

Concept User Manual

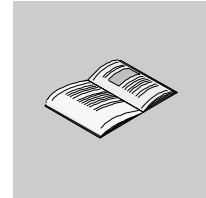
Volume 1

840 USE 503 00 eng Version 2.6 - SR1



© 2002 Schneider Electric All Rights Reserved

Table of Contents



	About the Book	XIII
Chapter 1	General description of Concept	1
1.1	General description of Concept	3
	At a Glance	3
	Introduction	4
	PLC hardware configuration	5
1.2	Programming	6
	At a Glance	6
	General information	7
	Libraries	8
	Editors	9
	Online functions	14
	Communication	14
	Secure Application	15
	Utility program	17
Chapter 2	New Performance Attributes of Concept 2.6 in Comparison with Concept 2.5	21
	New Performance Attributes of Concept 2.6 Compared with Concept 2.5	22
Chapter 3	Project structure	29
	At a Glance	29
	Project Structure and Processing	30
	Programs	35
	Sections	40
	Configuration data	46

Chapter 4	Creating a Project	47
	At a Glance	47
	Overview	48
	Step 1: Launching Concept	49
	Step 2: Configuring the PLC	50
	Step 2.1: Required Configuration	51
	Step 2.2: Optional Configuration	53
	Step 3: Creating the User Program	57
	Step 4: Save	60
	Step 5: Loading and Testing	61
	Step 6: Optimize and Separate	66
	Step 7: Documentation	67
Chapter 5	PLC configuration	69
	At a Glance	69
5.1	General information about hardware configuration	71
	At a Glance	71
	General information	72
	Proceed in the following way with the configuration	73
5.2	Configuration in OFFLINE and ONLINE mode	74
	At a Glance	74
	General information	75
	Available Functions in OFFLINE and ONLINE Modes	76
5.3	Unconditional Configuration	78
	At a Glance	78
	Precondition	79
	PLC selection	80
	CPU Selection for the PLC Type	80
	PLC memory mapping	83
	Loadables	84
	Segment manager	86
	I/O Map	87
5.4	Optional configuration	90
	At a Glance	90
	Settings for ASCII Messages	91
	Making Additional Functions Available in the Configurator	92
	Data Exchange between Nodes on the Modbus Plus Network	93
	Protecting Data in the State RAM before Access	94
	Parameterize interfaces	94
	Special Options	96
5.5	Backplane Expander Config	98
	At a glance	98
	Generals to Backplane Expander	99
	Edit I/O Map	99
	Error handling	100

5.6	Configuration of various network systems.	101
	At a Glance	101
	Configure INTERBUS system	102
	Configure Profibus DP System	103
	Configure Ethernet	104
	RTU extension.	105
	Ethernet I/O Scanner.	106
	How to use the Ethernet / I/O Scanner	109
5.7	Quantum Security Settings in the Configurator	111
	Quantum Security Parameters	112
Chapter 6	Main structure of PLC Memory and optimization of memory	115
	At a Glance	115
6.1	Main structure of the PLC Memory	117
	General structure of the PLC Memory.	117
6.2	General Information on Memory Optimization.	118
	Introduction	118
	Possibilities for Memory Optimization	119
	PLC-Independent	119
6.3	Memory Optimization for Quantum CPU X13 0X and 424 02.	122
	Introduction	122
	General Information on Memory Optimization for Quantum CPU X13 0X and 424 02	123
	Selecting Optimal EXEC File.	125
	Using the Extended Memory (State RAM for 6x references)	129
	Harmonizing the IEC Zone and LL984 Zone.	131
	Harmonizing the Zones for Global Data and IEC Program Memory.	133
6.4	Memory Optimization for Quantum CPU 434 12(A) and 534 14(A)	136
	Introduction	136
	General Information on Memory Optimization for Quantum CPU 434 12(A) and 534 14(A)	137
	Harmonizing IEC Zone and LL984 Zone.	139
	Harmonizing the Zones for Global Data and IEC Program Memory (CPU 434 12(A) / 534 14 (A))	144
6.5	Memory optimization for Compact CPUs	147
	At a Glance	147
	General Information on Memory Optimization for Compact CPUs	148
	Harmonizing IEC Zone and LL984 Zone.	150
	Harmonizing the Zones for Global Data and IEC Program Memory (Compact)	155

6.6	Memory optimization for Momentum CPUs	157
	Introduction	157
	General Information on Memory Optimization for Momentum CPUs	158
	Selecting Optimal IEC Zone	160
	Harmonizing the Zones for Global Data and IEC Program Memory (Momentum)	161
6.7	Memory optimization for Atrium CPUs	163
	At a Glance	163
	General Information on Memory Optimization for Atrium CPUs	164
	Harmonizing IEC Zone and LL984 Zone	166
	Harmonizing the Zones for Global Data and IEC Program Memory (Atrium)	171
Chapter 7	Function Block language FBD	173
	At a Glance	173
7.1	General information about FBD Function Block	175
7.2	FBD Function Block objects	177
	At a Glance	177
	Functions and Function Blocks (FFBs)	178
	Link	182
	Actual parameters	182
	Text Object	184
7.3	Working with the FBD Function Block language	185
	At a Glance	185
	Positioning Functions and Function Blocks	186
	FFB Execution Order	187
	Configuring Loops	189
7.4	Code generation with the FBD Function Block language	190
	Code Generation Options	191
7.5	Online functions of the FBD Function Block language	192
7.6	Creating a program with the FBD Function Block language	195
Chapter 8	Ladder Diagram LD	199
	At a Glance	199
8.1	General information about Ladder Diagram LD	201
8.2	Objects in Ladder Diagram LD	204
	At a Glance	204
	Contacts	205
	Coils	206
	Functions and Function Blocks (FFBs)	209
	Link	214
	Actual Parameters	215
	Text object	217

8.3	Working with the LD Ladder Diagram	218
	At a Glance	218
	Positioning Coils, Contacts, Functions and Function Blocks.	219
	Execution sequence	221
	Configuring Loops	221
8.4	Code generation with LD Ladder Diagram	223
8.5	Online functions with the LD Ladder Diagram.	225
8.6	Creating a program with LD Ladder Diagram.	228
Chapter 9	Sequence language SFC	233
	At a Glance	233
9.1	General information about SFC sequence language	235
9.2	SFC sequence language elements	237
	At a Glance	237
	Step.	238
	Action	240
	Transition.	242
	Transition section	243
	Link	245
	Jump	246
	Alternative Branch.	248
	Alternative connection.	250
	Parallel branch	251
	Parallel connection	252
	Text object.	252
9.3	Working with the SFC Sequence Language	253
	Introduction	253
	General information on editing objects	254
	Declaring step properties	257
	Declaring actions.	259
	Identifier.	262
	Declaring a Transition	264
	Alias Designations for Steps and Transitions	266
9.4	Online functions of the SFC sequence language	269
	At a Glance	269
	Animation	270
	Controlling a Step String	272
	Learn monitoring times	274
	Transition diagnosis	277

Chapter 10	Instruction list IL	279
	At a Glance	279
10.1	General information about the IL instruction list.	281
10.2	Instructions.	283
	At a Glance	283
	General information about instructions	284
	Operands	285
	Modifier	287
	Operators	288
	Tag	291
	Declaration (VAR...END_VAR)	292
	Comment	294
10.3	IL instruction list operators.	295
	At a Glance	295
	Load (LD and LDN)	296
	Store (ST and STN)	297
	Set (S)	298
	Reset (R)	299
	Boolean AND (AND, AND (), ANDN, ANDN ())	300
	Boolean OR (OR, OR (), ORN, ORN ())	302
	Boolean exclusive OR (XOR, XOR (), XORN, XORN ())	304
	Invert (NOT)	305
	Addition (ADD and ADD ())	306
	Subtraction (SUB and SUB ())	307
	Multiplication (MUL and MUL())	308
	Division (DIV and DIV ())	309
	Compare on "Greater Than" (GT and GT ())	310
	Compare to "Greater than/Equal to" (GE and GE ())	311
	Compare to "Equal to"(EQ and EQ ())	312
	Compare to "Not Equal to" (NE and NE ())	313
	Compare to "Less than/Equal to" (LE and LE ())	314
	Compare to "Less Than"(LT and LT ())	315
	Jump to label (JMP, JMPC and JMPCN).	316
	Call Function Block/DFB (CAL, CALC and CALCN)	319
	FUNCNAME.	319
	Right parenthesis ")"	319
10.4	Call up of functions, Function Blocks (EFBs) and Derived Function Blocks (DFBs)	320
	At a Glance	320
	Use of Function Blocks and DFBs.	321
	Invoking a Function Block/DFB	323
	Function call.	327

10.5	Syntax check and Code generation	329
	At a Glance	329
	Syntax Check	330
	Code generation	332
10.6	Online functions of the IL instruction list	333
	At a Glance	333
	Animation	334
	Monitoring field	337
10.7	Creating a program with the IL instruction list	338
Chapter 11	Structured text ST	341
	At a Glance	341
11.1	General information about structured Text ST	343
11.2	Expressions	345
	At a Glance	345
	Operands	346
	Operators	347
11.3	Operators of the programming language of structured ST text	350
	At a Glance	350
	Use of parentheses "(")	351
	FUNCNAME	351
	Exponentiation (**)	351
	Negation (-)	352
	Complement formation (NOT)	352
	Multiplication (*)	352
	Division (/)	353
	Modulo (MOD)	353
	Addition (+)	353
	Subtraction (-)	354
	Comparison on "Greater Than" (>)	354
	Comparison on "Greater than/Equal to" (>=)	354
	Comparison with "Equal to" (=)	354
	Comparison with "Not Equal to" (<>)	355
	Comparison with "Less Than"(<)	355
	Comparison with "Less than or Equal to" (<=)	355
	Boolean AND (AND or &)	356
	Boolean OR (OR)	356
	Boolean Exclusive OR (XOR)	356
11.4	Assign instructions	357
	At a Glance	357
	Instructions	358
	Assignment	359
	Declaration (VAR...END_VAR)	360
	IF...THEN...END_IF	362
	ELSE	363

	ELSIF...THEN	364
	CASE...OF...END_CASE	365
	FOR...TO...BY...DO...END_FOR	366
	WHILE...DO...END_WHILE	368
	REPEAT...UNTIL...END_REPEAT	370
	EXIT	371
	Empty instruction	371
	Comment	371
11.5	Call up of functions, Function Blocks (EFBs) and Derived Function Blocks (DFBs)	372
	At a Glance	372
	Function Block/DFB Invocation	373
	Function Invocation	377
11.6	Syntax check and code generation	379
	At a Glance	379
	Syntax Check	380
	Code generation	381
11.7	Online functions of the ST programming language	382
11.8	Creating a program with the structured ST text	384

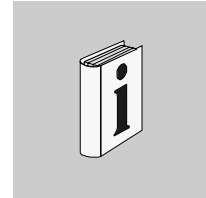
Index **i**

The chapters marked gray are not included in this volume.

Chapter 12	Ladder Logic 984	387
Chapter 13	DFBs (Derived Function Blocks)	415
Chapter 14	Macros	455
Chapter 15	Variables editor	479
Chapter 16	Project Browser	491
Chapter 17	Derived data types	499
Chapter 18	Reference data editor	531
Chapter 19	ASCII Message Editor	543
Chapter 20	Online functions	561
Chapter 21	Import/Export	619
Chapter 22	Documentation and Archiving	661

Chapter 23	Simulating a PLC	677
Chapter 24	Concept Security	691
Appendices	705
Appendix A	Tables of PLC-dependent Performance Attributes	707
Appendix B	Windows interface	729
Appendix C	List of symbols and short cut keys.....	751
Appendix D	IEC conformity	779
Appendix E	Configuration examples.....	805
Appendix F	Convert Projects/DFBs/Macros.....	911
Appendix G	Concept ModConnect.....	915
Appendix H	Conversion of Modsoft Programs	923
Appendix I	Modsoft and 984 References.....	929
Appendix J	Presettings when using Modbus Plus for startup.....	933
Appendix K	Presettings when using Modbus for startup	947
Appendix L	Startup when using Modbus with the EXECLoader	953
Appendix M	Startup when using Modbus with DOS Loader	973
Appendix N	Startup when using Modbus Plus with the EXECLoader ...	987
Appendix O	Startup when using Modbus Plus with DOS Loader	1007
Appendix P	EXEC files	1023
Appendix Q	INI Files	1027
Appendix R	Interrupt Processing.....	1041
Appendix S	Automatic Connection to the PLC	1067
Glossary	1077

About the Book



At a Glance

Document Scope This user manual is intended to help you create a user program with Concept. It provides authoritative information on the individual program languages and on hardware configuration.

Validity Note The documentation applies to Concept 2.6 for Microsoft Windows 98, Microsoft Windows 2000, Microsoft Windows XP and Microsoft Windows NT 4.x.

Note: Additional up-to-date tips can be found in the Concept README file.

Related Documents

Title of Documentation	Reference Number
Concept Installation Instructions	840 USE 502 00
Concept IEC Block Library	840 USE 504 00
Concept EFB User Manual	840 USE 505 00
Concept LL984 Block Library	840 USE 506 00

User Comments We welcome your comments about this document. You can reach us by e-mail at TECHCOMM@modicon.com

About the Book

General description of Concept



1

At a Glance

Overview

This chapter contains a general description of Concept. It should provide an initial overview of Concept and its helper programs.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
1.1	General description of Concept	3
1.2	Programming	6

General description of Concept

1.1 General description of Concept

At a Glance

Overview

This section describes the performance features of Concept and provides an overview of the hardware that may be programmed using Concept.

What's in this Section?

This section contains the following topics:

Topic	Page
Introduction	4
PLC hardware configuration	5

Introduction

Operating System

Nowadays, a graphical user interface is a requirement for tasks of this kind. For this reason, Concept has been established as an MS Windows application. Concept can be operated in Windows 98, Windows 2000, Windows XP and Windows NT. These operating systems have the advantage that they are used all over the world. Therefore PC users have a basic knowledge of Windows technology and mouse operation. In addition to this all common monitors, graphic cards and printers can be used with MS Windows. As a user, you are not therefore tied to specific hardware configurations.

International Standard IEC 1131-3

For effective system configuration Concept offers a unified configuration environment in accordance with international standard regulations IEC 1131-3.

PLC Independence when Programming

The guiding principle behind the development of Concept was that all the system configuration procedures and all the editors should have the same look and feel. Most of the configuration steps, especially program creation, are designed independently of the PLC to be programmed.

Graphical Interface

The entire program is divided up into sections corresponding to the logic structure. The Concept configuration tool enables objects (such as function blocks, steps, and transitions) to be selected, placed and moved easily in graphical form. Plausibility tests already take place in the SFC editor (Sequential Function Chart/ sequence language) during object placing, as most of the links between objects are generated automatically during placing. In the FBD editor (Function Block Diagram/Function Block language) and LD editor (Ladder Diagram), plausibility tests take place when blocks are linked. Unauthorized links, such as those between different data types have already been rejected during configuration. A plausibility test also takes place in the LL984 editor (Ladder Logic 984) during placing. In the IL editor (Instruction List) and ST editor (Structured Text) unauthorized instructions are identified via a colored outline. After the first successful program run, the program may be optimized in graphic terms by moving links, blocks or texts to improve the display.

Print

If desired the sections may be displayed with print preview information, in order to individually control pages of documentation. Signals receive an expansive designation with symbol names and comments. Unique notes on signal tracking are provided at the signal breaks. The individual block processing sequences from one section may be displayed and documented in the FBD editor.

Import/Export Functions	<p>Sections from various projects can be combined as desired in another project using import/export functions.</p> <p>It is also possible to convert the sections of one IEC programmer language into sections of another IEC programmer language.</p> <p>Variables may be imported into and exported from the text using text delimited or FactoryLink format.</p>
Runtime System	<p>The runtime system on the PLC offers quick reactions to signal state process changes (short cycle time), Simulating signal transmitters (See <i>Simulating a PLC</i>, p. 677), Online display (See <i>Online functions</i>, p. 561), online parameter changes and online program changes.</p>
Open Software Architecture	<p>Concept possesses open software architecture to enable connection to external systems (e.g. for visualization) via standard interfaces.</p>
Online Help	<p>Special care was taken when developing the help function. The context sensitive Online help function (See <i>How the Online Help is set out</i>, p. 748) provides support for every configuration situation just by clicking on the subject using the mouse or pressing the F1 key. Menu commands and dialogs are also context sensitive, as are, function blocks and hardware components of the individual PLC families.</p>

PLC hardware configuration

Description	<p>Concept is the unified projection tool for Quantum, Compact, Momentum and Atrium products.</p> <p>Hardware components (for example CPU, program memory, input/output units etc.) can be specified before, during or after program creation.</p> <p>This projection task can be performed both online (linked to the PLC) and locally (PC alone). Projection is supported by Concept, and only suggests valid combinations. Misprojection is therefore prevented. In online mode the projected hardware is tested for plausibility immediately and input errors are rejected.</p> <p>After linking the programmer device (PC) to the PLC, a plausibility test is performed on the projected values (e.g. from the Variable Editor) using the actual hardware resources and if necessary an error message will appear.</p>
--------------------	---

1.2 Programming

At a Glance

Overview This section provides an overview of the editors which are available in Concept.

What's in this Section? This section contains the following topics:

Topic	Page
General information	7
Libraries	8
Editors	9
Online functions	14
Communication	14
Secure Application	15
Utility program	17

General information

At a Glance

As a solution for automatic control engineering tasks, Concept provides the following IEC 1131-3 compatible programming languages:

- Function Block language FBD (Function Block Diagram) (See *FBD editor*, p. 10),
- LD (Ladder Diagram) (See *LD editor*, p. 10),
- Sequential language SFC (Sequential Function Chart) (See *SFC editor*, p. 11),
- Instruction List IL (See *IL editor*, p. 11) and
- Structured Text ST (See *ST editor*, p. 12).

The Modsoft orientated language is also available

- Ladder Diagram LL984 (Ladder Logic) (See *LL984 editor*, p. 12).

The IEC programming language (FBD, LD, SFC, ST and IL) basic elements are Functions and Function Blocks, which make up assembled logic units. Concept contains various Block libraries (See *Libraries*, p. 8) with predefined elementary functions/Function Blocks (EFBs). In order to locate the individual EFBs without difficulty, they are split into different groups according to their area of use.

For the Modsoft orientated programming language LL984, there is a Block library (See *Libraries*, p. 8) with Instructions available.

Sections

The control program is constructed from sections according to the logic structure. Only one programming language is used within a section.

Merging these sections makes up the entire control program and the automation device uses this to control the process. Any IEC sections (FBD, LD, SFC, IL, ST) may be mixed within the program. The LL984 sections are always edited as a block before the IEC sections.

Data types

A subset of Data types from the international standard IEC1131-3 is available.

In the Data type editor (See *Data type editor (DDT editor)*, p. 13) intrinsic data types can be derived from IEC data types.

Using variables

Variables for linking basic elements (objects) within a section are not usually necessary with the graphic programming languages FBD, LD, SFC and LL984, as these links are usually made graphically. (An additional link using variables is only necessary for incredibly complex sections.) Graphic links are managed by the system and therefore no projection requirement is created. The Variable Editor (See *Variable Editor*, p. 13) is used to project all other variables such as those for data transfer between various sections.

Libraries

At a Glance

For program creation Concept provides various block libraries with predefined Functions and Function Blocks.

There are 2 different types of block libraries:

- **IEC library**
Block libraries for sections in the IEC programming languages (FBD, LD, SFC, IL and ST)
 - **LL984 Library**
Block library for sections in the Modsoft orientated programming language LL984
-

IEC library

The following IEC libraries are available for applications:

- **AKFEFB**
This library contains the AKF/ALD EFBs, which are not covered by the IEC library.
- **ANA_IO**
This library is for analog value processing.
- **COMM**
This library is used for exchanging data between a PLC and another Modbus, Modbus Plus or Ethernet node.
- **CONT_CTL**
This library is for projecting process-engineering servoloops. It contains controller, differential, integral, and polygon graph EFBs.
- **DIAGNOSTICS**
This library is used to investigate the control program for misbehaviors. It contains action diagnostics, Reaction diagnostics, locking diagnostics, process prerequisite diagnostics, dynamic diagnostics and signal group monitoring EFBs.
- **EXPERTS**
This library contains EFBs, which are necessary for using expert modules.
- **EXTENDED**
This library contains useful supplements for different libraries. It has EFBs for creating average values, selecting maximum values, negating, triggering, converting, creating a polygon with 1st degree interpolation, edge recognizing, and for specifying an insensitive zone for control variables.
- **FUZZY**
This library contains EFBs for fuzzy logic.
- **IEC**
This library contains the EFBs defined in IEC 1131-3. It has for example EFBs for mathematical calculations, counters, timers etc.
- **LIB984**
This library contains IEC 1131 compatible EFBs from the LL984 library, for example, EFBs for register transfer.

- **SYSTEM**

This library contains EFBs for using system functions. It has EFBs for cycle time recognition, for various system cycle use, for SFC section control and for system status display.

LL984 Library

The LL984 library contains the LL984 editor instructions (blocks). It contains instructions for mathematical calculations, counters, timers, instructions for displaying system status, controller, differential and integral instructions and instructions for exchanging data between a PLC and another Modbus or Modbus Plus node.

Editors

At a Glance

When generating a section specify which programming language you are going to use.

The following editors are available for creating sections in the various programming languages:

- FBD editor (Function Block Language) (See *FBD editor*, p. 10)
- LD editor (Ladder Diagram) (See *LD editor*, p. 10)
- SFC editor (Sequence language) (See *SFC editor*, p. 11)
- IL editor (Instruction List) (See *IL editor*, p. 11)
- ST editor (Structured Text) (See *ST editor*, p. 12)
- LL984 editor (Modsoft orientated Ladder Logic) (See *LL984 editor*, p. 12)

The following editors are available for declaring variables, creating data types and displaying variables.

- the Variable Editor (for declaring variables), (See *Variable Editor*, p. 13)
- the reference data editor (for displaying and online changing of values) (See *Reference data editor*, p. 13) and
- the data type editor (for creating user specific data types) (See *Data type editor (DDT editor)*, p. 13).

The following editors are available for creating user specific functions and Function Blocks:

- Concept DFB (for creating Derived Function Blocks and macros) (See *Concept DFB*, p. 17)
 - Concept EFB (for creating user specific elementary functions and Function Blocks) (See *Concept EFB*, p. 18)
-

FBD editor

The FBD editor (See *Function Block language FBD*, p. 173) is used for graphic function plan programming according to IEC 1131-3.

Elementary functions, Elementary Function Blocks (EFBs) and Derived Function Blocks (DFBs) are connected with signals (variables) onto FBD sections for the function plan. The size of a FBD section is 23 lines and 30 columns.

EFBs are equipped with a fixed or variable number of input variables and may be placed anywhere on the section. Variables and EFBs may have comments separately added to them, column layouts on a section may be commented on anywhere using text boxes. All EFBs may be performed conditionally or unconditionally.

All the EFBs are divided into function- and use-orientated libraries in various groups, to make them easier to locate.

LD editor

The LD editor (See *Ladder Diagram LD*, p. 199) is used for graphic ladder programming according to IEC 1131-3.

Contacts and coils are connected to the Ladder Diagram in LD sections using signals (variables).

The size of a FBD section is 23 lines and 30 columns.

Furthermore, the elementary functions and Function Blocks (EFBs), which are named in the FBD editor, the Derived Function Blocks (DFBs) and User Defined Function Blocks (UDFBs) may also be bound in the ladder diagram (see *FBD editor*, p. 10).

The structure of a LD section corresponds to a rung for relay switching. The left power rail is located on its left-hand side. This left power rail corresponds to the phase (L ladder) of a rung. With LD programming, in the same way as in a rung, only the LD objects (contacts, coils) which are linked to a power supply, that is to say connected with the left power rail, are "processed". The right power rail, which corresponds to the neutral ladder, is not shown optically. However, all coils and EFB outputs are linked with it internally and this creates a power flow.

SFC editor

The SFC editor (See *Sequence language SFC*, p. 233) is used to graphically program an IEC 1131-3 compatible sequential control.

The SFC elements are connected in a SFC section to one of the sequential controls corresponding to the task setting. The size of a SFC section is 32 lines and 200 lines.

The following sequential control programming objects are available in Concept.

- Step (including actions and action sections)
- Transition (including transition section)
- Alternative branch and merge
- Parallel branch and merge
- Jump
- Connection

Simple diagnostics monitoring functions are already integrated in the steps.

IL editor

The IL editor (See *Instruction list IL*, p. 279) is used for programming IEC 1131-3 compatible instruction lists.

Existing IL instructions, elementary functions and Elementary Function Blocks (EFBs), and Derived Function Blocks (DFBs) are written in series in text form in IL sections from operators (commands) and operands (signals, variables).

When the program is entered, all the standard Windows services and some additional commands for text-processing are available. The size of an IL section is 64 Kbyte maximum.

The following instruction list programming operators are available in Concept:

- Logic (AND, OR etc.)
- Arithmetic (ADD, SUB, MUL, DIV, ...)
- Comparative (EQ, GT, LT, ...)
- Jumps (JMP, ... conditional/unconditional)
- EFB call (CAL , ... conditional/unconditional)

IL programming is done in text form. When text is entered, all the standard Windows services for text-processing are available. The IL editor also contains some further commands for text-processing.

A spell check is performed immediately after text has been entered (instructions, key words, separators), highlighting errors with a colored outline.

ST editor

The ST editor (See *Structured text ST*, p. 341) is used for programming IEC 1131-3 structured text.

Existing ST statements, elementary functions and Elementary Function Blocks (EFBs), and Derived Function Blocks (DFBs) are written in text form in IL sections by printing (operator lists) and operands (signals, variables).

When the program is entered, all the standard Windows services and some additional commands for text-processing are available. The size of a ST section is 64 Kbyte maximum.

The following structured text programming statements and operators are available in Concept:

- conditional/unconditional statement execution (IF, ELSIF, ELSE, ...)
- conditional/unconditional loop execution (WHILE, REPEAT)
- Mathematical, comparative, and logic operators
- conditional/unconditional EFB call

ST programming is done in text form. When text is entered, all the standard Windows services for text-processing are available. The ST editor also contains some further commands for text-processing.

A spell check is performed immediately after text has been entered (instructions, key words, separators), highlighting errors with a colored outline.

LL984 editor

Using the Modsoft orientated LL984-Editor (See *Ladder Logic 984*, p. 387) (Ladder Diagram 984), instructions, contacts, coils and signals (variables) are connected to a ladder diagram. Instructions, contacts, coils and variables may be commented on.

The structure of a LL984 section corresponds to a rung for relay switching. The left power rail is located on its left-hand side, but it is not visually displayed. This left power rail corresponds to the phase (L ladder) of a rung. With LL984 programming, in the same way as in a rung, only the LL984 objects (instructions, contacts, coils) connected to a power supply, i.e. connected to the left power rail, are "processed". The right power rail, which corresponds to the neutral ladder is not visually displayed either. However, all coils and instruction outputs are linked with it internally and this creates a power flow.

Concept has various predefined instructions for ladder programming using LL984. These may be found in the block library LL984. Additional instructions for special applications are available as loadables and may be loaded at a later time.

Variable Editor The Variable Editor (See *Variables editor, p. 479*) is used to declare and comment on all necessary symbolic signal names (variables). Only declared variables may be used in Concept programs.

A data type must be assigned to each symbolic signal name! If this variable is assigned a reference address, a Located variable (without reference address = Unlocated variable) is received. An initial value may also be provided for each variable, which will be transferred into the PLC during the first load.

Data type editor (DDT editor) The Data type editor (See *Derived data types, p. 499*) may be used to define specific Derived Data Types (Derived Data Type = DDT).

Derived Data Types combine several Elementary data types (BOOL, WORD, ...) in one data record. It is not only the same data types which may be combined as ARRAY, but also various data types may be combined as STRUCT. In Concept, a number of Derived Data Types are already available, which for instance may be used for DFBs.

DDTs appear in DFBs or EFBs only as a connection, i.e. for instance in FBD a variable input is only necessary in the block. It is thus recommended that frequently recurring groups of elementary data types (and also DDTs) be defined as DDTs, in order to improve accessibility of an application.

The definition appears in text form, and all the standard Windows services and some additional commands for text-processing are available. The size of a data type file is 64 Kbyte maximum.

Reference data editor The Reference data editor (See *Reference data editor, p. 531*) may be used in online mode to display the variable value, to force variables and to set variables. There is also the possibility of separating variables from the process. Inputs may be saved in a data file and be reused.

Online functions

Available online functions

After the programming device has been linked to the PLC, a range of online Startup and maintenance functions become available.

- the program on the programming device is compared with the program on the PLC
 - the PLC can be started and stopped
 - Object information is displayed
 - Programs can be loaded, sections can be changed online and loaded
 - Variable values can be entered online
 - Animation mode shows the program with its current signal states
-

Operating and monitoring

Declaration of special operating and monitoring variables is not necessary in Concept. The variables to be visualized can be identified as such in the Variable Editor and then be exported into a ModLink or FactoryLink configuration data file. This data file can be used for visualizing.

Communication

Description

Communication between the PLC and another Modbus-, Modbus Plus-, SY/MAX-Ethernet or TCIP/IP Ethernet node is projected using IEC languages (FBD, LD, SFC, ST, IL) with the EFBs from the block library COMM. The instruction MSTR from the block library may be use

A peer to peer transfer of register contents is possible using the peer cop, independent of these blocks/instructions.

Communication is projected between the PLC and the decentralized I/O via the INTERBUS by simply entering the NOA module in the component list and loading a loadable (ULEX).

Communication is projected between the programming device and a PLC via Ethernet by simply entering and parametering the appropriate couple module in the component list.

Secure Application

At a Glance

In several areas of industry, the need for security demands regulated access to PLCs, recording program changes and archiving those recordings. Following a standardized procedure ensure that records may not be falsified. To enable these requirements, new features have been implemented in Concept that ensure secure application. To guarantee that all of these parameters are defined, the user can activate the **Secure Application** check box in the **Project** → **Project Properties** dialog. Concept will then ensure that all of these parameters are set and that their contents remain valid. The project is then indicated as being a secure application, and this information is included in the information that is downloaded to the PLC.

Secure Application

The secure application is defined in the **Project** → **Project Properties** dialog by activating the **Secure Application** check box. These settings are then exported, imported, read and loaded to the PLC.

Note: When the secure application is activated, a NOT EQUAL status is generated and required reloading to the PLC. Unchecking the check box also creates a NOT EQUAL status so that loading is again required as well. If Concept is connected to a PLC that is already defined with the "Secure Application" setting, the setting is automatically accepted in Concept in case of upload the controller.

The log file is stored in the Concept directory and has the name of the current date (YEARMONTHDAY.ENC, e.g. 20020723.ENC). The path of the log file can be defined in dialog **Common Preferences**. If no path is defined then Concept uses the default log path (Concept directory, e.g. C:\CONCEPT).

Among other things, logging write-access to the PLC can record the following data:

- Section name
 - EFB/DFB Instance name, FB Type name
 - Pin Name
 - [Variable name] [Literal] [Address]
 - Old value
 - New value
 - User name (if the Concept (Login) password is activated in Concept Security)
 - Data and Time (see also *Address format in LOG file [Logging], p. 1036*)
-

Requirements

The secure application can only be activated if the following prerequisites are met:

- can only be used with 140 CPU 434 12A or 140 CPU 534 14A
 - at least one IEC section (if no IEC section exists then the download is aborted.)
 - Offline mode (**Online** → **Disconnect...**)
 - Supervisor Rights (see Concept under **Help** → **About...** → **Current User:**)
-

**Activation
Combination for
Secure
Application**

Various Activation Combinations for Secure Application:

"Secure Application" activated in Concept	"Secure Application" loaded to PLC	Reaction to connection with the PLC
Not activated	Not activated	Normal operation without secure application
Not activated	Activated	When uploading, the Secure Application check box is activated in Concept and encrypted logging is activated.
Activated	Not activated	Download required because the status is NOT EQUAL.
Activated	Activated	Normal operation with secure application (e.g. encrypted logging).

**Reading the
Encrypted Log
File**

To read the encrypted log file, the View tool is opened automatically in the **View Logfile** dialog.

Note: If an encrypted log file has been improperly modified in any way, the log is decoded as much as is possible, and the lines that have been modified will remain unreadable. The first line will contain the message: "This log file has been modified".

Utility program

At a Glance

In addition to Concept the following range of utility programs are available:

- Concept DFB
 - Concept EFB
 - Concept SIM (16 bit)
 - Concept PLCSIM32 (32 bit)
 - Concept Security
 - Concept WinLoader
 - Concept Converter
 - Concept ModConnect
-

Concept DFB

Concept DFB is used to create DFBs (Derived Function Blocks) (See *DFBs (Derived Function Blocks)*, p. 415) and Macros (See *Macros*, p. 455).

DFBs (Derived Function Blocks)

DFBs can be used for setting both the structure and the hierarchy of a program. In programming terms, a DFB represents a subroutine.

DFBs can be created in the programming languages FBD, LD, IL, and ST. In Concept, DFBs can be called up in any programming language, regardless of the programming language they were created in. One or several existing DFBs can be called up within one DFB, with the called-up DFBs themselves able to call up one or several DFBs.

Macros

Macros are used to duplicate frequently used sections and networks (including their logic, variables and variable declaration).

Macros have the following properties:

- Macros can only be created in the programming language FBD.
- Macros only contain one section.
- Macros can contain a section of any complexity.
- In programming terms, there is no difference between an instanced macro, i.e. a macro inserted into a section and a conventionally created section.
- It is possible to call up DFBs in a macro.
- It is possible to declare macro-specific variables for the macro.
- It is possible to use data structures specific to the macro
- Automatic transfer of the variables declared in the macro.
- Initial values are possible for the macro variables.
- It is possible to instance a macro many times in the entire program with different variables.
- Section names, variable names and data structure names can contain the character ~ as an exchange marking.

Concept EFB

The optional tool Concept EFB can be used to generate, in C++ programming language, your own application specific Functions and Function Blocks (EFBs) and to integrate them in the form of libraries with groups in your version of Concept.

The operating rules for these user-defined blocks (UDFBs) are identical to those for standard EFBs.

It is, for instance, recommended that complex program parts with a high number of calls and program parts, whose solution is to remain hidden from the user, e.g. special technology objects etc. be generated using Concept EFB.

Note: Concept EFB is not included as part of the Concept package and may be ordered in addition.

**Concept SIM
(16 bit)**

The 16 bit simulator Concept SIM (See *Simulating a PLC (16-bit simulator)*, p. 679) is available for simulating a PLC, i.e. to test your user program online without hardware. Concept SIM simulates a coupled PLC via Modbus Plus.

Note: The simulator is only available for the IEC languages (FBD, SFC, LD, IL and ST).

**Concept PLCSIM
(32 bit)**

The 32 bit simulator Concept PLCSIM32 (See *Simulating a PLC (32-bit simulator)*, p. 682) is available for simulating a PLC, i.e. to test your user program online without hardware. Concept PLCSIM32 simulates a PLC coupled via TCP/IP, where the signal states of the I/O modules can also be simulated. Up to 5 programming devices can be coupled to the simulated PLC at the same time.

Note: The simulator is only available for the IEC languages (FBD, SFC, LD, IL and ST).

Concept Security

Concept Security (See *Concept Security*, p. 691) can be used to assign access. Access signifies that the function of Concept and its utility programs is limited depending on the user.

The access defined for one user is applicable to all Concept installation projects. A maximum of 128 users may be defined.

Concept Converter Projects, DFBs, macros, and data structures (Derived Data Types), created for an earlier version of Concept, can be converted without hassle to work in the current version of concept in the Concept Converter (See *Convert Projects/DFBs/Macros*, p. 911).

Concept EXECLoader The Concept EXECLoader can be used to load Exec data files onto the PLC.

Concept ModConnect Concept-ModConnect (See *Concept ModConnect*, p. 915) can be used to extend the configurator for new (specific) I/O modules.

General description of Concept

**New Performance Attributes of
Concept 2.6 in Comparison with
Concept 2.5**



2

New Performance Attributes of Concept 2.6 Compared with Concept 2.5

Highlights

New general performance attributes:

- **Interrupt sections**
 - **Global variables**
 - **Security features**
-

New EFBs

New EFBs in the SYSTEM library:

New EFBs	Description
I_LOCK	Disable all interrupt sections
I_UNLOCK	Enable all interrupt sections
I_MOVE	Interrupt protected assignment
ISECT_OFF	Disable specific interrupt sections
ISECT_ON	Unlock a specific interrupt section
ISECT_STAT	Interrupt section status
PRJ_VERS	States project name and version
GET_IEC_INF	Read IEC status flags
RES_IEC_INF	Reset IEC status flags

New EFBs in the COMM library:

New EFBs	Description
PORTSTAT	States Modbus Port status

Start Concept

New features when starting Concept:

New performance attributes	Description
Automatic connection to every desired PLC	Startup using the Concept Project Symbol creates automatic connection to any desired PLC. This connection is defined by the Command line parameter (See <i>Automatic Connection with Command Line Parameters (Modbus, Modbus +, TCP/IP)</i> , p. 1068).
When starting Concept using the CCLaunch tool, a connection is made to every desired PLC	In large networks, a topology file is created and is then used in the CCLaunch tool. You can use this to create a complete MB+ Routing path (See <i>Automatic Connection with the CCLaunch Tool (Modbus Plus)</i> , p. 1071), which then creates a connection to the PLC automatically.
Displays list of previously opened Projects/DFBs	When starting Concept a list of previously opened Projects/DFBs (max. 4) is displayed in the File main menu.
Archive content display	When unpacking an archived project, all archived files are shown first.

Animation

12 different color schemes for animation in the FBD, IL, ST, SFC and LD editors:

New performance attributes	Description
CONCEPT.INI: [Colors] AnimationColors= (0-12)	Defines the color scheme for online animation in all editors.

Reference data editor

New feature in the reference data editor:

New performance attributes	Description
Address format IEC (QW0000X)	The IEC (QW0000X) address format can be displayed.

Online functions New online features:

New performance attributes	Description
Quantum password protection	Quantum PLC is write protected by entering a password.
Event sections	Online diagnostics are displayed for Interrupt sections.
Event viewer	Error descriptions can be defined in a project specific INI file (See <i>INI Settings for the Event Viewer [Online Events]</i> , p. 1039) that should appear in the event viewer (Online → Online events...).

Message window New performance attributes in the Windows menu:

New performance attributes	Description
Save messages	After messages are displayed they can be saved to file using the Save Messages... (main menu Window) menu command.

New CPU New CPU:

PLC family	Description
Atrium	CPU 180-CCO-241-11

New Module New Quantum module:

Module	Description
140-NOE-771-01	Ethernet module without Hot Standby features.
140-NOE-771-11	Ethernet module (Factory Cast) without Hot Standby features.
140-CPS-114-20	Power supply module
140-CPS-124-20	Power supply module
140-NOG-111-00	1/SFB Master module
140-NWM-100 00	Ethernet module (Factory Cast HMI)

New Momentum module:

Module	Description
170-ANR-120-91	Analog/Digital Input/Output module

Project Browser New features in the Project browser:

New performance attributes	Description
Display interrupt sections	When I/O event sections and Timer event sections are used, they are displayed in the Project browser structure.
Show detailed view	The Project browser window is split vertically, and a second window displays the substructure (e.g. DFBs, Transitions sections, etc.) of the selected elements in a structure tree.

Analyze section New features when analyzing sections:

New performance attributes	Description
Analyze interrupt sections	There is now an additional analysis for Interrupt sections.
Analyzing global variables in DFBs	There is an analysis for global variables in DFBs.

DFB New features for DFB programming:

New performance attributes	Description
Located variables	Located variables are permitted in DFBs when the option in the IEC Extensions dialog box is enabled. Global variables can be created throughout the program with located variables in DFBs.

Data types New features for DFB programming:

New performance attributes	Description
View comments for data structure elements	Comments for data type components defined in data type files (*.ddt, *.dty) are displayed in: <ul style="list-style-type: none"> ● Editors status line ● Variables editor for the definition of initial values ● Inspect Animation field
<i>Extended Data Type Definition (larger than 64 Kbytes), p. 507</i>	The 64 kb restriction is not imposed for local data type definition with the introduction of unlocated Include files.

Configuration

New features in the Configurator:

New performance attributes	Description
1/SFB Coupler configuration	Required to provide support for the A500/A350 I/O module. Extended I/O range up to 160 input/output words.
Quantum security parameter	The following parameters can be defined in the new dialog box (submenu of the Config. Extensions): <ul style="list-style-type: none"> ● Secure data area ● Network write restrictions ● Enable the Auto-Logout option
Interbus configuration with Atrium	The Interbus configuration is done with Atrium CPUs 180 CCO 241 01 (= 1 INTERBUS) and 180 CCO 241 11 (= 2 INTERBUS).

Logging (*.LOG, *.ENC)

New features for DFB logging:

New performance attributes	Description
Additional contents	When logging PLC write access, modifications made to variable and literal values are displayed in addition.
New Date/Time format	By activating the check box Universal Date Format in dialog Common Preferences (setting also affects the CONCEPT.INI file) the format can be changed. The month is then stated within Concept with 3 characters and in English. Example: 24-Dec-2002 14:46:24
Encrypting the log	By activating the check box Encrypt Logfile in dialog Common Preferences (or indirectly using the check box Secure Application in dialog Project Properties) login the write access to the PLC will be encrypted. The encrypted file contains the file extension *.ENC.

Secure Application

New features for a secured application:

New performance attributes	Description
Application backup	If you activate the check box in the Project → Project Properties dialog box, program modifications are automatically logged and encrypted in a *.ENC file. These settings can be loaded using Export/Import and transferred to the PLC.

New Tools

New Tools for Concept:

New Tool	Description
CCLaunch	This tool is used for making an automatic connection (See <i>Automatic Connection with the CCLaunch Tool (Modbus Plus)</i> , p. 1071) with a PLC in a large network.
View Tool	This tool allows you to view encoded LOG files (*.ENC). It is started automatically with menu instruction View Logfile if log encrypting has been activated.

Project structure

3

At a Glance

Overview

This chapter describes the structure of projects in Concept.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Project Structure and Processing	30
Programs	35
Sections	40
Configuration data	46

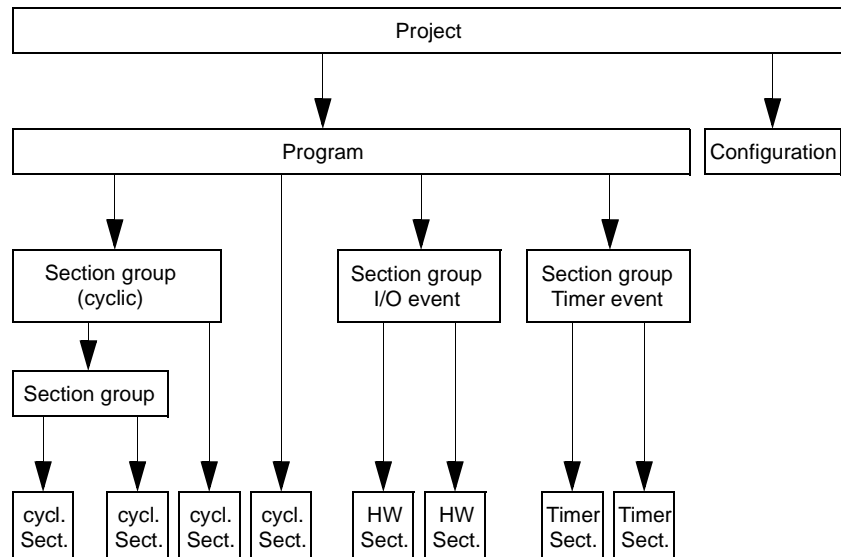
Project Structure and Processing

Structure of a project

The creation of a PLC program with Concept is carried out hierarchically in a project using PLC configuration (See *Configuration data*, p. 46) and Program (See *Programs*, p. 35). The program is divided into section groups and Sections (See *Sections*, p. 40).

The PLC configuration and required program parts can be created in any order within a project (top down or bottom up).

Structure of a project:



Processing an IEC/LL984 project

This table describes the processing of a LL984/IEC project (Quantum):

Step	Logic processor	I/O processor
1	Overhead, e.g. communication with NOM, NOE etc.	-
2	Executing LL984 segment 1	Writing outputs calculated in segment n
		Reading inputs required in segment 2
3	Executing LL984 segment 2	Writing outputs calculated in segment 1
		Reading inputs required in segment 3
4	Executing LL984 segment 3	Writing outputs calculated in segment 2
		Reading inputs required in segment 4
...
n	Executing LL984 segment n (n =< 32)	Writing outputs calculated in segment n-1
		Reading inputs required in segment 1
n+1	Executing IEC section 1	-
n+2	Executing IEC section 2	-
n+3	Executing IEC section 3	-
	..	-
m	Executing IEC section n (n =< 1600) and back to stage 1	-

- 1** The overhead is executed in this stage (e.g. communication with the coupling modules NOM, NOE).
- 2 - 4** In these stages, the logic for the LL984 sections is executed by the logic processor in segments 1-3 (corresponding to the settings in the Segment scheduler (See *Segment manager*, p. 86)).
At the same time the I/O processor transfers the output values calculated in the respective previous segment to the hardware and the hardware reads the input values required for the next respective segment.
- n** In this step, the logic processor in segment n runs the LL984 sections logic.
At the same time the I/O processor transfers the output values calculated in the previous segment to the hardware and the hardware reads the input values required for segment 1.
Note: The output values calculated in this segment are only executed on next execution of stage 2, i.e. after the IEC logic and the overhead have been processed. Therefore no time critical logic should be executed in this segment.
- n+1 - m** The logic processor runs the IEC sections logic in these steps.
It then "jumps back" to stage 1.

Note: No hardware signals are read or written. The values calculated/read in stages 2 to n are used exclusively. The outputs calculated in these stages are transferred in stages 2 to n (corresponding to the settings in the segment scheduler).

Processing a LL984 project

This table describes the processing of a LL984 project (Quantum):

Step	Logic processor	I/O processor
1	Overhead, e.g. communication with NOM, NOE etc.	-
2	Executing LL984 segment 1	Writing outputs calculated in segment n
		Reading inputs required in segment 2
3	Executing LL984 segment 2	Writing outputs calculated in segment 1
		Reading inputs required in segment 3
4	Executing LL984 segment 3	Writing outputs calculated in segment 2
		Reading inputs required in segment 4
...
n	Executing LL984 segment n (n =< 32) and back to stage 1	Writing outputs calculated in segment n-1
		Reading inputs required in segment 1

- 1** The overhead is executed in this stage (e.g. communication with the coupling modules NOM, NOE).
- 2 - 4** In these stages, the logic for the LL984 sections is executed by the logic processor in segments 1-3 (corresponding to the settings in the Segment scheduler (See *Segment manager*, p. 86)).
At the same time the I/O processor transfers the output values calculated in the respective previous segment to the hardware and the hardware reads the input values required for the next respective segment.
- n** In this step, the logic processor in segment n runs the LL984 sections logic.
At the same time the I/O processor transfers the output values calculated in the previous segment to the hardware and the hardware reads the input values required for segment 1.
It then "jumps back" to stage 1.
Note: The output values calculated in this segment are only processed the next time stage 2 is completed, i.e. after the overhead has been processed. Therefore no time critical logic should be executed in this segment.

Processing an IEC project

This table describes the processing of an IEC project (Quantum):

Step	Logic processor	I/O processor
1	Overhead, e.g. communication with NOM, NOE etc.	-
2	-	Writing outputs allocated to segment 1
		Reading inputs allocated to segment 1
3	-	Writing outputs allocated to segment 2
		Reading inputs allocated to segment 2
4	-	Writing outputs allocated to segment 3
		Reading inputs allocated to segment 3
...
n	-	Writing outputs allocated to segment n (n =< 32)
		Reading inputs allocated to segment n (n =< 32)
n+1	Executing IEC section 1	-
n+2	Executing IEC section 2	-
n+3	Executing IEC section 3	-
	..	-
m	Executing IEC section n (n =< 1600) and back to stage 1	-

1 The overhead is executed in this stage (e.g. communication with the coupling modules NOM, NOE).

2 - n The hardware signals from the allocated modules respective segments are written and read by the I/O processor in these stages (corresponding to the settings in the Segment scheduler (See *Segment manager*, p. 86)).

n+1 - m The logic processor runs the IEC sections logic in these steps. It then "Returns" to stage 1.

Note: No hardware signals are read or written. The values read in stage 2 to n are used exclusively. The outputs calculated in these stages are transferred in stages 2 to n (corresponding to the settings in the Segment manager).

Processing an IEC project

This table describes the processing of an IEC project (Quantum):

Step	Logic processor	I/O processor
1	Overhead, e.g. communication with NOM, NOE etc.	-
2	-	Writing outputs allocated to segment 1
		Reading inputs allocated to segment 1
3	-	Writing outputs allocated to segment 2
		Reading inputs allocated to segment 2
4	-	Writing outputs allocated to segment 3
		Reading inputs allocated to segment 3
HE1	1. I/O event section, spontaneous execution, when Hardware Interrupt occurs	-
HE2	2. I/O event section, spontaneous execution, when Hardware Interrupt occurs	-
...
HE64	64. (last) I/O event section, spontaneous execution, when Hardware Interrupt occurs	-
TE1	1. Timer event section, only executed when time interrupt occurs	-
TE2	2. Timer event section, only executed when time interrupt occurs	-
...
TE16	16. Timer event section, only executed when time interrupt occurs	-
...
n	-	Writing outputs allocated to segment n (n =< 32)
		Reading inputs allocated to segment n (n =< 32)
n+1	Executing IEC section 1 (cyclically)	-
n+2	Executing IEC section 2 (cyclically)	-
n+3	Executing IEC section 3 (cyclically)	-
	..	-
m	Executing IEC section n (n =< 1600) and return to stage 1	-

-
- 1** The overhead is executed in this stage (e.g. communication with the coupling modules NOM, NOE).
 - 2 - n** The hardware signals from the allocated modules respective segments are written and read by the I/O processor in these stages (corresponding to the settings in the Segment scheduler (See *Segment manager, p. 86*)).
 - n+1 - m** The logic processor processes the IEC sections logic in these steps. It then "Returns" to stage 1.
 - Note:** No hardware signals are read or written. The values read in stage 2 to n are used exclusively. The outputs calculated in these stages are transferred in stages 2 to n (corresponding to the settings in the Segment scheduler).
 - HE1 - HE64** If a hardware interrupt signal specially assigned to a section changes its value according to its parameter configuration, the cyclical processing and if necessary the processing of a Timer event section is immediately stopped and returned to the I/O event section. Once all event sections (and Timer event sections) are processed, the cyclical processing is continued at the point where the interrupt occurred. (See also chapter "*I/O Event Sections, p. 1060*")
 - TE1 - TE16** When a specially configured Timer interrupt signal for a section occurs, cyclical processing is immediately stopped and jumps to the Timer event section. Once Timer event sections are processed, the cyclical processing is continued at the point where the interrupt occurred as long as there are no further instructions for Timer event sections. (See also chapter "*Timer Event Sections, p. 1047*")
-

Programs

Structure of a program

A program consists of one or more Sections (See *Sections, p. 40*) or section groups. Section groups can contain sections and other section groups. Section groups can be created exclusively and filled using **Project** → **Project browser** (See *Project Browser, p. 491*). Sections describe the entire systems mode of operating.

Moreover the variables, constants, literals and direct addresses are managed within the program.

Variables

Variables are used to exchange data within a section, between several sections and between the program and the PLC.

Variables are declared using the menu command **Project** → **Variable declaration**. If the variable with this function is assigned an address, it is called a Located variable. If the variable has no address assigned to it, it is called an Unlocated variable. If the variable is assigned with a derived data type, it is called a Multi-element variable.

There are also constants and literals.

The following table provides an overview of the various types of variables:

Variable type	Description
Located variables	<p>Located variables are allocated a State RAM address (reference address 0x, 1x, 3x,4x). The value of this variable is saved in the State RAM and can be changed online using the Reference data editor. These variables can be addressed using their symbolic names or using their reference address.</p> <p>All PLC inputs and outputs are connected to the State RAM. The program can only access peripheral signals attached to the PLC via located variables. Access from external pages via Modbus or Modbus Plus interfaces of the PLC, e.g. from visualization systems can be made using located variables.</p>
Unlocated variables	<p>Unlocated variables are not assigned a State RAM addresses. They therefore do not occupy any State RAM addresses. The value of this variable is saved internally in the system and can be changed using the Reference data editor. These variables are only addressed using their symbolic names.</p> <p>Signals requiring no peripheral access, e.g. intermediate results, system tags etc, should primarily be declared as unlocated variables.</p>
Multi element variables	<p>A variable which is assigned a Derived data type.</p> <p>A distinction is made here between Structured variables and Array variables.</p>
Structured variables	<p>Variables to which a Derived data type defined using a STRUCT (structure) is assigned.</p> <p>A structure is a collection of data elements with generally different data types (Elementary data types and/or Derived data types).</p>

Variable type	Description
Array variables	A variable which is assigned a defined data type with the key word ARRAY. An array is a collection of data elements with the same data type.

Variable start behavior

In start behavior of PLCs there is a distinction between cold restarts and warm restarts:

- **Cold restart**

Following a cold restart (loading the program with **Online** → **Download**) all variables (irrespective of type) are set to "0" or their initial value if available.

- **Warm restart**

In a warm restart (stopping and starting the program or **Online** → **Download changes**) different start behaviors are valid for located variables/direct addresses and unlocated variables:

- **Located variables/direct addresses**

In a warm restart all 0x, 1x and 3x registers are set to "0" or their initial value if available.

The buffered coils are an exception to this. Buffered coils retain their current value (storage behavior).

4x registers retain their current value (storage behavior).

- **Unlocated variables**

In a warm restart all unlocated variables retain their current value (storing behavior).

This varying behavior in a warm restart leads to peculiarities in the warm restart behavior of set and reset functions.

- **Set and Reset in LD and IL**

Warm restart behavior is dependent on the variable type used (storage behavior in use of unlocated variables; non storage behavior in use of located variables/direct addresses)

- **SR and RS Function Blocks in FBD, LD, IL and ST**

These function blocks work with internal unlocated variables and therefore always have a storage behavior.

Constant variables

Constants are unlocated variables assigned a value, which cannot be modified by the logic program (read only).

Literals (values) Literals are used to describe FFB inputs, and transition conditions etc using direct values. These values cannot be overwritten by the program logic (read only).

The values of literals can be changed online.

There are two different types of literal; generic and standardized.
The following table provides an overview of the various types of literals:

Literal	Description
Generic literals	If the literal's data type is not relevant, simply specify the value for the literal. In this case, Concept automatically assigns a suitable data type to the literal.
Standardized literals	If you would like to manually determine a literal's data type, this may be done using the following construction: "Data type name"#"Literal value" For example INT#15 (Data type: Integer, value: 15), BYTE#00001111 (Data type: Byte, value: 00001111) REAL#23.0 (Data type: Real, value: 23.0) To assign the data type REAL the value may also be specified in the following manner: 23.0. Entering a comma will automatically assign the data type REAL.

Direct addresses Direct addresses are memory ranges in the PLC. They are located in the State RAM and can be assigned Input/Output modules.

Direct addresses can be entered or displayed in various formats. The display format is specified in the dialog box **Options** → **Preferences** → **Common**. Setting the display format has no impact on the entry format, i.e. direct addresses can be entered in any format.

The following address formats are possible:

- **Standard format (400001)**
The five character address comes directly after the first digit (the Reference).
- **Separator format (4:00001)**
The first digit (the Reference) is separated from the following five-character address by a colon (:).
- **Compact format (4:1)**
The first digit (the Reference) is separated from the following address by a colon (:), and the leading zeros of the address are not given.
- **IEC format (QW1)**
In first place, there is an IEC identifier, followed by the five-character address.
 - %0x12345 = %Q12345
 - %1x12345 = %I12345
 - %3x12345 = %IW12345
 - %4x12345 = %QW12345

The values of direct address can be modified online using the Reference data editor (See *Reference data editor*, p. 531).

Start behavior of digital outputs Outputs that are assigned 0x registers are deleted during PLC startup. Digital outputs that assigned 4x registers keep their current value when the PLC is stopped or started.

Sections

Introduction

A program consists of one or more sections. A section describes the mode of functioning of a systems technological unit (for example a motor).

Each section has its own document window in Concept. For overview purposes it is useful to divide a very large section into several small ones. The scroll bar is used to move within a section.

The page break can be made visible for each section, so that the page format can be monitored when programming. In this way, a readable printout of the section is assured.

Section types

There are three different types of sections in Concept provided for Quantum processing.

- **Cyclical section** are executed in every program cycle. The reaction time depends on the cycle time and is a minimum of one cycle and maximum of two cycles.
 - **I/O event sections** are not executed cyclically, but are started and processed spontaneously when a specially assigned Interrupt signal value changes state (corresponding to the setting in the Configurator and Section properties). The 140-HLI-340-00 module provides 16 Interrupt inputs. The local backplane has space for a maximum of 4 HLI modules.
The reaction time to an I/O event generally depends on the process duration of the EFBs to be processed in the section as well as the transition times.
 - **Timer event sections** are started and processed in precise user defined intervals.
The time intervals are defined in multiples of 1ms and a Phase in the Section properties for Timer Event Sections dialog box.
The reaction time is independent of the cycle time. Reactions to outputs are also carried out in defined time intervals.
-

Maximum number of sections

There can be up to a maximum of 1,600 sections per program.

Programming languages


Sections can be programmed using the IEC programming languages FBD (Function Block Diagram), LD (Ladder Diagram), SFC (Sequential Control), IL (Instruction List), or ST (Structured Text), or in the LL984 programming language (Ladder Logic), which resembles Modsoft. Only one of the stated programming languages is permitted to be used within a section.

Exchanging values	Values are exchanged within sections via links, variables, or direct addresses. Values are exchanged between different sections via variables or direct addresses.
Section execution order	The LL984 sections are the first to be executed. The LL984 section vertical sequence can be defined via the Project → Configurator → Configure → Segment scheduler... dialog box. Once the entire LL984 section has been processed, the IEC sections are then processed (FBD, SFC, LD, IL, ST). The execution order can be determined using either the Project → Execution order... or the Project browser (See <i>Project Browser, p. 491</i>) dialog box.
Printing sections	Sections are divided into pages when printing out. The amount of information on these pages is dependent on the settings in the menu File → Print . Page division can be displayed using the menu option View → Page breaks .
Section variable	<p>A Multi-element variable is automatically generated for each IEC section (FBD, SFC, LD, IL, and ST) and has the same name as the section.</p> <p>This variable is SECT_CTRL data and has two elements:</p> <ul style="list-style-type: none">• The "disable" BOOL data type element for disabling sections.• The "hsbyState" BYTE data type element for displaying the Hot Standby status of sections. <p>If the smallest bit of this element is set, the data from this section is transferred/received, see the <i>Hot Standby User's manual</i>. (This bit corresponds to the exclamation mark in the project browser.)</p>

Disabling sections

The component "disable" can be used to enable/disable the section variable. If the multi element address is not used or if the value 0 has been assigned to "disable", the corresponding section is executed. If "disable" is assigned the value "1", the corresponding section will not be executed. By using this variable, the execution of sections can be controlled according to events.

Note: If a disabled section is animated, the DISABLED status is displayed in the status bar.

	CAUTION
	<p>Risk of unwanted process states.</p> <p>Disabling a section does not mean that programmed outputs will be deactivated within the section if an output has already been set in a prior cycle, this status remains even after the section is disabled. The status of these outputs cannot be modified.</p> <p>Failure to follow this precaution can result in injury or equipment damage.</p>

Disabling Interrupt Sections

A specific Interrupt section can be disabled using the ISECT_OFF block. It can be enabled again using the ISECT_ON block. The section names are provided by the SECT_CTRL control variable.


The I_LOCK block can disable all interrupt sections. They can be enabled again using the I_UNLOCK block.

Note: A possible interrupt on an interrupt section has no effect.

