer Area is integrated in a menu of the Display Manager using the resolution-dependent INI files of the Display Manager.

In the relevant file, under switch **Operate buttons** , a menu item should be created that contains the reference to area **CustomXML**:

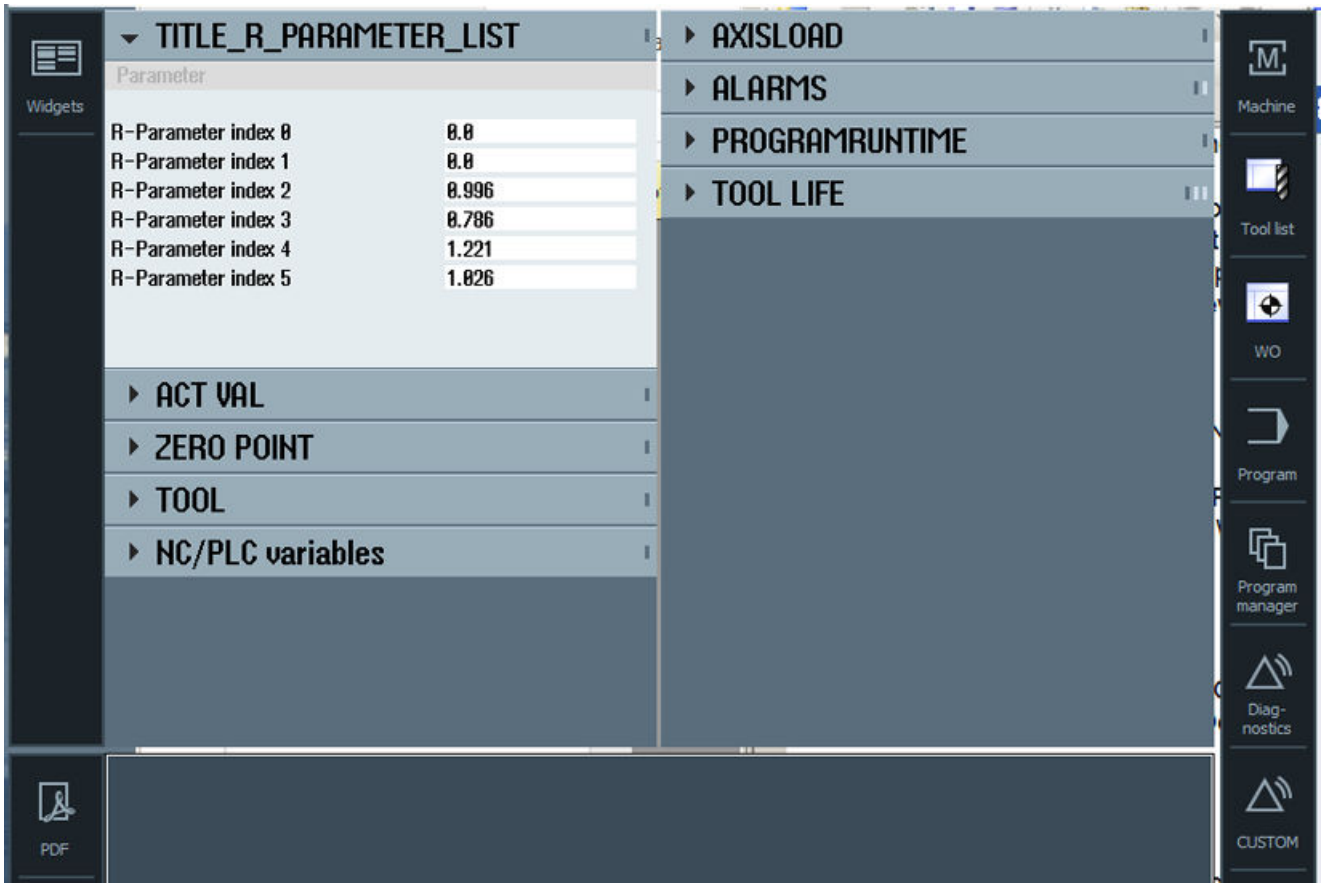
```
MENUITEM107= name:=mieasyXML, menuItemStyle:=misMenu,
onClicked:="hidePopup(); sendCmd(OPERATE, CustomXML);
showApp(defaultFrame, OPERATE)", image:=<bild.png>,
textID:= <textid>
```

In turn, this menu element is assigned to menu in the Display Manager:

```
MENU020= name:=mOperate3, menuStyle:=msMenu,
defaultFrame:=fOperate3, items:="miMachine, miToolList,
miWorkOffset, miProgramEdit, miProgramManager, miDiagnosis,
mieasyXML, miStretch, miMaximizeOperate3"
```

3.15.3 Display Manager widget

Widgets in the Display Manager are created and managed analogously to the sidescreen widgets. However, "Sidescreen Widgets" are described in the configuration files of the Display Manager, e.g. in the INI files **sldmwidgets*.ini**. The reference to the files is established via **systemconfiguration.ini**.



An example with file **rparameter_widget.xml** is provided in Chapter "Sidescreen widget (Page 232)".

Example

systemconfiguration.ini

```
DLG109= name:=SlWidgetsApp,
implementation:=sldmsidescreenapp.SlSideScreenDialog,
process:=SlHmiHost1, preload:=false, cmdline:="-sidescreen1
sldmwidgets1.ini -sidescreen2 sldmwidgets2.ini -spacing 3"
```

The dialog name is assigned a frame:

```
FRAME020= name:=fTop, x:=66, y:=66, width:=760, height:=505,
app:=SlWidgetsApp
```

The defined frames in a display:

```
DISPLAY001= name:=d3, frames:="fHeader, fOperate3, fMenuOperate3,
fTop, fMenuTop, fBottom, fMenuBottom"
```

