SIGMATEK

LASAL CLASS 2

Operating Manual

Date of Creation: 17.05.2005 | Current Version: 18.01.2024 | Article No.: 08-001-012-E

Schnipsel Impressum.xml nicht gefunden!

Copyright © 2005
SIGMATEK GmbH & Co KG

Translation from German

Schnipsel Rechte.xml nicht gefunden!

Table of Contents

INTRODUCTION 49

All in One **50**

LASAL supports several target platforms **51**

Real time with a multilevel, multi-tasking system **52**

Open data Interface **53**

Worldwide service interface **54**

Online help system **55**

Multilanguage programming tool **56**

Why LASAL CLASS 2 - Object oriented programming? **57**

LASAL and OOP **57**

Class Library **57**

Inheritance, Aggregating and Creating Complex Classes **57**

Client-Server technology **58**

Graphic presentation **59**

Helpful additional tools **60**

Information on LASAL CLASS 2 Help **61**

Disclaimer **62**

THE DESIGN ENVIRONMENT 63

Menu Bar **64**

File Menu **65**

New Project... 65

Open Project... 66

Save Project Changes 66

Save Project As... 66

Save Copy of Project 67

Save All 68

Save 68

Close Project 69

New File... 69

Open File... 69
Save File As... 69
Close File 69
Print... 70
Recent Files 70
Recent Projects 70
Exit 70
Edit Menu 71
Undo 72
Redo 72
Cut 72
Copy 72
Paste 72
Find... 72
Find in Files... 72
Find in Tree... 72
Find Retentive Server 73
Replace... 73
Replace in Files... 73
Goto Line... 73
Reset Editor Colors 75
Mark 80
Bookmark 83
Navigate Backward 84
Navigate Forward 84
Select All 84
Comment 85
Dent 86
NetEdit (active only when an object network is displayed) 87
DataAnalyzer (active only when the DataAnalyzer is opened) 87
Load Configuration 88
Save Configuration As 88
Import Data 88
Export Data 89

Upload Offline Data 89
Dias Drive (active only when the Dias drive is opened) 90
View Menu 91
Output Pane 91
Properties Window 91
Debugger Pane 91
Python Pane 92
Data Analyzer Pane 92
Drive Pane 93
Multimaster Pane 93
Trees 94
Toolbars 95
Status bar 95
Layout 96
Graphical User Interface 98
Project Menu 99
Info 100
Task Configuration 105
Unit Configuration 105
OPC UA Configuration 112
Exception Handling 114
Reengineering... 115
Cleanup 117
Compare 119
EasyCase 120
General Function mode 120
Import Process 122
Export Process 124
Keywords in the EasyCase Export File 125
Restrictions for EasyCase/EasyCode 126
Export Visualized Servers 127
Matlab Simulink Import 127
Relevant Settings in Embedded Coder 136

Solver 137
Math and Data Types 138
Hardware Implementation 139
Model Referencing 140
Code Generation 141
Optimization 142
Identifiers 144
Interface 145
Templates 148
Code Placement 149
AutomationML 150
Functionality 151
Import Hardware Tree 159
Validate GUID 164
Update Includes and Loader 165
Import Class / Network 165
Import Safety Parameter List 166
Multimaster 175
Functions, concerning a MultimasterStation 191
LDR_MM_StatNrGetName 191
LDR_MM_StatNrConnStop 191
LDR_MM_StatNrConnStart 192
LDR_MM_StatNrConnSetStr 192
LDR_MM_StatNrConnGetStr 193
LDR_MM_StatNrIsConnected 194
LDR_MM_SetWaitForReqTimeout 195
LDR_MM_StatNameGetNr 195

Functions, concerning a MultimasterServer 196
LDR_MM_SvrIsConnected 196
LDR_MM_SvrGetHandle 196
LDR_MM_SvrGetHandleFromCmpName 197
LDR_MM_SvrGetNameFromHandle 197
LDR_MM_ReadSvr 198
LDR_MM_ReadSvrByHandle 199
LDR_MM_SvrNewInst 199
LDR_MM_SvrGetState 200
Callback Functions 201
LDR_MM_UserCallbackPrototyp 201
LDR_MM_InstallCallback 201
_MMState 202
Multilanguage Comments 202
Add Multilanguage Identifiers To Current Project 204
Import / Reimport Multilanguage Comments 209
Export Multilanguage Comments 210
Remove Multilanguage Identifiers (GUID + Language ID) 212
Remove Multilanguage Comments 214
Set Active Languages 215
Build Menu 219
Debug Menu 221
Online 222
Set / Delete online password 222
Download 223
Download Module On the Fly 225
Class Configurations 226
Workarounds 227
Process 227

Return Message 228
Special Cases 229
Build Changes Download and Run 229
Start 229
Reset 229
Step & Instructionpointer 230
Breakpoint(s) 231
Settings 234
File Transfer 235
Interpreter 241
Tools Menu 242
Advanced Debug Tools 243
Data Analyzer... 243
Motion 244
Hardware Editor... 244
Grafical Hardware Editor... 245
IO Connection Manager... 245
Varan Service Tool... 245
Hardware Info Viewer 245
Interpreter... 245
Scripts 246
Update Lobs... 251
Create Bootdisk... 253
Extras Menu 257
Service Menü 257
Service Password 257
Options 258
State Info 273
Code Editor 276
Sfc Options 281
Advanced IOs 281
Macros 282
Window Menu 286
Help Menu 287

Tool Symbol Bars (Overview) ... 289
Standard Toolbar ... 290
Build Toolbar ... 290
Debug Toolbar ... 291
Net Edit Toolbar ... 292
Toolbox Toolbar ... 292
Layout Manager Toolbar ... 292
Macros Manager Toolbar ... 293
Edit Toolbar ... 293
Project Tree ... 294
Libraries Tree ... 295
Files Tree ... 312
Additional Files ... 314
Header Files ... 315
Source Files ... 316
Network Files ... 317
Object Files ... 317
Library Files ... 318
Dias Drive Files ... 319
Connection Files ... 319
Global Tree ... 320
The "Types" Folder ... 321
The "Variables" Folder ... 324
Networks Tree ... 325
Classes Tree ... 329
Method and server symbols ... 331
Folder External ... 333
Tool-dependent Temporary Trees ... 349
Properties Window ... 350
Project Properties ... 350
Properties of a project ... 350
Context menu of a project ... 374
Class Properties ... 377
Properties of a class ... 378

Context menu of a class 384
Server Properties 392
Properties of a server 393
Context menu of a server 396
Client Properties 400
Properties of a client 401
Context menu of a client 403
Method Properties 404
Properties of a method 404
Context menu of a method 405
Variable Properties 406
Properties of a variable 406
Context menu of a variable 410
Type Properties 411
Properties of a type (class oriented type) 411
Context menu of a type 413
Properties of a type (global type) 414
Table Properties 415
Properties of a table 416
Context menu of a table 417
Object Properties 418
Properties of an object 418
Context menu of an object 420
Revision 421
Password Protection and Encryption of a Project / a Class 422
Work Space 425
Code editor 426
Comfort functions 427
Code completion 429
Network view 430
Vertices 434
Other Displays 438
The Hardware Editor 438
The Graphical Hardware Editor 440

KOP Class ... 442
SFC Class ... 443
Information Window ... 444
Output Window ... 444
Output ... 445
Logging ... 445
Debugger Trace ... 445
Find Results 1 ... 446
Find Results 2 ... 446
Debugger Window ... 446
THIS ... 447
Auto ... 448
Watch 1-3 ... 449
The SetValue column and its context menu ... 449
Drag & Drop in the Watch Window ... 451
Casting in the Watch window ... 454
Register ... 455
Breakpoints ... 455
Call Stack ... 456
Bookmarks ... 456
Python Script window ... 456
Tool-dependent temporary window ... 457
Status bar ... 461
PROGRAMMING 462
Classes and Objects in LASAL CLASS 2 ... 463
Introduction to Object Orientation ... 463
Object Oriented Programming (OOP) Basics ... 463
Advantages of OOP ... 463
Important Concepts in OOP ... 464
Classes ... 464
Objects ... 464
Instantiation ... 464
Attributes ... 464

Methods 464
Data Encapsulation 465
Inheritance 465
Polymorphism 465
Class Definition 466
Basic Information on Methods (Privat, Global, Virtual Global) 466
Private Methods 466
Global Methods 467
Virtual global Methods 468
Virtual private Methods 469
Classes and Objects in LASAL Class 2 470
The Client and Server Model 472
Methods of a Class 473
The Standard methods of a class 474
Constructor 475
Init 476
CyWork (Cyclic Work) 477
RtWork (Realtime Work) 477
Background 478
Calling a Method 479
Calling Methods with Transfer parameters 480
Calling a Method with Transfer and Return Parameter 483
Calling a Method via the object channel 485
Local Variables of a Method 489
Register settings for IO variables 490
The Read and Write Methods of Server 494
Read methods for servers 494
Write methods for servers 496
Objects 498
Complex Objects general 498
Connecting Objects in different networks 502
Moving and copying of objects 507
The Default Width and Auto size Functions 508

Duplicating Object Networks 510
Search and Goto Definition in object networks 512
Complex Classes 512
Important Information on Deriving Classes 512
Deriving Classes and Overwriting Methods 513
Processing Tasks 516
General Information about the Different Task Types 517
RealTime 517
Cyclic 517
Background 517
Configuration of the Tasks 517
In the class 518
In object 519
Processing the tasks by the CPU 519
Online configuration of the tasks 521
Structured Text 528
Structured Text Basics 528
Basic ST Syntax 528
Upper and Lower Case writing 528
The Semicolon 528
Program Structure 529
Commentary 530
Variables and Data Types 530
Data Types 530
Alignment for REAL/LREAL Numbers 536
The Difference Between Conversion and Casting 537
Variable Declaration 542
The Variable Identifier 543
Validity period (Lifespan) of variables 544
Variable Types 545
Structures 546
Special Variable forms 546
Data fields (Arrays) 550

Declaration and Initialization of an Array 550
Accessing Array Elements 551
Memory Image of an Array 552
Multidimensional Array 553
Instructions 557
The If-Instruction 558
The Case Instruction 559
Program Loops 560
For-Loop 560
While-Loop 560
Repeat-Loop 560
Interrupt instructions for Loops 561
Exit 561
Return 561
GoTo 562
ST instructions 562
SEG, OFF 562
MK_FP 563
STI 563
CLI 563
SAVE_FLG 564
RESTORE_FLG 564
SAVE_REG 564
RESTORE_REG 564
TO_BOOL 564
TO_SINT 565
TO_INT 565
TO_DINT 565
TO_USINT 566
TO_UINT 566
TO_UDINT 567
TO_REAL 567
TO_LREAL 568

Operators 569
Arithmetic Operators 569
Bit Operators 569
NOT 570
AND 570
OR 571
XOR 572
SHL 572
SHR 572
ROL 573
ROR 573
Logic Operators 574
Assignment operators 574
Comparison Operators 575
Component Selection Operator 575
Address Operators 576
Indirect Operator 576
Operator sequence 576
Standard functions 577
ABS 578
SQRT 578
LN 578
LOG 578
EXP 579
SIN 579
COS 579
TAN 579
ASIN 579
ACOS 580
ATAN 580

SIZEOF ... 580
Examples of Data Types ... 580
BOOL ... 580
SINT ... 580
INT ... 580
DINT ... 581
USINT ... 581
UINT ... 581
UDINT ... 581
REAL ... 581
LREAL ... 581
CHAR ... 582
BSINT ... 582
BINT ... 582
BDINT ... 582
HSINT ... 582
HINT ... 582
HDINT ... 583
ARRAY ... 583
STRUCT ... 583
ENUM ... 584
Type Conversion ... 584
Pointer ... 585
Definition of a pointer ... 586
Pointer Contents - NIL ... 586
Variable Addresses ... 587
Table Contents ... 588
Additional Components of Structured Text Programming ... 589
Working with Pointers ... 589
What is a Pointer? ... 589
Definition of a Pointer ... 589
THIS-Pointer ... 590

NIL-Pointer ... 591
Transferring a Pointer to a Method ... 591
Pointers Used as Return Parameters of a Method ... 594
Summarization of Pointers ... 597
Working with Tables ... 597
What are Tables Used for? ... 597
General Information on Tables ... 597
Syntax for Texts and Numbers in Tables ... 597
Table Content with Text ... 600
Table Content with Numbers (Constants) ... 605
Instruction List (IL) in LASAL CLASS 2 ... 607
Principle Construction of CPU Memory ... 608
Schematic Memory Distribution of a LASAL CPU ... 608
The Most Important CPU Register ... 609
Instruction Groups for Programming IL ... 609
IL Instructions ... 610
Introduction ... 610
STL instructions overview ... 611
Processor ... 615
Register ... 617
Special Registers ... 618
Processor Status Register ... 618
Data Transfer Instructions ... 620
L - LOAD ... 623
S - SAVE ... 623
SET ... 624
CLR ... 624
XCH ... 625
LODSB ... 626
LODSW ... 626
LODSD ... 627
STOSB ... 627
STOSW ... 628

STOSD 628
MOVSB 629
MOVSW 630
MOVSD 632
INB 633
INW 633
IND 634
INSB 634
INSW 634
INSD 634
OUTB 635
OUTW 635
OUTD 635
OUTSB 636
OUTSW 636
OUTSD 636
PUSH 637
PSH 638
PSHA 638
PSHF 638
PUSHAD 638
PSHFD 639
POP 639
POPA 639
POPAD 639
POPF 639
POPFD 640
LEA 640
LDS 641
LES 641
LAHF 641
SAHF 641

RST 642
PRS 642
LSS 642
LGS 643
LFS 643
SETA 644
SETAE 644
SETB 644
SETBE 645
SETC 645
SETE 645
SETG 646
SETGE 646
SETL 646
SETLE 647
SETNA 647
SETNAE 647
SETNB 648
SETNBE 648
SETNC 648
SETNE 649
SETNG 649
SETNGE 649
SETNL 650
SETNLE 650
SETNO 650
SETNP 651
SETNS 651
SETNZ 651
SETO 652
SETP 652
SETPE 652

SETPO 653
SETS 653
SETZ 653
LSXB 654
LSXW 654
LZXB 654
LZXW 655
Logic Instructions 655
O 656
OR 656
A 657
AND 657
X (Abbreviation for XOR) 658
XOR 659
NOT 660
Comparison Instructions 660
CMP 661
CMP0 661
CMP1 662
CMPSB 663
CMPSW 664
CMPSD 664
SCASB 664
SCASW 665
SCASD 665
TEST 665
Shift Instructions 666
SHLD 666
SHRD 667
ROL 667
ROR 668
RCL 668

RCR 668
SAL 669
SAR 669
SHL 670
SHR 670
Jump Instructions 670
JE 672
JNZ 672
JNE 673
JNBE 673
JA 673
JC 673
JNAE 673
JB 673
JNC 674
JNB 674
JAE 674
JBE 674
JNA 675
JS 675
JNLE 675
JG 676
JNL 676
JGE 676
JNGE 676
JL 677
JLE 677
JNG 677
JO 677
JNO 678
JPE 678
JP 678

JPO 678
JNP 678
JNS 678
JCXZ 678
JECXZ 679
LOOP 679
LOOPE 679
LOOPNE 680
JMP 680
CALL 681
RET 682
FC 682
CALLF 682
MODEND 682
IRET 682
Arithmetic instructions 683
INC 683
DEC 684
ADD 684
ADC 685
SUB 685
SBC 686
NEG 686
MUL 687
IMUL 689
DIV 689
IDIV 691
MOD 691
Status register manipulation 691
STC 691
CLC 692
CMC 692

CLI 692
STI 692
STD 693
CLD 693
Special instructions 693
- (NOP) 694
ENTER 694
LEAVE 694
CBW 695
CWD 695
XLAT 696
AAA 696
AAM 698
AAS 698
AAD 699
DAA 699
DAS 700
BOUND 701
BSF 701
BSR 702
BT 702
BC 702
BTR 703
BS 703
CWDE 704
CDQ 704
Contact Plan (COP) in LASAL CLASS 2 705
Fundamentals for Creating a Contact Plan 705
Contact Types 717
Normally Open (Open Contact) 718
Normally Closed (Closed Contact) 718
Positive Edge Contact 719
Negative Edge Contact 719

Coil Types **720**
Coil 721
Negated Coil 721
Set Coil 721
Reset Coil 722
Memory Coil 722
Set Memory Coil 724
Reset Memory Coil 724
Positive Transition 725
Negative Transition 725
Additional Contact Plan Elements **726**
Jump 726
Label 726
Return 726
Additional Contact Plan Functions **727**
Insert New Row 727
Delete Empty Row 727
Important Display Modes and/or Settings **727**
Switch Status Display in the Contact Plan 727
Class Properties with a Contact Plan 729
Inputs and/or Outputs as Alarm displays 731
SFC **733**
Description of SFC Components **754**
SFC Initstep 754
SFC Step 754
SFC Transition 754
SFC Jump 755
SFC Action 755
SFC Action Qualifier 756
N (not saved) 756
S (saved) 756
R (reset) 756
P (with rising and falling flank) 756
P0 (with falling flank) 757

P1 (with rising flank) ... 757
L (time-limited) ... 757
D (time-delayed) ... 758
SD (saved and time-delayed) ... 758
DS (time-delayed and saved) ... 759
SL (saved and time-limited) ... 759
SFC Alternative Branch ... 760
SFC Parallel Branch ... 760
SFC User Interface Options ... 764
THIS and Watch Window in SFC ... 766
Call Squence of SFC Components ... 769
Context Menu of the SFC Editor Components ... 769
Init Step ... 770
Step ... 771
Transition ... 772
Action ... 773
Jump ... 774
IO ... 774
Pre-Processor Instructions ... 775
Predefined Constants ... 775
Trace Macro ... 775
TRACE0 - TRACE2 ... 775
Constant _PI ... 777
_LSL_COMPILEREVERSION ... 777
LSL_entriesof ... 777
Available Pre Processor Instructions ... 777
#define ... 778
#undef ... 778
#ifdef ... 779
#ifndef ... 780
#elif, #else, #endif ... 780
#if ... 781
Pragma Instruction ... 781
#pragma warning ... 782

#pragma pack ... 783
#pragma pack (push, n) ... 783
#pragma pack (pop) ... 784
#pragma pack (n) ... 784
#pragma message ... 786
#pragma once ... 786
#pragma define (store) ... 787
#pragma define (restore) ... 787
#pragma checkarray (enable | disable | default) ... 787
#include ... 787
#include "File.h" ... 787
#include <File.h> ... 787
#includeC "file.h" ... 787
Linking Header Files ... 789
Class Linked Header Files ... 790
Global Header Files ... 793
Creating a Header File ... 796
A Note on Global Header Files ... 799
Interpreter in LASAL CLASS 2 ... 800
Interpreter Applications ... 800
The Interpreter Tree ... 801
Layout of the Tree ... 802
Project ... 803
PLC ... 807
Applying an Interpreter in LASAL CLASS 2 ... 809
General Information on the Interpreter ... 813
Starting the Interpreter Automatically ... 813
Initializing the Interpreter ... 814
Interpreter Syntax ... 814
Label Definition ... 814
Subdirectory Definition ... 814
End Interpreter ... 814
Exit Program ... 814
Allocation ... 815

Incrementation ... 815
Decrementation ... 815
IF - THEN Condition ... 815
Instructions for Stopping an Interpreter ... 815
Interpreter Online Mode ... 817
Debugging in the Interpreter ... 818
Setting Breakpoints in the Interpreter ... 818
Debugging in Single Step Mode ... 818
Commands for Debugging in the Interpreter ... 818
Downloading an Interpreter ... 819
Download in Reset Mode ... 820
Downloading during Project Runtime ... 821
Uploading an Interpreter ... 822
Freeze Project Comparison ... 823
The Standard Methods NewInst(), GetState() and Kill() ... 823
General Information over the Standard Methods ... 823
Applications for the NewInst() and GetState() Methods ... 826
NewInst() ... 826
GetState() ... 827
Kill() ... 828
Using the NewInst() and GetState() Methods via Multimaster ... 828
Debugging with NewInst Commands ... 829
IPR Editing during Runtime ... 840
Starting an Interpreter ... 840
Stopping an Interpreter ... 841
Emergency Stop of an Interpreter ... 841
Setting the Instruction Pointer ... 841
Determining the Interpreter Address ... 842
Breakpoints ... 843
Setting Breakpoints ... 843
Clearing Breakpoints ... 843
Single Stepping ... 844
Continuing Normal Operation after Single Step ... 844
Determining the Current Line Number for Monitoring ... 844

Exchanging the Interpreter during Runtime 845
Required Functions 845
Example: Exchange during Runtime 846
Required Types 847
Using C-Files in LASAL CLASS 2 849
How are C-Files Linked in LASAL Projects? 849
Creating a C-File 850
Linking a C-File 851
Important Details when Using C-Files 854
C-Functions Supported by the Runtime Library 856
Loader 858
Memory, String 858
_memmove 858
_memicmp 858
_memcmp 859
_memcpy 859
_memcpyrev 860
_memset 860
_strcat 861
_stricmp 861
_strupr 862
_strchr 862
_strcmp 863
_strcpy 863
_strlen 864
_strncpy 864
_itoa 865
Classes and Objects 865
_LookUpEmbed_C 865
Methods, Clients, Server 866
RD 866
RD_CHNL 866
WR 867
WR_CHNL 867

Interpreter Programs 868
Calling Interpreter Programs 868
ProgIp_SetMaxProgSeqDuration_ms 868
IprMgr_SetMaxProgSeqDurationAll_ms 869
IprMgr_SetProgSeqCycleTime_ms 869
ProgIp_InstallIprCallback 869
Creating Interpreter Programs 871
ProgMgr_CreateIprProg 871
ProgMgr_LoadIprProg 872
Comlink 872
Loader Helpfunctions for the Data Type REAL 873
Real_IsSigned 873
Real_IsNormal 873
Real_IsFinite 873
Real_IsZero 874
Real_IsSubnormal 874
Real_IsInfinite 874
Real_IsNaN 874
Real_IsSignaling 875
Real_GetClass 875
Debugging a Program 876
Debugging with Conditional Breakpoints 881
Debugging with Data Breakpoints 883
Debugging with NewInst Commands 888
The Views Auto and LiveView 888
Scripting 891
What is Scripting? 891
Introduction in Python 891
LASAL CLASS 2 Python Commands 891
AddFile 892
AddLibrary 894
AddPrjConnectionFile 895
AddPrjDriveFile 896
AddVirtualCommissioning 898

CalculateResourcesAndExport ... 899
ChangeClass ... 900
ChangeIOServer ... 901
ChangeLoadLibraryFlag ... 902
ChangeVirtualCommissioningClass ... 903
CleanupClassesAndLibraries ... 904
CleanupClasses ... 905
CleanupProjectDirectory ... 905
ClearRamImage ... 906
CloseAllProjects ... 907
CloseLogfile ... 908
CloseModemConnection ... 909
CloseProject ... 910
Compile ... 911
CompileSuppressRebuild ... 912
CopyAll ... 913
CopyClass ... 914
CopyClassList ... 916
CopyNetwork ... 917
CopyNetworkAndLinkClasses ... 918
CopyNetworkList ... 919
CopyNetworkListAndLinkClasses ... 920
CreateBootdisk ... 921
CreateBootdiskMngr ... 923
CreateBootdiskViaContainer ... 927
CreateConnection ... 928
CreateConnection2 ... 929
CreateIOObject ... 931
CreateLabel ... 932
CreateNetwork ... 933
CreateObject ... 935
CreateProject ... 937
DecryptConnectionFile ... 938
DeleteClass ... 939

DeleteConnection 940
DeleteConnection2 942
DeleteFileOnPLC 944
DeleteNetwork 945
DeleteObject 947
DeletePrjConnectionFile 949
DeleteVirtualCommissioning 951
Download 951
DownloadConnectionFile 953
DownloadFile 954
DownloadOS 956
DuplicateNetwork 957
EncryptConnectionFile 959
ExistsClass 960
ExistsConnection 960
ExistsConnection2 963
ExistsIOLabel 966
ExistsNetwork 967
ExistsObject 967
GetClassObjectNames 969
GetCompilerVersion 970
GetCreateBootdiskContainer 971
GetCurrentCompilerVersion 972
GetVirtualCommissionProperties 974
ImportAmlHwConfig 974
ImportFromFile 976
IsExceptionOnError 979
LinkClass 979
LinkClassList 981
LinkObjectToPrjDriveFile 982
LoadProject 984
MoveNetworkToFolder 985
OpenLogfile 986
RemoveFile 988

RemovePrjDriveFile 989
RenameNetwork 990
RenameObject 992
RenameVirtualCommissioningObject 994
ReplaceObject 994
ResetNetworkOptions 997
Save 998
SaveAs 999
SaveProjectOnPLC 1000
SetAdvancedObject 1001
SetClassProtection 1001
SetCommentNetwork 1003
SetCommentObject 1004
SetCompilerVersion 1006
SetExceptionOnError 1007
SetFileProtection 1008
SetInitValue 1010
SetInitValue2 1013
SetMultiCPUCore 1016
SetNetworkOptions 1018
SetOPCUAFolderObjectServer 1019
SetOPCUAHostnameObjectServer 1020
SetOPCUAPhysicObjectServer 1021
SetOPCUASettingObjectServer 1023
SetOPCUAUnitObjectServer 1024
SetOPCUAWriteProtectedObjectServer 1025
SetParameterValue 1026
SetPrjConnectionFileActive 1027
SetPrjOPCUAEnable 1029
SetTaskCPUCore 1030
SetTaskOrder 1031
SetTaskOrder2 1033
SetTaskTime 1034
SetTaskTime2 1035