**Lock section UN-
CONDITIONAL-
LY (possibility 1)**

The procedure for locking a section unconditionally is as follows:

Step	Action
1	Using Online → Reference data editor open the Reference data editor (See <i>Reference data editor, p. 531</i>).
2	By double clicking on a line number, open the Lookup variables dialog box.
3	From the area Data type first choose the option Structured and then from this list SECT_CTRL . Result: The names of all sections are displayed.
4	Now select the names of the section to be locked.
5	Use the command button Components... to select the ANY type components dialog box.
6	Select the line disable: BOOL and confirm with OK .
7	If the following has not been performed yet: Create a connection between the PLC and the programming device and load your program onto the PLC.
8	Change the entry in the column Value to 1 (TRUE) to lock the section or 0 (FALSE) to enable the section.
9	Using Online → Animation activate the animation if it is inactive. Result: The section is disabled or enabled according to the value. Note: Locking a section does not mean that programmed outputs will be deactivated within the section if an output has already been set in a prior cycle, this status remains even after the section has been disabled. The status of these outputs cannot be modified.

CAUTION	
	Risk of unwanted process states. The entry in the column Value remains even after the reference data editor has been closed (even if the entries are not saved), or in other words, the section remains disabled and must be explicitly re-enabled via the reference data editor (value = 0).
	Failure to follow this precaution can result in injury or equipment damage.

**Lock section UN-
CONDITIONAL-
LY (possibility 2)**

The procedure for locking a section unconditionally is as follows:

Step	Action
1	Using Project → Project browser open the Project browser (See <i>Project Browser</i> , p. 491).
2	From Online → Connect... create a connection between the programming device and the PLC.
3	From Online → Download... (if the program is in NOT EQUAL mode) or Online → Download changes (if in MODIFIED mode) restore the consistency between the programming device and the PLC.
4	Select the section to be locked from the project browser.
5	Activate the context menu for sections using the right mouse button, and activate Animate enable state .
6	<p>Change the enable status using the menu command Switch enable state from the context menu (right mouse button) of the selected section.</p> <p>Note: Sections may only be disabled or enabled via the Project browser, if they have not already been disabled/enabled via another Section (See <i>Locking a section CONDITIONALLY</i>, p. 45) or via the Reference data editor (See <i>Lock section UNCONDITIONALLY (possibility 1)</i>, p. 43).</p> <p>Result: The section is locked.</p> <p>Note: Locking a section does not mean that programmed outputs will be deactivated within the section if an output has already been set in a prior cycle, this status remains even after the section has been disabled. The status of these outputs cannot be modified.</p>

Locking a section CONDITIONALLY

The procedure for locking a section conditionally (program dependent) is as follows:

Step	Action
1	<p>Create the logic according to the section to be locked, for example in an FBD section.</p> <p>When doing this, please note that the logic must carry a BOOL data "output" and that the section to be disabled will be disabled at logic "1".</p> <p>Note: The section containing a logic for disabling/enabling other sections should not be disabled.</p>
2	By double clicking on your logic's "output", open the Connect FFB dialog box.
3	Use the command button Lookup... to open the Lookup Variable dialog box.
4	<p>From the area Data type first choose the option Structured and then from this list SECT_CTRL.</p> <p>Reaction: The names of all sections are displayed.</p>
5	By double clicking, now select the names of the section to be locked.
6	<p>Select the line disable: BOOL and confirm with OK.</p> <p>Result: The multi-element variable from the section to be locked (Section name.disable) now creates the "output" of the logic.</p>
7	From Project → Execution order... open the Section Execution Order dialog box.
8	Using the command buttons, ensure that the section containing the logic for locking is executed before the section to be locking is executed.
9	<p>If the following has not been performed yet:</p> <p>Create a connection between the PLC and the programming device.</p>
10	<p>Download your program to the PLC.</p> <p>Result: When logic "1" is at the "Output" the section to be locked is not edited.</p> <p>Note: Locking a section does not mean that programmed outputs will be deactivated within the section if an output has already been set in a prior cycle, this status remains even after the section has been disabled. The status of these outputs cannot be modified.</p>

Configuration data

Description

The PLC configuration is the interface between the program and the hardware.

The configuration data consists essentially of the component list and the entry in the address field of the program.

Loadables facilitate communication with the IEC programming language and the loading of further LL984-Instructions.

Creating a Project

4

At a Glance

Overview

This chapter describes the general procedure for the initial creation of a project. The most linear sequence possible is used here, in order to show a Concept-newcomer an easily manageable way of creating a project. Crosslinks between the Menu Commands are of course possible. As they gain experience, users will learn shortcuts and alternatives. For more detailed information, please see the relevant chapters in the user manual.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Overview	48
Step 1: Launching Concept	49
Step 2: Configuring the PLC	50
Step 2.1: Required Configuration	51
Step 2.2: Optional Configuration	53
Step 3: Creating the User Program	57
Step 4: Save	60
Step 5: Loading and Testing	61
Step 6: Optimize and Separate	66
Step 7: Documentation	67

Overview

Project Creation The creation of a project has 7 main steps:

Step	Action
1	Launching Concept (See <i>Step 1: Launching Concept, p. 49</i>) Launch Concept and start a new project.
2	Configuring the PLC (See <i>Step 2: Configuring the PLC, p. 50</i>) Set the hardware configuration.
3	Creating the user program (See <i>Step 3: Creating the User Program, p. 57</i>) Create new sections and create your program.
4	Save (See <i>Step 4: Save, p. 60</i>) Save your project
5	Loading and testing the project (See <i>Step 5: Loading and Testing, p. 61</i>) Create a link between the PC and the PLC. Load the project in the PLC and start it. Test the program with the Online Test Function. Now eliminate any mistakes in the program! Load the altered sections into the PLC.
6	Optimize and Separate (See <i>Step 6: Optimize and Separate, p. 66</i>) It is now advisable to optimize the program storage capacity and to reload the optimized program into the PLC. After successfully loading, testing and (if necessary) optimizing, you may disconnect the PC from the PLC. The program will now run offline.
7	Documenting (See <i>Step 7: Documentation, p. 67</i>) Create a complete set of documentation of your project.

Notes

Note: The steps "Configuring the PLC" and "Creating the User Program" can be performed in either order, i.e. the PLC configuration can also be changed after the creation of the program.

Note: In order to prevent loss of data, you should save your program regularly.

Step 1: Launching Concept

Launching Concept

The procedure for launching Concept is as follows:

Step	Action
1	Double click on the Concept icon to launch Concept.
2	Select File → New Project . Response: The project will be opened as an untitled project.
3	Later on, save this project with a name <i>Step 4: Save, p. 60</i> . A saved project can be invoked with the Open Project... command, or by using its project icon.

Note

Note: When making additional changes please note the settings in the submenu **Options** → **Preferences!**

Resume

Now proceed with Step 2: Configuring the PLC (See *Step 2: Configuring the PLC, p. 50*).

Step 2: Configuring the PLC

What should be configured?

Using **Project** → **PLC configuration** configure the entire hardware configuration for your project.

Required Configuration

Note: The PLC type must first be set! All further configurations can then be executed independently of the processing sequence.

The following configurations are necessary for the configuration:

- *Specifying the type of PLC (minimum configuration), p. 51*
 - *Set memory partitions, p. 51*
 - *Install loadables, p. 52*
 - *Set I/O map, p. 52*
-

Optional Configuration

The following configurations are to be used according to the project:

- *Set head setup, p. 53*
 - *Set Modbus communication, p. 53*
 - *Set Peer Cop communication, p. 54*
 - *Set data protection, p. 55*
 - *Various PLC settings, p. 56*
 - *ASCII messages (only for 984 LL), p. 56*
-

Step 2.1: Required Configuration

Precondition The PLC type must first be set! All further configurations can then be executed independently of the processing sequence.

Specifying the type of PLC (minimum configuration) The procedure for specifying the type of PLC (minimum configuration) is as follows:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the PLC Selection menu command from the list. Response: The PLC selection dialog is opened.
3	From the PLC family list select your PLC type.
4	Select your CPU from the CPU/Executive list.
5	From the Runtime list select the Enable status. Response: It is possible to program sections in IEC languages (FBD, LD, IL and ST). Note: In the Runtime list, the status Not available , Disabled or Only 984 is displayed, then the selected CPU does not support any IEC programming languages. If in the list the status Only IEC is displayed, then the selected CPU exclusively supports IEC languages and these do not have to be explicitly enabled.
6	With simple tests and programs the configuration can now be exited and the procedure continued from <i>Step 3: Creating the User Program, p. 57</i> or <i>Step 4: Save, p. 60</i> .

Set memory partitions The procedure for setting the memory partition is as follows:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the PLC memory partition menu command from the list. Response: The PLC memory partition dialog is opened.
3	In the Discretes and Words ranges select the probable number of I/O flag bits and I/O words, to be required by the user program Note: The maximum address range, that must not be exceeded, can be read on the right-hand side of the dialog.

Install loadables The procedure for installing the loadables is as follows:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the Loadables menu command from the list box. Response: The Loadables dialog is opened.
3	Select the loadable in the Available: list. Note: Loadables are assigned in the <i>Loadables, p. 84</i> section.
4	Select the Install => command button. Response: The selected loadable is moved to the Installed: field.
5	Repeat the steps 3 and 4 until all the loadables required have been installed.

Set I/O map The procedure for setting the I/O map is as follows:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the I/O map menu command from the list. Response: The I/O map dialog is opened.
3	Select the Supervision time column and enter a time, within which a communication exchange must take place. If this time is exceeded, an error message appears.
4	Select the Edit... command button. Response: The dialog for entering modules is opened.
5	In the Module column, select the ... command button. Response: The I/O Module Selection dialog is opened.
6	In the Modules column, select the module. Response: The module is displayed in the current slot.
7	Select the Input start and/or Output start columns and enter the first address of the occupied input and/or output reference range for the module.
8	Select the module and choose the Params command button. Response: If the module has a parameter dialog, you can define the parameter (e.g. disconnect behavior, data format, measuring range) here.

Resume Now proceed with Step 3: Creating the user program (See *Step 3: Creating the User Program, p. 57*).

Step 2.2: Optional Configuration

General Information

The following configurations do not need to be executed urgently, but they offer extended functions.

Set head setup

The procedure for specifying the remote I/O is as follows (this procedure is optional for minimum configuration):

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the I/O map menu command from the list. Response: The I/O map dialog is opened.
3	Select the Head setup... command button. Response: The Head Setup dialog is opened.
4	Enter the slots for the RIO or NOM modules. Response: Return to the I/O map dialog.
5	Select the head setup in the Go To list.
6	Select an empty line (last line) in the table, and select the Insert command button. Response: In the Type column another I/O station is entered.
7	Select the Drop column and enter the station number. Note: Only as many remote I/O stations can be configured as there are segments registered in the segment scheduler.
8	Select the head setup in the Go To list for the 2nd drop.
9	Next, carry out steps 3 to 6 of the <i>Set I/O map, p. 52</i> procedure.

Set Modbus communication

To set the Modbus communication (Quantum slave, terminal, printer, etc.) proceed as follows:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the Modbus Port settings menu command from the list. Response: The Modbus port settings dialog is opened.
3	Make the corresponding settings.

Set Peer Cop communication

If a Modbus Plus link exists, the Peer Cop functionality is able to transfer state RAM data globally or directly between several nodes on a local network. The procedure for setting the Peer Cop communication is as follows:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the Config. Extensions → Select Extensions list. Response: The Select extensions dialog is opened.
3	Check the Peer Cop box. Response: Return to the PLC configuration window and the Peer Cop menu command is now available.
4	Select Config. Extensions → Peer Cop . Response: The Peer Cop dialog is opened.
5	In the Go To range select the local bus devices, and enter the slot.
6	Select in the Global range the Receive... and Send... command buttons to define the destination and source addresses of the transmission data and/or the address of the other bus devices.
7	Select in the Specific range the Receive... and Send... command buttons to define the destination and source addresses of the transmission data and/or the address of the other bus devices.

Set data protection

Address ranges of coils and holding registers can be protected from being overwritten by external signals. The procedure for setting the data protection is as follows:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the Config. Extensions → Configuration extensions . Response: The Configuration extensions dialog is opened.
3	Check the Data protection box. Response: Return to the PLC configuration window and the Data protection menu command is now available.
4	Select Config. Extensions → Data protection . Response: The Data protection dialog is opened.
5	Check the Data protection box. Response: The Data protection... menu command can now be selected in the PLC configuration overview dialog.
6	Select the range for the coils and holding registers. This range should contain write-protection.

Various PLC settings

Diverse internal PLC data can be evaluated, a watchdog timeout for the user program can be specified, the time windows for the communication (I/O time disk) parameterized and the multiple assignment of outputs authorized. The procedure for setting the PLC settings is as follows:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select the Specials menu command from the list. Response: The Specials dialog is opened.
3	Check the Battery coil , Timer register and Time of Day check boxes and enter an address in the corresponding text boxes.
4	Check the Allow Duplicate Coils check box and enter the address from which this should be allowed in the text box..
5	In the Watchdog timeout (ms*10) : text box enter a numeric value between 2 and 255 (ms). This enables you to set an impulse watchdog for the user program. Response: As soon as the count pulses exceed the specified time, an error message appears.
6	In the Online Editing Timeslice (ms) : text box enter a numeric value between 3 and 100 (ms). This enables you to define a time for executing the multi-cycle edit functions (paste, delete, find etc.)

ASCII messages (only for 984 LL)

To set the ASCII messages (only for 984LL), execute the following steps:

Step	Action
1	Select Project → PLC configuration . Response: The PLC configuration window is opened, this contains further menu commands for hardware configuration.
2	Select from the list ASCII → ASCII Setup . Response: The ASCII Setup dialog is opened.
3	Enter the total messages, the size of the message width and the number of ASCII ports (from the I/O periphery) in the text boxes. Response: In the PLC configuration → ASCII window the ASCII Port Settings menu command is available.
4	Select from the list ASCII → ASCII port settings . Response: The ASCII port settings dialog is opened.
5	Make the corresponding settings. Note: ASCII messages can now be created under Project → ASCII messages...

Resume Now proceed with Step 3: Creating the user program (See *Step 3: Creating the User Program*, p. 57).

Step 3: Creating the User Program

General A user program is created in sections. Each section is programmable in one of the available languages and has a unique name in the project. Sections can be generated at any time during the programming.

Overview The creation of a user program consists of 9 steps:

Step	Action
1	Generating a New Section (See <i>Generating a New Section</i> , p. 57)
2	Declaring the Variables (See <i>Declaring the Variables</i> , p. 58)
3	Programming a Section (See <i>Programming a Section</i> , p. 58)
4	Analyzing Program/Section (See <i>Analyzing Program/Section</i> , p. 58)
5	Specifying the section execution sequence (See <i>Set execution order of sections</i> , p. 59)

Generating a New Section The procedure for generating a new section is as follows:

Step	Action
1	In the main menu File call up the menu command New section... . Result: The dialog box New program section is opened.
2	Click on the programming language desired for this section.
3	In the text box Section name enter the unique name for this section.
4	Generate all the required sections in this way.

Declaring the Variables

A program consists of functions and Function Blocks (FFBs) or of instructions with the statement of variables (e.g. signals), addresses or literals. While direct addresses and literals can be used immediately, variables must be declared before they can be used in programming. The procedure for declaring variables is as follows:

Step	Action
1	In the main menu Project call the menu command Variable declaration... . Result: The dialog box Variable declaration is opened.
2	Enter the variable name, the associated data type, and if necessary the reference address, the initial value and a comment.
3	Confirm the entries with OK . Note: Further editing is also possible from a FFB connection or contact etc. by double-clicking -> Var. Declaration... . This starts the Variables editor.

Programming a Section

The procedure for programming a section is as follows:

Step	Action
1	Using File → Open section open the section to be programmed.
2	Create programs according to the rules of the individual programming languages: <ul style="list-style-type: none"> ● Function Block Diagram FBD (See <i>Function Block language FBD</i>, p. 173) ● Ladder Diagram LD (IEC) (See <i>Ladder Diagram LD</i>, p. 199) ● SFC (Sequential Control) (See <i>Sequence language SFC</i>, p. 233) ● Instruction list (IL) (See <i>Instruction list IL</i>, p. 279) ● Structured text (ST) (See <i>Structured text ST</i>, p. 341) ● LL984 (Ladder Diagram (Modsoft)) (See <i>Ladder Logic 984</i>, p. 387)

Analyzing Program/Section

Check a section or the entire program for syntax violations! The procedure for analyzing a program/section is as follows:

Step	Action
1	In the main menu Project call up the menu command Analyze section or Analyze program .
2	Remove the cause of the displayed or reported error. Note: Loading a section or program into the PLC is only possible after an error-free check. (The removal of the cause of warnings is not absolutely necessary. Checking the warnings is, however, sensible.)

Set execution order of sections

The sections are initially stored in the order of their creation and are executed after the program has started. In general this sequence must be adjusted project-specifically to suit the task setting. The procedure for specifying the section execution sequence is as follows:

Step	Action
1	To specify the section execution sequence there are two alternatives: <ul style="list-style-type: none">● In the main menu Project call the menu command Execution order... and using the command buttons First, Last, Next, Previous sequence the sections as required.● In the main menu Project call up the menu command Project browser and sequence them as required by moving them around in the <i>Project Browser</i>, p. 491.

Resume

Now proceed with Step 4: Saving (See *Step 4: Save*, p. 60).

Step 4: Save

General Information

General information about saving:

- If you exit a project without saving, you will be automatically asked if you want to save the project or not. If you answer yes to this question, this begins the same procedure described below.
 - In order to prevent loss of data, projects should be saved regularly during long periods of configuration or programming sessions.
-

Saving a Project for the First Time

The procedure for saving a project for the first time is as follows:

Step	Action
1	In the File main menu invoke the Save Project As... menu command.
2	In the File name text box, enter the project name name.prj.
3	Select the desired drive and directory from the Directory list. Alternatively, it is possible to enter the whole path specification in the File name text box, e.g. <code>c:\product1\reactor3.prj</code> (max. 28 characters + .prj). If these directories do not yet exist, they will be automatically created. Note: According to IEC 1131, a project includes all programs, data etc which belong to a PLC. If several projects (i.e. PLCs) belong to one system, then all projects should be stored in a common directory named after the system.
4	Click the OK command button. Response: The project has now been stored in the specified directory under the given name.

Supplementary Saving

The procedure for supplementary saving is as follows:

Step	Action
1	From the File main menu simply select the Save menu command.

Resume

Now proceed with Step 5: Loading and testing the project (See *Step 5: Loading and Testing, p. 61*).

Step 5: Loading and Testing

General Information

Loading and testing programs is only possible if

- either the 16-bit simulator Concept SIM is switched on or
- the Concept SIM 16-bit simulator is switched off and a PLC is attached with a Modbus Plus, Modbus, TCP/IP cable, or
- the Concept PLCSIM32 simulator is switched on.

Note: Testing using Concept SIM (See *Simulating a PLC (16-bit simulator)*, p. 679) and Concept PLCSIM32 (See *Simulating a PLC (32-bit simulator)*, p. 682) simulators is only possible with IEC user programs.

Overview

Loading and testing macros is divided into 9 main steps:

Step	Action
1	Loading the EXEC file into the PLC (see <i>Concept Installation Instructions</i>)
2	Connecting the PC and PLC (See <i>Connecting the PC and PLC</i> , p. 61)
3	Loading and Starting the Program (See <i>Loading and Starting the Program</i> , p. 62)
4	Activating the Animation (See <i>Activating the Animation</i> , p. 63)
5	Changing the Values of Literals (See <i>Changing the Values of Literals</i> , p. 63)
6	Changing the Values of Variables (See <i>Changing the Values of Variables</i> , p. 64)
7	Locating Errors (See <i>Locating Errors</i> , p. 64)
8	Downloading Changes (See <i>Downloading Changes</i> , p. 65)
9	Starting and Stopping the PLC (See <i>Starting and Stopping the PLC</i> , p. 65)

Connecting the PC and PLC

The procedure for linking the PC and the PLC is as follows:

Step	Action
1	From the Online main menu invoke the Connect... menu command. Response: The Connect to PLC dialog box opens.
2	Set the protocol type (Modbus, Modbus Plus, TCP/IP or Simulator) and the PLC node (when working in a network) with which you wish to communicate.
3	Under Access right select the Change Configuration option
4	Confirm the details with OK .

Loading and Starting the Program

The procedure for loading and launching the program is as follows:

Step	Action
1	From the Online main menu invoke the Connect... menu command. Response: The Download Controller dialog box will be opened in the PLC.
2	When loading the program for the first time, use the All command button.
3	Click the Download command button. Response: Various dialog boxes will be displayed.
4	Answer the question <i>Stop the program in PLC? Yes/No</i> with Yes . Note: This question only appears when a program is already running in the PLC.
5	Answer the question <i>Start a program in PLC? Yes/No</i> with Yes , if there are no errors. If warnings or errors are reported, these will be listed in the Messages window. Correct the warnings or errors at the specified point.

Activating the Animation

With the animation (online status report) it is possible to monitor the status of variables, steps, transitions etc within individual sections of the editor window. The procedure for activating the animation is as follows:

If...	Then...
To display binary values exclusively.	To display binary values exclusively, invoke the Online main menu and click on the Animate booleans menu command. Response: The valences of all booleans (variables, direct addresses, literals) are displayed in colour (0-Signal = red, 1-Signal = green).
If you want to display the values of all variables.	To display the values of all variables invoke the Editing main menu option and select the Select All menu command (selects all items in the current section). Thereafter invoke from the Online main menu option the Animate selection menu command. Response: The valences of all values (variables, direct addresses, literals) are displayed in colour (red = 0-Signal, green = 1-Signal, yellow = either, for variables, immediate display of the value or, for multi-element-variables, displays the value by double-clicking on the variable).
If you want to enter monitoring fields in the text languages (IL and ST).	Use the Selected Inspect menu command to paste the text languages IL and ST into section monitoring fields. Response: The current value of the allocated variables is shown in these monitoring fields. With multi element variables, only the value of the first element is shown. This can be changed by double-clicking on the monitoring field of the Numeric Inspect Settings dialog box, which invokes the options available.

Changing the Values of Literals

The procedure for changing literals is as follows:

Step	Action
1	Activate the animation, as described in <i>Activating the Animation</i> , p. 63.
2	Double-click on the literal to be changed.
3	Enter a new value and confirm with OK . Response: The new value will be sent to the PLC during the next logic scan.

Changing the Values of Variables

With the Reference data editor (See *Reference data editor, p. 531*) it is possible to show and set the values of variables (state, control, force). The procedure for changing variables is as follows:

Step	Action
1	From the main menu, select Online and then the Reference data editor menu command.
2	Enter the variables to be displayed in the dialog box marked RDE Templates .
3	To set the value highlight the Disable check box, and enter the desired value.
4	The RDE template can be saved under a unique name. To do this, invoke the RDE main menu option and select the Save template as... menu command. Note: Several RDE templates can be invoked at once. To do this, invoke the RDE main menu option and select the Open template... menu command.

Locating Errors

If errors occur during the processing of the program by the PLC, these will generally be reported on screen **Messages** and entered in an events list in log book form. The procedure for locating errors is as follows:

Step	Action
1	From the Online main menu invoke the Event Viewer menu command. Response: A window is opened, in which all errors are listed and described.
2	Select an error line and use the command button Go to Error . Response: This will go directly to the section in which the error occurred. The faulty object is highlighted.
3	Correct the program.
4	If your program now has the UNEQUAL status carry out the steps in <i>Downloading and Starting the Program, p. 62</i> once again. If the program now has the MODIFIED status perform the steps in <i>Downloading Changes, p. 65</i> once again.

Downloading Changes

If the project has the **MODIFIED** status after it has been altered, these changes can be loaded online into the PLC without stopping the program currently running. The procedure for downloading changes is as follows:

Step	Action
1	From the Online main menu access the Download Changes... menu command.
2	Click on OK . Response: The changes will be downloaded to the controller.

Starting and Stopping the PLC

The procedure for starting and stopping the PLC is as follows:

Step	Action
1	If the same project is running on the PC and PLC (EQUAL), then the PLC can be started or stopped with Online → Online Control Panel... .

Resume

Now proceed with Step 6: Optimize and Separate (See *Step 6: Optimize and Separate, p. 66*).

Step 6: Optimize and Separate

Optimizing Projects

At the end of the installation and/or after several runs of **Download Changes...** it is useful to perform an optimization, so that any gaps in the program data memory management are filled. After optimization the project is **UNEQUAL** on the PC and PLC and the program must be loaded into the PLC with **Download...** (**Warning:** Program must be stopped and restarted!). The procedure for optimizing projects is as follows:

Step	Action
1	Save the project with File → Save Project .
2	In the File main menu invoke the Close project menu command and take note of the dialog boxes which then appear.
3	In the File main menu invoke the Optimize Project... menu command and select the project to be optimized. Take note of the dialog boxes which subsequently appear.
4	Check the size of the program data memory in the Online main menu with the Memory Statistics... menu command.
5	The sizes can then be altered with PLC configuration .
6	Save the project with File → Save Project .
7	Reload the optimized program into the PLC using Online → Download... . To do this the program currently running must be stopped.
8	Start the newly loaded program using Online → Online Control Panel .

Separating the PC and Controller

After successfully testing the program in the PLC (with a connected process) the PC can be separated from the controller. The procedure for separating the PC and the controller is as follows:

Step	Action
1	Please take note of the program status in the footnote! To maintain consistency EQUAL must be there. <ul style="list-style-type: none"> ● if it reads MODIFIED, modifications must be loaded first <i>Downloading Changes, p. 65.</i> ● If it reads UNEQUAL the program must be reloaded into the PLC <i>Loading and Starting the Program, p. 62.</i>
2	From the Online main menu access the Disconnect... menu command. Take note of the information in the displayed dialog box.
3	The project can be closed after separation. In the File main menu invoke the Close project... menu command. Take note of the information in the dialog box, if displayed.

Resume

Now proceed with Step 7: Documenting (See *Step 6: Optimize and Separate, p. 66*).

Step 7: Documentation

General information

Each project should be fully documented. Changes and additions should also be documented (partial documentation).

Among other things documentation includes:

- Comments on the project (**Project** → **Properties**),
- Comments on each separate section (**File** → **Section properties**),
- Comments on variables,
- Comments on the functions applied, function modules and DFBs (command button **Comment** in the property dialog of each module),
- Comments on steps and transitions (command button **Comment** in the property dialog of each element),
- Comments in the form of freely placed text elements in the graphic programming languages (**Object** → **Text**),
- Comments on each line of commands in the textual programming languages
- Comments on user-specific data types,
- Comments on derived function modules (DFBs).

Printing the documentation

The procedure for printing documentation is as follows:

Step	Action
1	In the main menu call up File menu command Print... .
2	In dialog box Documentation contents select Page layout whether each page should have a uniform header and footer as well as printing a front page. The appearance of header, footer and front page is stored in the available ASCII files.
3	In the area Contents and in dialog box Documentation contents , select what is to be printed.
4	If Variable list has been selected, call up Options in order to select the variables which are to be printed.
5	When Sections has been selected, <ul style="list-style-type: none"> ● call up Select and specify the sections that are to be printed and ● also call up Options. In area Graphics enlargement factor also specify the appropriate size of the logic which is to be printed.
6	Activate command button OK . Reaction: All entries are saved.
7	Make sure that the page set-up of the sections is as desired. In the main menu call up View follow this with the successive menu commands Overview and Page Break .
8	Change the order of for example the FFBS in such a way, that there are as few transitions between adjoining pages as possible.
9	In the main menu call up File the menu command Print... again and activate command button Print . The printout is made with defined settings and the dialog box is closed.

PLC configuration

5

At a Glance

Overview

This section describes the single process for the hardware configuration.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
5.1	General information about hardware configuration	71
5.2	Configuration in OFFLINE and ONLINE mode	74
5.3	Unconditional Configuration	78
5.4	Optional configuration	90
5.5	Backplane Expander Config	98
5.6	Configuration of various network systems	101
5.7	Quantum Security Settings in the Configurator	111

5.1 General information about hardware configuration

At a Glance

Overview

This section contains general information about hardware configuration.

What's in this Section?

This section contains the following topics:

Topic	Page
General information	72
Proceed in the following way with the configuration	73

General information

At a Glance

The system configuration has far-reaching consequences as it influences the entire control work mode. It has to define all control-specific information as well as general information, allocate the necessary memory space and determine the input/output area. For the first configuration the user must enter several basic details for the PLC area, such as PLC type and memory. Only valid configurations are authorized. A configuration always refers to a Project, i.e. the menu command **PLC configuration** is only available when a project has been opened. The configuration is available offline or online.

Proceed in the following way with the configuration

Introduction

In this section you are given a general overview on how to proceed with the configuration.

Use Configuration Menu

There are menu commands that absolutely must be carried out and are available in the **PLC Configuration** window. Grayed out menu commands are currently unavailable and can be enabled for extending the hardware-configuration in the **Config. Extensions** directory with the menu command **Select Extensions**.

Read in Module Set-up

The PLC module set-up is entered manually and can be compared with the connected hardware in ONLINE mode. After it has been read in, the modules missing in Concept are shown in the I/O map, and can be re-edited.

The I/O addressing must then be done for each module.

When doing this, please ensure the permitted references are used:

Modules	References
Analog input modules	3x references
Analog output modules	4x references
Digital input modules	3x or 1x references
Digital output modules	4x or 0x references
Expert modules - input	3x or 1x references
Expert modules - output	4x or 0x references

Downloading the Hardware Configuration

The hardware configuration of a project is saved and can be downloaded to the simulation program Concept-SIM, Concept-SIM32 or an automation installation. By doing this, the EQUAL status is established between the host computer and the PLC.

Note: The Concept-SIM must be deactivated for transfer of the configuration to a real PLC.

5.2 Configuration in OFFLINE and ONLINE mode

At a Glance

Overview This section contains information for configuration in OFFLINE and ONLINE mode.

What's in this Section? This section contains the following topics:

Topic	Page
General information	75
Available Functions in OFFLINE and ONLINE Modes	76

General information

At a Glance

In OFFLINE mode no link is created between programming device and PLC, and the configuration can be performed. In ONLINE mode there is a link between programming device and PLC, so that only one conditional configuration can take place.

Available Functions in OFFLINE and ONLINE Modes

Introduction This section contains an overview of the available functions in OFFLINE and/or ONLINE mode. The possibilities in the ONLINE mode are different in their use of the simulator and the real PLC.

Configuration in OFFLINE Mode In OFFLINE mode all menu commands are available for the hardware configuration in the **PLC Configuration** window. The submenus in the **Config. Extensions** directory can be enabled in the **Select Extensions** dialog to extend the configuration.
If the PLC is in ONLINE mode, you can switch to OFFLINE mode using the menu command **Online** → **Disconnect....** In the footer of the editor window, the status-bar indicator NOT CONNECTED appears.

Configuration in ONLINE Mode and in the Active Simulator A configuration is not possible in ONLINE mode with an active simulator or a Modbus Plus connection, i.e. no entries can occur. The available dialogs can only be invoked and read.
You can switch to ONLINE mode using the menu command **Online** → **Connect...** and establishing a connection between the host computer and the PLC.

Configuration in ONLINE Mode and in the Real PLC Using the connection to a real PLC a configuration in ONLINE mode is possible, as long as the **Change Configuration** access level is activated.
It is not possible to configure or reconfigure a PLC while the PLC is in RUN mode. If a program is already running in the PLC, it must be stopped before reconfiguration can be implemented. Stop the PLC with **Online** → **Online Control Panel** → **Stop PLC**. After editing, the changes are automatically transferred to the hardware when the PLC is started up.

Note: When you delete an Expert module in ONLINE mode in the I/O map, the allocated loadable is also automatically deleted. If you wish to place this module back in the I/O map at a later time, it will be necessary to download again.

You can switch to ONLINE mode using the menu command **Online** → **Connect...** and establishing a connection between the host computer and PLC.

**Effects of
ONLINE
Changes**

If the following conditions are satisfied, all animated windows are automatically closed if a change is made in the I/O map (e.g. deleting or adding to a module)

Conditions:

- ONLINE mode
 - animated section(s)
 - Status between PLC and host computer is EQUAL
 - Controller stopped
 - Access level **Change Configuration** is activated.
-

5.3 Unconditional Configuration

At a Glance

Overview This section contains a description of the configuration to be performed unconditionally and an overview of the presettings in the configuration menu.

What's in this Section? This section contains the following topics:

Topic	Page
Precondition	79
PLC selection	80
CPU Selection for the PLC Type	80
PLC memory mapping	83
Loadables	84
Segment manager	86
I/O Map	87

Precondition

Introduction

Only when the CPU has been selected in the **PLC Selection** dialog will all the other menu commands become available in the **PLC Configuration** window.

The following dialogs are a minimum selection and **MUST** be edited as part of the hardware configuration.

- PLC Selection
- PLC Memory Partition
- Loadables
- Segment Scheduler
- I/O Map

The preferences can be adopted as long as they are compatible with the hardware being used.

PLC selection

Introduction Select the PLC family (Quantum, Compact, Momentum or Atrium) and the CPU, as well as the memory size, according to use. All the available CPUs are listed in the list box.

Determine logic zone The logic zone for the desired programming language (IEC or LL984) can be expanded to the corresponding PLC type with the PLC family selection. The assignment and installation of the loadables is determined according to the following settings:

Selection	Meaning
Enable	Installation of the IEC loadables. A desired memory area for the IEC zone can be set up. The assignment and installation of the loadable pairing to the selected CPU is performed automatically in the Loadables dialog.
Disable	No installation of the IEC loadables. This will completely switch off the IEC zone and the entire logic zone will be made available for the LL984.
984 only/IEC only	Some Momentum CPUs can only be programmed in the IEC zone or only in the LL984 zone.

Determine total IEC memory By defining the total IEC memory size and the global data, you also automatically determine the IEC-program memory size. On the basis of this size, the available memory space for the LL984 user program can also be determined.

Note: With global data it is the memory space of the unlocated variables.
--

Note: Total IEC memory = IEC program memory + global data
--

CPU Selection for the PLC Type

Introduction When installing hardware (Concept EXECLoader), you are required to load various EXEC data files (*.BIN). This determines the firmware for various PLC types. The available PLC types, which can be operated by loading the EXEC data files with the corresponding CPUs, are shown in the following tables.

**Loading
Firmware for
Quantum PLC
Types**

The following table shows the current EXEC versions, which are located on the Service Release CD and supplied with Concept.
Quantum PLC type:

140 CPU	Q186Vxxx (IEC+LL984)	Q486Vxxx (IEC+LL984)	Q58Vxxxx (IEC+LL984)	Q5RVxxxx (IEC+LL984)	QIECVxxx (IEC only) *	IEC Memory (kbyte)
113 02	X (LL984 only)	-	-	-	-	
113 02S	-	-	-	-	X	max. 150
113 02X	X (LL984 only)	-	-	-	-	
113 03	X	-	-	-	-	max. 136
113 03S	-	-	-	-	X	max. 379
113 03X	X	-	-	-	-	max. 136
213 04	X	-	-	-	-	max. 305
213 04S	-	-	-	-	X	max. 610
213 04X	X	-	-	-	-	max. 305
424 0x	-	X	-	-	-	max. 465
424 0xX	-	X	-	-	-	max. 465
434 12	-	-	X	-	-	max. 890
534 14	-	-	X	-	-	max. 2550
434 12A (Redesigned CPU)	-	-	-	X	-	max. 890
534 14A (Redesigned CPU)	-	-	-	X	-	max. 2550

Note: * After the QIECVxxx.BIN EXEC data file has been loaded, the EMUQ.EXE loadable must be loaded into Concept in the **Loadables** dialog.

**Loading
Firmware for
Quantum LL984
Hot Standby
Mode**

The Quantum CPUs not ending in X or S can be used for the LL984 Hot Standby mode. A special EXEC file must be downloaded onto the CPU for this. The loadable for LL984 Hot Standby (CHS_208.DAT) is automatically installed by the system.

Loading Firmware for Quantum IEC Hot Standby Mode

The 140 CPU 434 12 and 140 CPU 534 14 CPUs can also be used for IEC Hot Standby. A special EXEC file must be downloaded onto the CPU for this. The loadables for IEC Hot Standby (IHSB196.EXE and CHS_208.DAT) are automatically installed by the system.

Loading Firmware for Quantum Equation Editor

The Quantum CPUs not ending in X or S can be used for the LL984 equation editor. A special EXEC file must be downloaded onto the CPU flash for this. This EXEC file is not part of the Concept delivery range but can be obtained over the Internet at www.schneiderautomation.com.

Loading Firmware for Momentum PLC Type

The following table shows the current EXEC versions, which are located on the Service Release CD and supplied with Concept. Momentum PLC type (CPU 171 CCC 7x0 x0):

171 CCC	M1Vxxx (LL984 only)	M1IECxxx (IEC only)	IEC Memory (kbyte)
760 10-984	X	-	
760 10-IEC	-	X	256
780 10-984	X	-	
780 10-IEC	-	X	256

Momentum PLC type (CPU 171 CCC 9x0 x0):

171 CCC	M1EVxxx (LL984 only)	M1EWIxxx (IEC only)	IEC Memory (kbyte)
960 20-984	X	-	
960 30-984	X	-	
960 30-IEC	-	X	220
980 20-984	X	-	
980 30-984	X	-	
980 30-IEC	-	X	220

Momentum PLC type (CPU 171 CCS 7x0 x0):

171 CCS	M1Vxxx (LL984 only)	M1IECxxx (IEC only)	IEC Memory (kbyte)
700 10	X	-	
700/780 00	X	-	
760 00-984	X	-	
760 00-IEC	-	X	160

The stripped EXEC of the M1 supports up to a maximum of 44 I/O modules.

**Loading
Firmware for
Compact PLC
Types**

The **CTSxxxxD.BIN** EXEC file must be downloaded onto the CPU flash for all Compact CPUs.

**Loading
Firmware for
Atrium PLC
Types**

A special EXEC file (see table below) must be downloaded onto the CPU flash for each Atrium CPU.

180 CCO	EXEC file
121 01	AI3Vxxxx.BIN
241 01	AI5Vxxxx.BIN
241 11	AI5Vxxxx.BIN

PLC memory mapping

At a Glance


For the creation of the program, sufficient address zones for the necessary number of input bits, output/flag bits, input words and output/flag words are to be entered. An overview of the state RAM value is also given:

- Max. state RAM
- State RAM in use
- State RAM use

An unassociated value is shown with an error message, and can be automatically suited to the given value.

IEC Hot Standby data

After configuration of an IEC Hot Standby system, enter sufficient address zones for the required number of input words. The higher the number of IEC Hot Standby input words, the larger the transmit buffers for the IEC component. This means all the bigger the IEC application in use can be.

	CAUTION
	<p>System cycle time influence!</p> <p>The size of the configured state RAM in an IEC Hot Standby project has a significant effect on the system cycle time. As soon as a configured cycle ends, the next starts after the transfer of all state RAM data to the CHS module.</p> <p>Failure to follow this precaution can result in injury or equipment damage.</p>

Loadables

Introduction

Loadables are loadable programs, which are only loaded into the PLC when required.

The various uses of loadables are described in the following sections.

Note: When you delete an Expert module in online mode in the I/O map the allocated loadable is also automatically deleted. If you wish to place this module back in the I/O map at a later time, it will be necessary to download again.

Downloading Loadables for the IEC Runtime System

The following loadables for the combined execution of IEC and LL984 programs (CPU 113 0x, CPU 213 0x or CPU 424 02) are available:

If...	Then...
you want to use CPUs with the mathematics processor for IEC programming,	install the loadable pairing @1S7196 and @2I7196.
you want to use CPUs without the mathematics processor for IEC programming,	install the loadable pairing @1SE196 and @2IE196.

Downloading Loadables for Expert Modules

The following loadables are available for Expert modules:

If...	Then...
you are configuring the 140 ESI 062 00 module with 32 bit runtime system and the 140-NOA-611-x0 module	install the loadable ASUP196. Note: The ULEX196 loadable is automatically installed. The ASUP 196 loadable is only installed automatically on 32-bit CPUs. On 16-bit CPUs with Stripped EXEC (QIEC\xxx.BIN), the ASUP196 loadable must be installed afterwards.
you are configuring the 140 ESI 062 10 module,	install the loadable pairing NSUP + ESI. Note: These two loadables do not come with the Concept software package, but are supplied with the 140 ESI 062 10 module and must be unpacked at the time of installation (Unpack...).

Downloading Loadables for LL984

These are not included in the Concept delivery range. You can order these loadables via the "Automation Customer Service Bulletin Board (BBS)" (related topics README).

Downloading Loadables for Hot Standby

The following loadables for Hot Standby mode are available:

If...	Then
you are using the LL984 Hot Standby mode,	the loadable CHS_208 is automatically installed.
you are using the IEC Hot Standby mode,	the loadables IHSB196 and CHS_208 will be loaded automatically.

Downloading User Loadables

Loadables that are created by the user are called user loadables (*.EXE, *.DAT). They are located in the Concept directory DAT and using the **Unpack...** command button they can be inserted into the **Loadables** dialog at installation.

Downloading Loadables for IEC Support Only

The following loadables for IEC support only (CPU 113 xxS without mathematics processor) are available:

If...	Then
your application uses REAL arithmetic,	install the loadable EMUQ196. Note: The loadable is installed together with the EXEC-file QIECVxxx (installation in Concept EXECLoader).

Downloading Loadables for INTERBUS and IEC Support Only

The following loadables for IEC support are available:

If the CPU	Then
<ul style="list-style-type: none"> ● 113 02S ● 113 03S ● 213 04S ● 534 14 ● 434 12 is configured,	install the loadable ASUP196. Note: The ULEX196 loadable is automatically installed. The ASUP 196 loadable is only installed automatically on 32-bit CPUs. On 16-bit CPUs with Stripped EXEC (QIECVxxx.BIN), the ASUP196 loadable must be installed afterwards.
113 03 is configured	install the loadable pairing @1SE196 + @2IE196. The ULEX196 loadable is automatically installed.
213 04 is configured,	install the loadable pairing @1S7196 + @2I7196. The ULEX196 loadable is automatically installed.

Downloading Loadables for INTERBUS and LL984 Support Only

The following loadables for LL984 support are available:

If the CPU	Then
<ul style="list-style-type: none"> ● 113 02 ● 113 03 ● 213 04 is configured,	you can install the following loadables: <ul style="list-style-type: none"> ● ULEX196 ● @1S7196 + @2I7196 + ULEX196 Note: The ULEX196 loadable is automatically installed with this.
<ul style="list-style-type: none"> ● 534 14 ● 434 12 is configured,	the loadables ASUP196 and ULEX196 will be loaded automatically.

Segment manager

At a Glance

If a remote I/O st. (Drop) is configured, the sequence and method of processing the LL984 section can be defined in the dialog box **Segment manager**.
 When deleting (in the dialog box **I/O map**) a configured remote I/O st. (Drop), it is automatically deleted in the segment manager.

Mode of Functioning

Every I/O st. (Drop) is assigned a segment. It is therefore not permitted to enter fewer segments in the segment scheduler, than there are I/O st.s (Drops) configured in the I/O map. In the segment scheduler, the maximum segment numbers is by default set at 32.
 The configurator checks the agreement between the two dialogs and classifies the I/O st.s (Drops) in the segment scheduler. A window informs you which I/O stations (Drops) have been inserted.

Altering the segment processing sequence

The sequence for segment processing can be altered manually, in that the segment number or I/O st. number can be edited in the corresponding line. For the local I/O st. (Drop), 1 is entered in the first line of the dialog box in the columns **In stat.** and **Out stat.** automatically.¹
 If no sequence was defined, the segments are processed in ascending order.

Sorting criteria for additional I/O st.s

Recently added I/O st.s (Drops) are classified in the segment manager according to the following criteria:

If...	Then...
A new I/O st. is added,	it is automatically classified behind the last available line.
All determined segments are already in use,	the last segment is reused for the input of the new I/O st. (Drop), i.e. a segment number can be repeated, as the stations are differentiated.

Available methods for segment processing

When setting the segment manager, the following methods of processing can be selected:

Processing type	Meaning
Continuous	Cyclic processing
Controlled	Manually controlled processing
WDT reset	Reset watchdog timer
End of logic	End of processing

Note: If subprograms are to be used in LL984, the last configured segment cannot be processed in the segment manager. The type of solution must unconditionally be **End of logic**.

Advanced settings in the segment manager

With the "Controlled" type of processing, only the reference numbers 0x and 1x are authorized, which determines when the logic for the corresponding section is processed.

The field **In. stat.** and **Out stat.** allow the input of corresponding I/O st. numbers, which must be configured. If a **0** is entered, no input/output is served by this segment number.

I/O Map**Introduction**

In the I/O map, configure the I/O stations (drops) with the modules in use. Afterwards perform the I/O addressing and the parameterization of the configured modules.

Allocating Drops

Drop numbers can be allocated optionally except for the first one (from 2 to). The first drop number is automatically recognized as the local drop, and cannot be edited.

Configuring the Backplane Expander

The 140 XBE 100 00 module is necessary to expand the backplane. By doing this you can connect a second backplane, and gain 13 extra slots. The 140 XBE 100 00 module is mounted in both backplanes and, in addition, requires an independent power supply (power supply unit).

Expanded backplanes are configured in Concept in the first drop using slots 2-1 to 2-16.

A more detailed description about the configuration of expanded backplanes with the 140 XBE 100 00 module is given in the chapter *Backplane Expander Config*, p. 98.



CAUTION

The slot assignment of the 140 XBE 100 00 is not shown in the configurator, so a double assignment is possible.

You should take note of the hardware slots of the module and the power supply, and should not occupy these slots with other modules in the I/O map.

Failure to follow this precaution can result in injury or equipment damage.

Note: The flow of data via an expanded backplane is quicker than via the remote system.

Allocating the I/O Ranges

When allocating the I/O ranges the following references are allowed:

- 3x references for analog input modules
- 4x references for analog output modules
- 3x or 1x references for digital input modules
- 4x or 0x references for digital output modules
- 1x or 3x references for Expert modules (input)
- 0x or 4x references for Expert modules (output)

Note: The unique addressing is checked so that no addresses are occupied twice within the configuration.

Parameterization

Configured modules can be individually parameterized to determine the variable process conditioned settings.

Connection to other Network Systems

In addition to local and remote drops, links to other network systems can be established with configured coupling modules:

- Ethernet
- INTERBUS
- Profibus DP

See also the chapter entitled *Configuration of various network systems, p. 101* and *Configuration examples, p. 805*.

Read in Map

In the ONLINE mode of the stopped PLC, the hardware modules are listed in the I/O map and can be transferred as follows:

Step	Action
1	Open a project.
2	Open the PLC Configuration window.
3	Using the PLC Type menu command, open the PLC Type dialog and select the PLC type.
4	Connect the host computer to the PLC (Online → Connect...).
5	Open the I/O Map dialog (PLC Configuration → I/O Map).
6	Use the Edit command button to open the Local Quantum I/O station dialog.
7	Check the Poll check box. Response: The recognized modules are listed in the Read column in color.
8	Double click on the colored text boxes in the Read column. Response: The listed modules are transferred to the Module column.
9	Enter the address zone in the corresponding columns (In.Ref. , In End , Out Ref. , Out End).
10	After the hardware matching between the host computer and the PLC, the configuration can continue.

5.4 Optional configuration

At a Glance

Overview This section contains the description of the optional configuration.

What's in this Section? This section contains the following topics:

Topic	Page
Settings for ASCII Messages	91
Making Additional Functions Available in the Configurator	92
Data Exchange between Nodes on the Modbus Plus Network	93
Protecting Data in the State RAM before Access	94
Parameterize interfaces	94
Special Options	96

Settings for ASCII Messages

Introduction To create the ASCII messages, you are required first of all to set a mask, which contains the number of messages, the message area size and the ASCII ports. Once you have done that you can create the ASCII messages, which are then processed with the Ladder Logic programming language.

Precondition ASCII messages are only possible in the Quantum family, and can only be processed with the LL984 processing language.

Procedure To create the ASCII messages, you must first set the mask:

Step	Action
1	In the PLC Configuration → ASCII window, open the ASCII Setup dialog.
2	In the Total Messages text box specify a value from 1 to 999.
3	In the Message Area Size text box specify a value from 1 to 9999 bytes.
4	In the ASCII Ports text box specify an interface from 2 to 32.
5	Confirm your entries with the OK command button. Response: The settings are saved and the dialog is exited.
6	In the Project main menu open the ASCII Message Editor dialog (with the ASCII Messages... menu command).
7	Create the ASCII messages here, see also the description <i>ASCII Message Editor</i> , p. 543.

Making Additional Functions Available in the Configurator

Introduction

Additional functions can be used for the configuration, if they have previously been enabled or set in the **Select Extensions** dialog.

Activating Advanced functions/ Dialogs

By checking the check box or setting the Ethernet modules the corresponding menu commands are enabled and can be edited in the **PLC Configuration → ASCII** window.

The following functions/dialogs can be activated:

- Data protection
- Peer Cop
- Hot Standby
- Ethernet I/O-Scanner

Note: The available functions are dependent upon the configured CPU. Also see the online help "Select Extensions".

Specify Coupling Modules

Coupling modules must be configured in order to connect to other network systems. To do this, specify the number of modules in the corresponding list box, which are then available in the I/O map.

The following systems can be configured:

- TCP/IP Ethernet
- Symax-Ethernet
- MMS-Ethernet
- Profibus DP

Note: The maximum number of coupling modules depends upon the configured CPU. Also see the online help "Select Extensions".

Data Exchange between Nodes on the Modbus Plus Network

Introduction	<p>With a Modbus Plus (MB+) connection you can configure a PLC using the Peer Cop functionality, so that data exchange with another PLC is possible. In such a case, the Peer Cop takes data from a reference area within a "source" PLC and places this via the Modbus Plus (MB+) network into a determined reference range of a "destination" PLC. This operation is performed in the same identical way for each token rotation.</p> <p>Using the Peer Processor, input data from other nodes on the local network can be received by the user program. Likewise, output data from the user program can be transmitted to other nodes on the local network.</p> <p>The Peer Cop has two variants for data exchange:</p> <ul style="list-style-type: none">● global data exchange● specific data exchange
Precondition	<p>The Peer Cop menu command is only available if, in the Select extensions dialog the Peer Cop check box is checked.</p>
Global Data Exchange	<p>With global data exchange, the data sent from the current "source" PLC is received by all "destination" PLC devices in the Modbus Plus (MB+) network. Up to 64 destination devices can be reached in this way, which can each receive the data in 8 destination addresses of the State RAM.</p>
Specific Data Exchange	<p>With specific data exchange, data is sent from a selected "source" PLC to a selected "destination" PLC in the Modbus Plus (MB+) network. To do this, enter the respective addresses for the data exchange in a table at the corresponding source and destination nodes (1-64).</p> <p>The address must correspond to the MB+ node address on the back of the respective module. This address setting can be altered and must be specified before mapping. (See also hardware description)</p> <p>Select the node to be read or written according to the hardware configuration.</p>

Protecting Data in the State RAM before Access

Introduction	Output address ranges (coils and registers) can be protected by specifying the address from which writing is possible in the Data Protection dialog. All addresses before this are write-protected.
Precondition	The Data Protection menu command is only available if, in the Select Extensions dialog, the Data Protection check box is checked.
Entering Access Protection	This access protection operates in connection with "normal" data access, which happens externally via a Modbus or Modbus Plus interface. Access from the host computer out is in any case permitted and bypasses this protection mechanism.


Parameterize interfaces

At a Glance	Depending on their use in Concept, the following interfaces must be parameterized: <ul style="list-style-type: none">● ASCII interface● Modbus interface
Parameterize ASCII interface	For an ASCII message transmission, in the dialog box ASCII port settings the serial communication parameters for the port interfaces can be specified.

Note: The dialog box **ASCII port settings** is only available when the number of ASCII ports has been specified beforehand in the dialog box **ASCII set up**.

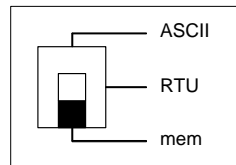
**Parameterize
Modbus
interface**

For a Modbus coupling, in the dialog box **Modbus port settings** the serial communication parameters of the port interface can be entered on the programming device, on a CPU and the NOM assemblies (Network Option Module).

	CAUTION
	Do not make any online changes since this will cause all Editors to close!
	The Modbus port settings should not be altered in Online mode, or else all Editors are automatically closed. Failure to follow this precaution can result in injury or equipment damage.

Note: The settings of a Modbus coupling in Concept only have an effect if the switch on the front of the assembly is at the lowest position (mem). In this case, the baud rate must be set at 19200 Bd.

Switch on the NOM



**Interface
parameterizing
with network
connections
between Modbus
and Modbus Plus**

A network connection between Modbus and Modbus Plus nodes can be made in the dialog box **Modbus port settings** by checking the check box **Bridge mode**.

Note: The settings are then only effective when the switch on the front of the assembly is in the mid-position (RTU).

Special Options

Introduction

In the **Specials** dialog you can configure special options:

- Battery coil
 - Timer register:
 - Time stamp for MMI applications (TOD)
 - Allow duplicate coils
 - Watchdog-Timeout (ms)
 - Time slice for online changes (ms)
-

Battery coil

You can specify an address of a coil, which shows the status of the battery. This assignment is used for battery monitoring. In this way, the weak battery can be replaced early to avoid a loss of data.

Timer Register:

The content of the time register is incremented every 10 ms and has a free value between 0000 and FFFF hex.

Time for MMI applications (Date/Time)

This time stamp is only intended for a MMI application. Eight registers are reserved for setting the clock.

The TOD input (Time of Day) is in the American format:

4xxxx	Control register	
	Discrete 1 (MSB) Discrete 2 Discrete 3 Discrete 4	1 = set clock values 1 = read clock values 1 = preset discrete 1 = error discrete
4xxxx+1	Day of week (1 - 7)	
4xxxx+2	Month (1 - 12)	
4xxxx+3	Day (1 - 31)	
4xxxx+4	Year (00 - 99)	
4xxxx+5	Hours (0 - 23)	
4xxxx+6	Minutes (0 - 59)	
4xxxx+7	Seconds (0 - 59)	

Allow Duplicate Coils You can assign several outputs to a coil. To do this, check the check box, and specify the first address to which several outputs can be allocated in the **First Coil Address**: text box.

Note: This function is unavailable with the Momentum PLC family.

Watchdog Timeout (ms*10) You can set a pulse supervision for the user program by entering a numerical value of between 2 and 255 (ms). As soon as there are no count pulses within the specified time, an error message will appear.

Time Slice for Online Changes (ms) You can set a time supervision for the communication between the nodes by entering a numerical value between 3 and 30 (ms). As soon as there is no communication within the specified time, an error message will appear.

5.5 Backplane Expander Config

At a glance

Introduction This chapter describes the function and configuration of the backplane expander.

What's in this Section? This section contains the following topics:

Topic	Page
Generals to Backplane Expander	99
Edit I/O Map	99
Error handling	100

Generals to Backplane Expander

Introduction The Quantum backplane expander provides a single backplane expansion to a local drop or a RIO drop through the 140 XBE 100 00 module.

Function description The module connects two Quantum backplanes (primary and secondary) through a custom cable and support all data communication between the backplanes. Each backplane requires a 140XBE10000 module that occupy a single slot and requires its own power supply.

Procedure at an Error The backplane expander is designed in the way that if it is not installed or improperly connected, it will not effect the functionality of the primary rack. Only the backplane expander installed and connected properly, the both racks are then able to communicate and controlled by prime CPU or RIO drop controller.

Edit I/O Map

Requirements Currently only Quantum controllers support backplane expander. Primary rack contains the CPU or RIO drop controller and is allowed to config all type of additional modules up to the physical slot address limitation. All I/O modules can be also added to the secondary rack. However, option modules, such as NOMs, NOEs and CHSS must reside in the primary rack.
To place a module in proper rack, it is necessary to add an extra attribute in the I/O module database to specify that the module is available only for the primary or secondary or both.

Configuration in I/O Map

Exist Quantum local drop or RIO drop only support one rack up to sixteen slots. With backplane expander, it is extended as if the drop support two racks, and each has sixteen slots. By clicking at the button ... on **Module** column, all modules available to the rack clicked (primary or secondary) will show in the module selection dialog that can be selected and assigned to the current slot.

Each rack requires a 140 XBE 100 00 module to make backplane expander work properly.

Note: The 140 XBE 100 00 module does not have a personality code and therefore can not be recognized by the Concept.

The module will just look like an unfilled slot in the Concept I/O map. If any module is configured in the secondary rack, it is user's responsibility to ensure there is one slot in each rack that is reserved for 140 XBE 100 00 module and all hardware are connected properly.

Error handling

Introduction

The validate processes for the primary rack will be applied to the secondary rack too, such as duplicate reference, missing input or output reference, etc. Besides existing regular validation, traffic cop will do some special check for the backplane expander.

No reserved slot for 140 XBE 1000 00

If any module is found in the secondary rack and there is no empty slot left in either of racks when user trying to exit the rack editor dialog, an error message will be displayed: "There must be one empty slot reserved for 140 XBE 100 00 module in each rack to make backplane expander work." The rack editor dialog will then not be closed.

Special module in secondary rack

To prevent any special module (such as, NOE, CHS, etc) being added to the secondary rack, rack editor dialog do not allow to cut/copy these head modules. It will also check module personalities before user try to do any paste operation. If some unsupported module for the secondary rack is found, an error message will be displayed: "The buffer contains some module that can not reside in the secondary rack." The paste operation will be aborted.

5.6 Configuration of various network systems

At a Glance

Overview This section contains the description of the configuration of various network systems.

What's in this Section? This section contains the following topics:

Topic	Page
Configure INTERBUS system	102
Configure Profibus DP System	103
Configure Ethernet	104
RTU extension	105
Ethernet I/O Scanner	106
How to use the Ethernet / I/O Scanner	109

Configure INTERBUS system

At a Glance

The configuration of the INTERBUS system can take place within the PLC families of Quantum and Atrium.

INTERBUS configuration with Quantum

With the Quantum family the coupling of a remote bus takes place in a Quantum I/O station (Drop). To do this, the INTERBUS Master NOA 611 00 must be configured and parameterized in the CMD tool (Configuration Monitoring and Diagnostic Tool).

See also Configuration example 4 (See *Quantum Example – INTERBUS Control*, p. 835).

INTERBUS configuration with Atrium

With the Atrium family, the coupling of the remote bus takes place via the master assembly 180 CCO 121 01, 180 CCO 241 01 or 180 CCO 241 11 in this way, the INTERBUS Master CRP 660 0x is automatically inserted into the local I/O station (Drop). The INTERBUS I/O station (Drop) nodes are configured in the CMD tool (Configuration Monitoring and Diagnostic tool), saved as a *.SVC data file and imported to Concept. After the import into the I/O map the configuration can be changed afterwards in Concept.

See also Configuration example 9 (See *Atrium Example – INTERBUS Controller*, p. 877).

Configure Profibus DP System

Introduction The configuration of the Profibus DP system can take place within the PLC families of Quantum and Atrium.

Profibus DP Configuration with Quantum With the Quantum family the connection to the Profibus DP system takes place in a Quantum drop. To do this, you must first of all set the number of bus controllers (CRP 811 00) used in the **Select Extensions** dialog. The modules then appear in the list box of the **I/O Module Selection** dialog and can be inserted into the I/O map. The configuration of the Profibus DP node is created in the SyCon configuration tool, saved as a *.CNF file and transferred directly to Concept. However, the configuration (*.CNF) can be imported to Concept at a later time.

Importing the Profibus DP Configuration To import the configuration (*.CNF) to Concept, proceed as follows:

Step	Action
1	In the PLC Configuration window, open the I/O Map dialog.
2	Select the drop and use the Edit... command button to open the Edit Drop dialog.
3	Double click on the ... text box in the Module column. Response: The I/O Module Selection dialog is opened.
4	In the Specials column, select the CRP-811-00 module, and press the OK command button. Response: The CRP-811-00 module is entered in the I/O map.
5	In the Edit Drop dialog, select the line of the mapped bus controller (CRP-811-00) and press the Parameters command button. Response: The Edit CRP-811-00 (Profibus DP) dialog is opened.
6	Using the Import... command button, open the Select Import File window.
7	To import, specify the path of the CNF file, and press the OK command button. Response: The Profibus DP configuration is entered in the Concept I/O map. Note: After the Profibus DP nodes are entered into Concept, the reference ranges for all modules and diagnostic data must be edited later.

Configuration Example An example of configuration is given in Example 11 (See *Quantum Example - Profibus DP Controller, p. 849*).