sldmwidgets1.ini

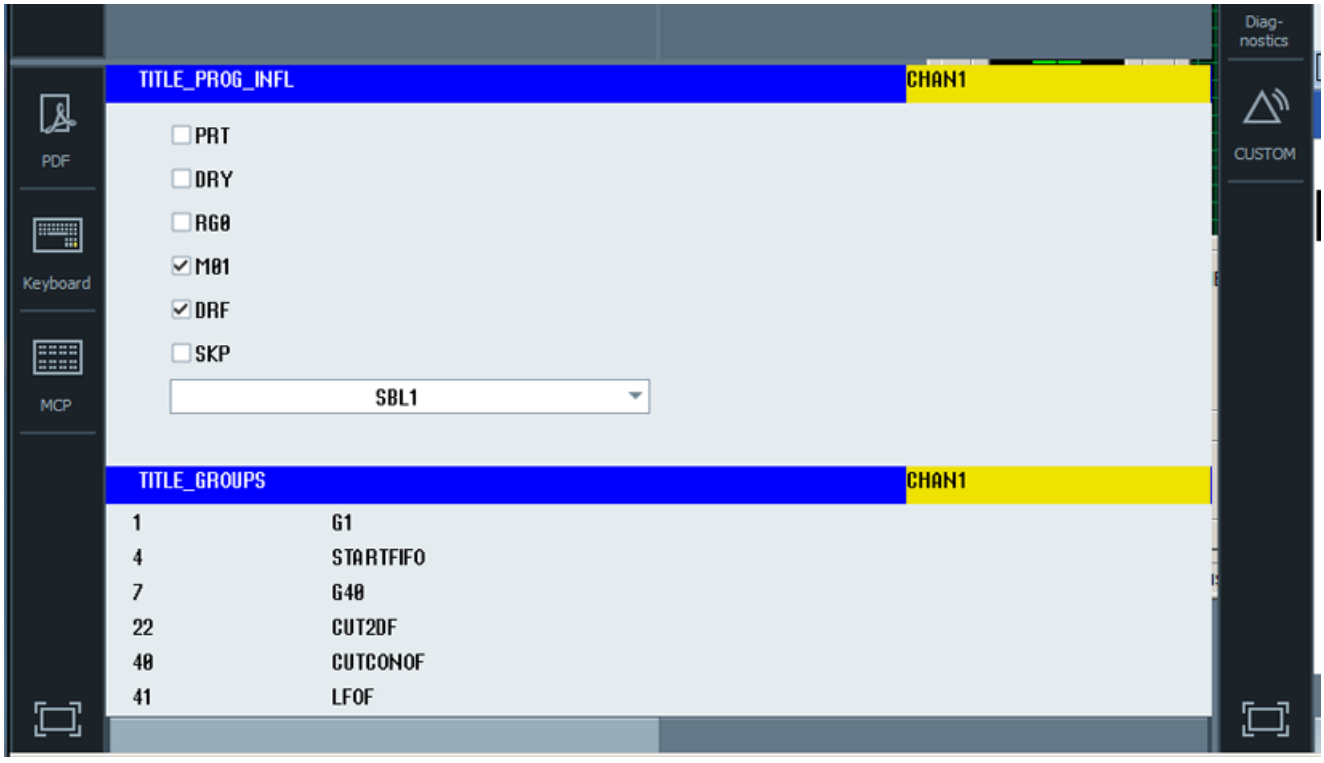
The structure and the significance of the switches and elements is described in Chapter "Sidescreen application (Page 228)".

```
[Sidescreen]
PAGE001= name:=WidgetsPage, implementation:=SlSideScreenPage
[Page_WidgetsPage]
ELEMENT001= name:=EE, implementation:=SlSideScreenElement
ELEMENT002= name:=AxesPosition, implementation:=SlSideScreenElement
ELEMENT003= name:=WorkOffset, implementation:=SlSideScreenElement
ELEMENT004= name:=ActTool, implementation:=SlSideScreenElement
ELEMENT005= name:=VariableView, implementation:=SlSideScreenElement

[Element_EE]
TextId=TITLE_R_PARAMETER_LIST
TextFile=rparameter_widget
TextContext=EASY_XML
WIDGET001= name:=rparameter_widget,
implementation:=slagmforms.SlEESideScreenWidget
```

WIDGET001 is assigned the script to be run (without file extension) and the implementation **slagmforms.SlEESideScreenPage**.

3.15.4 Display Manager sidescreen page

Sidescreen pages in the Display Manager is created and managed analogously to the sidescreen pages. However, "Sidescreen pages" is described in INI files **sldmxxxpage*.ini**. The reference to these files is established via file **systemconfiguration.ini**.



An example with file **sidescreen_proginfluence.xml** is provided in Chapter "Sidescreen page (Page 234)".

Example

sldmmcppage.ini

```
DLG110= name:=SlMcpApp,
implementation:=sldmsidescreenapp.SlSideScreenDialog,
process:=SlHmiHost1, preload:=false, cmdline:="-sidescreen1
sldmmcppage.ini"
```

The dialog name is assigned a frame:

```
FRAME028= name:=fBelowOperate, x:=896, y:=838, width:=1024,
height:=242, app:=SlKeyboardApp, runnableApps:="SlKeyboardApp,
SlMcpApp"
```

The defined frames of a display:

```
DISPLAY002= name:=d4, frames:="fHeader, fOperate4, fMenuOperate4,
fBelowOperate, fMenuBelowOperate, fTop, fMenuTop, fBottom,
fMenuBottom"
MENUITEM124= name:=miMcp, menuItemStyle:=misMenu,
onClicked:="hidePopup(); showApp(defaultFrame, SlMcpApp)",
image:=dm_mcp.png, textID:=SL_DM_MCP
```

sldmmcppage.ini

```
[Sidescreen]
PAGE001= name:=McpPage,
implementation:=slsidescreenmcppage.SlSideScreenMcpPage
PAGE002= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage
```

PAGE002 is assigned the script to be run (without file extension) and the implementation **slagmforms.SlEESideScreenPage**.

3.16 Dialog selection via PLC hardkeys

Field of application

The following functions can be initiated by the PLC in the operating software:

- Select an operating area
- Select certain scenarios within operating areas
- Execute functions configured at softkeys

Hardkeys

All keys - also the PLC keys - are subsequently referred to as hardkeys. A maximum of 254 hardkeys can be defined. The following allocation applies:

Key number	Application
Key 1 – key 9	Keys on the operator panel front
Key 10 – key 49	Reserved
Key 50 – key 254 Key 50 – key 81	PLC keys: Reserved for OEMs
Key 255	Pre-assigned with control information

Hardkeys 1 - 9 are pre-assigned as follows:

Key designation		Action / effect
HK1	MACHINE	Selects "Machine" operating area, last dialog
HK2	PROGRAM	Selects "Program" operating area, last dialog or last program
HK3	OFFSET	Selects "Parameter" operating area, last dialog
HK4	PROGRAM MANAGER	Selects "Program" operating area, "Program manager" root screen No function
HK5	ALARM	Selects "Diagnostics" operating area, "Alarm list" dialog
HK6	CUSTOM	Selects "Custom" operating area

Configuration

The configuring is realized in the systemconfiguration.ini configuration file in the section [keyconfiguration]. Each line defines what is known as a hardkey event. A hardkey event is the n-th actuation of a specific hardkey. For example, the second and third actuation of a specific hardkey can result in different responses.

The entries in the systemconfiguration.ini configuration file can be overwritten with user-specific settings. The directories [System user directory]/cfg and [System oem directory]/cfg are available for this purpose.

The lines for configuring the hardkey events have the following structure:

```
KEYx.n = area:=area, dialog:=dialog, screen:=screen, forms:=form,
menus:=menu, action:=menu.action, cmdline:=cmdline
KEYx.n = area:=area, dialog:=dialog, cmdline:=cmdline,
action:= action
```

x: Number of the hardkey, range of values: 1 – 254

n: Event number – corresponds to the nth actuation of the hardkey, range of values: 0 – 9

Requirement

The PLC user program must fulfill the following requirement:

Only one hardkey is processed. As a consequence, a new request can only be set if the operating software has acknowledged the previous request. If the PLC user program derives the hardkey from an MCP key, it must provide sufficient buffer storage of the key(s) to ensure that no fast keystrokes are lost.

PLC interface

An area to select a hardkey is provided in the PLC interface. The area is in DB19.DBB10. Here, the PLC can directly specify a key value of between 50 and 254.

Acknowledgment by the operating software takes place in two steps. This procedure is necessary so that the operating software can correctly identify two separate events if the same key code is entered twice consecutively. In the first step, control information 255 is written to byte DB19.DBB10. This defined virtual key stroke enables the HMI to identify every PLC key sequence uniquely. The control information is of no significance to the PLC user program and must not be changed. In the second step, the actual acknowledgment takes place with respect to the PLC by clearing DB19.DBB10. From this point in time, the PLC user program can specify a new hardkey. In parallel, the actual hardkey request is processed in the operating software.