SetVirtualCommissionProperties ... 1037
SetVisualizedFlag ... 1037
SetVisualizedFlag2 ... 1038
SetVisualizedFlagOPCUA ... 1039
Start ... 1040
StartMatlabSimulinkImport ... 1041
StartRecordingImportFiles ... 1042
Stop ... 1044
StopRecordingImportFiles ... 1045
UpdateAllUsedClassesOfLib ... 1046
UpdatePrjIncludesAndLoader ... 1048
UploadFile ... 1049
Calling Scripts in LASAL CLASS 2 ... 1050
Calling from the Console ... 1050
Calling from LASAL Class 2 ... 1050
Calling a Python Script in LASAL Class 2 from another Python Script ... 1050
Python Script Window ... 1051
LASAL CLASS 2 - Python Classes ... 1051
Project ... 1051
Application ... 1053
Additional Commands ... 1054
_internal.msgbox(strText) ... 1055
_internal.write(strText) ... 1055
_internal.logInfo(strText) ... 1055
_internal.logWarn(strText) ... 1055
_internal.logError(strText) ... 1055
_internal.GetApplicationPath() ... 1055
_internal.inputBox(strText) ... 1056
Scripting - Example ... 1056
Error Handling ... 1064
Setting of Project Settings ... 1064
Setting the Network Options ... 1065
Setting the Server list Options ... 1066
Setting the Target Options ... 1066

Setting the Compiler Settings 1067
Setting the Directory Settings 1069
Setting the Virtual Commissioning Settings 1069
Brief Overview of the Python Modules 1070

ADDITIONAL VIEWS, TOOLS,... 1074

Online Settings 1075
Basics 1076
Connection Types 1085
Connection over the Serial Interface 1085
Settings in the LASAL Online Configurations Window 1085
Connection over USB Cable 1088
Connecting over the CAN Bus 1089
Settings in the Operating System 1090
Settings in the LASAL Online Configurations Window 1091
Connecting Over TCP/IP 1093
Settings in the Operating System 1093
Settings in the LASAL Online Configurations Window 1095
Connecting via Modem 1101
Block Diagram of the Connection 1101
Destination System Settings 1101
Example of WESTERMO TD-32 DC 1101
Online Configuration 1102
Operating System Settings 1104
Settings in the LASAL Online Configurations Window 1105
Connection via ROUTE 1107
LARS 1112
Loopback Address 1113
Own IP Address 1114
Hardware Editor 1116
Configuring the PLC 1117
Using 2 VARAN Managers in LASAL Class 2 1126
EtherCAT 1128
EtherCAT - Feature Übersicht 1138

Edit Ethercat Slave Parameter Dialog ... 1140
General ... 1141
PDO ... 1142
Slots ... 1144
DC ... 1145
StartUp ... 1147
The Properties Window of the PLC ... 1151
Adding Modules ... 1157
Context Menu of the Modules ... 1175
Properties of the VaranManager ... 1178
The Properties Window of the Module ... 1180
The Properties Window of the In- and Outputs ... 1182
Connections in the Hardware Editor ... 1183
View Options in the HW Editor ... 1187
Online with the Hardware-Editor ... 1189
Resources Calculation for VARAN and S-DIAS ... 1194
Overview (VARAN) ... 1194
Detail (VARAN) ... 1197
SDIAS ... 1199
Power Supply ... 1200
Showing the Resources for 2 VaranManagers ... 1204
IO Connection Manager ... 1206
IO Connection Manager Areas ... 1207
Entering Connections in the IO Connection Manager ... 1218
Properties of an IO object ... 1239
The Project Connection Files Window ... 1240
Advanced IO functionality ... 1259
The Virtual Commissioning Manager ... 1266
Graphical Hardware Editor ... 1269
The Toolbar ... 1271
Product Catalog ... 1276
The Workspace ... 1278
Connection of SDCI Devices ... 1298
Creating a SDCI Device Class ... 1298

Connecting a SDCI Device ... 1301
DiasDrive ... 1305
What is VARAN? ... 1305
Accessibility ... 1305
DiasDrive Edit ... 1307
DiasDrive Tree ... 1309
DiasDrive Config and DiasDrive Debug Window (Overview) ... 1310
Parameter Files Processing ... 1312
Implementation of Parameter Files to a LASAL Class 2 Project ... 1317
Online Configuration ... 1318
Interpreter Window ... 1328
Watch Window ... 1330
State Window ... 1330
Error Buffer ... 1332
Controller Setup Window ... 1332
Error Configuration Window ... 1335
Motor Configuration Window ... 1336
Device Configuration Window ... 1339
Comm.(unication) Setup Window ... 1340
DataAnalyzer for DiasDrive ... 1341
CAM-Designer ... 1345
Accessibility ... 1345
The Input and Control Area (left side) ... 1346
Settings ... 1347
The Input Table ... 1348
Error Message Area ... 1349
Control Area ... 1350
The Display Area (right side) ... 1352
Curve Display and Zoom ... 1353
Change of Data ... 1354
VOV (Visual Object View) ... 1359
VOV-(Visual Object View-) Attaching Files to Classes ... 1359
DataAnalyzer ... 1360
DataAnalyzer User Interface ... 1360

DataAnalyzer Menu 1361
DataAnalyzer Main Window 1363
DataAnalyzer Tree 1364
DataAnalyzer Configuration 1365
DataAnalyzer Properties 1371
Trigger Properties 1375
Properties of a Measuring Channel 1376
Saving and Loading the Configuration 1379
Recording with the DataAnalyzer window 1380
Control Unit 1380
Display Area 1382
Start Measuring 1388
Measuring and/or Recording in OFFLINE Status 1389
Saving and Loading Measuring Values 1389
Importing / Exporting Data 1390
Advanced Debug Tools 1395
Accessibility 1395
The Individual Debug Tools 1397
Memory Dump 1397
Profiler 1400
Object List 1401
Target Info 1402
Can Trace 1404
Make Bootdisk 1405
Remote CLI 1410
File Commander 1412
Ram Image 1414
PLC Trace View 1419
Run Trace Recording Window 1428
Record of Data 1430
PLC Log Viewer 1437
PLC Backup and Restore 1444
PLC Debug Level 1455
Flash Media Lifetime Calculation 1464

The Encryption Tool "Encrypter.exe" 1467

TIPS AND HELPS FOR WORKING WITH LASAL CLASS 2 1471

Using Online Values as Initial Values 1472

Transfering Init Values from Servers/Clients to Classes of all Existing Objects 1476

User-defined Folders in the Project Tree 1478

Removing of Duplicate GUIDs 1481

Drag & Drop in the Program Code Editor 1483

Go To Definition 1485

Toggle Breakpoint / Toggle Bookmark 1488

Toggle Function Folding 1490

Insert Client-Update Source Code... 1492

Insert NewInst command 1496

Mark Text and Clear Marks Function 1505

Quick Change of the Online Settings 1506

Context Menu of the Network View: Highlighting Connected Server, Creating a New IO Object 1507

Run Find References for a Server 1510

Run Find References for a Type 1512

Finding Clients / Server in the Implementation 1514

Navigation in the Object Network 1516

- Context Menu of a Client/Server in the Object Network 1516
- Navigation with Cursor Keys 1519

Execute Set init value to all instances resp. Set connection to all instances for an embedded Client 1521

Go to Datatype with Elements that can have a Type 1525

Loading of projects with missing elements 1526

- Library could not be found 1526
- Class in the library could not be found 1527
- Header file could not be found 1529

Open Dialogs with Quick Navigation 1530

Search and Replace 1531

Simple Search **1531**
Find in Files **1533**
Find in Networks **1536**
Search in Trees **1537**
Search for Retentive Server **1539**
Simple Replacement **1540**
Replace in Files **1541**
Easy Derivation of Classes **1544**
Drag & Drop of Class Elements **1547**
Change of Order of the Class Elements **1547**
Moving / Copying of Class Elements **1547**
Displaying the Connections in the Object Network with Tooltip **1548**
Displaying the Variable Type in the Code Editor with Tooltip **1549**
Display of Method Names in the Status Bar for Simpler Orientation **1550**
Task Sequence without RtPostScan (up to Lasal 02.02.052) **1551**
Task Sequence with RtPostScan (with Lasal 02.02.054 and higher) **1553**
Effects of RtPostScan **1555**
Attach Help Files to a Class **1556**
Print functionality **1559**
Project **1560**
Class **1563**
Network **1567**
Preset Folders in the Open Project window **1570**
Known problems and limitations **1575**
Real Numbers - Conversion Errors **1575**
Structure of a 32-bit Real Number in Memory 1575
Example with Real Numbers 1575
The decimal number 100 is converted to the binary system 1575
The decimal break should be converted to the binary system 1576
Combining the number 1576
Calculating an Exponent 1577
Determine the mantissa 1577

Real number 100.1 combined 1577
Real number 200.2 combined 1577
Multiplying real numbers 1578
Source directory 1578
Mixing Copy and Link Classes 1578
Using linked classes 1578
Using copied classes 1579
Mixing both methods 1579
Example 1580
How to fix this issue 1584
General recommendations 1584

THE FIRST LASAL PROJECT 1588

Requirements 1589
Implementation 1590
Creating a New Project 1590
Creating a Class 1592
Selecting a Method 1595
Creating a Client 1599
Program Code Editor 1604
Automatic Code Extension (auto completion) 1607
Object Network 1608
Library 1613
Hardware Editor 1619

APPENDIX 1628

File Types in LASAL Class 2 1629
Project File *.lcp 1629
Source Code Files 1629
*.ST 1629
*.AWL 1629
*.IPR 1629
*.H 1630
*.C, *.CPP 1630

*.lcn 1630
Project Administration *.py 1630
Temporary Files 1631
*.lcb 1631
*.lob 1631
*.lba 1631
*.obj 1631
*.ldi 1631
*.bin 1631
Miscellaneous Files 1631
*.vov 1631
*.lda 1632
*.lce 1632
*.lcf 1632
*.lcc 1632
*.lds 1632
*.ldc 1632
*.csv 1632
*.txt 1633
*.etf 1633
*.lcz 1633
*.luc 1633
*.mme 1633
*.mmc 1633
*.xml 1633
Git-instructions for LASAL Tools 1634
Introduction 1634
Describing the incompatibility 1635
How to fix the problem 1635
Number Systems 1637
Decimal Number System 1637
Binary Number System 1637
Hexadecimal Number System 1638
The MathBSD Library 1640

Properties 1640
Usage / Activation in LASAL Class 2 1642
Function Reference 1644
ACOS 1644
ACOSH 1645
ASIN 1646
ASINH 1647
ATAN 1647
ATAN2 1648
ATANH 1649
CEIL 1650
COS 1651
EXP 1652
EXP2 1653
EXPM1 1654
FABS 1655
FLOOR 1656
FMOD 1657
FREXP 1658
HYPOT 1659
LOG 1660
LOG1P 1661
LN 1662
MODF 1663
POW 1664
REMAINDER 1665
REMQUO 1666
ROUND 1667
SCALBN 1668
SIN 1669
SINCOS 1670
SINH 1671
SQRT 1672
TAN 1673

TANH 1674
TRUNC 1675
C Programming Language Interface 1676
Application 1676
Example with LREAL 1676
Example with REAL 1677
Compatibility between LASAL CLASS 2 and CDECL functions with REAL return values 1679
Keyboard Shortcuts 1682
Loader Trace Messages 1686
Compiler Messages 1690
E 0001: Value missing instead of '%s' 1690
E 0002: Missing '%s' instead of '%s' 1690
E 0003: Multiple declaration of '%s' 1691
E 0004: Undefined type '%s' 1691
E 0005: '%s' is no integer value 1692
E 0006: '%s' is no constant value 1692
E 0007: Value '%s' out of range 1692
E 0008: Expected constant instead of '%s' 1693
E 0009: '%s' has wrong type 1694
E 0010: Incorrect string '%s' 1694
E 0011: Incorrect value '%s' 1695
E 0012: Different types: using '%s' instead of '%s' 1695
E_0013: Expected '%s' init values instead of '%s' 1695
E_0014: Unknown error 1696
E 0015: Error reading file '%s' 1696
E 0016: Not expected input '%s' 1696
E 0017: Syntax error: '%s' 1697
E 0018: Expected value instead of '%s' 1697
E 0019: Expected identifier instead of '%s' 1698
E 0020: UNUSED 1698
E 0021: Use only one return value with a register 1698
E 0022: Invalid register '%s' 1699
E 0023: UNUSED 1699

E 0024: Register '%s' already used as parameter 1700
E 0025: Expected 0 or 1, not '%s' 1700
E 0026: Undefined symbol '%s' 1701
E 0027: STRUCT '%s' has no element '%s' 1701
E 0028: Bitfield '%s' has no element '%s' 1701
E 0029: Class '%s' has no element '%s' 1702
E 0030: Misplaced EXIT 1702
E 0031: Constant not allowed in multiple assignment 1703
E 0032: Calculation not allowed in multiple assignment 1703
E 0033: Can not evaluate the address of a bit 1703
E 0034: Wrong data type for index 1704
E 0035: Only '=' and '<>' are possible 1704
E 0036: UNUSED 1705
E 0037: UNUSED 1705
E 0038: Duplicate case '%s' 1705
E 0039: Variable is not a pointer 1705
E 0040: Undefined function '%s' 1706
E 0041: Missing parameter '%s' 1706
E 0042: Extra parameter '%s' 1707
E 0043: UNUSED 1707
E 0044: Function '%s' has no output parameter 1707
E 0045: Class '%s' is not defined 1708
E 0046: Only constant expressions allowed 1708
E 0047: STRUCT '%s' without component 1708
E 0048: Bit is not allowed 1709
E 0049: Expected value is 0 or 1 1709
E 0050: Instruction is not allowed 1710
E 0051: Unknown command '%s' 1710
E 0052: Invalid register 1710
E 0053: Address missing 1711
E 0054: Syntax error 1711
E 0055: Invalid address %s %s 1711
E 0056: Not the same data length 1712
E 0057: Reserved system call 1712

E 0058: N, P and R require BIT-address and bit processor available 1713
E 0059: UNUSED 1713
E 0060: Use SI or DI, not BX 1713
E 0061: Invalid register. Use SI, DI, BX or BP 1713
E 0062: Segment register not allowed 1714
E 0063: Missing ',' 1714
E 0064: Missing constant 1714
E 0065: Register is expected 1715
E 0066: Division by 0 1715
E 0067: Label '%s' already exists 1716
E 0068: NOT 0/1 only inverts the least significant bit. Use NOT (FALSE/TRUE) instead. 1716
W 0069: Condition is always %s 1717
W 0070: Possibly mixing of '%s' and '%s' 1717
W 0071: Variable '%s' is also used as type name: '%s' 1718
W 0072: '%s' declared but never used 1718
W 0073: Parameter '%s' is never used 1719
W 0074: '%s' assigned a value which is never used 1719
W 0075: Possible use of '%s' before definition 1720
W 0076: Return value '%s' is not defined 1720
W 0077: Define directive needs an identifier 1721
W 0078: Redefinition of '%s' 1721
W 0079: Bad preprocessor directive syntax 1722
W 0080: Misplaced %s directive 1722
E 0081: Unknown preprocessor directive 1722
E 0082: Incorrect argument for SEGMENT or OFFSET 1723
E 0083: Extra parameter '%s' for macro '%s' 1723
E 0084: Missing parameter for macro '%s' 1724
E 0085: Different redefinition of function '%s' because of '%s' 1724
E 0086: UNUSED 1724
E 0087: Function '%s' already has a body 1725
E 0088: Linker: Unresolved external '%s' in "%s" 1725
E 0089: UNUSED 1725
E 0090: Different redefinition of variable '%s' 1726

E 0091: File '%s' could not be generated 1726
E 0092: Error reading file '%s' 1726
E 0093: '%s' is not a Sigmatek OBJ file 1727
E 0094: '%s' is not a Sigmatek DEF file 1727
E 0095: Wrong version of '%s' 1727
E 0096: UNUSED 1727
E 0097: '%s' is not a channel 1727
E 0098: Label '%s' is not defined 1727
E 0099: Redefinition of function '%s' in "%s"(%d) and in "%s"(%d) 1728
E 0100: Register as parameter is not allowed for __cdecl functions 1728
E 0101: Only one return value is allowed for __cdecl functions 1729
E 0102: '%s' is not for Intel 386 or later processors 1729
E 0103: '%s' contains not supported type of relocation '%s' 1729
E 0104: Not supported Symbol Table entry in '%s' 1729
E 0105: Could not execute compiler '%s' 1729
E 0106: Variable '%s' in "%s" and variable '%s' in "%s" use same address 1730
E 0107: Missing function 'main' 1730
E 0108: Standard methods are already defined 1730
E 0109: Using pointer to undef type '%s' 1730
E 0110: Incorrect use of channel '%s' 1731
E 0111: Too many errors 1731
E 0112: '%s' is not a LIB file 1731
E 0113: Cannot use '%s' in LIB '%s' 1731
E 0114: Base Class '%s' undefined 1731
E 0115: '%s' is not a base class for '%s' 1732
E 0116: Redefinition of variable '%s' in '%s' and '%s' 1732
E 0117: Size of channel data element must be 4 Bytes! %s 1732
E 0118: Private Function '%s' of class '%s' cannot be called here. 1733
E 0119: Function '%s' is declared as a table. Tables cannot be called. 1733
E 0120: Different redefinition of function '%s' 1733
E 0121: Different redefinition of variable '%s' 1734
E 0122: Linker is set for case sensitive linking: '%s' 1734
W 0123: Function '%s' is declared as VIRTUAL in base class '%s' 1734
E 0124: Loader error '%s' 1734

E 0125: RT-DataBase error: '%s' 1734
I 0126: Trace '%s': '%s' 1734
E 0127: No memory left for variable '%s' 1735
E 0128: File '%s' was compiled with a different alignment setting (%d) 1735
W 0129: Missing '%s' instead of '%s' 1735
W 0130: '%s' '%s' '%s' 1735
E 0131: %s 1736
E 0132: The size of class '%s' is greater than 64 KByte 1736
E 0133: The file '%s' does not exist 1736
W 0134: Label '%s' is declared but never used 1736
E 0135: The interpreter program is too long. The maximum code size is 64 Kbytes. 1737
E 0136: '%s' not found! 1737
W 0137: (Private) data element '%s' of class '%s' cannot be accessed here. 1738
W 0138: Class::Channel.Method results in a virtual call. Use Class::Channel::Method to call the method directly 1738
W 0139: #Objch/ObjPtr.Channel returns the address of the data element.Use #Obj.Ch.pMeth for Servers or #Obj.Ch.pCh for Clients target the Channel's address 1738
W 0140: '%s' has been compiled with an other version of LASAL 1738
W 0141: '%s' 1739
W 0142: skipping #define '%s' because of endless recursion. 1739
E 0143: Name collision with define '%s' - not replaced. 1739
E 0144: Offset of element '%s' exceeds 64 kB limit within struct '%s' 1739
W 0145: Identical redefinition of '%s' 1740
W 0146: Multiple comparisons within one instruction. This leads to a boolean evaluation 1740
W 0147: '%s' is not supported by this architecture 1741
W 0148: Unaligned access detected. REAL / LREAL elements have to start on 4 Byte-aligned addresses 1741
W 0149: Using X86 register in ARM code 1741
W 0150: Missing '%s' instead of '%s' 1741
W 0151: Class '%s' was assigned to variable 1742
W 0152: Datatype '%s' was assigned to variable 1742
W 0153: Virtual function '%s' is not declared as VIRTUAL in base class '%s' 1742
E 0154: Multiple declaration of enum '%s'. The enum has to be specified with the typescope 1742

E 0155: Multiple declaration of type ‘%s’. The type has to be specified with the classscope 1743
W 0156: Stacksize ('%u' byte) of function ‘%s’ exceeds the warning level. Check the size of local variables. The value can be set in the compileroptions 1743
E 0157: Circular dependency in type ‘%s’ from ‘%s’ 1744
W 0158: Compiler-Bug detected, please update to compilerversion ‘%s’ or higher to fix the bug 1746
E 0159: Negative Array limit ‘%s’ 1746
W 0160: Array of Struct with misaligned LREAL element 1746
W 0161: Possible syntax error in FOR-LOOP. Check for ">,>=,<,<=,<>,=" in the condition aera (to ...) 1747
E 0162: Table for object network '%s' not found 1747
E 0163: Classtable '%s' not found 1747
W 0164: C++ static constructors are not supported 1747
W 0165: Name collision with define ‘%s’ replaces bitfield 1748
E 0166: Incompatible types: Expression '%s' cannot be converted to '%s' 1748
W 0167: C file contains unresolved addresses: The program may fail to run 1749
W 0168: Debugging of Variables on this C file may not work as expected 1749
W 0169: Compiler optimization on this file may cause critical errors 1750
W 0170: '%s' cast of a '%s' type channel can lead to unexpected results 1750
W 0171: Accuracy of the constant value will get lost when applied to a single precision expression 1750
W 0172: Cannot resolve 'this' pointer for method 'NotMyBase-Class::MyVirtual'. Non-static methods will crash your program 1751
W 0173: Mixing '%s' with a REAL/LREAL constant is known to cause portability issues. It is recommended to use explicit conversion to the intended type 1752
W 0174: Subexpression '%s' may cause overflow for huge values and produce a negative value. 1752
E 0175: 16-byte alignment required 1753
Online Status and Error Messages 1754

1 INTRODUCTION

The new LASAL project software offers all the advantages of modern automation technology. The use of innovative program techniques significantly reduce the development time and makes possible simple and fast implementation of machine concepts. The addition of individual machine objects to LASAL simplifies communication between the constructor and the software designer.

1.1 All in One

The LASAL Project tool is an object oriented programming tool with client-server technology and graphic representation. LASAL provides a package of all functions needed for the creation of automated tasks:



1.2 LASAL supports several target platforms

With LASAL class 2, all programs for machine process control are prepared. Therefore the complete SIGMATEK product palette such as CPUs, terminals and industrial PCs are supported. A change of the target platform (within the SIGMATEK product line) is possible at any time with which little or no changes required in the LASAL CLASS 2 project.

1.3 Real time with a multilevel, multi-tasking system

The operating system supports and uses the pre-emptive as well as cooperative tasks. The visualization and application can therefore be run from a single processor.

1.4 Open data Interface

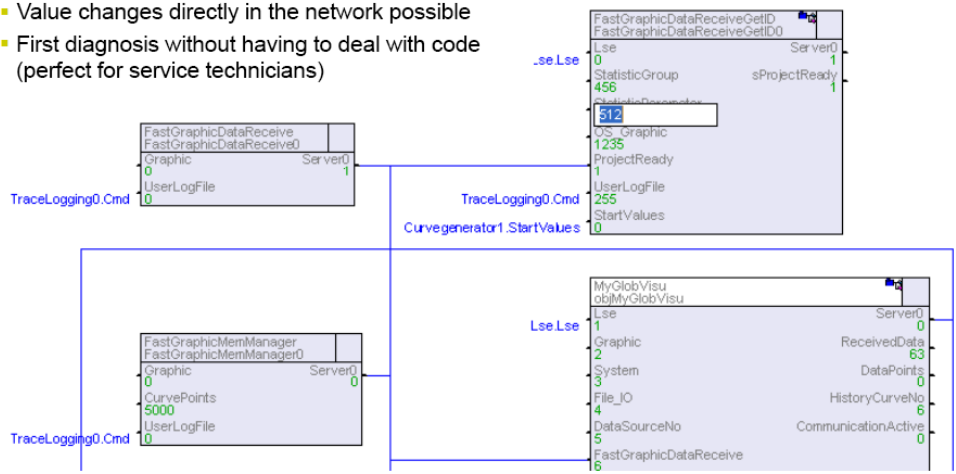
For individual data exchange using external programs, an OPC server (OLE for Process Control) with a standard interface is available. It is then possible for the use to link process data in visualization, control system or in Excel and generate individual evaluations, for example.

1.5 Worldwide service interface

Maintenance and diagnosis over a modem and/or Internet enable service technicians to communicate with the control, make program adjustments or make remote diagnoses.

Graphical Online diagnosis

- Actual status of the single objects is visible
- Value changes directly in the network possible
- First diagnosis without having to deal with code (perfect for service technicians)



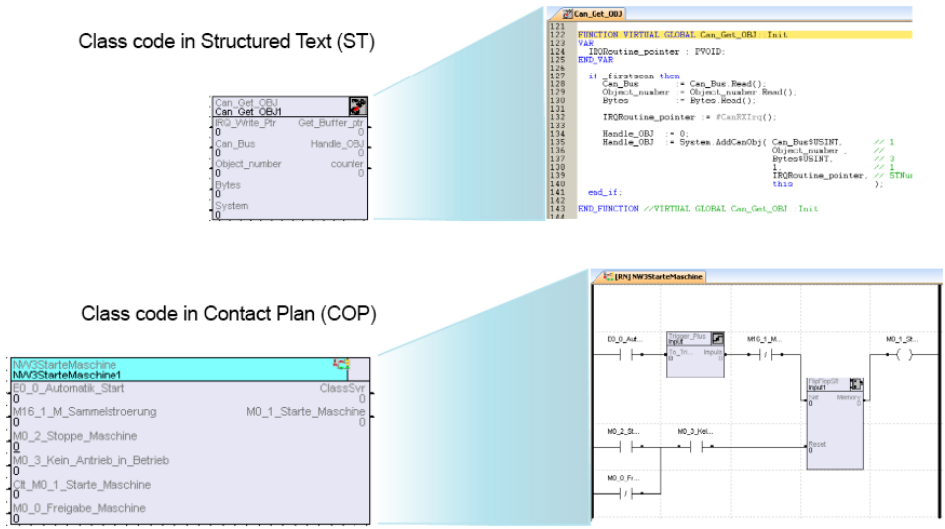
1.6 Online help system

LASAL has an extensive help system, which describes all functions in detail. With the Demo project, a quick start in the system is ensured.

1.7 Multilanguage programming tool

Applications can be created in four of the five IEC 61131-3 standardized programming languages of programmable logic controls (Structured Text (**ST**), Instruction List (**IL**), Sequential Function Chart (**SFC**) and Ladder logic (**LD**)), as well as in the **interpreter** and **C**.

Mix of different programming languages



1.8 Why LASAL CLASS 2 - Object oriented programming?

LASAL CLASS (Control Logic Application Software System) is the Project tool for the creation of automated tasks.

1.8.1 LASAL and OOP

LASAL expands the IEC 61131-3 standard to object orienting programming and ensures the simple and safe creation of applications using the most modern technology. In addition the software costs are reduced due to the modularity and reusability of the classes and the encapsulation of the user programs in the application is more efficient and easier to test.