Configure Ethernet

Introduction An Ethernet bus system can be configured within the following PLC families:

- Quantum
- Atrium
- Momentum

Precondition In order to connect to the Ethernet bus system, a PCI network card must be available in the host computer. Afterwards the Ethernet interface needs to be parameterized and the drivers that are provided on CD need to be installed (*Configure Ethernet, p. 896*).
 After the Ethernet module has been slotted into the central backplane, the internet address, subnet mask, gateway and frame type can be allocated by the network administrator.

Configuration with Quantum The procedure for Ethernet configuration in Concept is as follows:

Step	Action
1	In the PLC Configuration window, open the Select Extensions dialog.
2	Enter the number of Ethernet modules (NOE) in the text boxes. Response: The modules then appear in the list box in the I/O Module Selection dialog and can be inserted into the I/O map.
3	In the PLC Configuration window, open the Ethernet I/O Scanner dialog, in which you enter the information from the network administrator (Internet address, subnet mask, gateway, frame type).
4	In the Online main menu, open the Connect to PLC dialog (menu command Connect...).
5	In the Protocol Type list box, select the option TCP/IP , and in the IP address or DNS Hostname text box, enter the address of the TCP/IP card.
6	After programming, in the Online main menu, open the Download Controller dialog (menu command Download...), and click on the Download command button. Response: A message appears, asking whether you would like to start the PLC.
7	Before you confirm the message with the Yes command button, the display "link" must appear on the Ethernet module.

Error Action After configuration, only start the PLC once the display "link" has appeared on the Ethernet module. If this is not the case, withdraw the Ethernet module from the central backplane and then slot it in again. If the display "link" is still not shown, there must be a serious error.

Available Ethernet Modules

The maximum number of NOE modules is dependent upon the configured CPU (select in the **PLC Selection** dialog):

CPUs	Number of NOE modules
113 02/S/X	0 - 2
113 03/S/X	0 - 2
213 04/S/X	0 - 2
424 0x/X	0 - 6
434 12	0 - 6
534 14	0 - 6

Configuration with Momentum

The configuration of the Ethernet bus system with Momentum is described in the section *Momentum Example - Ethernet Bus System*, p. 895.

RTU extension**Requirements**

To make the RTU menu command available you have to choose a Compact CPU with LL984 programming language in the **PLC Selection** dialog.

CTS-/RTS-Delay

In this dialog you can set time delay for CTS or RTS independently for Comm port 1 of your Compact PLC. This feature allows modem communications with radios that require longer time frames. The delay time range is 0 ... 500 ms using 10 ms units. Enter the time delays your require.

Secured Data Area (SDA)

This feature allows you to configure an area in RAM that is secured from being overwritten. Secured Data Area (SDA) is a block of the Compact PLCs RAM that is set aside as 6x data space. The SDA can only be written to by specific functions that require secured data storage. General purpose Modbus commands, builtins, can not write to the SDA. Modbus Read (function 20) is able to read from the SDA, Modbus Write (function 21) is not able to write to the SDA. The SDA size range is 0 ... 128 K words using only 1 K word blocks. Enter the size your require. Refer to the applicable user manual for the specific function for the required SDA size. For example, for Gas Flow, refer to the "Starling Associates Gas Flow Loadable Function Block" User Guide (890 USE 137 00).

PLC Login Password Protection

For the description of password protection, refer to section *Set PLC Password*, p. 589.

Ethernet I/O Scanner

Introduction

This function is for the following Quantum modules available:

- 140-NOE-211-x0
- 140-NOE-251-x0
- 140-NOE-771-00
- 140-NOE-771-10

This function is for the following Momentum modules available:

- 171-CCC-960-20
- 171-CCC-980-20
- 171-CCC-980-30
- 171-CCC-960-30

Ethernet address and I/O scanning parameters can be modified using the **Ethernet / I/O Scanner** dialog box. From the **PLC Configuration** window, select **Ethernet / I/O Scanner**. This menu option will only be available if you have selected an M1 Processor Adapter with an Ethernet port or have Quantum TCP/IP Ethernet modules (NOE) as specified above.

This section describes how to configure the Ethernet port, including IP address, other address parameters and I/O scanning.

Ethernet Configuration Options

The Ethernet / I/O Scanner screen offers three options for configuring the Ethernet port on an M1 Processor Adapter:

Configuration options	Meaning
Specify IP Address	This is the default option. It allows you to type the IP address, gateway and subnet mask in the text boxes in the upper righthand corner of the screen.
Use Bootp Server	Click this radio button if you want the address parameters to be assigned by a Bootp server. If you select this option, the address parameter text boxes in the upper righthand corner of the screen will be grayed out. They will not display the actual address parameters.
Disable Ethernet	Click this radio button if you want to disable the Ethernet port. Disabling the port will reduce the scan time for the Processor Adapter.

Setting Ethernet Address Parameters

If you choose to specify the IP address, you should complete all four text boxes in the upper righthand corner of the dialog box:

Parameters	Meaning
Internet Address	Type a valid IP address in the Internet Address text box (for example: 1.0.0.1). Caution: POTENTIAL FOR DUPLICATE ADDRESSES! Obtain a valid IP addresses from your system administrator to avoid duplication. Failure to observe this precaution can result in injury or equipment damage.
Gateway	Consult your system administrator to determine the appropriate gateway. Type it in the Gateway text box.
Subnet Mask	Consult your system administrator to obtain the appropriate subnet mask. Type it in the Subnet Mask text box (for example: 255.255.255.0).
Frame Type	For NOE there is an additional Frame Type field. Your two possible choices are ETHERNET II or IEEE 802.3

Configuring I/O

Once the Ethernet port address parameters have been set, you may assign parameters for I/O scanning.

The text box **Master Module (Slot)** contains the Module type that you have configured for Ethernet communications. In the case of the Momentum Ethernet controller the slot will always be number 1, and the configured module type is displayed in the variable dialog field. If you are configuring a NOE in a standard rack the slot number assigned in the I/O Map will be displayed along with the module type. Until the I/O Map is completed this test field will indicate "Unassigned". In instances where more than one NOE is configured the I/O Scan parameters reflect the unit currently in the dialog box from which you can select the additional unit by activating the Pulldown list.

The text field **Health Block (1x/3x)** is only available by using the 140-NOE-771-00. The health timeout is used for setting the health bit. If the response arrives before the end of the HealthTimeout period, the health bit is set; otherwise it is cleared. If the Health Timeout is zero, the health bit is set to true once communications are established, and it is never cleared.

Note: The configuration of the health block, refer to chapter 5.2 in the user guide Quantum NOE 771 x0 Ethernet Modules, model no. 840 USE 116 00.

The text box **Diagnostic Block (3x/4x)** is only available by using the 140-NOE-771-00 and allows you to define the starting register of a number of bits which are used for diagnostic. The block can be specified in either 3x or 4x registers. For more information, refer to the user guide Quantum NOE 771 x0 Ethernet Modules, model no. 840 USE 116 00.

I/O Scanner Configuration table:

Column	Description
Slave IP Address	Type the IP address of the slave module in this column (for example: 128.7.32.54). This address will be stored in a pulldown menu, so that you may use it in another row by clicking on the down arrow and selecting it.
Unit ID	If the slave module is an I/O device attached to the specified slave module, use the Unit ID column to indicate the device number. The Unit ID is used with the Modbus Plus to Ethernet bridge to route to Modbus Plus networks.
Health Timeout	Use this column to specify the length of time in ms to try the transaction before timing out. Valid values are 0 ... 50 000 ms (1 min). To avoid timing out, specify 0.
Rep Rate	Use this column to specify how often in ms to repeat the transaction. Valid values are 0 ... 50 000 ms (1 min). To repeat the transaction continually, specify 0.
Read Ref Master	Use the read function to read data from the slave to the master. This column specifies the first address to be read (for example: 400001).
Read Ref Slave	Use the read function to transfer data from the slave to the master. This column specifies the first address of up to 125 to read to (for example: 400050).
Read Length	Use the read function to read data from the slave to the master. This column specifies the number of registers to read (for example: 20).
Write Ref Master	Use the write function to write data from the master to the slave. This column specifies the first address to write (for example: 400100).
Write Ref Slave	Use the write function to write data from the master to the slave. This column specifies the first address of up to 100 to write to (for example: 400040).
Write Length	Use the write function to write data from the master to the slave. This column specifies the number of registers to write (for example: 40).
Description	You can type a brief description (up to 32 characters) of the transaction in this column.

Note: You may include read and write commands on the same line.

How to use

For more information about how to use the Ethernet / I/O Scanner dialog see section *How to use the Ethernet / I/O Scanner, p. 109.*

How to use the Ethernet / I/O Scanner

Introduction This section describes how to complete your Ethernet I/O configuration using the **Copy**, **Cut**, **Paste**, **Delete** and **Fill Down** buttons.

Copy and Paste To save time when typing similar read and write commands, you may copy and paste entire rows within your configuration:

Step	Action
1	Select the row you want to copy by clicking on the row number at the far left.
2	Click the Copy button above the I/O configuration list.
3	Select the row where you would like to paste the data (by clicking on the row number at the far left).
4	Click the Paste button.

Cut and Paste To move a row within the configuration list, follow the direction:

Step	Action
1	Select the row you want to move by clicking on the row number at the far left.
2	Click the Cut button above the I/O configuration list.
3	Select the row where you would like to paste the data (by clicking on the row number at the far left).
4	Click the Paste button. Note: Multiple rows may be cut/copy and pasted. The number of rows actually pasted is limited by the number of rows selected. For example if you copy 10 rows to the clipboard, then select an area of 6 rows to past, only the first six rows of clipboard data is pasted.

Delete To delete a row within the configuration list, follow the direction:

Step	Action
1	Select the row you want to delete by clicking on the row number at the far left.
2	Click the Delete button above the I/O configuration list. Note: Multiple rows may be deleted.

Fill down

To copy part of any row to the next row or to a series of adjoining rows, use the **Fill Down** button, following the steps in the table

Step	Action
1	Use your mouse to select the data you would like to copy and the cells you would like to copy it to. Note: You must select one contiguous block of cells, with the data to be copied in the first row. You cannot select two separate blocks.
2	Click the Fill down Button. Result: The data from the first row is copied to the selected cells in the defined block.

NOE Ethernet modules

In this dialog the NOE Ethernet modules 140 NOE 211 x0, 140 NOE 251 x0 and 140 NOE 771 10 are parameterized (in the **Ethernet Configuration** area).

In this dialog the NOE Ethernet module 140 NOE 771 00 is parameterized and addressed (in the **I/O Scanner Configuration** area).

For the followings modules you receive an function description:

- 140 NOE 211 x0 (with Web Embedded Server)
- 140 NOE 211 x0 (with Ethernet TCP/IP)
- 140 NOE 251 x0 (with Web Embedded Server)
- 140 NOE 251 x0 (with Ethernet TCP/IP)
- 140 NOE 771 00
- 140 NOE 771 10

Momentum Ethernet modules

In this dialog the Momentum Ethernet modules are addressed (in the **I/O Scanner Configuration** area).

For the followings modules you receive an function description:

- 171 CCC 980 30 IEC
- 171 CCC 980 30 984
- 171 CCC 980 20 984
- 171 CCC 960 30 IEC
- 171 CCC 960 30 984
- 171 CCC 960 20 984

5.7 Quantum Security Settings in the Configurator

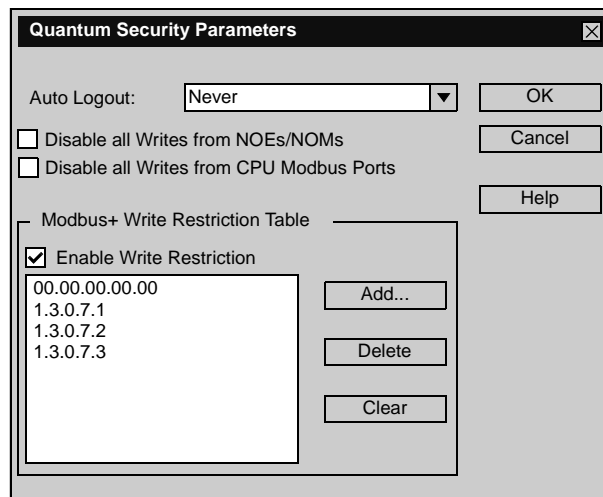
Quantum Security Parameters

Introduction

Various security parameters can be defined in the configuration of the Quantum CPUs 140 434 12A and 140 534 14A which are indicated in the log file *.LOG. This guarantees secure process documentation which includes the logging with the automatic logout, write access of NOEs/NOMs on the PLC as well as limited participants (max. 12) for network write access.

The definition of the security parameters can be found in dialog **Configuration** → **Quantum Security Parameters**.

Dialog **Quantum Security Parameters**:



Requirements

The security parameters are only available if the following conditions have been met:

- Supervisor Rights (see Concept under **Help** → **About...** → **Current User:**)
- only with CPUs 140 CPU 434 12A and 140 CPU 534 14A

Automatic Logout

The automatic logout procedure logs a user out as soon as a predefined time limit (max. 90 minutes) is reached with no activity on the connection. This could be a lack of read or write activity from the programming device to the PLC for example. The **Never** setting disables this function, i.e. automatic logout cannot occur.

Disable all Writes from NOEs/NOMs

By disabling all writes from NOEs/NOMs to the PLC, all write instructions are ignored by the CPU and responded to with an error message.

Note: MSTR-Read-Operations are not executed when the check box **Disable all Writes from NOEs/NOMs** is checked. (This also means the error state of the MSTR block shows no error!)

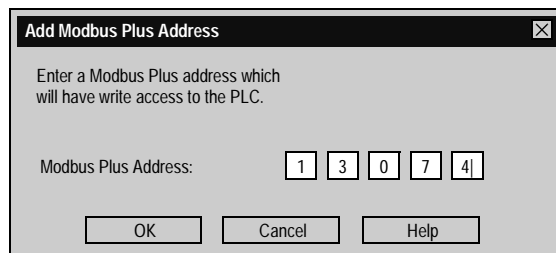
Disable all Writes from CPU Modbus Ports

To disable writes from the Quantum CPU Modbus connections, check the **Disable all Writes from CPU Modbus Ports** check box.

Limited Write Access on the Modbus Plus Network

A restricted number of participants that have access to the PLC can be configured for the Modbus Plus network. A maximum of 12 participants are allowed, the participant address of the programming device is automatically entered in the participant list and cannot be deleted.

Dialog **Add Modbus Plus Address** (press **Add...**)



Add Modbus Plus Address [X]

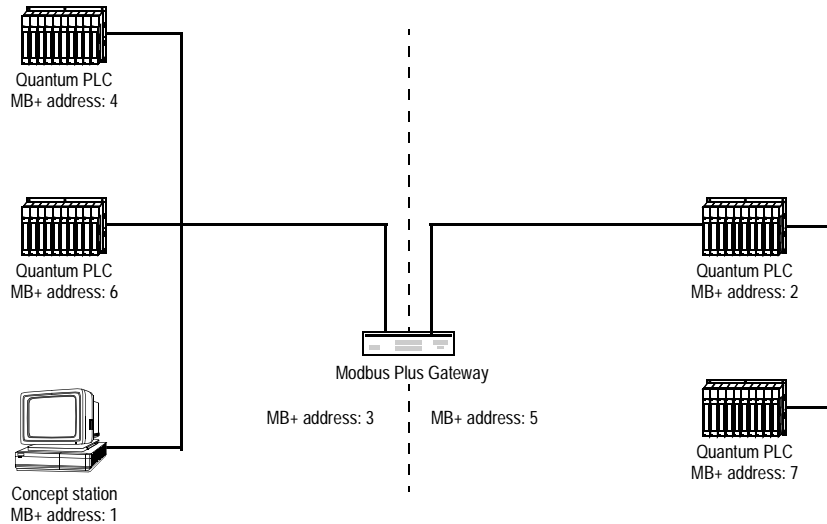
Enter a Modbus Plus address which will have write access to the PLC.

Modbus Plus Address: 1 3 0 7 4

OK Cancel Help

Examples of Modbus Plus Paths

Modbus Plus Network:



The address must be entered from the point of view of the PLC which is ready to receive to the sender and therefore begins with the first gateway or the next PLC. This depends whether the sender and receiver are in the same Modbus Plus segment (no bridges/gateways), or if the sender and receiver are in different segments (separated by one or more bridges/gateways).

Example 1:

Concept (MB+ address 1) writes to PLC 6. There are no bridges or gateways between the two nodes. Therefore the entered address looks like this: 1 or 1.0.0.0.0

Example 2:

PLC 2 (MB+ address 2) writes to PLC 6. A gateway is between the nodes (MB+ address 3). Therefore the entered address looks like this: 3.2.0.0.0

Note: Only the first Modbus Plus address can be detected by the PLC, i.e. as soon as the first bridge or gateway address is recognized, all devices in the network behind bridge or gateway have write access to the PLC. This means that PLC 7 can also write to PLC 6 in our example (Address: 3.7.0.0.0).

Main structure of PLC Memory and optimization of memory

6

At a Glance

Overview

This Chapter describes the main structure of the PLC Memory and the optimization of the memory with the different PLC families.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
6.1	Main structure of the PLC Memory	117
6.2	General Information on Memory Optimization	118
6.3	Memory Optimization for Quantum CPU X13 0X and 424 02	122
6.4	Memory Optimization for Quantum CPU 434 12(A) and 534 14(A)	136
6.5	Memory optimization for Compact CPUs	147
6.6	Memory optimization for Momentum CPUs	157
6.7	Memory optimization for Atrium CPUs	163

6.1 Main structure of the PLC Memory

General structure of the PLC Memory

At a Glance	<p>In principle, the memory of a PLC consists of three parts:</p> <ul style="list-style-type: none">● the memory for the Exec file,● the state RAM and● the program memory.
Memory for the EXEC file	<p>The EXEC file contains the operating system and one or two runtime systems (IEC and/or LL984) for operating the user programs.</p>
State RAM	<p>The state RAM can be divided into different zones:</p> <ul style="list-style-type: none">● the used 0x, 1x, 3x and 4x references,● a reserve for further 0x, 1x, 3x and 4x references,● possibly an extended memory zone for 6x references.
Program Memory	<p>The program memory can be divided into different zones:</p> <ul style="list-style-type: none">● the I/O map etc.,● a reserve for extensions,● the ASCII messages (if used), the Peer Cop configuration (if used), the Ethernet configuration (if used) etc.,● a reserve for extensions,● the IEC loadables (if required),● the Global Data, consisting of the Unlocated Variables,● the IEC program memory with the program codes, EFB-Codes and program data (section data and DFB instance data),● possibly the ULEX loadable for INTERBUS or other loadables,● the LL984 program memory.

6.2 General Information on Memory Optimization

Introduction

Overview

This Section contains general information on memory optimization.

What's in this Section?

This section contains the following topics:

Topic	Page
Possibilities for Memory Optimization	119
PLC-Independent	119

Possibilities for Memory Optimization

Description	<p>The possibilities for memory optimization are partly dependent on the PLC family and CPU used:</p> <ul style="list-style-type: none">● <i>PLC-Independent</i>, p. 119● <i>Memory Optimization for Quantum CPU X13 0X and 424 02</i>, p. 122● <i>Memory Optimization for Quantum CPU 434 12(A) and 534 14(A)</i>, p. 136● <i>Memory optimization for Compact CPUs</i>, p. 147● <i>Memory optimization for Momentum CPUs</i>, p. 157● <i>Memory optimization for Atrium CPUs</i>, p. 163
--------------------	--

PLC-Independent

Introduction	<p>There are 3 PLC-independent possibilities for memory optimization:</p> <ul style="list-style-type: none">● <i>Optimize State RAM for 0x and 1x References</i>, p. 119● <i>Only Download Required Loadables</i>, p. 120● <i>Optimize Expansion Size</i>, p. 121
---------------------	---

Optimize State RAM for 0x and 1x References

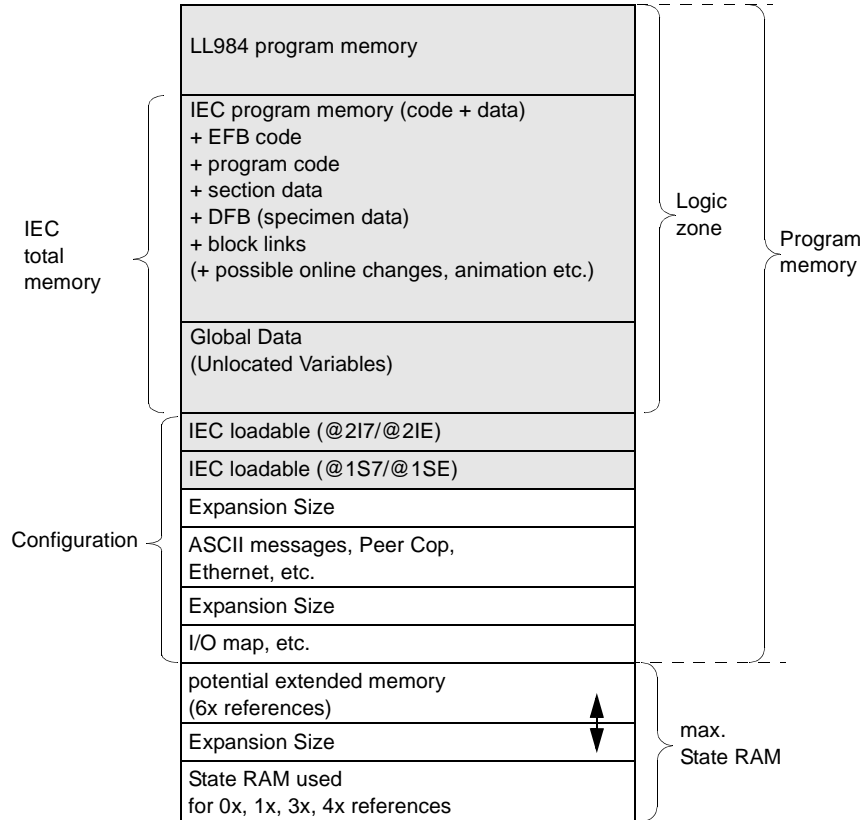
The state RAM contains the current values of the 0x, 1x, 3x and 4x references.

Even if the state RAM zone is outside the program memory zone, the size of the state RAM for 0x and 1x references influences the size of the program memory. Therefore, do not select a state RAM zone that is too large. In theory, the procedure only needs as many 0x and 1x references as the hardware requires. However, you will require a somewhat larger number of references if the I/O map is to be extended. It is advisable to be generous with the number of references during the creation phase of the user program when frequent changes are still being made. At the end of the programming phase, the number of these references can be reduced in order to create more space for the user program.

The settings for the 0x-, 1x-references can be found in **Project** → **PLC Configurator** → **PLC Memory Partition**.

In this dialog box, there is an overview of the size of the occupied state RAM zone and the percentage of the maximum state RAM that this represents.

Optimize state RAM for 0x, 1x, 3x and 4x references:



Only Download Required Loadables

All the installed loadables are downloaded into the program memory zone and occupy space. Therefore, only install those loadables which you really need (related topics *Loadables*, p. 84).

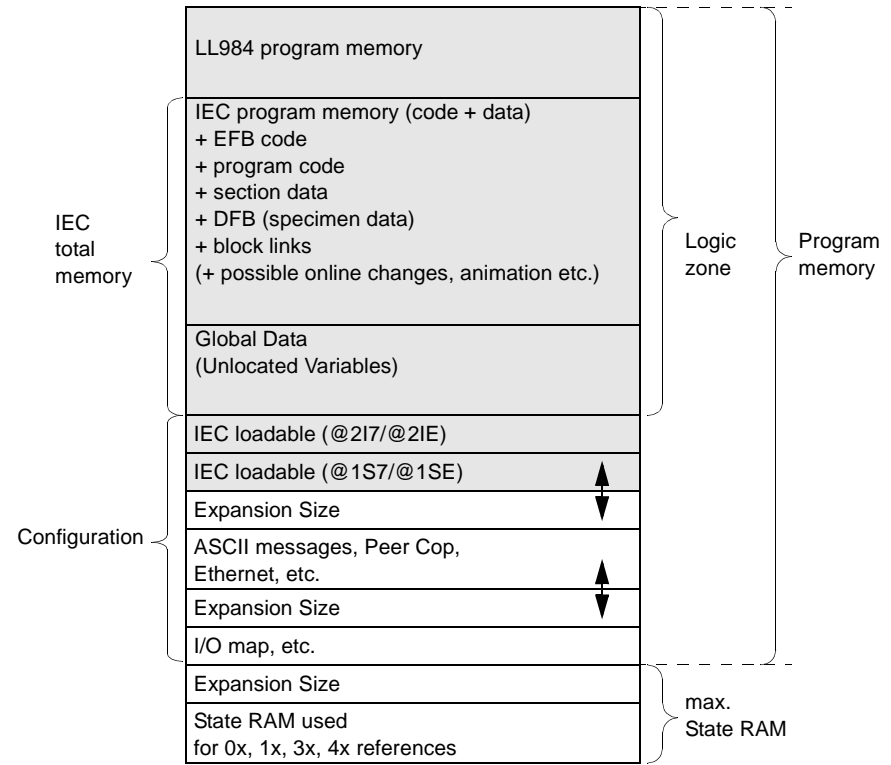
The memory space occupied by the installed loadables is displayed in the **Loadables** dialog box under **Project** → **PLC configurator** → **Used Bytes**. This information is calculated from the size of the loadable files and from the memory size assigned to the loadables.

Optimize Expansion Size

Each time, there is the possibility to reserve memory space for later expansion in the mapping zone (I/O map) and in the configuration expansion zone (Peer Cop). This memory space is necessary if e.g. the I/O map or the Peer Cop settings should be changed online. It is advisable to overestimate the reserves during the installation phase of the user program, that is, when modifications are often being made. At the end of the programming phase the reserves may be reduced again, to provide more space for the user program.

The settings for the mapping reserves are found in **Project → PLC Configurator → I/O Map → Expansion Size**. The settings for the Peer Cop reserves can be found in **Project → PLC Configurator → Config. Extensions → Select Extensions → Peer Cop → Expansion Size**.

Optimize Expansion Size



6.3 Memory Optimization for Quantum CPU X13 0X and 424 02

Introduction

Overview This Section describes the memory optimization for the Quantum CPUs CPU X13 0X and CPU 424 02.

What's in this Section? This section contains the following topics:

Topic	Page
General Information on Memory Optimization for Quantum CPU X13 0X and 424 02	123
Selecting Optimal EXEC File	125
Using the Extended Memory (State RAM for 6x references)	129
Harmonizing the IEC Zone and LL984 Zone	131
Harmonizing the Zones for Global Data and IEC Program Memory	133

General Information on Memory Optimization for Quantum CPU X13 0X and 424 02

Logic Memory

The program memory zone, in which the user program is located, is called the logic zone. This zone therefore determines the maximum size of your user program.

The current size of the logic zone is displayed under **Project** → **PLC Configuration** in the configurations overview in the **PLC** zone. The entry for the memory size is given in Nodes for LL984 (1 node equals 11 bytes) and in kilobytes for IEC.

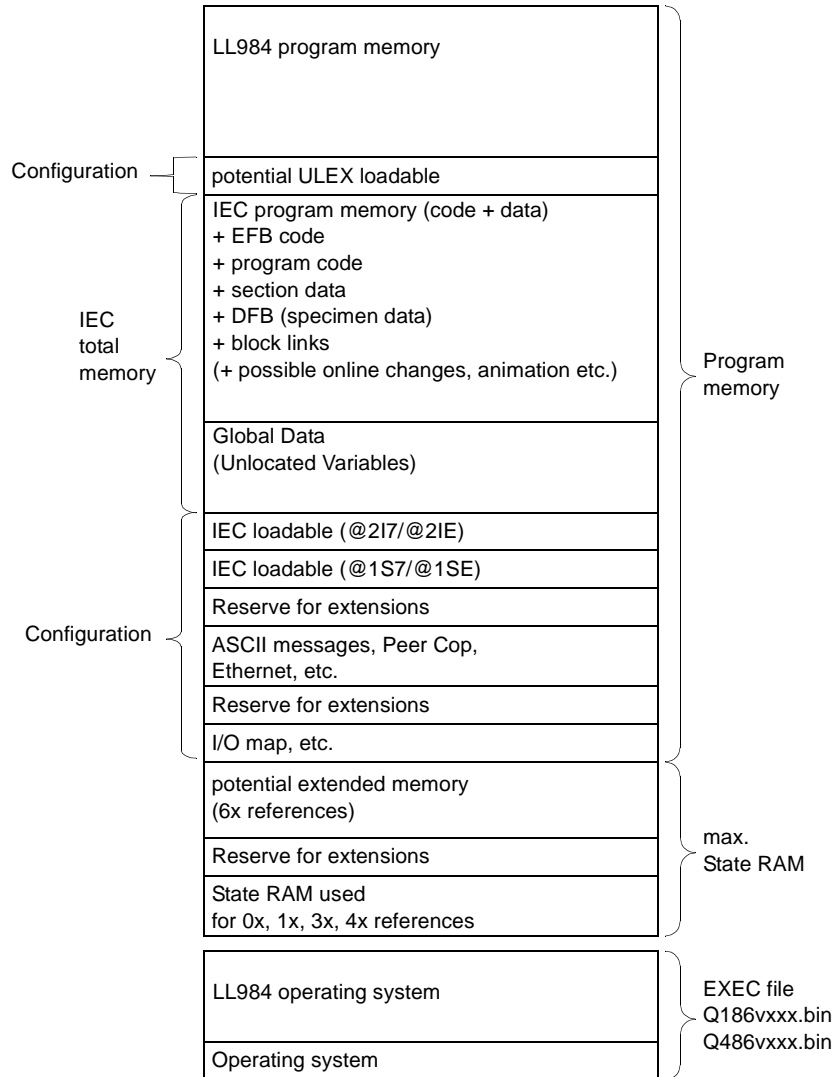
Optimizing the Logic Memory

You have various possibilities for optimising the logic memory to suit your requirements:

- *Selecting Optimal EXEC File, p. 125*
- *Using the Extended Memory (State RAM for 6x references), p. 129*
- *Harmonizing the IEC Zone and LL984 Zone, p. 131*
- *Harmonizing the IEC Zone and LL984 Zone, p. 131*

Note: Also note the PLC-independent possibilities for memory optimization (See *General Information on Memory Optimization, p. 118*).

Structure of the CPU X13 0X memory (simplified representation):



Selecting Optimal EXEC File

Introduction

The simplest and most basic option is to download the optimal EXEC file for your requirements onto the PLC (see also *Installation Instructions*).

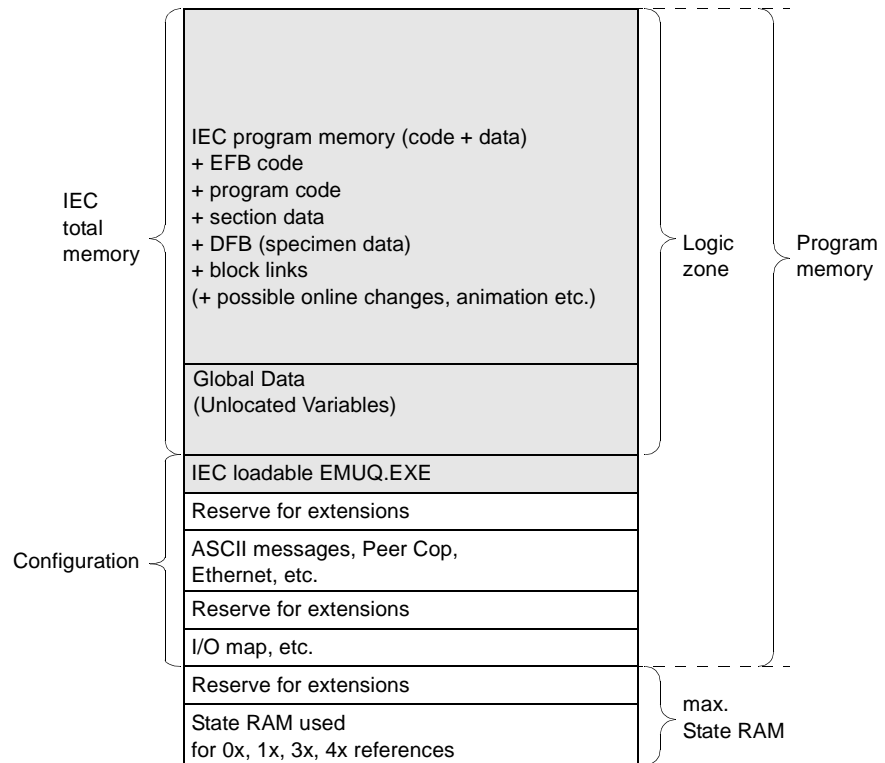
Depending on which EXEC file you select, zones will be reserved in the program memory of the PLC for IEC and/or LL984 programs. Therefore, if you install a 'combined EXEC file' and then only use one of the two language types in the user program, the program memory will not be used optimally.

Therefore, decide which languages you want to use:

- *Exclusive Use of IEC, p. 126*
 - *Exclusive Use of LL984, p. 127*
 - *Joint Use of IEC and LL984, p. 128*
-

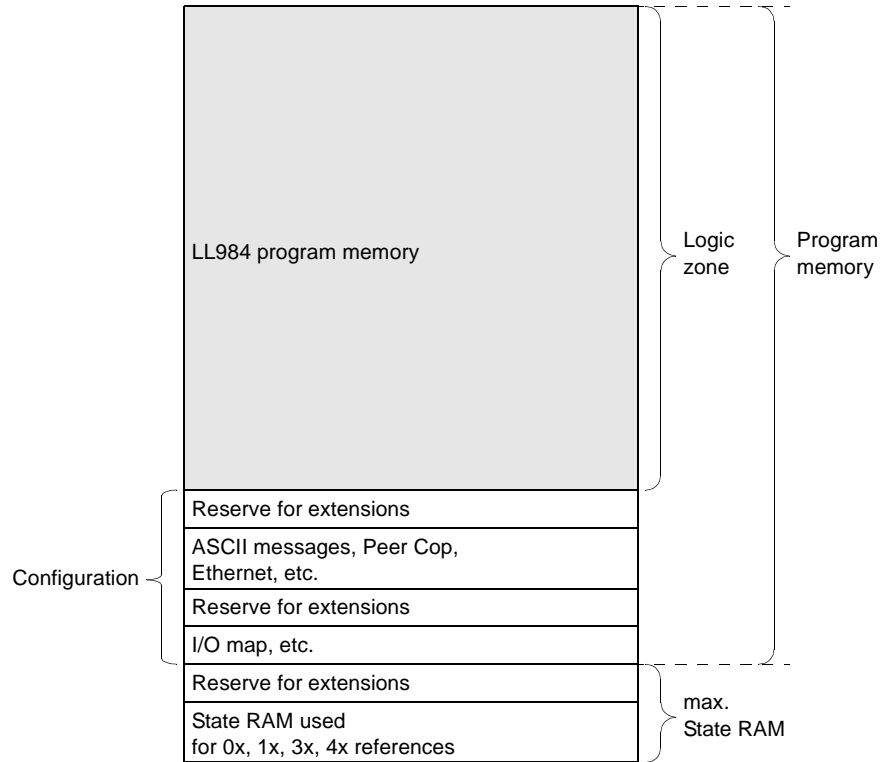
Exclusive Use of IEC

If you want to use IEC exclusively, download the EXEC file "QIEC_xxx.bin" (not available for CPU 424 02). Since this EXEC file does not contain an operating system, you have to download the IEC runtime system onto the PLC in the form of a loadable (EMUQ.exe) (related topics *Loadables*, p. 84). The loadable is downloaded into the program memory zone and takes up memory space. Structure of the CPU X13 0X memory with exclusive use of IEC:



Exclusive Use of LL984

If you want to use LL984 exclusively, download the EXEC file "Q186Vxxx.bin" for a CPU X13 0X and the EXEC file "Q486Vxxx.bin" for a CPU 424 02.
 Structure of the CPU X13 0X memory with exclusive use of LL984:

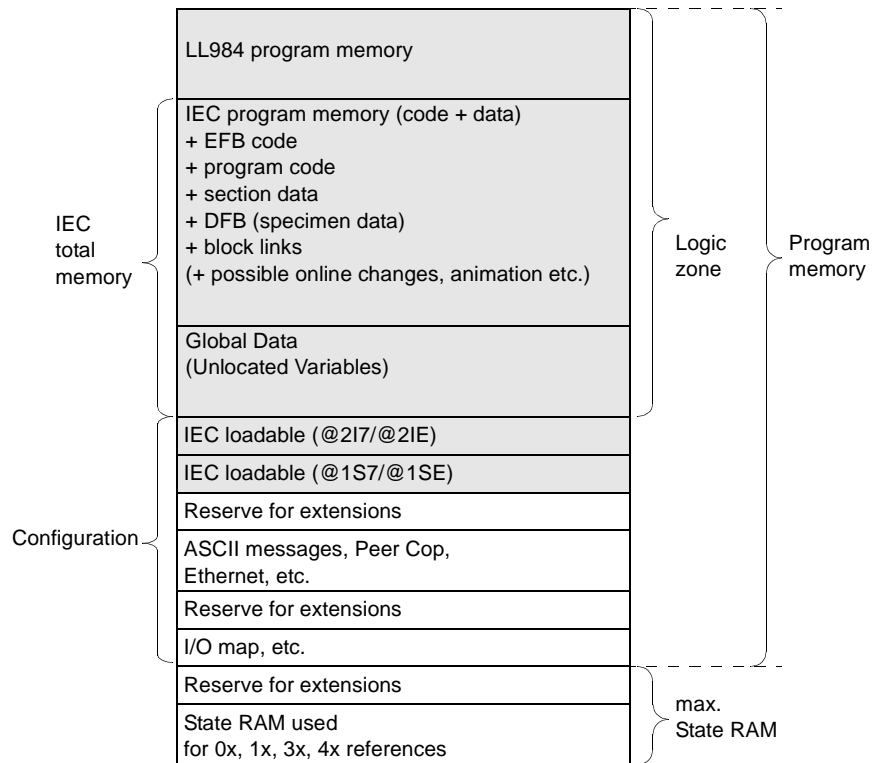


Joint Use of IEC and LL984

If joint use of IEC and LL984 is required, download the EXEC file "Q186Vxxx.bin" for a CPU X13 0X and the EXEC file zone "Q486Vxxx.bin" for a CPU 424 02. Since these EXEC files only contain the LL984 operating system, you have to download the IEC operating system onto the PLC in the form of loadables (@2I7/@2IE or @1S7/@1SE) (see also *Loadables*, p. 84). Both loadables will be downloaded into the program memory zone and occupy memory space.

Note: Joint use of IEC and LL984 is not possible with the CPU 113 02 because its memory is too small for this application.

Structure of the CPU X13 0X memory with joint use of IEC and LL984:



Using the Extended Memory (State RAM for 6x references)

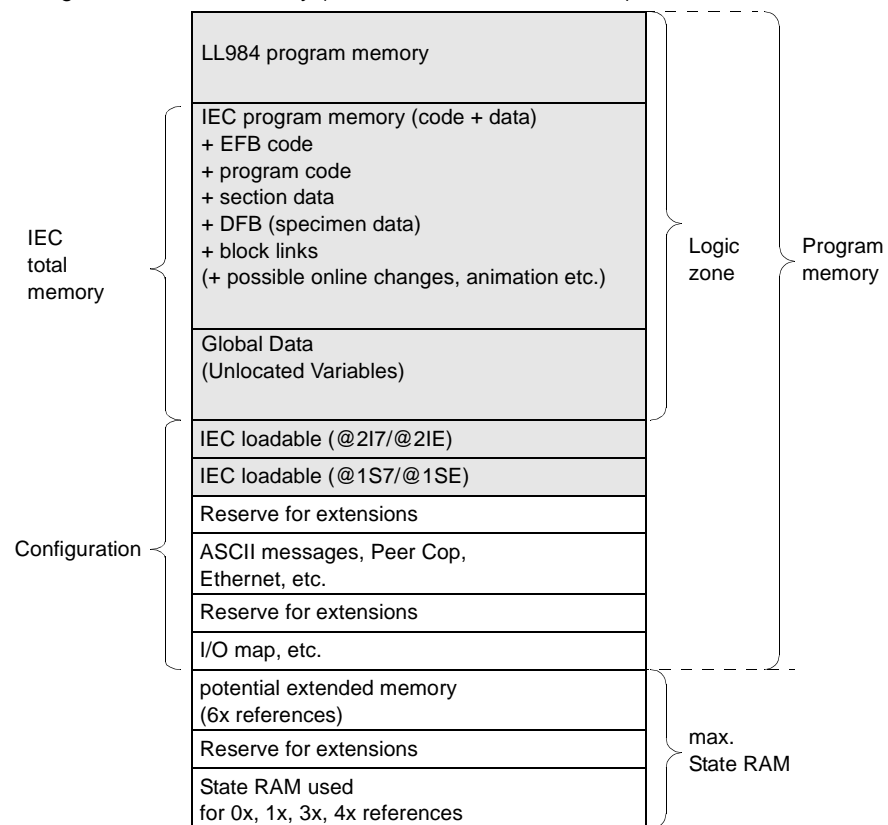
Introduction

If a CPU 213 04 or CPU 424 02 is used, you can make a zone in the state RAM available for the 6x references.

Note: 6x references are registers and can only be used with LL984 user programs.

Even if the state RAM memory zone is outside the program memory zone, the size of the state RAM influences the size of the program memory.

Using the extended memory (state RAM for 6x references):



If you do NOT use 6x

If you do not want to use any 6x references, you can, with a CPU 213 04, select whether to reserve state RAM 6x references or not.

Under **Project** → **PLC Configuration** → **PLC Selection** select from the **Memory Partition** the **48 K Logic / 32 K Memory** entry.

Note: With a CPU 424 02 there is no option for deactivating the 6x zone.

If you use 6x

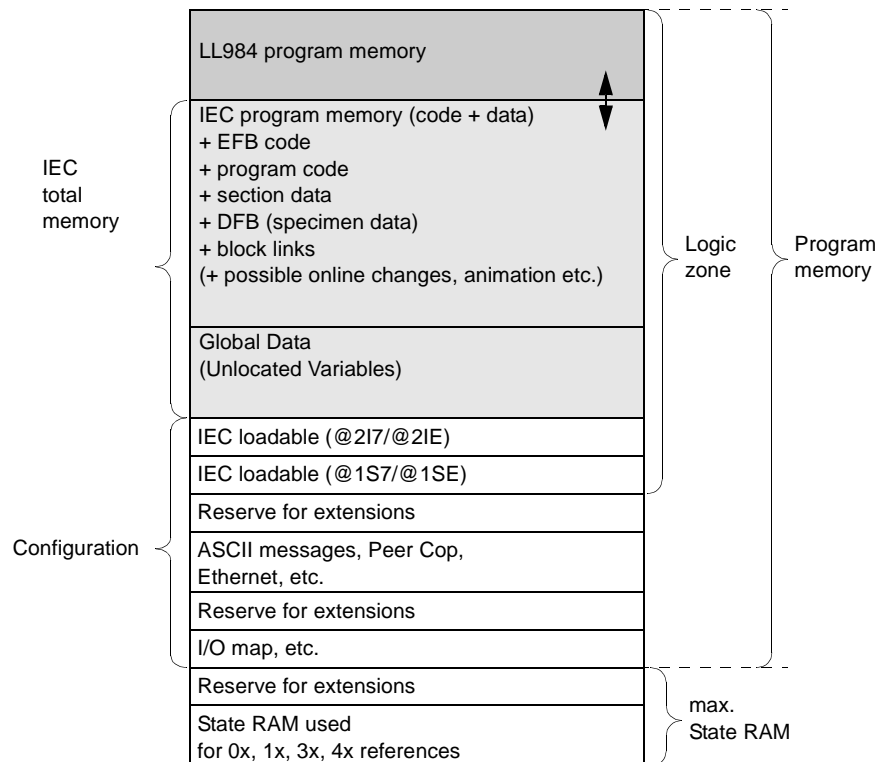
If you want to use 6x references, select under **Project** → **PLC Configuration** → **PLC selection** in the **Memory Partition** list box, the **32 K Logic / 64 K Memory** entry.

Harmonizing the IEC Zone and LL984 Zone

Introduction

With joint use of IEC and LL984 sections, the sizes of both zones should be harmonized with each other.

Harmonizing the IEC zone and LL984 zone:



Size of IEC Zone The size of the total IEC memory and also the available space for LL984 data (user program) is determined by the memory size of the loadable @2I7 or @2IE.

You can define the memory size of the loadables in **Project** → **PLC Configuration** → **Loadables** → **Install @2I7 or @2IE** → **Edit...** → **Memory Size**.

The total size is given in paragraphs. A paragraph equals 16 bytes.

For the @1S7 or @1SE loadables, no memory size is needed. Ensure that "0" is specified here.

The fixed total IEC memory size is again made up of several zones. You will find the explanation of how to harmonize these zones vertically in the chapter *Harmonizing the Zones for Global Data and IEC Program Memory*, p. 133.

Size of LL984 Zone

The size of the available memory for LL984 user programs is calculated using the following formula:

LL984 zone = available LL984 nodes – memory size of loadable @2I7/@2IE – size of loadables @2I7 or @2IE – size of loadables @1S7 or @1SE

When doing this calculation, it must be ensured that the size of the LL984 zone is node-oriented and the remaining instructions are byte-oriented.

Error Message during Download of Program

There are three possible causes for an error message, which says that the user program is too large for the PLC memory, appearing during download:

1. The memory is currently too small.
 2. The loadable memory size is too small (see current chapter).
 3. The zone for global data and the IEC program memory zone are not optimally harmonized (see chapter *Harmonizing the Zones for Global Data and IEC Program Memory*, p. 133).
-

Harmonizing the Zones for Global Data and IEC Program Memory

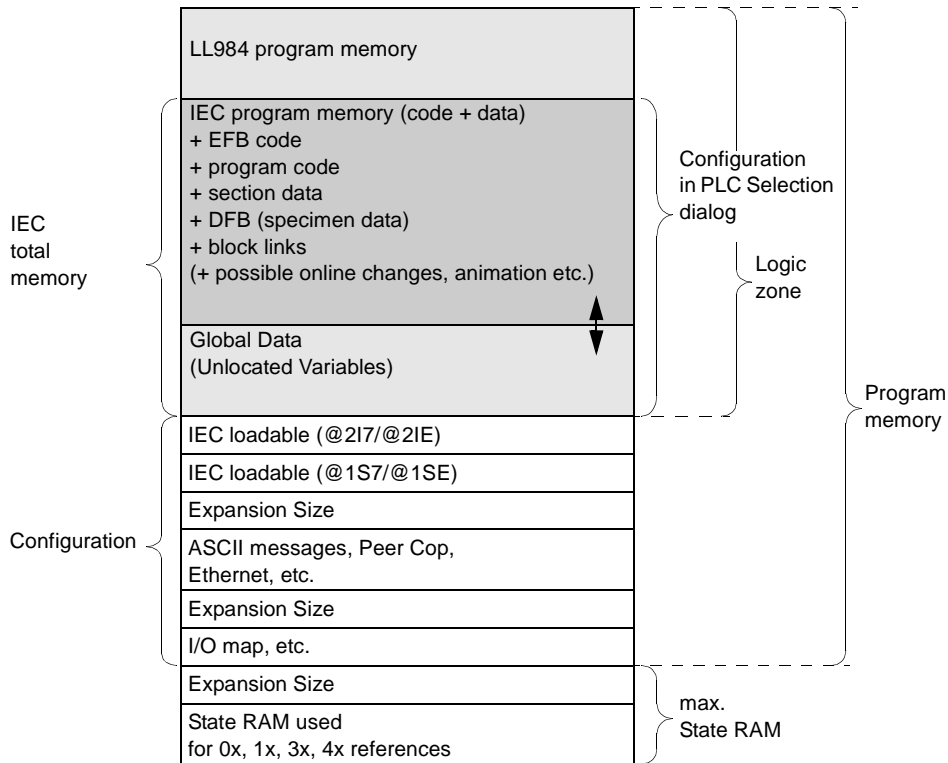
Introduction

The total IEC memory space, determined by the loadable memory size, (see Chapter *Harmonizing the IEC Zone and LL984 Zone*, p. 131) is made up of two zones:

- **IEC Program Memory**
 - comprising the EFB codes,
 - the program codes,
 - the section data,
 - the DFB specimen data,
 - the block links,
 - possibly data from online changes,
 - possibly animation data etc.
- **Global Data**
 - comprising the Unlocated Variables

The zones for global data and IEC program memory can be harmonized with one another.

Harmonizing the Zones for IEC Program Memory and Global Data:



Size of the IEC Program Memory Zone

You change the settings for the IEC program memory in **Project** → **PLC Configuration** → **PLC selection** in the **IEC** zone. Enter the size of the total IEC memory and the global data, so that the IEC program memory size will be calculated (IEC program memory size = total IEC memory - global data). This setting is only possible when the PC and PLC are offline. If you do not use any or only a few unlocated variables and have no or only a few block links, you can select the IEC program memory as very large, because hardly any memory is needed for global data.

Size of the Zone for Global Data

The zone for global data (unlocated variables) is calculated using the following formula:

Zone for global data = memory size of the loadable - IEC program memory

The current content of the individual zones (EFBs, specimen data, user program etc.) is displayed under **Online** → **Memory statistics...** → **Memory statistics**. This display is only possible when the PC and PLC are online.

Error Message during Download of Program

There are three possible reasons for an error message, which says that the user program is too large for the PLC memory, appearing while downloading the program onto the PLC:

1. The memory is currently too small.
 2. The loadable memory size is too small (see Chapter *Harmonizing the IEC Zone and LL984 Zone*, p. 131).
 3. The zone for global data and the IEC program memory zone are not optimally harmonized (see current chapter).
-

6.4 Memory Optimization for Quantum CPU 434 12(A) and 534 14(A)

Introduction

Overview This section describes the memory optimization for the Quantum CPUs 434 12(A) and 534 14(A).

What's in this Section? This section contains the following topics:

Topic	Page
General Information on Memory Optimization for Quantum CPU 434 12(A) and 534 14(A)	137
Harmonizing IEC Zone and LL984 Zone	139
Harmonizing the Zones for Global Data and IEC Program Memory (CPU 434 12(A) / 534 14 (A))	144

General Information on Memory Optimization for Quantum CPU 434 12(A) and 534 14(A)

Logic Memory

The program memory zone, in which the user program is located, is called the logic zone. This zone therefore determines the maximum size of your user program.

The current size of the logic zone is displayed under **Project** → **PLC Configurator** in the configurations overview in the **PLC** zone. The memory size is given in nodes for LL984 (1 node equals 11 bytes) and in kilobytes for IEC.

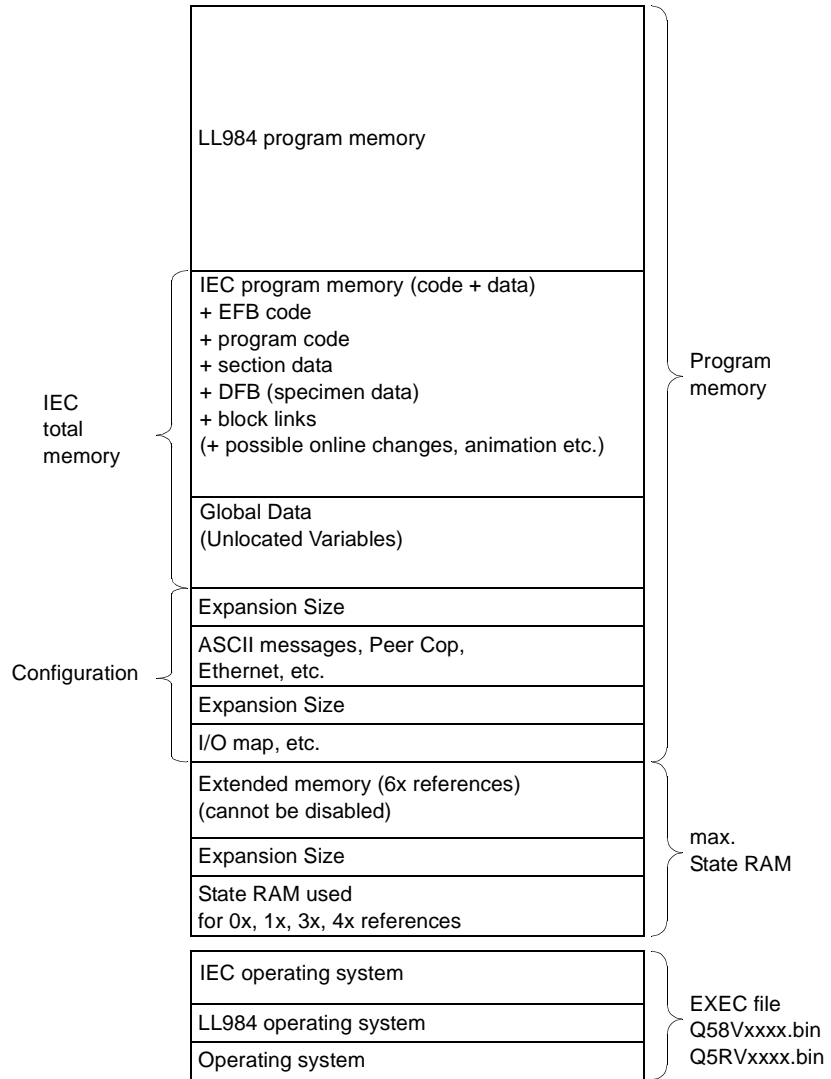
Optimizing the Logic Memory

You have various possibilities for optimising the logic memory to suit your requirements:

- *Harmonizing IEC Zone and LL984 Zone, p. 139*
- *Harmonizing the Zones for Global Data and IEC Program Memory (CPU 434 12(A) / 534 14 (A)), p. 144*

<p>Note: Also note the PLC-independent possibilities for memory optimization (See <i>General Information on Memory Optimization, p. 118</i>).</p>
--

Structure of the CPU 434 12(A) / 534 14(A) memory (simplified representation):



Harmonizing IEC Zone and LL984 Zone

Introduction

The EXEC file "Q58Vxxx.bin" is required for the CPU 434 12 and 534 14.
The EXEC file "Q5RVxxx.bin" is required for the CPU 434 12A and 534 14A (redesigned CPUs).
These EXEC files contain the runtime systems for IEC and LL984.

The sizes of the logic zones for IEC and LL984 should be harmonized with each other. The size of both zones can be defined in **Project** → **PLC Configurator** → **PLC type...** → **PLC selection**.

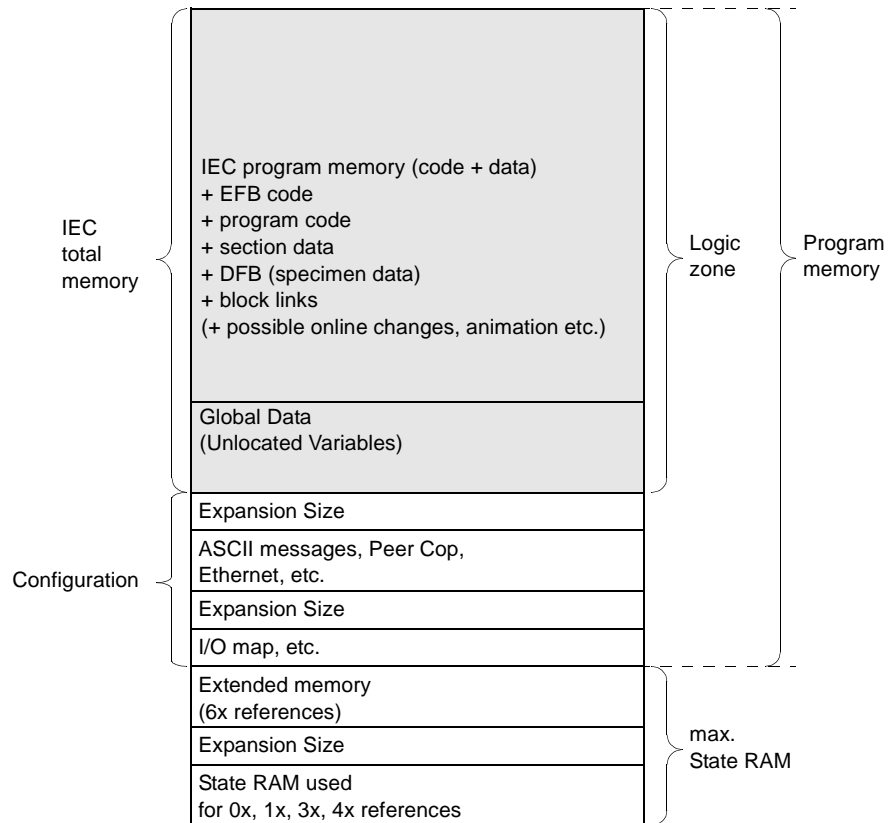
Depending on the size you select for the IEC zone, zones will be reserved in the program memory of the PLC for IEC and/or LL984 programs. Therefore, if you define a combined IEC and LL984 zone and then only use one of the two language types in the user program, the program memory will not be used optimally. Therefore, decide which languages you want to use:

- *Exclusive Use of IEC, p. 140*
 - *Exclusive Use of LL984, p. 141*
 - *Joint Use of IEC and LL984, p. 142*
-

Exclusive Use of IEC

If you require exclusive use of the IEC, select in **Project** → **PLC Configuration** → **PLC Selection** in the **IEC Operating System** list box, the entry **Enable** and drag the **total IEC memory** slider to the right hand margin (highest value). This will completely switch off the LL984 zone and the entire logic zone will be made available for the IEC user program.

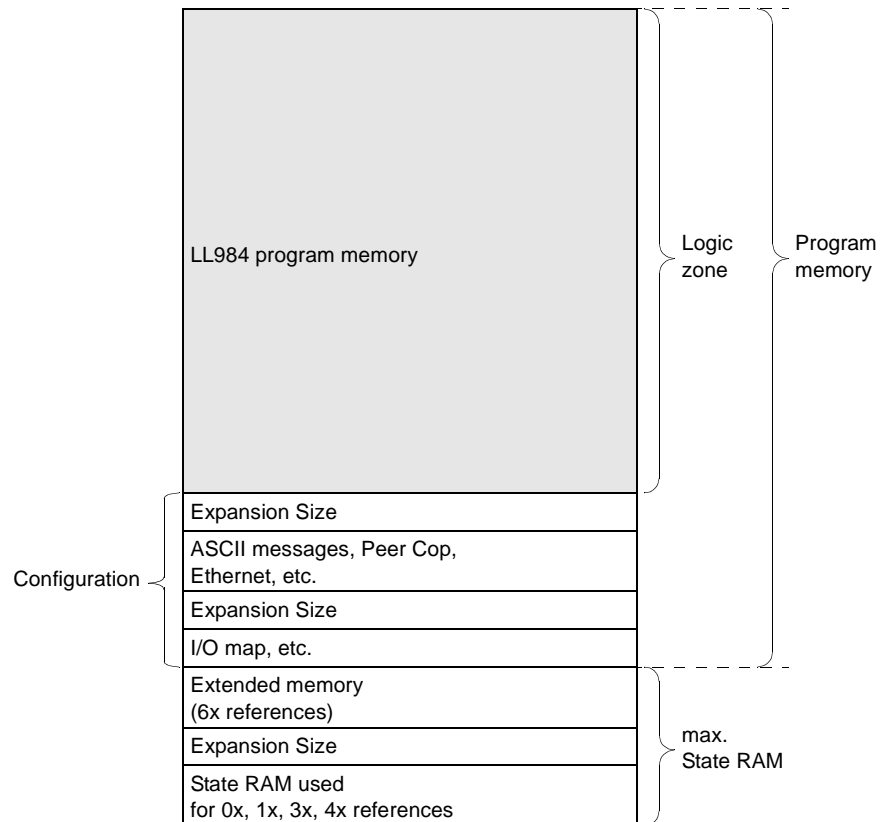
Structure of the CPU 434 12 (A)/ 534 14(A) memory with exclusive use of IEC:



Exclusive Use of LL984

If you require exclusive use of LL984, select from **Project** → **PLC Configuration** → **PLC Selection** in the **IEC Operating System** list box, the **Disable** entry. This will completely switch off the IEC zone and the entire logic zone will be made available for the LL984 user program.