Example

Configuration file:

```
; configuration of OP hardkeys (KEY1-KEY9) and
; PLC hardkeys (KEY50-KEY254)
[keyconfiguration]
; MACHINE key (hardkey block)
KEY1.0 = area:=AreaMachine, dialog:=SlMachine
; PROGRAM key (hardkey block)
KEY2.0 = area:=AreaProgramEdit
; OFFSET key (hardkey block)
KEY3.0 = area:=AreaParameter
; PROGRAM MANAGER key (hardkey block)
KEY4.0 = area:=AreaProgramManager
; ALARM key (hardkey block)
```

```
KEY5.0 = area:=AreaDiagnosis, dialog:=SlDgDialog,
cmdline:="-slGfwHmiScreen SlDgAeAlarmsScreen"
; CUSTOM (hardkey block)
KEY6.0 = area:=Custom
; MACHINE key (optional, Shift + F10)
KEY7.0 = area:=AreaMachine, dialog:=SlMachine,
cmdline:="-MKey"
; MENU USER (hardkey block, Ctrl + F10)
KEY10.0 = area:=MenuUser
; MENU FUNCTION (hardkey block, Ctrl + Shift + F10)
KEY11.0 = area:=MenuFunction
KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog
```

The area and dialog identifiers can be found in the systemconfiguration.ini from [System Siemens directory]/cfg.

If only the **SlEECustomDialog** is used, the parser expects the **xmldial.xml** file in the application directory as well as a menu tag with the name **main**. Other file names or main menu names can be declared by adding the **cmdline** key. The **-mainModule** parameter specifies the XML description to be loaded. The parser expects the main menu in this script. The **-entry** parameter specifies the name of the main menu. If this parameter is missing, the parser uses the name **main** to start the function. The **-conf** parameter with the value **slagmdialog.hmi** must always be included.

Here is an example:

```
KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog,
cmdline:="-conf slagmdialog.hmi -mainModule restore.xml -entry cmc2"
KEY51.0 = area:=CustomXML, dialog:=SlEECustomDialog,
cmdline:="-conf slagmdialog.hmi -mainModule activate.xml
-entry cmc1"
```

User interface

The following entries in the systemconfiguration.ini file enable the use of Easy XML for the function described above.

```
AREA0111= name:=CustomXML, dialog:=SlEECustomDialog,
panel:=SlHdStdHeaderPanel
DLG056= name:=SlEECustomDialog,
implementation:=slagmdialog.SlEECustomDialog, process:=SlHmiHost1,
preload:=false, terminate:=true, cmdline:="-conf slagmdialog.hmi"
```

3.17 Assigning user dialogs to reserved softkeys

A softkey is reserved in all standard operating areas for expansion of the existing functionality. Corresponding files are stored in the following directory in the system for activation and linking of an OEM softkey with a user project:

/siemens/sinumerik/hmi/template/cfg

These files contain operating area-specific XML descriptions which are interpreted and integrated in the menu tree of the operating area upon start-up of the operating software.

For integration of a separate application, the file relevant for the corresponding operating area is to be copied from the template directory into the following directory and modified:

/oem/sinumerik/hmi/template/cfg

3.17.1 Defining language-neutral softkey text

The TEXTFILE tag contains the file name of the text file which is to be assigned to the OEM area. The file name is to be specified without language and file extension.

Syntax

```
<TEXTFILE>oemtextfile</TEXTFILE>
```

Example

oem_xml_screens_deu.ts

```
<TEXTFILE>oem_xml_screens</TEXTFILE>
```

The softkey text to be displayed is managed in the **property** tag in the **softkey** tag. The OEM value is to be replaced by the desired text ID. The **translationContext** property references an area in the text file to which the text is assigned. In the case of easyXML, the EASY_XML context is to be specified.

Syntax

```
<PROPERTY name="textID" type="QString">OEM</PROPERTY>
<PROPERTY name="translationContext" type="QString">OEMContext</
PROPERTY>
```

Example

```
<PROPERTY name="textID" type="QString">MY_TEXT1</PROPERTY>
<PROPERTY name="translationContext" type="QString">EASY_XML</
PROPERTY>
```

3.17.2 Assigning an Easy XML area

The navigation target must also be defined in the **softkey** tag. Thus, the navigation target **CustomXML** is to be entered in the **name** attribute in the **area** tag. The attributes **dialog** and **screen** are to be deleted as these parameters are defined automatically.

Syntax

```
<NAVIGATION target="area">
<AREA name="OEMArea" dialog="OEMDialog" screen="OEMScreen"/>
</NAVIGATION>
```

Example

```
<NAVIGATION target="area">
<AREA name="CustomXML" />
</NAVIGATION>
```

If navigation target **area** is used, the parser expects the file xmldial.xml as a start module and the **main** menu as an entry point for menu management. In order to be able to offer multiple applications at the same time, the **switchToDialog** function is to be used. The NAVIGATION tag must then be replaced with the FUNCTION tag in this regard. The **CustomXML** area, the **SlEECustomDialog** dialog, and the name of the start module and of the main menu are to be transferred to this function as call arguments.

You return to the standard user area from the easyXML area with the **switchToArea** tag.

Syntax

```
<switchToArea name="<Area>"/>
...
<FUNCTION args="-area CustomXML -dialog SlEECustomDialog -mainModule
<Filename> -entry <Menuname>" name="switchToDialog"/>
```

Example project

sldiagnose_oem.xml

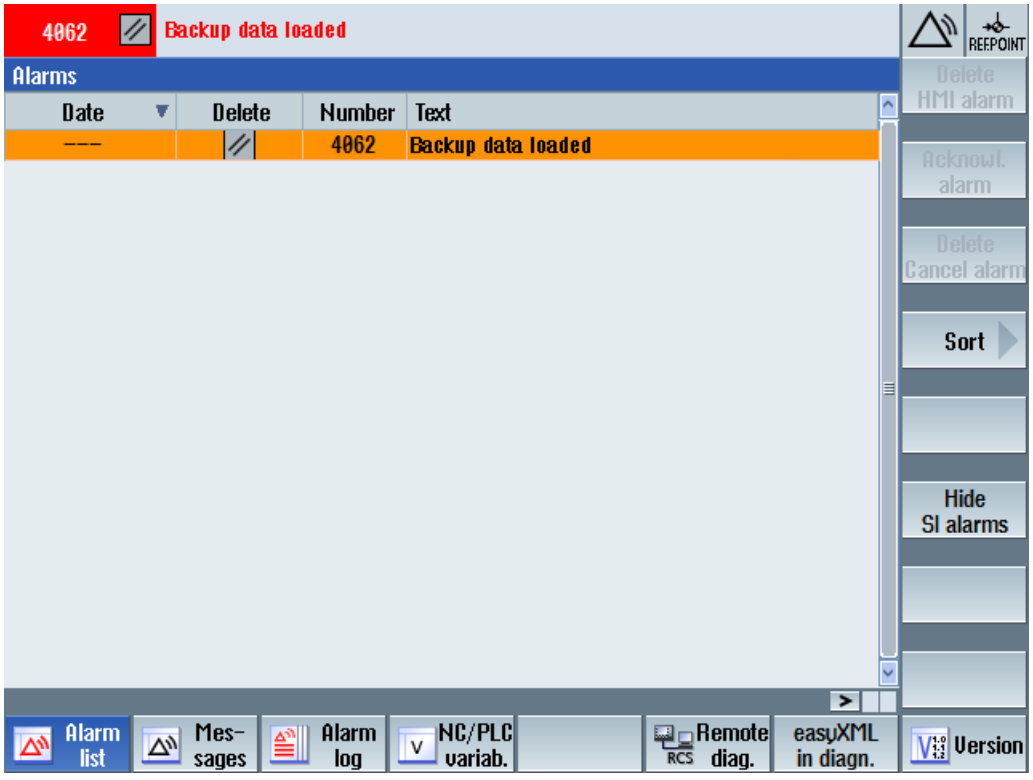
```
<!DOCTYPE DIALOG>
<DIALOG>
<TEXTFILE>oem_xml_screens</TEXTFILE>
<!-- ========================================================================
-->
<!-- OEM softkey number 7 on first horizontal main menu in area diagnosis --
>
<!-- ========================================================================
-->
<MENU name="DgGlobalHu">
 <ETCLEVEL id="0">
 <SOFTKEYGROUP name="GroupEtc">
 <SOFTKEY position="7">
  <PROPERTY name="textID" type="QString">DG_SK</PROPERTY>
  <PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>
  <NAVIGATION target="area">
   <AREA name="CustomXML" />
  </NAVIGATION>
 </SOFTKEY>
</SOFTKEYGROUP>
</ETCLEVEL>
</MENU>
</DIALOG>
```