Behind every object is a class with a program and the corresponding data elements. Therefore, each class can be assigned specific tasks such as measurement and evaluation of temperature, a filter function or motor control etc.

The management of the various classes occurs in a clearly visible library (Class Library). Classes are linked in the project tree through importation. Defined interfaces (connections) are available for the exchange of information, which, can send and receive reports. A class is placed in the network via Drag & Drop to instantiate an object. The object must only be connected and an application is created.

1.8.2 Class Library

The possibility also exists to create and centrally manage an individual library. Should a change in a library component be necessary, it must be made only once in the library and all projects containing the effected class are updated. The necessary programs components are loaded to the project tree by copying or linking with the import process.

In LASAL CLASS 2, an extensive collection of various classes are already available, including memory, time and date, motion, logic, controls, triggers etc. It is also possible to add user defined class libraries.

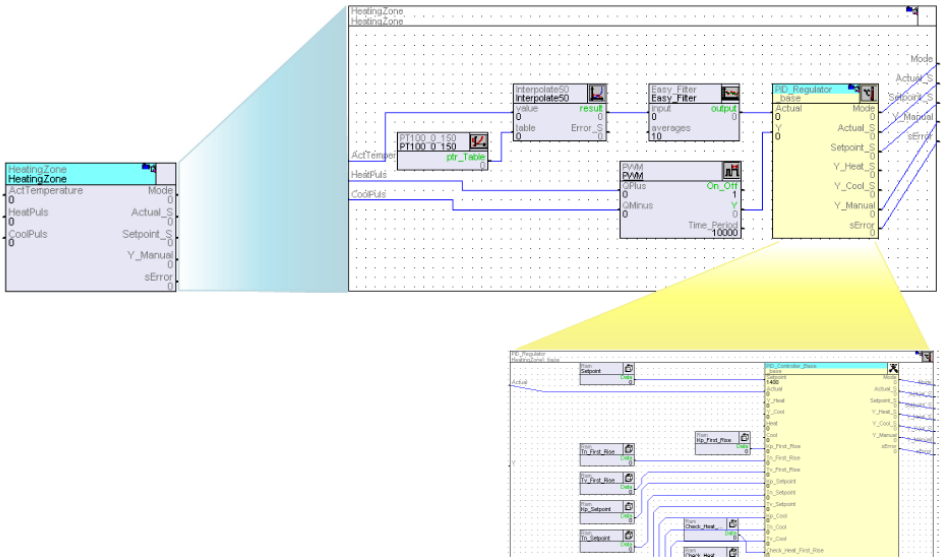
1.8.3 Inheritance, Aggregating and Creating Complex Classes

LASAL CLASS 2 supports modern programming techniques such as inheritance, aggregating and the creation of complex classes.

Inheritance is the relationship between the base class and a complex class. The complex conforms with the base completely and contains additional information (attributes, operations and association).

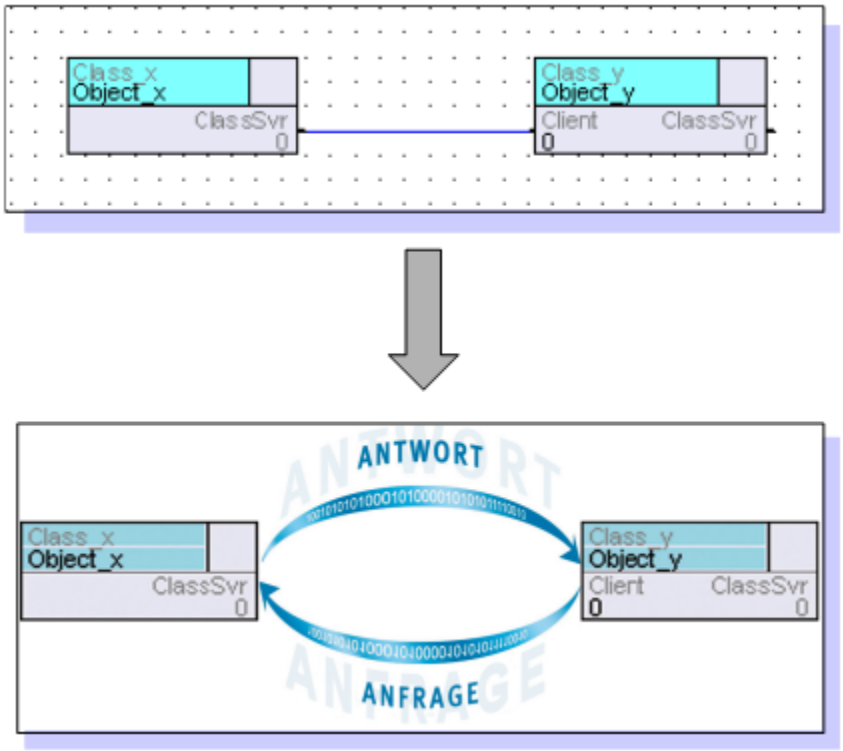
The base class can also be a derived class, which means that inheritance is possible over several hierarchical levels.

Through aggregation, several individual classes can be combined to form a complex class. An object is part of a “whole", complex object.



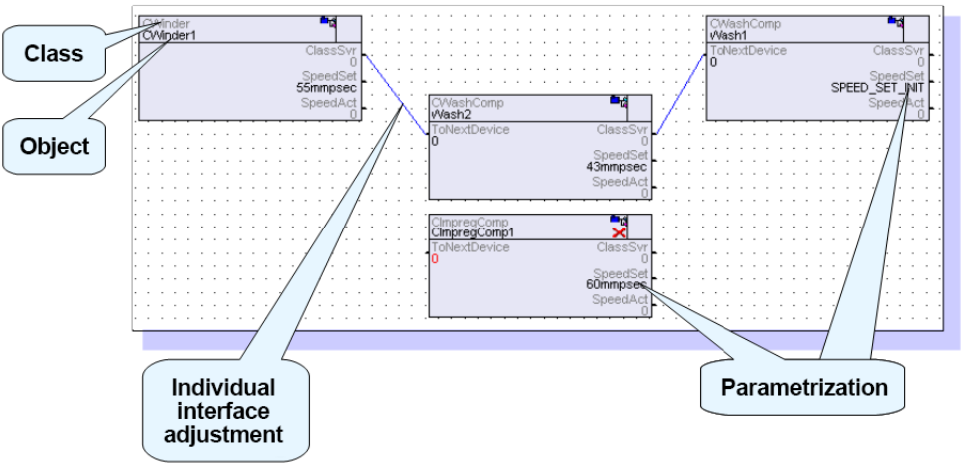
1.8.4 Client-Server technology

As with a PC network, the client requires services that are provided by a corresponding server. These communicate bi-directional over ONE connection. The server automatically provides standardized methods as a structure for the development of a class. This standardization makes it possible to interchange classes.



1.8.5 Graphic presentation

An essential characteristic of LASAL is the object oriented programming with graphic representation, project construction and debugging. An overview of the project, functions, data flow and interface connections can be obtained at a glance. Complex combinations are therefore easier to recognized, change and control.



1.8.6 Helpful additional tools

In addition to the previously mentioned technologies and views, LASAL also provides useful additional tools such as the CAM-Designer, DataAnalyzer, hardware editor with Connection Manager, Graphical Hardware Editor, DiasDrive, Visual Object View (VOV) etc. With these tools, work is simplified, more clearly organized and accelerated.

1.9 Information on LASAL CLASS 2 Help

Helpful Symbols

To make the LASAL Help as easy and comfortable to use as possible, three different symbols were provided. These symbols should simplify navigating through the help, provide additional information as well as highlight important sections and announce examples. The meaning of the three help symbols is as follows:



Attention

When you see this symbol, the lines following are VERY IMPORTANT and should be given your COMPLETE ATTENTION. It is recommended that the paragraph next to this symbol be THOROUGHLY READ.



Tip

The thumb up is the symbol for a helpful tip for how something can be done faster or better.



Example

This symbol indicates the beginning of an example.

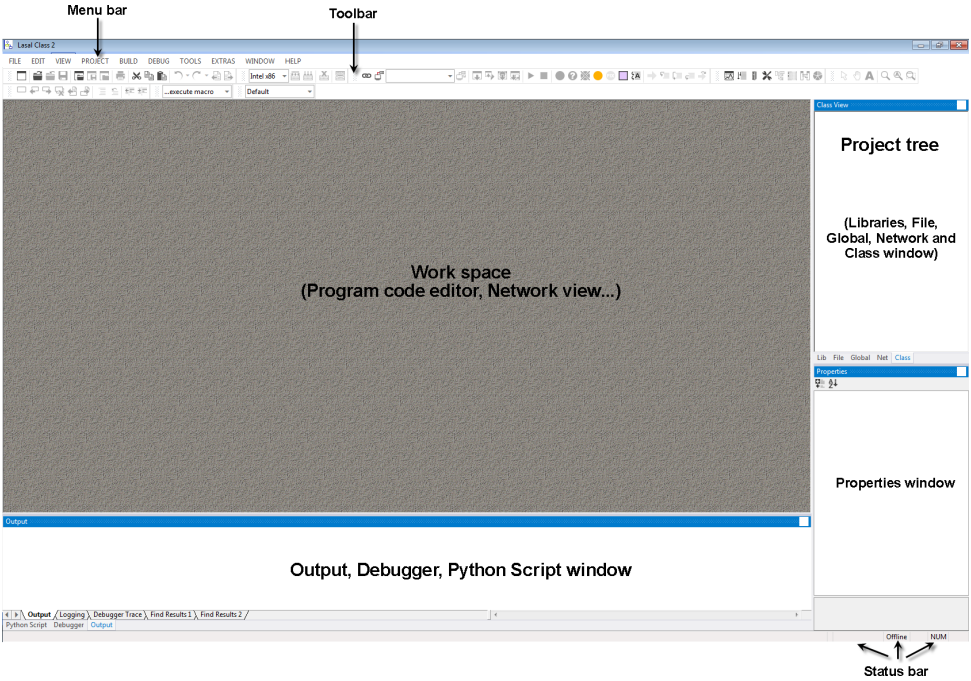
1.10 Disclaimer

In the case of direct or indirect links to third-party websites ("hyperlinks") not included in the responsibilities of SIGMATEK, SIGMATEK cannot be held liable unless it had knowledge of the site’s content and it was technically possible and reasonable for it to prevent the use of any unlawful content contained therein.

SIGMATEK explicitly declares that no illegal content could be identified on the linked websites at the time they were linked. SIGMATEK cannot influence in any way the current or future content or the authorship of the linked websites. SIGMATEK has no influence on the current and future form, content or authorship of the linked/referenced sites. SIMATEK therefore, expressly declines any liability for the content of all linked sites which were changed after the creation of the link. This declaration applies to all links and references within the Online help or documentation, as well as to external entries in the guest books, discussion forums, link directories, mailing lists, and all other types of databases to which external access is possible. The provider of the linked website, and not the party that merely provided the link, is solely liable for illegal, incorrect, or incomplete content on that website, and in particular for damages resulting from the use or non-use of such content.

2 THE DESIGN ENVIRONMENT

User interface of the design environment



2.1 Menu Bar

The menu bar in LASAL CLASS 2 contains the following entries:

File Menu

Edit Menu

NetEdit (active only when an object network is displayed)

DataAnalyzer (active only when the DataAnalyzer is opened)

View Menu

Project Menu

Build Menu

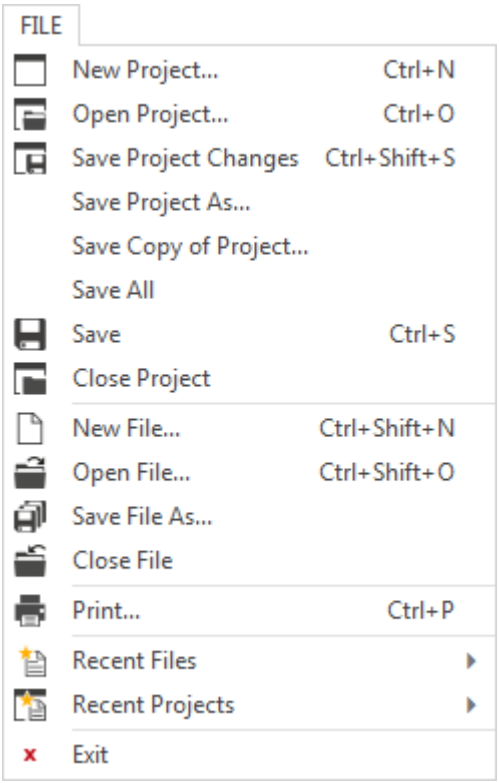
Debug Menu

Tools Menu

Window Menu

Help Menu

2.1.1 File Menu



2.1.1.1 New Project...

Create a new project (Open the **Create New Project window**). For more information, see Creating a New Project in the **My first project** section.

2.1.1.2 Open Project...

Opens an existing project. The desired project can be selected from the **Open a LASAL Projectfile window** and opened.

In comparison with Windows' **File Open window**, this window has an additional function for path presets – see chapter Preset Folders in the Open Project window..

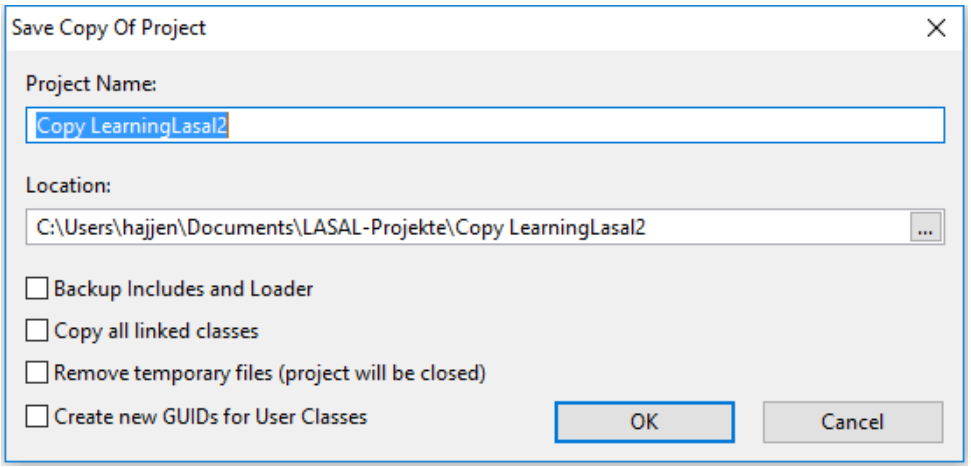
A LASAL project can also be opened by Drag & Drop from the explorer. Here, whether a project is already opened in LASAL (in this case, the project file / directory must be released over the title bar) or not (the project file / directory can be released over the workspace) is determined.

2.1.1.3 Save Project Changes

Saves the changes in the current project.

2.1.1.4 Save Project As...

A copy of the currently open project can be saved under a different name and path. **The new project remains opened.**



Checkbox “Backup Includes and Loader”

If this check box is activated, **all Includes and the loader** are saved with the project under the new name.

Checkbox "Copy all linked classes"

If this checkbox is activated, all linked classes of the project are assumed in as copies (and not as links) in the project copy.

Checkbox "Remove temporary files (project will be closed)"

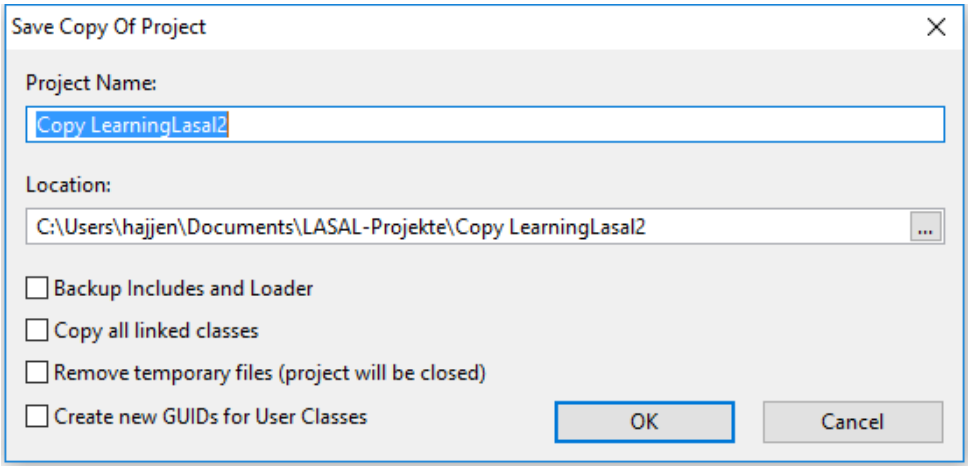
If this check box is activated, a copy of the project is saved without temporary files (e.g. *.LOB). See also: Temporary Files

Checkbox "Create new GUIDs for User Classes"

If this checkbox is enabled, the GUIDs are recalculated for all user classes. This is required so that when using the original projects and the copy under a different name in a Multimaster constellation, no double GUID are generated.

2.1.1.5 Save Copy of Project

A copy of the currently opened project can be saved under a different name and path. **The original project remains opened.**



Checkbox “Backup Includes and Loader”

If this check box is activated, **all Includes and the loader** are saved with the project under the new name.

Checkbox "Copy all linked classes"

If this checkbox is activated, all linked classes of the project are saved as copies (and not as links).

Checkbox "Remove temporary files (project will be closed)"

If this check box is activated, a copy of the project is saved without temporary files (i.e. *.LOB). See also: Temporary Files

Checkbox "Create new GUIDs for User Classes"

If this checkbox is enabled, the GUIDs are recalculated for all user classes. This is required so that when using the original projects and the copy in a Multimaster constellation, no double GUID are generated.

2.1.1.6 Save All

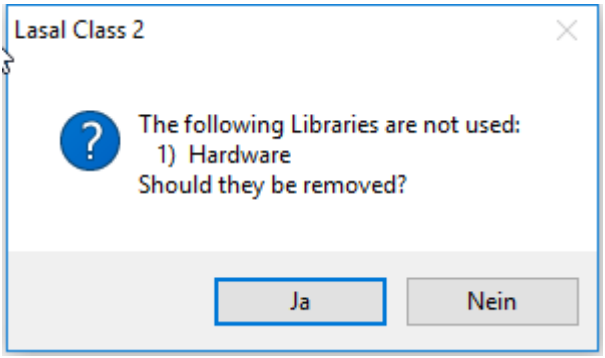
With "Save All" **all project files** and **all opened files** are saved.

2.1.1.7 Save

With "Save" **all changed (modified) project files** and **all changed opened files** are saved.

2.1.1.8 Close Project

Closes the current project. A dialog box then appears, which asks whether the library(ies) should be removed from the project if no class can be linked there from. The next download of the project is therefore faster, since the library must no longer be included.



2.1.1.9 New File...

Opens a new file for editing in the program code editor. The name and ending of the file can be defined when saving.

2.1.1.10 Open File...

Opens an existing file (opens the **file open window** so that the file can be selected).

2.1.1.11 Save File As...

The currently opened file can be saved under a different name and path.

2.1.1.12 Close File

Closes the current file.

2.1.1.13 Print...

Opens the print menu for the selected object (network, class and so on).

2.1.1.14 Recent Files

Shows a list of the most recently opened files.

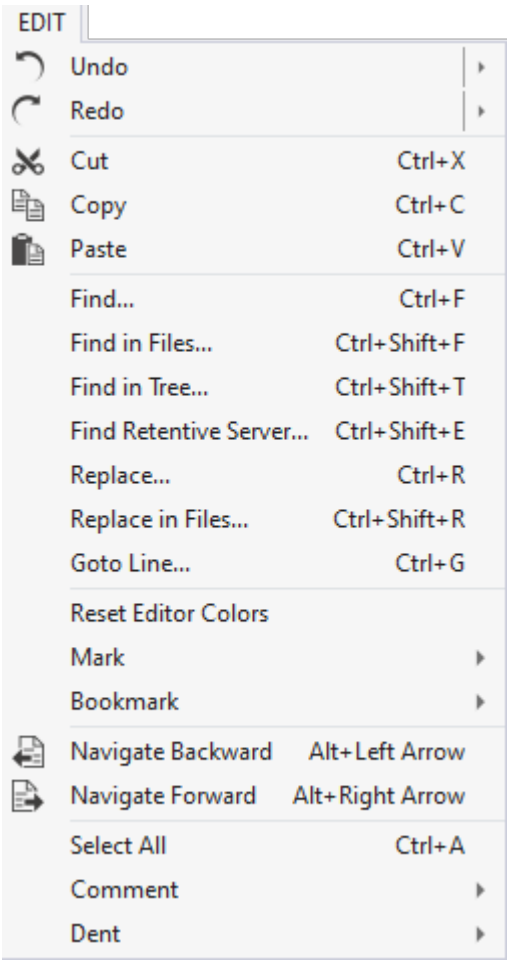
2.1.1.15 Recent Projects

Displays a list of the most recently opened projects.

2.1.1.16 Exit

Closes LASAL CLASS 2. If the changes were made to the project, which were not saved, a request to save the project appears before it closes.

2.1.2 Edit Menu



2.1.2.1 Undo

Reverses the last action.

2.1.2.2 Redo

Reinstates an action that was reversed.

2.1.2.3 Cut

Removes the selected text (the marked object).

2.1.2.4 Copy

Copies the selected text or object to the buffer.

2.1.2.5 Paste

Inserts the text or object from the buffer in the selected position.

2.1.2.6 Find...

Opens the Find window, in this window words or text can be found. If an element / text was previously marked / selected in any one of the windows (class tree, network, code editor…), the marked text is inserted into the search field.

2.1.2.7 Find in Files...

Opens the Find in Files window. Using this window, words or text can be found in specific files. If an element / text was previously marked / selected in any one of the windows (class tree, network, code editor…), the marked text is inserted into the search field.

2.1.2.8 Find in Tree...

Opens the Find in Tree window, with which the project tree can be searched for elements

2.1.2.9 Find Retentive Server

Opens the Find Retentive Server window, which is used to search for zero-voltage proof servers.

2.1.2.10 Replace...

Opens the Replace window, which searches for words or text and then replaces them with other words or text.

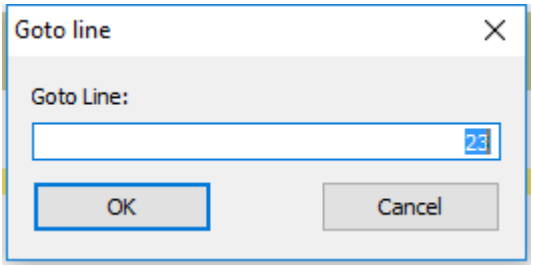
2.1.2.11 Replace in Files...

Opens the Replace in Files window with which words or text in a defined file are located and replaced.

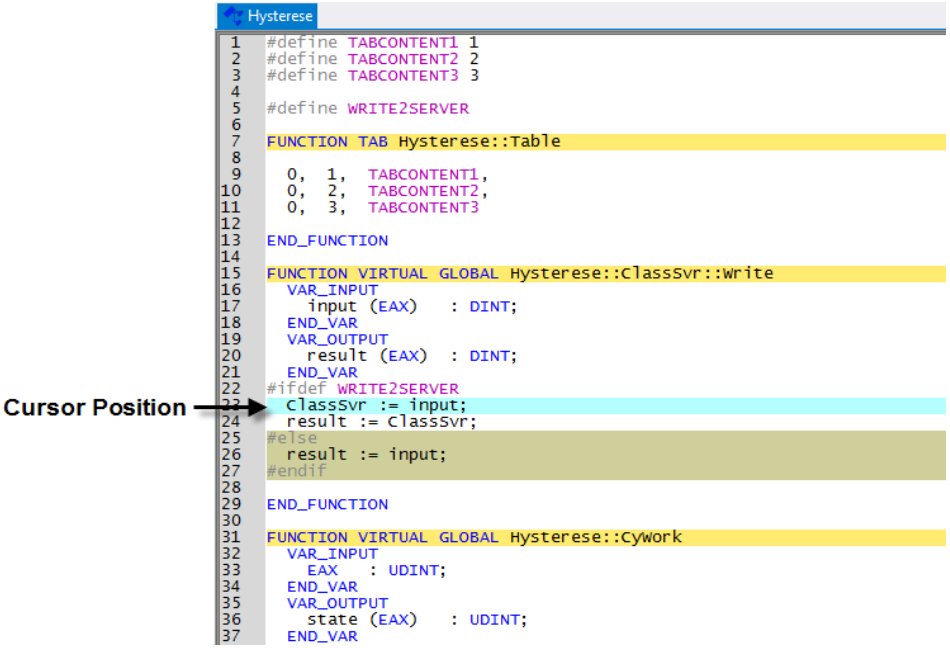
2.1.2.12 Goto Line...

With this function, it is possible to jump to a specific line in the program code editor. This function can also be called with the shortcut **Ctrl +G**.

In the opened Goto Line window, enter the line in the program code to which the cursor could jump (in the example, 23). Next confirm with **OK**.



As can be seen in the image below, the cursor is now located at the beginning of the 23rd program line. Which for better orientation, is marked with light blue.