Structure of the CPU 434 12(A)/ 534 14(A) memory with exclusive use of LL984:



Joint Use of IEC and LL984

When using IEC and LL984 jointly, you should harmonize the sizes of both zones with each other.

By setting the **total IEC memory size** and **Global Data** you can automatically determine the size of the IEC program memory, and also the available space for LL984-data (user program).

The size of the available memory for LL984 user programs is calculated using the following formula:

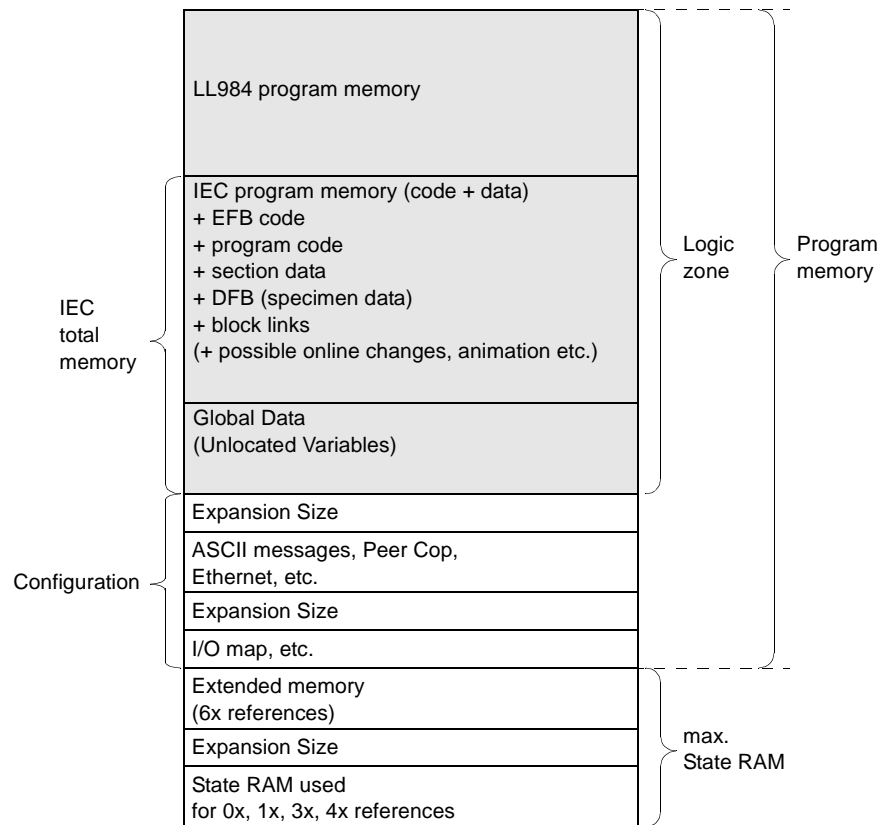
$$\text{LL984 zone} = \text{available LL984 nodes} - \text{total IEC memory}$$

When performing this calculation, it must however be ensured that the size of the LL984 zone is node-oriented and the remaining instructions are kilobyte-oriented.

To set the total IEC memory, select from **Project** → **PLC Configuration** → **PLC selection** in the **IEC Operating System** list box, the **Enable** entry. The IEC zone is now enabled and you can enter the required memory size in the **Total IEC Memory** text box. The memory size is given in kilobytes.

The fixed total IEC memory size is again made up of several zones. You will find the explanation of how to harmonize these zones vertically in the chapter *Harmonizing the Zones for Global Data and IEC Program Memory*, p. 133.

Structure of the CPU 434 12(A)/ 534 14(A) memory with exclusive use of IEC and LL984:



Error Message during Download of Program

There are three possible causes for an error message, which says that the user program is too large for the PLC memory, appearing during download:

1. The memory is currently too small.
2. The logic zone is too small (see current chapter).
3. The zone for global data and the IEC program memory zone are not optimally harmonized (see chapter *Harmonizing the Zones for Global Data and IEC Program Memory (CPU 434 12(A) / 534 14 (A))*, p. 144).

Harmonizing the Zones for Global Data and IEC Program Memory (CPU 434 12(A) / 534 14 (A))

Introduction

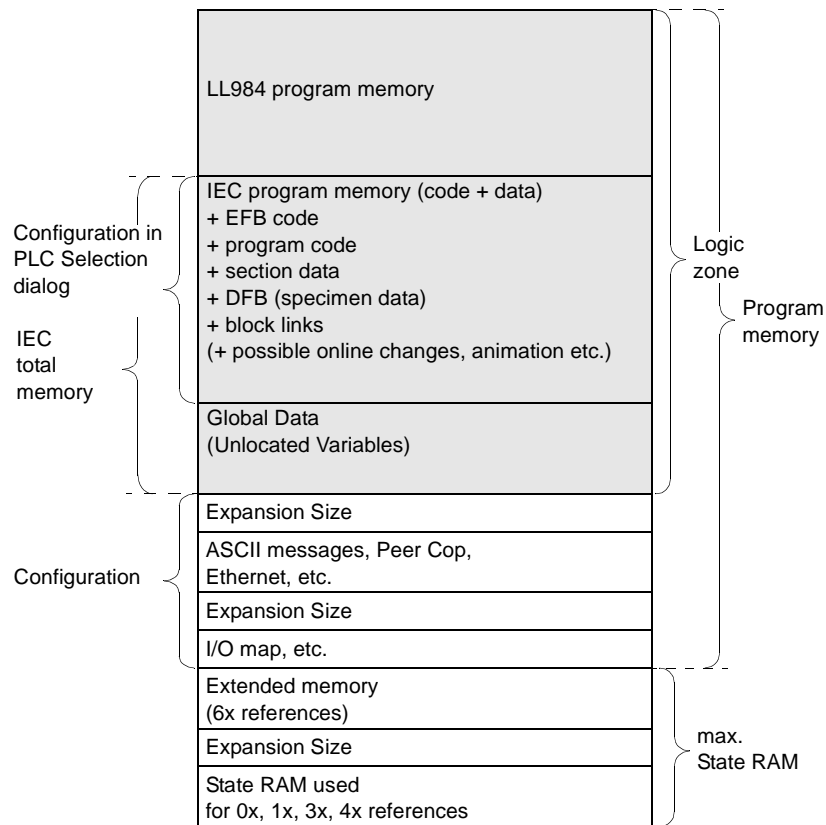
The fixed total IEC memory (see chapter *Harmonizing IEC Zone and LL984 Zone*, p. 139) is made up of two zones.

The total IEC memory space, determined by the loadable memory size, (see Chapter *Harmonizing the IEC Zone and LL984 Zone*, p. 131) is made up of two zones:

- **IEC Program Memory**
 - comprising the EFB codes,
 - the program codes,
 - the section data,
 - the DFB specimen data,
 - the block links,
 - possibly data from online changes,
 - possibly animation data etc.
- **Global Data**
 - comprising the Unlocated Variables

The zones for global data and IEC program memory can be harmonized with one another.

Harmonizing the Zones for Global Data and IEC Program Memory (CPU 434 12(A)
/ 534 14 (A))



Size of the IEC Program Memory Zone

You change the settings for the IEC program memory in **Project** → **PLC Configuration** → **PLC selection** in the **IEC** zone. Enter the size of the total IEC memory and the global data, so that the IEC program memory size will be calculated (IEC program memory size = total IEC memory - global data). This setting is only possible when the PC and PLC are offline. If you do not use any or only a few unlocated variables and have no or only a few block links, you can select the IEC program memory as very large, because hardly any memory is needed for global data.

Size of the Zone for Global Data

The zone for global data (unlocated variables) is calculated using the following formula:

Zone for global data = memory size of the loadable - IEC program memory

The current content of the individual zones (EFBs, specimen data, user program etc.) is displayed under **Online** → **Memory statistics...** → **Memory statistics**. This display is only possible when the PC and PLC are online.

Error Message during Download of Program

There are three possible reasons for an error message, which says that the user program is too large for the PLC memory, appearing while downloading the program onto the PLC:

1. The memory is currently too small.
 2. The total IEC memory size is too small (see Chapter *Harmonizing IEC Zone and LL984 Zone*, p. 139).
 3. The zone for global data and the IEC program memory zone are not optimally harmonized (see current chapter).
-

6.5 Memory optimization for Compact CPUs

At a Glance

Overview

This Section describes the memory optimization for Compact CPUs.

What's in this Section?

This section contains the following topics:

Topic	Page
General Information on Memory Optimization for Compact CPUs	148
Harmonizing IEC Zone and LL984 Zone	150
Harmonizing the Zones for Global Data and IEC Program Memory (Compact)	155

General Information on Memory Optimization for Compact CPUs

Logic Memory

The program memory zone, in which the user program is located, is called the logic zone. This zone therefore determines the maximum size of your user program.

The current size of the logic zone is displayed under **Project** → **PLC Configuration** in the configurations overview in the **PLC** zone. The entry for the memory size is given in Nodes for LL984 (1 node equals 11 bytes) and in kilobytes for IEC.

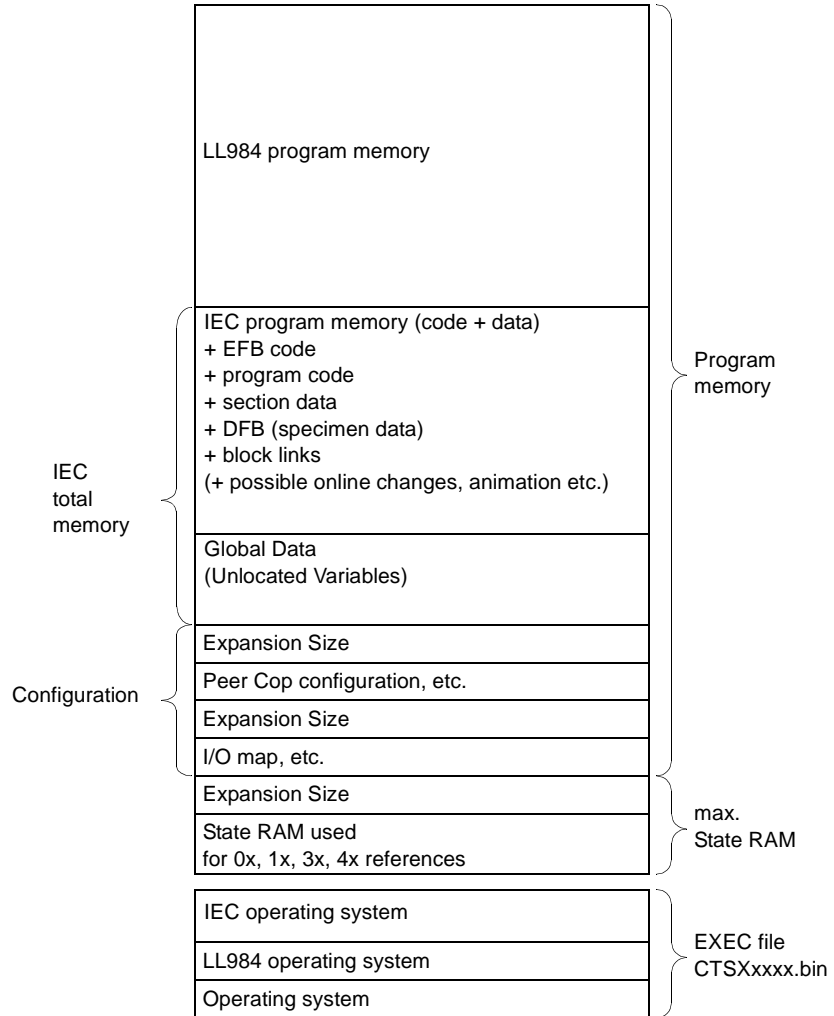
Optimizing the Logic Memory

You have various possibilities for optimising the logic memory to suit your requirements:

- *Harmonizing IEC Zone and LL984 Zone, p. 150*
- *Harmonizing the Zones for Global Data and IEC Program Memory (Compact), p. 155*

Note: Also note the PLC-independent possibilities for memory optimization (See *General Information on Memory Optimization, p. 118*).

Structure of a Compact CPU memory (simplified representation)



Harmonizing IEC Zone and LL984 Zone

Introduction

The IEC zone "CTSxxxx.bin", required for Compact CPUs, contains the runtime systems for IEC and LL984 (see also *Installation instructions*).

The sizes of the logic zones for IEC and LL984 should be harmonized with each other. You can define the size of both zones in **Project** → **PLC Configurator** → **PLC Type...** → **PLC Selection**.

Depending on the size you select for the IEC zone, zones will be reserved in the program memory of the PLC for IEC and/or LL984 programs. Therefore, if you define a combined IEC and LL984 zone and then only use one of the two language types in the user program, the program memory will not be used optimally.

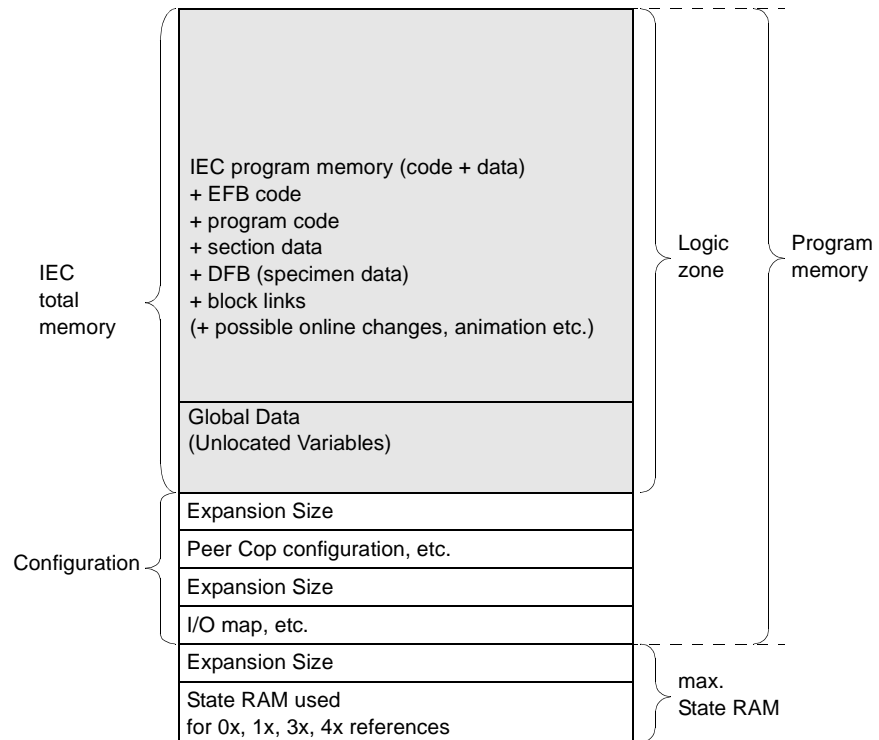
Therefore, decide which languages you want to use:

- *Exclusive Use of IEC, p. 151*
 - *Exclusive Use of LL984, p. 152*
 - *Joint Use of IEC and LL984, p. 153*
-

Exclusive Use of IEC

If you require exclusive use of the IEC, select in **Project** → **PLC Configuration** → **PLC Selection** in the **IEC Operating System** list box, the entry **Enable** and drag the **total IEC memory** slider to the right hand margin (highest value). This will completely switch off the LL984 zone and the entire logic zone will be made available for the IEC user program.

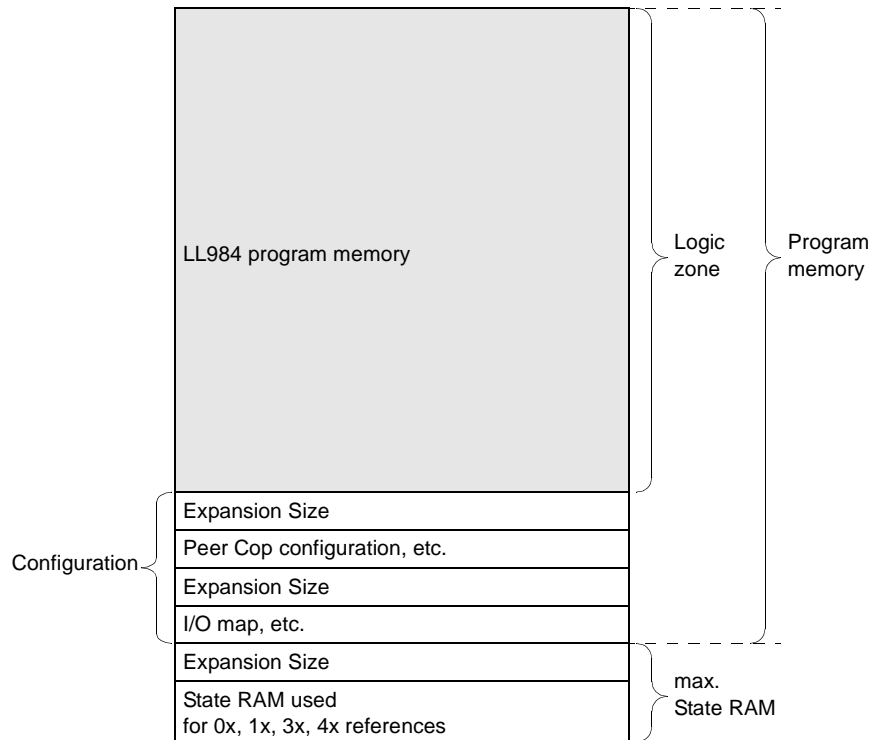
Structure of the Compact CPU memory with exclusive use of IEC



Exclusive Use of LL984

If you require exclusive use of LL984, select from **Project** → **PLC Configuration** → **PLC Selection** in the **IEC Operating System** list box, the **Disable** entry. This will completely switch off the IEC zone and the entire logic zone will be made available for the LL984 user program.

Structure of the Compact CPU memory with exclusive use of LL984



Joint Use of IEC and LL984

When using IEC and LL984 jointly, you should harmonize the sizes of both zones with each other.

By setting the **total IEC memory size** and **Global Data** you can automatically determine the size of the IEC program memory, and also the available space for LL984-data (user program).

The size of the available memory for LL984 user programs is calculated using the following formula:

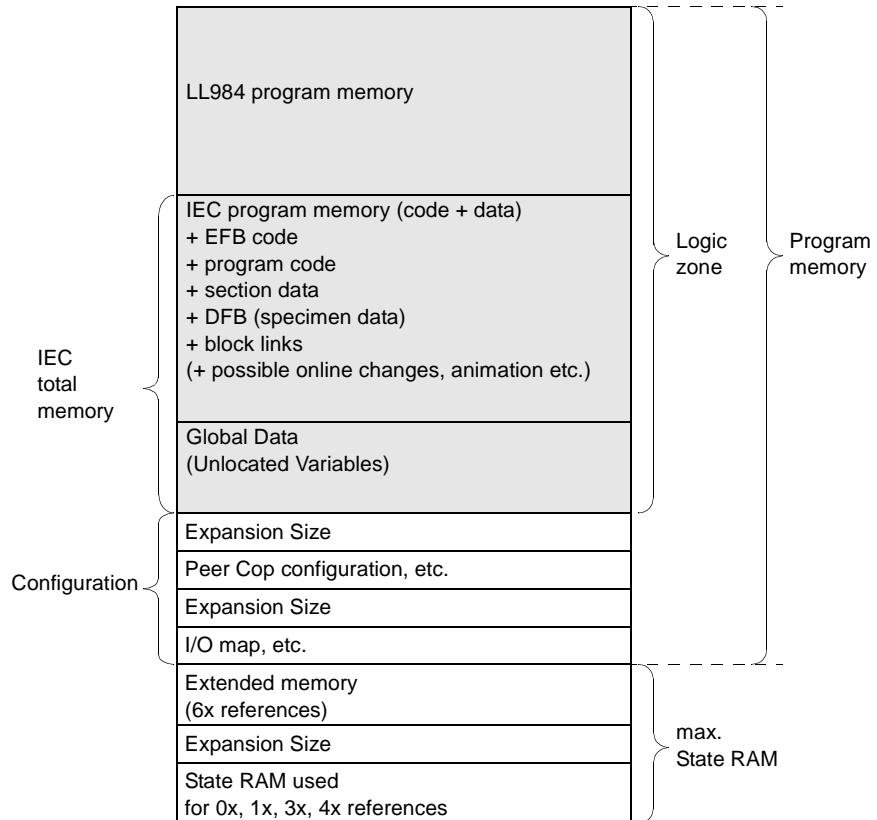
LL984 zone = available LL984 nodes - total IEC memory

When performing this calculation, it must however be ensured that the size of the LL984 zone is node-oriented and the remaining instructions are kilobyte-oriented.

To set the total IEC memory, select from **Project** → **PLC Configuration** → **PLC selection** in the **IEC Operating System** list box, the **Enable** entry. The IEC zone is now enabled and you can enter the required memory size in the **Total IEC Memory** text box. The memory size is given in kilobytes.

The fixed total IEC memory size is again made up of several zones. You will find the explanation of how to harmonize these zones vertically in the chapter *Harmonizing the Zones for Global Data and IEC Program Memory (Compact)*, p. 155.

Structure of the Compact Memory with joint use of IEC and LL984:



Error Message during Download of Program

There are three possible causes for an error message, which says that the user program is too large for the PLC memory, appearing during download:

1. The memory is currently too small.
2. The logic zone is too small (see current chapter).
3. The zone for global data and the IEC program memory zone are not optimally harmonized (see chapter *Harmonizing the Zones for Global Data and IEC Program Memory (Compact)*, p. 155).

Harmonizing the Zones for Global Data and IEC Program Memory (Compact)

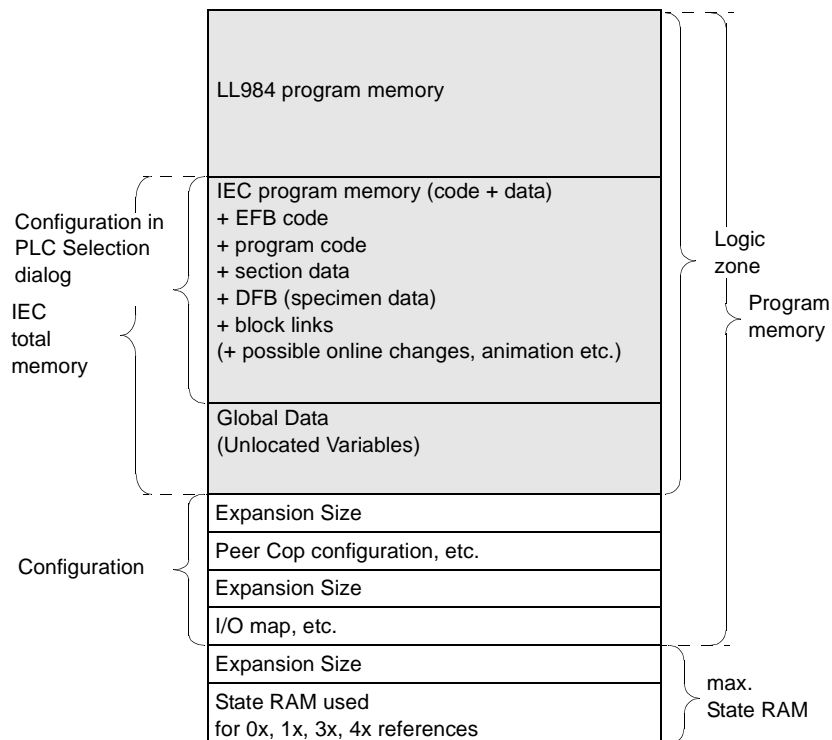
Introduction

The fixed total IEC memory (see chapter *Harmonizing IEC Zone and LL984 Zone*, p. 150) is made up of two zones.

- **IEC Program Memory**
 - comprising the EFB codes,
 - the program codes,
 - the section data,
 - the DFB specimen data,
 - the block links,
 - possibly data from online changes,
 - possibly animation data etc.
- **Global Data**
 - comprising the Unlocated Variables

The zones for global data and IEC program memory can be harmonized with one another.

Harmonizing the Zones for Global Data and IEC Program Memory (Compact):



Size of the IEC Program Memory Zone	You change the settings for the IEC program memory in Project → PLC Configuration → PLC selection in the IEC zone. Enter the size of the total IEC memory and the global data, so that the IEC program memory size will be calculated (IEC program memory size = total IEC memory - global data). This setting is only possible when the PC and PLC are offline. If you do not use any or only a few unlocated variables and have no or only a few block links, you can select the IEC program memory as very large, because hardly any memory is needed for global data.
Size of the Zone for Global Data	The zone for global data (unlocated variables) is calculated using the following formula: Zone for global data = memory size of the loadable - IEC program memory The current content of the individual zones (EFBs, specimen data, user program etc.) is displayed under Online → Memory statistics... → Memory statistics . This display is only possible when the PC and PLC are online.
Error Message during Download of Program	There are three possible reasons for an error message, which says that the user program is too large for the PLC memory, appearing while downloading the program onto the PLC: <ol style="list-style-type: none">1. The memory is currently too small.2. The total IEC memory size is too small (see Chapter <i>Harmonizing IEC Zone and LL984 Zone</i>, p. 150).3. The zone for global data and the IEC program memory zone are not optimally harmonized (see current chapter).

6.6 Memory optimization for Momentum CPUs

Introduction

Overview

This Section describes the memory optimization for Momentum CPUs.

What's in this Section?

This section contains the following topics:

Topic	Page
General Information on Memory Optimization for Momentum CPUs	158
Selecting Optimal IEC Zone	160
Harmonizing the Zones for Global Data and IEC Program Memory (Momentum)	161

General Information on Memory Optimization for Momentum CPUs

Logic Memory

The program memory zone, in which the user program is located, is called the logic zone. This zone therefore determines the maximum size of your user program.

The current size of the logic zone is displayed under **Project** → **PLC Configuration** in the configurations overview in the **PLC** zone. The entry for the memory size is given in Nodes for LL984 (1 node equals 11 bytes) and in kilobytes for IEC.

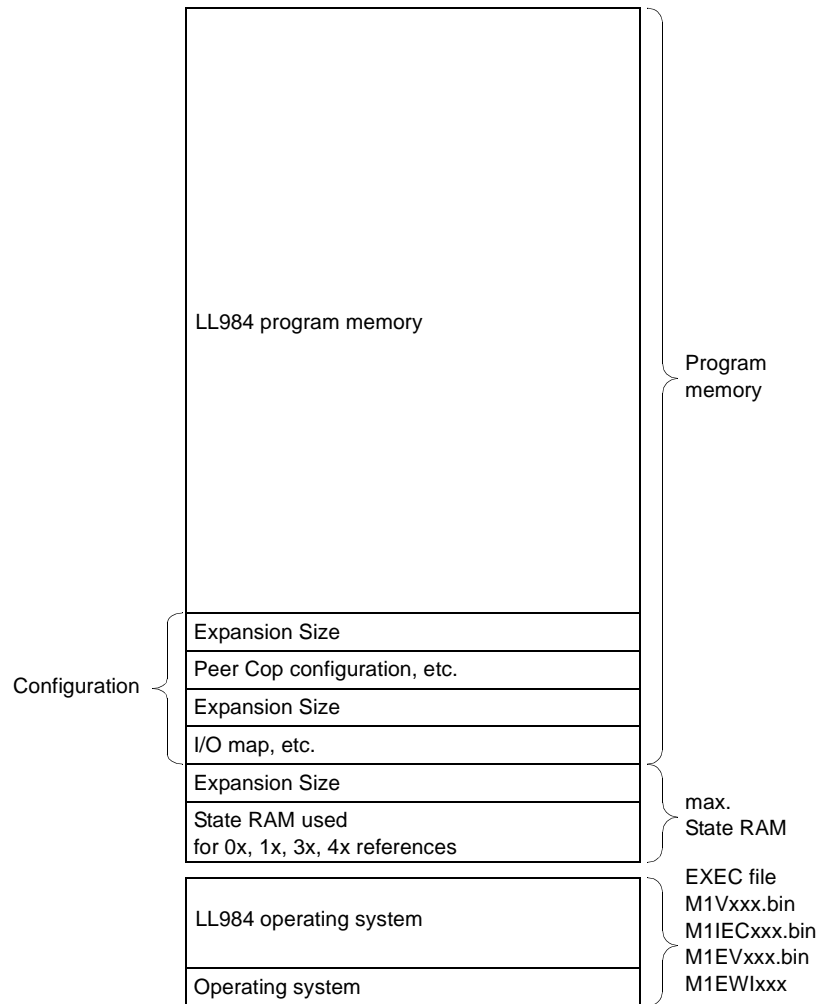
Optimizing the Logic Memory

You have various possibilities for optimising the logic memory to suit your requirements:

- *Selecting Optimal IEC Zone, p. 160*
- *Harmonizing the Zones for Global Data and IEC Program Memory (Momentum), p. 161*

Note: Also note the PLC-independent possibilities for memory optimization (See *General Information on Memory Optimization, p. 118*).

Structure of a Momentum CPU memory (simplified representation):



Selecting Optimal IEC Zone

Introduction It is not possible to use IEC and LL984 jointly in Momentum.

Using IEC EXEC file assignment during IEC use:

171 CCS	M1IECxxx	M1EWIxxx
760 00	x	-
760 10	x	-
780 10	x	-
960 30	-	x
980 30	-	x

Using LL984 EXEC file assignment during LL984 use:

171 CCS	M1Vxxx	M1EVxxx
700 10	x	-
700/780 00	x	-
760 00	x	-
760 10	x	-
780 10	x	-
960 20	-	x
960 30	-	x
980 20	-	x
980 30	-	x

Harmonizing the Zones for Global Data and IEC Program Memory (Momentum)

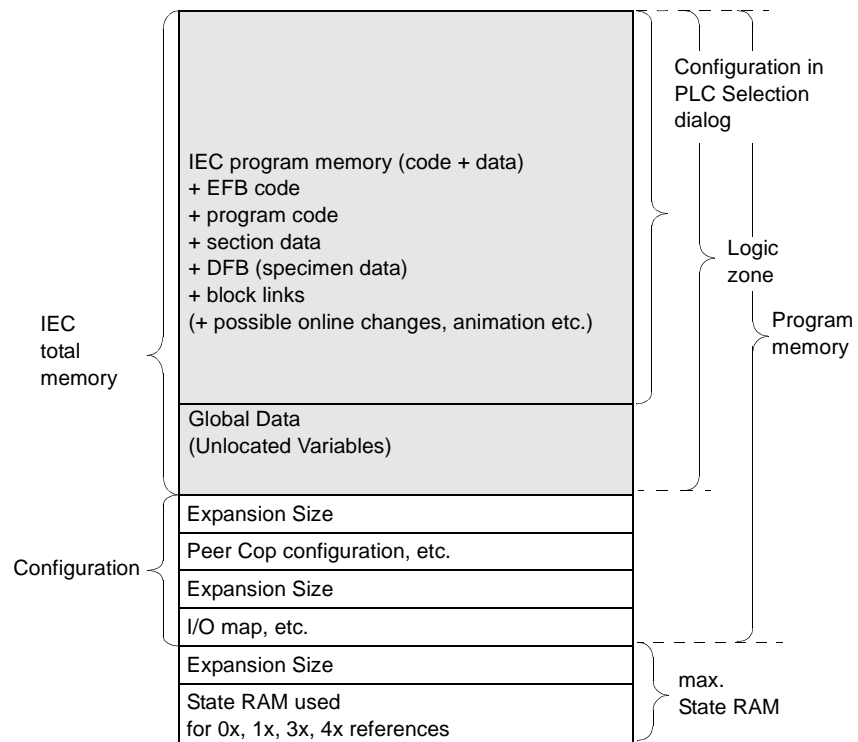
Introduction

The logic zone for the total IEC memory is made up of two zones.

- **IEC Program Memory**
 - comprising the EFB codes,
 - the program codes,
 - the section data,
 - the DFB specimen data,
 - the block links,
 - possibly data from online changes,
 - possibly animation data etc.
- **Global Data**
 - comprising the Unlocated Variables

The zones for global data and IEC program memory can be harmonized with one another.

Harmonizing the Zones for Global data and IEC Program Memory (Momentum 171 CCS 760 00-IEC):



Size of the IEC Program Memory Zone

The settings for the IEC user program zone are available in **Online** → **Memory statistics...** → **Memory statistics** in the **Configured** text box. This setting is only possible when the PC and PLC are offline. If you do not use any or only a few unlocated variables and have no or only a few block links, you can select the IEC program memory as very large, because hardly any memory is needed for global data.

Size of the Zone for Global Data

The zone for global data (unlocated variables and block links) is calculated using the following formula:

Zone for global data = memory size of the loadable - IEC program memory

The current content of the individual zones (EFBs, specimen data, user program etc.) is displayed under **Online** → **Memory statistics...** → **Memory statistics**. This display is only possible when the PC and PLC are online.

Error Message during Download of Program

There are two possible reasons for an error message, saying that the user program is too large for the PLC memory, appearing while downloading the program onto the PLC:

1. The memory is currently too small.
 2. The zone for global data and the IEC program memory zone are not optimally harmonized (see current chapter).
-

6.7 Memory optimization for Atrium CPUs

At a Glance

Overview

This Section describes the memory optimization for Atrium CPUs.

What's in this Section?

This section contains the following topics:

Topic	Page
General Information on Memory Optimization for Atrium CPUs	164
Harmonizing IEC Zone and LL984 Zone	166
Harmonizing the Zones for Global Data and IEC Program Memory (Atrium)	171

General Information on Memory Optimization for Atrium CPUs

Logic Memory

The program memory zone, in which the user program is located, is called the logic zone. This zone therefore determines the maximum size of your user program.

The current size of the logic zone is displayed under **Project** → **PLC Configurator** in the configurations overview in the **PLC** zone. The memory size is given in nodes for LL984 (1 node equals 11 bytes) and in kilobytes for IEC.

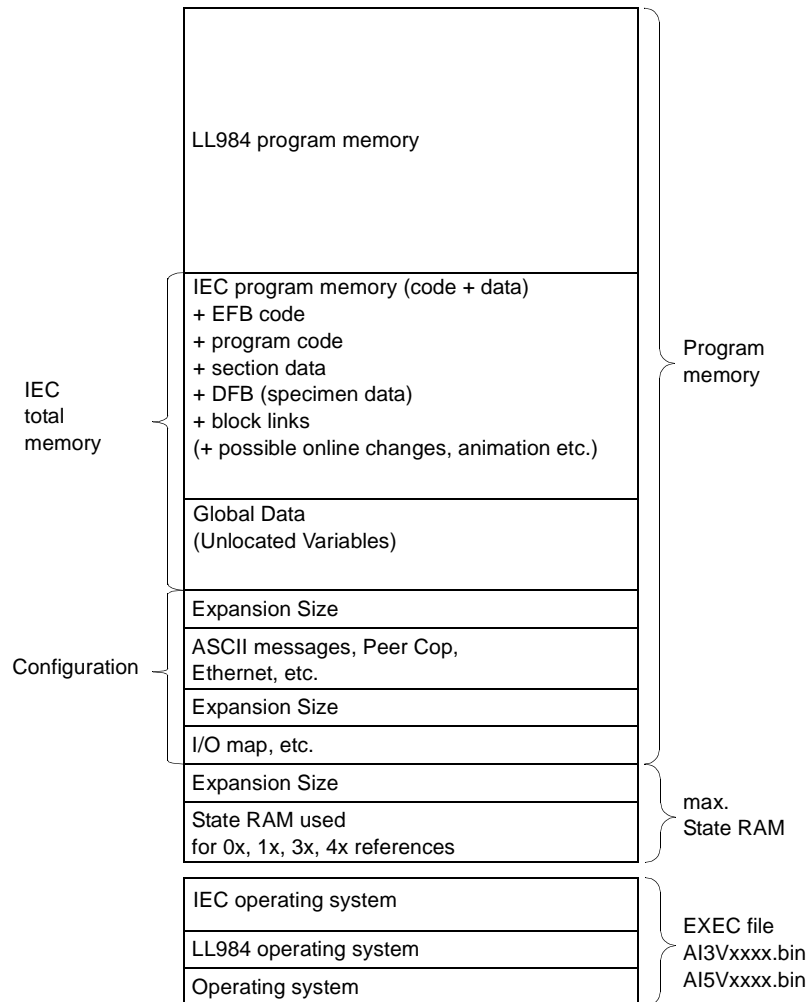
Optimizing the Logic Memory

You have various possibilities for optimising the logic memory to suit your requirements:

- *Harmonizing IEC Zone and LL984 Zone, p. 166*
- *Harmonizing the Zones for Global Data and IEC Program Memory (Atrium), p. 171*

Note: Also note the PLC-independent possibilities for memory optimization (See *General Information on Memory Optimization, p. 118*).

Structure of the Atrium CPU Memory (simplified representation):



Harmonizing IEC Zone and LL984 Zone

Introduction

The EXEC files required for the CPUs of the Atrium family contain the operating systems for IEC and LL984 (see also *Installation Instructions*).

When using the Atrium 180 CCO 121 01, load the EXEC file "AI3Vxxxx.bin".

When using the Atrium 180 CCO 241 01 and 180 CCO 241 11 load the EXEC file "AI5Vxxxx.bin".

The sizes of the logic zones for IEC and LL984 should be harmonized with each other. You can define the size of both zones in **Project** → **PLC Configuration** → **PLC Selection**.

Depending on the size you select for the IEC zone, zones will be reserved in the program memory of the PLC for IEC and/or LL984 programs. Therefore, if you define a combined IEC and LL984 zone and then only use one of the two language types in the user program, the program memory will not be used optimally.

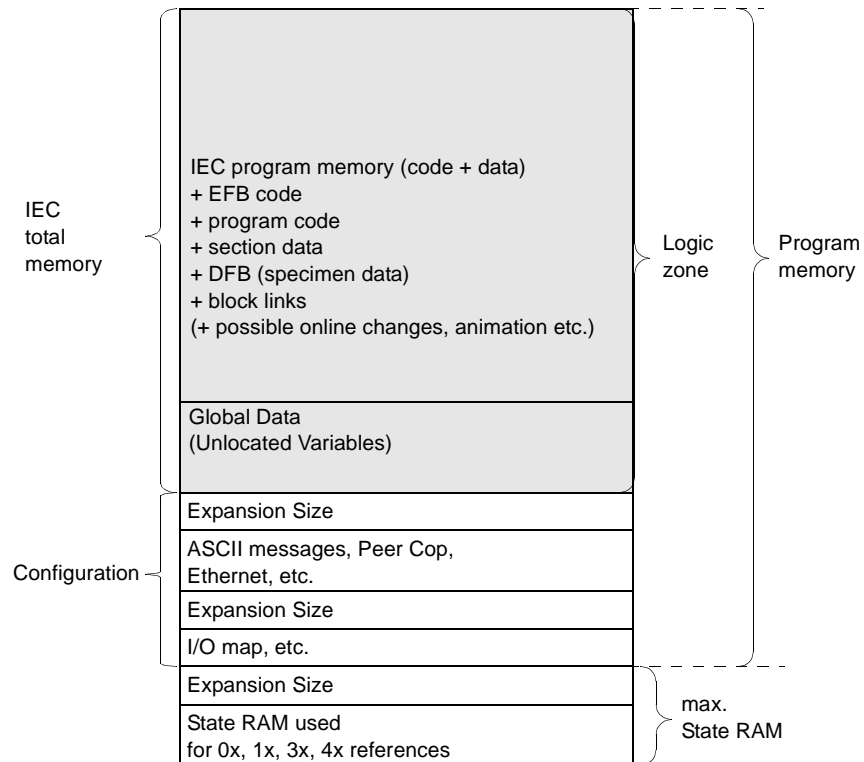
Therefore, decide which languages you want to use:

- *Exclusive Use of IEC, p. 167*
 - *Exclusive Use of LL984, p. 168*
 - *Joint Use of IEC and LL984, p. 169*
-

Exclusive Use of IEC

If you require exclusive use of the IEC, select in **Project** → **PLC Configuration** → **PLC Selection** in the **IEC Operating System** list box, the entry **Enable** and drag the **total IEC memory** slider to the right hand margin (highest value). This will completely switch off the LL984 zone and the entire logic zone will be made available for the IEC user program.

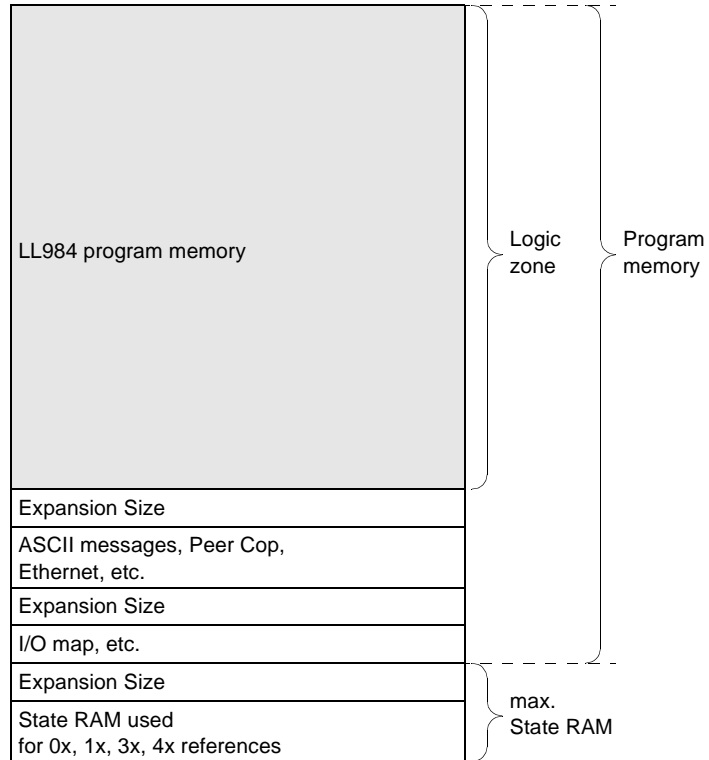
Structure of the Atrium CPU memory with exclusive use of IEC:



Exclusive Use of LL984

If you require exclusive use of LL984, select from **Project** → **PLC Configuration** → **PLC Selection** in the **IEC Operating System** list box, the **Disable** entry. This will completely switch off the IEC zone and the entire logic zone will be made available for the LL984 user program.

Structure of the Atrium CPU memory with exclusive use of LL984:



Joint Use of IEC and LL984

When using IEC and LL984 jointly, you should harmonize the sizes of both zones with each other.

By setting the **total IEC memory size** and **Global Data** you can automatically determine the size of the total IEC memory, and also the available space for LL984 data (user program).

The size of the available memory for LL984 user programs is calculated using the following formula:

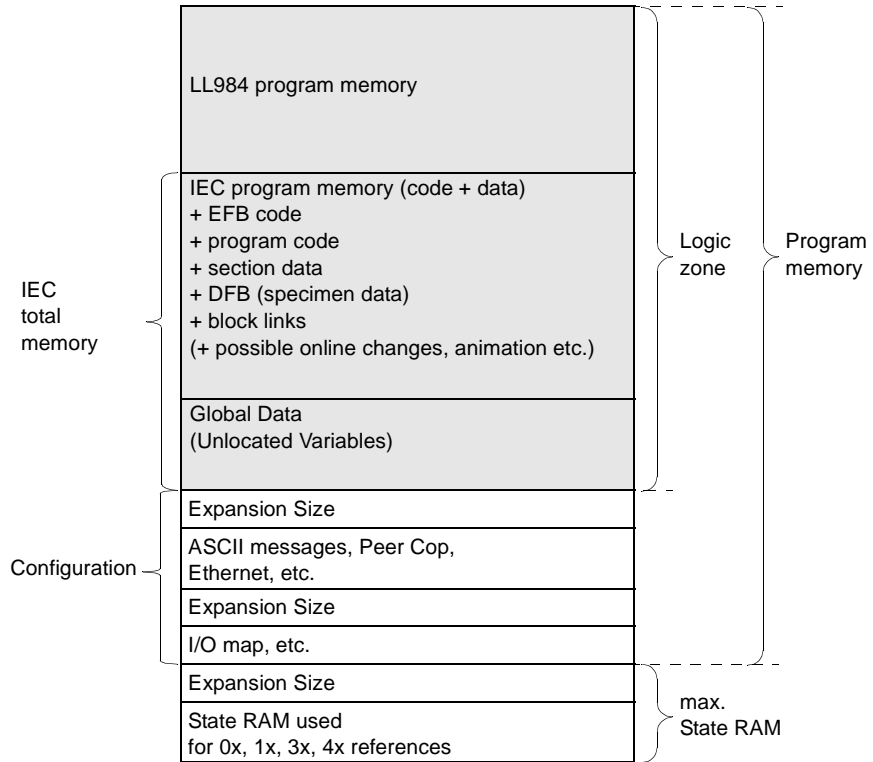
LL984 zone = available LL984 nodes - total IEC memory

When performing this calculation, it must however be ensured that the size of the LL984 zone is node-oriented and the remaining instructions are kilobyte-oriented.

To set the total IEC memory, select from **Project** → **PLC Configuration** → **PLC selection** in the **IEC Operating System** list box, the **Enable** entry. The IEC zone is now enabled and you can enter the required memory size in the **Total IEC Memory** text box. The memory size is given in kilobytes.

The fixed total IEC memory size is again made up of several zones. You will find the explanation of how to harmonize these zones vertically in the chapter *Harmonizing the Zones for Global Data and IEC Program Memory (Atrium)*, p. 171.

Structure of the Atrium CPU Memory with joint use of IEC and LL984:



Error Message during Download of Program

There are three possible causes for an error message, which says that the user program is too large for the PLC memory, appearing during download:

1. The memory is currently too small.
2. The logic zone is too small (see current chapter).
3. The zone for global data and the IEC program memory zone are not optimally harmonized (see chapter *Harmonizing the Zones for Global Data and IEC Program Memory (Atrium)*, p. 171).

Harmonizing the Zones for Global Data and IEC Program Memory (Atrium)

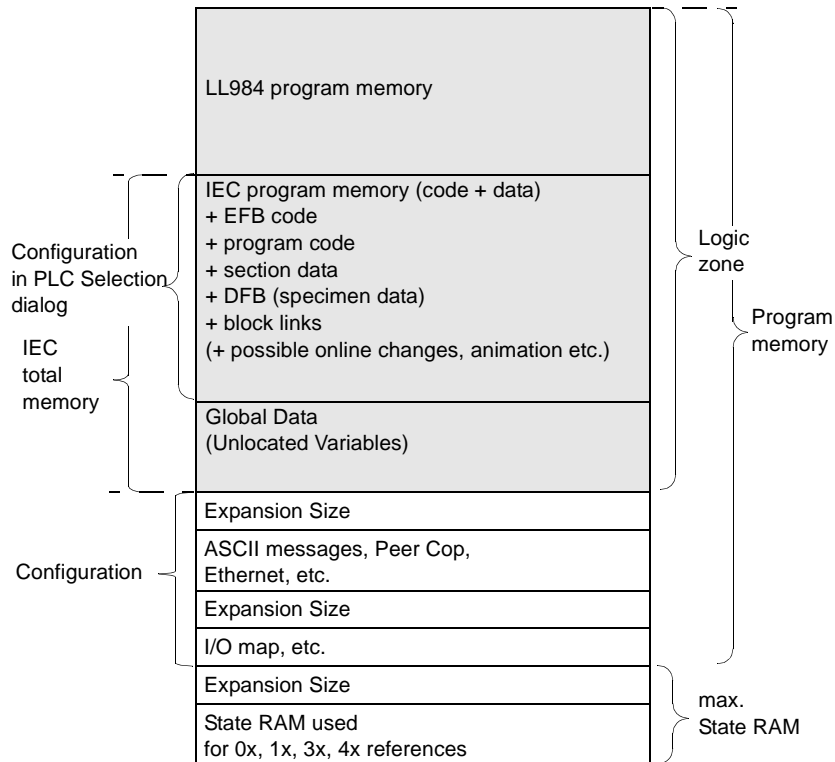
Introduction

The fixed total IEC memory (see chapter *Harmonizing IEC Zone and LL984 Zone*, p. 166) is made up of two zones.

- **IEC Program Memory**
 - comprising the EFB codes,
 - the program codes,
 - the section data,
 - the DFB specimen data,
 - the block links,
 - possibly data from online changes,
 - possibly animation data etc.
- **Global Data**
 - comprising the Unlocated Variables

The zones for global data and IEC program memory can be harmonized with one another.

Harmonizing the Zones for Global Data and IEC Program Memory (Atrium):



Size of the IEC Program Memory Zone	<p>You change the settings for the IEC program memory in Project → PLC Configuration → PLC selection in the IEC zone. Enter the size of the total IEC memory and the global data, so that the IEC program memory size will be calculated (IEC program memory size = total IEC memory - global data). This setting is only possible when the PC and PLC are offline. If you do not use any or only a few unlocated variables and have no or only a few block links, you can select the IEC program memory as very large, since hardly any memory is needed for global data.</p>
Size of the Zone for Global Data	<p>The zone for global data (unlocated variables) is calculated using the following formula:</p> $\text{Zone for global data} = \text{memory size of the loadable} - \text{IEC program memory}$ <p>The current content of the individual zones (EFBs, specimen data, user program etc.) is displayed under Online → Memory statistics... → Memory statistics. This display is only possible when the PC and PLC are online.</p>
Error Message during Download of Program	<p>There are three possible reasons for an error message, which says that the user program is too large for the PLC memory, appearing while downloading the program onto the PLC:</p> <ol style="list-style-type: none">1. The memory is currently too small.2. The total IEC memory size is too small (see Chapter <i>Harmonizing IEC Zone and LL984 Zone</i>, p. 166).3. The zone for global data and the IEC program memory zone are not optimally harmonized (see current chapter).

Function Block language FBD

7

At a Glance

Overview

This Chapter describes the Function Block language FBD which conforms to IEC 1131.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
7.1	General information about FBD Function Block	175
7.2	FBD Function Block objects	177
7.3	Working with the FBD Function Block language	185
7.4	Code generation with the FBD Function Block language	190
7.5	Online functions of the FBD Function Block language	192
7.6	Creating a program with the FBD Function Block language	195

7.1 General information about FBD Function Block

General information on Function Block language FBD

At a Glance

The objects of the programming language FBD (Function Block Diagram) help to divide a section into a number of:

- EFBs (Elementary Functions and Elementary Function Blocks) (See *EFB*, p. 178),
 - DFBs (Derived Function Blocks) (See *DFB*, p. 180) and
 - UDEFBs (User-defined Functions and Function Blocks) (See *UDEFB*, p. 181).
- These objects, combined under the name FFBs, can be linked with each other by:
- Links (See *Link*, p. 182) or
 - Current parameters (See *Actual parameters*, p. 182).

Expansive logic can also be placed in the FBD section in the form of macros (see also *Macros*, p. 455).

Theoretically, each section can contain as many FFBs and also as many inputs and outputs as required. However, it is advisable to subdivide a whole program in logic units, that is to say in different sections.

Comments can be provided for the logic of the section with text objects (see *Text Object*, p. 184).

Processing sequence

The processing sequence of the individual FFBs in an FBD section is determined by the data flow within the section (see also *FFB Execution Order*, p. 187).

Editing with the keyboard

Normally editing in Concept is performed with the mouse, however it is also possible with the keyboard (see also *Short Cut Keys in the FBD and SFC Editor*, p. 768)

IEC conformity

For a description of the IEC conformity of the FBD programming language see *IEC conformity*, p. 779.

7.2 FBD Function Block objects

At a Glance

Overview

This section describes the FBD Function Block objects.

What's in this Section?

This section contains the following topics:

Topic	Page
Functions and Function Blocks (FFBs)	178
Link	182
Actual parameters	182
Text Object	184

Functions and Function Blocks (FFBs)

Introduction

FFB is the generic term for:

- EFB (Elementary Function and Elementary Function Block) (See *EFB*, p. 178)
 - DFB (Derived Function Block) (See *DFB*, p. 180)
 - UDEFB (Derived Elementary Function and Derived Elementary Function Block) (See *UDEFB*, p. 181)
-

EFB

EFB is the generic term for:

- Elementary Function (See *Elementary Function*, p. 178)
- Elementary Function Block (See *Elementary Function Block*, p. 179)

EFBs are functions and function blocks that are available in Concept in the form of libraries. The logic of EFBs is built in C programming language and cannot be changed in the FBD editor.

Elementary Function

Functions have no internal conditions. If the input values are the same, the value at the output is the same for all executions of the function. E.g. the addition of two values gives the same result at every execution.

An Elementary Function is represented graphically as a frame with inputs and outputs. The inputs are always represented on the left and the outputs always on the right of the frame. The name of the function, that is the function type, is displayed in the center of the frame. The function counter is displayed above the frame.

The function counter cannot be changed and always has an .n.m. structure.

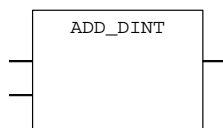
.n = current section number

.m = current function number

Functions are only executed in FBD if the input EN=1 or if the input EN is grayed out (see also *EN and ENO*, p. 181).

Elementary Function

. 6 . 5



Elementary Function Block

Function blocks have internal conditions. If the inputs have the same values, the value at the output at every execution is another value. E.g. with a counter, the value on the output is incremented.

A function block is represented graphically as a frame with inputs and outputs. The inputs are always represented on the left and the outputs always on the right of the frame. The name of the function block, that is the function block type, is displayed in the center of the frame. The instance name is displayed above the frame. The instance name serves as a unique identification for the function block in a project.

The instance name is produced automatically with the following structure: FBI_n_m

FBI = Function Block Instance

n = Section number (current number)

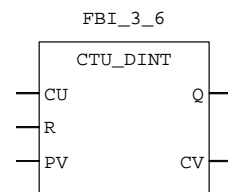
m = Number of the FFB object in the section (current number)

The instance name can be edited in the **Object** → **Properties** dialog box of the function block. The instance name must be unique throughout the whole project and is not case sensitive. If the name entered already exists, you will be warned and you will have to choose another name. The instance name must correspond to the IEC name conventions, otherwise an error message occurs.

Note: In compliance with IEC1131-3 only letters are permitted as the first character of instance names. Should numbers be required as the first character however, the menu command **Options** → **Preferences** → **IEC Extensions...** → **Allow leading digits in identifiers** will enable this.

Function blocks are only executed in FBD if the input EN=1 or if the input EN is grayed out (related topics *EN and ENO*, p. 181).

Elementary Function Block



DFB

Derived Function Blocks (DFBs) are function blocks that have been defined in Concept DFB.

With DFBs, there is no distinction between functions and function blocks. They are always treated as function blocks regardless of their internal structure.

A DFB is represented graphically as a frame with double vertical lines and with inputs and outputs. The inputs are always represented on the left and the outputs always on the right of the frame. The DFB name is displayed centrally within the frame. The instance name is displayed above the frame. The instance name serves as a unique identification for the function block in a project.

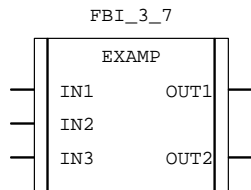
The instance name is produced automatically with the following structure: FBI_n_m
 FBI = Function Block Instance
 n = Section number (current number)
 m = Number of the FFB object in the section (current number)

The instance name can be edited in the **Object** → **Properties** dialog box of the DFB. The instance name must be unique throughout the whole project and is not case sensitive. If the name entered already exists, you will be warned and you will have to choose another name. The instance name must correspond to the IEC name conventions, otherwise an error message occurs.

Note: In compliance with IEC1131-3 only letters are permitted as the first character of instance names. Should numbers be required as the first character however, the menu command **Options** → **Preferences** → **IEC Extensions...** → **Allow leading digits in identifiers** will enable this.

Derived function blocks are only executed in FBD if the input EN=1 or if the input EN is grayed out (related topics *EN and ENO*, p. 181).

Derived Function Block



UDEFB

UDEFB is the generic term for:

- User-defined Elementary Function
- User-defined Elementary Function Block

UDEFBs are functions and function blocks that have been programmed with Concept EFB in C++ programming language and are available in Concept in the form of libraries.

In Concept, there is no functional difference between UDEFBs and EFBs.

EN and ENO

With all FFBs, an EN input and an ENO output can be configured.

The configuration of EN and ENO is switched on or off in the **FFB Properties** dialog box. The dialog box can be called up with the **Objects** → **Properties...** menu command or by double-clicking on the FFB.

If the value of EN is equal to "0" when the FFB is invoked, the algorithms that are defined by the FFB will not be executed and all outputs keep their previous values. The value of ENO is automatically set to "0" in this case.

If the value of EN is equal to "1", when the FFB is called up, the algorithms which are defined by the FFB will be executed. After successful execution of these algorithms, the value of ENO is automatically set to "1". If an error occurs during execution of these algorithms, ENO will be set to "0".

The output behavior of the FFBs in FBD does not depend on whether the FFBs are called up without EN/ENO or with EN=1.

Link

Description	Links are connections between FFBs. Several links can be connected with one FFB output. The link points are identified by a filled-in circle.
Data Types	The data types of the inputs/outputs to be linked must be the same.
Creating Links	Links can be created using Objects → Link .
Editing Links	Links can be edited in select mode. An overlap with other objects is permitted.
Configuring Loops	No loop can be configured with links because in this case, the execution order in the section cannot be determined uniquely. Loops must be resolved with actual parameters (see <i>Configuring Loops</i> , p. 189).

Actual parameters

At a Glance	In the program runtime, the values from the process or from other actual parameters are transferred to the FFB over the actual parameters and then re-emitted after processing. These actual parameters can be: <ul style="list-style-type: none">• direct addresses (See <i>Direct addresses</i>, p. 39)• Located variables (See <i>Variables</i>, p. 36)• Unlocated variable (See <i>Variables</i>, p. 36)• Constants (See <i>Constant variables</i>, p. 37)• Literals (See <i>Literals (values)</i>, p. 38)
--------------------	---

Direct addresses The information on/display of direct addresses can be given in various formats. The display format is set in the dialog box **Options** → **Preferences** → **Common...**. Setting the display format has no impact on the entry format, i.e. direct addresses can be entered in any format.

The following address formats are possible:

- **Standard format (400001)**
The five-character address comes directly after the first digit (the Reference).
- **Separator format (4:00001)**
The first digit (the Reference) is separated from the following five-character address by a colon (:).
- **Compact format (4:1)**
The first digit (the Reference) is separated from the following address by a colon (:), and the leading zeros of the address are not given.
- **IEC format (QW1)**
In first place, there is an IEC identifier, followed by the five-character address.
 - %0x12345 = %Q12345
 - %1x12345 = %I12345
 - %3x12345 = %IW12345
 - %4x12345 = %QW12345

Data types The data type of the actual parameter must match the data type of the input/output. The only exceptions are generic inputs/outputs, of which the data type is determined by the formal parameter. If all actual parameters consist of literals, a suitable data type is selected for the Function Block.

Initial values FFBs, which use actual parameters on the inputs that have not yet received any value assignment, work with the initial values of these actual parameters.

Unconnected inputs

Note: Unconnected FFB inputs are specified as "0" by default.
--

Text Object

At a Glance

Text can be positioned in the form of text objects using FBD Function Block language. The size of these text objects depends on the length of the text. The size of the object, depending on the size of the text, can be extended vertically and horizontally to fill further grid units. Text objects may not overlap with FFBS; however they can overlap with links.

Memory space

Text objects occupy no memory space on the PLC because the text is not downloaded onto the PLC.

7.3 Working with the FBD Function Block language

At a Glance

Overview

This section describes working with the FBD Function Block object language.

What's in this Section?

This section contains the following topics:

Topic	Page
Positioning Functions and Function Blocks	186
FFB Execution Order	187
Configuring Loops	189

Positioning Functions and Function Blocks

Selecting FFBs

Using **Objects** → **Select FFB...** you can open a dialog for selecting FFBs. This dialog is modeless, that is, it is not automatically closed once an FFB is positioned, but remains open until you close it. If you have several FBD sections open, and invoke the dialog, only one dialog box is opened that is available for all sections. The dialog box is not available for any other sections (non-FBD editor). If the FBD sections are changed into icons (minimize window), the dialog box is closed. If one of the FBD section icons is called up again, the dialog box is automatically re-opened.

The first time Concept is started the FFB is displayed oriented to the library. This means that, when selecting an FFB, the **Library** command button must first of all be used to select the corresponding library. Then you can select the corresponding **Group** in the list box. Now, you can select the required FFB from the EFB type list.