or

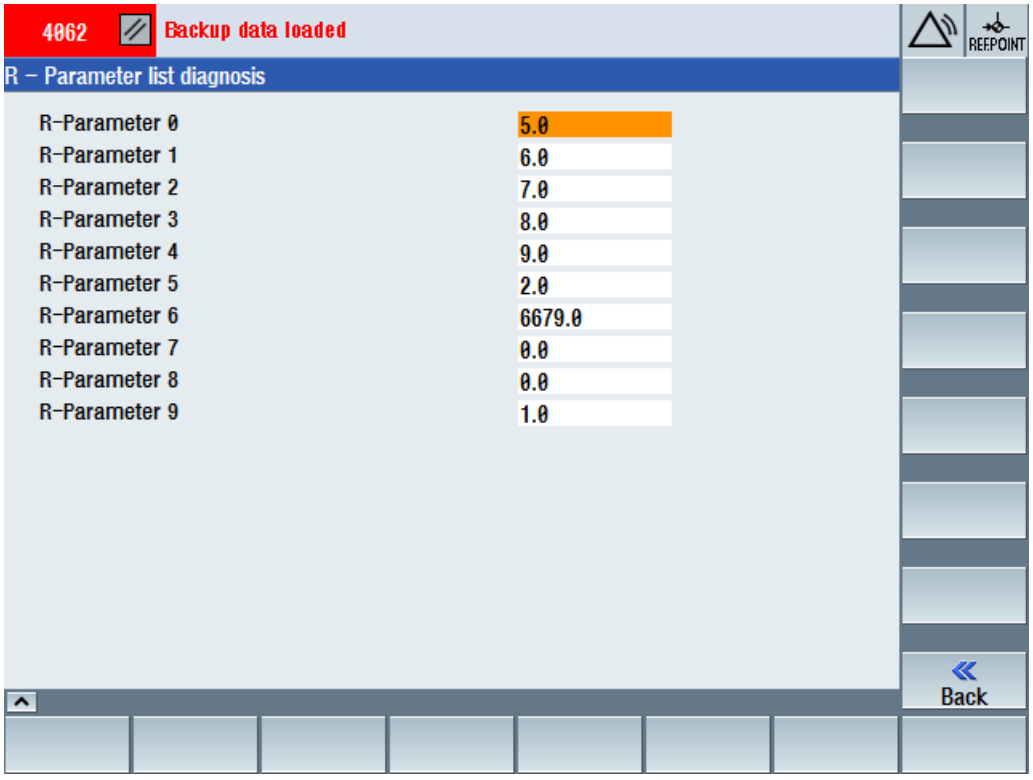
```
<!DOCTYPE DIALOG>
<DIALOG
<!-- ========================================================================
-->
<!-- OEM softkey number 7 on first horizontal main menu in area diagnosis --
>
<!-- ========================================================================
-->
<TEXTFILE>oem_xml_screens</TEXTFILE>
<MENU name="DgGlobalHu">
 <ETCLEVEL id="0">
 <SOFTKEYGROUP name="GroupEtc">
 <SOFTKEY position="7">
  <PROPERTY name="textID" type="QString">DG_SK</PROPERTY>
  <PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>
  <FUNCTION args="-area CustomXML -dialog SlEECustomDialog -mainModule
dg.xml -entry main" name="switchToDialog"/>
 </SOFTKEY>
</SOFTKEYGROUP>
</ETCLEVEL>
</MENU>
</DIALOG>
```

dg.xml

```
<DialogGui>
<menu name = "main">
 <open_form name = "R_PARAMETER_LIST" />
 <softkey_back>
  <switchToArea name="AreaDiagnosis" dialog="SlDgDialog"/>
 </softkey_back>
</menu>
<!-- This form shows a R - parameter list -->
<form name = "R_PARAMETER_LIST">
 <init>
 <caption>R - Parameter list diagnosis</caption>
 <data_access type="true" />
  <control name = "c1" xpos = "322" ypos = "34" refvar="nck/Channel/
Parameter/R[0]" hotlink="true" />
  <control name = "c2" xpos = "322" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
  <control name = "c3" xpos = "322" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
  <control name = "c4" xpos = "322" ypos = "94" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
  <control name = "c5" xpos = "322" ypos = "114" refvar="nck/Channel/
Parameter/R[4]" hotlink="true" />
  <control name = "c6" xpos = "322" ypos = "134" refvar="nck/Channel/
Parameter/R[5]" hotlink="true" />
  <control name = "c7" xpos = "322" ypos = "154" refvar="nck/Channel/
Parameter/R[6]" hotlink="true" />
  <control name = "c8" xpos = "322" ypos = "174" refvar="nck/Channel/
Parameter/R[7]" hotlink="true" />
  <control name = "c9" xpos = "322" ypos = "194" refvar="nck/Channel/
Parameter/R[8]" hotlink="true" />
  <control name = "c10" xpos = "322" ypos = "214" refvar="nck/Channel/
Parameter/R[9]" hotlink="true" />
 </init>
 <paint>
  <text xpos = "23" ypos = "34">R-Parameter 0</text>
  <text xpos = "23" ypos = "54">R-Parameter 1</text>
  <text xpos = "23" ypos = "74">R-Parameter 2</text>
  <text xpos = "23" ypos = "94">R-Parameter 3</text>
  <text xpos = "23" ypos = "114">R-Parameter 4</text>
  <text xpos = "23" ypos = "134">R-Parameter 5</text>
  <text xpos = "23" ypos = "154">R-Parameter 6</text>
  <text xpos = "23" ypos = "174">R-Parameter 7</text>
  <text xpos = "23" ypos = "194">R-Parameter 8</text>
  <text xpos = "23" ypos = "214">R-Parameter 9</text>
 </paint>
</form>
</DialogGui>
```


4062
Backup data loaded
REFPOINT
Alarms
Date
Delete
Number
Text

4062
Backup data loaded
Delete HMI alarm
Acknowl. alarm
Delete Cancel alarm
Sort
Hide SI alarms
Alarm list
Mes-sages
Alarm log
NC/PLC variab.
Remote diag.
easyXML in diagn.
Version


4062
Backup data loaded
REFPOINT
R – Parameter list diagnosis
R-Parameter 0 5.0
R-Parameter 1 6.0
R-Parameter 2 7.0
R-Parameter 3 8.0
R-Parameter 4 9.0
R-Parameter 5 2.0
R-Parameter 6 6679.0
R-Parameter 7 0.0
R-Parameter 8 0.0
R-Parameter 9 1.0
Back

3.17.3 Tag descriptions

Tag softkey

Outside of the **softkey** tag, the softkey state can be set with the **state** attribute using the POSITION attribute. The softkey number for which the state should be set is to be specified as the attribute value.