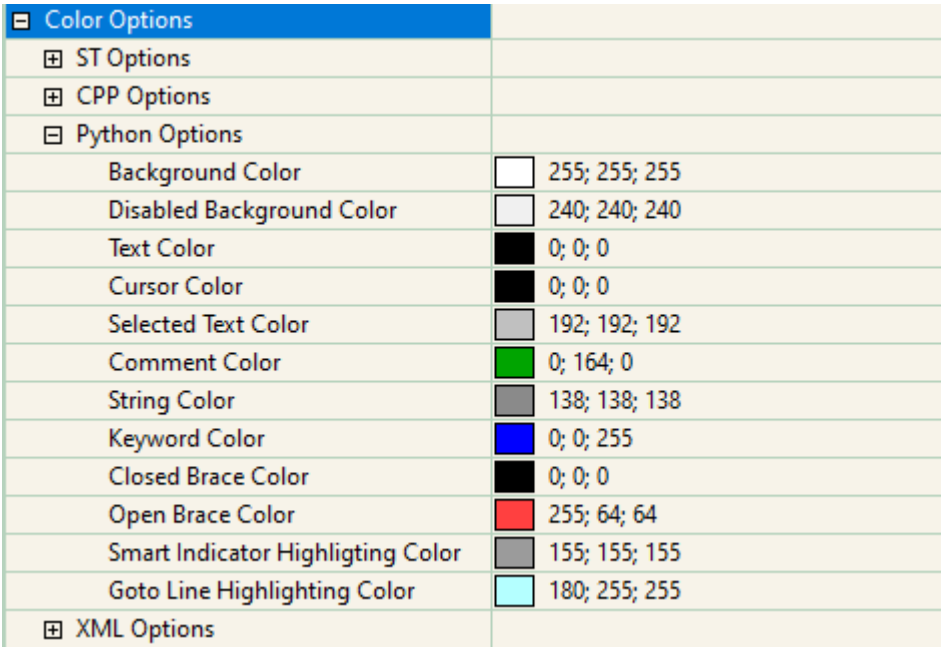

Hysterese
Cursor Position
1 #define TABCONTENT1 1
2 #define TABCONTENT2 2
3 #define TABCONTENT3 3
4
5 #define WRITE2SERVER
6
7 FUNCTION TAB Hysterese::Table
8
9 0, 1, TABCONTENT1,
10 0, 2, TABCONTENT2,
11 0, 3, TABCONTENT3
12
13 END_FUNCTION
14
15 FUNCTION VIRTUAL GLOBAL Hysterese::ClassSvr::Write
16 VAR_INPUT
17 input (EAX) : DINT;
18 END_VAR
19 VAR_OUTPUT
20 result (EAX) : DINT;
21 END_VAR
22 #ifdef WRITE2SERVER
23 ClassSvr := input;
24 result := ClassSvr;
25 #else
26 result := input;
27 #endif
28
29 END_FUNCTION
30
31 FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
32 VAR_INPUT
33 EAX : UDINT;
34 END_VAR
35 VAR_OUTPUT
36 state (EAX) : UDINT;
37 END_VAR

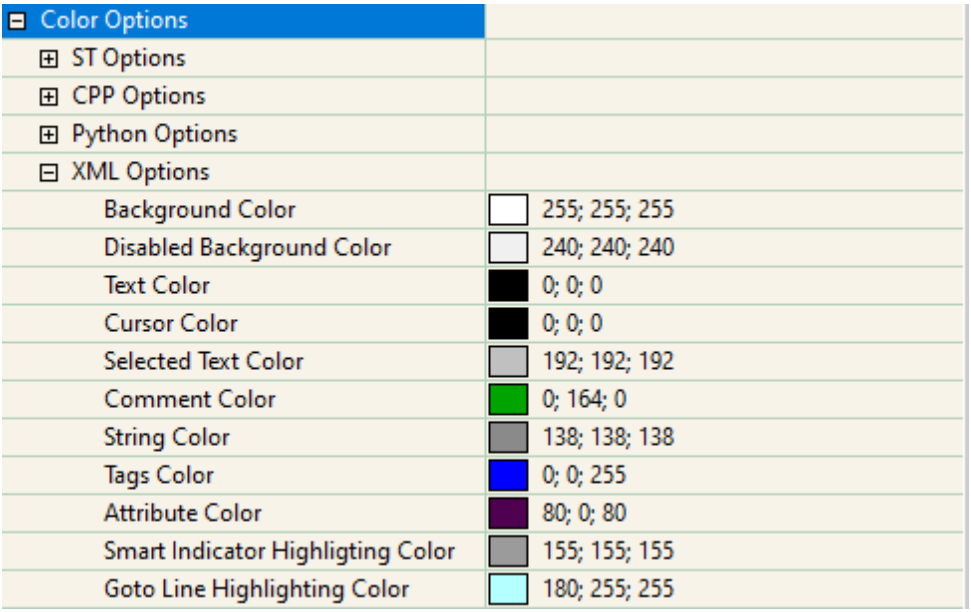
2.1.2.13 Reset Editor Colors

This menu item **resets all colors** (text, background, highlighting colors) **to the default settings**.

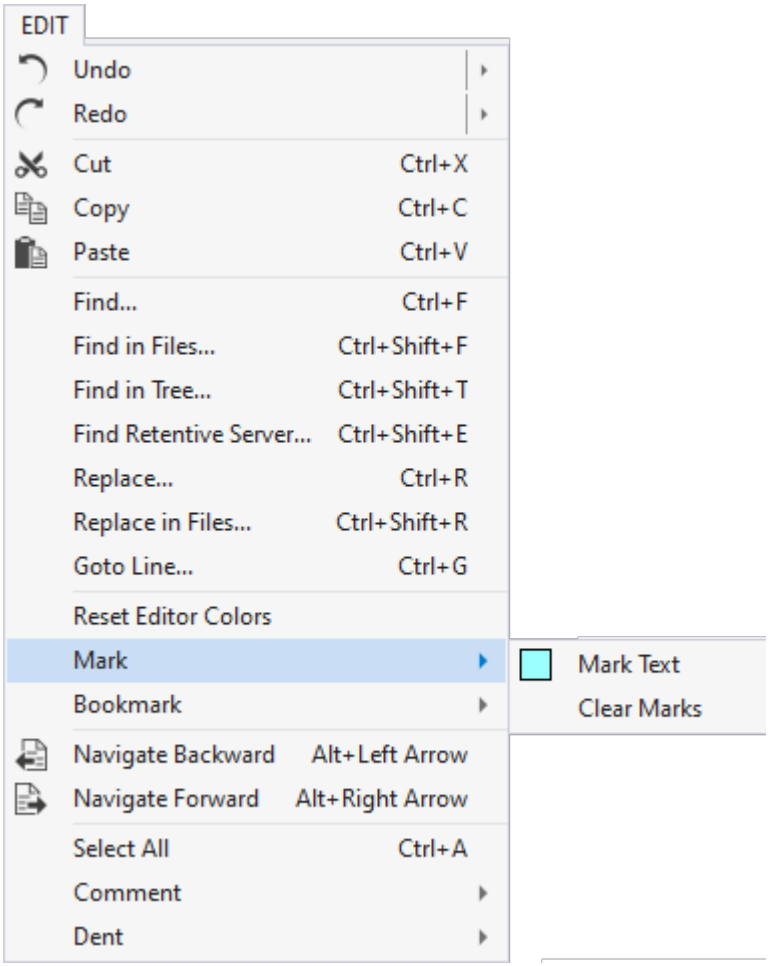
Color Options	
ST Options	
Background Color	255; 255; 255
Disabled Background Color	240; 240; 240
Text Color	0; 0; 0
Cursor Color	0; 0; 0
Selected Text Color	192; 192; 192
Comment Color	0; 164; 0
Comment GUID Color	255; 182; 0
Comment Language ID Color	255; 128; 0
String Color	138; 138; 138
Keyword Color	0; 0; 255
Preprocessor Color	138; 138; 138
Defines Color	180; 0; 180
Pointer Color	138; 138; 138
Function Header Highlighting Color	255; 235; 112
Closed Brace Color	0; 0; 0
Open Brace Color	255; 64; 64
Smart Indicator Highligting Color	155; 155; 155
Closed Indicator Highlighting Color	155; 155; 255
Open Indicator Highlighting Color	255; 155; 155
Inactive Code Color	207; 207; 156
Goto Line Highlighting Color	180; 255; 255
CPP Options	
Python Options	
XML Options	

⊟ Color Options	
⊞ ST Options	
⊟ CPP Options	
Background Color	255; 255; 255
Disabled Background Color	240; 240; 240
Text Color	0; 0; 0
Cursor Color	0; 0; 0
Selected Text Color	192; 192; 192
Comment Color	0; 164; 0
String Color	138; 138; 138
Keyword Color	0; 0; 255
Preprocessor Color	138; 138; 138
Closed Brace Color	0; 0; 0
Open Brace Color	255; 64; 64
Smart Indicator Highligting Color	155; 155; 155
Goto Line Highlighting Color	180; 255; 255
⊞ Python Options	
⊞ XML Options	


Color Options
ST Options
CPP Options
Python Options
Background Color
255; 255; 255
Disabled Background Color
240; 240; 240
Text Color
0; 0; 0
Cursor Color
0; 0; 0
Selected Text Color
192; 192; 192
Comment Color
0; 164; 0
String Color
138; 138; 138
Keyword Color
0; 0; 255
Closed Brace Color
0; 0; 0
Open Brace Color
255; 64; 64
Smart Indicator Highligting Color
155; 155; 155
Goto Line Highlighting Color
180; 255; 255
XML Options


Color Options
ST Options
CPP Options
Python Options
XML Options
Background Color
255; 255; 255
Disabled Background Color
240; 240; 240
Text Color
0; 0; 0
Cursor Color
0; 0; 0
Selected Text Color
192; 192; 192
Comment Color
0; 164; 0
String Color
138; 138; 138
Tags Color
0; 0; 255
Attribute Color
80; 0; 80
Smart Indicator Highligting Color
155; 155; 155
Goto Line Highlighting Color
180; 255; 255

2.1.2.14 Mark



With this feature you can permanently mark all occurrences of text on which the cursor is in the code editor.

Hysterese*

```
31  FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
32    VAR_INPUT
33      EAX    : UDINT;
34    END_VAR
35    VAR_OUTPUT
36      state (EAX)    : UDINT;
37    END_VAR
38    VAR
39      cAlarm     : ARRAY[0..255] OF CHAR;
40      pAlarm     : ^CHAR;
41      cEvent     : ARRAY[0..255] OF CHAR;
42      pEvent     : ^CHAR;
43    END_VAR
44
45
46
47
48    // *** Read the actual value from the peripherie ***
49    ActualValue :=  ActualValue.Read();
50    MaxValue    :=  MaxValue.Read();
51    MinValue    :=  MinValue.Read();
52
53    if Begin = 0 then
54      TRACE("Start der Messages");
55      Begin := 1;
56    end_if;
57
58    // *** Control hnow the Alarm and Pumpcontrol ***
59    if (MinValue >= MaxValue) then
60      Alarm := 1;
61    else
62      Alarm := 0;
63    end_if;
64
65    if (ActualValue := MaxValue) then
66      Event := 1;
67    elsif (ActualValue <= MinValue) then
68      Event := 0;
69    end_if;
70
71    pAlarm := sigclib_itoa(Alarm, #cAlarm[0], 10);
72    pEvent := sigclib_itoa(Event, #cEvent[0], 10);
73
74    // *** Messaging ***
75
76    if Alarm := 1 then
77      TRACE1_ERR("Alarm ist {0}, Event ist {1}", pAlarm, pEvent);
78    elsif Event = 1 then
79      TRACE1_WARN("Alarm ist {0}, Event ist {1}", pAlarm, pEvent);
80    else
81      TRACE("Alarm ist {0}, Event ist {1}", pAlarm, pEvent);
82    end_if
83
84    Event.Write(Event);
85    Alarm.Write(Alarm);
```

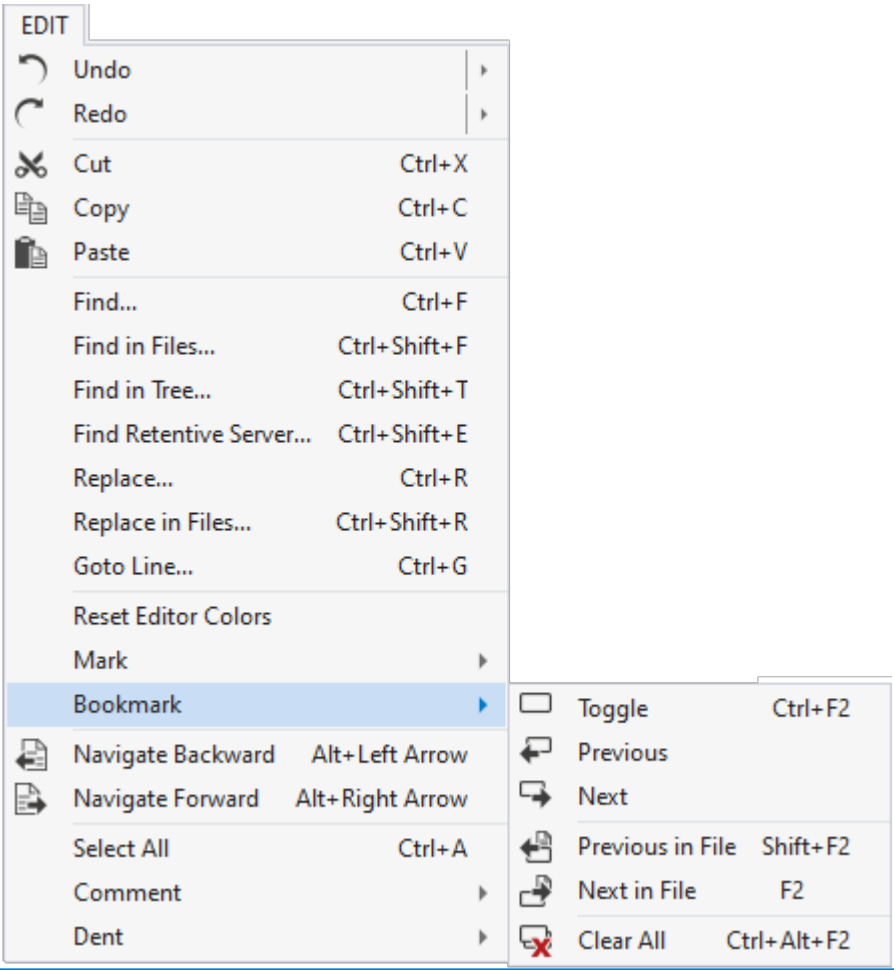
Mark Text

Permanently highlights all occurrences in the open editor. Several texts can be marked one after another.

Clear Marks

Deletes all permanent markings, which were labeled with the **Mark Text** function.

2.1.2.15 Bookmark



Toggle	Sets/clears a bookmark in the actual line of the opened file.
Previous	Jumps to the previous bookmark, even between files.
Next	Jumps to the next bookmark, even between files.
Previous in File	Jumps to the previous bookmark in the file.

Next in File	Jumps to the next bookmark in the file.
Clear All	Clears all bookmarks in the file.

2.1.2.16 Navigate Backward

Jumps to the last edited / marked / selected / jumped-to location in the code; even when it is located in a different file..

2.1.2.17 Navigate Forward

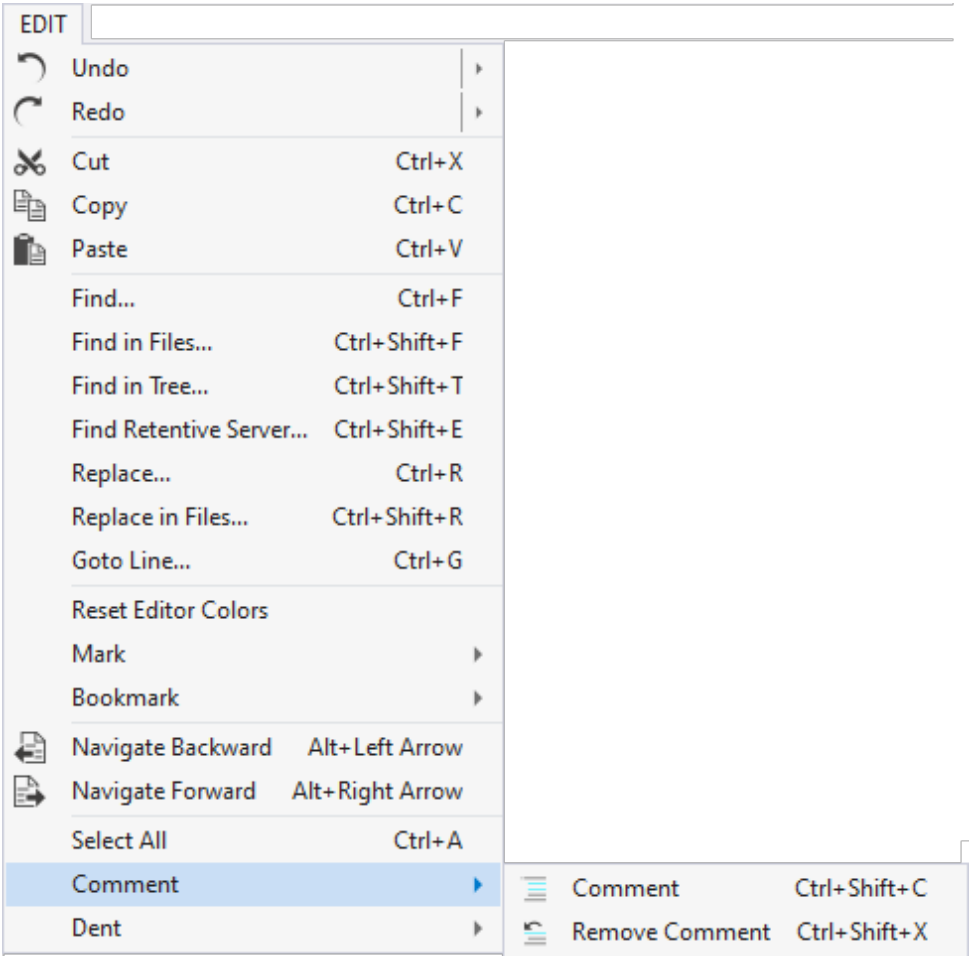
Jumps to the next edited / marked / selected / jumped-to location in the code; even when it is located in a different file.

2.1.2.18 Select All

Selects the entire text of the opened file in the program code editor.

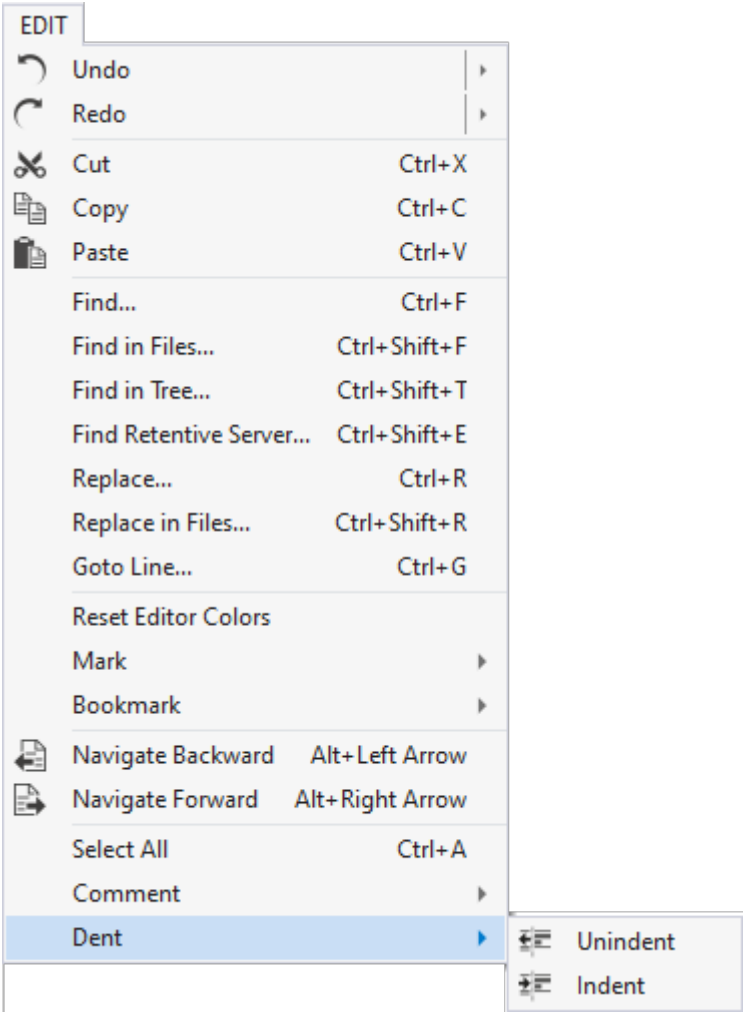
2.1.2.19 Comment

Comment the selected section of text out or back in.

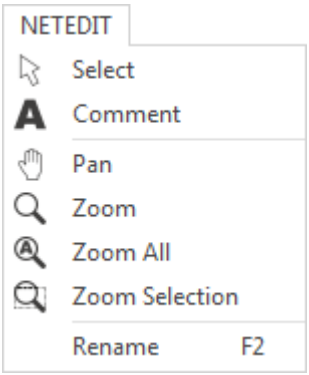


2.1.2.20 Dent

Shifts the selected section of text by on tabulator position in (to the right, **Indent**) or out (to the left, **Unindent**).

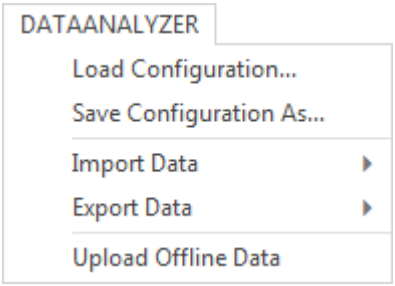


2.1.3 NetEdit (active only when an object network is displayed)



Select	Selection tool to select objects in the network.
Comment	Select this tool to add comments in the network.
Pan	Hand tool for navigation on the network.
Zoom	Zoom tool to zoom in, respectively zoom out of objects on the network.
Zoom All	Zooms all objects on the network so that they are visible.
Zoom Selection	Zooms a desired sector on the network.
Rename	Serves to rename the selected object name.

2.1.4 DataAnalyzer (active only when the DataAnalyzer is opened)



2.1.4.1 Load Configuration

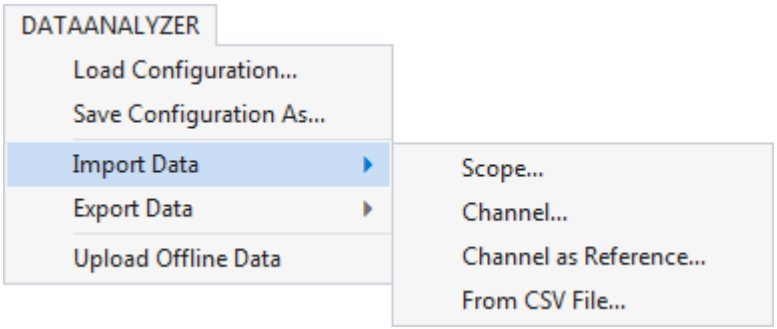
Loads a DataAnalyzer configuration.

2.1.4.2 Save Configuration As

Saves the current configuration with a defined name.

2.1.4.3 Import Data

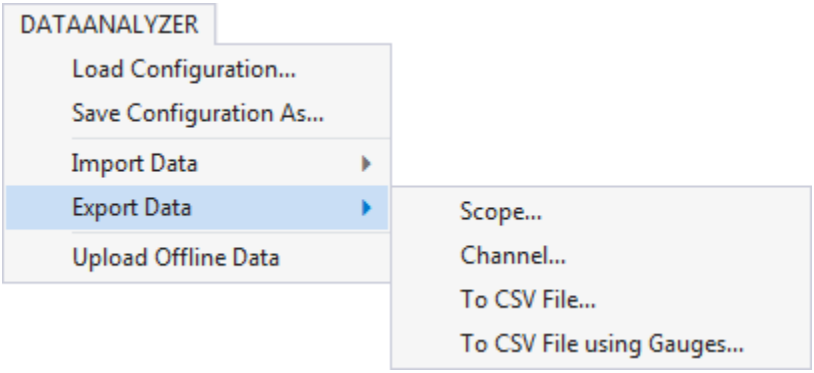
Imports DataAnalyzer data from the hard disk.



Scope	Scope data (*.lds) are imported.
Channel	Channel data (*.ldc) are imported.
Channel As Reference	Reference curve data are imported (*.ldc as well).
From CSV-File	*.csv (comma separated values files) data are imported.

2.1.4.4 Export Data

Exports DataAnalyzer data to the hard disk.



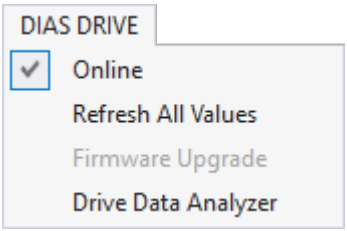
Scope	Scope data is exported in *.lds format.
Channel	Channel data is exported in *.ldc format.
To CSV-File	Data is exported in *.csv (comma separated values files) format
To CSV-File using Gauges	Data is exported in *.csv format, whereby only the area between the help lines is stored.

2.1.4.5 Upload Offline Data

If the DataAnalyzer is started and then goes offline, data recording continues in the background. When back online at later time, this menu item is enabled and can be used to retrieve the data that was recorded while offline.

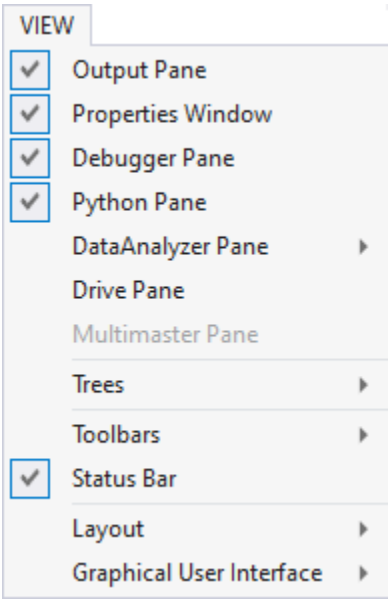
More information can be found under DataAnalyzer.

2.1.5 Dias Drive (active only when the Dias drive is opened)



Online	To establish a VARAN connection with a DiasDrive, it is first necessary to connect to a so-called "VARAN Master" (C-IPC, LARS, ...) via the online dialog. A dialog can then be opened via the DiasDrive, Online menu, which lists a selection of possible target devices. You can find more information on this under: Online configuration in the Dias Drive chapter.
Refresh All Values	If you select this menu item, all online values are read or updated from the drive.
Firmware Upgrade	This menu item is disabled by default and is only activated if the required password has been entered under Extras / Service Password
Drive Data Analyzer	There is a separate DataAnalyzer for DiasDrive, which is started here. For more information see: DataAnalyzer for DiasDrive

2.1.6 View Menu



2.1.6.1 Output Pane

Activates/deactivates the checks to display or hide the Output window.