If you do not know which library/group the FFB required is in, you can invoke an FFB-oriented dialog with the **Sorted by FFB** command button. This contains all FFBs of all libraries and groups in an alphabetical list.

After each subsequent project start, the view you selected appears.

Once the FFB has been selected, its position in the section must be selected. The cursor becomes a small FFB and the cross shows the position (upper left corner of the FFB) where the FFB is positioned. The FFB is positioned by clicking on the left-hand mouse button.

Positioning FFBs (Functions and Function Blocks)

In the FBD function block language editor, the window appears with a logic grid. FFBs (See *Functions and Function Blocks (FFBs)*, p. 178) are aligned in this grid as they are positioned. If FFBs are positioned outside of the section frame or if there is overlapping with another FFB, an error warning will appear and the FFB will not be positioned. Actual parameters may overlap another object when being positioned at an FFB input/output, but they must not go outside the limits of the section frame. If a link to another FFB is established, this link is checked. If this link is not permitted, a message is received, and the link is not established. When links are created, overlaps and crossing with other links and FFBs are permitted. If an FFB is selected, the comment relating to it is displayed in the first column of the status line. If an actual parameter is selected, its name and, if applicable, its direct address, its I/O map and its comment are displayed in the first column of the status line.

Change FFB Type

With the **Objects** → **Replace FFBs...** menu command the FFBs already positioned in the section can be replaced with FFBs of another type (e.g. an AND with an OR). The variables given to the FFB remain if the data type and position of the inputs/outputs are the same as the "old" and the new FFB.

Note: FFBs with inputs / outputs of the ANY data type (generic FFBs) cannot be replaced.

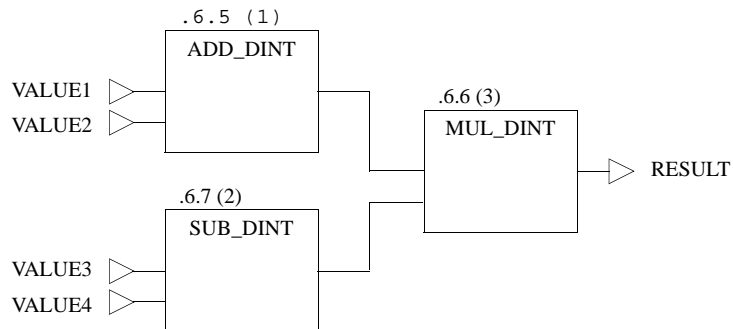
FFB Execution Order**Introduction**

The execution order is first determined by the order when positioning the FFB. If the FFBs are then linked graphically, the execution order is determined by the data flow.

Display FFB Execution Order.

The execution order can also be displayed with the **Objects** → **FFB Execution Order** menu command. This is represented by the execution number (number in brackets behind the instance name or function counter).

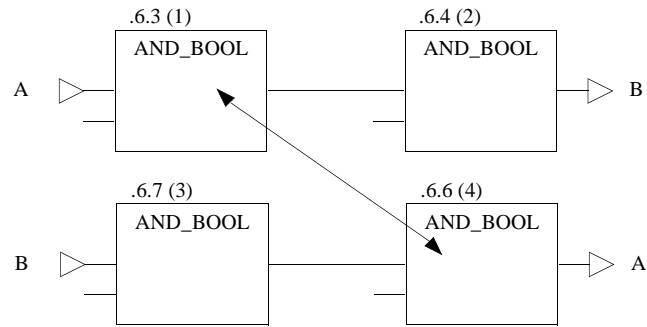
Display FFB Execution Order

**Reverse FFB Execution Order**

The execution order of two FFBs can be specifically switched afterwards with the menu command **Objects** → **Reverse FFB Execution Order**, but only if the rules regarding data flow are not broken.

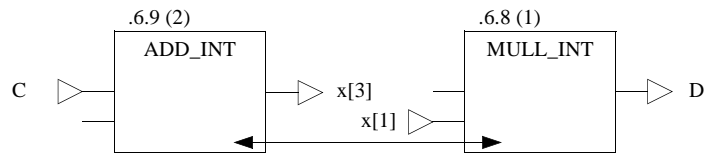
Switching the Execution Order of Two Networks within One Loop

The swap can be made by switching the two FFBs that are linked by the feedback variable of the loop.



Changing the Execution Order of FFBs Executed According to the Positioning Order

The switching operation permits the creation of a different, desired order (possibly step by step if several FFBs are involved).

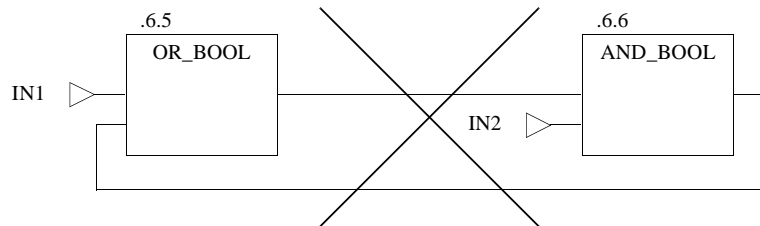


Configuring Loops

Non-permitted Loops

Configuring loops exclusively via links is not permitted, as it is not possible to uniquely set the data flow (the output of one FFB is the input of the next FFB, and the output of this one is the input of the first).

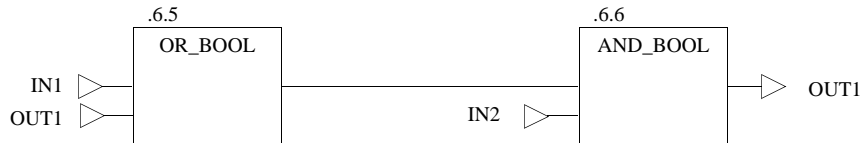
Non-permitted Loops via Links



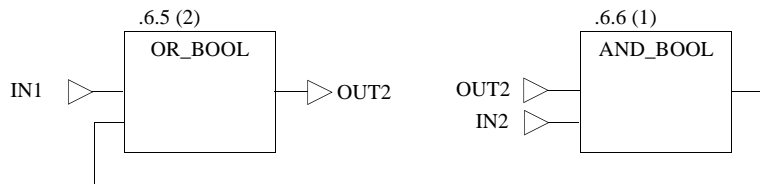
Resolution using an Actual Parameter

This type of logic must be resolved using actual parameters so that the data flow can be determined uniquely.

Resolved loop using an actual parameter: Variant 1



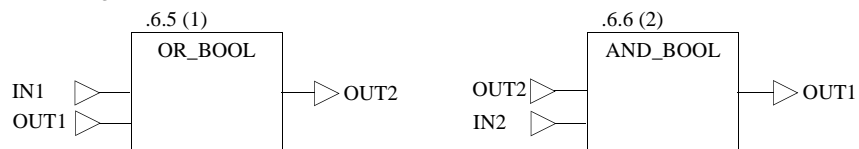
Resolved loop using an actual parameter: Variant 2



Resolution using Several Actual Parameters

Loops using several actual parameters are also allowed. With such loops, the execution order can later be influenced by executing – possibly several times – the menu command **Objects** → **Reverse FFB Execution Order** (see also *FFB Execution Order*, p. 187).

Loop using several actual parameters



7.4 Code generation with the FBD Function Block language

Code Generation Options

Introduction

Using the **Project** → **Code Generation Options** menu command, you can define options for code generation.

Include Diagnosis Information

If the **Include Diagnosis information** check box is checked, additional information for the process diagnosis (e.g. Transition Diagnosis (See *Transition diagnosis*, p. 277), diagnosis codes for diagnosis function blocks with extended diagnosis, such as e.g. XACT, XLOCK etc.) will be produced during code generation. This process diagnosis can be evaluated with MonitorPro or FactoryLink, for example.

Fastest Code (Restricted Checking)

If you check the **Fastest code (Restricted Checking)** check box, a runtime-optimized code is generated. This runtime optimization is achieved by realizing the integer arithmetic (e.g. "+" or "-") using simple CPU commands instead of EFB invocations.

CPU commands are much quicker than EFB invocations, but they do not generate any error messages, such as, for example, arithmetic or array overflow. This option should only be used when you have ensured that the program is free of arithmetic errors.

If **Fastest Code (Restricted Checking)** was selected, the addition $IN1 + 1$ is solved with the "add" CPU command. The code is now quicker than if the ADD_INT EFB were to be invoked. However, no runtime error is generated if "IN1" is 32767. In this case, "OUT1" would overrun from 32767 to -32768!

7.5 Online functions of the FBD Function Block language

Online Functions

Introduction

There are two animation modes available in the FBD editor:

- Animation of binary variables and links
- Animation of selected objects

These modes are also available on display of a DFB item (command button **Refine...** in the dialog box **Function block: xxx**).

Note: If the animated section is used as a transition section for SFC and the transition (and therefore also the transition section) is not processed, the status **DISABLED** appears in the animated transition section.

Animation of binary variables and links

The animation of binary variables and links is activated with the menu command **Online** → **Animate Booleans**.

In this mode, the current signal status of binary variables, direct addresses in the 0x and 1x range and binary links is displayed in the Editor window.

Animation of selected objects

The animation of the selected objects is activated with the menu command **Online** → **Animate selected**.

In this mode, the current signal status of the selected links, variables, multi-element variables and literals are displayed in the Editor window.

Note: If all variables/links of the section need to be animated, the whole section can be selected with **CTRL+A** and then **Online** → **Animate selected (CTRL+W)** all variables and links of the section will be animated.

If a numerical value is selected on an input/output, the name of the variable, its direct address and I/O assignment (if available) and its comment will be displayed in the status bar.

Note: The selected objects remain selected even after "Animate selected" has been selected again, in order to keep these for a further reading, and/or to be able to easily modify the list of objects.

Color key

There are 12 different color schemes available for animation. An overview of the color scheme and the meaning of each color can be found in the Online help (Tip: Search the online help for the index reference "Colors").

7.6 Creating a program with the FBD Function Block language

Creating a Program in the FBD Function Block Language

Introduction

The following description contains an example for creating a program in the function block language (FBD). The creation of a program in the function block language is divided into 2 main steps:

Step	Action
1	Creating a Section (See <i>Creating a Section</i> , p. 196)
2	Creating the Logic (See <i>Creating the Logic</i> , p. 197)

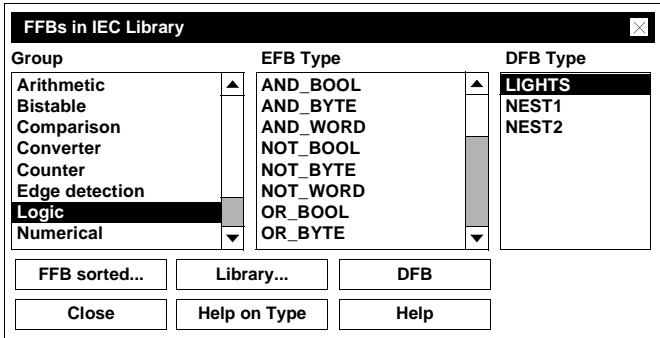
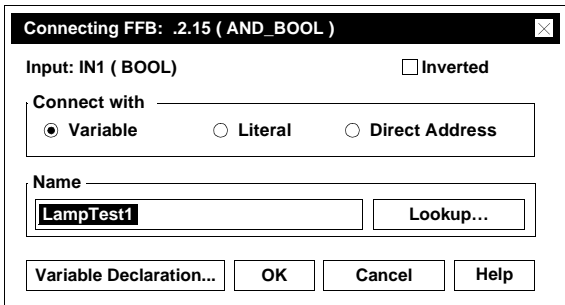
Creating a Section

The procedure for creating a section is as follows:

Step	Action
1	<p>Using the File → New Section... menu command, create a new section and enter a section name.</p> <p>Note: The section name (max. 32 characters) is not case-sensitive and must be unique within the whole project. If the name entered already exists, you will be warned and you will have to choose a different name. The section name must comply with the IEC name conventions, otherwise an error message appears.</p> <p>Note: In compliance with IEC1131-3 only letters are permitted as the first character of names. However, if you wish to use numbers as the first character, you can enable this using the Options → Preferences → IEC Extensions... → Allow Leading Digits in Identifiers menu command.</p>

Creating the Logic

The procedure for creating the logic is as follows:

Step	Action
1	<p>To insert an FFB into the section, select the Objects → Select FFB... menu command.</p> <p>Response: The FFB dialog box from the library is opened.</p> 
2	<p>In this dialog box you can select a library and an FFB from it by using the Library... command button. You can, however, also display the DFBs that you created and select one of them using the DFB command button.</p>
3	Place the selected FFB in the section.
4	When all FFBs have been placed, close the dialog box with Close .
5	Activate the selection mode with Objects → Select Mode , click on the FFB and move the FFBs to the desired position.
6	Activate the link mode with Objects → Link and connect the FFBs.
7	<p>Then re-activate select mode with Objects → Select Mode and double-click on one of the unconnected inputs/outputs.</p> <p>Response: The Connect FFB dialog box opens, where an actual parameter can be allocated to the input/output.</p> 

Step	Action
8	<p>Depending on the program logic you can allocate the following to the input/output:</p> <ul style="list-style-type: none">● Variable<ul style="list-style-type: none">● Located variable You can allocate a hardware input/output signal to the input/output of the FFB using a located variable. The name of the variable is shown at the input/output in the editor window.● Unlocated variable You can use the unlocated variable allocated to the input/output of the FFB as a discrete, i.e. when resolving loops, or when transferring values between different sections. The name of the variable is shown at the input/output in the editor window.● Constant You can allocate a constant to the input of the FFB. The constant can be transferred to other sections. You determine the value of the constant in the variable editor. The name of the constant is shown at the input in the editor window.● Literal You can allocate a literal to the input, i.e. directly allocate a value to the input/output. The value is shown at the input in the editor window.● Direct address You can allocate a hardware input/output signal to the input/output using an address. The address is shown at the input/output in the editor window. <p>Note: For an example for invocation of multi element variables see <i>Calling Derived Data Types, p. 524.</i></p> <p>Note: Unconnected FFB inputs are specified as "0" by default.</p>
9	Save the FBD section with the menu command File → Save Project .

Ladder Diagram LD



At a Glance

Overview

This Chapter describes the Ladder Diagram LD which conforms to IEC 1131.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
8.1	General information about Ladder Diagram LD	201
8.2	Objects in Ladder Diagram LD	204
8.3	Working with the LD Ladder Diagram	218
8.4	Code generation with LD Ladder Diagram	223
8.5	Online functions with the LD Ladder Diagram	225
8.6	Creating a program with LD Ladder Diagram	228

8.1 General information about Ladder Diagram LD

General Information about the LD Ladder Diagram Language

Introduction

This section describes the Ladder Diagram (LD) according to IEC 1131-3.

The structure of a LD section corresponds to a rung for relay switching. The window in the LD editor is shaded with a logic grid, on the left side of which there is the so-called left power rail. This left power rail corresponds to the phase (L ladder) of a rung. With LD programming, in the same way as in a rung, only the LD objects (contacts, coils) which are linked to a power supply, that is to say connected with the left power rail, are "processed". The right power rail, which corresponds to the neutral ladder, is not shown optically. However, all coils and FFB outputs are linked with it internally and this creates a power flow.

Objects

The objects of the programming language LD (Ladder Diagram) help to divide a section into a number of:

- Contacts (See *Contacts*, p. 205),
- Coils (See *Coils*, p. 206) and
- FFBs (Functions and Function Blocks) (See *Functions and Function Blocks (FFBs)*, p. 209).

These objects can be linked with each other through:

- Links (See *Link*, p. 214) or
- Actual Parameters (See *Actual Parameters*, p. 215).

Expansive logic can also be positioned in the LD section in the form of macros (related topics *Macros*, p. 455).

Theoretically, each section can contain as many FFBs and also as many inputs and outputs as required. It is therefore advisable to subdivide a whole program into logical units, that is to say into different sections.

Comments can be provided for the logic of the section with text objects (related topics *Text object*, p. 217).

Processing Sequence

The process sequence of the individual objects in a LD section is determined by the data flow within the section. Networks connected to the left power rail are processed from top to bottom (link with the left power rail). Networks that are independent of each other within the section are processed in order of positioning (from top to bottom) (related topics *Execution sequence*, p. 221).

Editing with the Keyboard

Normally editing in Concept is performed with the mouse, however it is also possible with the keyboard (related topics *Shortcut keys in the LD-Editor, p. 771*). In order to make editing with the keyboard easier, you can specify the number of columns per section in the CONCEPT.INI (See *INI Settings for the LD Section, p. 1035*) file, after which an automatic carriage return should appear when you are expanding a rung. This means that when you reach the last column, the next object is automatically placed in the second column of the next row. Objects on different rows are automatically linked, i.e. the objects are generated within a common rung.

IEC Conformity

For a description of the IEC conformity of the LD programming language see *IEC conformity, p. 779*.

8.2 Objects in Ladder Diagram LD

At a Glance

Overview This section describes the objects in LD Ladder Diagram.

What's in this Section? This section contains the following topics:

Topic	Page
Contacts	205
Coils	206
Functions and Function Blocks (FFBs)	209
Link	214
Actual Parameters	215
Text object	217

Contacts

At a Glance

A contact is an LD element that transfers a status on the horizontal link to its right side. This status comes from the boolean AND link of the status of the horizontal link on the left side, with the status of the relevant variable/direct address.

A contact does not change the value of the relevant variable/direct address.

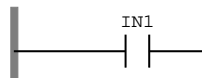
The following contacts are available:

- Closer (See *Closer*, p. 205)
- Opener (See *Opener*, p. 205)
- Contact for detection of positive transitions (See *Contact for detection of positive transitions*, p. 205)
- Contact for detection of negative transitions (See *Contact for detection of negative transitions*, p. 206)

Closer

On closing, the status of the left link is copied onto the right link, if the status of the relevant boolean variable is ON. Otherwise, the status of the right link is OFF.

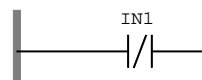
Closer



Opener

On opening, the status of the left link is copied onto the right link, if the status of the relevant boolean variable is OFF. Otherwise, the status of the right link is OFF.

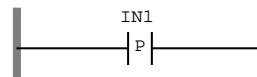
Opener



Contact for detection of positive transitions

With contacts for detection of positive transitions, the right link for a program cycle is ON if a transfer of the relevant boolean variable is made from OFF to ON and the status of the left link is ON at the same time. Otherwise, the status of the right link is OFF.

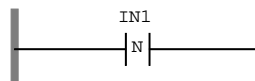
Contact for detection of positive transitions



Contact for detection of negative transitions

With contacts for detection of negative transitions, the right link for a program cycle is ON if a transfer of the relevant boolean variable is made from ON to OFF and the status of the left link is ON at the same time. Otherwise, the status of the right link is OFF.

Contact for detection of negative transitions



Coils

At a Glance

A coil is an LD element which transfers the status of the horizontal link on the left side, unchanged, to the horizontal link on the right side. The status is saved in the relevant variable/direct address.

Start behavior of coils

In the start behavior of PLCs there is a distinction between cold starts and warm starts:

- **Cold start**
Following a cold start (load the program with **Online** → **Download**) all variables (independent of type) are set to "0" or, if available, their initial value.
- **Warm start**
In a warm start (stop and start the program or **Online** → **Download changes**) different start behaviors are valid for located variables/direct addresses and unlocated variables:
 - **Located variables/direct addresses**
In a warm start all coils (0x registers) are set to "0" or, if available, their initial value.
 - **Unlocated variable**
In a warm start all unlocated variables retain their current value (storing behavior).

This different behavior in a warm start leads to particular characteristics in the warm start behavior of LD objects "Coil – set" and "Coil – reset". Warm start behavior is dependent on the variable type used (storing behavior in use of unlocated variables; non storing behavior in use of located variables/direct addresses)

If a buffered coil is required with a located variable or with direct addresses, the RS or SR Function Block from the IEC block library should be used.

Available coils

The following coils are available:

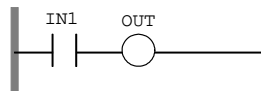
- Coil (See *Coil*, p. 207)
- Coil - negated (See *Coil - negated*, p. 207)
- Coil - set (See *Coil - set*, p. 208)
- Coil - reset (See *Coil - reset*, p. 208)
- Coil – positive edge (See *Coil – positive edge*, p. 207)
- Coil – negative edge (See *Coil – negative edge*, p. 208)

Coil

With coils, the status of the left link is copied onto the relevant Boolean variable and the right link.

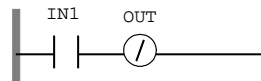
Normally, coils follow contacts or EFBs, but they can also be followed by contacts.

Coil

**Coil - negated**

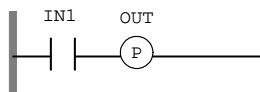
With negated coils, the status of the left link is copied onto the right link. The inverted status of the left link is copied onto the relevant Boolean variable. If the left link is OFF, then the right link will also be OFF and the relevant variable will be ON.

Coil - negated

**Coil – positive edge**

With coils for detection of positive transfers, the status of the left link is copied onto the right link. The relevant Boolean variable is ON for a program cycle, if a transfer of the left link from OFF to ON is made.

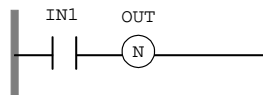
Coil – positive edge



Coil – negative edge

With coils for detection of negative transfers, the status of the left link is copied onto the right link. The relevant Boolean variable is ON for a program cycle, if a transfer of the left link from ON to OFF is made.

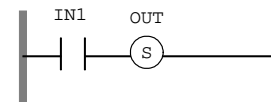
Coil – negative edge



Coil - set

With "set coils", the status of the left link is copied onto the right link. The relevant Boolean variable is set to ON status, if the left link is in ON status, otherwise it remains unchanged. The relevant Boolean variable can only be reset through the "reset coil".

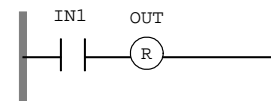
Coil - set



Coil - reset

With "reset coils", the status of the left link is copied onto the right link. The relevant Boolean variable is set to OFF status, if the left link is in ON status, otherwise it remains unchanged. The relevant Boolean variable can only be set through the "set coil".

Coil - reset



Functions and Function Blocks (FFBs)

Introduction

FFB is the generic term for:

- EFB (Elementary Function and Elementary Function Block) (See *EFB*, p. 209)
- DFB (Derived Function Block) (See *DFB*, p. 211)
- UDEFB (Derived Elementary Function and Derived Elementary Function Block) (See *UDEFB*, p. 212)

EFB

EFB is the generic term for:

- Elementary Function (See *Elementary Function*, p. 209)
- Elementary Function Block (See *Elementary Function Block*, p. 210)

EFBs are functions and function blocks that are available in Concept in the form of libraries. The logic of EFBs is built in C programming language and cannot be changed in the FBD editor.

Note: The EFBs AND_BOOL, NOT_BOOL, OR_BOOL, R_TRIG and F_TRIG are not available in LD. Their function is executed with contacts. The MOVE function cannot be used with the data type BOOL.

Elementary Function

Functions have no internal conditions. If the input values are the same, the value at the output is the same for all executions of the function. E.g. the addition of two values gives the same result at every execution.

An Elementary Function is represented graphically as a frame with inputs and outputs. The inputs are always represented on the left and the outputs always on the right of the frame. The name of the function, that is the function type, is displayed in the center of the frame. The function counter is displayed above the frame.

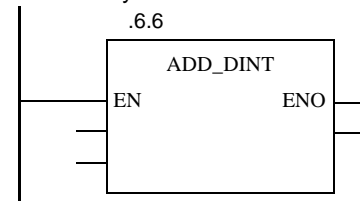
The function counter cannot be changed and always has an .n.m. structure.

.n = current section number

.m = current function number

Functions are only executed if the input EN=1 or if the input EN is grayed out (see also *EN and ENO*, p. 213).

Elementary Function



**Elementary
Function Block**

Function Blocks have internal conditions. If the inputs have the same values, the value at the output at every execution is another value. E.g. with a counter, the value on the output is incremented.

A function block is represented graphically as a frame with inputs and outputs. The inputs are always represented on the left and the outputs always on the right of the frame. The name of the function block, that is the function block type, is displayed in the center of the frame. The instance name is displayed above the frame. The instance name serves as a unique identification for the function block in a project.

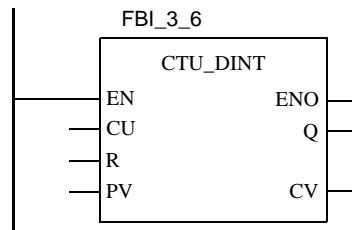
The instance name is produced automatically with the following structure: FBI_n_m
 FBI = Function Block Instance
 n = Section number (current number)
 m = Number of the FFB object in the section (current number)

The instance name can be edited in the Properties dialog box of the function block. The instance name must be unique throughout the whole project and is not case sensitive. If the name entered already exists, you will be warned and you will have to choose another name. The instance name must comply with the IEC name conventions otherwise an error message appears.

Note: In compliance with IEC1131-3 only letters are permitted as the first character of instance names. Should numbers be required as the first character however, the **Options** → **Preferences** → **IEC Extensions...** → **Allow leading digits in identifiers** menu command will enable this.

Function blocks are only executed if the input EN=1 or if the input EN is grayed out (see also *EN and ENO*, p. 213).

Elementary Function Block



DFB

Derived Function Blocks are function blocks that have been defined in Concept DFB.

With DFBs, there is no distinction between functions and function blocks. They are always treated as function blocks regardless of their internal structure.

A DFB is represented graphically as a frame with double vertical lines and with inputs and outputs. The inputs are always represented on the left and the outputs always on the right of the frame. The DFB name is displayed centrally within the frame. The instance name is displayed above the frame. The instance name serves as a unique identification for the function block in a project.

The instance name is produced automatically with the following structure: FBI_n_m
FBI = Function Block Instance

n = Section number (current number)

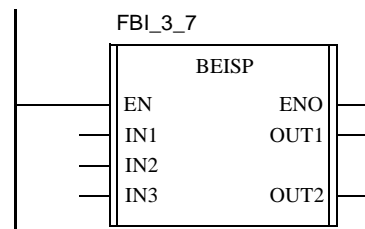
m = Number of the FFB object in the section (current number)

The instance name can be edited in the Properties dialog box of the DFB. The instance name must be unique throughout the whole project and is not case sensitive. If the name entered already exists, you will be warned and you will have to choose another name. The instance name must comply with the IEC name conventions otherwise an error message appears.

Note: In compliance with IEC1131-3 only letters are permitted as the first character of instance names. Should numbers be required as the first character however, the **Options** → **Preferences** → **IEC Extensions...** → **Allow leading digits in identifiers** menu command will enable this.

Derived Function Blocks are only executed if the input EN=1 or if the input EN is grayed out (see also *EN and ENO*, p. 213).

Derived Function Block



UDEFB

UDEFB is the generic term for:

- User-defined Elementary Function
- User-defined Elementary Function Block

UDEFBs are functions and function blocks that have been programmed with Concept EFB in C++ programming language and are available in Concept in the form of libraries.

In Concept, there is no functional difference between UDEFBs and EFBs.

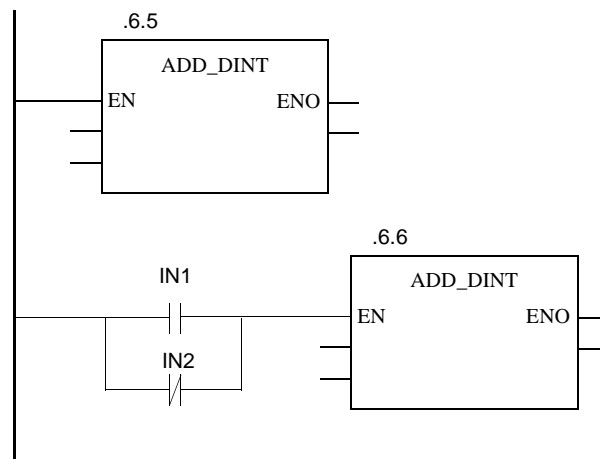
Editing FFBs

FFBs are only edited if at least one Boolean input is linked with the left power rail. If the FFB has no Boolean input, the EN input of the FFB must be used. If the FFB is to be conditionally executed, the Boolean input can be pre-linked through contacts or other FFBs.

Note: If the EN input is not linked with the left power rail, it must be deactivated in the Properties dialog box, otherwise the FFB will never be edited.

Note: Each FFB without Boolean link to the left power rail gives rise to an error message when downloading onto the PLC.

Connection to an FFB with the left power rail:



EN and ENO

With all FFBs, an EN input and an ENO output can be configured.

EN and ENO configuration is switched on or off in the FFB properties dialog box. The dialog box can be invoked with the **Objects** → **Properties...** menu command or by double-clicking on the FFB.

If the value of EN is equal to "0" when the FFB is invoked, the algorithms that are defined by the FFB will not be executed and all outputs keep their previous values. The value of ENO is automatically set to "0" in this case.

If the value of EN is equal to "1", when the FFB is invoked, the algorithms which are defined by the FFB will be executed. After successful execution of these algorithms, the value of ENO is automatically set to "1". If an error occurs during execution of these algorithms, ENO will be set to "0".

<p>Note: If the EN input is not linked with the left power rail, it must be deactivated in the Properties dialog box, otherwise the FFB will never be edited.</p>
--

The output behavior of the FFBs does not depend on whether the FFBs are invoked without EN/ENO or with EN=1.

Link

Description

Links are connections between contacts, coils and FFBs.

Several links can be connected with one contact, one coil or one FFB output. The link points are identified with a filled circle.

Note: Unconnected contacts, coils and FFB inputs are specified as "0" by default.

Data Types

The data types of the inputs/outputs to be linked must be the same.

Editing Links

Links can be edited in select mode. An overlap with other objects is permitted.

Configuring Loops

No loop can be configured with links because in this case, the execution order in the section cannot be determined uniquely. Loops must be resolved with actual parameters (related topics *Configuring Loops*, p. 189).

Horizontal Links

Contacts and coils are automatically connected during positioning with a neighboring, unconnected contact/coil that has the same vertical position. A connection to the power rail is only established if the contact is placed nearby (also see *Defining the Contact Connection*, p. 1035 in the *Concept INI-File* chapter). If a coil or a contact is positioned on an existing horizontal link, the link is automatically separated and the contact/coil is inserted. When positioned, actual parameters may overlap another object, but they must not go outside the limits of the section frame. If a link to another object is established, this link is checked. If this link is not permitted, you will receive a message and the link will not be generated.

Once objects are positioned, horizontal links with directly adjacent objects are automatically created.

Vertical Links

An exceptional link is the "vertical link". The vertical link serves as a logical OR. With this form of the OR link, 32 inputs (contacts) and 64 outputs (coils, links) are possible.

Actual Parameters

Possible Actual Parameters

In the program runtime, the values from the process or from other actual parameters are transferred to the FFB via the actual parameters and then re-emitted after processing.

Table of possible actual parameters

Element	Actual Parameters
Contacts	<ul style="list-style-type: none"> ● Direct addresses (See <i>Direct addresses</i>, p. 39) ● Located variables (See <i>Variables</i>, p. 36) ● Unlocated variable (See <i>Variables</i>, p. 36)
Coils	<ul style="list-style-type: none"> ● Direct addresses (See <i>Direct addresses</i>, p. 39) ● Located variables (See <i>Variables</i>, p. 36) ● Unlocated variable (See <i>Variables</i>, p. 36)
FFB inputs	<ul style="list-style-type: none"> ● Direct addresses (See <i>Direct addresses</i>, p. 39) ● Located variables (See <i>Variables</i>, p. 36) ● Unlocated variable (See <i>Variables</i>, p. 36) ● Constant (See <i>Constant variables</i>, p. 37) ● Literals (See <i>Literals (values)</i>, p. 38)
FFB outputs	<ul style="list-style-type: none"> ● Direct addresses (See <i>Direct addresses</i>, p. 39) ● Located variables (See <i>Variables</i>, p. 36) ● Unlocated variable (See <i>Variables</i>, p. 36)

Direct Addresses The information on/display of direct addresses can be given in various formats. The display format is set in the **Options** → **Preferences** → **Common** dialog box. Setting the display format has no impact on the entry format, i.e. direct addresses can be entered in any format.

The following address formats are possible:

- **Standard Format (400001)**
The five figure address comes directly after the first digit (the reference).
- **Separator Format (4:00001)**
The first digit (the reference) is separated from the five figure address that follows by a colon (:).
- **Compact format (4:1)**
The first digit (the Reference) is separated from the address that follows by a colon (:) where the leading zeros are not specified.
- **IEC Format (QW1)**
There is an IEC type designation in initial position, followed by the five-character address.
 - %0x12345 = %Q12345
 - %1x12345 = %I12345
 - %3x12345 = %IW12345
 - %4x12345 = %QW12345

Data Types The data type of the actual parameter must be of BOOL type with contacts and coils. With FFB inputs/outputs, the data type of the actual parameter must match the data type of the inputs/outputs. The only exceptions are generic FFB inputs/outputs, whose data type is determined by the formal parameter. If all actual parameters consist of literals, a suitable data type is selected for the function block.

Initial Values FFBs, which use actual parameters on the inputs and coils that have not yet received a value assignment, work with the initial values of these actual parameters.

Unconnected Inputs

Note: Unconnected contacts, coils and FFB inputs/outputs are specified as "0" by default.

Text object

At a Glance

Text can be positioned in the form of text objects in the Ladder Diagram (LD). The size of these text objects depends on the length of the text. The size of the object, depending on the size of the text, can be extended vertically and horizontally to fill further grid units. Text objects may not overlap with other objects; however they can overlap with links.

Memory space

Text objects occupy no memory space on the PLC because the text is not downloaded onto the PLC.

8.3 Working with the LD Ladder Diagram

At a Glance

Overview This section describes working with LD Ladder Diagram.

What's in this Section? This section contains the following topics:

Topic	Page
Positioning Coils, Contacts, Functions and Function Blocks	219
Execution sequence	221
Configuring Loops	221

Positioning Coils, Contacts, Functions and Function Blocks

Positioning Objects

In the LD contact plan editor, the window has a logic grid in the background. The objects are aligned in the bars of this grid (52 x 230 fields) during positioning. With the exception of vertical shorts, FFBs and text fields, all elements require exactly one grid field. Objects can only be positioned within such a field. If an object is positioned between two fields, the object is automatically placed in the nearest field.

When objects are positioned outside the section frame with another object, an error message occurs and the object is not positioned.

When being positioned, contacts and coils are automatically linked with a directly adjacent, unconnected contact/coil, if the contact/ coil has the same vertical position. A link to the power rail is therefore created even if the contact is positioned 2 fields away. If contacts or coils are positioned on existing contacts or coils, the existing ones are replaced by the current ones (only applies to same types, i.e. when replacing coils with coils and contacts with contacts). If a coil or a contact is positioned on an existing horizontal short, the link is automatically separated and the contact/coil is inserted.

When positioned, actual parameters may overlap another object, but they must not go outside the limits of the section frame. If a link to another object is established, this link is checked. If this link is not permitted, you will receive a message and the link will not be generated. When producing links, overlaps and crossings with other links and objects are permitted.

If an FFB is selected, its comment is displayed in the first column of the status line. If an actual parameter is selected, its name and, if applicable, its direct address and its comment are displayed in the first column of the status line.

Automatic Carriage Return

As a keyboard user, you have the possibility of determining the number of columns/ fields in the CONCEPT.INI (See *Defining the Number of Columns/Fields*, p. 1035) file after which an automatic carriage return will appear during editing as soon as the last column/field is reached. The following object is then inserted into the second column/field and linked to the last object of the previous row. I.e. the objects are created inside the same rung.

Selecting FFBs

Using **Objects** → **Select FFB...** you can open a dialog for selecting FFBs. This dialog is modeless, which means it is not automatically closed once an FFB has been positioned, but remains open until you close it. If you have several LD sections open and you invoke the dialog, only one dialog box is opened and is available for all sections. The dialog box is not available for any other sections (not LD editor). If the LD sections are changed into symbols (Minimize window), the dialog box is closed. If one of the LD section symbols is invoked again, the dialog box is automatically re-opened.

The first time Concept is started, the FFB is displayed oriented to the library. This means that to select an FFB, the corresponding library must first be selected using the **Library** command button. Then you can select the corresponding group in the **Group** list box. Now, you can select the required FFB from the EFB type list box.

If you do not know which library/group the FFB required is located in, you can invoke an FFB-oriented dialog with the **FFB sorted** command button. This contains all FFBs in all libraries and groups in an alphabetical list.

After each subsequent project start, the view that you select will appear.

Once the FFB has been selected, its position in the section must be selected. The cursor becomes a small FFB and the cross shows the position (upper left corner of the FFB) in which the FFB is placed. The FFB is positioned by clicking on the left-hand mouse button.

Change FFB-Type

With the **Objects** → **Replace FFBs...** menu command, the FFBs already positioned in the section can be replaced with FFBs of another type (e.g. an AND with an OR). The variables given to the FFB remain if the data type and position of the inputs/outputs are the same in the "old" as the new FFB.

<p>Note: FFBs with inputs/outputs of the ANY data type (generic FFBs) cannot be replaced.</p>
--

Change contact/coil

Contacts and coils which are already positioned can simply be replaced. In order to do this, select the new element and click on the one to be replaced.

Execution sequence

Description

The execution sequences of contacts, coils and FFBs are determined by the data flow. This means that the coils and FFBs whose inputs have already received value assignments will be processed first.

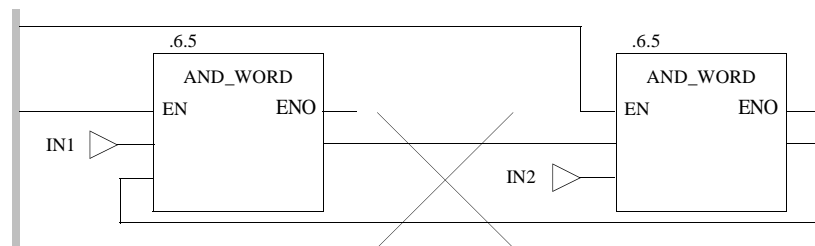
The execution sequence of networks which are only linked by the left power rail, is determined by the graphic sequence (from top to bottom) in which these are connected to the left power rail.

Configuring Loops

Non-permitted Loops

Configuring loops exclusively via links is not permitted, as it is not possible to make a unique specification of the data flow (the output of one FFB is the input of the next FFB, and the output of this one is the input of the first).

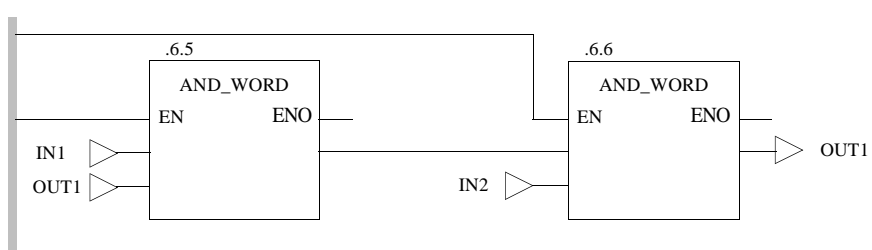
Non-permitted Loops via Links



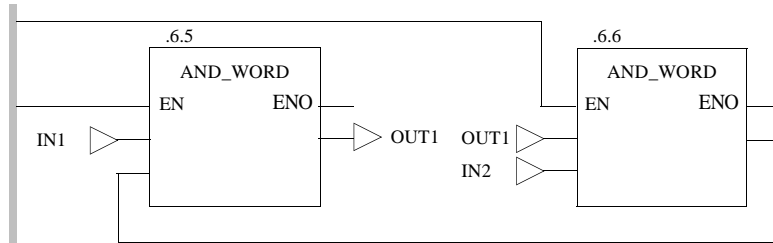
Resolution using an Actual Parameter

This type of logic must be resolved using actual parameters so that the data flow can be determined uniquely.

Resolved loop using an actual parameter: Variant 1

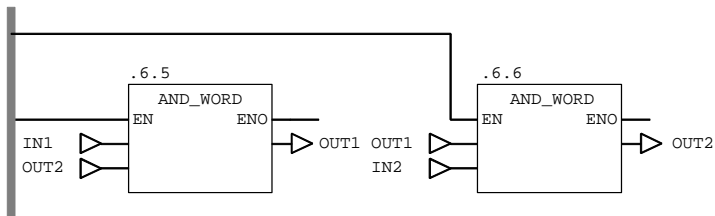


Resolved loop using an actual parameter: Variant 2



**Resolution using
Several Actual
Parameters**

Loops using several actual parameters are also allowed.
Loop using several actual parameters



8.4 Code generation with LD Ladder Diagram

Code Generation Options

Introduction

Using the **Project** → **Code Generation Options** menu command, you can define options for code generation.

Include Diagnosis Information

If you check the **Include Diagnosis Information** check box, additional information for the process diagnosis (e.g. transition diagnosis, diagnosis codes for diagnosis function blocks with extended diagnosis, such as XACT, XLOCK etc.) will be created during code generation. This process diagnosis can be evaluated with MonitorPro or FactoryLink, for example.

Fastest Code (Restricted Checking)

If you check the **Fastest code (Restricted Checking)** check box, a runtime-optimized code is generated. This runtime optimization is achieved by realizing the integer arithmetic (e.g. "+" or "-") using simple CPU commands instead of EFB invocations.

CPU commands are much quicker than EFB invocations, but they do not generate any error messages, such as, for example, arithmetic or array overflow. This option should only be used when you have ensured that the program is free of arithmetic errors.

If **Fastest Code (Restricted Checking)** was selected, the addition $IN1 + 1$ is solved with the "add" CPU command. The code is now quicker than if the ADD_INT EFB were to be invoked. However, no runtime error is generated if "IN1" is 32767. In this case, "OUT1" would overrun from 32767 to -32768!

8.5 Online functions with the LD Ladder Diagram

Online Functions

Introduction

There are two animation modes available in the LD editor:

- Animation of binary variables and links
- Animation of selected objects

These modes are also available when a DFB instance is displayed (command button **Refine...** in the **Function Block: xxx** dialog box).

Note: If the animated section is used as a transition section for SFC and the transition (and therefore also the transition section) is not processed, the status **DISABLED** appears in the animated transition section.

Animation of Binary Variables and Links

The animation of binary variables and links is activated using the **Online** → **Animate Booleans** menu command.

In this mode, the current signal status of binary variables, direct addresses in the 0x and 1x range and binary links is displayed in the editor window.

Meaning of Colors

Color	Meaning
Contact, coil, input/output, link red	Contact, coil, input/output, link transferring the value 0
Left power rail, contact, coil, input/output, link green	Left power rail, contact, coil, input/output, link transferring the value 1
Variable highlighted in beige	Variable forced
Variable highlighted in purple	Variable cyclically set
The name of the multi-element variable (e.g. motor) highlighted in color.	In the editor, a multi-element variable (e.g. motor) is displayed, in which one or more elements is forced or cyclically set.
The whole element name of the multi-element variable (e.g. right.motor.on) is highlighted in color.	In the editor, an element of a multi-element variable (e.g. right motor on) that is forced or cyclically set is displayed.
The name of the multi-element variable (e.g. right.motor.on) is highlighted in color, but the name of the element is not.	In the editor, an element of a multi-element variable (e.g. right motor on) that is not forced or cyclically set is displayed, but a different element of this multi-element variable is cyclically set or forced.

Animation of Selected Objects

The animation of the selected objects is activated with the **Online → Animate Selection** menu command.

In this mode, the current signal status of the selected links, variables, multi-element variables and literals is displayed in the editor window.

Note: If you want to animate all variables/links in the section, you can select the whole section using **CTRL+A** and then animate all variables and links in the section using **Online → Animate Selection (CTRL+W)**.

If a numerical value is selected on an input/output, the name of the variable, its direct address and I/O mapping (if existent) and its comment will be displayed in the status bar.

Note: The selected objects remain selected even after "animate selection" has been selected again, to retain these objects for a further reading, and/or to be able to easily modify the list of objects.

Color key

There are 12 different color schemes available for animation. An overview of the color scheme and the meaning of each color can be found in the Online help Tip: Search the online help for the index reference "Colors").

8.6 Creating a program with LD Ladder Diagram

Creating a Program in LD

Introduction

The following description contains an example for creating a program in Ladder Diagram (LD). The creation of a program in LD Ladder Diagram is divided into 2 main steps:

Step	Action
1	Creating a Section (See <i>Creating a Section</i> , p. 229)
2	Creating the Logic (See <i>Creating the Logic</i> , p. 230)

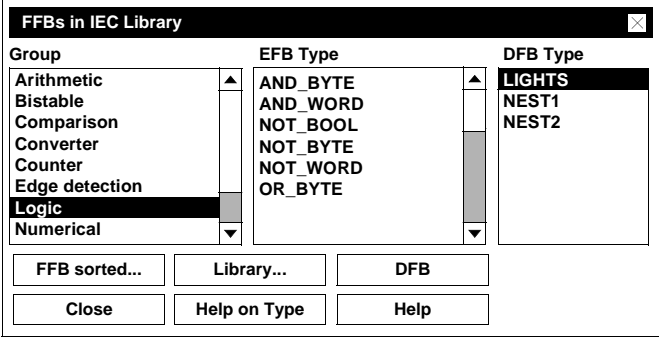
Creating a Section

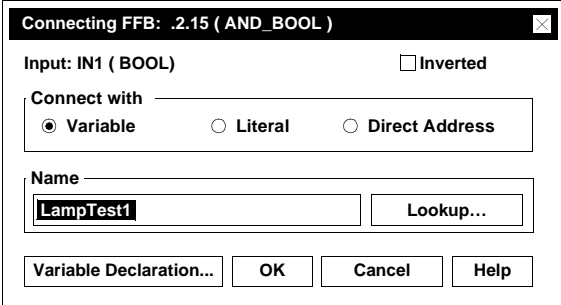
The procedure for creating a section is as follows:

Step	Action
1	<p>Using the File → New Section... menu command, create a new section and enter a section name.</p> <p>Note: The section name (max. 32 characters) is not case-sensitive and must be unique within the whole project. If the name entered already exists, you will be warned and you will have to choose a different name. The section name must comply with the IEC name conventions, otherwise an error message appears.</p> <p>Note: In compliance with IEC1131-3 only letters are permitted as the first character of names. However, if you wish to use numbers as the first character, you can enable this using the Options → Preferences → IEC Extensions... → Allow Leading Digits in Identifiers menu command.</p>

Creating the Logic

The procedure for creating the logic is as follows:

Step	Action
1	To insert a contact or coil in the section, open the Objects main menu and select the desired contact or coil. Contacts and coils can also be selected using the tool bar. Place the contact or coil in the section.
2	To insert an FFB into the section, select the Objects → Select FFB... menu command. Response: The FFBs from Library dialog box is opened. 
3	In this dialog box you can select a library and an FFB from it by using the Library... command button. You can, however, also display the DFBs that you created and select one of them using the DFB command button.
4	Place the selected FFB in the section.
5	When all FFBs have been placed, close the dialog box with Close .
6	Activate select mode using Objects → Select Mode , and move the contacts, coils and FFBs to the required position.
7	Activate link mode with Objects → Link , and connect the contacts, coils and FFBs. Connect the contacts, FFBs and the left power rail.
8	Then re-activate select mode with Objects → Select mode , and double-click on a contact or coil. Response: The Properties: LD objects dialog box is opened, in which you can allocate an actual parameter to the contact/coil.

Step	Action
9	<p>Depending on the program logic you can allocate the following to the contact/coil:</p> <ul style="list-style-type: none"> ● Variable <ul style="list-style-type: none"> ● Located variable You can allocate a hardware input/output signal to the input/output using a located variable. The name of the variable is shown at the input/output in the editor window. ● Unlocated variable You can use the unlocated variable allocated to the input/output as a discrete, i.e. to resolve loops, or to transfer values between different sections. The name of the variable is shown at the input/output in the editor window. ● Direct address You can allocate a hardware input/output signal to the input/output using an address. The address is shown at the input/output in the editor window. <p>Note: For an example for invocation of multi-element variables see <i>Calling Derived Data Types</i>, p. 524.</p> <p>Note: Unconnected FFB inputs are specified as "0" by default.</p>
10	<p>To connect the FFB input/outputs to the actual parameters, double-click on one of the unconnected input/outputs.</p> <p>Response: The Connect FFB dialog box is opened, in which you can allocate an actual parameter to the input/output.</p> 

Step	Action
11	<p>Depending on the program logic you can allocate the following to the input/output:</p> <ul style="list-style-type: none">● Variable<ul style="list-style-type: none">● Located variable You can allocate a hardware input/output signal to the input/output using a located variable. The name of the variable is shown at the input/output in the editor window● Unlocated variable You can use the unlocated variable allocated to the input/output as a discrete, i.e. to resolve loops, or to transfer values between different sections. The name of the variable is shown at the input/output in the editor window.● Constant You can allocate a constant to the input. The constant can be transferred to other sections. You determine the value of the constant in the variable editor. The name of the constant is shown at the input in the editor window.● Literal You can allocate a literal to the input, i.e. directly allocate a value to the input/output. The value is shown at the input in the editor window.● Direct address You can allocate a hardware input/output signal to the input/output using an address. The address is shown at the input/output in the editor window. <p>Note: For an example for invocation of multi-element variables see <i>Calling Derived Data Types</i>, p. 524. Note: Unconnected FFB inputs are specified as "0" by default.</p>
12	Save the LD section using the File → Save Project menu command.

Sequence language SFC

9

At a Glance

Overview

This Chapter describes the sequence language SFC which conforms to IEC 1131.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
9.1	General information about SFC sequence language	235
9.2	SFC sequence language elements	237
9.3	Working with the SFC Sequence Language	253
9.4	Online functions of the SFC sequence language	269

9.1 General information about SFC sequence language

General information about SFC language

At a Glance	<p>The sequence language SFC is described in this section according to IEC 1131-3.</p> <p>In the SFC (Sequential Function Chart) sequence language, a section is split into single configured sequential steps, through steps and transitions, which alternate in the sequence plan.</p>
Objects	<p>A sequential control uses the following objects when creating a program:</p> <ul style="list-style-type: none">● Step (See <i>Step</i>, p. 238)● Transition (See <i>Transition</i>, p. 242)● Jump (See <i>Jump</i>, p. 246)● Connection (See <i>Link</i>, p. 245)● Alternative branch (See <i>Alternative Branch</i>, p. 248)● Simultaneous branch (See <i>Parallel branch</i>, p. 251)● Alternative connection (See <i>Alternative connection</i>, p. 250)● Parallel connection (See <i>Parallel connection</i>, p. 252)● Text object (See <i>Text object</i>, p. 252)
Structure of an SFC section	<p>Steps and transitions are linked with one another through directional links. Two steps can never be directly linked, and must always be separated by a transition. The processes of the active signal status take place along the directional links, triggered by the connecting of a transition. The direction of the string process follows the directional links and runs from the under side of the predecessor step to the top side of the successive step. Branches are processed from left to right.</p> <p>A jump can be put in the place of a step. Step strings are always concluded with a jump to another step on the same step string. It is run down cyclically.</p> <p>Nil or more action belong to every step. Steps without action are known as waiting steps. A condition for transition belongs to every transition.</p>
Editing with the keyboard	<p>Normally editing in Concept is performed with the mouse, however it is also possible with the keyboard (see also <i>Short Cut Keys in the FBD and SFC Editor</i>, p. 768)</p>
IEC conformity	<p>For a description of the IEC conformity of the SFC programming language see <i>IEC conformity</i>, p. 779.</p>

9.2 SFC sequence language elements

At a Glance

Overview

This section describes the SFC sequence language elements.

What's in this Section?

This section contains the following topics:

Topic	Page
Step	238
Action	240
Transition	242
Transition section	243
Link	245
Jump	246
Alternative Branch	248
Alternative connection	250
Parallel branch	251
Parallel connection	252
Text object	252

Step

Introduction

A step is represented using a block that contains a step name. Step names must be unique within the project.

A step becomes active when the upstream transition is satisfied and is normally inactive when the downstream transition is satisfied.

Initial Step

A special case with steps is the initial step. The initial status of a SFC section is characterized by the initial step, which is active when initializing the project containing the section. A step in a SFC section must always be defined as an initial step. In Concept it is possible to define a step in the middle of a step string as initial.

The initial step is denoted by double lined borders.

Waiting Step

Zero or more actions belong to every step. Steps without action are known as waiting steps.

Step Delay Time

A time can be entered, which is the least amount of time the step must be active for. This is called the step delay time (step duration).

Note: This time is only applicable to the step, not for the actions allocated to it. Individual times can be defined for these.

Maximum Supervision Time

The maximum supervision time specifies the maximum time in which the step should normally be active. If the step is still active after this period of time, an error message occurs, which you can view using the **Online** → **Event Viewer**. In animation mode, the error is additionally identified by a colored outline around the step object.

Note: This time supervision applies only to the step, not to the actions allocated to it. Individual times can be defined for these.

Minimum Supervision Time

The minimum supervision time sets the minimum time for which the step should normally be active. If the step is still active after this period of time, an error message occurs, which you can view in the **Online** → **Event Viewer**. In animation mode, the error is additionally identified by a colored outline around the step object.

Note: This time supervision applies only to the step, not to the actions allocated to it. Individual times can be defined for these.

Coordinating the Times

Step delay time < minimum supervision time < maximum supervision time

Setting the Times

In the properties dialog, the time values can be entered directly as time literals or can be set as multi element variables of data type SFCSTEP_TIMES. The values can be automatically determined in learn supervision time mode.

The time literals can be modified in animation mode.

SFCSTEP_TIMES Variable

In 'SFCSTEP_TIMES' variable usage, the learned times of these variables are assigned as the initial values. If these initial values are to be used for a long period of time, corresponding elements (min., max.) of these variables must not be written. After the supervision times have been learned, the modified initial values must be downloaded to the PLC using **Online** → **Download Changes**.

The 'SFCSTEP_TIMES' variable can be used everywhere and has the following structure:

```
'varname': SFCSTEP_TIMES
  delay: TIME
  min: TIME
  max: TIME
```

The elements have the following meaning:

- 'varname'.delay = delay time
- 'varname'.min = minimum supervision time
- 'varname'.max = maximum supervision time

Step Variable

Every step is implicitly allocated a (read only) variable of data type SFCSTEP_STATE. This step variable has the name of the allocated step. The step variable can be used everywhere and has the following structure:

```
'Step name': SFCSTEP_STATE
  t: TIME
  x: BOOL
  tminErr: BOOL
  tmaxErr: BOOL
```

The elements have the following meaning:

- 'Step name'.t = current dwell time in step
 - 'Step name'.x
 - 1: Step active
 - 0: Step inactive
 - 'Step name'.tminErr
 - 1: Underflow of minimum supervision time
 - 0: No underflow of minimum supervision time
 - 'Step name'.tmaxErr
 - 1: Overflow of maximum supervision time
 - 0: No overflow of maximum supervision time
-

Action

At a Glance

The actions, which are to be performed, as the step is active must be connected to the step.

Actions are declared in the properties dialog of the triggering step, see *Declaring actions*, p. 260.

A step can be assigned none or several actions. A step which is assigned no action, has a waiting function, i.e. it waits until the assigned transition is completed.

An action is a variable of BOOL data type.

The control of actions is expressed through the use of identifiers.

Signal assignment

The following signals can be assigned to an action:

- **Direct address**

An action can be assigned a hardware output via a direct address. In this case, the action can be used as an enabling signal for a transition, as an input signal in another section and as an output signal for the hardware.

- **Variable**

The action can be used as an input signal with assistance from a variable in another section. This variable is also called action variable.

- **Unlocated variable**

With Unlocated variable the action can be used as an enabling signal for a transition and as an input signal in an FBD section. Unlocated variables are declared in the Variable Editor (See *Variables editor, p. 479*).

- **Located variable**

With Located variable the action can be used as an enabling signal for a transition, as an input signal in another section and as an output signal for the hardware. Located variables are declared in the Variable Editor (See *Variables editor, p. 479*).

Direct addresses

The information on/display of direct addresses can be given in various formats. The display format is set in the dialog box **Options** → **Preferences** → **Common...** Setting the display format has no impact on the entry format, i.e. direct addresses can be entered in any format.

The following address formats are possible:

- **Standard format (X00001)**

The five-character address comes directly after the first digit (the Reference).

- **Separator format (X:00001)**

The first digit (the Reference) is separated from the following five-character address by a colon (:).

- **Compact format (X:1)**

The first digit (the Reference) is separated from the following address by a colon (:), and the leading zeros of the address are not given.

- **IEC format (XW1)**

In first place, there is an IEC identifier, followed by the five-character address.

- %0x12345 = %Q12345
- %1x12345 = %I12345
- %3x12345 = %IW12345
- %4x12345 = %QW12345

Transition

Introduction

A transition specifies the condition through which the check of one or more pre-transition steps passes on to one or more consecutive steps along the corresponding link.

Transition Condition

A transition condition is one of the variables of data type BOOL allocated to the transition.

Transition conditions are declared in the properties dialog of the transition, see also *Declaring a Transition*, p. 264.

The transition condition can be:

- a direct address (input or output),
- a variable (input or output) or
- a Transition Section (See *Transition section*, p. 243).

Variable name position:

If...	Then...
If you allocate a direct address or a variable to the transition.	Then the name of the address/variable is displayed below the transition icon.
If you allocate a transition section to the transition.	Then the name of the transition section is displayed above the transition icon.

Note: The variable or address allocated to the transition is only read by the transition, never written.

Enabling a Transition

A transition is enabled if the steps immediately preceding it are active. Transitions whose immediately preceding steps are not active are not analyzed.

Note: If no transition condition is defined, the transition will never be active.

Transition Switch Time

The transition switch time can theoretically be as short as possible, but can never be zero. The transition switch time lasts at least the duration of the scan.

Transition Diagnosis Transition switching can be supervised by the Transition Diagnosis (See *Transition diagnosis*, p. 277).

Transition Trigger Sweep Transition trigger sweep occurs when the transition is enabled and the associated transition conditions are satisfied.
 Triggering a transition leads to the disabling (resetting) of all immediately preceding steps that are linked to the transition, followed by the activation of all immediately following steps.
 If triggering a transition leads to the activation of several steps at the same time, then the sequence belonging to these steps is called Parallel Chain (See *Parallel branch*, p. 251). After simultaneous activation, each of these chains is processed independently of each other. To emphasize this specific type of construction, the branch and connection of parallel chains are displayed with a double horizontal line.

Transition section

At a Glance For every Transition (See *Transition*, p. 242) a transition section can be created. This is a section containing the logic of the transition condition and it is automatically linked with the transition.

Generating a transition section Transition sections are generated in the properties dialog of the transition, see also *Declaring a Transition*, p. 264.

Name of transition section Name of transition section:

If...	Then...
If in the dialog Options → Presettings... the option Dynamically enumerated has been selected.	Then the alias designation of the transition is displayed in the Transition properties dialog automatically.
Should a name for the transition section be entered manually.	Please ensure that the name is unique throughout the whole project (the name is not case-sensitive). If the section name entered already exists, a warning is given, and another name must be chosen. The name must correspond to the IEC Name conventions, otherwise an error message appears.

Note: Do NOT alter the name of a transition section through **Data file** → **Section properties**, otherwise the link to the transition is will be lost.

Occupying a transition section

When first opening the transition section (**Edit...** key in the **Transition properties** dialog) this is automatically generated. The name of the transition section is displayed above the transition symbol in the SFC editor.

Altering the transition conditions

Should another option be selected after the creation of the transition section as **Transition section**, a query appears, whether the transition section should be deleted. If the question is replied in the negative, the transition section remains.

A list can be displayed with the currently unused transition section with help from the command button **Look up...** .

Programming languages for transition section

FBD, LD, IL and ST are possible as programming languages for transition sections.

The programming language to be used can be defined in the dialog **Options** → **Preferences** → **Common...** with the option **Editor type of Transition Sections**. Should the FBD programming language be selected, the section is automatically preallocated with a UND block with 2 inputs whose outputs is preallocated with the name of the transition section. The proposed block can then be linked or altered. No such provision is evident for the other programming languages.