Example

```
<menu name = "menu_sktest">

 <state type="$$$define_sk_type" POSITION="2"/>

<softkey POSITION="2" type="user_controlled" picture="$$$bmp_name">
 <caption>Toggle%nSK</caption>
 <if>
 <condition>sk_type == 0 </condition>
 <then>
  <op> sk_type = 1 </op>
  <op> define_sk_type = _T"PRESSED" </op>
  <op>bmp_name = _T"f:\appl\red_led_on.bmp"</op>
  <state type="$$$define_sk_type" />
 </then>
 <else>
  <op> define_sk_type = _T"NOTPRESSED" </op>
  <op>bmp_name = _T"f:\appl\red_led_off.bmp"</op>
  <op> sk_type = 0 </op>
  <state type="$$$define_sk_type" />
 </else>
 </if>
  <print text="%s">sk_type_name</print>
  <navigation>menu_sktest</navigation>
</softkey>
...
...
```

Tag typedef

Note

Pre-allocation of the elements in the examples must be removed.

Within the type definition, a variable is declared with the **element** tag. The attributes of the tag correspond with the attributes of the **let** instruction. The elements can be preallocated when creating the variables.

Example

RECT:

```
<typedef name="StructRect" type="struct" >
<element name="left" type="int" />
<element name="top" type="int" />
<element name="right" type="int" />
<element name="bottom" type="int" />
</typedef>
```

POINT:

```
<typedef name="StructPoint" type="struct" >
<element name="x" type="int" />
<element name="y" type="int" />
</typedef>
```

SIZE:

```
<typedef name="StructSize" type="struct" >
<element name="width" type="int" />
<element name="height" type="int" />
</typedef>
```

Tag let

The number of field elements is to be specified with the **dim** attribute. For a two-dimensional field, the second dimension is specified after the first dimension separated by a comma. A field element is accessed via the field index, which is specified in square brackets after the variable name. Dimensions of the arrays can be defined directly or via the content of a variable.

Example

```
 <let name="d1" >3</let>

<let name="list_string" dim="3" type="string" />
```

or

```
<let name="list_string" dim="$d1" type="string" />
...
...
<op>
  list_string[2] = _T"test";
</op>
```

or

```
<let name="d1" >3</let>
<let name="d2" >6</let>

<let name="list_string3" dim="3, $d2" type="string" />

<op>
  list_string3[2, 5] = _T"test";
</op>
```

3.18 Creating language-neutral texts

All texts used in the dialog forms can be managed language-neutral, where text identifiers are used in the dialogs which are replaced with text in the currently selected language during runtime. This text is to be saved together with the text identifier for every available language in a text file in UTF8 format.

The file oem_xml_screens_<language code>.ts is assigned to the Easy XML function as a text file by default. The EASY_XML identifier is to be specified as text context.

Two dollar symbols are to be prefixed to the text identifier in the script to mark a test identifier.

Structure

oem_xml_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>MY_TEXT1</source>
    <translation>text1 from textfile</translation>
    </message>
    <message>
    <source>MY_TEXT2</source>
    <translation>text2 from textfile</translation>
    </message>
</context>
</TS>
```

Example

```
<text xpos ="120" ypos="158">$$MY_TEXT1</text>
```

3.19 Project-specific text files

3.19.1 Creating project-specific text files

For managing separate projects, separate text files can be used for a project, each form, and each menu. The TEXTFILE attribute gives notification in the corresponding tags.

The name of the text file is to be transferred as an attribute value without language identifier and file extension. The texts can continue to be accessed until the form or the menu is closed. If a text file is loaded with the tag **DialogGui**, the content may be accessed until the project is exited. If you use a text context more than once, the search for text identifiers is performed in the last loaded file. One text context can be used per text file.

Example

main_form_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>MAIN_FORM_TITLE</source>
    <translation>user text file title</translation>
    </message>
</context>
</TS>
```

Activation

```
<DialogGui textfile="main_form_screens">
...
...
</DialogGui>
```

or

```
<form name="main_form" textfile="main_form_screens">
...
...
</form>
```

or

```
<menu name="main" textfile="main_form_screens">
...
...
</menu>
```

Example dialog

main_form_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>MAIN_FORM_TITLE</source>
    <translation>user text file title</translation>
    </message>
    <message>
    <source>MAIN_FORM_TEXT</source>
    <translation>text from text file</translation>
    </message>
</context>
</TS>
```

main2_sktext_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>SK1</source>
    <translation>Softkey1</translation>
    </message>
</context>
</TS>
```

form2_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>FORM2_TITLE</source>
    <translation>Form2 Title</translation>
    </message>
    <message>
    <source>FORM2_TEXT</source>
    <translation>Form2 text from text file</translation>
    </message>
</context>
</TS>
```

Dialog script

xmldial.xml

```
<DialogGui textfile="main_form_screens">
<menu name = "main">
    <open_form name = "main_form" />
    <softkey POSITION="1">
    <caption>Menu 2</caption>
    <navigation>menu_2</navigation>
    </softkey>
</menu>
<menu name = "menu_2" textfile="main2_sktext_screens">
    <open_form name = "form2" />
    <softkey POSITION="1">
    <caption>$$SK1</caption>
    </softkey>
    <softkey_back>
    <navigation>main</navigation>
    </softkey_back>
</menu>
<form name="main_form" >
 <init>
    <data_access type = "true" />
    <caption>$$MAIN_FORM_TITLE</caption>
 </init>
 <paint>
    <text xpos="5" ypos="30">$$MAIN_FORM_TEXT</text>
 </paint>
</form>
<form name="form2" textfile="form2_screens">
 <init>
    <data_access type = "true" />
    <caption>$$FORM2_TITLE</caption>
 </init>
 <paint>
    <text xpos="5" ypos="30">$$FORM2_TEXT</text>
 </paint>
</form>
</DialogGui>
```

3.19.2 Specifying text context

Grouping of the texts is possible within a text file by way of independently named area names. The area identifier is defined within the **context** tag with the **name** tag.

Syntax

```
<context>
 <name>name</name>
</context>
```

Example

```
<context>
 <name>my context 1</name>
...
...
</context>
<context>
 <name>my context 2</name>
...
...
</context>
```

If particular context is being used, the context name is to be specified together with the text file name for a project, a form or a menu via the TEXTCONTEXT attribute. Access to texts stored within the context remains possible until a new context is set. Thus, a context set for the project is replaced by the context set for a form or a menu.

Project example

form2_screens_deu.ts

```
?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
 <name>EASY_XML</name>
  <message>
   <source>FORM2_TITLE</source>
   <translation>Form2 Title</translation>
  </message>
  <message>
   <source>FORM2_TEXT</source>
   <translation>Form2 text from text file</translation>
  </message>
</context>
<context>
 <name>MY_FORM_CONTEXT</name>
  <message>
   <source>FORM3_TITLE</source>
   <translation>Title from the text context MY_FORM_CONTEXT</translation>
  </message>
  <message>
   <source>FORM3_TEXT</source>
   <translation>Form3 text from text file</translation>
  </message>
</context>
<context>
 <name>MY_SOFTKEY_CONTEXT</name>
  <message>
   <source>MENU3_SK1</source>
   <translation>SK%n1</translation>
  </message>
</context>
</TS>
```

using_textcontext.xml