2.1.6.2 Properties Window

Activates/deactivates the checks to display or hide the Properties window.

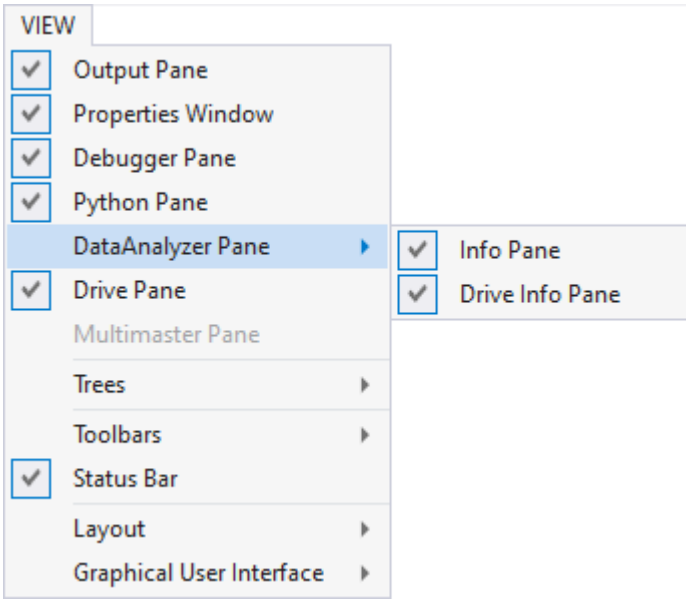
2.1.6.3 Debugger Pane

Activates/deactivates the checks to display or hide the Debugger window.

2.1.6.4 Python Pane

Activates/deactivates the checks to display or hide the Python window.

2.1.6.5 Data Analyzer Pane



Info Pane	Activates/deactivates the checkbox to show or hide the **Data Analyzer Information Pane**.  Can only be selected when the Data Analyzer is switched on!
Drive Info Pane	Activates/deactivates the checkbox to show or hide the **Data Analyzer (Drive) Information Pane**.  Can only be selected when DiasDrive is switched on!

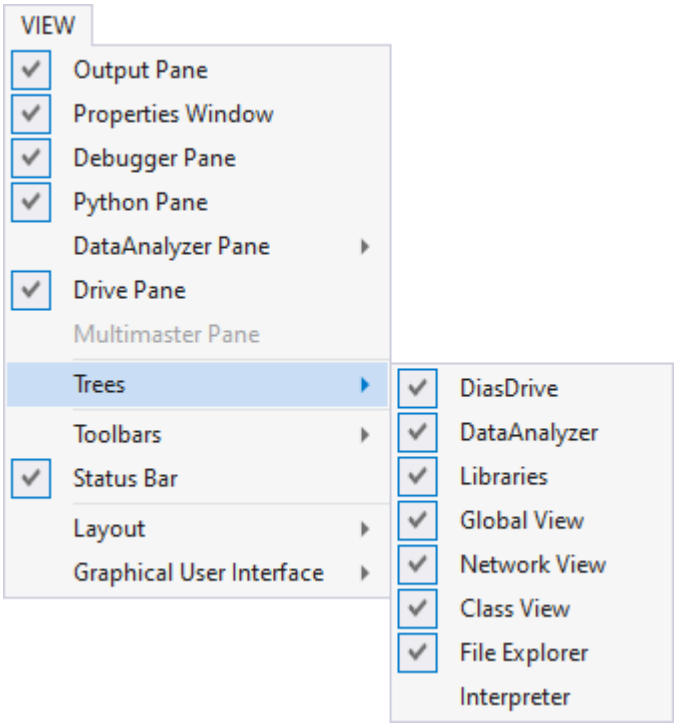
2.1.6.6 Drive Pane

Activates/deactivates the checkmark to display or hide the Dias Drive command terminal window.

2.1.6.7 Multimaster Pane

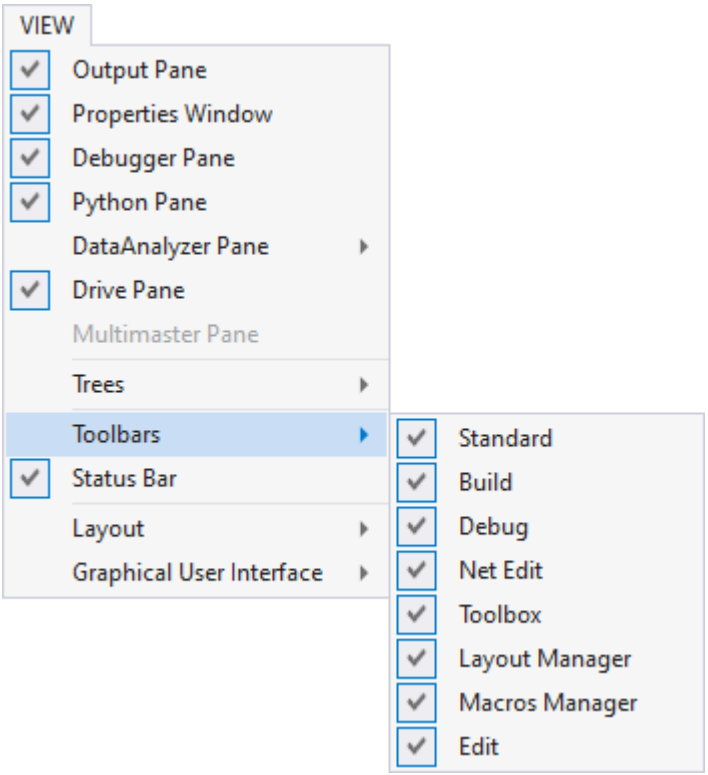
Activate/deactivate the checkmark, to show or hide the Multimaster Pane.

2.1.6.8 Trees



Activate or deactivate the respective check to display or hide the individual **project trees** (**DiasDrive, DataAnalyzer, Libraries, Global View, Network View, Class View, File Explorer and Interpreter window**).

2.1.6.9 Toolbars

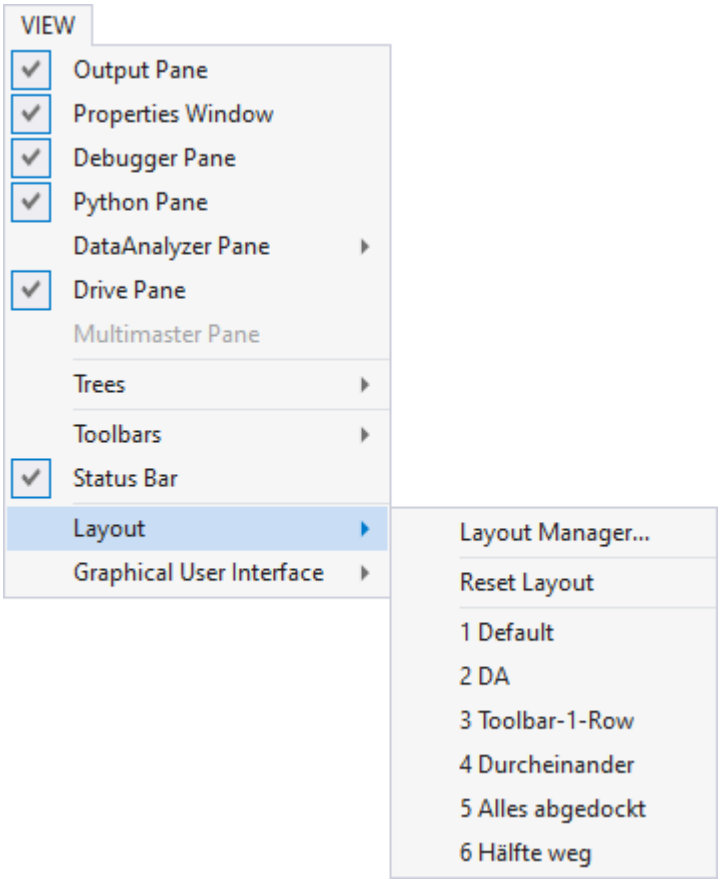


Activate or deactivate the respective check to display or hide the individual tool bars (**Standard, Build, Debug, Net Edit, Toolbox, Layout Manager, Macros Manager, Edit**). A listing and definition of the tool bare are found under Tool Symbol bars (overview).

2.1.6.10 Status bar

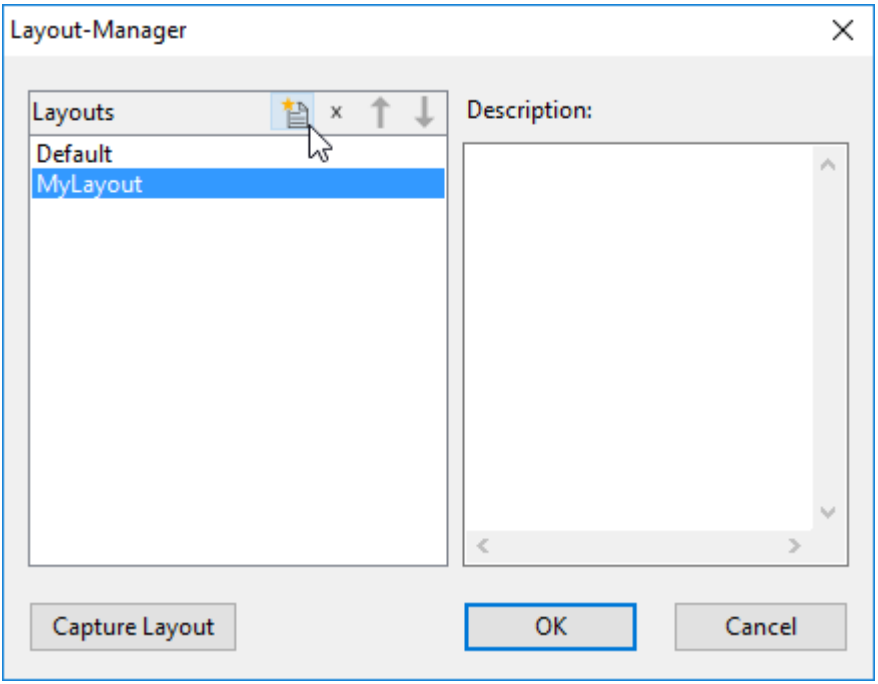
Activate or deactivate the respective check to display or hide the **Status bar**.

2.1.6.11 Layout



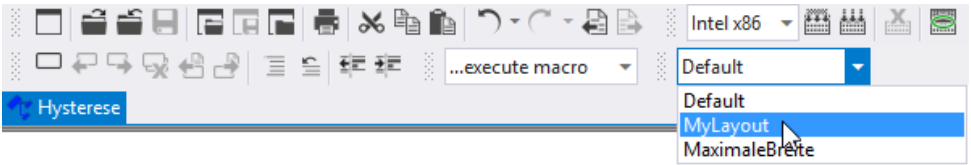
Layout Manager

With the Layout Manager, a (or several) previously created individual layout can be stored and organized. To create a new layout, open the Layout manager … and the following dialog box will appear:



With **New**, specify the name for the new layout and with **Capture Layout**, the previously created layout (here: **MyLayout**) is then saved. Exit the Layout Manager with **OK**.

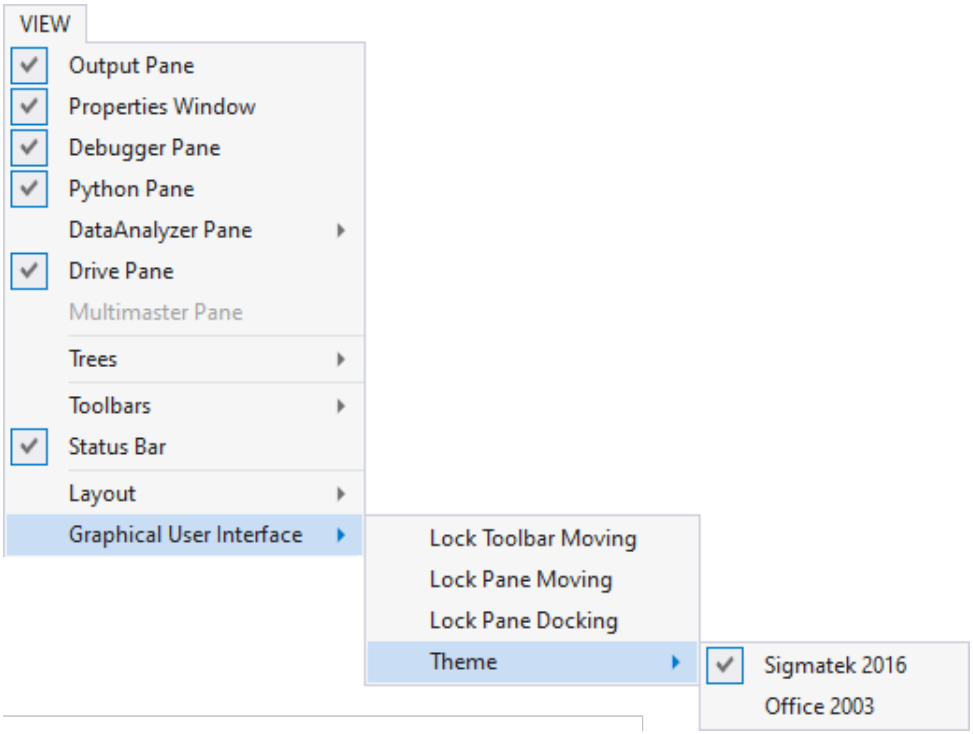
The Tools toolbar can now be used to switch between the assigned layouts:



Reset Layout

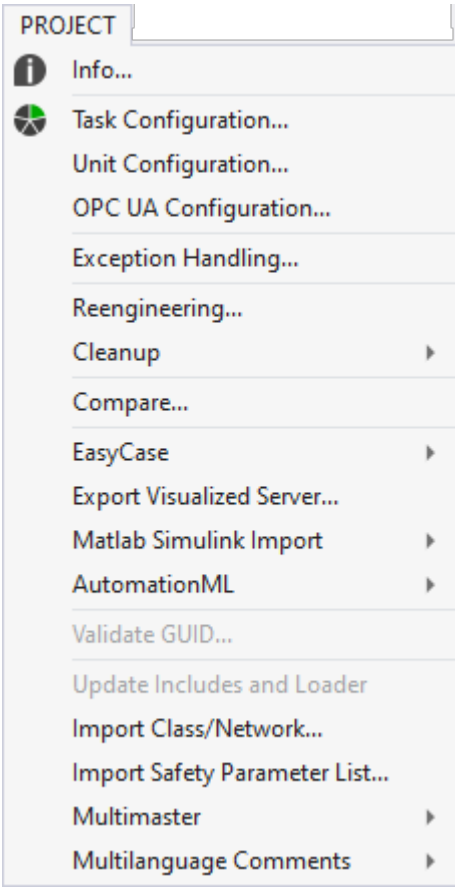
Resets the default window layout of the LASAL CLASS 2 surface (if while working, various windows were shifted).

2.1.6.12 Graphical User Interface



Activate/deactivate the respective check mark to enable or disable the moving or docking of the toolbars or panes. Moreover you can choose several display themes. The selected theme has no influence to the functionality, but only to the visual display.

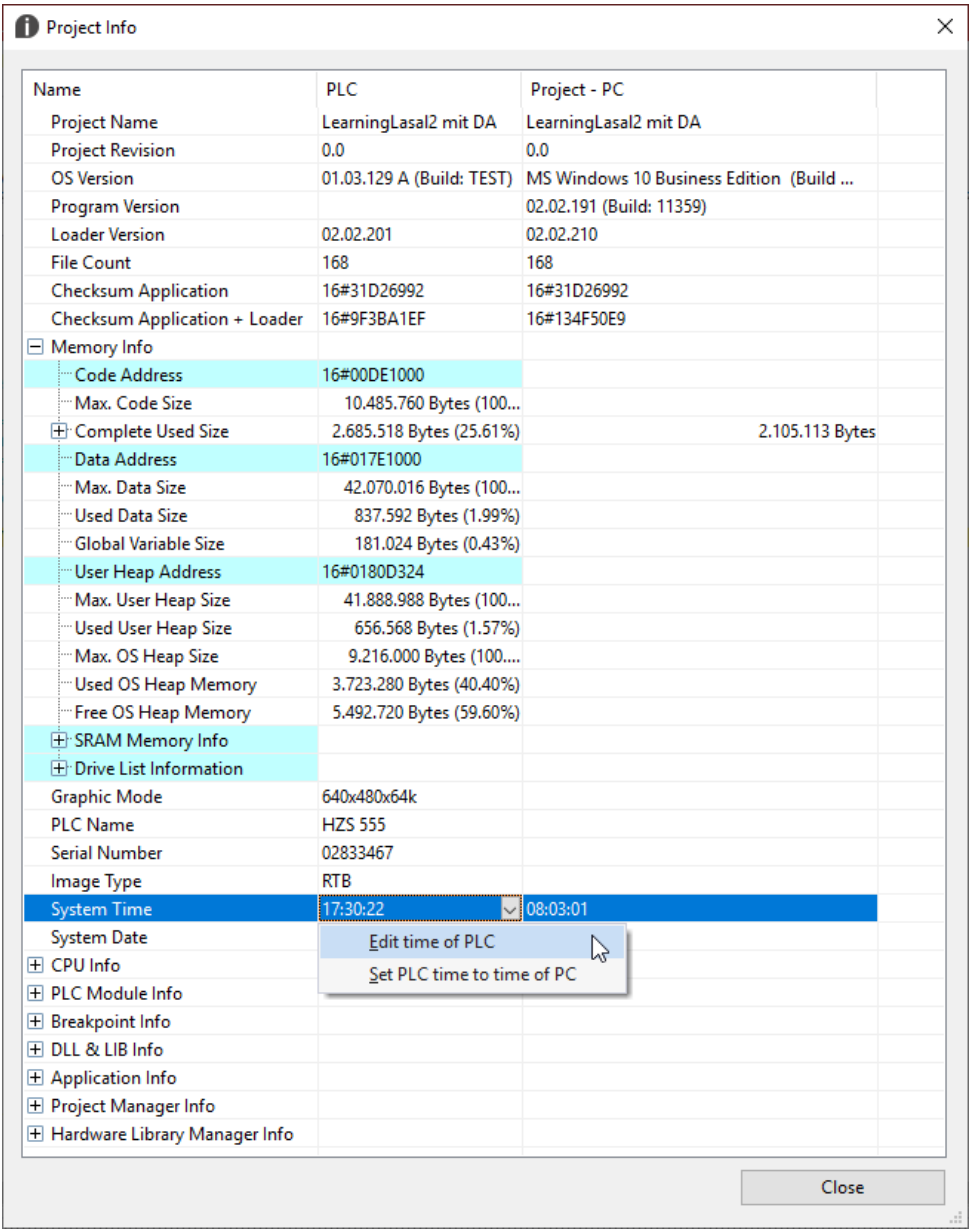
2.1.7 Project Menu



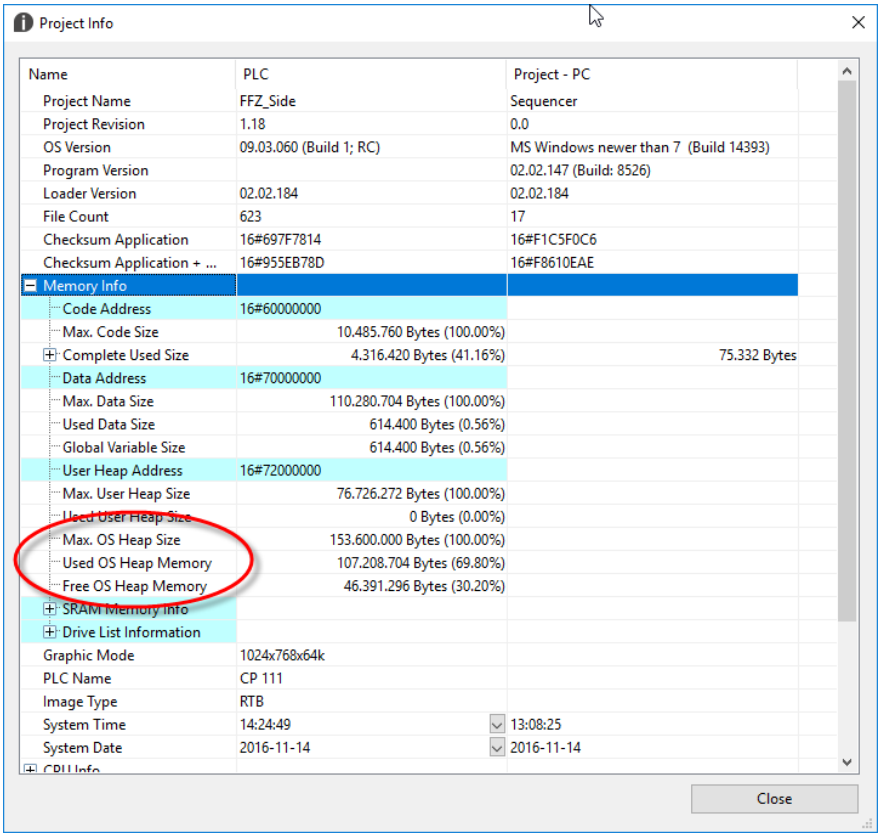
2.1.7.1 Info

For a listing of project information, select **Project, Info** from the menu bar.

In the **Project Info window**, information is displayed which cannot be changed, except PLC Time and PLC date.


Project Info
Name
PLC
Project - PC
Project Name
LearningLasal2 mit DA
LearningLasal2 mit DA
Project Revision
0.0
0.0
OS Version
01.03.129 A (Build: TEST)
MS Windows 10 Business Edition (Build ...
Program Version
02.02.191 (Build: 11359)
Loader Version
02.02.201
02.02.210
File Count
168
168
Checksum Application
16#31D26992
16#31D26992
Checksum Application + Loader
16#9F3BA1EF
16#134F50E9
Memory Info
Code Address
16#00DE1000
Max. Code Size
10.485.760 Bytes (100...
Complete Used Size
2.685.518 Bytes (25.61%)
2.105.113 Bytes
Data Address
16#017E1000
Max. Data Size
42.070.016 Bytes (100...
Used Data Size
837.592 Bytes (1.99%)
Global Variable Size
181.024 Bytes (0.43%)
User Heap Address
16#0180D324
Max. User Heap Size
41.888.988 Bytes (100...
Used User Heap Size
656.568 Bytes (1.57%)
Max. OS Heap Size
9.216.000 Bytes (100....
Used OS Heap Memory
3.723.280 Bytes (40.40%)
Free OS Heap Memory
5.492.720 Bytes (59.60%)
SRAM Memory Info
Drive List Information
Graphic Mode
640x480x64k
PLC Name
HZS 555
Serial Number
02833467
Image Type
RTB
System Time
17:30:22
08:03:01
System Date
Edit time of PLC
Set PLC time to time of PC
CPU Info
PLC Module Info
Breakpoint Info
DLL & LIB Info
Application Info
Project Manager Info
Hardware Library Manager Info
Close

This information affects the destination system (**PLC**) and the PC (**Project – PC**). Since OS version 09.03.060 also the OS Heap is displayed in the column PLC.



Project Name	Name of the Project.
Project Version	Project version.
Version OS	Operating system version.
Version Loader	Version of the loader in the target system.
Version Program	LASAL CLASS 2 version.

File Count	Number of LOB files.	
Checksum	Address used to compare the project in the PC with that in the target system.	
Memory Info	Code Address	The address of the program code in the target system.
	Max. Code Size	Gives the available space in the target system for the program code. The information in the PC shows the actual size of the program code in bytes (the code size of the loader is not included (unless the project loaded is the loader itself).
	Complete Used Size/Loader Size	The actual size of the loader.
	Complete Used Size/Application Size	The actual size of the application.
	Data Address	The start address of the data in the target system.
	Max. Data Size	Gives the maximum data memory available in the target system.
	Used Data Size	Gives the total data memory used by the program. This value is comprised of the used heap memory and the size of the global variables.
	Global Variable Size	Gives the size of the data memory that is used by the global variables.
	Max. User Heap Size	This value corresponds to the total available data memory minus the size of the global variables.
	Used User Heap Size	Gives the size of the used Heap memory.
	SRAM Memory Info/Version	Indicates the version of the storage mechanism for the zero-voltage proof memory.
	SRAM Memory Info/Max Size	Indicates the size of the zero-voltage proof memory, including the SRAM hearder (64 bytes).
	SRAM Memory Info/Free Size	Indicates the free zero-voltage proof memory size.  **Attention** Since in SRAMFORMAT 1, the control structures are stored in the SRAM and the SRAM storage has a certain fragmentation, the value displayed here is only an approximation. This means that from this value, the number of RAM or RAMEX objects that can still be created cannot be clearly defined!
	SRAM Memory Info/Used Size	Indicates the size of the used zero-voltage proof memory, including the SRAM hearder (64 bytes).

	SRAM Memory Info/Valid: Indicates the validity of the data stored in the zero-voltage memory. (**true** = the data received is valid)
Graphic Mode	Shows the graphic mode set in the target system.
PLC Name	Name of the target system.
Serial Number	Serial number of the CPU, as long as the output from the operating system is supported.
Image Type	Shows the file type (file ending, here RTB) of the operating system image in the target system.
System Time	Displays the System time. In online, the arrow can be clicked and the system time can be set in the control from the context menu with Edit Time of PLC or with Set PLC time to time of PC, the PLC system time can be set to that in the PC.
System Date	Displays the system date. If online, the arrow can be clicked and the system date can be set in the control from the context menu with Edit Date of PLC or with Set PLC date to date of PC, the date in the control can be set to that in the PC.
CPU Info	Gives information on the actual CPU load.
PLC Module Info	Provides information on the individual code modules (*.awl, *.st, *.c etc.) of the project in the PLC.
Breakpoint Info	Shows information over set breakpoints.
DLL & LIB Info	Shows information over the version status of DLL's and libraries used in LASAL 2.
Application Info	Shows information of the LASAL 2 application such as the location of the exe file, the temporary path used as well as the path of the Log and SysInfo files. With a right click on **Log Filename** and **SysInfo Filename**, the corresponding file can be displayed or the windows explorer with the desired path can be opened.
Project Manager Info	Shows information over the currently loaded project (number of classes, networks, header files, objects, server and clients).
Hardware Library Manager Info	Shows information on the actual hardware configuration (Name, file / path, hardware library: true/false, connected project and its file name/ path, manually loaded: yes/no).