Editing function for transition section

The editing function for transition sections is restricted as opposed to "normal" sections in the following ways:

- The transition section only has one single output (transition variable), whose data type is BOOL. The name of this variable must be identical to the name entered in the **Transition section** field.
 - The transition variable can only be used once in written form.
 - Only functions can be used, Function Blocks cannot.
 - There is only one network, i.e. all functions used are linked with each other either directly or indirectly.
 - Transition sections can only be reached via the menu command button **Edit...** in the **Transition properties** dialog. They do not appear in the **Open section** dialog.
 - In the **Delete section** dialog transition sections are denoted by a "T" in front of the section name.
-

Transition section animation

If the transition, and therefore the transition section, is not processed, the status INHIBITED appears in the animated transition section.

Link

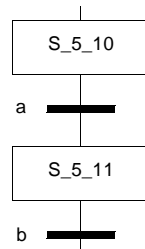
At a Glance

Links connect steps and transitions. Links are normally generated automatically when positioning objects. If objects are positioned in cells which do not immediately follow each other, a link must explicitly be made.

Simple sequences

The change of step and transition is consequentially repeated with simple sequences.

A process of S_5_10 to S_5_11 only takes place, if step 5_10 is in an active state and the condition for transition a is true.



Jump

General information

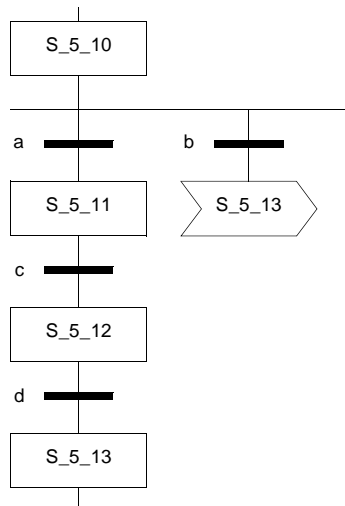
A jump enables a program to continue in another place. Jumps into a Parallel chain (See *Parallel branch*, p. 251) in or out of a parallel chain are not possible.

Differences are made between chain jumps and chain loops with jumps.

Chain jump

A chain jump is a special case of alternative branch, with one or more branches containing no steps.

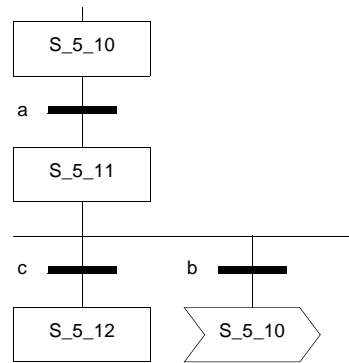
A process of S_5_10 via S_5_11 and S_5_12 after S_5_13 only occurs, if S_5_10 is active and the condition for transition a is true. A process of S_5_10 directly after S_5_13 only occurs, if S_5_10 is active and the condition for transition b is true and a is false.



Chain loop

A chain loop is a special case of alternative branch, with which one or more branches lead back to a previous step.

A process of S_5_11 via S_5_10 only occurs if the condition for transition c is false and b is true.



Alternative Branch

Introduction The alternative branch offers the possibility to program branches conditionally in the control flow of the SFC structure.

Structure With alternative branches, as many transitions follow a step under the horizontal line as there are different sequences. Only one of these transitions can ever be switched. The branch to be solved is determined by the result of the transition conditions of the transitions, which come after the alternative branch.

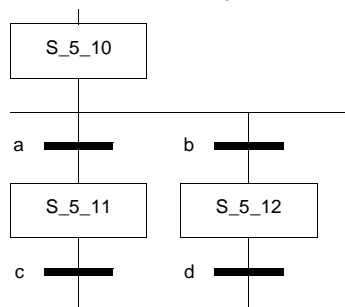
Processing Sequence Branch transitions are processed from left to right. If a transition condition is satisfied, the remaining transitions are no longer processed. The branch with the satisfied transition is activated. This gives rise to a left to right priority for branches.

If none of the transitions is switched, the currently set step remains set.

Processing Sequence processing:

If...	Then...
If S_5_10 is active and the transition condition a is true.	Then a sequence from S_5_10 to S_5_11 occurs.
If S_5_10 is active and the transition condition b is true and a is false.	Then a sequence from S_5_10 to S_5_12 occurs.

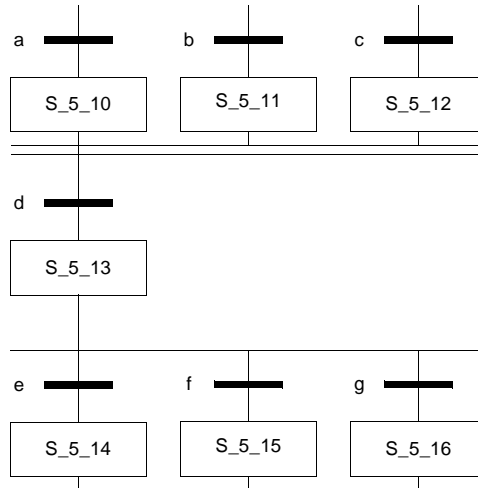
Sequence processing:



Alternative Branch after Parallel Joint

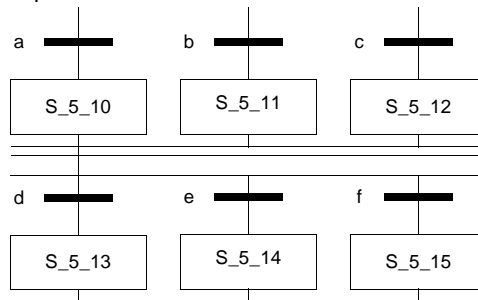
According to IEC 1131-3, alternative branches may not directly follow parallel joints. The joint and the branch must be separated by a transition step sequence.

Example:



If you want to insert an alternative branch directly after a parallel joint, you can use the **Options** → **Preferences** → **Graphic Editors** → **Allow Alternative Branches after Parallel Joints** to do so.

Example:



Joint

All alternative branches must be rejoined to a single branch through Alternative Joints (See *Alternative connection*, p. 250) or Jumps (See *Jump*, p. 246).

Alternative connection

At a Glance

In the alternative connection, the various branches of an alternative branch are again connected to one branch in which additional processing can be performed. This connection can also be performed with a jump.

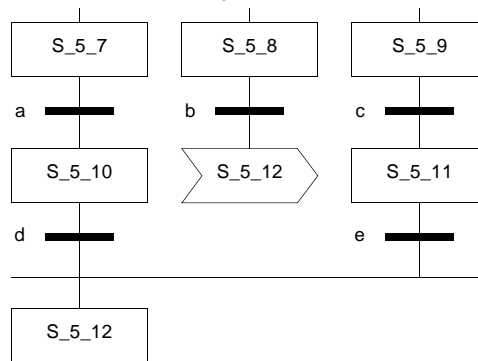
Processing

Sequence processing:

If...	Then...
If S_5_10 is active and the transition condition d is true.	Then a process of S_5_10 to S_5_12 takes place.
If S_5_8 is active and the transition condition b is true, and therefore a jump to S_5_12 is performed.	Then a process of S_5_8 to S_5_12 takes place.
If S_5_11 is active and the transition condition e is true.	Then a process of S_5_11 to S_5_12 takes place.

Note: Only a single one of these branches is active, corresponding to the transition condition in the alternative branch.

Sequence processing:



Parallel branch

At a Glance

With parallel branches, the edit is split into two or more strings, which will be processed in parallel. Only a joint transition immediately through the horizontal double synchronization lines is possible.

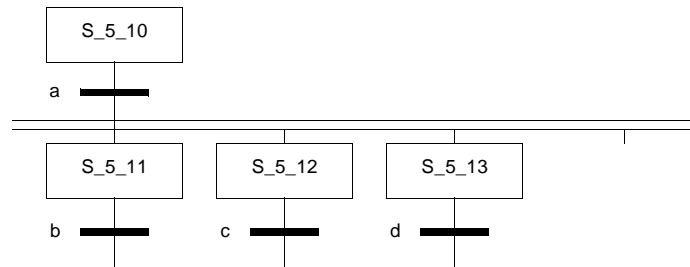
Processing

Processing a sequence:

If...	Then...
If S_5_10 is active and the transition condition a, which shares the same transition, is likewise true.	Then a process of S_5_10 to S_5_11, S_5_12,... takes place.

Note: After the simultaneous activation of S_5_11, S_5_12 etc., the sequences run independent of each other.

Processing a sequence:



Definition of initial steps

If a step is to become an initial step within a parallel branch, a step must be defined as the initial step in each branch of the parallel branch.

Parallel connection

At a Glance

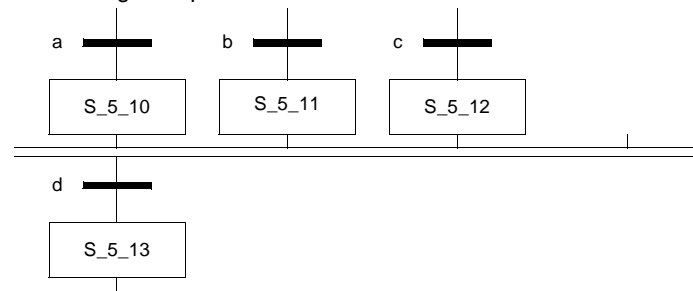
The parallel connection reconnects two or more parallel branches to a branch. The transition to a parallel connection is evaluated when all previous steps of the transition are set. Only a joint transition immediately through the double horizontal synchronisation lines is possible.

Processing

Processing a sequence:

If...	Then...
If S_5_10, S_5_11 etc. are active at the same time and the transition condition d, sharing a joint transition, is true.	Then a process of S_5_10, S_5_11, ...to S_5_13 takes place.

Processing a sequence:



Text object

At a Glance

Text can be positioned in the form of text objects using SFC sequence language. The size of these text objects depends on the length of the text. This text object is at least the size of a cell and can be vertically and horizontally enlarged to other cells according to the size of the text. Text objects can only be placed in free cells.

Memory space

Text objects occupy no memory space on the PLC because the text is not downloaded onto the PLC.

9.3 Working with the SFC Sequence Language

Introduction

Overview

This section describes working with the SFC sequence language.

What's in this Section?

This section contains the following topics:

Topic	Page
General information on editing objects	254
Declaring step properties	257
Declaring actions	259
Identifier	262
Declaring a Transition	264
Alias Designations for Steps and Transitions	266

General information on editing objects

At a Glance	<p>In the SFC editor the background consists of a logical grid. SFC objects can theoretically be placed in every unoccupied cell. If a link with another object is established (explicitly or by vertically placing objects in neighboring cells), this link will be tested. If this link is not permitted, a report of this is given and the object is not inserted.</p> <p>Steps, transitions and jumps each require a cell. Parallel branches, parallel connections, alternative branches and alternative connections do not require a separate cell each, but are inserted into the corresponding cell of the step or transition.</p>
Maximum number of elements	<p>To prevent step strings being subdivided, 99 linked steps with the transitions are vertically shown along with a locking jump with its transition. To limit the complexity and to enable the animation to be performed, the number of objects (Steps + Transitions + Branches + Connections) in one section is limited to 2000.</p>
Inserting Objects	<p>The SFC object (Step, Transition etc.) can be inserted individually via the menu command in the main menu Objects or in the form of a group (Step transition string, structured parallel string etc.) of the required size.</p> <p>After selection of the object, a position in the step string can be selected, in which the object should be inserted. If the position selected is already occupied, space is made before insertion into the step string, if desired, and then the object placed in it. If the object is placed on a connection, it is separated, the object is inserted and a link to the newly placed object is generated.</p>
Shifting objects	<p>If the object is shifted onto a connection, it is separated, the object is inserted and a link to the newly placed object is generated.</p>
Copying steps	<p>By copying and inserting it is possible to copy steps through projects. Since the definition of actions displays a reference to a variable, which is defined by the Variable Editor for the particular project, copying between projects can result in this reference no longer being valid. In this instance, the action is deleted, the action list is updated and an error message is displayed.</p>
Deleting steps	<p>Steps can only be deleted after an action has been saved if the action(s) were unconnected before the step was performed.</p>

Selecting an object

The procedure for selecting an object is as follows:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	Position the cursor on the object to be selected and left-click. Reaction: The selected object is displayed in a blue border.

Selecting several objects (by pressing Shift)

The procedure for selecting several objects (by pressing Shift) is as follows:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	Position the cursor on the object to be selected first and left-click.
3	Press and hold the Shift key, select additional objects and left-click. Reaction: The selected objects are displayed in a blue border.

Selecting several objects (by using the rubber band function)

The procedure for selecting several objects (by using the rubber band function) is as follows:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	Press and hold the left mouse button, and pull a border over the objects to be selected. Reaction: On releasing the mouse key, all objects touching the border will be selected. The selected objects are displayed in a blue border.

Selecting all objects in a column/line

The procedure for selecting all objects in a column/line is as follows:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	In the column ruler/line ruler, click on the column number/line number whose objects are to be selected. Note: To select several columns/lines, press and hold the Shift key. Reaction: The selected objects are displayed in a blue border.

Inserting additional columns

The procedure for inserting additional columns within an existing step string is as follows:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	In the column ruler, click on the column number in front of which the insertion is to be performed. Note: In order to insert several columns, press the Shift key to select several columns and insert a corresponding number of empty spaces.
3	Use the menu command Edit → Insert . Reaction: From the selected column, the entire step string is moved one column to the right. The links (branches) will remain intact.

Inserting additional lines

The procedure for inserting additional lines within an existing step string is as follows:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	In the line ruler, click on the line number in front of which the insertion is to take place. Note: Should the insertion of several lines be required, several lines are selected and a corresponding number of empty spaces are inserted by pressing the Shift key.
3	Use the menu command Edit → Insert . Reaction: From the selected line, the entire step string is moved one line downwards. The links (branches) therefore remain even.

Declaring step properties

At a Glance

The step properties are declared in the properties dialog of the step.
Declaring step properties:

Step properties

Step name Initial step

Action

Time
 Variable Literal

Action
 Variable Direct address

Mon. times and delay time
 'SCFSTEP_TIMES' variable Literals **Delay**

Declaring step properties

The following description contains an example of declaring the step properties:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	Double-click on a step. Reaction: The dialog Step properties of the step is opened.
3	A name can be manually defined for the step, or the proposed name can remain. If a name is to be assigned, please note that the step name (max. 32 characters) must be unique throughout the entire project. If the step name entered already exists, a warning is given and another name must be chosen. The section name must correspond to the IEC name conventions, otherwise an error message is displayed. Note: In accordance with IEC1131-3, only letters are permitted as the first character of step names. Should numbers be required as the first character, however, the menu command Options → Preferences → IEC Extensions... → Allow leading digits in identifiers . Instead of the free names an alias designation can also be selected, see also <i>Alias Designations for Steps and Transitions, p. 266</i> This is then shown in SFC and FBD sections and with search functions, application documentation and analysis.
4	Next, define whether or not it concerns the initial step of the sequence of the step. A step must be defined as an initial step for each sequence.
5	If desired, the Mon. time and delay time can be defined for the step. The time values can be entered in the properties dialog either directly as time duration literal (this can be automatically transmitted in the Learn Mon. time mode, see also <i>Learn monitoring times, p. 274</i>) or as multi-element variable of SFCSTEP_TIMES data type, see also ' <i>SFCSTEP_TIMES</i> ' Variable, p. 239. Hence: Delay time < minimum Mon. time < maximum Mon. time
6	Using the command button Comment... click on the dialog box Enter with comment , in which a comment on the step may be entered. This comment is shown in the status bar of the editor window, if the step is selected.

Declaring actions

At a Glance

The actions are declared in the properties dialog of the step.
Declaring actions:

Step properties

Step name: Initial step

Action

Time: Variable Literal

Cdet:

Action: Variable Direct address

Mon. times and delay time

'SCFSTEP_TIMES' variable Literals

Declaring actions

The following description contains an example of declaring the actions:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	Double-click on a step. Reaction: The dialog Step properties of the step is opened.
3	From the Cdet list, select an Identifier (See <i>Identifier</i> , p. 262) for the Action. In this way, the behavior of the action is determined (e.g. saving, non saving, delayed etc.). Note: With the identifiers L, D, and DS, in the text box Time duration: an additional time duration of TIME data type must be defined.
4	Next define the type of action (variable or direct address) in the zone Type: with the option buttons.
5	<ul style="list-style-type: none"> ● If the Variable has been selected, it is possible with the button Var. declaration... to open the Variable Editor and define a new output variable there. Also with the command button Look up... a list of all the variables can be shown and one selected through Select. ● If the Direct address has been selected, in the text box Direct address: the output address must be entered.
6	After all the definitions for the actions have been met, confirm this with the command button New Note: Confirmation with the Enter key is not possible in this case and leads to an error message

Altering an action

The procedure for altering an action declaration is as follows:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	Double-click on a step. Reaction: The dialog Step properties of the step is opened.
3	To alter an action declaration, select an action in the list. Reaction: All definitions (identifiers, time duration, variable or address and type) of the action are transferred into the corresponding text boxes and lists.
4	If these definitions are altered, as described in the <i>Declaring actions, p. 260</i> section.
5	<ul style="list-style-type: none"> ● Should it be necessary to assign these new definitions as a new action in the step, use the command button New. Reaction: The action is additionally recorded in the list of actions. ● Should it be necessary to overwrite the current action with the new action, use the command button Accept Reaction: The old action is overwritten.

Deleting an action declaration

The procedure for deleting an action declaration is as follows:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	Double-click on a step. Reaction: The dialog Step properties of the step is opened.
3	To delete an action declaration, select an action in the list. Reaction: All definitions (identifiers, time duration, variable or address and type) of the action are transferred into the corresponding text boxes and lists.
4	Use the command button Delete . Reaction: The selected action is deleted.

Identifier

At a Glance

For every connection of an action to a step, an identifier must be defined for the action. The identifier must define the control of the action. The identifier can be introduced as the input of an internal Function Block for the configured link of the step with the action. If the step is active, the input of this internal Function Block is set to 1. The Function Block is then processed according to its type. If all conditions are satisfied, the output Q (action) is set to 1.

The following identifiers are usable in Concept:

- N / none (See *Identifier N / none*, p. 262)
- S (See *Identifier S*, p. 262)
- R (See *Identifier R*, p. 263)
- L (See *Identifier L*, p. 263)
- D (See *Identifier D*, p. 264)
- P (See *Identifier P*, p. 264)
- DS (See *Identifier DS*, p. 264)

For the identifiers L, D and DS, a time duration of the data type TIME must additionally be defined.

Identifier N / none

The identifiers N and none have the same meaning and stand for "Not saved" and/or "No identifier".

Identifier S

The identifier S stands for "set (saved)".

The set action also remains active, when the associated step is inactive. The action first becomes inactive, when reset is used with the Identifier R (See *Identifier R*, p. 263) in another step.

<p>Note: The identifier is automatically declared as unbuffered. This means that the value is reset to "0" after stop and cold restart, e.g. when voltage is on/off. Should a buffered output be required, please use the RS or SR Function Block from the IEC block library.</p>
--

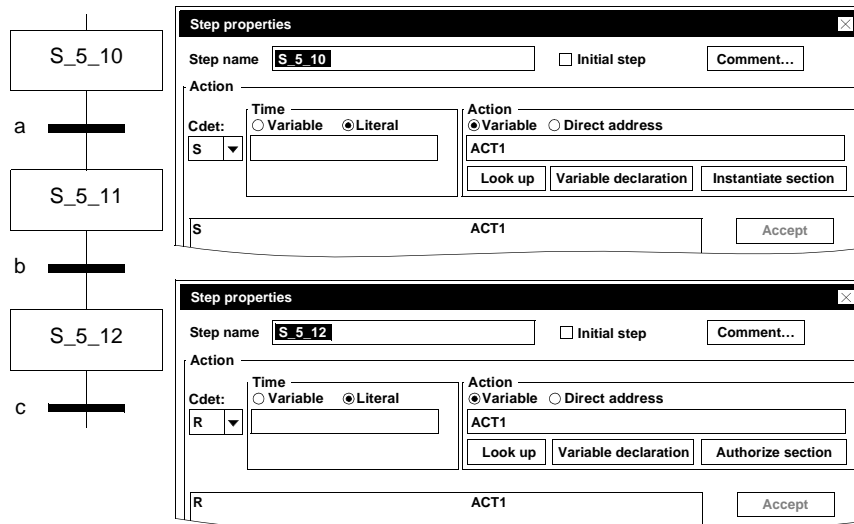
Identifier R

The identifier R stands for "overriding reset".

The action, which is set in another step with the Identifier S (See *Identifier S*, p. 262), is reset. The activation of any action can also be prevented.

Note: The identifier is automatically declared as unbuffered. This means that the value is reset to "0" after stop and cold restart, e.g. when voltage is on/off. Should a buffered output be required, please use the RS or SR Function Block from the IEC block library.

In the step S_5_10 the action ACT1 becomes and remains active, until the reset in step S_5_12.

**Identifier L**

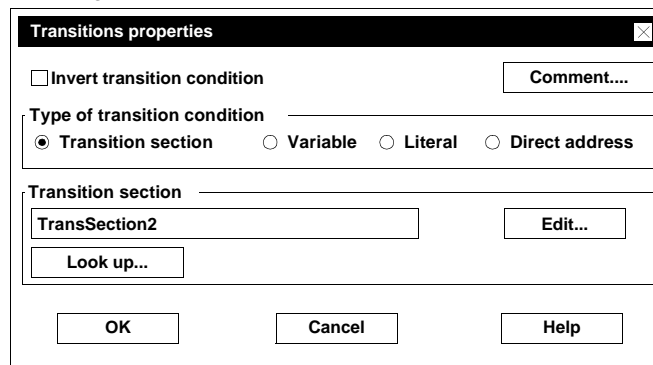
The identifier L stands for "Limited".

If the step is active, the action is also active. After the process of the time duration, defined manually for the action, the action returns to 0, even if the step is still active. The action also becomes 0 if the step is inactive.

- Identifier D** The identifier D stands for "delayed".
- If the step is active, the internal timer is started and the action becomes 1 after the process of the time duration, which was defined manually for the action. If the step becomes inactive after that, the action becomes inactive as well. If the step becomes inactive before the process of the internal time, the action does not become active.
-
- Identifier P** The identifier P stands for "Pulse".
- If the step becomes active, the action becomes 1 and this remains for one program cycle, independent of whether or not the step remains active.
-
- Identifier DS** The identifier DS stands for "delayed and saved". It is a combination of the identifiers D (See *Identifier D*, p. 264) and S (See *Identifier S*, p. 262).
- If the step becomes active, the internal timer is started and the action becomes active after the process of the manually defined time duration. The action first becomes inactive once again, when reset is used with the IdentifierR (See *Identifier R*, p. 263) in another step. If the step becomes inactive before the process of the internal time, the action does not become active.
-

Declaring a Transition

Introduction Transitions are declared in the properties dialog of the transition.
 Declaring a transition:



Declaring a transition:

The following example describes the procedure when declaring a transition:

Step	Action
1	With Objects → Selection mode go to selection mode.
2	Double-click on a transition. Response: The dialog Transition properties of the transition is opened.
3	Begin by determining Kind of transition condition: determine the type (Transition section, Variable, Literal, Direct address) of transition condition.
4	<ul style="list-style-type: none"> ● After selecting the Transition section has been selected, enter in the text box Transition section the name of the transition section to be created. This is a section containing the logic of the transition condition and it is automatically linked with the transition. To process this section, press the command button Process.... ● After selecting the Variable has been selected, enter in the text box BOOL variable the name of the selected unlocated variable, located variable or constants. Note: For an example for invocation of multi-element variables see <i>Calling Derived Data Types, p. 524.</i> ● If the Literal has been selected, select in the field Value the value of the literal. ● If the Dir. address , enter in the text box Direct address the required address.
5	The transition condition can now be inverted with the Invert trans. cond. check box. Response: An inverted transition condition is displayed with a (-) symbol in front of the name of the variable on the transition.
6	With the command button Comment click on the dialog box Enter with comment , in which a comment about the transition can be entered. This comment is shown in the status bar of the editor window, if the transition is selected.
7	After all the definitions for the transition have been met, confirm this with the command button OK .

Copying transition conditions

By copying and inserting it is possible to copy transitions through projects. Since the definition of a transition displays a reference to a variable, which is defined by the Variable Editor for the particular project, copying between projects can result in this reference no longer being valid. In this instance, the transition condition is deleted and an error message appears.

Alias Designations for Steps and Transitions

Introduction

Instead of free names you can also select alias designations for steps and transitions. These are then displayed in SFC and FBD sections during search functions, application documentation and analysis.

Import and export functions do not recognize the alias designations, since they are dynamically generated. The visualization can retrieve the alias designations dynamically, however they cannot be used for the configuration of fixed references, since they can change constantly.

The languages ST, IL and LD do not support alias designations and display the free names.

Name Definition

The alias designations are dynamically generated during editing procedures, and the same applies when the **Dynamic Numbered** option is switched on.

Alias designations remain empty until numbering can take place i.e. when all objects are linked to one chain.

The alias designations are made up of the position of the steps and transitions in the section and the section name.

The length of the section name part displayed in the alias designation is freely definable in the **Options** → **Preferences** → **Graphical Editors Preferences** dialog. You can define how many characters from the section name (beginning with the first character) should go into the alias designations here.

Note: The settings in this dialog are used in the project description (PRJ.DSK) and in the Concept installations description (CONCEPT.DSK), i.e. they are valid for the entire Concept installation.
If a project is opened, which was created using alternative settings (e.g. Settings from **Presentation of Steps and Transitions** numbered **IEC_like** in the project and **Dynamic numbered** in the current Concept installation), errors can occur when opening projects.

Alias Designations for Steps


With steps, the lines and columns occupied by steps are each numbered beginning with the top left. A four-figure step number is made from the column and line numbers (ccll). The alias designation for steps is made from S_ string, part of the section name (nnn), a further underscore (_) and the step number (ccll) (S_nn_ccll).


Alias Designations for Transitions

The alias designations for transitions are derived from the alias designation of the preceding step cell, even when this is empty. The alias designation for transitions is made from the T_ string, part of the section name (nnn), a further underscore (_) and the number of the preceding step cell (ccl) (T_nn_ccl).

Activating the Alias Designations

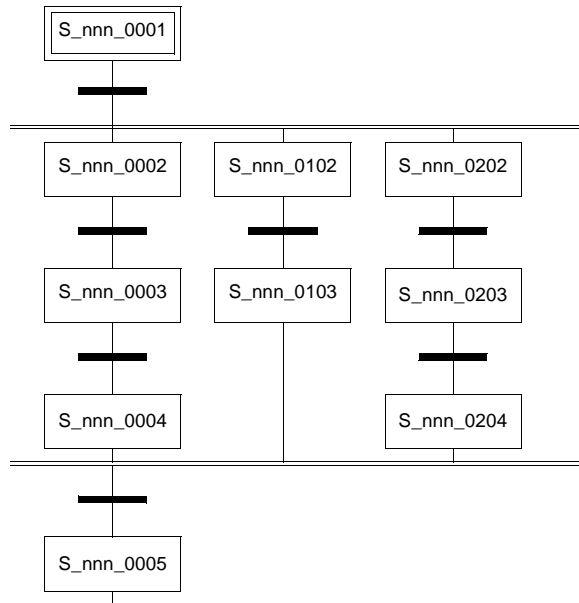
The free name is entered as the default for steps and transitions. If you require alias designations, you can activate them in the **Options** → **Preferences** → **Graphical Editors Preferences** dialog using the **Dynamic Numbered** option.

	CAUTION
	<p>Danger of loss of data.</p> <p>The free names (IEC_like) are overwritten by the alias names when this option is selected. If you want to restore the free names, close the project without saving.</p> <p>Failure to follow this precaution can result in injury or equipment damage.</p>

	CAUTION
	<p>Danger of loss of data.</p> <p>You must not switch between the IEC_like and Dynamic Numbered display modes if an FBD transition section is already open. Otherwise, this could result in section and variable names containing spaces. Therefore, close all FBD transition sections before you change the representation mode.</p> <p>Failure to follow this precaution can result in injury or equipment damage.</p>

**Example for
Alias
Designations**

Example for alias designations:



**Inserting and
Deleting Objects**

When inserting and deleting objects (steps and transitions) the alias designations are renumbered.

9.4 Online functions of the SFC sequence language

At a Glance

Overview This section describes the online functions of the SFC sequence language elements.

What's in this Section? This section contains the following topics:

Topic	Page
Animation	270
Controlling a Step String	272
Learn monitoring times	274
Transition diagnosis	277

Animation

Introduction

In the animation mode the following are displayed in different colors in the editor window:

- the active steps,
- the time the steps are or were active for,
- time out errors of the steps and
- the status of the transitions (made, not made).

<p>Note: If the transition, and therefore the transition section, is not processed, the status DISABLED appears in the animated transition section.</p>

Activating the Animation

The animation is activated with the menu command **Online** → **Animation**.

Color key

There are 12 different color schemes available for animation. An overview of the color scheme and the meaning of each color can be found in Online help (Tip: Search the online help for the index reference "Colors").

Changing Values

In this mode the following can be changed:

- With transitions:
 - the transition condition, if this is a literal.
- With steps:
 - the maximum supervision time,
 - the minimum supervision time,
 - the delay time and
 - the times of the actions.

These changes are sent online to the PLC.

Transition Animation

Normally, only the currently evaluated transitions are animated and their status (transition condition satisfied/not satisfied) is displayed.

It is also possible to display the status of the transitions not currently being processed. This will only show the status of the transitions. It has no influence on the behavior of the sequence. To do this you require the XSFCNTRL function block of the SYSTEM block library. Additionally, in the **Options** → **Preferences** → **Graphical Editors** dialog, you must check the **Animate All Conditions of the Transition Section** check box.

Note: This function leads to a considerable burden on the logic scan. This results from the fact that all the transitions in the affected section are solved and animated in one logic scan, whereas this is normally solved sequentially depending on the process status (preceding step active/inactive).

Displaying all Transition Conditions

The procedure for showing all transition conditions is as follows:

Step	Action
1	Create an FBD section and enter the XSFCNTRL function block of the SYSTEM block library.
2	Enter the names of the SFC section to be animated as the instance name (block name) of the XSFCNTRL function block.
3	Assign the value "1" to the ALLTRANS input of the XSFCNTRL function block (using a literal or, depending on the process, a variable). Response: By doing this, the calculation of all transition conditions is activated. Otherwise an old status of the transition condition would be displayed.
4	With the menu command Project → Execution Order... (or the project browser) ensure that the FBD section is executed before the SFC section to be animated
5	Check the Animate All Conditions of the Transition Section check box in the Options → Preferences → Graphical Editors dialog.
6	Download the program to the PLC and start the animation of the SFC section. Response: All transition conditions are then displayed.

Controlling a Step String

Introduction

There are 3 ways of controlling a string:

- with the animation control
- with the menu commands in the main menu **Online**
- with the SFCCNTRL or XSFCCNTRL function block (SYSTEM block library)

If controlling a string through the different options simultaneously, these control operations have equal priority.

The control operations triggered using the menu commands in the **Online** main menu and using the animation panel can be locked by the function blocks SFCCNTRL and XSFCCNTRL.

A control operation in one of the methods is also displayed in the other two methods.

Requirements

It is only possible to control the step string when the animation mode for the section is active.


Animation Panel




The animation panel is activated with the menu command **Online** → **Show Animation Panel**.

The animation panel contains all the possibilities that are also available as menu commands.

Mode of Functioning

You can test the processing of an SFC section with the animation panel and the menu commands. For example, steps can be relayed, the processing of the string can be controlled (whether or not transitions and/or actions are to be processed), time errors can be reset or the string can be reset to initialization status.

	<p>WARNING</p>
	<p>Danger of unsafe, dangerous and destructive tool operations.</p> <p>Set/Reset flag, Disable Transitions, Disable Actions, Step Unconditional, Step/Trans. dependant and Force Selected Steps should not be used for debugging on controllers of machine tools, processes or material maintenance systems when they are running. This can lead to unsafe, dangerous and destructive operation of tools or processes linked to the controller.</p> <p>Failure to follow this precaution can result in death, serious injury, or equipment damage.</p>


Set/Reset Flag	<p>The Set/Reset flag resets the string and starts it as standard.</p> <ul style="list-style-type: none"> ● Reset chain To reset the string, activate Set/Reset Flag. This stops the chain and all actions are reset. No operator interventions are possible. ● Starting the chain in a standardized way For a standardized start of the string, Set/Reset Flag must first be activated and then deactivated. With the 1 → 0 slope the chain is reset i.e. the initial step is activated. 		
Disable Time Check	<p>If Disable Time Check is activated, there is no longer any time supervision of the steps. The step delay time, however, still remains active.</p>		
Disable Transitions	<p>If Disable Transitions is activated, the transition conditions are no longer utilized. The string remains in its current state, independent of the signals on the transitions. The string can still only be used via the control commands (Set/Reset Flag, Step Unconditional, Step/Trans. Dependant).</p>		
Disable Actions	<p>If Disable Actions is activated, the step actions are no longer processed.</p>		
Step Unconditional	<p>The next step is activated independently of the transition status, but not until the step delay time of the active step has elapsed.</p> <p>With Step Unconditional, all branches are activated in parallel branches, and the left branch is always activated in alternate branches.</p> <p>Step/Trans. dependent is used for activating process-dependent branches.</p>		
<table border="1"> <tr> <td data-bbox="459 1294 571 1512" style="text-align: center; vertical-align: middle;">  </td> <td data-bbox="571 1294 1343 1512"> <p>WARNING</p> <p>Danger of unsafe, dangerous and destructive tool operations.</p> <p>Step Unconditional activates the next step, even if the transition is not satisfied.</p> <p>Failure to follow this precaution can result in death, serious injury, or equipment damage.</p> </td> </tr> </table>			<p>WARNING</p> <p>Danger of unsafe, dangerous and destructive tool operations.</p> <p>Step Unconditional activates the next step, even if the transition is not satisfied.</p> <p>Failure to follow this precaution can result in death, serious injury, or equipment damage.</p>
	<p>WARNING</p> <p>Danger of unsafe, dangerous and destructive tool operations.</p> <p>Step Unconditional activates the next step, even if the transition is not satisfied.</p> <p>Failure to follow this precaution can result in death, serious injury, or equipment damage.</p>		
Step/Trans. Dependent	<p>The next step is activated when the transition conditions are satisfied.</p> <p>Step/Trans. Dependent is advisable only when Disable Transitions is active. By freezing the transitions (Disable Transitions) it is possible, with Step/Trans. Dependent to process the string elements manually step by step. In this way the transitions commute depending on the transition condition.</p>		

Reset Time Error If **Reset Time Error** is activated, the error message display for time supervision in the SFC section is reset.

Force Selected Steps The selected step(s) are activated independent of the status of the transitions and steps.

In alternative branches, only one single step and one single branch can be activated.

In parallel branches, steps can only be set, if the process is already located in the parallel branch and one step in every branch is active. If one step is set in a parallel branch, all other parallel branches remain unaffected by it.

	WARNING
	Danger of unsafe, dangerous and destructive tool operations.
	Force Selected Steps activates the selected steps, even if the transition is not satisfied. Failure to follow this precaution can result in death, serious injury, or equipment damage.

This functionality is not available via the function blocks SFCCNTRL or XSFCNTRL (SYSTEM block library).

Select Active Steps The active step of the step string is searched for and selected.

Learn monitoring times

At a Glance In this mode, the minimum and maximum times, for which the steps were active, are determined. After mode deactivation, the determined times for the single steps are shown in the **Learn step monitoring times** dialog box. From there, the minimum (See *Minimum Supervision Time*, p. 239) and maximum monitoring time (See *Maximum Supervision Time*, p. 238) are accepted in the step properties. During the transfer, a factor can be specified for the minimum and maximum time.

Note: This functionality is not available via the Function Blocks SFCCNTRL or XSFCNTRL (Block library SYSTEM).

Note on determining values

Please ensure that at least 2 cycles typical for the process were gathered.

The determined values are first saved after the single step becomes inactive, i.e. if a step was never active during the "Learn monitoring times" mode, no value is determined for this step.

The storage of all determined step cells of a cycle can take some time. Because of this, very long step sequence times and very short individual step durations may be indeterminable, due to internal time overlaps.

Use of 'SFCSTEP_TIME S' variable or constants

Should the step have been assigned a 'SFCSTEP_TIMES' variable or constant in the **Step properties** dialog, the times learned for these variables/constants are shown as the initial value. Should these initial values be used for a long period of time, do not allow corresponding elements (min., max.) of these variables/constants to be written.

After learning the monitoring times, the altered initial values must be loaded into the PLC.

- This is performed for variables with the menu command **Online** → **Download**.
 - This is performed for constants with the menu command **Online** → **Download changes**.
-

Calculating "learned" times

A factor can be defined for the determined values, which are multiplied when calculating the monitoring times.

- Minimum monitoring time = minimum determined time x Minimum [%]
 - Maximum monitoring time = maximum determined time x Maximum [%]
-

Calculating "learned" times " Example 1

Calculating "learned" times

- The determined times for one step are: 1 s, 2 s, 2 s
- Minimum [%]: 50
- Maximum [%]: 200

Following the above formula, this results in a minimum monitoring time of 500 ms and a maximum monitoring time of 4 s.

**Calculating
"learned" times "
Example 2**

If a delay time is given for the step, this is considered when calculating the minimum monitoring time. I.e. if the delay time is larger than the calculated value for the minimum monitoring time, the calculated value for the minimum monitoring time is ignored and set to 0 ms (i.e. there is no monitoring of the minimum time).

Calculating "learned" times

- The determined times for one step are: 1 s, 2 s, 2 s
- Delay time: 2 s
- Minimum [%]: 50
- Maximum [%]: 200

This results in a minimum monitoring time of 0 ms and a maximum monitoring time of 4 s.

**Calculating
"learned" times "
Example 3**

If a delay time is given for the step, this is likewise considered when calculating the maximum monitoring time. I.e. if the delay time is larger than the calculated value for the maximum monitoring time, the calculated value for the maximum monitoring time is ignored and in its place, a suitable value is calculated.

In such a case 2 cases are considered:

- A value for the minimum monitoring time is available.
Then the value for the maximum monitoring time is calculated according to the following formula: Minimum monitoring time + 20 ms

Example:

- The determined times for one step are: 2 s, 2 s, 2 s
- Delay time: 3 s
- Minimum [%]: 200
- Maximum [%]: 100

Following the above formula, this results in a minimum monitoring time of 4 s and a maximum monitoring time of 4s20ms.

- No value for the minimum monitoring time is available, see *example 2*.
Then the value for the maximum monitoring time is calculated according to the following formula: Delay time + 20 ms

Example:

- The determined times for one step are: 1 s, 2 s, 2 s
- Delay time: 1 s
- Minimum [%]: 50
- Maximum [%]: 100

Following the above formula, this results in a minimum monitoring time of 0 s and a maximum monitoring time of 1s20ms.

Transition diagnosis

Preview

The transition diagnosis monitors that the immediately preceding step was active following the transition, commutated within a certain time in the step sequence (with parallel branches in the step sequences). Should this not be the case, the associated transition network (with alternative branches, the transition network of all associated transitions) is analysed, and the error, including the analysed signal, is entered in the signal buffer. This can now be evaluated using visualization software (e.g. MonitorPro, Factory Link).

Note: The transition diagnosis only runs when the string is active.

Transition diagnosis vs. Reaction diagnosis

The performance of the transition diagnosis is about equal to that of the reaction diagnosis (see *Function Block REA_DIA from the block library DIAGNO*). Contrary to the reaction diagnosis the re-registration of all the actions started and possible additional conditions are monitored here.

Activating the transition diagnosis

Activating the transition diagnosis:

Step	Action
1	Activate the transition diagnosis by entering a Mon. time in the field Maximum step properties of the immediately preceding step (see also <i>Learn monitoring times, p. 274</i>). If the field remains empty or the time 0 is entered the transition monitoring is inactive.
2	Aktivate in the dialog Project → Code generation options... → Code generation options... the option Include diagnosis information to make memory available in the PLC for the error buffer.
3	Load the altered configuration into the PLC.

Instruction list IL

10

At a Glance

Overview

This Chapter describes the programming language instruction list IL which conforms to IEC 1131.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
10.1	General information about the IL instruction list	281
10.2	Instructions	283
10.3	IL instruction list operators	295
10.4	Call up of functions, Function Blocks (EFBs) and Derived Function Blocks (DFBs)	320
10.5	Syntax check and Code generation	329
10.6	Online functions of the IL instruction list	333
10.7	Creating a program with the IL instruction list	338

10.1 General information about the IL instruction list

General Information about the IL Instruction List

Introduction	With assistance from the programming language (IL) instruction list e.g. Function Blocks and functions can be called up conditionally or unconditionally, assignments can be performed, and jumps can be performed conditionally or unconditionally within a section.
Spell Check	Spelling is immediately checked when key words, separators and comments are entered. If a key word, separator or comment is recognized, it is identified with a color surround. If unauthorized key words (instructions or operators) are entered, it is likewise identified in color.
IEC Conventions	<p>The IEC 1131 does not permit the input of direct addresses in the usual Concept form. To input direct addresses see <i>Operands</i>, p. 285.</p> <p>In accordance with IEC 113-3, key words must be entered in upper case. Should the use of lower case letters be required, they can be enabled in the dialog box Options → Preferences → IEC Extensions... → IEC Extensions with the option Allow case insensitive keywords.</p> <p>Blank spaces and tabs have no influence upon the syntax and can be used freely.</p>
Context help	With the right mouse button an object can be selected and at the same time a context sensitive menu called up. Therefore, for example, with FFBs the right mouse button can call up the associated block description.
Syntax Check	A syntax check can be performed during the program/DFB creation with Project → Analyze section , see also <i>Syntax Check</i> , p. 330.
Codegeneration	Using the Project → Code Generation Options menu command, you can define options for code generation, see also <i>Code generation</i> , p. 332.
Editing with the Keyboard	Normally editing in Concept is performed with the mouse, however it is also possible with the keyboard (see also <i>Short Cut Keys in the IL, ST and Data Type Editor</i> , p. 765).
IEC Conformity	For a description of the IEC conformity of the IL programming language see <i>IEC conformity</i> , p. 779.

10.2 Instructions

At a Glance

Overview This section contains an overview of the instructions for the programming language instruction list.(IL)

What's in this Section? This section contains the following topics:

Topic	Page
General information about instructions	284
Operands	285
Modifier	287
Operators	288
Tag	291
Declaration (VAR...END_VAR)	292
Comment	294

General information about instructions

At a Glance

An instruction list is composed of a series of instructions.

Each instruction begins on a new line and consists of:

- an Operator (See *Operands*, p. 285),
- if necessary with modifier (See *Modifier*, p. 287) and
- if necessary one or more operands (See *IL instruction list operators*, p. 295).

Should several operands be used, they are separated by commas. It is possible for a mark (See *Tag*, p. 291) to be in front of the instruction, which is followed by a colon. A comment (See *Comment*, p. 294) can follow the instruction.

Example:

Tag		Operators		Operands
	START:	LD	A	(* Keyboard 1 *)
		ANDN	B	(* AND keyboard 2 *)
		ST	C	(* Ventilator on *)
			Modifier	Comments

Structure of the programming language

IL is a so-called battery orientated language, i.e. each instruction uses or alters the current content of the battery (a form of internal cache). The IEC 1131 refers to this battery as the "result".

For this reason, an instruction list should always begin with the LD operand ("Load in battery command").

Example of an addition:

Command	Meaning
LD 10	The value "10" is loaded into the battery.
ADD 25	"25" is added to the battery content.
ST A	The result is stored in the "A" variable. The content of the "A" variable and the battery is now "35". A possible ensuing instruction would be worked with the battery content "35", should it not begin with LD.

Comparative operations likewise always refer to the battery. The Boolean result of the comparison is stored in the battery and is therefore the current battery content.

Example of a comparison:

Command	Meaning
LD B	The value "B" is loaded into the battery.
GT 10	"10" is compared with the battery content.
ST A	The result of the comparison is stored in the "A" variable. If "B" is less than or equal to "10", the value of both the "A" variable, and the battery content is "0" (FALSE). If "B" is greater than or equal to "10", the value of both the "A" variable, and the battery content is "1" (TRUE).

Operands

At a Glance

An operand can be:

- a literal,
- a variable,
- a multi-element variable,
- an element of a multi-element variable,
- a FB/DFB output or
- a direct address.

Access to the field variables

When accessing the field variable (ARRAY), only literals and variables of ANY_INT type are permitted in the index entry.

Example: Saving a field variable

```
LD var1[i]
ST var2.otto[4]
```

Type conversion

The operand and the current accu content must be of the same type. Should operands of various types be processed, a type conversion must be performed beforehand.

An exception is the data type TIME in conjunction with the arithmetic operators MUL and DIV. With both these operators, an operand of TIME data type can be processed together with an operand of ANY_NUM data type. The result of this instruction has in this instance the data type TIME.

Example: Integer variable and real variable

In the example the integer variable "i1" is converted into a real variable, before being added to the real variable "r4".

```
LD i1
INT_TO_REAL
ADD r4
ST r3
```

Example: Integer variable and time variable

In the example the time variable "t2" is multiplied by the integer variable "i4" and the result is stored in the time variable "t1".

```
LD t2
MUL i4
ST t1
```

Default data types of direct addresses

The following table shows the default data types of direct addresses:

Input	Output	Default data type	possible data type
%IX,%I	%QX,%Q	BOOL	BOOL
%IB	%QB	BYTE	BYTE
%IW	%QW	INT	INT, UINT, WORD
%ID	%QD	REAL	REAL, DINT, UDINT, TIME

Using other data types

Should other data types be assigned as default data types of a direct address, this must be done through an explicit declaration (VAR...END_VAR (See *Declaration (VAR...END_VAR)*, p. 292)). VAR...END_VAR cannot be used in Concept for the declaration of variables. The variable declaration conveniently follows the Variable Editor (See *Variables editor*, p. 479).

Modifier

At a Glance	Modifiers influence the implementation of the preceding operators (see <i>Operators</i> , p. 288).
Modifier N	The Modifier N is used to invert the value of the operands bit by bit. The modifier can only be used on operands with the ANY_BIT data type.
Example: N	In the example C will be "1", when A is "1" and B is "0". <pre>LD A ANDN B ST C</pre>
Modifier C	The modifier C is used to carry out the associated instruction, should the value of the battery be "1" (TRUE). The modifier can only be used on operands with the BOOL data type.
Example: C	In the example the jump after START is only performed, when A is "1" (TRUE) and B is "1" (TRUE). <pre>LD AAND BJMPC START</pre>
Modifier CN	If the modifiers C and N are combined, the associated instruction is only performed, should the value of the battery be a Boolean "0" (FALSE).
Example: CN	In the example, the jump after START is only performed, when A is "0" (FALSE) and/or B is "0" (FALSE). <pre>LD A AND B JMPCN START</pre>
Left bracket modifier "("	The left bracket modifier "(" is used to move back the evaluation of the operand, until the right bracket operator appears. The number of right bracket operations must be equal to the number of left bracket modifiers. Brackets can be nested.

Example: Left bracket "("

In the example E will be "1", if C and/or D is "1", just as A and B are "1".

```
LD A
AND B
AND( C
OR D
)
ST E
```

The example can also be programmed in the following manner:

```
LD A
AND B
AND(
LD C
OR D
)
ST E
```

Operators

At a Glance

An operator is a symbol for:

- an arithmetic operation to be executed,
- a configured operation to be executed or
- the function call up.

Operators are generic, i.e. they are automatically suited to the operands data type.

<p>Note: Operators can be either entered by hand or generated with assistance from the menu Objects.</p>
--

Table of operators

IL programming language operators:

Operator	Operator key	possible modifier	possible operand	see also
LD	Loads the operands value into the battery	N	Literal, variable, direct address of ANY data type	<i>Load (LD and LDN), p. 296</i>
ST	Saves the battery value in the operand	N	Variable, direct address of ANY data type	<i>Store (ST and STN), p. 297</i>
S	Sets the operand to 1, when the battery content is 1		Variable, direct address of BOOL data type	<i>Set (S), p. 298</i>
R	Sets the operand to 0, when the battery content is 1		Variable, direct address of BOOL data type	<i>Reset (R), p. 299</i>
AND	Configured AND	N, N(, (Literal, variable, direct address of ANY_BIT data type	<i>Boolean AND (AND, AND ()), ANDN, ANDN ()), p. 300</i>
OR	Configured OR	N, N(, (Literal, variable, direct address of ANY_BIT data type	<i>Boolean OR (OR, OR ()), ORN, ORN ()), p. 302</i>
XOR	Configured exclusive OR	N, N(, (Literal, variable, direct address of ANY_BIT data type	<i>Boolean exclusive OR (XOR, XOR ()), XORN, XORN ()), p. 304</i>
ADD	Addition	(Literal, variable, direct address of ANY_NUM data type or TIME data type	<i>Addition (ADD and ADD ()), p. 306</i>
SUB	Subtraction	(Literal, variable, direct address of ANY_NUM data type or TIME data type	<i>Subtraction (SUB and SUB ()), p. 307</i>
MUL	Multiplication	(Literal, variable, direct address of ANY_NUM data type or TIME data type	<i>Multiplication (*), p. 352</i>

Operator	Operator key	possible modifier	possible operand	see also
DIV	Division	(Literal, variable, direct address of ANY_NUM data type or TIME data type	<i>Division (DIV and DIV ()), p. 309</i>
GT	Comparison: >	(Literal, variable, direct address of ANY_ELEM data type	<i>Compare on "Greater Than" (GT and GT ()), p. 310</i>
GE	Comparison: >=	(Literal, variable, direct address of ANY_ELEM data type	<i>Compare to "Greater than/Equal to" (GE and GE ()), p. 311</i>
EQ	Comparison: =	(Literal, variable, direct address of ANY_ELEM data type	<i>Compare to "Equal to" (EQ and EQ ()), p. 312</i>
NE	Comparison: <>	(Literal, variable, direct address of ANY_ELEM data type	<i>Compare to "Not Equal to" (NE and NE ()), p. 313</i>
LE	Comparison: <=	(Literal, variable, direct address of ANY_ELEM data type	<i>Compare to "Less than/Equal to" (LE and LE ()), p. 314</i>
LT	Comparison: <	(Literal, variable, direct address of ANY_ELEM data type	<i>Compare to "Less Than" (LT and LT ()), p. 315</i>
JMP	Jump to tag	C, CN	TAG	<i>Jump to label (JMP, JMPC and JMPCN), p. 316</i>
CAL	Calling up a Function Block or DFB	C, CN	FBNAME (item name)	<i>Call Function Block/DFB (CAL, CALC and CALCN), p. 319</i>

Operator	Operator key	possible modifier	possible operand	see also
FUNCNAME	Performing a function		Literal, variable, direct address (data type is dependent on function)	<i>Function call, p. 327</i>
)	Editing on-hold operations			<i>Right parenthesis ")", p. 319</i>

Tag

At a Glance

Tags serve as destinations for Jumps (See *Jump to label (JMP, JMPC and JMPCN)*, p. 316).

Properties

Tag properties:

- Tags must always be the first element in a line.
- Tags must be unique throughout the project/DFB, and are not case-sensitive.
- Tags can be 32 characters long (max.).
- Tags must conform to the IEC name conventions.
- Tags are separated by a colon ":" from the following instruction.
- Tags are only permitted at the beginning of "Expressions", otherwise an undefined value can be found in the battery.

Destinations

Possible destinations are:

- the first LD instruction of a FB/DFB call up with assignment of input parameters (see *start2*),
- a normal LD instruction (see *start1*),
- a CAL instruction, which does not work with assignment of input parameters (see *start3*),
- a JMP instruction (see *start4*),
- the end of an instruction list (see *start5*).

Example

```
start2: LD A
        ST counter.CU
        LD B
        ST counter.R
        LD C
        ST counter.PV
        CAL counter
        JMPCN start4
start1: LD A
        AND B
        OR C
        ST D
        JMPC start3
        LD A
        ADD E
        JMP start5
start3: CAL counter (
        CU:=A
        R:=B
        PV:=C )
        JMP start1
start4: JMPC start1
start5:
```

Declaration (VAR...END_VAR)

At a Glance

The VAR instruction is used to declare the function blocks and DFBS used, and direct addresses if they are not to be used with the default data type. VAR cannot be used for declaring a variable in Concept. Declaring the variables may conveniently be done via the Variables editor. The END_VAR instruction marks the end of the declaration.

<p>Note: The declaration of the FBs/DFBs and direct addresses applies only to the current section. If the same FFB type or the same address are also used in another section, the FFB type or the address must be declared again in this section.</p>
--

Declaration of function blocks and DFBs

During declaration for each FB/DFB example, a unique example name is assigned. The example name is used to mark the function block uniquely in a project. The example name must be unique in the whole project; no distinction is made between upper/lower case. The example name must correspond to the IEC Name conventions, otherwise an error message will be displayed.

After specifying the example name, the function block type, e.g. CTU_DINT is specified.

In the case of function block types no data type is specified. It is determined by the data type of the actual parameters. If all actual parameters consist of literals, a suitable data type will be selected.

Any number of example names may be declared for an FB/DFB.

Note: The dialog **Objects** → **Insert FFB** provides you with a form for creating the FB-/DFB declaration in a simple and speedy manner.

Note: In contrast to graphic programming languages (FBD, LD), it is possible to call up multiple calls in FB/DFB examples within IL.

Example

Declaration of function blocks and DFBs

```

VAR
  RAMP_UP, RAMP_DOWN, RAMP_X : TON ;
  COUNT : CTU_DINT ;
  CLOCK : SYSCLOCK ;
  Pulse : TON ;
END_VAR

```

Declaration of direct addresses

In the case of this declaration, every direct address used whose data type does not correspond to the default data type will be assigned the required data type (see also Default data types of direct addresses (See *Default data types of direct addresses*, p. 286)).

Example

Declaration of direct addresses

```

VAR
  AT %QW1 : WORD ;
  AT %IW15 : UINT ;
  AT %ID45 : DINT ;
  AT %QD4 : TIME ;
END_VAR

```

Comment

Description

Within the IL Editor, comments always start with the string (*) and end in the string (*). Any comments may be entered between these two strings. Comments are shown in colors.

Note: In accordance with IEC 1131-1, comments are only permissible at the end of a line. However, if you wish to place these elsewhere, you can do this by using **Options** → **Preferences** → **IEC Extensions** → **Allow comments anywhere in text (IL)**.

Note: In accordance with IEC 1131-1, nested comments are not permissible. However, if you wish to place these elsewhere, you can do this by using **Options** → **Preferences** → **IEC Extensions** → **Allow nested comments**.

10.3 IL instruction list operators

At a Glance

Overview

This section describes the IL instruction list operators.

What's in this Section?

This section contains the following topics:

Topic	Page
Load (LD and LDN)	296
Store (ST and STN)	297
Set (S)	298
Reset (R)	299
Boolean AND (AND, AND (), ANDN, ANDN ())	300
Boolean OR (OR, OR (), ORN, ORN ())	302
Boolean exclusive OR (XOR, XOR (), XORN, XORN ())	304
Invert (NOT)	305
Addition (ADD and ADD ())	306
Subtraction (SUB and SUB ())	307
Multiplication (MUL and MUL())	308
Division (DIV and DIV ())	309
Compare on "Greater Than" (GT and GT ())	310
Compare to "Greater than/Equal to" (GE and GE ())	311
Compare to "Equal to"(EQ and EQ ())	312
Compare to "Not Equal to" (NE and NE ())	313
Compare to "Less than/Equal to" (LE and LE ())	314
Compare to "Less Than"(LT and LT ())	315
Jump to label (JMP, JMPC and JMPCN)	316
Call Function Block/DFB (CAL, CALC and CALCN)	319
FUNCNAME	319
Right parenthesis ")"	319

Load (LD and LDN)

LD Description With LD the value of the Operands is downloaded into the accumulator. The data width of the accumulator adapts itself automatically to the data type of the operand. This also applies to derived datatypes.

Example LD Example LD

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
ADD B	The accu contents are added to the value of "B".
ST E	The result is saved in "E".

LDN Description The downloaded operand can be negated with the Modifier N (only if the Operand is of data type ANY_BIT).

LDN Example LDN Example

Operation	Description
LDN A	The value of "A" is inverted and downloaded onto the accu.
ADD B	The accu contents are added to the value of "B".
ST E	The result is saved in "E".

Store (ST and STN)

ST Description With ST the current value of the accu is saved in the operand. The data type of the operand must therefore agree with the "data type" of the accu. Depending on whether an LD follows after ST or not, calculation proceeds with the "old" result.

ST Example ST Example

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
ADD B	The accu contents are added to the value of "B".
ST E	The result is saved in "E".
ADD B	Afterwards the value of "E" (current accu contents) is added to the value of "B" again
ST F	The result is saved in "F".
LD X	The value of "X" is now downloaded onto the accu.
SUB 3	3 is subtracted from the accu contents.
ST Y	The result is saved in "Y".

STN Description The operand to be saved can be negated with the N modifier (only if the operand is on the ANY_BIT data type).

STN Example ST Example

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
ADD B	The accu contents are added to the value of "B".
STN E	The result is inverted and saved in "E".

Set (S)

Description S sets the operand to "1" when the current content of the accu is a Boolean "1".

Example S Example S

Command	Description
LD A	The value of "A" is loaded into the accu.
s OUT	If the content of the accu (the value of A) is "1", "OUT" is set to "1".

Use Usually this operator is used together with the Reset operator R (flip flop) as a pair.

Example RS flip flop The example shows an RS flip flop (Reset dominant).

Command	Description
LD A	The value of "A" is loaded into the accu.
s OUT	If the content of the accu (the value of "A") is "1", "OUT" is set to "1".
LD C	The value of "C" is loaded into the accu.
R OUT	If the content of the accu (the value of "C") is "1", "OUT" is set to "0".

Start behavior The start behavior of PLC's is divided into cold and warm starts.

- **Cold Start**
Following a cold start (loading the program with **Online** → **Download**) all variables are set (independently of their type) to "0" or, if available, to their initial value.
- **Warm Start**
On a warm start (stopping and starting of the program or **Online** → **Download changes**) different start behavior applies for located variables/direct addresses and unlocated variables:
 - **Located variables/direct addresses**
During a warm start the located variable/direct address, is set to "0", or to its initial value if present, via the set instruction.
 - **Unlocated variables**
On a warm start the unlocated variable, set via the set instruction, maintains its present value (storing behavior).

Note: Should a buffered located variable/direct address be required, please use the RS or SR function blocks from the block library IEC.

Reset (R)

Description R sets the operand to "0" when the current content of the accu is a Boolean "1".

Example R Example R

Command	Description
LD A	The value of "A" is loaded into the accu.
R OUT	If the content of the accu (the value of "A") is "1", "OUT" is set to "0".

Use Usually this operator is used together with the Set operator S (flip flop) as a pair.

Example SR flip flop The example shows an SR flip flop (Set dominant).

Command	Description
LD A	The value of "A" is loaded into the accu.
R OUT	If the content of the accu (the value of "A") is "1", "OUT" is set to "0".
LD C	The value of "C" is loaded into the accu.
S OUT	If the content of the accu (the value of "C") is "1", "OUT" is set to "1".

Start behavior PLC start behavior is divided into cold and warm starts:

- **Cold Start**
Following a cold start (loading the program with **Online** → **Download**) all variables are set (independently of their type) to "0" or, if available, to their initial value.
- **Warm Start**
On a warm start (stopping and starting of the program or **Online** → **Download changes**) different start behavior applies for located variables/direct addresses and unlocated variables:
 - **Located variables/direct addresses**
On a warm start the located variable/direct address, is set to "0", or to its initial value if present, via the reset instruction.
 - **Unlocated variables**
On a warm start the unlocated variable, set via the reset instruction, maintains its present value (storing behavior).

Note: Should a buffered located variable/direct address be required, please use the RS or SR function blocks from the block library IEC.

Boolean AND (AND, AND (), ANDN, ANDN ())

AND Description With AND a logical AND link occurs between the accu contents and the operand. For the data types BYTE and WORD the link is made by bit.

AND Example In the example D is "1", if A, B and C are "1".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
AND B	The accu contents are AND-linked with the contents of "B".
AND C	The accu contents (result of the AND link from "A" and "B") are AND-linked with the contents of "C".
ST D	The link result is saved in "D".

AND () Description AND can be used with the "(" left bracket modifier.

AND () Example In the example D is "1", if A is "1" and B or C are "1".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
AND (The AND link is deferred until the right bracket is reached.
LD B	The contents of "B" are downloaded onto the accu.
OR C	The contents of "C" are OR-linked with the accu contents.
)	The deferred AND link is solved. The accu contents (result of the OR link from "B" and "C") are AND-linked with the contents of "A".
ST D	The link result is saved in "D".