```
...
<menu name = "menu3" textfile="form2_screens"
textcontext="MY_SOFTKEY_CONTEXT">
 <open_form name = "menu3_form" />
 <softkey POSITION="1">
  <caption>$$SK1</caption>
 </softkey>
 <softkey_back>
  <navigation>main</navigation>
 </softkey_back>
</menu>
<form name="menu3_form" textfile="form2_screens"
textcontext="MY_FORM_CONTEXT">
 <init>
  <data_access type = "true" />
  <caption>$$FORM3_TITLE</caption>
 </init>

 <paint>
  <text xpos="5" ypos="30">$$FORM3_TEXT</text>
 </paint>
</form>
...
...
```

3.19.3 Specifying Textfilelist

If you want to use multiple text files for a project, you can forward the names of the required text files in a list to the parser using the **textfilelist** attribute.

Notation of the text file names is realized line by line and must be concluded with a line break. The name of the file without language extension and file extension is expected as text file name.

Example

The file `form2_screens_deu.ts` is to appear in the list as `form2_screens`.

textfilelist.ini

```
form2_screens
...
...
```

Activation

```
<DialogGui textfilelist="txtfilelist.ini">
...
...
</DialogGui>
```

3.20 Restoring the session

When an Easy XML project is closed, all local variables are enabled and script instructions are unloaded. If you want to retain data for variables after the control is switched off, these variables are to be designated the **permanent** attribute. Data which is to be retained until the control is switched off must be designated the **poweroff** attribute.

If the Easy XML application is selected again, it is possible in the main menu to control the menu or the form which is to be activated following renewed selection of the application. It is the responsibility of the programmer to ensure that all data required in this regard is made available.

If the **restoresession** attribute is set in the **DialogGUI** tag, the parser organizes reselection of the menu last opened and the last opened form. The programmer is responsible for providing the necessary data.

Generating commissioning dialogs

4

4.1 Overview of functions

Purpose

The "Easy Extend" function provides a simple way of commissioning, activating, deactivating, or testing optional equipment. The available equipment and device states are displayed in a list by the control system. The system can manage a maximum of 64 devices.

Softkeys are used to activate or deactivate a device.

The "Easy Extend" function is available in the "Parameters" operating area.

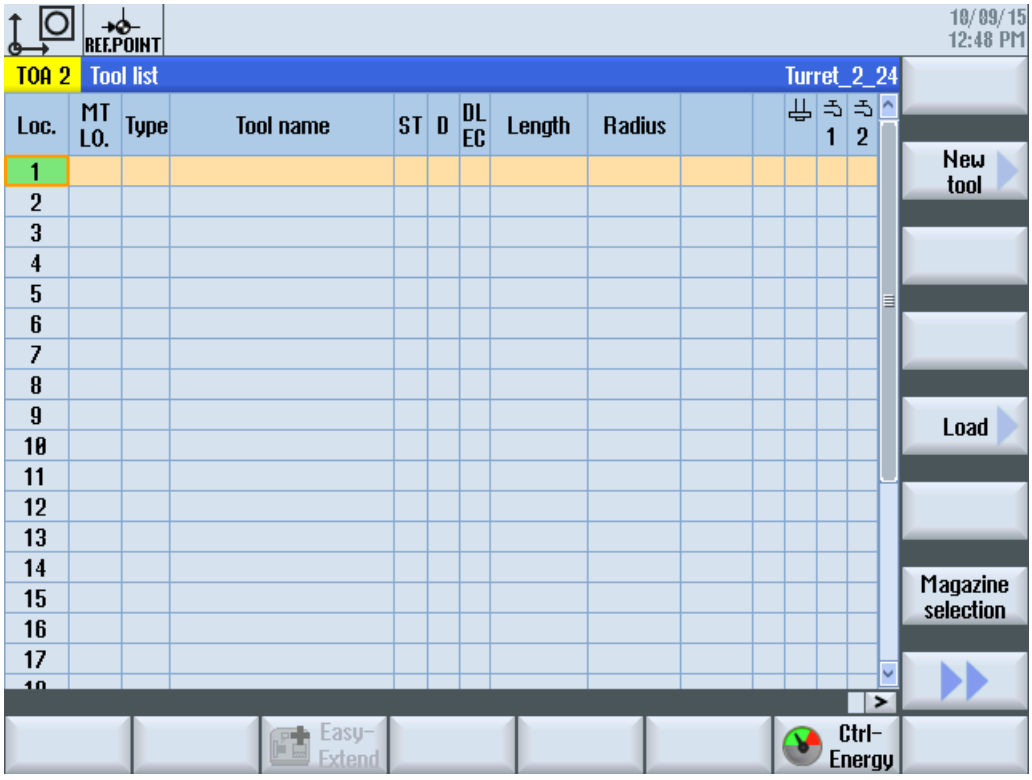


Figure 4-1 Main screen "Parameters"

Configuration

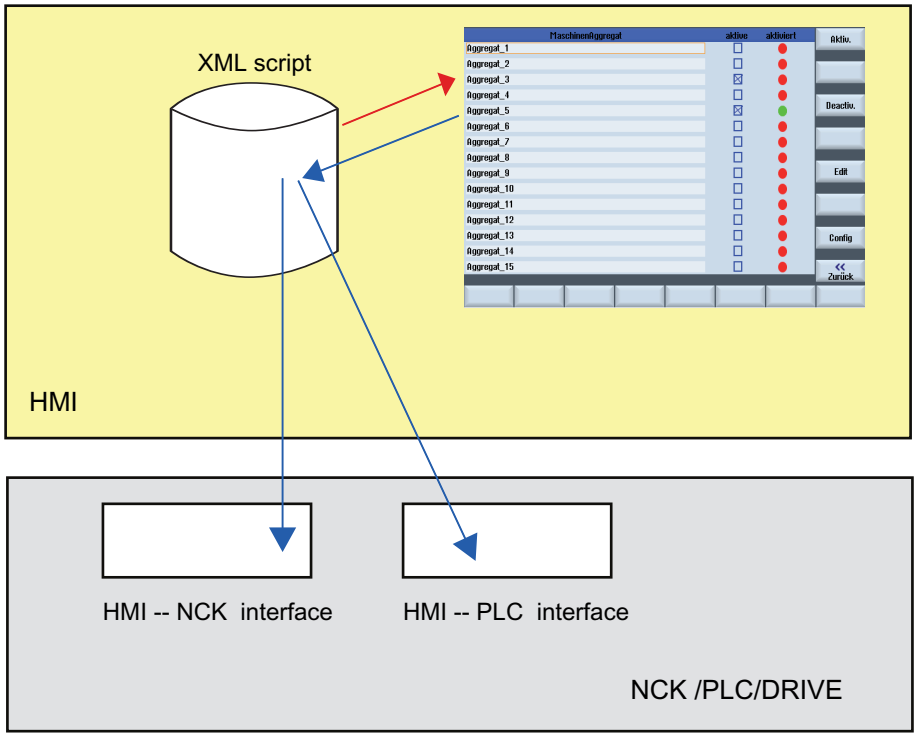


Figure 4-2 How "Easy Extend" works

To use the "Easy Extend" function, the following functions must be configured by the machine manufacturer:

- **PLC ↔ HMI interface**
  The optional devices are managed via the interface between the user interface and the PLC.
- **Script processing**
  The machine manufacturer saves the sequences to be executed for commissioning, activating, deactivating or testing a device, in a statement script.
- **Parameter dialog (optional)**
  The parameter dialog shows device information that is saved in the script file.

Storage of the files

The files belonging to "Easy Extend" are stored on the system CompactFlash card in directory "dvm" (machine manufacturer).

File	Name	Target directory
Text file	oem_aggregate_xxx.ts	`\oem\sinumerik\hmi\lng`
Script file	agm.xml	`\oem\sinumerik\hmi\dvm`
Archive file	Any	`\oem\sinumerik\hmi\dvm\archives`
PLC user program	Any	PLC

4.2 Configuration in the PLC user program

Loading configurations

The configurations created are transferred to the manufacturer directory of the control, with the script and text file. Additionally, the corresponding PLC user program should be loaded.

Programming the equipment

Communication between the operator component and the PLC takes place in the PLC user program via a data block defined by the programmer, in which 128 words are reserved for the management of the devices. The description for the data block identifier is provided in Chapter "PLC_INTERFACE (Page 274)."

The data block must be generated and loaded as a user block. The relevant block must be declared to the EasyExtend function using the **plc_interface** tag in the "agm.xml" script.

Note

Compatibility

We recommend defining data block DB9905 as the PLC interface so that the scripts are also compatible with SINUMERIK 828D.

Example:

```
<plc_interface name = "plc/db1000.dbb0" />
```

To create the block, data formats and symbolic identifiers from the following table can be used:

Data format/symbolic identifier	Meaning
DBX0.0 Enable_1 BOOL Bit OFF OFF HMI → PLC	Device has been started up
DBX0.1 Activate_1 BOOL Bit OFF OFF HMI → PLC	Device is to be activated
DBX0.2 Deactivate_1 BOOL Bit OFF OFF HMI → PLC	Device is to be deactivated
DBB1 Res_1 BYTE Unsigned 0 0	Reserved for future use
DBX2.0 IsActive_1 BOOL Bit OFF OFF PLC → HMI	Device is active
DBX2.1 Error_1 BOOL Bit OFF OFF PLC → HMI	Device has an error
DBB3 DeviceId_1 BYTE Unsigned 0 0	Unique device number

PLC words are assigned beginning with Device 1:

Data block	Device designation
DB9905.DBB0	Device 1
DB9905.DBB4	Device 2
...	...
DB9905.DBB192	Device 49
DB9905.DBB196	Device 50

Four bytes with the following meanings are used for each device:

Byte	Bit	Description	
0	0	== 1	Device has been started up (HMI acknowledgment)
	1	== 1	Device is to be activated (HMI request)
	2	== 1	Device is to be deactivated (HMI request)
	3-7	Reserved	
1	0-7	Reserved	
2	0	== 1	Device is active (PLC acknowledgment)
	1	== 1	Device has an error
	2-7	Reserved	
3	0-7	Unique identifier for the device	

Storage of the files

The Easy Extend files are stored on the system CompactFlash card in the "oem" (MANUFACTURER) and "oem_i" (INDIVIDUAL) directory.

Note

The file name may only contain lower case letters.

File	Name	Target directory
Text file	oem_aggregate_xxx.ts	/oem/sinumerik/hmi/lng/ /oem_i/sinumerik/hmi/lng/
Script file	agm.xml	/oem/sinumerik/hmi/dvm /oem_i/sinumerik/hmi/dvm
Archive file	Any	/oem/sinumerik/hmi/dvm/archives /oem_i/sinumerik/hmi/dvm/archives
PLC user program	Any	PLC

General sequence

The machine manufacturer must execute the following steps to make the required data available:

1. Creating a PLC user program which activates the device during activation on the PLC.
2. Commissioning of the "standard machine" followed by backup of the data in a series startup archive.
3. Installation of the devices, commissioning, followed by read-out of the data as a differential series startup archive.

Note

Changing the machine configuration

Should there be any need to edit the drive machine data, this should be adapted in the control first. This procedure should be repeated for all devices and constellations.

Adding axes

If the machine is extended with machine axes, it is important to install the drive objects (DO) in a fixed sequence because the series startup archive contains the constellation of the machine manufacturer's reference machine and cannot be applied if the sequence is changed.

It is recommended that the following settings be selected for the "control components":

- NC data
- PLC data
- Drive data
  - ACX format (binary)

4.3 Display on the user interface

Dialogs on the user interface

The following dialogs are available for the "Easy Extend" function:

- The control offers a **configurable dialog**, in which the available devices are shown.
- If first commissioning has not taken place yet, the control opens the **commissioning dialog**.

If a commissioning procedure (XML instruction: "START_UP") has been programmed, and the device has still not been commissioned, then the control starts the commissioning procedure.

This involves a complete data backup before the series startup archives saved in the script file are read in.

In the event of an error, the commissioning engineer can decide whether to roll back the commissioning procedure or to rectify possible errors in machine configurations manually.

- Commissioning can be aborted early with the "Cancel" function. The control then copies the previously saved commissioning files back.

If the machine has to be switched off after successful completion of the commissioning, the XML statement "POWER_OFF" can be used to program that a corresponding message is output on the control.

4.4 Creating language-dependent texts

Structure of text file

The XML files with the language-dependent texts must be created in UTF8 format:

Examples

oem_aggregate_eng.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_EXTEND</name>
    <message>
    <source>DEVICE_ONE</source>
    <translation>Device one</translation>
    </message>
    <message>
    <source>DEVICE_TWO</source>
    <translation>Device two</translation>
    </message>
</context>
</TS>
```

oem_aggregate_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_EXTEND</name>
    <message>
    <source>DEVICE_ONE</source>
    <translation>First device</translation>
    </message>
    <source>DEVICE_TWO</source>
    <translation>Second device</translation>
    </message>
</context>
</TS>
```

4.5 User example for a power unit

Activating the drive object

The drive object to be activated has already been commissioned and deactivated again by the machine manufacturer, to market the axis (axes) as an option.

To activate the axis carry out the following steps:

- Activate the drive object via p0105.
- Enable 2nd axis in the channel machine data.
- Back up the drive machine data via p0971.
- Wait until the data has been written.
- Restart the NCK and the drives.

Programming:

```
<DEVICE>
  <list_id>1</list_id>
  <name> "Activate the drive" </name>

  <SET_ACTIVE>
    <data name = "drive/dc/p105[DO5]">1</data>
    <data name = "$MC_AXCONF_MACHAX_USED[4]">5</data>
    <data name = "drive/dc/p971[DO5]">1</data>
    <while>
      <condition> "drive/dc/p971[DO5]" !=0 </condition>
    </while>
    <control_reset resetnc ="true" resetdrive = "true"/>
  </SET_ACTIVE>

  <SET_INACTIVE>
    <data name = "drive/dc/p105[DO5]">0</data>
    <data name = "$MC_AXCONF_MACHAX_USED[4]">0</data>
    <data name = "drive/dc/p971[DO5]">1</data>
    <while>
      <condition> "drive/dc/p971[DO5]" !=0 </condition>
    </while>
    <control_reset resetnc ="true" resetdrive = "true"/>
  </SET_INACTIVE>

</DEVICE>
```

Activating the PLC-controlled device

The device is addressed via output byte 10 and signals data set ready to the PLC via input byte 9.

The output byte is set to the specified coding for activation. The While loop then waits for the data set ready of the device.

Programming:

```
<SET_ACTIVE>
  <DATA name = "plc/qb10"> 8 </DATA>
  <while>
    <condition> "plc/ib9" !=1 </condition>
  </while>
</SET_ACTIVE>
```

4.6 Script language

Note

All script elements described in the Generating user dialogs (Page 25) function form the basis for the "Easy Extend" function. Additional script elements are defined to manage additional devices.

Program parts of the script

The script is divided into the following areas:

- "Easy Extend" identifier
- Frame to define the actions that can be executed for a device
- Identifier for the device
- Identifier for commissioning the device
- Identifier for activating the device
- Identifier for deactivating the device
- Identifier for testing the device
- Identifier for the parameter dialog

The individual tags are described in the following chapters.

Description

Identifier <tag>	Meaning
AGM	Identifier for the "Startup Wizard"
DEVICE	Identifier for the description of the device. **Attributes:** • **option_bit** The device is assigned a fixed bit number for the option management.
NAME	The identifier specifies the name of the device to be displayed in the dialog. If a text reference is used, the dialog displays the text which is saved for the identifier.
START_UP	The identifier contains a description of the sequences required for commissioning the device.
SET_ACTIVE	The identifier contains a description of the sequences required to activate the device. **Attributes:** • **timeout** The attribute permits a timeout to be specified in seconds. The system interrupts processing if the script has still not been completed after this time.

Identifier <tag>	Meaning
SET_INACTIVE	The identifier contains a description of the sequences required to shut down the device. **Attributes:** • **timeout** The attribute permits a timeout to be specified in seconds. The system interrupts processing if the script has still not been completed after this time.
TEST	The identifier contains the statements for testing the operating capability of a device. **Attributes:** • **timeout** The attribute permits a timeout to be specified in seconds. The system interrupts processing if the script has still not been completed after this time.
UID	Unique numerical identifier to identify the device in the PLC ↔ HMI interface.
VERSION	Identifier for a version

Negative acknowledgment of the function execution

With the automatically provided variable "$actionresult", the system can inform the XML parser of a negative execution result. If the value is set to zero, the parser aborts the function processing.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE AGM>
<AGM>                              Identifier for the "Startup Wizard"
<DEVICE>
  <NAME> Device 1 </NAME>          Identifier for the device
  <START_UP>                       Identifier for commissioning the device
  …
  </START_UP>
  <SET_ACTIVE>                     Identifier for activating the device
  …
  </SET_ACTIVE>
  <SET_INACTIVE>                   Identifier for deactivating the device
  …
  </SET_INACTIVE>
  <TEST>                           Identifier for testing the device
  …
  </TEST>
</DEVICE>
…
</AGM>
```

4.6.1 CONTROL_RESET

Description

This identifier allows one or more control components to be restarted. Execution of the script is only continued when the control has resumed cyclic operation.

Programming

Identifier:	**CONTROL_RESET**	
Syntax:	<CONTROL_RESET resetnc="TRUE" />	
Attributes:	resetnc="true"	The NC component is restarted.
	resetdrive="true"	The drive components are restarted.

4.6.2 FILE

Description

The identifier enables standard archives to be read in or created.

- Reading in an archive:
  The file name of the archive must be specified for reading in an archive.
- Creating an archive:
  If the attribute create= "true" is specified, the function creates a standard archive (*.arc) under the specified name and stores the file in the .../dvm/archives directory.

Programming

Identifier:	**FILE**	
Syntax:	<file name ="<archive name>" />	
	<file name ="<archive name>" create="true" class="<data classes>" group="<area>" />	
Attributes:	name	Identifier for the file name

create	A commissioning archive is created under the specified name in the .../dvm/archives/ directory.
group	Specifies the data groups that are to be contained in the archive. If several data groups are to be saved, the groups should be separated by a blank. The following data groups can be contained in the archive: • NC • PLC • HMI • DRIVES

Example

```
<!-- Create data class archive -->
<file name="user.arc" create="true"
group="nc plc hmi" />

<!—Read archive into the control->
<file name="user.arc" />
```

4.6.3 OPTION_MD

Description

The identifier allows option machine data to be redefined. As delivered, the system uses MD14510 $MN_USER_DATA_INT[0] to $MN_USER_DATA_INT[3].

If the PLC user program manages the options, the appropriate data words must be provided in a data block or GUD.

The data is structured in bits. Starting with bit 0, there is a fixed assignment of the bits to the listed devices, i.e. bit 0 is assigned to device 1, bit 1 to device 2, etc. If more than 16 devices are managed, the address identifiers of the device groups 1-3 are assigned via the area index.

Note

Converting the value range

The value range of MD14510 $MN_USER_DATA_INT[i] is from -32768 to +32767. To activate the devices bit-by-bit via the machine data dialog, the bit combination must be converted to decimal representation.

Programming

Identifier:	**OPTION_MD**	
Syntax:	Area 0:	
	<option_md name = "Address identifier of the data" />	
	OR:	
	<option_md name = "Address identifier of the data" index= "0"/>	
	Area 1 to 3:	
	<option_md name = "Address identifier of the data" index= "Area index"/>	
Attributes:	name	Identifier for the address, e.g. $MN_USER_DATA_INT[0]
	index	Identifier for the area index: 0 (default setting): Device 1 to 16 1: Device 17 to 32 2: Device 33 to 48 3: Device 49 to 64

4.6.4 PLC_INTERFACE

Description

This identifier permits the PLC ↔ HMI interface to be redefined. The system expects 128 addressable words.

Note

The identifier is not preset and must be defined by the programmer.

Programming

Identifier:	**PLC_INTERFACE**	
Syntax:	<plc_interface name = "Address identifier of the data" />	
Attributes:	name	Identifier for the address, e.g. "plc/mb170"

Example: `plc/mb170`

4.6.5 POWER_OFF

Description

Identifier for a message prompting the operator to switch the machine off. The message text is permanently saved in the system.

Programming

Identifier:	**POWER_OFF**
Syntax:	<power_off />
Attributes:	--

4.6.6 WAITING

Description

After a reset of the NC or the drive, there is a wait for the restart of the respective component.

Programming

Identifier:	**WAITING**	
Syntax:	<WAITING WAITINGFORNC ="TRUE" />	
Attributes:	waitingfornc="true"	There is a wait for the restart of the NC.
	waitingfordrive="true"	There is a wait for the restart of the drive.

4.6.7 XML identifiers for the dialog

Dialog for the parameterization

A dialog can be configured for each device so that additional parameters can be set or output during runtime. This is displayed by pressing the "Additional parameters" softkey.

All of the script elements described in Chapter Generating user dialogs (Page 25) can be used to generate the dialog.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE AGM>
<AGM>
<DEVICE>
  <NAME> Device 1 </NAME>
  <START_UP>
  …
  </START_UP>
  <SET_ACTIVE>
  …
  </SET_ACTIVE>
…
  <FORM name="<dialog name>">          Identifier for a user dialog
    <INIT>
      <CONTROL name = "edit1" .../>    Identifier for an input field
    </INIT>
    <PAINT>                            Identifier for text or image display
      <TEXT>hello world !</TEXT>
    </PAINT>
  </FORM>

</DEVICE>
…
</AGM>
```

4.6.8 SOFTKEY_OK, SOFTKEY_CANCEL

Description

The identifier SOFTKEY_OK overwrites the standard behavior when closing a dialog by means of the "OK" softkey. The identifier SOFTKEY_CANCEL overwrites the standard behavior when closing a dialog by means of the "CANCEL" softkey.

The following functions can be performed within this identifier:

- Data manipulation
- Conditional processing
- Loop processing

Programming

Identifier:	**SOFTKEY_OK**
Syntax:	<SOFTKEY_OK>
	...
	</SOFTKEY_OK>
Identifier:	**SOFTKEY_CANCEL**
Syntax:	<SOFTKEY_CANCEL>
	...
	</SOFTKEY_CANCEL>

Index