Note:

If the mouse stops over a field, a tooltip appears with an explanation of the meaning.

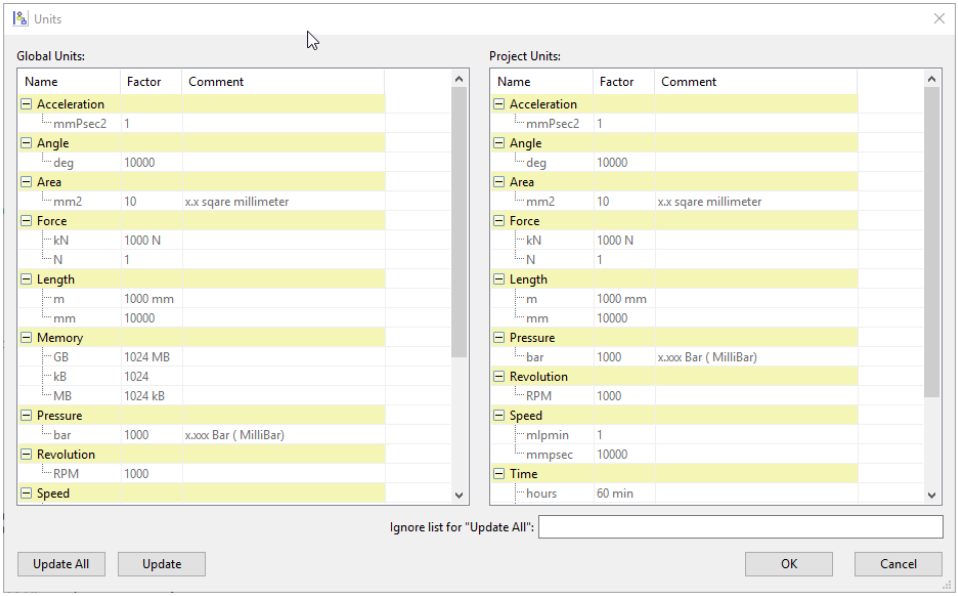
2.1.7.2 Task Configuration

Open **Task Configuration window** to manage Realtime, Cyclic and Background Objects. Further information can be found under Online configuration tasks.

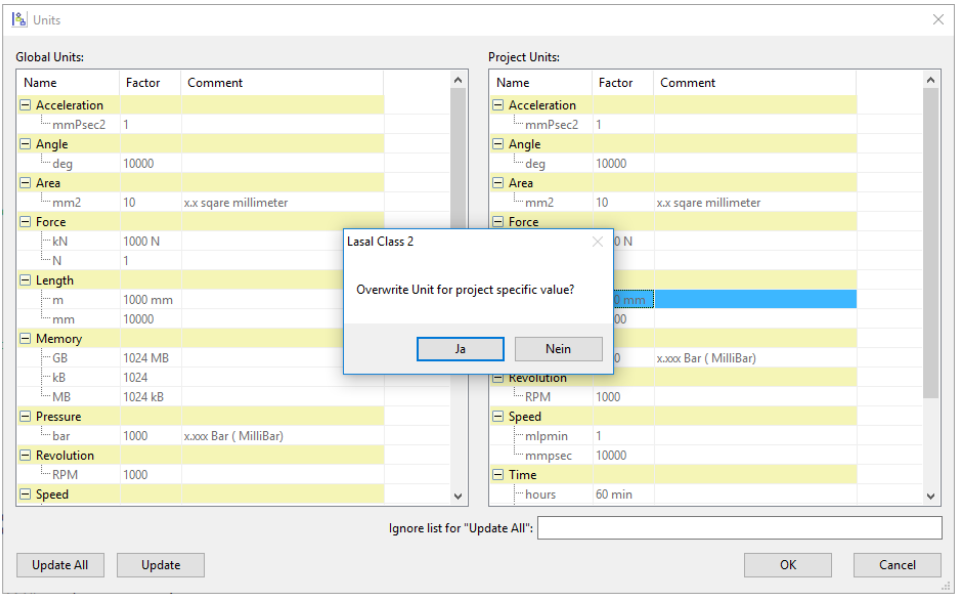
2.1.7.3 Unit Configuration

Opens the Unit Configuration window with which the units are managed.

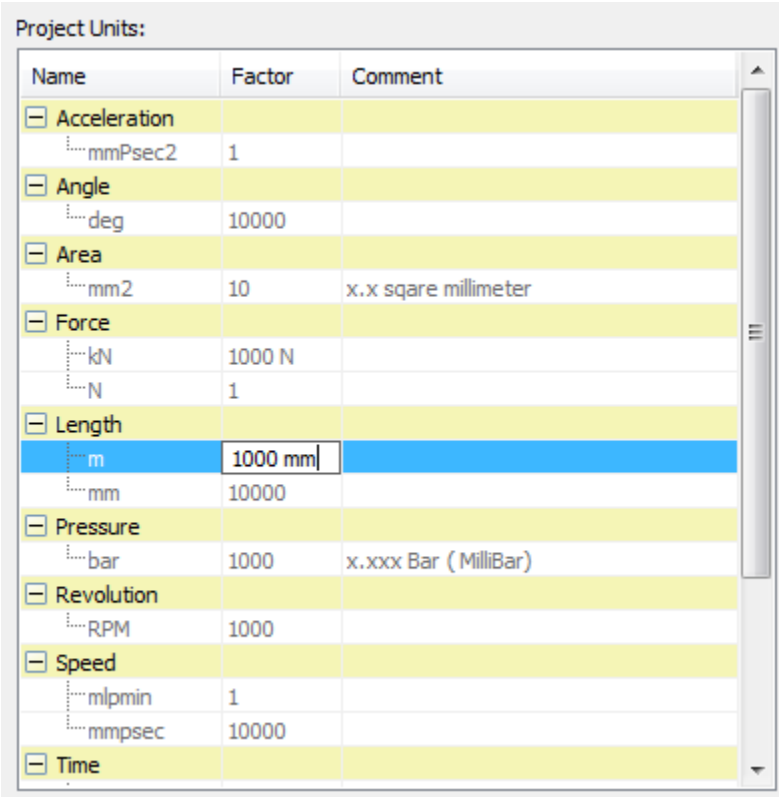
The **Unit Configuration window** is divided in two. The **left side** contains the preconfigured **Global Units**; on the **right** side, the units **used in the project** are shown. After creating a new project, both sides are identical.



The Global Units cannot be changed. The **Project Units** can be overwritten with a double click. In this case, the user is asked whether they should really be overwritten:

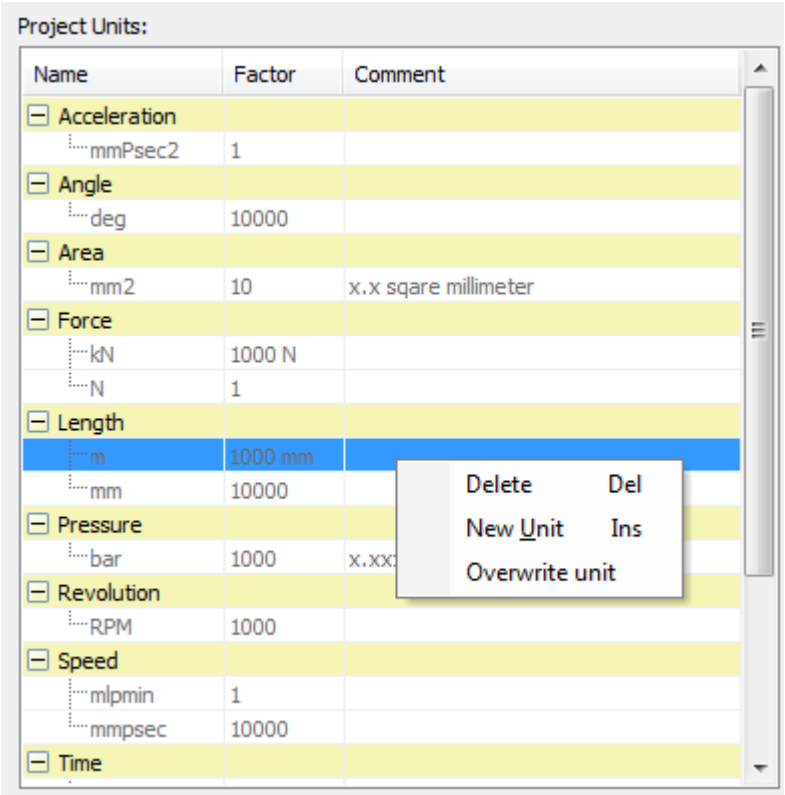


When confirmed with **Yes**, the field is editable and the desired change can be made.



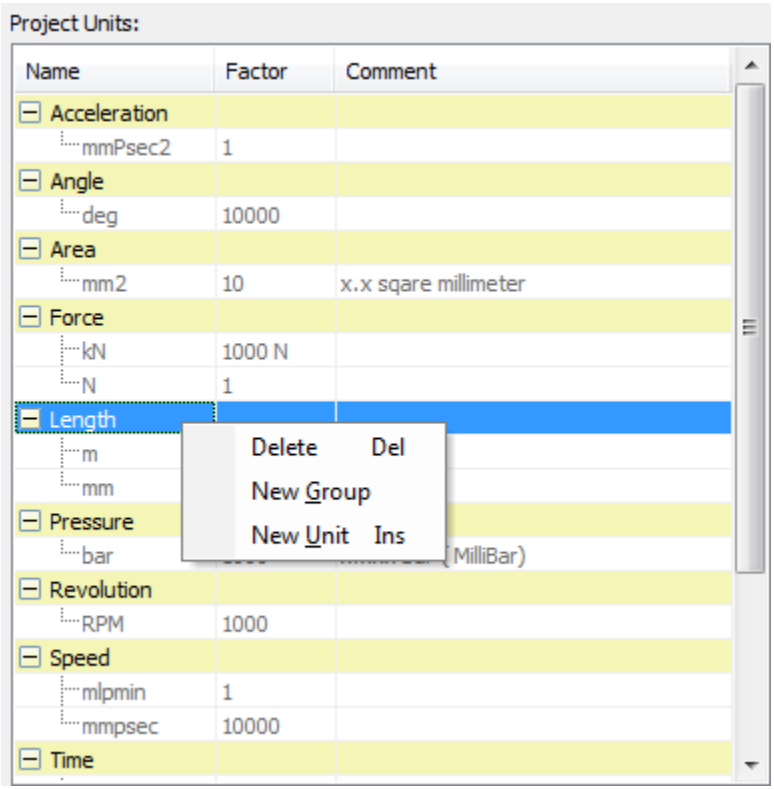
A value can also be overwritten through the context menu. Depending on whether a unit or group is clicked, a different context menu is shown:

Unit Context Menu



Delete	Deletes the selected unit.
New Unit	Adds a new unit.
Overwrite Unit	Overwrites the selected unit.

Group Context Menu



Delete	Deletes the group.
New Group	Adds a new group.
New Unit	Adds a new unit to a group.

If the changes should be reversed or the unit newly edited from the global pool, the Update is clicked. Only the selected unit is overwritten! Previouslly entered changes to other units remain unchanged.

If all deleted units from the global pool should be restored, Update All is clicked. Only units no longer available are replaced, previous changes remain untouched. Units that should not restored with Update All must be entered in the Ignore list for “Update All”.



Entries in the list can be separated with spaces, a comma or semi colon.

With OK the changes are accepted, with Cancel they are reversed.

Files and #defines

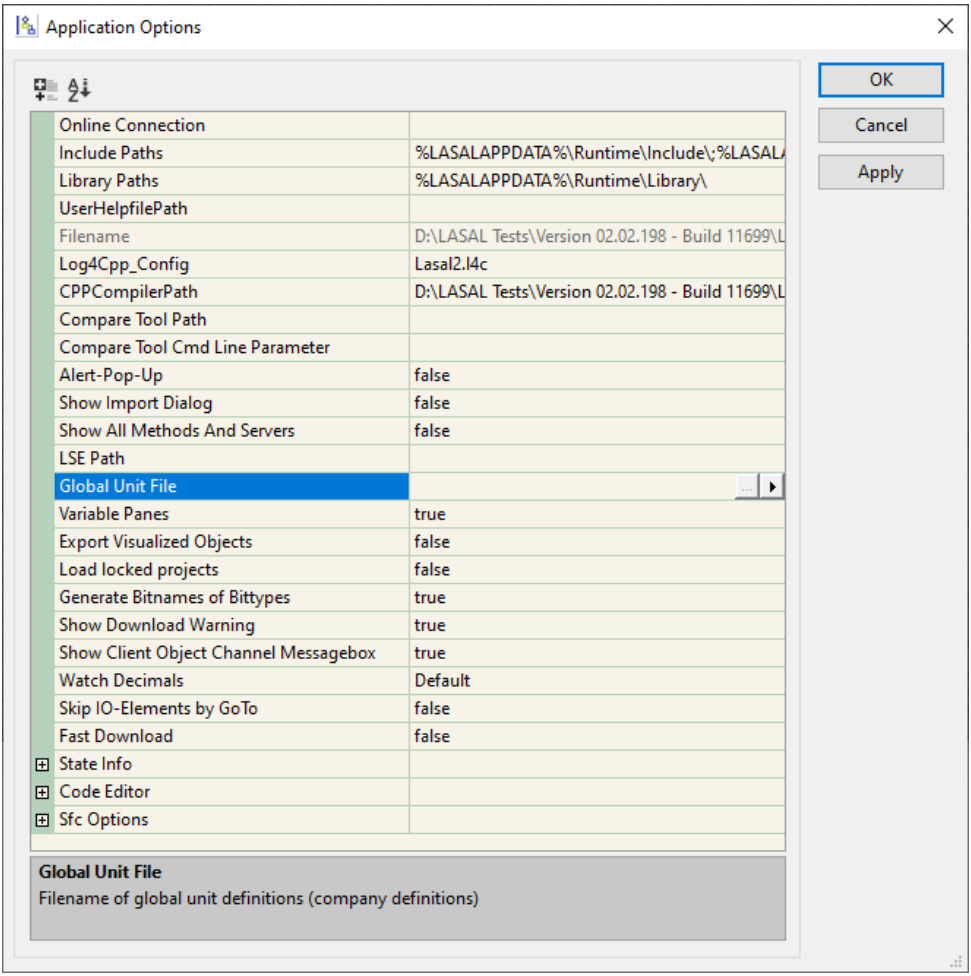
Before being entered in the LASAL **Unit Configuration**, all units are stored in the global **Units.h** file (in Runtime directory). In an old project is opened, the right half of the **Unit Configuration** window is empty. If a new project is created or when in an old project, a change is made on the right side of the **Unit Configuration window** so that it is not empty, the unit.h file is automatically created. This file is a local project file and can be found under **Header Files**. The **unit.h** file always contains the `#define __LSLPRJ_UNITS_H__` and in the **UserDef.h** file, the lines are found

```
#ifndef __LSLPRJ_UNITS_H__
  #include <Units.h>
#endif
```

It is therewith possible to decide whether the old or new unit handling should be used.

If a new project is created and the old version should still be used (global **Units.h** without local **unit.h**), all units groups / units must be deleted from the right side of the **Unit Configurations window** (click on the unit /group to delete and press the **delete** button or select **Delete** from the **context menu**).

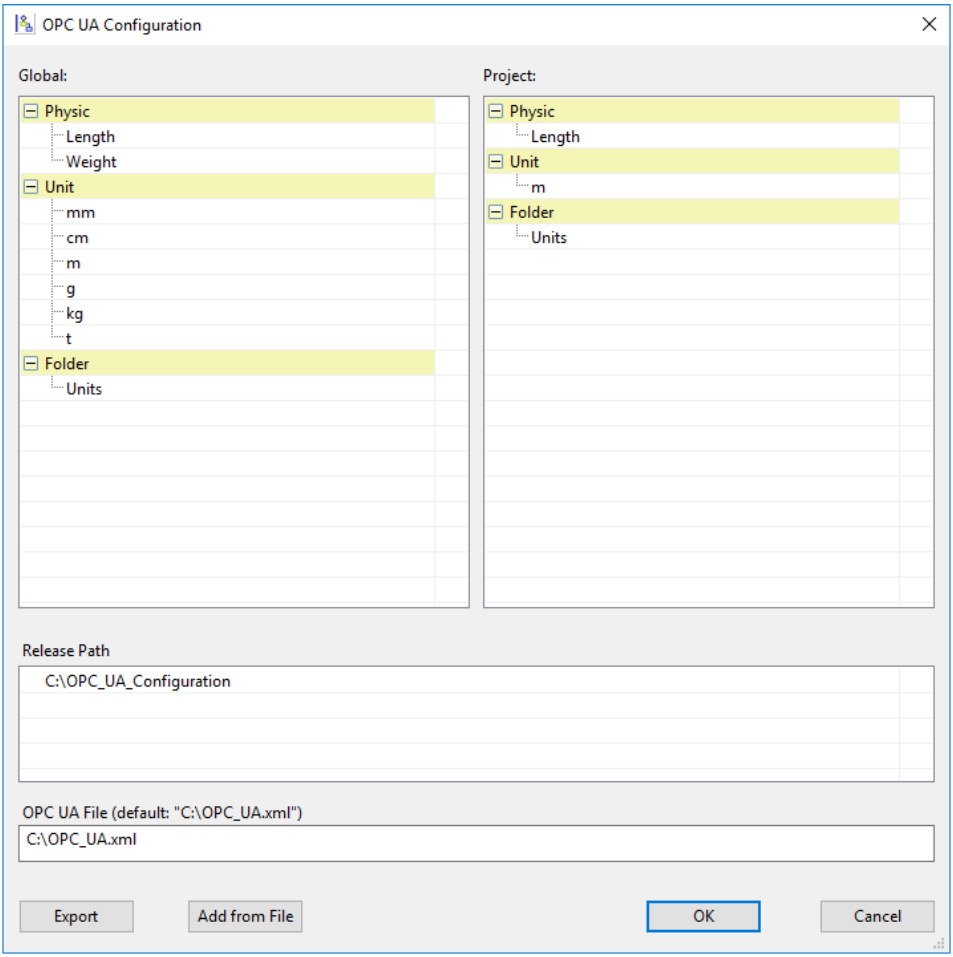
A global unit file can also be created, which can be customized using predefined SIGMATEK units. Here, the path and file name of the company-specific unit file must be entered in Extras / Options under **Application Options**. The expansion for unit files is ***.luc** (**L**ASAL **U**nit **C**onfiguration).



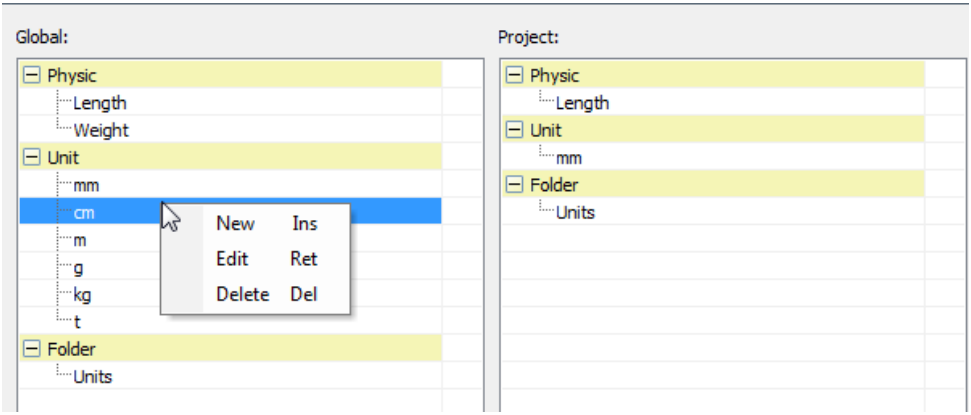
This file is assembled using the predefined units from SIGMATEK **only once during the creation of a new project** and has nothing to do with the project-specific unit.h header file. The file therefore cannot be stored in the project directory since when saving the project, LASAL will **automatically delete it**!

2.1.7.4 OPC UA Configuration

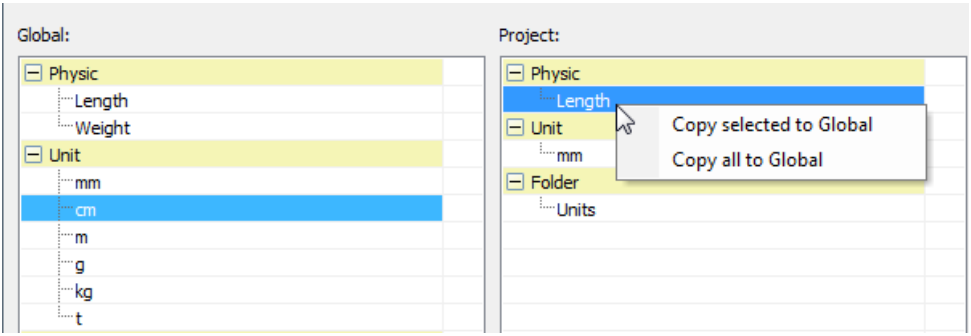
Opens the **OPC UA Configuration window**.



The upper part is split in two: On the **left side**, the **global** variables are listed that can be selected in the **server properties** under **OPC UA Settings** - provided that **Enable OPC UA** is set to true in the **project properties**. This list can be expanded with a right-click of via the context menu and the items added (**New**), edited (**Edit**) and deleted (**Delete**):



On the **right side**, the values **stored in the project** are displayed. These cannot **be edited**; **they are display only**. With a right-click or via the context menu, the selected (**Copy selected to Global**) values or all values (**Copy all to Global**) can be transferred to the global list. This function is designed to **incorporate** settings from projects, which already have **OPC UA settings** stored:



The entries selected for transfer are then added to the global list.

In the **Release** field, one or more released directories **in the control** can be specified for **File Up or Download**.

The **OPC UA File (default: „C:\OPC_UA.xml“)** field indicates the **location in the control**, in which the OPC UA configuration file is found.

With the **Export** button, an OPC UA configuration can be stored **in the PC**; via **Add from File**, the configuration can be read from an existing file. With **OK**, the current settings are accepted, with **Cancel** they are discarded.

2.1.7.5 Exception Handling

Select **Project, Exception Handling** in the menu bar to detect program errors (Access Violations, Division Errors).

This command is only available when the "_SIG_LSL_CexceptionHandlerBase" class from the "OS Interface" library is included in the project. If not, LASAL indicates it in the output window.

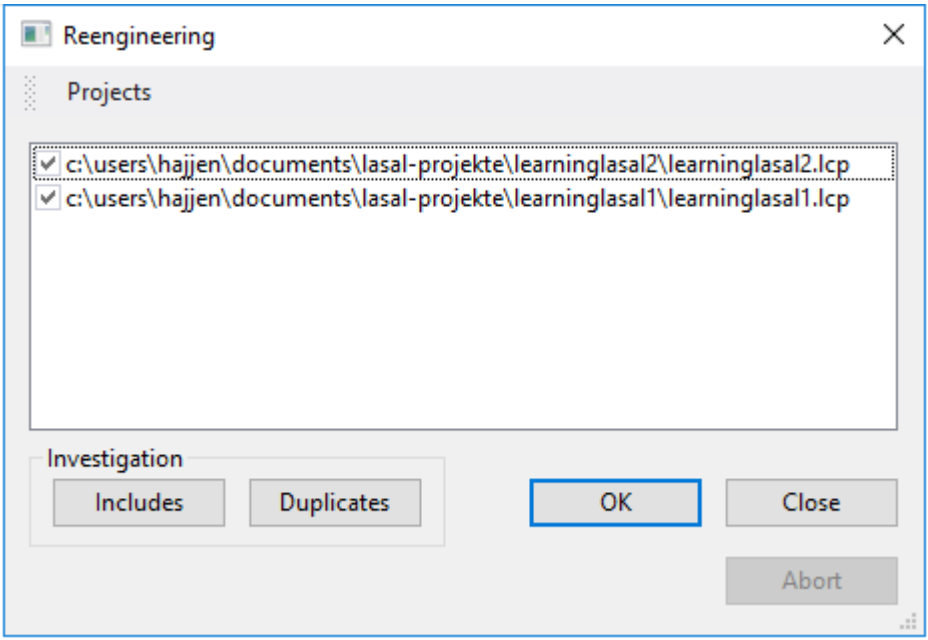
If the class is existent, through execution of this menu command a derivation of this class is created. Additionally an object of the class in the write protected "__ExceptionHandling" network is instanced.

Through overwriting the methods "EHAccessViolation" and "EHIntDivideByZero" in derivation it can be reacted to the exceptions with user code. By this mechanism it is guaranteed that the handlings of the exceptions are installed before first call of the user code (before call of the first constructor of a user class).

Be rerunning the menu command “Exception Handling”, the network and the derivation the class "_SIG_LSL_CexceptionHandlerBase" are **deleted from the project**.

2.1.7.6 Reengineering...

Opens the Reengineering window, in which the project can be checked for irregularities regarding global includes and multiple definitions.



In this window, individual (**Includes** only) or multiple (partial) projects can be selected to run **Includes** or **Duplicates**. **Includes** shows all globally included header files; Duplicate shows all classes, objects or Defines that appear or are defined in the project more than once.