ANDN Description AND can be used with the N modifier.

ANDN Example In the example D is "1", if A is "1" and B and C are "0".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
ANDN B	The contents of "B" are inverted and AND-linked with the accu contents.
ANDN C	The contents of "C" are inverted and AND-linked with the accu contents (Result of the AND link from "A" and "B").
ST D	The link result is saved in "D".

**ANDN ()
Description**

AND can be used with the N modifier and the "(" left bracket modifier.

ANDN () Example In the example D is "1", if A is "1", B is "0" and C is "1".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
ANDN (The AND link is deferred until the right bracket is reached.
LD B	The contents of "B" are downloaded onto the accu.
ORN C	The contents of "C" are inverted and OR-linked with the accu contents.
)	The deferred AND link is solved. The contents of "A" are inverted and AND-linked with the accu contents (Result of the OR link from "B" and "C").
ST D	The link result is saved in "D".

Boolean OR (OR, OR (), ORN, ORN ())

OR Description With OR a logical OR link occurs between the accu contents and the operand. For the data types BYTE and WORD the link is made by bit.

OR Example In the example D is "1", if A or B is "1" and C is "1".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
OR B	The accu contents are OR-linked with the contents of "B".
AND C	The accu contents (result of the AND link from "A" and "B") are AND-linked.
ST D	The link result is saved in "D".

OR () Description OR can be used with the "(" left bracket modifier.

OR () Example In the example D is "1", if A is "1" or B and C are "1".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
OR (The OR link is deferred until the right bracket is reached.
LD B	The contents of "B" are downloaded onto the accu.
AND C	The contents of "C" are AND-linked with the accu contents.
)	The deferred OR link is solved. The accu contents (Result of the AND link from "B" and "C") are OR linked with the contents of "A".
ST D	The link result is saved in "D".

ORN Description ORN can be used with the N modifier.

ORN Example

In the example D is "1", if A is "1" or B is "0" and C is "1".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
ORN B	The contents of "B" are inverted and OR linked with the accu contents.
AND C	The contents of "C" are AND linked with the accu contents (result of the OR link from "A" and "B").
ST D	The link result is saved in "D".

**ORN ()
Description**

ORN can be used with the N modifier and the "(" left bracket modifier.

ORN () Example

In the example D is "1", if A is "1" or B or C are "0".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
ORN (The OR link is deferred until the right bracket is reached.
LD B	The contents of "B" are downloaded onto the accu.
AND C	The contents of "C" are AND-linked with the accu contents.
)	The deferred OR link is solved. The accu contents (result of the AND link from "B" and "C") are OR linked with the contents of "A".
ST D	The link result is saved in "D".

Boolean exclusive OR (XOR, XOR (), XORN, XORN ())

XOR description With XOR, a logical exclusive OR link is made between the accu contents and the operand.
 If more than two operands are linked the result is "1" for an odd number of 1 conditions and "0" for an even number of 1 conditions.
 For the data types BYTE and WORD the link is made by bit.

XOR example In the example, D is "1" if A or B is "1". If A and B have the same status (both "0" or both "1"), D is "0".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
XOR B	The accu contents are linked with the contents of the "B" exclusive OR.
ST D	The equation result is saved in "D".

XOR () description XOR can be used with the "(" left bracket modifier.

XOR () example In the example, D is "1" if A or the AND link from B and C is "1". If A and the result of the AND link have the same status (both "0" or both "1"), D is "0".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
XOR (The exclusive OR link is deferred until the right bracket is reached.
LD B	The contents of "B" are downloaded onto the accu.
AND C	The contents of "C" are AND-linked with the accu contents.
)	The reset exclusive OR link is performed. The accu contents (result of the AND link from "B" and "C") are exclusive OR-linked with the contents of "A".
ST D	The equation result is saved in "D".

XORN description XOR can be used with the N modifier.

XORN example In the example, D is "1" if A and B have the same contents (both "1" or both "0"). If A and B do not have the same status, D is "0".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
XORN B	The contents of "B" are inverted and exclusive OR-linked with the accu contents.
ST D	The equation result is saved in "D".

**XORN ()
description**

XORN can be used with the "(" left bracket and N modifiers.

XORN () example

In the example, D is "1" if A and the AND link from B and C have the same contents (both "1" or both "0"). If A and B and the AND link from B and C do not have the same status, D is "0".

Operation	Description
LD A	The contents of "A" are downloaded onto the accu.
XORN (The exclusive OR link is deferred until the right bracket is reached.
LD B	The contents of "B" are downloaded onto the accu.
AND C	The contents of "C" are AND-linked with the accu contents.
)	The reset exclusive OR link is performed. The accu contents (result of the AND link from "B" and "C") are exclusive OR-linked with the contents of "A".
ST D	The equation result is saved in "D".

Invert (NOT)

NOT Description

The accumulator content is inverted with NOT.
NOT can only be used with Boolean data types (BIT, BYTE, WORD).

Note: This operator does not conform to IEC 61131-1.

Example NOT

Example NOT

Operation	Description
LD A	The contents of "A" are downloaded onto the accumulator.
NOT	The accumulator content is inverted.
ST B	The result is saved in "B".

Addition (ADD and ADD ())

ADD Description With ADD the value of the operand is added to the accu contents.

ADD Example The example corresponds to the formula $D = A + B + C$

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
ADD B	The accu contents are added to the value of "B".
ADD C	The accu contents (sum of "A"+"B") are added to the value of "C".
ST D	The result is saved in "D".

ADD () Description ADD can be used with the "(" left bracket modifier.

ADD () Example The example corresponds to the formula $D = A + (B - C)$

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
ADD (The addition is deferred until the right bracket is reached.
LD B	The value of "B" is downloaded onto the accu.
SUB C	The value of "C" is subtracted from the accu contents.
)	The deferred addition is solved. The accu contents (result of "B"- "C") are added to the value of "A".
ST D	The result is saved in "D".

Subtraction (SUB and SUB ())

SUB Description With SUB the value of an operand is subtracted from the accu contents.

SUB Example The example corresponds to the formula $D = A - B - C$

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
SUB B	The value of "B" is subtracted from the accu contents.
SUB C	The value of "C" is subtracted from the accu contents (result of "A"- "B").
ST D	The result is saved in "D".

Description SUB (SUB can be used with the "(" left bracket modifier.

Example SUB (The example corresponds to the formula $D = A - (B - C)$

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
SUB (The subtraction is reset until the right bracket is reached.
LD B	The value of "B" is downloaded onto the accu.
SUB C	The value of "C" is subtracted from the accu contents.
)	The reset subtraction is performed. The accu contents (result of "B"- "C") are subtracted from the value of "A".
ST D	The result is saved in "D".

Multiplication (MUL and MUL())

MUL Description With MUL the accu contents are multiplied by the value of an operand.

MUL Example The example corresponds to the formula $D = A \times B \times C$

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
MUL B	The accu contents are multiplied by the value of "B".
MUL C	The accu contents (Result of "A"x"B") are multiplied by the value of "C".
ST D	The result is saved in "D".

Multiplication of TIME values Normally the operand and the current accu contents must be of the same data type. The TIME data type in relation to MUL is an exception. In this case the accu content of data type TIME can be used together with an operand of data type ANY_NUM. After the execution of this instruction list the accu contents have, in this case, the data type TIME.

Example MUL with TIME values The example corresponds to the formula $t1 = t2 \times i4$.

Operation	Description
LD t2	The value of the time variables "t2" are downloaded onto the accu.
MUL i4	The accu contents are multiplied by the value of the integer variable "i4".
ST t1	The result is saved in the time variable "t1".

Description MUL () MUL can be used with the "(" left bracket modifier.

Example MUL () The example corresponds to the formula $D = A \times (B - C)$

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
MUL (The multiplication is reset until the right bracket is reached.
LD B	The value of "B" is downloaded onto the accu.
SUB C	The value of "C" is subtracted from the accu contents.
)	The reset multiplication is performed. The accu contents (result of "B"- "C") are multiplied by the value of "A".
ST D	The result is saved in "D".

Division (DIV and DIV ())

DIV Description With DIV the accu contents are divided by the value of an operand.

DIV example The example corresponds to the formula $D = A / B / C$.

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
DIV B	The accu contents are divided by the value of "B".
DIV C	The accu contents (result of "A"/"B") are divided by the value of "C".
ST D	The result is saved in "D".

Division of TIME values Normally the operand and the current accu contents must be of the same data type. One exception is the data type TIME in connection with DIV. In this case the accu contents of data type TIME can be processed with an operand of data type ANY_NUM. After the execution of this instruction list the accu contents have, in this case, the data type TIME.

Example MUL with TIME values The example corresponds to the formula $t1 = t2 / i4$.

Operation	Description
LD t2	The value of the time variables "t2" is downloaded onto the accu.
DIV i4	The accu contents are divided by the value of the integer variable "i4".
ST t1	The result is saved in the time variable "t1".

DIV () Description DIV can be used with the "(" left bracket modifier.

DIV () Example The example corresponds to the formula $D = A / (B - C)$

Operation	Description
LD A	The value of "A" is downloaded onto the accu.
DIV (The division is reset until it the right bracket is reached.
LD B	The value of "B" is downloaded onto the accu.
SUB C	The value of "C" is subtracted from the accu contents.
)	The reset division is performed. The value of "A" is divided by the accu contents (result of "B"- "C").
ST D	The result is saved in "D".

Compare on "Greater Than" (GT and GT ())

Description GT With GT the accu contents is compared with the operand contents. If the accu contents is greater than the operand contents, the result is a boolean "1". If the accu contents is less than or equal to the operand contents, the result is a boolean "0".

Example GT Example GT

Command	Description
LD A	The value of "A" is loaded into the accu.
GT 10	The accu content is compared with the value "0".
ST D	If the value of "A" was less than "10" (or equal "10"), the value "0" is saved in "D". If the value of "A" was greater than "10", the value "1" is saved in "D".

Description GT () GT can be used with the modifier left bracket "(".

Example GT () Example GT ()

Command	Description
LD A	The value of "A" is loaded into the accu.
GT (The comparison is deferred until the right bracket has been reached.
LD B	The value of "B" is loaded into the accu.
SUB C	The value of "C" is subtracted from the accu content.
)	The deferred comparison is now carried out. The value of "A" is compared with the accu contents (result of "B"-C").
ST D	If the value of "A" was less than "B"-C" (or equal "B"-C"), the value "0" is saved in "D". If the value of "A" was greater than "B"-C", the value "1" is saved in "D".

Compare to "Greater than/Equal to" (GE and GE ())

Description GE With GE the accu contents is compared with the operand contents. If the accu contents is greater than or equal to the operand contents, the result is a Boolean "1". If the accu contents is less than the operand contents, the result is a Boolean "0".

GE example E.g. GE

Command	Description
LD A	The value of "A" is loaded into the accu.
GE 10	The accu content is compared with the value "10".
ST D	If the value of "A" was less than "10", the value "0" is saved in "D". If the value of "A" was equal to or greater than "10", the value "1" is saved in "D".

Description GT () GE can be used with the modifier left bracket "(".

GE () example GE () example

Command	Description
LD A	The value of "A" is loaded into the accu.
GE (The comparison is deferred until the right bracket has been reached.
LD B	The value of "B" is loaded into the accu.
SUB C	The value of "C" is subtracted from the accu content.
)	The deferred comparison is now carried out. The value of "A" is compared with the accu contents (result of "B"- "C").
ST D	If the value of "A" was less than "B"- "C", the value "0" is saved in "D". If the value of "A" was equal to or greater than "B" - "C", the value "1" is saved in "D".

Compare to "Equal to"(EQ and EQ ())

EQ description With EQ the accu contents are compared with the operand contents. If the accu contents are equal to the operand contents, the result is a Boolean "1". If the accu contents are not equal to the operand contents, the result is a Boolean "0".

EQ example EQ example

Command	Description
LD A	The value of "A" is loaded into the accu.
EQ 10	The accu contents are compared with the value "10".
ST D	If the value of "A" was not equal to "10", the value "0" is saved in "D". If the value of "A" was equal to "10", the value "1" is saved in "D".

Description EQ () EQ can be used with the modifier left bracket "(".

EQ () example EQ () example

Command	Description
LD A	The value of "A" is loaded into the accu.
EQ (The comparison is deferred until the right bracket has been reached.
LD B	The value of "B" is loaded into the accu.
SUB C	The value of "C" is subtracted from the accu content.
)	The deferred comparison is now carried out. The value of "A" is compared with the accu contents (result of "B"-C").
ST D	If the value of "A" was not equal to "B"-C", the value "0" is saved in "D". If the value of "A" was equal to "B"-C", the value "1" is saved in "D".

Compare to "Not Equal to" (NE and NE ())

NE description With NE the accu contents are compared with the operand contents. If the accu contents are not equal to the operand contents, the result is a Boolean "1". If the accu contents are equal to the operand contents, the result is a Boolean "0".

NE example NE example

Command	Description
LD A	The value of "A" is loaded into the accu.
NE 10	The accu contents are compared with the value "10".
ST D	If the value of "A" was equal to "10", the value "0" is saved in "D". If the value of "A" was not equal to "10", the value "1" is saved in "D".

Description NE () NE can be used with the modifier left bracket "(".

NE () example NE () example

Command	Description
LD A	The value of "A" is loaded into the accu.
NE (The comparison is deferred until the right bracket has been reached..
LD B	The value of "B" is loaded into the accu.
SUB C	The value of "C" is subtracted from the accu content.
)	The deferred comparison is now carried out. The value of "A" is compared with the accu contents (result of "B"- "C").
ST D	If the value of "A" was equal to "B"- "C", the value "0" is saved in "D". If the value of "A" was not equal to "B"- "C", the value "1" is saved in "D".

Compare to "Less than/Equal to" (LE and LE ())

Description With LE the accu contents are compared with the operand contents. If the accu contents are less than or equal to the operand contents, the result is a Boolean "1". If the accu contents are greater than the operand contents, the result is a Boolean "0".

LE example LE example

Command	Description
LD A	The value of "A" is loaded into the accu.
LE 10	The accu contents are compared with the value "10".
ST D	If the value of "A" was greater than "10", the value "0" is saved in "D". If the value of "A" was less than or equal to "10", the value "1" is saved in "D".

Description LE () LE can be used with the modifier left bracket "(".

LE () example LE () example

Command	Description
LD A	The value of "A" is loaded into the accu.
LE (The comparison is deferred until the right bracket has been reached.
LD B	The value of "B" is loaded into the accu.
SUB C	The value of "C" is subtracted from the accu content.
)	The deferred comparison is now carried out. The value of "A" is compared with the accu contents (result of "B"- "C").
ST D	If the value of "A" was greater than "B"- "C", the value "0" is saved in "D". If the value of "A" was less than "B"- "C" (or equal to "B"- "C"), the value "1" is saved in "D".

Compare to "Less Than"(LT and LT ())

LT description With LT the accu contents are compared with the operand contents. If the accu contents are less than the operand contents, the result is a Boolean "1". If the accu contents are greater than or equal to the operand contents, the result is a Boolean "0".

LT example LT example

Command	Description
LD A	The value of "A" is loaded into the accu.
LT 10	The accu contents are compared with the value "10".
ST D	If the value of "A" was greater than "10" (or equal to "10"), the value "0" is saved in "D". If the value of "A" was less than "10", the value "1" is saved in "D".

Description LT () LT can be used with the modifier left bracket "(".

LT () example LT () example

Command	Description
LD A	The value of "A" is loaded into the accu.
LT (The comparison is deferred until the right bracket has been reached.
LD B	The value of "B" is loaded into the accu.
SUB C	The value of "C" is subtracted from the accu content.
)	The deferred comparison is now carried out. The value of "A" is compared with the accu contents (result of "B"- "C").
ST D	If the value of "A" was greater than "B"- "C" (or equal to "B"- "C"), the value "0" is saved in "D". If the value of "A" was less than "B"- "C", the value "1" is saved in "D".

Jump to label (JMP, JMPC and JMPCN)

JMP Description With JMP a conditional or unconditional jump to a label is solved.

The label is used as an address and identifies the destination instruction. The destination instruction can be above or below the jump instruction. A label must always be the first element of a line. The label (max. 32 characters) must not be duplicated anywhere else in the project and there is no case sensitivity. The labels are separated by a colon ":" from the following instruction. Labels should only be at the beginning of "expressions", since otherwise an undefined value can be in the accu.

JMP Example In the example an unconditional jump to the label "start" is solved.

Operation	Description
start: LD A	The value of "A" is downloaded onto the accu.
AND B	Logical AND link between the accu contents and the contents of "B".
OR C	Logical OR link between the accu contents and the contents of "C".
ST D	The result of the links is saved in "D".
JMP start	Independent of the accu contents (value of "D") a jump to label "start" is solved.

**JMPC and
JMPCN
Description**

JMP can be used with the modifiers C and CN (only if the operand is of data type ANY_BIT).

JMPC Example In the example a conditional jump (with "1") to label "start" is solved.

Operation	Description
start: LD A	The value of "A" is downloaded onto the accu.
AND B	Logical AND link between the accu contents and the contents of "B".
OR C	Logical OR link between the accu contents and the contents of "C".
ST D	The result of the links is saved in "D".
JMPC start	The jump is only solved if the accu contents (value of "D") has the value "1".

JMPCN Example In the example a conditional jump (by "0") to label "start" is solved.

Operation	Description
start: LD A	The value of "A" is downloaded onto the accu.
AND B	Logical AND link between the accu contents and the contents of "B".
OR C	Logical OR link between the accu contents and the contents of "C".
ST D	The result of the links is saved in "D".
JMPCN start	The jump is only solved if the accu contents (value of "D") has the value "0".

Addresses

Possible addresses are:

- each LD instruction (see start1)
 - each CAL instruction (see start2)
 - the end of an instruction list (see start3)
- Jumps cannot be made into other sections.

Example for possible addresses:

Operation	Description
VAR Timer_1 : TON; END_VAR	Declaration of the function blocks TON.
LD IN1_BOOL	
ST OT1_BOOL	
JMPC start1	Jump to start1, if OT1_BOOL = 1
LD IN1_BOOL	
AND IN2_BOOL	
JMPCN start2	Jump to start2, if OT1_BOOL = 0
ST OT2_BOOL	
start1: LD IN1_INT	
ADD IN2_INT	
ST OT1_INT	
JMP start3	Unconditional jump after start3, JMPC/JMPCN is not allowed here as the accu contents are not of type BOOL.
start2: CAL Timer_1 (IN:=IN3_BOOL, PT:=t#6s)	
LD Timer_1.ET	
ST OT1_TIME	
LD Timer_1.Q	
ST OT3_BOOL	
start3	

Call Function Block/DFB (CAL, CALC and CALCN)

CAL Description With CAL a function block or a DFB is conditionally or unconditionally called.

CALC and CALCN Description CALC can be used with the Modifiers C and CN (only if the operand is of data type ANY_BIT).

Use of Function Blocks and DFBs *Use of Function Blocks and DFBs, p. 321*

FUNCNAME

Description A function is performed with the function name (see *Function call, p. 327*).

Right parenthesis ")"

At a Glance With the right parenthesis ")" the editing of the reset operator is started. The number of right parenthesis operations must be equal to the number of left bracket modifiers. Brackets can be nested.

Example In the example E will be "1", if C and/or D is "1", just as A and B are "1".

```
LD A
AND B
AND( C
OR D
)
ST E
```

10.4 Call up of functions, Function Blocks (EFBs) and Derived Function Blocks (DFBs)

At a Glance

Overview This section describes the call up of functions, Function Blocks (EFBs) and Derived Function Blocks (DFBs).

What's in this Section? This section contains the following topics:

Topic	Page
Use of Function Blocks and DFBs	321
Invoking a Function Block/DFB	323
Function call	327

Use of Function Blocks and DFBs

Use of Function Blocks and DFBs

Function blocks are provided by Concept in the form of libraries. The function block logic is created in C++ programming language and cannot be altered in the IL Editor. The names of the available function blocks can be taken from the block libraries. DFBs are function blocks, which have been defined in Concept-DFB. There is no difference between functions and function blocks for DFBs. They are always handled as function blocks regardless of their internal structure.

The use of function blocks and DFBs consists of three parts in IL:

- the declaration (See *Declaration*, p. 322),
- the function block/DFB invocation (See *Invoking a Function Block/DFB*, p. 323),
- the use of the function block/DFB outputs (See *Use of the Function Block/DFB Outputs*, p. 322).

Note: The declaration of the function block/DFB invocation can take place manually or you can create the block end and the assignment of the parameters using the menu command **Objects** → **Insert FFB**.

Function Blocks with Limited Use

Use of the following EFBs from the DIAGNO block library is limited in IL (the function blocks can be used, but the expanded diagnostic information cannot be evaluated):

- XACT, XACT_DIA,
- XDYN_DIA,
- XGRP_DIA,
- XLOCK,
- XPRE_DIA,
- XLOCK_DIA,
- XREA_DIA

Function Blocks with Limited Invocation

For EFBs which have one or more outputs with data type ANY but no inputs with data type ANY (generic outputs/inputs), the block invocation can only take place in compact form (See *CAL with a list of the input/output parameters (compact form)*, p. 325). e.g. in the block library **LIB984**:

- GET_3X
 - GET_4X
-

**Unusable
Function Blocks**

Unusable Function Blocks:

- EFBs which use several registers with only the entry for the first register on the input/output (e.g. MBP_MSTR from the COMM block library) cannot be used.
- EFBs, which contain outputs with input information (e.g. GET_BIT, R2T from the LIB984 block library) cannot be used
- The following EFBs from the **COMM** block library cannot be used for the technical reasons listed above:
 - CREADREG
 - CREAD_REG
 - CWITREG
 - CWRITE_REG
 - READREG
 - READ_REG
 - WRITEREG
 - WRITE_REG
 - MBP_MSTR
- The following EFBs from the **LIB984** block library cannot be used for the technical reasons listed above:
 - FIFO
 - GET_BIT
 - IEC_BMDI
 - LIFO
 - R2T
 - SET_BIT
 - SRCH
 - T2T

Declaration

Before invoking the function block/DFBs, they must be declared using VAR and END_VAR (See *Declaration (VAR...END_VAR)*, p. 292).

**Function Block/
DFB Invocation**

Invoking a Function Block/DFB, p. 323

**Use of the
Function Block/
DFB Outputs**

The outputs of the function block/DFBs can always be used when a variable (read only) can also be used.

Instance name

```
LD COUNT.Q
ST %QX1
```

Formal parameter

Invoking a Function Block/DFB

At a Glance

The invocation can be made in 4 forms:

- using CAL with a list of the input parameters (See *CAL with a list of the input parameters*, p. 323),
- using CAL with a list of the input/output parameters (compact form) (See *CAL with a list of the input/output parameters (compact form)*, p. 325),
- using CAL and Load/Save the input parameters (See *CAL with Loading/Saving of Input Parameters*, p. 325),
- when using the input operators (See *Using the Input Operators*, p. 326).

Note: Even if the function block has no inputs or the input parameters are not to be defined, the function block should be invoked (`CAL EFB_XY ()`) before the outputs can be used. Otherwise the initial values for the outputs are given, i.e. "0".

Note: In IL, unlike the graphic programming languages (FBD, LD), FB/DFB instances can be invoked multiple times.

CAL with a list of the input parameters

Function blocks/DFBs can be invoked using an instruction consisting of the CAL instruction followed by the instance names for the FBs/DFBs and a list, in brackets, of value assignments (current parameters) to formal parameters. The order of the formal parameters in a function block invocation is not significant. The list of the current parameters can be broken straight after a comma. It is not necessary for all formal parameters to be assigned a value. If a formal parameter is not assigned a value, the initial value defined in the variable editor is used when executing the function block. If an initial value is not defined, the default value (0) is used.

Note: Inputs of type VARINOUT (See also *Use of the DFB in IL*, p. 428) always have to be assigned a value.

Using the CAL (..) instruction, setting the parameters for the function blocks/DFBs is ended. Then no more values can be sent to the FB/DFB. Only the output values can be read.

Example CAL with a list of the input parameters

```

VAR
  CLOCK : SYSCLOCK ;
  COUNT : CTU_DINT ;
END_VAR

```

Instance name

```

CAL CLOCK ( )
CAL COUNT (CU:=CLOCK.CLK3, R:=%IX10, PV:=100)
:
LD COUNT.Q
ST out

```

Formal parameter

Current parameter

or

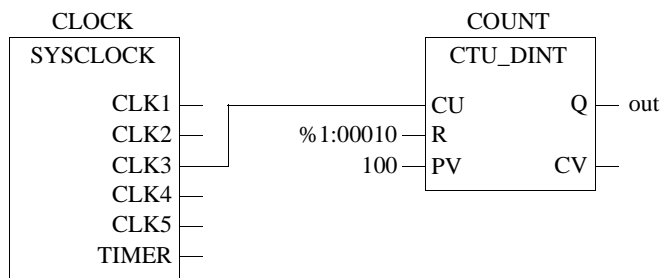
```

VAR
  CLOCK : SYSCLOCK ;
  COUNT : CTU_DINT ;
  Pulse : TP ;
END_VAR

CAL CLOCK ( )
CAL COUNT(
  CU:=CLOCK.CLK3,
  R:=reset,
  PV:=100)
:
LD COUNT.Q
ST out

```

Invocation of the function block in FBD.



CAL with a list of the input/output parameters (compact form)

Block invocation and the assignments for the inputs/outputs are also possible in a more compact form, which saves runtime:

```
VAR
    CLOCK : SYSCLOCK ;
    COUNT : CTU_DINT ;
END_VAR

CAL CLOCK ( ) ;
CAL COUNT (CU:=CLOCK.CLK3, R:=%IX10, PV:=100, Q=>out)
```

CAL with Loading/Saving of Input Parameters

Function blocks/DFBs can be invoked using an instruction list, which consists of loading the current parameters, followed by saving them in the formal parameters, followed by the CAL instruction. The order in which the parameters are loaded and saved is not significant. The list of the current parameters can be broken directly after a comma. It is not necessary for all formal parameters to be assigned a value. If a formal parameter is not assigned a value, the initial value defined in the variable editor is used when executing the function block. If an initial value is not defined, the default value (0) is used.

Note: Inputs of type VARINOUT (See also *Use of the DFB in IL, p. 428*) always have to be assigned a value.

Using the CAL FBNAME instruction, setting the parameters for the function blocks/DFBs is ended. Then no more values can be sent to the FB/DFB. Only the output values can be read.

Only load and save instructions for the current FB/DFBs to be configured are allowed to be between the first load instruction for the current parameters and invocation of the function block/DFBs. All other instructions are not allowed in this position.

Example

CAL with Loading/Saving of Input Parameters

```
CAL CLOCK
LD CLOCK.CLK3
ST COUNT.CU
LD %IX10
ST COUNT.R
LD 100
ST COUNT.PV
CAL COUNT
:
:
LD COUNT.Q
:
```

Using the Input Operators

Function blocks can be called using an instruction list which consists of loading the current parameters, followed by saving them in the formal parameters, followed by an input operator. The order in which the parameters are loaded and saved is not significant. The list of the current parameters can be broken directly after a comma. It is not necessary for all formal parameters to be assigned a value. If a formal parameter is not assigned a value, the initial value defined in the variable editor is used when executing the function block. If an initial value is not defined, the default value (0) is used.

Note: Inputs of type VARINOUT (See also *Use of the DFB in IL, p. 428*) always have to be assigned a value.

The possible input operators for the different function blocks can be taken from the table. Other input operators are not available.

Input operator	FB Type
S1, R	SR
S, R1	RS
CLK	R_TRIG
CLK	F_TRIG
CU, R, PV	CTU_INT, CTU_DINT, CTU_UINT, CTU_UDINT
CD, LD, PV	CTD_INT, CTD_DINT, CTD_UINT, CTD_UDINT
CU, CD, R, LD, PV	CTUD_INT, CTUD_DINT, CTUD_UINT, CTUD_UDINT
IN, PT	TP
IN, PT	TON
IN, PT	TOF

Using input operator invocation, setting the parameters for the function blocks is ended. Then no more values can be sent to the FB. Only the output values can be read.

Only load and save instructions for the current FB being configured are allowed to be between the first load instruction for the current parameters and the input operator for the function block. All other instructions are not allowed in this position.

Example

Using the Input Operators

```

CAL CLOCK
LD CLOCK.CLK3
ST COUNT.CU
LD %IX10
ST COUNT.R
LD 100
PV COUNT

```

Current parameter
 Formal parameter
 Instance name
 Input operator

Function call**Use of functions**

Functions are provided by Concept in the form of libraries. The logic of the functions is created in the programming language C++ and cannot be changed in the IL editor. You will find the names of the available functions in the block libraries.

Functions are called using an instruction list consisting of loading the **first** actual parameter into the battery and the name of the function. If necessary, this will be followed by a list of further actual parameters. The sequence in which the formal parameters in a function call are enumerated is not significant. Immediately following a comma, the list of the actual parameters may be wrapped. The result of the function becomes the battery content after the function has been executed, and can be saved using ST (See *Store (ST and STN)*, p. 297) in an operand, or may directly be processed further.

Note: The declaration of function calls may either be generated manually, or you may generate the block rump and the allocation of the parameters using the menu command **Objects** → **Insert FFB**.

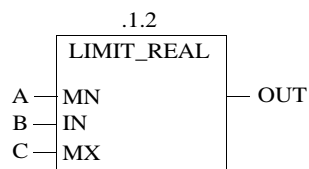
The picture shows calling a function in IL.

```

LD A
LIMIT_REAL B,C
ST OUT

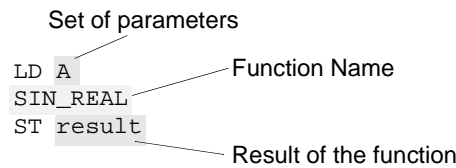
```

The picture shows calling a function in FDP.



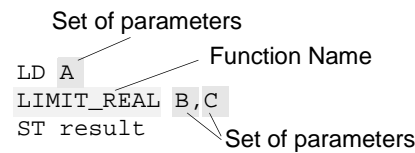
Functions which cannot be used Functions having one or more outputs of data type ANY but no inputs of data type ANY (generic outputs/inputs) cannot be used in IL.

Calling a function with an input If the function to be executed has only got one input, the name of the function is not followed by a list of actual parameters.

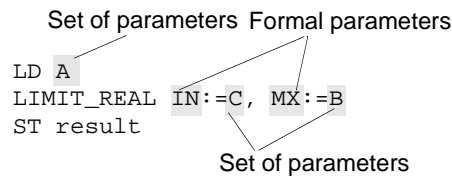


Calling a function with multiple inputs If the function to be executed has several inputs, there are two possibilities for assigning the actual parameters.

- The name of the function is followed by a list of the actual parameters



- The name of the function is followed by a list of the value assignments (actual parameters) to the formal parameters.



Function calls including processing of the battery value If the value to be processed is already in the battery, it is not necessary to use the loading instruction.

```
LIMIT_REAL B,C
ST result
```

Function calls including further direct processing of the result If the result is immediately to be processed further it is not necessary to include the memory instruction.

```
LD A
LIMIT_REAL B,C
MUL E
```

10.5 Syntax check and Code generation

At a Glance

Overview

This section describes the syntax check and the code generation with the IL instruction list.

What's in this Section?

This section contains the following topics:

Topic	Page
Syntax Check	330
Code generation	332

Syntax Check

Introduction A syntax check can be performed during the program/DFB creation with **Project** → **Analyze section**.

Syntax Check Options With the menu command **Options** → **Preferences** → **IEC Extensions...** → **IEC-Extensions** the syntax check options can be defined.

Note: The settings in this dialog are used in the project description (PRJ.DSK) and in the Concept installations description (CONCEPT.DSK), i.e. they are valid for the entire Concept installation.
If a project is opened, that was created using different settings (e.g. **Allow nested comments** in the project and not in the actual Concept Installation), errors can occur when opening the project.

Allow Case Insensitive Keywords If the check box **Allow case insensitive keywords** is checked, upper and lower case for all keywords is enabled.

Allow nested comments If the check box **Nested comments authorized** is checked, nested comments can be entered. There are no limits to the nesting depths.

Comments everywhere in the text permitted (IL) If the check box **Comments everywhere in the text permitted (IL)** is checked, comments can be placed anywhere in the IL section.

Additional Variable Names Permitted (IL) If the check box **Additional variable names permitted (IL)** is checked, the use of additional variable names (e.g. "S1" or "IN") is possible in IL. (These variables can always be used in FBD, LD and ST.)

Allow Leading Digits in Identifiers If the check box **Allow leading digits in identifiers** is checked, figures as the first character of identifiers (i.e. variable names, step names, EFB names) are possible. Identifiers, which consist solely of figures are, however, not authorized, they must contain at least one letter.

**Unassigned
Parameters
Cause Warnings**

The IEC 1131-3 permits functions and Function Blocks to be called up without calling up the assignment of all the input parameters. These unused parameters are implicitly assigned a 0, or they retain the value from the last call up (Function Blocks only).

If in the menu command **Options** → **Preferences** → **Analysis...** → **Analysis Preferences** the check box **Unassigned parameters cause warnings** is checked, a list of these unused parameters is displayed in the message window when generating the code.

Code generation

At a Glance

The menu command **Project** → **Options for code generation** is used to define options for code generation.

Fastest code (restricted check)

If the check box **Fastest Code (restricted check)** is activated, a runtime-optimized code will be generated.

This runtime optimization is achieved with integer arithmetic (e.g. "+" or "-") with simple process commands instead of EFB calls.

Process commands are much faster than EFB calls, but they generate no error messages, such as e.g. Arithmetic or Array overrun. This option should only be used if it has been ensured that the program is free of arithmetic errors.

Example: Fastest Code

```
LD in1
ADD 1
ST out1
```

If **Fastest Code (restricted check)** is selected, the addition "in1 + 1" is executed with the process command "add". The code is now faster than if EFB ADD_INT had been called up. However, no runtime error is generated if "in1" is 32767. In this case, "out1" would overrun from 32767 to -32768!

Activate loop control

This check box activates a software watchdog for continuous loops.

If this check box is checked, with loops within IL and ST sections, it is tested whether these loops are again exited within a certain time. The time authorized depends on the manually defined watchdog time. The authorized time for **all loops** combined constitutes 80% of the Hardware watchdog time. In this way triggering of the hardware watchdog by endless loops is disabled. If a time consuming loop or an endless loop is detected, processing of the affected section will stop, an entry in the Event display will be generated and processing of the next section will begin. In the next cycle, the segment will be re-processed until a time consuming loop or an endless loop is detected once again, or until the segment is completed correctly.

Note: If the hardware watchdog stops the PLC when a time consuming loop or an endless loop is detected, this option should not be activated. The hardware watchdog itself is not switched off by this function.

10.6 Online functions of the IL instruction list

At a Glance

Overview

This section describes the online functions of the IL instruction list.

What's in this Section?

This section contains the following topics:

Topic	Page
Animation	334
Monitoring field	337

Animation

At a Glance

There are two animation modes available in the IL and ST editor:

- Animation of binary variables
 - Animation of selected variables
-

Animation of binary variables

The animation of the selected objects is activated with the menu command **Online** → **Animate selection**.

In this mode, the current signal status of binary values is shown in the editor window.

The animation of direct addresses and direct FB input/outputs is not possible.

Animation of selected variables

The dialog box for the display of the current signal status of selected variables is activated with the menu command **Online** → **Watch Selected**.

Furthermore, at least one variable, which can be animated, must be selected.

Variables and multi-element variables that can be selected are denoted by red, green or yellow script.

Properties of the dialog box

The name of the selected variables or multi-element variables are shown in the dialog box, with the data type and current value.

The dialog box is modeless, i.e. it remains open until it is closed or the animation is terminated. If several text language sections are open and clicked on in this dialog box, a dialog box is opened for each section. The name of the section is displayed in the dialog box heading.

Color key

There are 12 different color schemes available for animation. An overview of the color scheme and the meaning of each color can be found in the Online help (Tip: Search the online help for the index reference "Colors").

Inserting several variables

The procedure for inserting several variables is as follows:

Step	Action
1	Select the desired variables or multi-element variables.
2	Accept this with Online → Animate selected in the dialog box.


Inserting all variables

The procedure for inserting all the variables is as follows:

Step	Action
1	Select the whole section with CTRL+A .
2	Migrate all variables and multi-element variables of the dialog section with Online → Animate selected to the dialog box.

Altering column width

The procedure for altering the column width is as follows:

Step	Action
1	Position the mouse pointer on the right margin button. Reaction: The mouse pointer changes its shape to  .
2	Alter the column width by dragging with the left mouse button depressed.

Multi-element variables

With multi-element variables the display of the elements can be switched on or off.

Action	Function	Condition
Click on symbol + or key +	The next component level for the current line is shown.	When using the keyboard, the cursor must remain on a + symbol.
Key x (number lock)	All component levels for the current line are shown.	The cursor must remain on a + symbol.
Click on symbol - or key -	All component levels for the current line, which are shown, are grayed out.	When using the keyboard, the cursor must remain on a - symbol.
CTRL++	The display of the components of the current line is restored (Restoration of display before the last activation of -	The cursor must remain on a + symbol.
CTRL+x (number lock)	All component levels of the current multi-element variables are shown.	The cursor must remain on an element of a multi-element variable.
CTRL+-	All component levels of the current multi-element variables are grayed out.	The cursor must remain on an element of a multi-element variable.
CTRL+end	to go to the end of the table	
CTRL+Pos1	to go to the start of the table	

Saving and restoring animation

With the menu command **Save animation** the settings (e.g. Position of monitoring fields) of the current animation can be saved. After terminating this animation, the animation can be restored with the same settings via the menu command **Restore animation**.

Note: To avoid inconsistencies between the program on the PC and the PLC and to also have the animation available in the next Concept sitting, the project must be saved when quitting Concept

Monitoring field

At a Glance With the menu command **Online** → **Selected in Inspect field** a monitoring field can be entered in the section. The current value of the assigned variables is shown in this monitoring field.

Limitations The generation of monitoring fields for direct addresses and direct FB input/outputs (INST.Q) is not possible.

Display of multi-element variables With multi-element variables, the value of the first element is shown.
If a view of more elements is desired, this can be defined in the dialog **Settings for monitoring field**, which is called up by double clicking on the monitoring field.

Minimum and maximum values In the dialog **Settings for monitoring field**, which can be called up with a double click on the monitoring field, a minimum and maximum value can be defined for the monitored variable. If the variable violates one of these thresholds, this will be displayed in color in the monitoring field.

An overview of the color scheme and the meaning of each color can be found in the Online help (Tip: Search the online help for the index reference "Colors").

Generating a monitoring field The procedure for generating a monitoring field is as follows:

Step	Action
1	Select a variable (e.g. double-click on variable).
2	Execute the menu command Online → Selected in Inspect field . Reaction: The section animation is started (gray section background) and the cursor symbol changes into box symbol.
3	Position the cursor to any position in the section and click with the left mouse button. Reaction: A monitoring field, consisting of variable name and value, is generated for the selected variable on the chosen position.

10.7 Creating a program with the IL instruction list

Creating a program in the IL instruction list.

At a Glance

The following description contains an example of creating a program in IL instruction list. The creation of a program in IL instruction list is organized into 2 main steps:

Step	Action
1	Generating a section (See <i>Generating a section</i> , p. 339)
2	Creating the logic (See <i>Creating the logic</i> , p. 340)

Generating a section

The procedure for generating a section is as follows:

Step	Action
1	<p>Using the menu command File → New section... and enter a section name.</p> <p>Note: The section name (max. 32 characters) must be clear throughout the project, so that there is no difference regarding case sensitivity. If the name entered already exists, a warning is given and another name must be chosen. The section name must correspond to the IEC name conventions, otherwise an error message arises.</p> <p>Note: In accordance with IEC1131-3, only letters are permitted as the first character of names. Should numbers be required as the first character, however, the menu command Options → Preferences → IEC Extensions... → Allow leading digits in identifiers .</p>

Creating the logic

The procedure for creating the logic is as follows:

Step	Action
1	<p>Declare the Function Block and DFBs, which are to be used, with assistance from VAR...END_VAR.</p> <p>Example:</p> <pre>VAR RAMP_UP, RAMP_DOWN, RAMP_X : TON ; COUNT : CTU_DINT ; END_VAR</pre>
2	<p>Declare the variables and their initial value in the Variable Editor.</p>
3	<p>Create the logic of the program.</p> <p>Example:</p> <pre>LD A SIN_REAL MUL_REAL B,C ST D LD Y AND X JMPC end1 LD M SIN_REAL MUL_REAL N,O ST P JMP end2 end1: LD D ST %QD4 end2: LD P ST %QD5</pre>
4	<p>save the section with the menu command Data file → Save project .</p>

Structured text ST

11

At a Glance

Overview

This Chapter describes the programming language structured text ST which conforms to IEC 1131.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
11.1	General information about structured Text ST	343
11.2	Expressions	345
11.3	Operators of the programming language of structured ST text	350
11.4	Assign instructions	357
11.5	Call up of functions, Function Blocks (EFBs) and Derived Function Blocks (DFBs)	372
11.6	Syntax check and code generation	379
11.7	Online functions of the ST programming language	382
11.8	Creating a program with the structured ST text	384

Structured text ST

11.1 General information about structured Text ST

General Information about the ST Structured Text

Introduction	With the programming language of structured text (ST), it is possible, for example, to call up Function Blocks, perform functions and assignments, conditionally perform instructions and repeat tasks.
Spell Check	Spelling is immediately checked when key words, separators and comments are entered. If a key word, separator or comment is recognized, it is identified with a color surround. If unauthorized key words (instructions or operators) are entered, it is likewise identified in color.
IEC Conventions	<p>The IEC 1131 does not permit the input of direct addresses in the usual Concept form. To input direct addresses see <i>Operands</i>, p. 346.</p> <p>In accordance with IEC 113-3, key words must be entered in upper case. Should the use of lower case letters be required, they can be enabled in the dialog box Options → Preferences → IEC Extensions... → IEC Extensions with the option Allow case insensitive keywords.</p> <p>Blank spaces and tabs have no influence upon the syntax and can be used freely.</p>
Context help	With the right mouse button an object can be selected and at the same time a context sensitive menu called up. Therefore, for example, with FFBs the right mouse button can call up the associated block description.
Syntax Check	A syntax check can be performed during the program/DFB creation with Project → Analyze section , see also <i>Syntax Check</i> , p. 380.
Codegeneration	Using the Project → Code Generation Options menu command, you can define options for code generation, see also <i>Code generation</i> , p. 381.
Editing with the Keyboard	Normally editing in Concept is performed with the mouse, however it is also possible with the keyboard (see also <i>Short Cut Keys in the IL, ST and Data Type Editor</i> , p. 765).
IEC Conformity	For a description of the IEC conformity of the ST programming language see <i>IEC conformity</i> , p. 779.

11.2 Expressions

At a Glance

Overview

This section contains an overview of the expressions in the programming language of structured text ST. expressions consists of operands and operators.

What's in this Section?

This section contains the following topics:

Topic	Page
Operands	346
Operators	347

Operands

At a Glance

An operand can be:

- a literal,
 - a variable,
 - a multi-element variable,
 - an element of a multi-element variable,
 - a function call up,
 - a FB/DFB output or
 - a direct address.
-

Access to the field variables

When accessing field variables (ARRAY), only literals and variables of ANY_INT type are permitted in the index entry.

Example: Using field variables

```
var1[i] := 8 ;  
var2.otto[4] := var3 ;  
var4[1+i+j*5] := 4 ;
```

Type conversion

Data types, which are in an instruction of processing operands, must be identical. Should operands of various types be processed, a type conversion must be performed beforehand.

An exception is the data type TIME in conjunction with the arithmetic operators "*" (multiplication) and "/" (division). With both these operators, an operand of TIME data type can be processed together with an operand of ANY_NUM data type. The result of this instruction has in this instance the data type TIME.

Example: Integer variable and real variable

In the example the integer variable i1 is converted into a real variable before being added to the real variable r4.

```
r3 := r4 + SIN_REAL(INT_TO_REAL(i1)) ;
```

Example: Integer variable and time variable

In the example the time variable t2 is multiplied by the integer variable i4 and the result is stored in the time variable t1.

```
t1 := t2 * i4 ;
```

Default data types of direct addresses

The following table shows the default data types of direct addresses:

Input	Output	Default data type	possible data type
%IX,%I	%QX,%Q	BOOL	BOOL
%IB	%QB	BYTE	BYTE
%IW	%QW	INT	INT, UINT, WORD
%ID	%QD	REAL	REAL, DINT, UDINT, TIME

Using other data types

Should other data types be assigned as default data types of a direct address, this must be done through an explicit declaration (VAR...END_VAR (See *Declaration (VAR...END_VAR)*, p. 360)). VAR...END_VAR cannot be used in Concept for the declaration of variables. The variable declaration is performed conveniently by using the Variable Editor (See *Variables editor*, p. 479).

Operators**Introduction**

An operator is a symbol for:

- an arithmetic operation to be executed or
- a configured operation to be executed or
- the function call up.

Operators are generic, i.e. they are automatically matched with the operands data type.

Note: Operators can be either entered manually or generated with assistance from the menu **Objects** → **Operators**.

Expression Evaluation

The evaluation of an expression consists of applying the operators to the operands, in the sequence, which is defined by the order of the operators rank (see table). The operator with the highest rank in an expression is performed first, followed by the operator with the next highest rank etc. until the evaluation is complete. Operators with the same rank are performed from left to right, as they are written in the expression. This sequence can be altered with the use of parentheses.

Table of Operators

ST programming language operators:

Operator	Meaning	possible operand	Order of rank	see also
()	Use of parentheses:	Expression	1 (highest)	<i>Use of parentheses "()", p. 351</i>
FUNCNAME (current parameter list)	Function editing (call up)	Expression, literal, variable, direct address of ANY data type	2	<i>Function Invocation, p. 377</i>
-	Negation	Expression, literal, variable, direct address of ANY_NUM data type	3	<i>Negation (-), p. 352</i>
NOT	Complement	Expression, literal, variable, direct address of ANY_BIT data type	3	<i>Complement formation (NOT), p. 352</i>
**	Exponentiation	Expression, literal, variable, direct address of REAL data type (basis), ANY_NUM (exponent)	4	<i>Exponentiation (**), p. 351</i>
*	Multiplication	Expression, literal, variable, direct address of ANY_NUM data type or TIME data type	5	<i>Multiplication (*), p. 352</i>
/	Division	Expression, literal, variable, direct address of ANY_NUM data type	5	<i>Division (/), p. 353</i>
MOD	Modulo	Expression, literal, variable, direct address of ANY_INT data type	5	<i>Modulo (MOD), p. 353</i>
+	Addition	Expression, literal, variable, direct address of ANY_NUM data type or TIME data type	6	<i>Addition (+), p. 353</i>
-	Subtraction	Expression, literal, variable, direct address of ANY_NUM data type or TIME data type	6	<i>Subtraction (-), p. 354</i>
<	Less-than comparison	Expression, literal, variable, direct address of ANY_ELEM data type	7	<i>Comparison with "Less Than"(<), p. 355</i>

Operator	Meaning	possible operand	Order of rank	see also
>	Greater-than comparison	Expression, literal, variable, direct address of ANY_ELEM data type	7	<i>Comparison on "Greater Than" (>), p. 354</i>
<=	Less or equal to comparison	Expression, literal, variable, direct address of ANY_ELEM data type	7	<i>Comparison with "Less than or Equal to" (<=), p. 355</i>
>=	Greater or equal to comparison	Expression, literal, variable, direct address of ANY_ELEM data type	7	<i>Comparison on "Greater than/ Equal to" (>=), p. 354</i>
=	Equality	Expression, literal, variable, direct address of ANY_ELEM data type	8	<i>Comparison with "Equal to" (=), p. 354</i>
<>	Inequality	Expression, literal, variable, direct address of ANY_ELEM data type	8	<i>Comparison with "Not Equal to" (<>), p. 355</i>
&, AND	configured AND	Expression, literal, variable, direct address of ANY_BIT data type	9	<i>Boolean AND (AND or &), p. 356</i>
XOR	Configured exclusive OR	Expression, literal, variable, direct address of ANY_BIT data type	10	<i>Boolean Exclusive OR (XOR), p. 356</i>
OR	Configured OR	Expression, literal, variable, direct address of ANY_BIT data type	11 (lowest)	<i>Boolean OR (OR), p. 356</i>

11.3 Operators of the programming language of structured ST text

At a Glance

Overview This section describes the operators of the programming language of structured ST text.

What's in this Section? This section contains the following topics:

Topic	Page
Use of parentheses "()"	351
FUNCNAME	351
Exponentiation (**)	351
Negation (-)	352
Complement formation (NOT)	352
Multiplication (*)	352
Division (/)	353
Modulo (MOD)	353
Addition (+)	353
Subtraction (-)	354
Comparison on "Greater Than" (>)	354
Comparison on "Greater than/Equal to" (>=)	354
Comparison with "Equal to" (=)	354
Comparison with "Not Equal to" (<>)	355
Comparison with "Less Than"(<)	355
Comparison with "Less than or Equal to" (<=)	355
Boolean AND (AND or &)	356
Boolean OR (OR)	356
Boolean Exclusive OR (XOR)	356

Use of parentheses "()"

Description Brackets are used to alter the execution sequence of the operators.

Example of parentheses "()" If the operands A, B, C, and D have the values "1", "2", "3", "and -4",

$A+B-C*D$

has the result 15 and

$(A+B-C)*D$

has the result 0.

FUNCNAME

Description The function processing is used to perform functions (see *Function Invocation*, p. 377).

Exponentiation (**)

Description For exponentiation "****" the value of the first operand (basis) is potentiated with that of the second operand (exponent).

Note: Exponentiation operates in the ST programming language and with a resolution of 23 bits. For the graphic languages the exponentiation operates with a resolution of 24 bits..

Example exponentiation "**"** In the example OUT will be "625.0", when IN1 is "5.0" and IN2 is "4.0".
`OUT := IN1 ** IN2 ;`

Negation (-)

Description During negation "-" a sign reversal for the value of the operand takes place.

Example negation "-" In the example OUT will be "-4", when IN1 is "4".
OUT := - IN1 ;

Complement formation (NOT)

Description In NOT a bit by bit inversion of the operands takes place.

Example NOT In the example OUT will be "0011001100", when IN1 is "1100110011".
OUT := NOT IN1 ;

Multiplication (*)

Description For multiplication "*" the value of the first operand is multiplied with that of the second operand (exponent).

Example multiplication "*" OUT := IN1 * IN2 ;

Multiplication of TIME values Normally the data types of the operands to be processed must be identical to an instruction. However, the multiplication forms an exception when combined with data type TIME. In this case an operand with the datatype TIME combined with an operand of data type ANY_NUM can be processed. In this case the result of this instruction has the data type TIME.

Example: Multiplication of TIME values In the example the Time variable t2 is multiplied by the integer variables i4 and the result is deposited in the t1 Time variables.
t1 := t2 * i4 ;

Division (/)

Description For division "/" the value of the first operand is divided by that of the second operand (exponent).

Example division "/" `OUT := IN1 / IN2 ;`

Division of TIME values Normally the data types of the operands to be processed must be identical to an instruction. However the division forms an exception when combined with data type TIME. In this case an operand with the data type TIME combined with an operand of data type ANY_NUM can be processed. In this case the result of this instruction has the data type TIME.

Example division of TIME values In the example the Time variable t2 is divided by the integer variables i4 and the result is deposited in the t1 Time variables.
`t1 := t2 / i4 ;`

Modulo (MOD)

Description For MOD the value of the first operand is divided by that of the second operand and the remainder of the division (Modulo) is displayed as the result.

Example MOD `OUT := IN1 MOD IN2 ;`

Addition (+)

Description For the addition "+" the value of the first operand is added to that of the second operand.

Example addition "+" `OUT := IN1 + IN2 ;`

Subtraction (-)

Description For the subtraction "-" the value of the second operand is subtracted from that of the first operand.

Example Subtraction "-" `OUT := IN1 - IN2 ;`

Comparison on "Greater Than" (>)

Description With ">" the value of the first operand is compared with that of the second operand. If the first operand is greater than the second, the result is a boolean "1". If the first operand is less than or equal to the second, the result is a Boolean "0".

Example greater than ">" In the example "OUT" will be "1" if "IN1" is greater than "10" and "0", if "IN1" is less than "0".
`OUT := IN1 > 10 ;`

Comparison on "Greater than/Equal to" (>=)

Description With ">=" the value of the first operand is compared to that of the second operand. If the first operation is greater than or equal to the second, the result is a Boolean "1". If the first operand is less than the second, the result is a Boolean "0".

Example greater than/equal to ">=" In the example "OUT" will be "1" if "IN1" is greater than/equal to "10" and otherwise "0".
`OUT := IN1 >= 10 ;`

Comparison with "Equal to" (=)

Description The value of the first operation is compared with the value of the second with "=". If the first operation is equal to the second, the result is a Boolean "1". If the first operation is not equal to the second, the result is a Boolean "0".

Example equal to "=" In the example, "OUT" will be "1", if "IN1" is equal to "10" – otherwise it will be "0".
`OUT := IN1 = 10 ;`

Comparison with "Not Equal to" (<>)

Description The value of the first operation is compared with the value of the second with "<>". If the first operation is not equal to the second, the result is a Boolean "1". If the first operation is equal to the second, the result is a Boolean "0".

Example Not equal to "<>" In the example, "OUT" will be "1", if "IN1" is not equal to "10" – otherwise it will be "0".
`OUT := IN1 <> 10 ;`

Comparison with "Less Than"(<)

Description The value of the first operation is compared with the value of the second with "<". If the first operation is smaller than the second, the result is a Boolean "1". If the first operation is bigger than or the same size as the second, the result is a Boolean "0".

Example less than "<" In the example, "OUT" will be "1", if "IN1" is less than "10" – otherwise it will be "0".
`OUT := IN1 < 10 ;`

Comparison with "Less than or Equal to" (<=)

Description The value of the first operation is compared with the value of the second with "<=". If the first operation is less than or equal to the second, the result is a Boolean "1". If the first operation is greater than the second, the result is a Boolean "0".

Example less than or equal to "<=" In the example, "OUT" will be "1", if "IN1" is less than or equal to "10" – otherwise it will be "0".
`OUT := IN1 <= 10 ;`

Boolean AND (AND or &)

Description With "AND" or "&" a configured AND link occurs between the operations.
With the BYTE and WORD data types, the link is performed bit by bit.

Example In the examples, "OUT" will be "1" if "IN1", "IN2" and "IN3" are "1".

Boolean "AND or &" `OUT := IN1 AND IN2 AND IN3 ;`

or

`OUT := IN1 AND IN2 AND IN3 ;`

Boolean OR (OR)

Description With OR, a configured OR link occurs between the operations.
With the BYTE and WORD data types, the link is performed bit by bit.

Example In the example, "OUT" will be "1" if "IN1", "IN2" or "IN3" is "1".

Boolean OR "OR" `OUT := IN1 OR IN2 OR IN3 ;`

Boolean Exclusive OR (XOR)

Description With XOR, a configured Exclusive OR link occurs between the operations.
With the BYTE and WORD data types, the link is performed bit by bit.

Example In the example "OUT" will be "1", if "IN1" and "IN2" are not equal. If "IN1" and "IN2" have the same state (both "0" or "1"), "OUT" is "0".

Boolean Exclusive OR "XOR" `OUT := IN1 XOR IN2 ;`

Linking more than 2 operations If more than two operations are linked, the result is "1" with an odd number of 1-states and "0" with an even number of 1-states.

Example: In the example, "OUT" will be "1" if 1, 3 or 5 operations are "1". "OUT" will be "0" if 0, 2 or 4 operations are "1".

Linking more than 2 operations `OUT := IN1 XOR IN2 XOR IN3 XOR IN4 XOR IN5 ;`

11.4 Assign instructions

At a Glance

Overview

This section describes the instructions for the programming language of structured ST text.

What's in this Section?

This section contains the following topics:

Topic	Page
Instructions	358
Assignment	359
Declaration (VAR...END_VAR)	360
IF...THEN...END_IF	362
ELSE	363
ELSIF...THEN	364
CASE...OF...END_CASE	365
FOR...TO...BY...DO...END_FOR	366
WHILE...DO...END_WHILE	368
REPEAT...UNTIL...END_REPEAT	370
EXIT	371
Empty instruction	371
Comment	371

Instructions

Description

Instructions are the "commands" of the ST programming language.

Instructions must be completed by semicolons. Several instructions can be entered in one line (separated by semicolons).

<p>Note: Instructions can be either entered manually or generated using the menu Objects.</p>

Assignment

At a Glance

When an assignment is performed, the current value of a single or multi-element variable is replaced by the result of the evaluation of the expression
An assignment consists of a variable specification on the left side, followed by the assignment operator ":", followed by the expression to be evaluated. Both variables must be of the same data type.

Assigning the value of a variable to another variable

Assignments are used to assign the value of a variable to another variable.
The instruction

```
A := B ;
```

is for instance used to replace the value of the variable "A" by the current value of the variable "B". If "A" and "B" are of an elementary data type, the individual value "B" is passed to "A". If "A" and "B" are of a derived data type, the values of all elements are passed from "B" to "A".

Assigning the value of a literal to a variable

Assignments are used to assign a literal to variables.

The instruction

```
C := 25 ;
```

is for instance used to assign the value "25" to the variable "C".

Assigning the value of an FFB to a variable

Assignments are used to assign a value to a variable which is returned by a function or a function block.

The instruction

```
B := MOD_INT(C, A) ;
```

is for instance used to assign the modulo of the variables "C" and "A" to the variable "B".

The instruction

```
A := TON1.Q ;
```

is for instance used to assign to the variable "A" the value of the output "Q" of the function block TONI.

Assigning the value of an operation to a variable

Assignments are used to assign to a variable a value which is the result of an operation.

The instruction

```
X := (A+B-C)*D ;
```

is for instance used to assign to the variable "X" the result of the operation "(A+B-C)*D".

Declaration (VAR...END_VAR)

At a Glance

The VAR instruction is utilized for declaring the function blocks used and DFBs and declaring direct addresses if they are not to be used with the default data type. VAR cannot be used for declaring a variable in Concept. Declaring the variables may conveniently be done via the Variables editor.

The END_VAR instruction marks the end of the declaration.

Note: The declaration of the FBs/DFBs and direct addresses applies only to the current section. If the same FFB type or the same address are also used in another section, the FFB type or the address must be declared again in this section.

Declaration of function blocks and DFBs

Every time a FB/DFB example is used, a unique example name is assigned when it is declared. The example name is used to mark the function block uniquely in a project. The example name must be unique in the whole project; no distinction is made between upper/lower case. The example name must correspond to the IEC Name conventions, otherwise an error message will be displayed.

After specifying the example name, the function block type, e.g.CTD_DINT is specified.

In the case of function block types no data type is specified. It is determined by the data type of the actual parameters. If all actual parameters consist of literals, a suitable data type will be selected.

Any number of example names may be declared for an FB/DFB.

Note: The dialog **Objects** → **Insert FFB** provides you with a form for creating the FB-/DFB declaration in a simple and speedy manner.

Note: In contrast to grafic programming languages (FBD, LD), it is possible to execute multiple calls in FB/DFB examples within ST.

Example

Declaration of function blocks and DFBs

Exemplar-Namen

```

VAR
  RAMP_UP, RAMP_DOWN, RAMP_X : TON ;
  COUNT : CTU_DINT ;
  CLOCK : SYSCLOCK ;
  Pulse : TON ;
END_VAR

```

Funktionsbaustein-Typen

Declaration of direct addresses

In the case of this declaration, every direct address used whose data type does not correspond to the default data type will be assigned the required data type (see also *Default data types of direct addresses*, p. 286).

Example

Declaration of direct addresses

```

VAR
  AT %QW1 : WORD ;
  AT %IW15 : UINT ;
  AT %ID45 : DINT ;
  AT %QD4 : TIME ;
END_VAR

```

IF...THEN...END_IF

Description

The IF instruction determines that an instruction or a group of instructions will only be executed if its related Boolean expression has the value 1 (true). If the condition is 0 (false), the instruction or the instruction group will not be executed.

The THEN-command identifies the end of the condition and the beginning of the command(s).

The END_IF instruction marks the end of the instruction(s).

Note: Any number of IF...THEN...ELSE...END_IF commands may be nested to generate complex selection commands.

Example

IF...THEN...END_IF

If FLAG is 1, the instructions will be executed; if FLAG is 0, they will not be executed.

```
IF FLAG THEN
    C:=SIN_REAL(A) * COS_REAL(B) ;
    B:=C - A ;
END_IF ;
```

Example IF

NOT...THEN...END_IF

Using NOT, the condition may be inverted (execution of both instructions at 0).

```
IF NOT FLAG THEN
    C:=SIN_REAL(A) * COS_REAL(B) ;
    B:=C - A ;
END_IF ;
```

Related topic(s)

ELSE (See *ELSE*, p. 363)
ELSEIF (See *ELSIF...THEN*, p. 364)

ELSE

Description

The ELSE command always comes after an IF...THEN-, ELSIF...THEN- or CASE-command.

If the ELSE command comes after IF or ELSIF, the command or group of commands will only be executed if the associated Boolean expressions of the IF and ELSIF command have the 0 value (false). If the condition of the IF or ELSIF command is 1 (true), the command or group of commands will not be executed.

If the ELSE command comes after CASE, the command or group of commands will only be executed if no identification contains the value of the selector. If an identification contains the value of the selector, the command or group of commands will not be executed.

Note: As many IF...THEN...ELSE...END_IF-commands as required can be encapsulated to create complex selection commands.

E.g. ELSE

```
IF A>B THEN
  C:=SIN_REAL(A) * COS_REAL(B) ;
  B:=C - A ;
ELSE
  C:=A + B ;
  B:=C - A ;
END_IF ;
```

Related topic(s)

IF (See *IF...THEN...END_IF*, p. 362)
ELSIF (See *ELSIF...THEN*, p. 364)
CASE (See *CASE...OF...END_CASE*, p. 365)

ELSIF...THEN

Description

The ELSIF-command always comes after an IF...THEN-command. The ELSIF-command establishes that a command or group of commands will only be executed if the associated Boolean expression of the IF-command has the 0 value (false) and the associated Boolean expression of the ELSIF command has the 1 value (true). If the condition of the IF-command is 1 (true) or the condition of the ELSIF-command is 0 (false), the command or group of commands will not be executed. The THEN-command identifies the end of the ELSIF-condition(s) and the beginning of the command(s).

Note: As many IF...THEN...ELSIF...THEN...END_IF-commands as required can be encapsulated to create complex selection commands.

**E.g.
ELSIF...THEN**

```
IF A>B THEN
  C:=SIN_REAL(A) * COS_REAL(B) ;
  B:=SUB_REAL(C,A) ;
ELSIF A=B THEN
  C:=ADD_REAL(A,B) ;
  B:=MUL_REAL(C,A) ;
END_IF ;
```

**E.g.
Encapsulated
Commands**

```
IF A>B THEN
  IF B=C THEN
    C:=SIN_REAL(A) * COS_REAL(B) ;
  ELSE
    B:=SUB_REAL(C,A) ;
  END_IF ;
ELSIF A=B THEN
  C:=ADD_REAL(A,B) ;
  B:=MUL_REAL(C,A) ;
ELSE
  C:= DIV_REAL (A,B) ;
END_IF ;
```

Related topic(s)

IF (See *IF...THEN...END_IF*, p. 362)
ELSE (See *ELSE*, p. 363)

CASE...OF...END_CASE

Description

The CASE instruction consists of an INT data type expression (the "selector") and a list of instruction groups. Each group is provided with a mark which consists of one or several whole numbers (ANY_INT) or zones of whole number values. The first group is executed by instructions, whose mark contains the calculated value of the selector. Otherwise none of the instructions will be executed. The OF instruction indicates the start of the mark. An ELSE instruction may be carried out within the CASE instruction, whose instructions are executed if no mark contains the selector value. The END_CASE instruction marks the end of the instruction(s).

**Example
CASE...OF...END_CASE**

Example CASE...OF...END_CASE

```

Selector
CASE SELECT OF 1,5: C:=SIN_REAL(A) * COS_REAL(B) ;
2: B:=C - A ;
6:..10: C:=C * A ;
ELSE B:=C * A ;
      C:=A / B ;
END_CASE ;
Mark

```

Related topic(s)ELSE (See *ELSE*, p. 363)

FOR...TO...BY...DO...END_FOR

Description

The FOR instruction is used when the number of occurrences can be determined in advance. Otherwise WHILE (See *WHILE...DO...END_WHILE*, p. 368) or REPEAT (See *REPEAT...UNTIL...END_REPEAT*, p. 370) are used

The FOR instruction repeats an instruction sequence until the END_FOR instruction. The number of occurrences is determined by start value, end value and control variable. Start value, end value and the control variable must be the same type of data (DINT or INT) and may not be modified by one of the repeated instructions. The FOR instruction increments the control variable value of one start value to an end value. The increment value has the default value 1. If a different value is to be used, it is possible to specify an explicit increment value (variable or constant). The control variable value is checked before each renewed loop running. If it is outside the start value and end value range, the loop will be left.

Before running the loop for the first time a check is made to determine whether incrementation of the control variables, starting from the initial value, is moving towards the end value. If this is not the case (e.g. initial value \leq end value and negative increment), the loop will not be processed.

Using this ruler, continuous loops will be prevented.

Note: For the end value of the data type DINT the range of values -2 147 483 646 to 2 147 483 645 will apply.

The DO command identifies the end of the repeat definition and the beginning of the instruction(s).

Repetition may be terminated early by using the EXIT instruction. The END_FOR instruction marks the end of the instruction(s).

Example: FOR with increment "1"

FOR with increment "1"

Control variable Start value End value

```
FOR i := 1 TO 50 DO
  C := C * COS_REAL(B) ;
END_FOR ;
```

FOR with increment not equal to "1"

If an increment other than "1" is to be used, this can be defined by BY. The increment, the initial value, the end value, and the control variable must be of the same data type (DINT or INT). The criterion for the processing direction (forward, backward) is the sign of the BY expression. If this expression is positive, the loop will run forward; if it is negative, the loop will run backward.

**Example:
Counting
forward in two
steps**

Counting forward in two steps

Control variable Start value End value Increment

```
FOR i:= 1 TO 10 BY 2 DO (* BY > 0 : Forward loop *)
  C:= C * COS_REAL(B) ; (* instruction will be 5 x executed *)
END_FOR ;
```

**Example:
Counting
backward**

Counting backward

```
FOR i:= 10 TO 1 BY -1 DO (* BY < 0 : Backward loop *)
  C:= C * COS_REAL(B) ; (* Application will be executed 10
x *)
END_FOR ;
```

**Example:
"Unique" loops**

The loops in the example are run once precisely, as the initial value = end value. In this context it does not matter whether the increment is positive or negative.

```
FOR i:= 10 TO 10 DO (* Unique Loop *)
  C:= C * COS_REAL(B) ;
END_FOR ;
```

or

```
FOR i:= 10 TO 10 BY -1 DO (* Unique Loop *)
  C:= C * COS_REAL(B) ;
END_FOR ;
```

**Example: Critical
loops**

If in the example there is the increment $j > 0$, the instructions will not be executed, as the situation initial value $>$ end value only permits an increment ≤ 0 . A continuous loop can only arise if the increment is 0. If this situation is identified during the section analysis, an error message will be generated. If the error is identified during running time, an error message will be generated in the event viewer.

```
FOR i:= 10 TO 1 BY j DO (* Backward loop *)
  C:= C * COS_REAL(B) ;
END_FOR ;
```

If in the example there is the increment $j < 0$, the instructions will not be executed, as the situation initial value $<$ end value only permits an increment ≥ 0 . A continuous loop can only arise if the increment is 0. If this situation is identified during the section analysis, an error message will be generated. If the error is identified during running time, an error message will be generated in the event viewer.

```
FOR i:= 1 TO 10 BY j DO (* Forward loop *)
  C:= C * COS_REAL(B) ;
END_FOR ;
```

Example: Illegal loops

Illegal loops

```
FOR i:= 1 TO 10 BY 0 DO (* Error with Section- *)
  C:= C * COS_REAL(B) ; (* Analysis, as continous loop *)
END_FOR ;
```

or

```
FOR i:= 1 TO 10 BY j DO (* at j=0, Error message *)
  C:= C * COS_REAL(B) ; (* in of Event indicator *)
END_FOR ;
```

WHILE...DO...END_WHILE


Description


The WHILE instruction has the effect that a sequence of instructions will be executed repeatedly until its related Boolean expression is 0 (false). If the expression is false right from the start, the group of instructions will not be executed at all.

The DO command identifies the end of the repeat definition and the beginning of the command(s).

The occurrence may be terminated early using the EXIT.

The END_WHILE instruction marks the end of the instruction(s).

	WARNING
	<p>Risk of program crashing</p> <p>WHILE must not be used to carry out synchronization between processes, e.g. as a "waiting loop" with an externally determined end condition. This means that a continous loop must not be created, unless you prevent this using the function Project → Options for code generation → Activate loop control.</p> <p>Failure to follow this precaution can result in death, serious injury, or equipment damage.</p>

	WARNING
	Risk of program crashing WHILE must not be used in an algorithm for which fulfilling the loop end condition or the execution of an EXIT instruction can not be guaranteed. This means that a continuous loop must not be created, as this may result in crashing the program, unless you prevent this by using the function Project → Options for code generation → Activate loop control . Failure to follow this precaution can result in death, serious injury, or equipment damage.

Example
WHILE...DO...EN
D_WHILE

```
var := 1  
WHILE var <= 100 DO  
  var := var + 4;  
END_WHILE ;
```


Related topic(s)


EXIT (See *EXIT*, p. 371)

REPEAT...UNTIL...END_REPEAT

Description

The REPEAT instruction has the effect that a sequence of instructions is executed repeatedly (at least once), until its related Boolean condition is 1 (true). The UNTIL instruction marks the end condition. The occurrence may be terminated early using the EXIT. The END_REPEAT instruction marks the end of the instruction(s).

	WARNING
	<p>Risk of program crashing</p> <p>REPEAT must not be used to carry out synchronization between processes, e.g. as a "waiting loop" with an externally determined end condition. This means that a continuous loop must not be created, unless you prevent this using the function Project → Options for code generation → Activate loop control.</p> <p>Failure to follow this precaution can result in death, serious injury, or equipment damage.</p>

	WARNING
	<p>Risk of program crashing</p> <p>REPEAT must not be used in an algorithm for which fulfilling the loop end condition or the execution of an EXIT instruction can not be guaranteed. This means that a continuous loop must not be created, as this may result in crashing the program, unless you prevent this by using the function Project → Options for code generation → Activate loop control.</p> <p>Failure to follow this precaution can result in death, serious injury, or equipment damage.</p>

Example

REPEAT...UNTIL..
..END_REPEAT

```

var := -1
REPEAT
  var := var + 2
  UNTIL var >= 101
END_REPEAT ;

```

Related topic(s)

EXIT (See *EXIT*, p. 371)

EXIT

Description	The EXIT command is used to terminate repeat instructions (FOR, WHILE, REPEAT) before the end condition has been met. If the EXIT instruction is within a nested occurrence, the innermost loop (in which EXIT is situated) is left. Next, the first instruction following the loop end (END_FOR, END_WHILE or END_REPEAT) is executed.
Example EXIT	If FLAG has the value 0, SUM will be 15 following execution of the instructions. If FLAG has the value 1, SUM will be 6 following execution of the instructions. <pre>SUM := 0 ; FOR I := 1 TO 3 DO FOR J := 1 TO 2 DO IF FLAG=1 THEN EXIT; END_IF; SUM := SUM + J ; END_FOR ; SUM := SUM + I ; END_FOR</pre>
Related topic(s)	CASE (See <i>CASE...OF...END_CASE</i> , p. 365) WHILE (See <i>WHILE...DO...END_WHILE</i> , p. 368) REPEAT (See <i>REPEAT...UNTIL...END_REPEAT</i> , p. 370)

Empty instruction

Description Empty instructions are generated by a semicolon (;).

Comment

Description Within the ST editor, comments start with the string (* and end in the string *). Any comments may be entered between these two strings. Comments may be entered in any position in the ST editor. Comments are shown in colour.

Note: In accordance with IEC 1131-1, nested comments are not permissible. However, if you wish to place these elsewhere, you can release them by using **Options** → **Preferences** → **IEC Extensions** → **Allow nested comments**.

11.5 Call up of functions, Function Blocks (EFBs) and Derived Function Blocks (DFBs)

At a Glance

Overview This section describes the call up of functions, Function Blocks (EFBs) and Derived Function Blocks (DFBs).

What's in this Section? This section contains the following topics:

Topic	Page
Function Block/DFB Invocation	373
Function Invocation	377

Function Block/DFB Invocation

Use of Function Blocks and DFBs

Function blocks are provided by Concept in the form of libraries. The logic of the function blocks is created in C++ programming language and cannot be altered in the ST Editor. The names of the available function blocks can be taken from the block libraries.

DFBs are function blocks which can be defined in Concept-DFB. There is no difference between functions and function blocks for DFBs. They are always handled as function blocks regardless of their internal structure.

The use of function blocks and DFBs consists of three parts in ST:

- the declaration (See *Declaration*, p. 374),
- the function block/DFB invocation (See *Function Block/DFB Invocation*, p. 375),
- the use of the function block/DFB outputs (See *Use of the Function Block/DFB Outputs*, p. 376).

Note: The declaration of the function block/DFB invocation can take place manually or you can create the block end and the assignment of the parameters using the menu command **Objects** → **Insert FFB**.

Function Blocks with Limited Use

Use of the following EFBs from the DIAGNO block library is limited in ST (function blocks can be used, but the expanded diagnostic information cannot be evaluated):

- XACT, XACT_DIA
- XDYN_DIA
- XGRP_DIA
- XLOCK,
- XPRE_DIA
- XLOCK_DIA
- XREA_DIA

Function Blocks with Limited Invocation

For EFBs which have one or more outputs with data type ANY but no inputs with data type ANY (generic outputs/inputs), the block invocation can only take place in compact form (See *Function Block/DFB Invocation in Compact Form*, p. 376). e.g. in the block library **LIB984**:

- GET_3X
 - GET_4X
-

**Unusable
Function Blocks**

Unusable Function Blocks:

- EFBs which use several registers with only the entry for the first register on the input/output (e.g. MBP_MSTR from the COMM block library) cannot be used.
- EFBs which contain outputs with input information (e.g. GET_BIT, R2T from the LIB984 block library) cannot be used
- The following EFBs from the **COMM** block library cannot be used for the technical reasons listed above:
 - CREADREG
 - CREAD_REG
 - CWITREG
 - CWRITE_REG
 - READREG
 - READ_REG
 - WRITEREG
 - WRITE_REG
 - MBP_MSTR
- The following EFBs from the **LIB984** block library cannot be used for the technical reasons listed above:
 - FIFO
 - GET_BIT
 - IEC_BMDI
 - LIFO
 - R2T
 - SET_BIT
 - SRCH
 - T2T

Declaration

Before invoking the function block/DFBs, they must be declared using VAR and END_VAR (See *Declaration (VAR...END_VAR)*, p. 360).

**Function Block/
DFB Invocation**

Function blocks/DFBs are invoked using an instruction consisting of the instance name for the FB/DFB, which is followed by a list, in brackets, of value assignments (current parameters) to formal parameters. The order of the formal parameters in a function block invocation is not significant. It is not necessary for all formal parameters to be assigned a value. If a formal parameter is not assigned a value, the initial value defined in the variable editor is used when executing the function block. If an initial value is not defined, the default value (0) is used.

Note: Inputs of type VARINOUT (See also *Use of the DFB in FBD/LD*, p. 425) always have to be assigned a value.

Function block/DFB invocation:

Instance name

```
CLOCK ( ) ;
COUNT (CU:=CLOCK.CLK3, R:=reset, PV:=100 + value) ;
Pulse (IN:=COUNT.Q, PT:=t#1s) ;
```

Formal parameter

Current parameter

Note: In ST, unlike the graphic programming languages (FBD, LD), FB/DFB instances can be called multiple times.

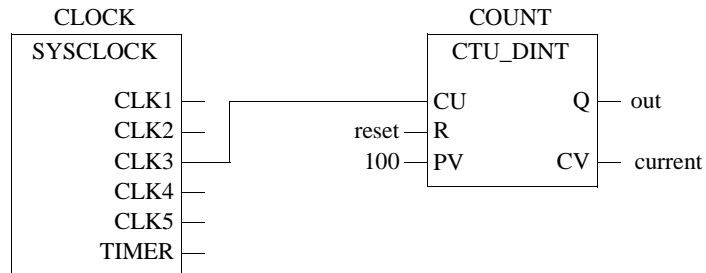
Note: Even if the function block has no inputs or the input parameters are not to be defined, the function block should be invoked before the outputs can be used. Otherwise the initial values for the outputs are given, i.e. "0".

Declaration and invocation of a function block in ST:

```
VAR
    CLOCK : SYSCLOCK ;
    COUNT : CTU_DINT ;
END_VAR

CLOCK ( ) ;
COUNT (CU:=CLOCK.CLK3, R:=reset, PV:=100) ;
out:=COUNT.Q ;
current:=COUNT.CV ;
```

Invocation of the function block in FBD.



**Function Block/
DFB Invocation
in Compact Form**

The block invocation and the assignments for the inputs/outputs are also possible in a more compact form, which saves runtime:

```
VAR
  CLOCK : SYSCLOCK ;
  COUNT : CTU_DINT ;
END_VAR

CLOCK ( ) ;
COUNT ( CU:=CLOCK.CLK3, R:=reset, PV:=100,
         Q=>out, CV=>current ) ;
```

**Use of the
Function Block/
DFB Outputs**

The outputs of the function block/DFBs can always be used when a variable (read only) can also be used.

```
Instance name      Formal parameter
out := COUNT.Q ;
current := COUNT.CV ;
Current parameter
```

Function Invocation

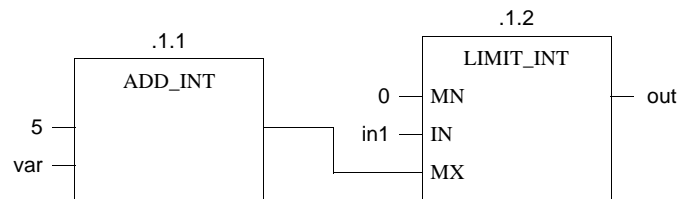
Using Functions Functions are provided by Concept in the form of libraries. The logic of the function is created in C++ and cannot be edited in the ST Editor. The names of the available function can be taken from the block libraries.

Note: The declaration of the function invocation can take place manually or you can create the block end and the assignment of the parameters using the menu command **Objects** → **Insert FFB**.

Invoking a function in ST:

```
out := LIMIT_INT (MN:=0, IN:=in1, MX:=5 + var) ;
```

Invoking the function FBD:



Unusable Functions

Functions which have one or more outputs with data type ANY but no inputs with data type ANY (generic outputs/inputs) cannot be used in ST.

Invoking a Function: Variant 1

The function can also be invoked using an instruction consisting of a current parameter (variable) followed by the instruction assignment "=" followed by the name of the function followed by a list of value assignments (current parameters) for the formal parameters in brackets. The order of the formal parameters in a function block invocation is not significant.

Current parameter (output) Formal parameter

```
out := LIMIT_INT (MN:=0, IN:=in1, MX:=5 + var) ;
```

Name of the function Current parameter (inputs)

**Invoking a
Function:
Variant 2**

Functions are invoked using an instruction. The instruction consists of the current parameter (variable) for the output followed by the instruction assignment "[:=" followed by the name of the function followed by a list of current input parameters in brackets. The order of the current parameters in a function invocation is significant.

Current parameter (output)

```
out := LIMIT_INT (0, in1, 5 + var) ;
```

Name of the function

Current parameter (inputs)

11.6 Syntax check and code generation

At a Glance

Overview This section describes the syntax check and the code generation of the structured ST text.

What's in this Section? This section contains the following topics:

Topic	Page
Syntax Check	380
Code generation	381

Syntax Check

Introduction A syntax check can be performed during the program/DFB creation with **Project** → **Analyze section**.

Syntax Check Options With the menu command **Options** → **Preferences** → **IEC Extensions...** → **IEC-Extensions** the syntax check options can be defined.

Note: The settings in this dialog are used in the project description (PRJ.DSK) and in the Concept installations description (CONCEPT.DSK), i.e. they are valid for the entire Concept installation.
If a project is opened, that was created using different settings (e.g. **Allow nested comments** in the project and not in the actual Concept Installation), errors can occur when opening the project.

Allow Case Insensitive Keywords If the check box **Allow case insensitive keywords** is checked, upper and lower case for all keywords is enabled.

Allow nested comments If the check box **Nested comments authorized** is checked, nested comments can be entered. There are no limits to the nesting depths.

Allow Leading Digits in Identifiers If the check box **Allow leading digits in identifiers** is checked, figures as the first character of identifiers (i.e. variable names, step names, EFB names) are possible. Identifiers, which consist solely of figures are, however, not authorized, they must contain at least one letter.

Unassigned Parameters Cause Warnings The IEC 1131-3 permits functions and Function Blocks to be called up without calling up the assignment of all the input parameters. These unused parameters are implicitly assigned a 0, or they retain the value from the last call up (Function Blocks only).

If in the menu command **Options** → **Preferences** → **Analysis...** → **Analysis** the check box **Unassigned parameters cause warnings** is checked, a list of these unused parameters is displayed in the message window when generating the code.

Code generation

At a Glance

The menu command **Project** → **Options for code generation** is used to define options for code generation.

Fastest code (restricted check)

If the check box **Fastest Code (restricted check)** is activated, a runtime-optimized code will be generated.

This runtime optimization is achieved with integer arithmetic (e.g. "+" or "-") with simple process commands instead of EFB calls.

Process commands are much faster than EFB calls, but they generate no error messages, such as e.g. Arithmetic or Array overrun. This option should only be used if it has been ensured that the program is free of arithmetic errors.

Example: Fastest code

```
IF i <= max THEN      (*i and max are of INT type*)
  i := i +1 ;
END_IF;
```

If **Fastest Code (restricted check)** is selected, the addition "i + 1" is executed with the process command "add". The code is now faster than if EFB ADD_INT had been called up. However, no runtime error is generated if "max" is 32767. In this case, "i" would overrun from 32767 to -32768!

Activate loop control

This check box activates a software watchdog for continuous loops.

If this check box is checked, with loops within IL and ST sections, it is tested whether these loops are again exited within a certain time. The time authorized depends on the manually defined watchdog time. The authorized time for **all loops** combined constitutes 80% of the Hardware watchdog time. In this way triggering of the hardware watchdog by endless loops is disabled. If a time consuming loop or an endless loop is detected, processing of the affected section will stop, an entry in the Event display will be generated and processing of the next section will begin. In the next cycle, the segment will be re-processed until a time consuming loop or an endless loop is detected once again, or until the segment is finished correctly.

Note: If the hardware watchdog stops the PLC when a time consuming loop or an endless loop is detected, this option should not be activated. The hardware watchdog itself is not switched off by this function.

11.7 Online functions of the ST programming language

Online functions

Description The online functions available in the programming language Instruction List (IL) are available here (see *Online functions of the IL instruction list* , p. 333).

11.8 Creating a program with the structured ST text

Creating a program in structured ST text

At a Glance

The following description contains an example of the creation of a program in the programming language of structured ST text. This creation is divided into 2 main steps:

Step	Action
1	Generating a section (See <i>Generating a section</i> , p. 385)
2	Creating the logic (See <i>Creating the logic</i> , p. 386)

Generating a section

The procedure for generating a section is as follows:

Step	Action
1	<p>Using the menu command File → New section... and enter a section name.</p> <p>Note: The section name (max. 32 characters) is not case-sensitive and must be unique throughout the project. If the name entered already exists, a warning is given and another name must be chosen. The section name must correspond to the IEC name conventions, otherwise an error message is displayed.</p> <p>Note: In accordance with IEC1131-3, only letters are permitted as the first character of names. Should numbers be required as the first character, however, the menu command Options → Preferences → IEC Extensions... → Allow leading digits in identifiers.</p>

Creating the logic

The procedure for creating the logic is as follows:

Step	Action
1	<p>Declare the Function Block and DFBs, which are to be used, with assistance from VAR...END_VAR.</p> <p>Example:</p> <pre>VAR RAMP_UP, RAMP_DOWN, RAMP_X : TON COUNT : CTU_DINT ; END_VAR</pre>
2	<p>Declare the variables and their initial value in the Variable Editor.</p>
3	<p>Create the logic of the program.</p> <p>Example:</p> <pre>SUM := 0 ; FOR I = 1 TO 3 DO FOR J = 1 TO 2 DO IF FLAG = 1 THEN EXIT; END_IF; SUM := SUM + J ; END_FOR ; SUM = SUM + I ; END_FOR</pre>
4	<p>Save the section with the menu command Data file → Save project .</p>

Index



A

- Access Rights, 701, 694, 702
- Action variable, 241
- Actions, 240
 - Process, 259
- Activate dialogs, 92
- Actual parameters
 - FBD, 182
 - LD, 215
- Alias designations
 - Step, 266
 - Transition, 266
- Alternative branch, 248
- Alternative connection, 250
- Animation, 539, 679, 682
 - FBD, 193
 - General information, 606
 - IEC section, 607
 - IL, 337
 - IL/ST, 334
 - LD, 226
 - LL984 section, 609
 - Section, 606
 - SFC, 270, 272
- ANY Outputs, 373
- Archiving
 - DFB, 674
 - EFB, 674
 - Project, 674
- ARRAY
 - Range Monitoring, 527
- ASCII message editor, 543, 545, 550
 - Combination mode, 559, 560
 - Control code, 549
 - Direct mode, 559, 560
 - Flush (buffer), 551
 - Generals, 546
 - How to continue after getting a warning, 557, 558
 - How to Use, 554
 - Message Number, 555
 - Message text, 556
 - Offline mode, 559, 560
 - Repeat, 552
 - Simulation text, 556
 - Spaces, 549
 - Text, 547
 - User interface, 553, 554
 - Variables, 548
- ASCII messages, 56, 91
- Assign instructions
 - ST, 357
- Assignment
 - =>, 376
- Atrium
 - Memory optimization, 163
- Atrium example
 - INTERBUS controller, 877
- Atrium first startup
 - DOS Loader, 1014
 - EXECLoader, 996
 - Modbus Plus, 996, 1014
- Auto-Log-Out, 112

Automatic Connection, 1068, 1071
 Available functions in OFFLINE and ONLINE modes, 76

B

Backplane Expander
 Configure, 98
 Edit I/O Map, 99
 Error handling, 100
 Generals, 99
 Block call up
 IL, 320
 ST, 372

C

Call
 DFB, 319
 FFB, 319
 FFB, 327
 Project, 744
 Chain jump, 246
 Chain loop, 247
 Change
 Coil, LD, 220
 Contact, LD, 220
 FFB, FBD, 187
 FFB, LD, 220
 Changing signal states of a Located variable
 Reference data editor, 535
 Close Column
 LL984, 398
 Closer
 LD, 205
 Code generation
 FBD, 191
 IL, 332
 ST, 381
 LD, 224
 Coil
 Change, LD, 220
 LD, 206
 Replace, LD, 220
 Coil - negated
 LD, 207

Coil – negative edge
 LD, 208
 Coil – positive edge
 LD, 207
 Coil - reset
 LD, 208
 Coil - set
 LD, 208
 Cold restart, 37
 Comments
 Data type editor, 518
 Derived data type, 518
 Communication, 14
 Compact
 Memory optimization, 147
 Compact configuration
 RTU extension, 105
 Compact example, 871
 Compact first startup
 DOS Loader, 977, 1011
 EXECLoader, 958, 992
 Modbus, 958, 977
 Modbus Plus, 992, 1011
 Concept DFB, 415, 455
 Concept ModConnect, 915
 Integrating new Modules, 919
 Removing modules, 920
 Use of Third Party Modules in Concept, 922
 Concept PLCSIM32, 682
 Concept Security, 691, 692, 694, 701, 702
 Concept SIM, 679
 CONCEPT.INI, 1027, 1029
 General, 1030
 LD section settings, 1035
 Path for Global DFBs, 1033
 Path for Help Files, 1033
 Path for MBPPATH.INI, 1033
 print settings, 1031
 Project Name Definition, 1032
 Reading Global DFBs, 1033
 Register Address Format Settings, 1032
 Representation of internal data, 1035
 Security Settings, 1037
 Setting for Online Processing, 1036
 Setting for the Address Format, 1036

- Settings for Warning Messages, 1036
 - Storage of Global DFBs during Upload, 1033
 - Variable Storage Settings, 1032
 - Configuration, 51, 69
 - Backplane Expander Config, 98
 - Ethernet, 104
 - Ethernet I/O Scanner, 106
 - General information, 71
 - INTERBUS, 102
 - Network systems, 92
 - Optional, 90
 - OFFLINE and ONLINE mode, 74
 - Profibus DP, 103
 - RTU extension, 105
 - Unconditional, 78
 - Various network systems, 101
 - Configuration example
 - Atrium-INTERBUS controller, 877
 - Compact controller, 871
 - Momentum-Ethernet bus system, 895
 - Momentum-Remote I/O bus, 887
 - Quantum-INTERBUS control, 835
 - Quantum-Peer Cop, 863
 - Quantum-Profibus DP controller, 849
 - Quantum-Remote control with DIO, 826
 - Quantum-Remote control with RIO, 807
 - Quantum-Remote control with RIO (series 800), 815
 - Quantum-SY/MAX controller, 841
 - Configuration extensions, 92
 - Connect
 - PLC, 565
 - Automatically with command line parameters, 1068
 - Automatically with the CCLaunch Tool, 1071
 - Connect to IEC Simulator (32-bit), 578
 - Constant Scan, 582
 - Constants, 35
 - Contact
 - Change, LD, 220
 - LD, 205, 206
 - Replace, LD, 220
 - Context help, 746
 - Convert
 - DFBs, 911
 - Macros, 911
 - Modsoft programs, 923
 - Projects, 911
 - RDE templates, 533
 - CPU selection for the PLC type, 80
 - Create
 - DFB, 436
 - FFB, FBD, 186
 - FFB, LD, 219
 - Macro, 467
 - Program, 47
 - Project, 47
 - Project Symbol, 744
 - Creating a program
 - IL, 339
 - Cyclical Setting of Variables
 - Reference Data Editor, 536
- ## D
- Data exchange between nodes on the Modbus Plus network, 93
 - Data flow, 221
 - FBD, 189
 - Data protection, 55
 - Data protection in the state RAM, 94
 - Data Type Definition
 - Extended (larger than 64 Kbytes), 507
 - Data type editor, 499, 501
 - Comments, 518
 - Elements, 510
 - Key words, 512
 - Names, 516
 - Separators, 517
 - Short Cut Keys, 765
 - Syntax, 509
 - Use of memory, 521
 - DDT, 507
 - Declaration of variables, 480
 - Declare
 - Actions, 259
 - Step properties, 257
 - Transition, 264

- Defining Colors
 - INI File, 1036
- Defining the LD contact connection
 - Settings in the INI file, 1035
- Defining the number of LD columns/fields
 - Settings in the INI file, 1035
- Delete
 - DFB, 676
 - Macro, 676
 - Memory zones from the PLC, 584
 - PLC contents, 584
 - Project, 676
- Derived Data Type, 499, 501
 - Comments, 518
 - Elements, 510
 - Export, 629
 - Global, 505
 - Key words, 512
 - Local, 505
 - Names, 516
 - Separators, 517
 - Syntax, 509
 - Use of memory, 521
- Derived Data Types
 - Use, 524
- Derived Function Block, 418
 - FBD, 180
 - LD, 211
- DFB, 415, 418
 - Archiving, 674
 - Call, 319
 - Convert, 911
 - Context sensitive help, 434
 - Create, 436
 - Creating Global Variables, 430
 - Delete, 676
 - Documentation, 663
 - FBD, 180
 - Global, 420
 - Invocation, 321, 373
 - LD, 211
 - Local, 420
 - Protect, 702
- Diagnosis
 - Transition diagnosis, 277
- Diagnosis viewer, 610
- Dialog boxes, 740
- Dialog interaction
 - LL984, 394
- Direct Addresses, 35
- Disable
 - Interrupt Sections, 42
 - Section, 42
- Document section options, 667
- Documentation
 - Contents, 664
 - DFB, 663
 - Keywords, 671
 - Layout, 665
 - Macro, 663
 - Project, 663
- DOS Loader
 - Atrium first startup, 1014
 - Compact first startup, 977, 1011
 - Momentum first startup, 980, 983, 1017, 1020
 - Quantum first startup, 974, 1008
 - Startup when using Modbus, 973
 - Startup when using Modbus Plus, 1007
- Download Changes, 600
- Driver for 16 bit application capability with Windows 98/2000/NT
 - Virtual MBX Driver, 940
- Driver for connection between ModConnect Host interface adapters and 32 bit applications with Windows 98/2000/NT
 - MBX-Treiber, 941
- Driver for Modbus Plus Function via TCP/IP
 - Ethernet MBX Driver, 943
- Driver for Remote Operation
 - Remote MBX Driver, 942
- DTY, 499, 501, 502
- DX Zoom
 - LL984, 400

E

- Edit
 - Actions, 259
 - LL984, 393, 397
 - SFC, 253
 - Step properties, 257
 - Transition, 264
- Edit I/O Map
 - Backplane Expander, 99
- Editing local Drop, 808
- Editing Networks
 - LL984, 398
- Editors, 9
- EFB
 - Archiving, 674
 - FBD, 178
 - LD, 209
- EFBs for Interrupt Sections, 1065
- Elementary Function
 - FBD, 178
 - LD, 209
- Elements
 - Data type editor, 510
 - Derived Data Type, 510
- EN
 - FBD, 181
 - LD, 213
- ENC File, 15, 613, 614
- Encrypt Logfile, 15, 693
 - ENC File, 614
- ENO
 - FBD, 181
 - LD, 213
- EQUAL, 566
- Equation network
 - LL984, 404, 405
- Equation network, Syntax and Semantics
 - LL984, 409
- Error handling
 - Backplane Expander, 100
- Establishing the hardware connection
 - Modbus Plus presettings, 945
 - Modbus presettings, 950
- Ethernet, 578
 - Ethernet / I/O Scanner
 - Configurator, 106
 - How to use the Ethernet / I/O Scanner, 109
 - Ethernet Bus System
 - Create online connection, 910
 - Momentum, 896
 - Ethernet MBX Driver
 - Driver for Modbus Plus Function via TCP/IP, 943
 - Ethernet with Momentum, 105
 - Ethernet with Quantum, 104
 - Event Viewer
 - INI Settings, 1039
 - Example of hardware configuration
 - Atrium-INTERBUS controller, 877
 - Compact controller, 871
 - Momentum-Ethernet bus system, 895
 - Momentum-Remote I/O bus, 887
 - Quantum-INTERBUS control, 835
 - Quantum-Peer Cop, 863
 - Quantum-Profibus DP controller, 849
 - Quantum-Remote control with DIO, 826
 - Quantum-Remote control with RIO, 807
 - Quantum-Remote control with RIO (Series 800), 815
 - Quantum-SY/MAX controller, 841
 - Exchange Marking
 - Macro, 462
 - EXEC file, 1023
 - CPU 424 02, 125
 - CPU X13 0X, 125
 - Momentum, 160
 - EXECLoader
 - Atrium first startup, 996
 - Compact first startup, 958, 992
 - Momentum first startup, 962, 967, 999, 1003
 - Quantum first startup, 954, 988
 - Startup when using Modbus, 953
 - Startup when using Modbus Plus, 987
 - Execution Order
 - FBD, 187
 - Section, 41
 - Timer Event Sections, 1051

-
- Execution sequence
 - LD, 221
 - Export, 619
 - Derived Data Type, 629
 - General Information, 622
 - PLC Configuration, 658
 - Section, 625
 - Variable, 629
 - Exporting located variables, 490
 - Expressions
 - ST, 345
 - Extended memory, 129
- F**
- Factory Link, 656
 - FBD, 173
 - Actual parameters, 182
 - Animation, 193
 - Calling a macro, 476
 - Code generation, 191
 - Data flow, 187, 189
 - Derived Function Blocks, 180
 - DFB, 180
 - EFB, 178
 - Elementary Function, 178
 - Elementary Function Block, 179
 - EN, 181
 - ENO, 181
 - Execution order, 187
 - FFB, 178
 - Function, 178
 - Function Block, 179
 - Icon bar, 755
 - Link, 182
 - Loop, 189
 - Online Functions, 193
 - Program creation, 196
 - Short Cut Keys, 768
 - Text Object, 184
 - UDEFB, 181
 - User-defined Elementary Function, 181
- FFB**
- Call, 319, 327
 - Change, FBD, 187
 - Change, LD, 220
 - Create, FBD, 186
 - Create, LD, 219
 - FBD, 178
 - Insert, FBD, 186
 - Insert, LD, 219
 - Invocation, 321, 373, 377
 - LD, 209
 - Position, 186, 219
 - Replace, FBD, 187
 - Replace, LD, 220
- Function**
- FBD, 178
 - LD, 209
- Function Block**
- FBD, 179
 - LD, 210
- Function Block language, 173**
- Function Blocks for Interrupt Sections, 1065**
- G**
- General, 1
 - Backplane Expander, 99
 - Hardware configuration, 71
 - Loading a project, 598
 - Online control panel, 581
 - Online functions, 564
 - OFFLINE and ONLINE mode, 75
 - PLC configuration, 72
 - PLC Connection, 566
 - Reference Data Editor, 532
 - Select process information, 593
 - Variables editor, 480
 - Generate
 - Project symbol, 743
 - Global data transfer
 - Peer Cop, 867
 - Global derived data type, 505
-

- Global DFB, 420
 - Defining the Path, 1033
 - INI File, 1033
 - Reading, 1034
 - Storing, 1034
 - Global macro, 460
 - Global Variables in DFBs, 430
- H**
- Hardware
 - Performance, 707
 - Head setup, 53
 - Help, 746
 - Help Files
 - Defining the Path, 1033
 - How to use the Ethernet / I/O Scanner
 - Ethernet / I/O Scanner, 109
- I**
- I/O Event Sections, 1060
 - Handling, 1041
 - Priority, 1061
 - Runtime Error, 1062
 - I/O map, 52, 87
 - Icon bar, 753, 754, 755, 756, 758
 - Icons, 751, 753, 754, 755, 756, 758, 759, 760, 762
 - Icons_Project Browser, 762
 - Identifier, 262
 - IEC
 - Hot Standby data, 83
 - Momentum first startup, 962, 999, 1017
 - IEC conformity, 779
 - IEC section
 - Animation, 607
 - IL, 279
 - Animation, 334, 337
 - Block call up, 320
 - Code generation, 332
 - Creating a program, 339
 - Instruction, 283, 284
 - List of Symbols, 759
 - Modifier, 287
 - Online functions, 333, 334
 - Operands, 285
 - Operators, 288, 295
 - Short Cut Keys, 765
 - Syntax check, 330
 - Tag, 291
 - IL command
 - call function block, 321
 - Compare, 310, 311, 314
 - Comments, 294
 - Compare, 312, 313, 315
 - Declaration, 292
 - DFB invocation, 321
 - Function call, 327
 - Invert, 305
 - Reset, 299
 - Set, 298
 - VAR...END_VAR, 292
 - IL operation
 - addition, 306
 - Boolean AND, 300
 - Boolean exclusive OR, 304
 - Boolean OR, 302
 - Call DFB, 319
 - Call function block, 319
 - Division, 309
 - Jump to label, 316
 - Load, 296
 - Multiplication, 308
 - Store, 297
 - Subtraction, 307
 - Import, 619
 - General Information, 622
 - INTERBUS configuration, 884
 - PLC Configuration, 658
 - Profibus DP configuration, 856
 - Section, 630, 635, 645, 646, 647
 - Structured variables, 653
 - Variables, 649, 653, 656
 - INC
 - Include File, 507
 - Include File
 - Extended Data Type Definition, 507

-
- INI File, 1027
 - CONCEPT.INI, 1029
 - Event Viewer Settings, 1039
 - General Information, 1030, 1039
 - LD section settings, 1035
 - Path for Global DFBs, 1033
 - Path for Help Files, 1033
 - Path for MBPPATH.INI, 1033
 - Print settings, 1031
 - Project Name Definition, 1032
 - Projectname.INI, 1038
 - Project Specific, 1027
 - Reading Global DFBs, 1033
 - Register Address Format Settings, 1032
 - Representation of internal data, 1035
 - Security Settings, 1037
 - Setting for Online Processing, 1036
 - Settings for the Address Format, 1036
 - Settings for Warning Messages, 1036
 - Storage of Global DFBs during Upload, 1033
 - Variable Storage Settings, 1032
 - Initial step, 238
 - Insert
 - FFB, FBD, 186
 - FFB, LD, 219
 - Install
 - Loadables, 52
 - EXEC file, 1024
 - Modbus Plus driver
 - Windows 98/2000/NT, 939
 - Install the SA85/PCI85
 - Modbus Plus Preferences, 934
 - Windows NT, 937
 - Windows 98/2000/XP, 934
 - Instruction
 - IL, 283, 284
 - ST, 358
 - Instruction list, 279
 - INTERBUS controller, 836
 - INTERBUS Controller with Atrium, 878
 - INTERBUS export settings in CMD, 879
 - Interface Settings in Windows 98/2000/XP
 - Modbus Preferences, 948
 - Interface settings in Windows NT
 - Modbus Preferences, 950
 - Interrupt Processing, 1041
 - General, 1044
 - Interrupt Sections
 - Disable, 42
 - EFBs, 1065
 - Examples for Setting Parameters, 1054
 - Execution Order, 1051
 - I/O Event Sections, 1060
 - Operating System, 1052
 - Priority, 1061
 - Runtime Error, 1062
 - Scan Rate for Timer Event Sections, 1048
 - Timer Event Sections, 1047, 1049
 - Invocation
 - DFB, 321, 373
 - FFB, 321, 373, 377
 - Project, 743
- J**
- Jump
 - SFC, 246
- K**
- Key combinations, 751, 763, 764, 765, 768, 771, 777
 - Key words
 - data type editor, 512
 - derived data type, 512
 - Keys, 751, 763, 764, 765, 768, 771, 777
- L**
- Ladder Diagram, 199
 - Ladder Logic 984, 387
 - LD, 199
 - Actual parameters, 215
 - Animation, 226
 - Calling a macro, 476
 - Closer, 205
 - Code generation, 224
 - Coil - negated, 207
 - Coil - negative edge, 208

- Coil – positive edge, 207
- Coil - reset, 208
- Coil - set, 208
- Coils, 206
- Contacts, 205, 206
- Data flow, 221
- Derived function block, 211
- EFB, 209
- Elementary function, 209
- EN, 213
- ENO, 213
- Execution sequence, 221
- FFB, 209
- Function, 209
- Function block, 210
- Icon bar, 758
- Link, 214
- Loops, 221
- Online functions, 226
- Opener, 205
- Program creation, 229
- Shortcut keys, 771
- Text object, 217
- UDEFB, 212
- User-defined elementary function, 212
- Learn monitoring times
 - SFC, 274
- Libraries, 8
- Limitations
 - LL984, 391
- Link
 - FBD, 182
 - LD, 214
- List of Symbols, 751, 759, 760
- List of Tools, 751, 759, 760
- Literals, 35
- LL984, 387
 - Close Column, 398
 - Combination mode, 414
 - Dialog interaction, 394
 - Direct programming, 414
 - DX Zoom, 400
 - Edit, 393, 397
 - Editing Networks, 398
 - Equation network, 404, 405
 - Equation network, Syntax and Semantics, 409
- List of Symbols, 760
- Momentum first startup, 967, 983, 1003, 1020
- Navigation, 393
- Online Restriction, 394
- Online Search, 401
- Open Column, 398
- Open Row, 398
- Programming modes, 413, 414
- Reference Offset, 396
- Reference Zoom, 399
- References, 395
- Replace References, 401
- Requirements, 393
- Section, 390
- Segment, 390
- Select, 397
- Short Cut Keys, 777
- Subroutines, 402
- Trace, 401
- Undo, 397
- Variables, 395
- LL984 Processing
 - speed optimized, 585
- LL984 section
 - Animation, 609
- Load reference data, 542
- Loadables, 84
 - Atrium, 166
 - compact, 150
 - CPU 424 02, 131
 - CPU X13 0X, 131
 - CPU 434 12, 139
 - CPU 534 14, 139
- Loading, 599
- Loading a project, 597
 - General information, 598
- Loading firmware, 1024
- Local derived data type, 505
- Local DFB, 420
- Local macro, 460
- Located variables
 - Changing signal states in RDE, 535
- Log Encrypting, 15
- LOG File, 613, 614

- Logging
 - LOG File, 614
- Logging Write Access to the PLC, 613
- Loop
 - FBD, 189
 - LD, 221

M

- Macro, 455, 458
 - Calls from SFC, 473
 - Calls from FBD, 476
 - Calls from LD, 476
 - Context sensitive help, 465
 - Convert, 911
 - Create, 467
 - Delete, 676
 - Documentation, 663
 - Exchange marking, 462
 - Global, 460
 - Local, 460
- Maximum supervision time, 238
- MBPPATH.INI
 - Defining the Path, 1033
- MBX Driver
 - Driver for connection between ModConnect Host interface adapters and 32 bit applications with Windows 98/2000/NT, 941
- Memory, 115
 - Optimize, 119
 - Structure, 117
- Memory and optimization
 - Atrium, 163
 - Compact, 147
 - Momentum, 157
 - Quantum, 122, 136
- Memory partitions, 51
- Memory statistics, 595
- Menu commands, 737
- Minimum configuration, 51
- Minimum supervision time, 239
- MMS-Ethernet
 - Specify coupling modules, 92

- Modbus
 - Compact first startup, 958, 977
 - Momentum first startup, 962, 967, 980, 983
 - Quantum first startup, 954, 974
 - Startup with DOS Loader, 973
 - Startup with the EXECLoader, 953
- Modbus communication, 53
- Modbus network link, 570
- Modbus Plus
 - Atrium first startup, 996, 1014
 - Compact first startup, 992, 1011
 - Momentum first startup, 999, 1003, 1017, 1020
 - Quantum first startup, 988, 1008
 - Remote MBX Driver, 942
 - Startup with DOS Loader, 1007
 - Startup with the EXECLoader, 987
 - Virtual MBX Driver, 940
 - Write Restriction, 112
- Modbus Plus Bridge, 576
- Modbus Plus Network Connection, 571
- Modbus Plus network node, 93
- Modbus Plus Preferences
 - Installing the SA85/PCI85, 934
 - Installing the Modbus Plus driver in Windows 98/2000/NT, 939
 - Establishing the hardware connection, 945
 - Startup, 933
- Modbus Plus Routing Path
 - Automatic Connection, 1068, 1071
- Modbus Preferences
 - Interface Settings in Windows 98/2000/XP, 948
 - Interface Settings in Windows NT, 950
 - Transfer problems, 951
 - Establishing the hardware connection, 950
 - Startup, 947
- ModConnect, 915
- MODIFIED, 566
- Modifier
 - IL, 287

Modsoft
 Conversion, 923
 Function compatibility, 932
 References, 929
 Momentum
 Memory optimization, 157
 Momentum example
 Ethernet bus system, 895
 Remote I/O bus, 887
 Momentum first startup
 DOS Loader, 980, 983, 1017, 1020
 EXECLoader, 962, 967, 999, 1003
 Modbus, 962, 967, 980, 983
 Modbus Plus, 999, 1003, 1017, 1020
 MSTR-Read-Operation, 113

N

Names
 Datatype editor, 516
 Derived datatype, 516
 Navigation
 LL984, 393
 Network Configuration
 TCP/IP, 897
 Network Connection
 Modbus Plus, 571
 Network link
 Modbus, 570
 TCP/IP, 578
 NOM/NOE
 Disable Write Access, 112
 NOT EQUAL, 566

O

Objects
 Insert, LD, 219
 SFC, 237
 Offline functions in the configurator, 76
 Online, 679, 682
 INI File, 1036
 SFC, 269
 Online Control Panel, 581, 585, 589
 Online diagnosis, 610

Online functionen, 14, 561
 Configurator, 76
 General information, 563, 564
 FBD, 193
 IL, 333
 IL/ST, 334
 LD, 226
 ST, 382
 SFC, 270, 272
 Online help, 746
 ONLINE Operation
 Presettings, 569
 Online Restriction
 LL984, 394
 Online Search
 LL984, 401
 Open
 Project, 743, 744
 Open Column
 LL984, 398
 Open Row
 LL984, 398
 Opener
 LD, 205
 Operands
 IL, 285
 ST, 346
 Operating System
 Timer Event Sections, 1052
 Operators
 IL, 288, 295
 ST, 347
 Optimize
 PLC Memory, 119
 Optional Configuration, 90

P

Page breaks for sections, 667
 Parallel branch, 251
 Parallel connection, 252
 Parameterize ASCII interface, 94
 Parameterize interfaces
 ASCII interface, 94
 Modbus interface, 94
 Parameterize Modbus interface, 94

-
- Parameters for Automatic Connection, 744
 - Password protection, 691
 - Path for Global DFBs
 - Settings in the INI File, 1033
 - Path for Help Files
 - Settings in the INI File, 1033
 - Peer Cop, 93, 864
 - Peer Cop communication, 54
 - Performance
 - hardware, 707
 - PLC family, 707
 - Phase
 - Timer Event Sections, 1049
 - PLC
 - Simulating, 677
 - Status, 733
 - PLC configuration, 50, 51, 69
 - Export, 658
 - General information, 72
 - Icons, 761
 - Import, 658
 - PLC Connection
 - General, 566
 - PLC family
 - Performance, 707
 - PLC Memory, 115
 - Optimize, 119
 - Structure, 117
 - PLC Memory and optimization
 - Atrium, 163
 - Compact, 147
 - Momentum, 157
 - Quantum, 122, 136
 - PLC memory mapping, 83
 - PLC selection, 80
 - PLC State, 566, 579, 594
 - Position
 - FFB, FBD, 186
 - FFB, LD, 219
 - Precondition for unconditional configuration, 79
 - Presettings for Modbus
 - Startup, 947
 - Presettings for Modbus Plus
 - Startup, 933
 - Presettings for ONLINE operation, 569
 - Print
 - Settings in the INI file, 1031
 - Sections, 667
 - Priority
 - I/O Event Sections, 1061
 - Proceed in the following way with the configuration, 73
 - Process
 - Actions, 259
 - Program, 30
 - Project, 30
 - Step properties, 257
 - Transition, 264
 - PROFIBUS
 - Specify coupling modules, 92
 - Profibus DP controller, 850
 - Profibus DP export settings in SyCon, 850
 - Program
 - Create, 47
 - Processing, 30
 - Status, 733
 - Structure, 29, 30
 - Program creation
 - FBD, 196
 - LD, 229
 - ST, 385
 - Programming, 6
 - Programming languages, 9
 - Programming modes
 - LL984, 413, 414
 - Programs, 35
 - Project
 - Archiving, 674
 - Call, 744
 - Convert, 911
 - Create, 47
 - Delete, 676
 - Documentation, 663
 - Invoke, 743
 - Open, 743, 744
 - Processing, 30
 - Protect, 702
 - Structure, 29, 30

- Project Browser, 491
 - Keyboard operation, 496
 - Mouse operation, 496
 - Toolbar, 762
 - Project Name Definition
 - INI File Settings, 1032
 - Project Symbol
 - Generate, 743
 - Create, 744
 - Projectname.INI, 1027, 1038
 - Event Viewer Settings, 1039
 - General Information, 1039
 - Protect
 - DFB, 702
 - Project, 702
- Q**
- Quantum
 - Memory optimization, 122, 136
 - Quantum example
 - INTERBUS control, 835
 - Profibus DP controller, 849
 - Quantum-Peer Cop, 863
 - Remote control with DIO, 826
 - Remote control with RIO, 807
 - Remote control with RIO (series 800), 815
 - SY/MAX controller, 841
 - Quantum first startup
 - DOS Loader, 974, 1008
 - EXECLoader, 954, 988
 - Modbus, 954, 974
 - Modbus Plus, 988, 1008
 - Quantum Security Parameters, 112
- R**
- Range Monitoring
 - ARRAY, 527
 - RDE, 531
 - Converting RDE templates, 533
 - Cyclical Setting of Variables, 536
 - General, 532
 - Toolbar, 762
 - Reactivate flash save, 588
 - Reading Global DFBs
 - Settings in the INI File, 1033
 - Reference data editor, 531
 - Changing signal states of a Located variable, 535
 - Converting RDE templates, 533
 - Cyclical Setting of Variables, 536
 - General, 532
 - Replacing variable names, 541
 - Reference Offset
 - LL984, 396
 - Reference Zoom
 - LL984, 399
 - References
 - LL984, 395
 - Register Address Format
 - INI File Settings, 1032
 - Remote controller with DIO, 831
 - Remote controller with RIO, 812
 - Remote controller with RIO (series 800), 820
 - Remote MBX Driver
 - Modbus Plus, 942
 - Replace
 - coil, LD, 220
 - contact, LD, 220
 - FFB, FBD, 187
 - FFB, LD, 220
 - Variable names, 541
 - Replace References
 - LL984, 401
 - Requirements
 - LL984, 393
 - RTU extension
 - Compact configuration, 105
 - Configure, 105
 - Runtime Error
 - I/O Event Sections, 1062
- S**
- Save To Flash, 585
 - Scan
 - Constant, 582
 - Scan rate
 - Timer Event Sections, 1048

-
- Scan times
 - single, 583
 - Search and Replace
 - Variable names and addresses, 483
 - Search and paste
 - Variable names and addresses, 487
 - Section, 40
 - Animation, 606
 - Disable, 42
 - Execution order, 41
 - Export, 625
 - Import, 630, 635, 645, 646, 647
 - Import, 631, 642
 - LL984, 390
 - Status, 733
 - Secure Application, 15
 - Security, 691, 692, 694, 701, 702
 - Segment
 - LL984, 390
 - Segment manager, 86
 - Select
 - LL984, 397
 - Select process information
 - General information, 593
 - Status and memory, 592
 - Selecting process information
 - Status and memory, 592
 - Separators
 - Data type editor, 517
 - Derived data type, 517
 - Set/Change PLC Password, 589
 - Setting up and controlling the PLC, 580
 - Setup and control PLC
 - General information, 581
 - SFC
 - 'SFCSTEP_STATE' variable, 240
 - 'SFCSTEP_TIMES' variable, 239
 - Action, 240, 259
 - Action variable, 241
 - Alternative branch, 248
 - Alternative connection, 250
 - Animation, 270, 272
 - Calling up macros, 473
 - Edit, 253
 - Icon bar, 756
 - Identifier, 262
 - Initial step, 238
 - Jump, 246
 - Learn monitoring times, 274
 - Link, 245
 - Maximum supervision time, 238
 - Minimum supervision time, 239
 - Objects, 237
 - Online, 269
 - Online functions, 270, 272
 - Parallel branch, 251
 - Parallel connection, 252
 - Short Cut Keys, 768
 - Step, 238
 - Step delay time, 238
 - Step duration, 238
 - Step properties, 257
 - String, 272
 - Text object, 252
 - Transition, 242, 264
 - Transition diagnosis, 277
 - Transition section, 243
 - Transition variable, 244
 - Waiting step, 238
 - Short cut keys, 751, 763
 - Simple sequences, 245
 - Simulation, 677, 679, 682
 - SPS, 679, 682
 - Single sweeps, 583
 - Special options, 96
 - Specific data transfer
 - Peer Cop, 869
 - Speed optimized LL984- Processing, 585
 - SPS
 - Simulate, 679, 682
 - ST, 341
 - Animation, 334
 - Assign instructions, 357
 - Block call up, 372
 - Code generation, 381
 - Expressions, 345
 - Instructions, 358
 - List of Symbols, 759
 - Online functions, 334, 382
 - Operands, 346
 - Operators, 350
 - operators, 347

- Program creation, 385
- Short Cut Keys, 765
- syntax check, 380
- ST Command
 - , 352, 354
 - (), 351
 - , 352
 - *, 352
 - ** , 351
 - +, 353
 - >, 354
 - >=, 354
 - Addition, 353
 - , 355, 355
 - &, 356
 - =, 354
 - /, 353
 - AND, 356
 - Assignment, 359
 - Boolean AND, 356
 - Boolean Exclusive OR, 356
 - Boolean OR, 356
 - Call function block, 373
 - CASE...OF...END_CASE, 365
 - Complement formation, 352
 - Declaration, 360
 - Division, 353
 - ELSE, 363
 - ELSIF...THEN, 364
 - Empty instruction, 371
 - Equal to, 354
 - EXIT, 371
 - Exponentiation, 351
 - FOR...TO...BY...DO...END_FOR, 366
 - FUNCNAME, 351
 - function invocation, 377
 - Greater than, 354
 - Greater than/Equal to, 354
 - IF...THEN...END_IF, 362
 - Less than, 355
 - Less than or equal to, 355
 - MOD, 353
 - Modulo, 353
 - Multiplication, 352
 - Negation, 352
 - NOT, 352
 - Not equal to, 355
 - OR, 356
 - REPEAT...UNTIL...END_REPEAT, 370
 - Subtraction, 354
 - Use of parentheses, 351
 - VAR...END_VAR, 360
 - WHILE...DO...END_WHILE, 368
 - XOR, 356
- ST Comment
 - Comment, 371
- Start behavior
 - Variable, 37
 - Digital outputs, 39
- Startup
 - Presettings for Modbus, 947
 - Presettings for Modbus Plus, 933
- Startup with DOS Loader
 - Modbus, 973
 - Modbus Plus, 1007
- Startup with the EXECLoader
 - Modbus, 953
 - Modbus Plus, 987
- State of the PLC, 579
- Status, 566
- Status bar, 733
- Step, 238
 - Alias designations, 266
- Step delay time, 238
- Step duration, 238
- Step properties
 - Process, 257
- Storage of Global DFBs during Upload
 - Settings in the INI File, 1033
- String
 - Control, 272
- Structure
 - PLC Memory, 117
 - Program, 29, 30
 - Project, 29, 30
- Structured text, 341
- Structured variables
 - Import, 653
- Subroutines
 - LL984, 402
- Symax-Ethernet
 - specify coupling modules, 92
- Symbols, 751, 759, 760

Syntax

- Data type editor, 509
- Derived Data Type, 509

Syntax check

- IL, 330
- ST, 380

T

Tag

- IL, 291

TCP/IP

- Network Configuration, 897
- Network link, 578

TCP/IP-Ethernet

- specify coupling modules, 92

Text Object

- FBD, 184
- LD, 217
- SFC, 252

Timer Event Sections, 1047

- Define Scan Rate, 1048
- Defining the Phase, 1049
- Examples for Parameterization, 1054
- Execution Order, 1051
- Handling, 1041
- Operating System, 1052

Toolbar, 753, 754, 755, 756, 758

Tools, 17

Trace

- LL984, 401

Transfer problems

- Modbus Presettings, 951

Transition, 242

- Alias designations, 266
- Declare, 264
- Process, 264

Transition diagnosis, 277

Transition section, 243

Transition variable, 244

U

UDEFB

- FBD, 181
- LD, 212

Unconditional Configuration, 78

- Precondition, 79

Unconditional locking of a section, 538

Undo

- LL984, 397

Uploading PLC, 603

User-defined Elementary Function

- FBD, 181
- LD, 212

Utility program, 17

V

Variable Editor, 479

- Declaration, 480
- Exporting located variables, 490
- Search and replace, 483
- Search and paste, 487

Variable Storage

- INI File Settings, 1032

Variables, 35

- ASCII message editor, 548
- Export, 629
- Import, 649, 653, 656
- LL984, 395
- Start behavior, 37

VARINOUT variables, 423

Various PLC settings, 56

View Tool, 614

Virtual MBX Driver

- Modbus Plus, 940

W

Waiting step, 238

Warm restart, 37

Window elements, 733

Window types, 732

- Windows, 729
 - Check box, 742
 - Command buttons, 741
 - Dialog boxes, 740
 - Lists, 741
 - Menu commands, 737
 - Option buttons, 741
 - Status bar, 733
 - Text boxes, 741
 - Window, 731
 - Window elements, 733
 - Window types, 732