The output is displayed in the output window below. An example output after running **Includes**:

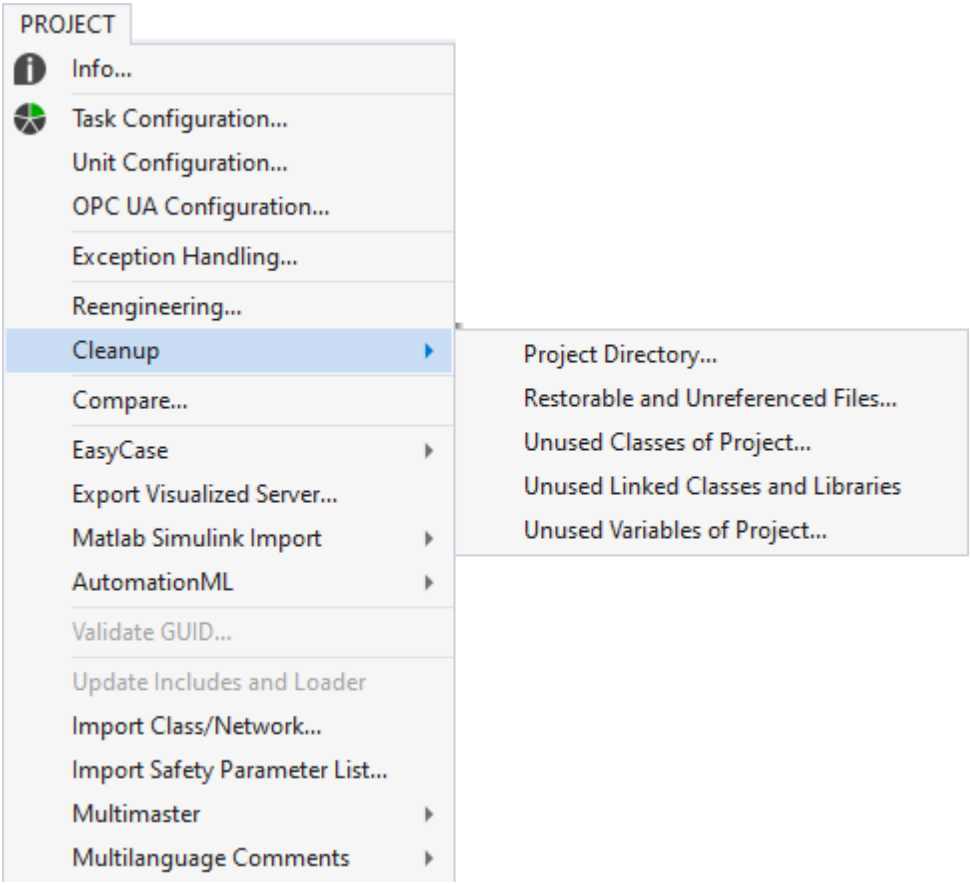
```
Output: [INFO] 'Includes investigation' started
Output: [INFO] Reengineering investigation result:
Output: [INFO] Global Project Includes:
Output: [INFO] ------------------------------------------------------
Output: [INFO] "D:\Projekte\Testprotokoll 08072010\DataAnalyserProject\Include\MyHeader.h"
Output: [INFO] "D:\Projekte\Testprotokoll 08072010\DataAnalyserProject\Include\Second.h"
Output: [INFO] ------------------------------------------------------
Output: [INFO] 'Includes investigation' finished
```

Here, two globally included header files are found. An example output after running **Duplicates** can now be given:

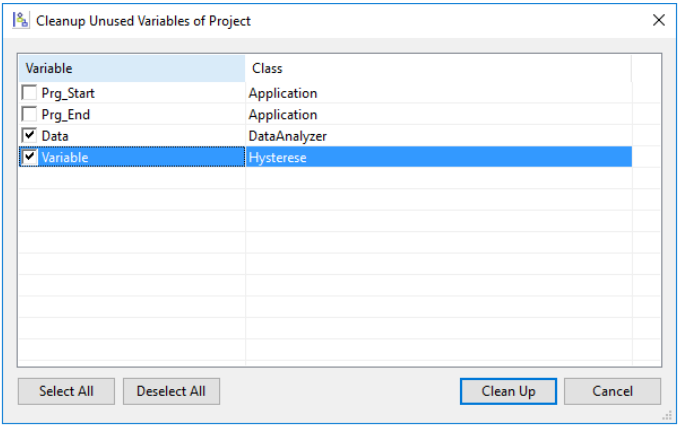
```
Output: [INFO] Reengineering investigation result:
Output: [INFO] Duplicate Classes:
Output: [INFO] ----------
Output: [INFO] Class0
Output: [INFO] d:\projekte\testprotokoll 08072010\duplicates1\duplicates1.lcp
Output: [INFO] d:\projekte\testprotokoll 08072010\duplicate2\duplicate2.lcp
Output: [INFO] ----------
Output: [INFO] Duplicate Defines:
Output: [INFO] ----------
Output: [INFO] USER_CNT_CLASS0
Output: [INFO] D:\Projekte\Testprotokoll 08072010\Duplicate2\class\class0\class0.st
Output: [INFO] d:\projekte\testprotokoll 08072010\duplicates1\class\class0\class0.st
Output: [INFO] ----------
Output: [INFO] Duplicate Objects:
Output: [INFO] ----------
Output: [INFO] Class01
Output: [INFO] d:\projekte\testprotokoll 08072010\duplicates1\duplicates1.lcp   MeinNetzwerk
Output: [INFO] d:\projekte\testprotokoll 08072010\duplicate2\duplicate2.lcp   MeinNetzwerk
Output: [INFO] ----------
Output: [INFO] 'Duplicates investigation' finished
```

If the dialog is closed with **OK**, the selected projects are displayed when the dialog is reopened; if the dialog is closed with the Close, the changes are not saved. The **Abort** button is used to interrupt longer tests (over several / large projects). In this case, "Duplicates investigation Cancelled" is shown in the output window.

2.1.7.7 Cleanup



Project Directory...	Searches the project directory for files that are no longer required.
Restorable and Unreferenced Files...	Deletes all files from the project that cannot be referenced or restored.
Unused Classes of Project	Removes all classes from which no object is placed and are not used by another class with a place object (for example, through an object channel, Public Type, used as a pointer type, embedded object...).

Unused Linked Classes and Libraries	Removes – taking the RCD project settings into account (ReducedClientDependency; found in the project properties under compiler) – all linked classed not being used and all libraries not needed by linked classes.
Unused Variables of Project...	Opens a dialog, where all unused variables are listed. Tese can be selected and deleted here. 

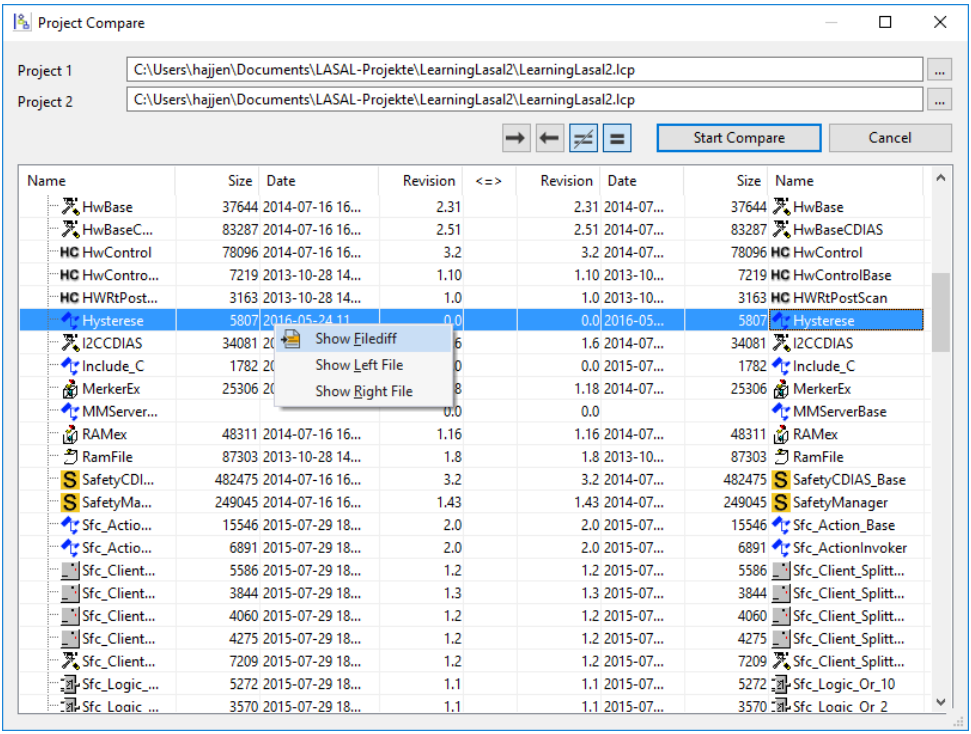


Attention

For all these items, the folders / files entered in the project properties under **Ignore at cleanup are ignored**!

2.1.7.8 Compare

Compares two LASAL projects. The individual source code files of the project are compared and the differences are displayed.



In the editor fields "Project 1" and "Project 2", LASAL projects can be entered for comparison. If the current project is loaded LASAL, it is located in the field "Project 1".

By clicking on the "Start Compare", the project is loaded and compared. The result of the comparison is shown in the list and is grouped by classes, object networks, header files, source file and object files.

With the 4 select buttons, the decision can be made whether to show newer elements (by revision), older elements (by revision), varying elements (revision is the same or not available) and/or like elements.

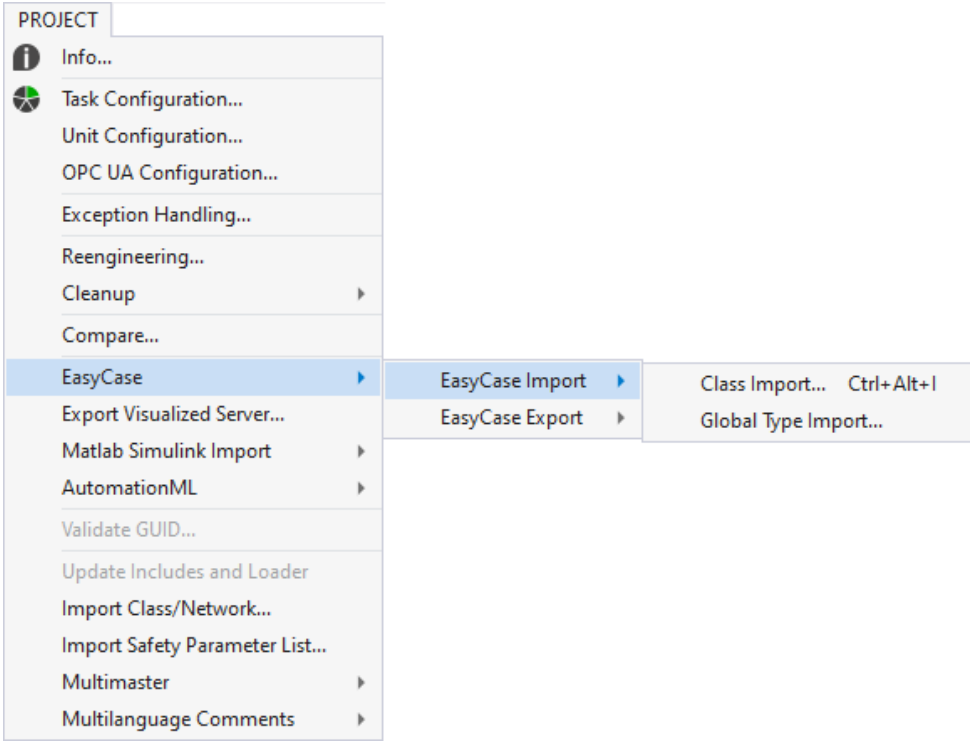
With a right click on the result line, the file from project 1, project 2 or the differences between the two files can be shown.

To view the differences, the **WinMerge** program (the exe file only) must be available either in the bin folder of LASAL 2 or in a folder with a set windows environment path (e.g. Windows\System32).

2.1.7.9 EasyCase

2.1.7.91 General Function mode

At **EasyCase Import** (also accessible via the shortcut keys **Ctrl + Alt + I**), a valid LASAL 2 class is created from an EasyCase export file.



The basic elements for the classes must be capable of being interpreted from the export file.

The basic elements include:

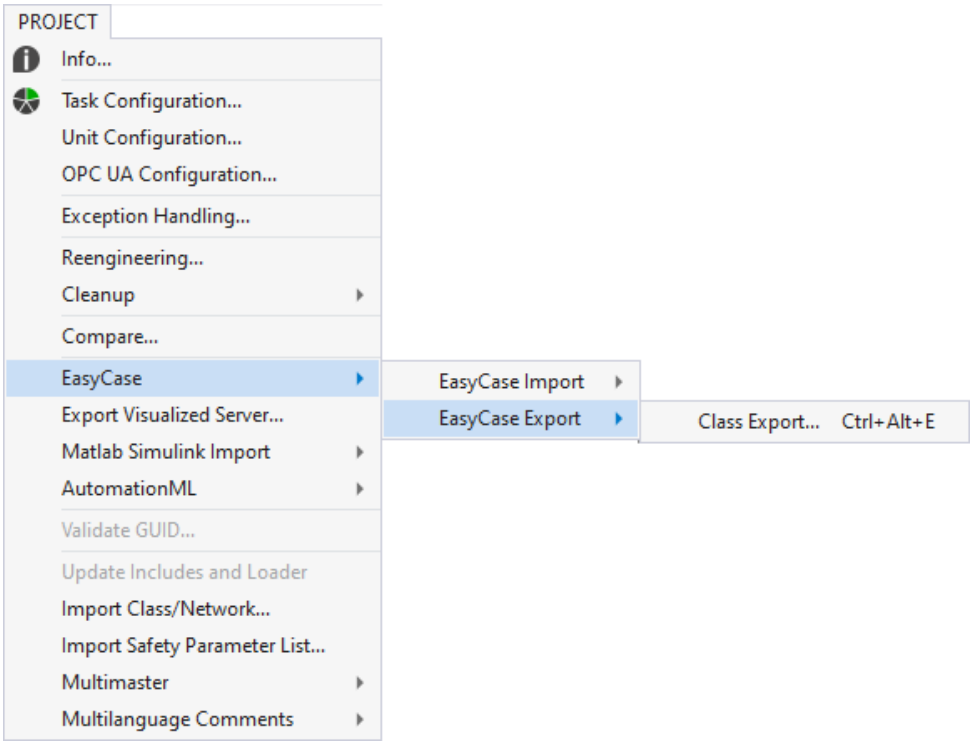
- Defined class name
- Server
- Clients
- Variables
- Types
- Functions

Each class must, as a minimum requirement, have a name and a server; the other basic elements are optional. The required server must not be available in the EasyCase export file, it is added by LASAL 2 automatically. To build valid connections to hardware classes the classes must have corresponding clients.

When a class is updated, the existing class is replaced by a new class and the existing objects are adjusted (e.g. new clients added). Manual settings, respectively, settings that cannot be made in EasyCase (here properties of server and clients are meant e.g. Required) must not be lost at update.

The creation of the networks with the objects must be done in LASAL 2. The settings of the networks with the objects as well as the connections to the hardware have to be made in LASAL 2. This can also be done via the corresponding script command.

With EasyCase Export (also accessible via the shortcut keys Ctrl + Alt + E), **all classes** (multiple choice possible) **selected in the Class View** are exported as ***.etf file**. For each class, the server, clients, methods, variables, types and tables are exported in EasyCase format.



2.1.7.92 Import Process

- The name of the export file corresponds to the class name.
- Global variables are written in a global header file with VAR_EXTERNAL that includes the class (name of the header file includes the class names) and is defined in an ST-File with VAR_GLOBAL (name of the ST-Files is Global_Klassenname.st). The variable is assigned through the definition in the ST file, and is made available in the entire project through the declaration in the global header file.



Attention!

Even when several objects of a class are placed, the global variable is declared only once.

- Constants are also written in the global header file and are located at the beginning (because types from these constants are possibly already used).
- Private variables are assigned as variables for the class.
- Input and output variables are linked in the corresponding function as input and output variables.
- Private functions are handled as private methods of the class.
- The declaration of all global functions are written in the header file of the class. The implementation of the function is written in Global_Klassenname.st. Through the declaration in the global header file, the function is available in the entire project.
- Private tables are treated as private tables of the class.
- The declaration of global tables are in the header files of the class. The implementation of the table is written in Global_Klassenname.st. Through the declaration in the global header file, the function is available in the entire project.
- If the properties "AWL" or "__cdecl" are required at a function, then the key word has to be defined in EasyCase at the function name (e.g. "FUNCTION_GLOBAL AWL Function name", or "FUNCTION__CDECL Function name").
- If a function is implemented in the HIDDEN area, the keyword HIDDEN must be entered in EasyCase with the function name (e.g.: “FUNCTION_GLOBAL HIDDEN Functionsname“). Mainly applies to NewInst functions and SFC classes.
- Server and clients are interpreted according to their identification (data, object channel).
- Behind the FUNCTION key, flags (properties) can be set. The respective key word is written without underscore, separated through a space character from the word FUNCTION. The four properties are: GLOBAL, VIRTUAL, AWL and _CDECL.
- With an underscore GLOBAL and VIRTUAL can also be connected with the word FUNCTION. However, the results are different:
- Through FUNCTION_GLOBAL, a global Header file *.h and a *.st file are created that are added to the class. With FUNCTION GLOBAL, the function is added to the class and the **Global access** Property flag is set to **true**.
- The VIRTUAL keyword behaves similarly - FUNCTION_VIRTUAL creates a **global** virtual method of the class; if VIRTUAL is behind FUNCTION separated through a space character, a **private** virtual method of the class is created.
- Remanent server have to be marked (details still have to be defined).
- The first server is created by LASAL automatically; this does not come from EasyCase.

- Global types in a class export file are added to the global header file of the class. However, a sequence problem arises (e.g. a global variable in the header file of class A uses a global type from the header file of class B). Types that conflict when referencing must be located in a single export file. During import to LASAL, it will be determined that global types are being handled. This special global type import generates a separate global header file that first links all include files in global.h. This avoids problems with the sequence of the include files.
- If a standard method is imported (Read, Write, CyWork, RtWork Background, Constructor, Init, NewInst, Kill, GetState), all defined transfer and return parameters as well as eventually wrong set function flags (e.g. VIRTUAL) are ignored and allocated with the correct standard values. At the standard methods only the flags AWL and __cdecl can be explicitly set in EasyCase.
- When using the respective configuration file, the level commands defined in EasyCase, are inserted into the **LASAL class code** as comment in the appropriate form (see Export Process).

2.1.7.93 Export Process

If a LASAL class is exported, all servers, clients, methods, variables, types, tables, etc. are written into a ***.etf file** one after another. This is the **Standard export**. In EasyCase the servers, methods, parameters, etc. are displayed in standardized mode (at the top level e.g. **Definition** and **Source**, one level lower **Clients**, **Servers**, **Private Variables** and **Types** below the Definition; then below **Source** the **Standard** and **Server methods** etc.)

With the appropriate commands (see below) it is now possible to define ones **own structure** for EasyCase. For example, **to set the levels in LASAL yourself**.

Therefore the following syntax is defined:

```
//{{!open_level Levelname !}}
.
FUNCTION...
.
.
END_FUNCTION
FUNCTION...
.
.
END_FUNCTION
.
//{{!close_level !}}
```

2.1.7.94 Keywords in the EasyCase Export File

Keywords for the individual areas are used to read the file, which must also defined in EasyCase.

Global VariableS	`VAR_GLOBAL`
	`  GlobaleVariable1 : DINT;`
	`END_VAR`

Global Constants	`CONSTANT`
	`  GlobaleConstante : Value;`
	`END_CONSTANT`

Private Variables	`VAR_PRIVATE`
	`  LokaleVariable1 : DINT;`
	`END_VAR`

Server	`VAR_SERVER`
	`  Server1 : DINT;`
	`  Server2 : CLASS0;`
	`END_SERVER`

Remanente Data	`VAR_GLOBAL RETAIN`
	`  GlobaleVariable1 : DINT;`
	`END_VAR`

Clients	`VAR_CLIENT`
	`  Client1 : DINT;`
	`  Client2 : CLASS0;`
	`END_CLIENT`

Functions of the class	`FUNCTION Function1 // (private)`
	`  Source`
	`END_FUNCTION`
	
	`FUNCTION GLOBAL Function1 // (global)`
	`  Source`
	`END_FUNCTION`
	
	`FUNCTION VIRTUAL Function1 // (private virtual)`
	`  Source`
	`END_FUNCTION`
	
	`FUNCTION_VIRTUAL Function1 // (global virtual)`
	`  Source`
	`END_FUNCTION`
	
	`FUNCTION GLOBAL VIRTUAL Function1 // (other syntax for global virtu`
	`  Source`
	`END_FUNCTION`
	
	`FUNCTION HIDDEN Function1 // (function that should be implemented i`
	`  Source`
	`END_FUNCTION`

Global Functions	`FUNCTION_GLOBAL Function1 // (is written into a global header file)` `  Source`

	`END_FUNCTION`

Table of the class	`TABLE Table1`
	`  Entries`
	`END_TABLE`

Global Tables	`TABLE_GLOBAL Table1`
	`  Entries`
	`END_TABLE`

Function variables	`VAR //(local variable of the function)`
	`  PrivateVariable1 : DINT;`
	`END_VAR`
	
	`VAR_INPUT //(input variable)`
	`  InputVariable1 : DINT;`
	`END_VAR`
	
	`VAR_OUTPUT //(output variable)`
	`  OutputVariable1 : DINT;`
	`END_VAR`

Global Types	`TYPE_GLOBAL`
	`  Type1 : DINT;`
	`END_TYPE`

Types of the class	`TYPE`
	`  Type1 : DINT;`
	`END_TYPE`

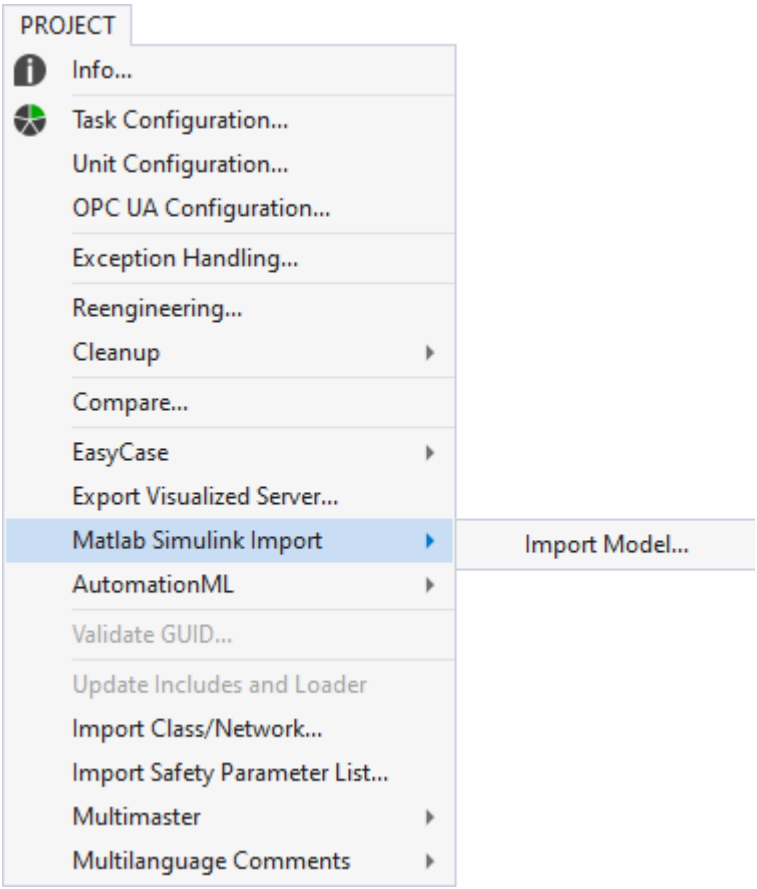
2.1.7.95 Restrictions for EasyCase/EasyCode

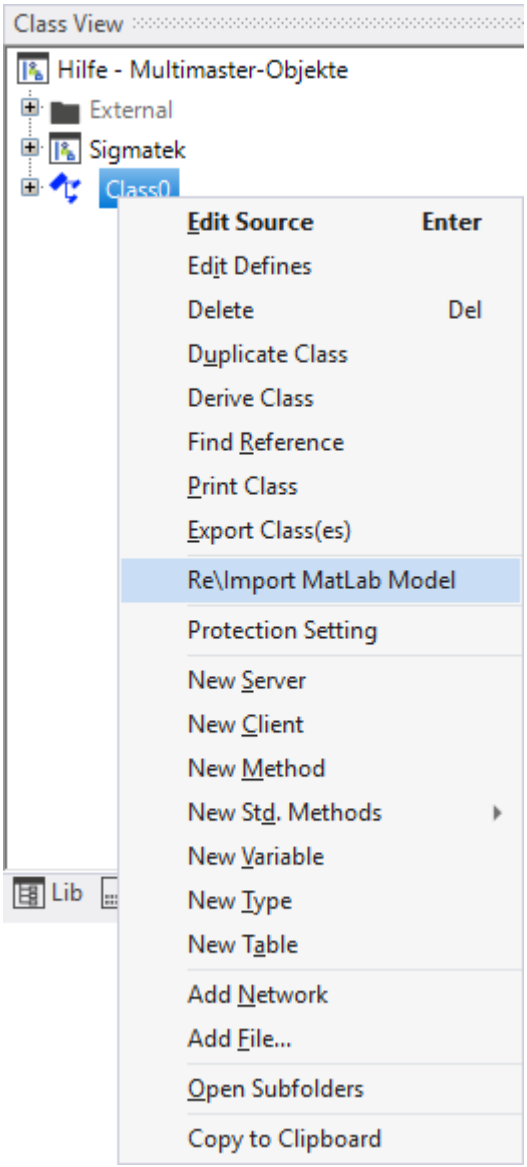
- To call other class methods, clients for communication have to be available.
- In global functions only local and global variables (no class variables) can be accessed.
- Clients have to be available for communication with hardware classes.
- Names of classes, functions, clients and server have to be explicit.
- The names of the export file and the function have to be different (class name and function name have to be different, if the function has the same name the constructor is concerned)
- The restriction that global variables can only be standard types when visualized is abandoned. The visualization of global structures has to be working too.
- All declarations (VAR_GLOBAL, TYPE, etc.) positioned within a function block, have to be written before the source code itself.

2.1.7.10 Export Visualized Servers

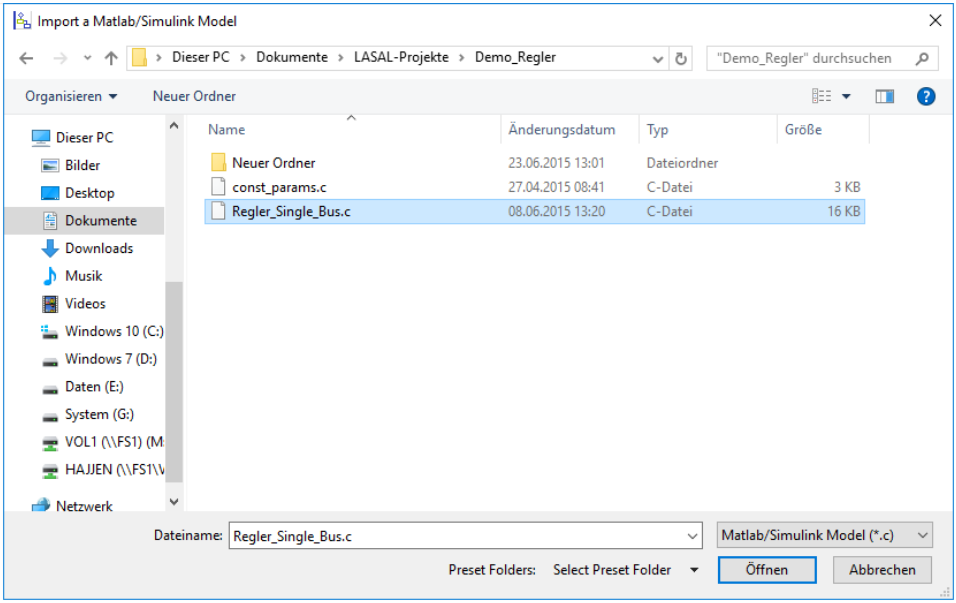
Exports all visualization servers in the project to a file (server property **“Visualized” = true**). If the **“Export visualized Objects“** property in the **Application Options** is set to **true**, the visualized object is exported. The name, type and commentary are exported to a *.txt file. If the object of a complex class is not visualized, but its embedded object is, **it will not be exported!**

2.1.7.11 Matlab Simulink Import



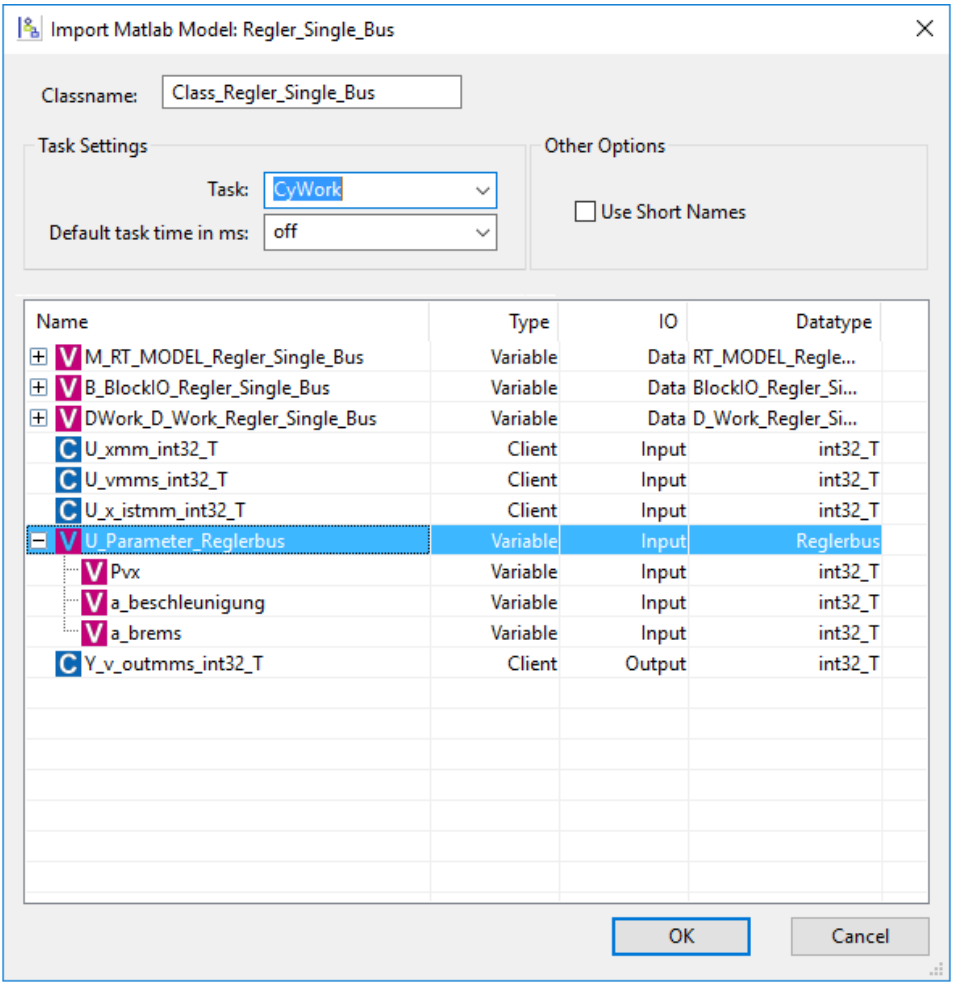

Class View
Hilfe - Multimaster-Objekte
External
Sigmatek
Class0
Edit Source Enter
Edit Defines
Delete Del
Duplicate Class
Derive Class
Find Reference
Print Class
Export Class(es)
Re\Import MatLab Model
Protection Setting
New Server
New Client
New Method
New Std. Methods
New Variable
New Type
New Table
Add Network
Add File...
Open Subfolders
Copy to Clipboard
Lib

With the Matlab Simulink Import, models can be exported from Matlab in C code and imported into LASAL CLASS 2. There are two options for importing: either through the project menu, or in the context menu of an existing class (here, select the menu item **Re\Import MatLab Model**).



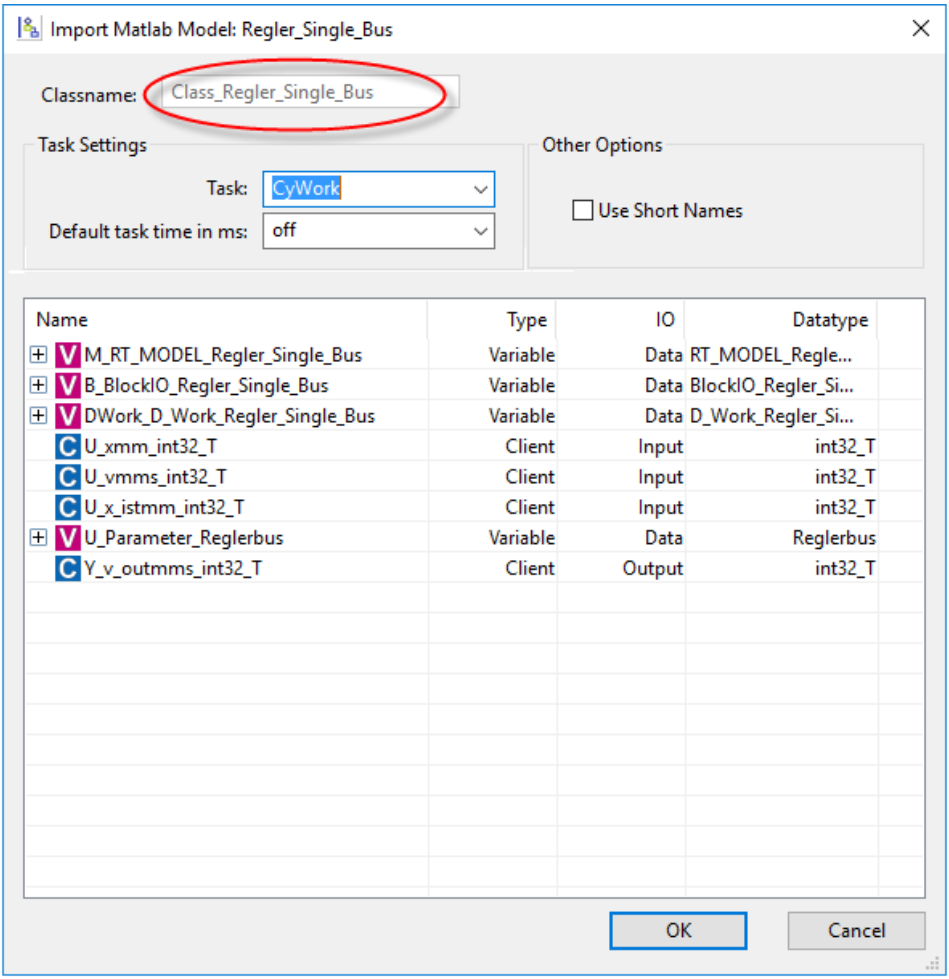
In both cased, a FileOpen dialog appears. To import the model, the .C file of the model must be selected. All files required for compiling the model exported in C Code, must be located in the same folder as the selected .C files of the model. These files are then added to the project directory as a copy of the class.

If a .C file of a model is selected, the import dialog opens:



Under **Classname**, a default name of the class to be created by LASAL CLASS is stored (here, **Class_Regler_Single_Bus**, derived from the name of the previously selected .C file), which can be changed as desired.

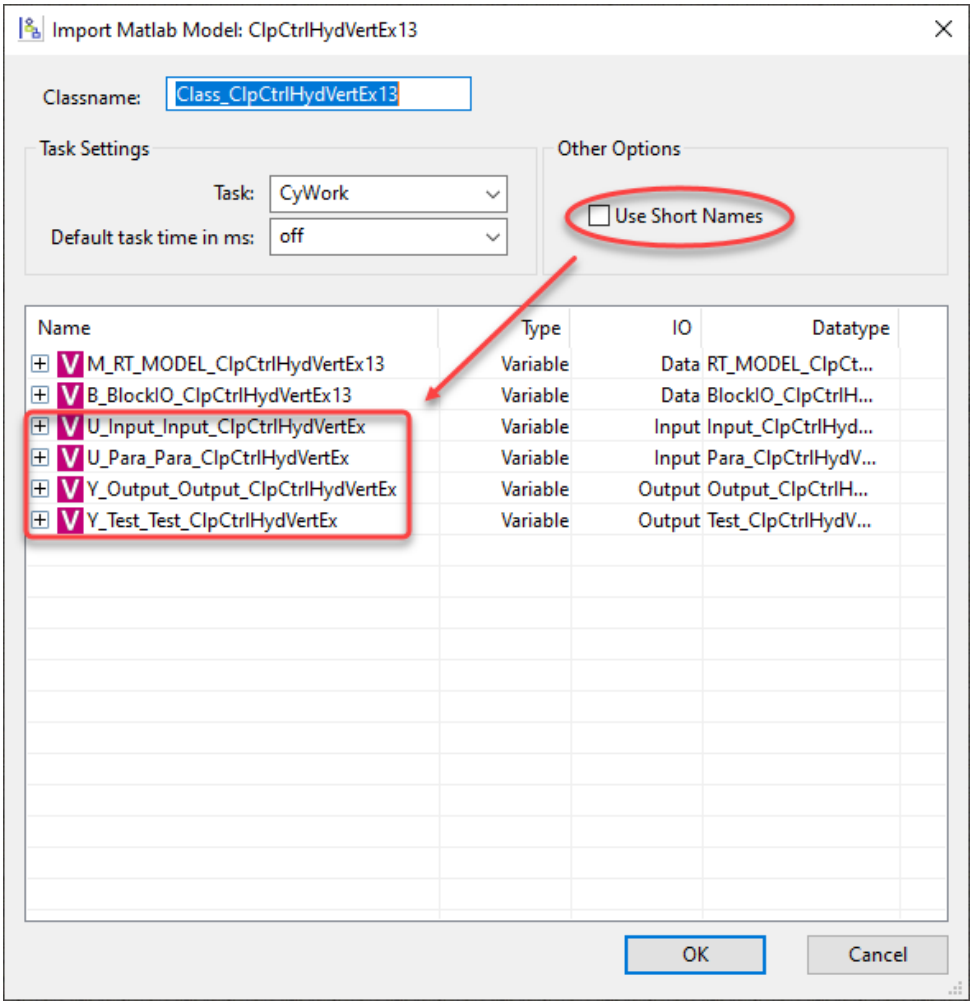
If a model is imported into an existing class (**Re\Import MatLab Model** in the context menu of the class), the class name is predefined and cannot be changed.



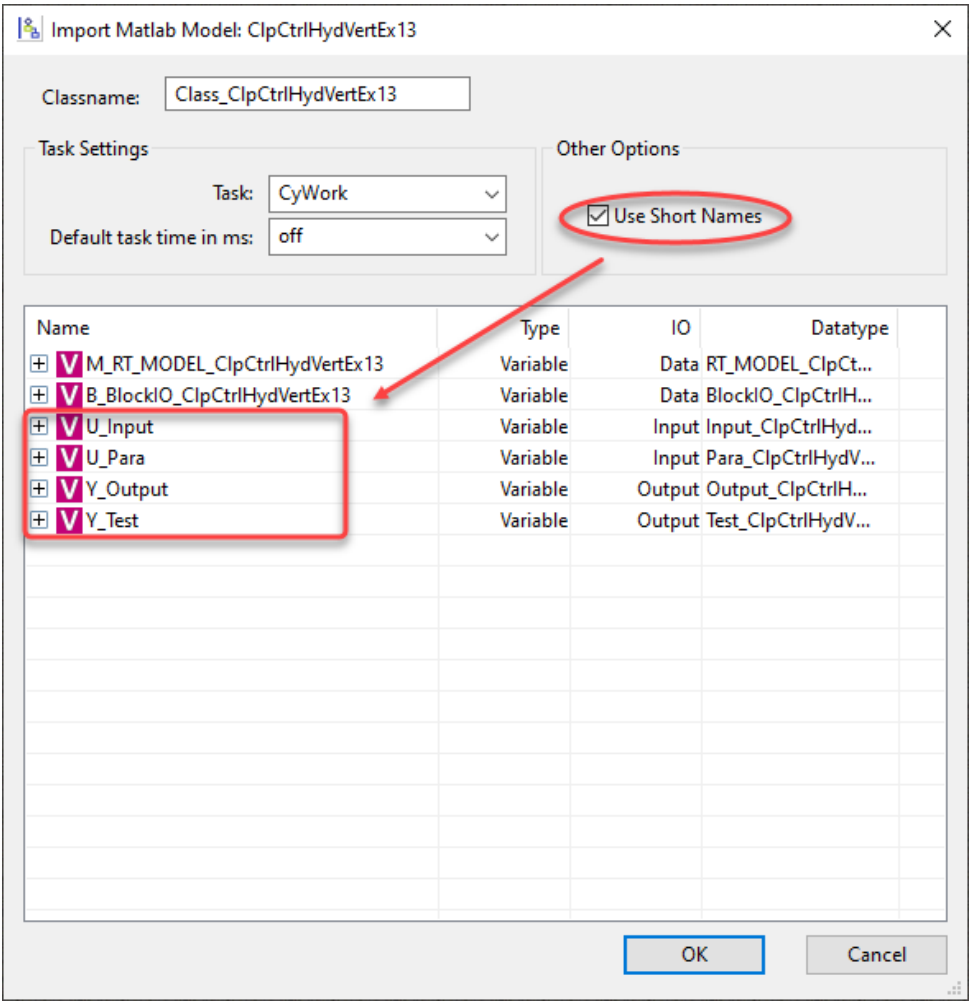
The **task**, in which the step function of the model should be called, as well as its **default cycle time** can be set.

Under **Task Settings / Task**, the task types are listed (**CyWork**, **RtWork** or **Background**) and under **Default task time**, the default time to automatically set in a new instance.

If the **Use Short Names** checkbox is not selected, **long names** are generated and used for servers, clients variables etc....

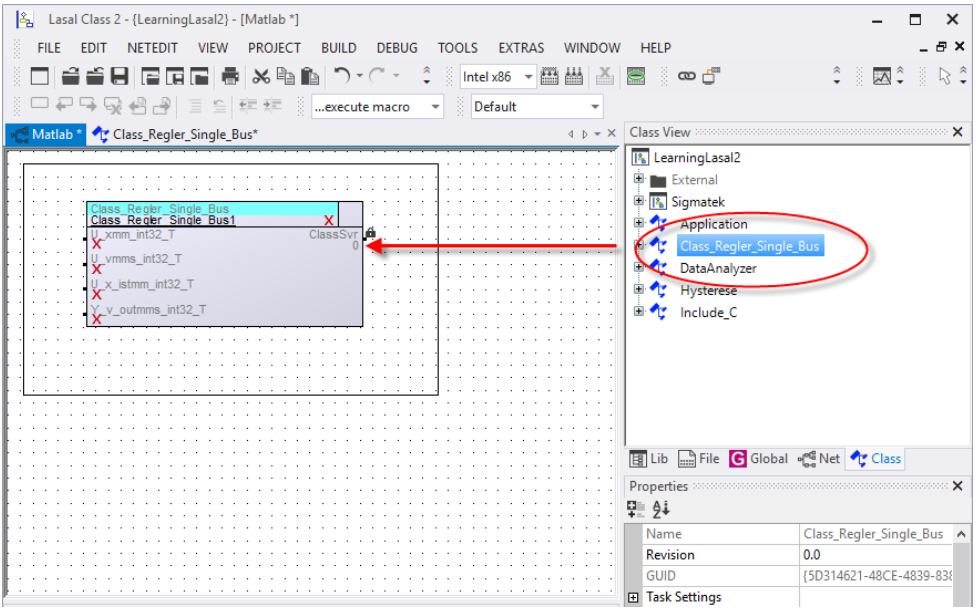


...on the other hand, if **Use Short Names** is selected, **shorter names** are generated and used:

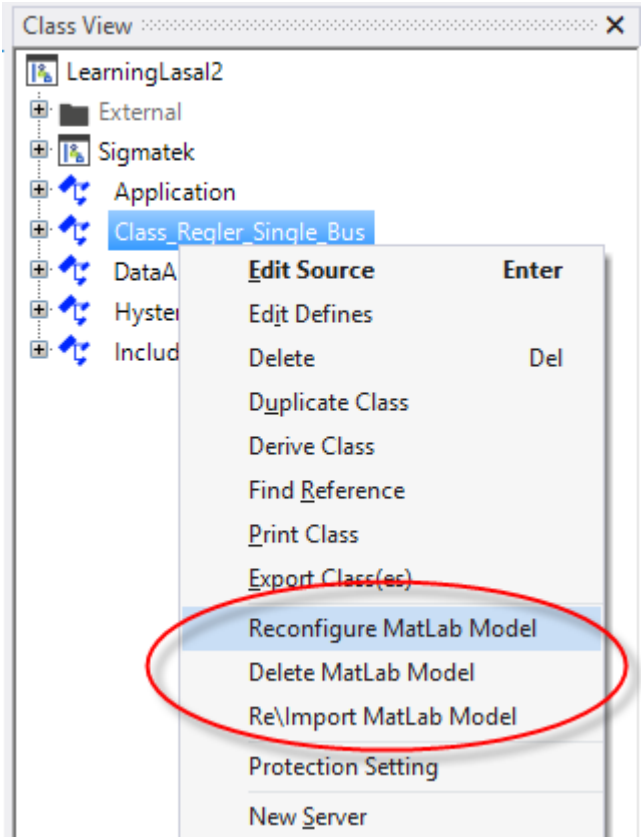


In the table below, all **imported variables** (required by the model) are listed. This is generated from the input parameters from the **Step** and **Init** functions of the model. The variables can be displayed in the class (here, in the **Type** column) as **clients**, **servers** or **member variables**. **Clients** and **servers** can also be defined (in the **IO** column) as **inputs** or **outputs**. **Inputs** are read **before calling** the step function and **outputs thereafter**.

If **imported via the menu**, LASAL creates a class with the defined name that can be used as a normal object.



If a model has already been imported into a class, it then has several MatLab-related point in its context menu:



If here, **Reconfigure MatLab Model** is selected, the Import dialog opens again and the variables (type, use) can be reconfigured.

With **Delete MatLab Model**, all elements generated during import (variables, clients, servers, methods, etc.) are deleted.

Re\Import MatLab Model is used to reimport models into an existing class or to update the model of a class (Reimport). The specified .C file must thereby have the same name as the original model. When the .C file is selected, the import dialog opens again in which the old settings for matching elements is unchanged and the new parameters/variables can be set. If a model with a different name is selected, the existing model is replaced with the new after confirmation.



Attention

Reconfigure and **Re\Import MatLab Model** causes the code for the implementation of the class to be regenerated and **thus overwritten**! If necessary, the generated code can be extended with user-defined logic by creating a derivation of the generated class and overwriting the Init or Task method of the class there accordingly.

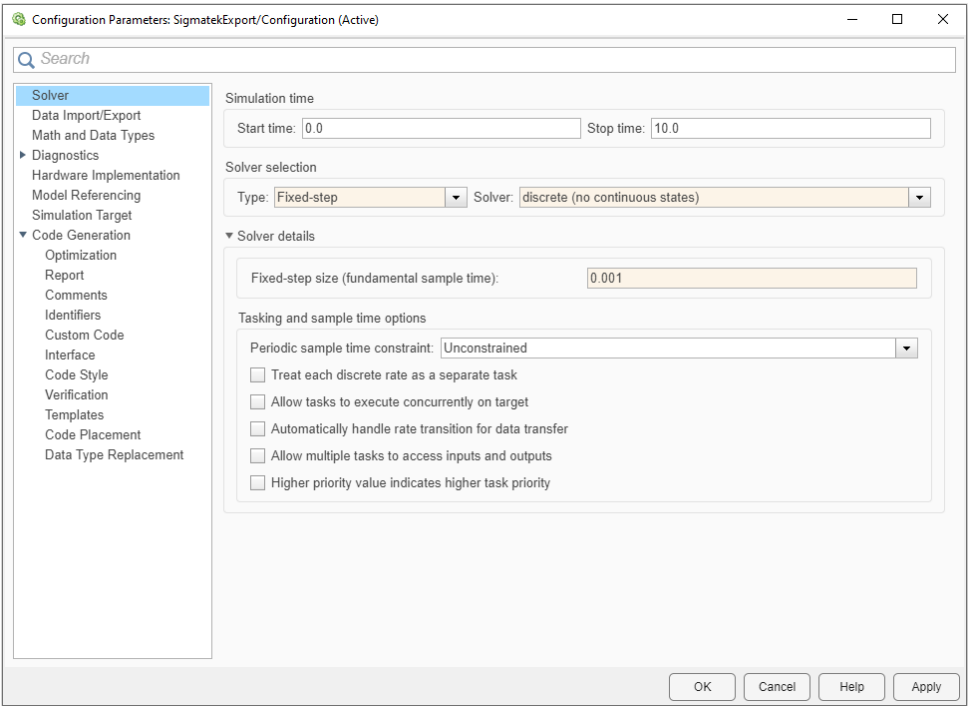
To get more information about the relevant settings in Matlab for the import, see chapter "Relevant Settings in Embedded Coder"!

2.1.7.111 Relevant Settings in Embedded Coder

Matlab Simulink Embedded Coder

The import was designed with MatLab version 2013a or 2022a and the following "MatLab Code Generator" settings. Only models created with the "Embedded Coder can be used. To ensure a smooth import, the following settings must made for exporting the model to MatLab. In addition, the "SIGMATEK code generator templates" provided must be used.

2.1.7.1111 Solver



Type:

- Fixed-step (mandatory)

Solver:

- discrete (recommended)

Fixed-step size:

- select an appropriate time interval in seconds that will also be used as cycle time for the imported model in Lasal Class 2 (use case dependent)

Treat each discrete rate as separate task:

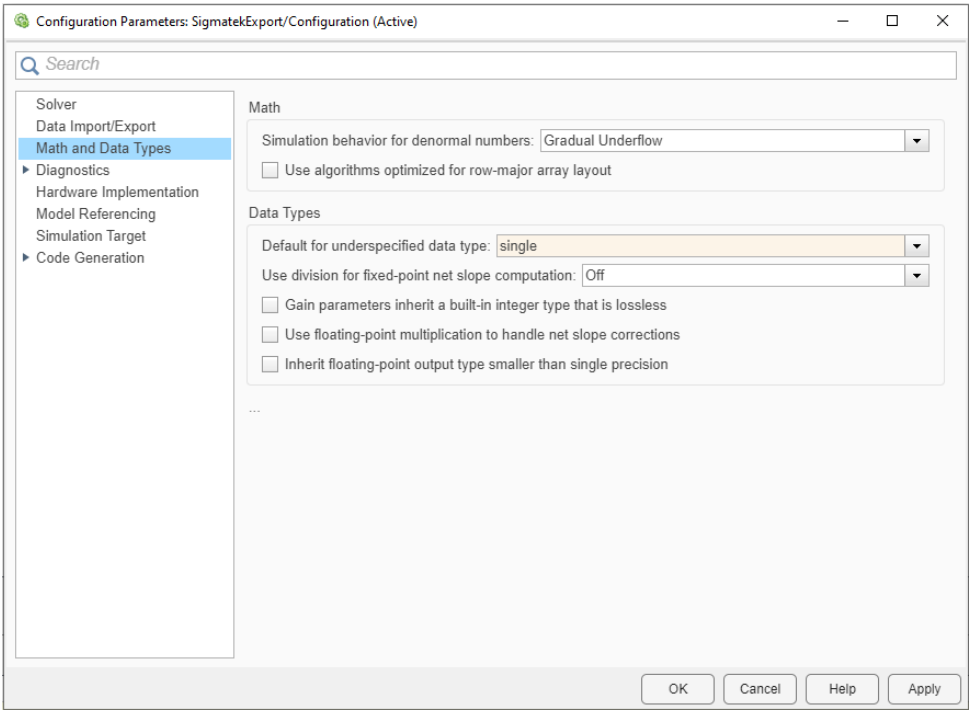
- **off** (mandatory) - the step function of a model instance in the Sigmatek integration is always called by the Sigmatek real-time task system and can therefore only run with a single fixed step scan time

Allow tasks to execute concurrently on target:

- **off** (mandatory) - the step function of a model instance in the Sigmatek integration is always called by the Sigmatek real-time task system, so multitasking within the model is currently not allowed

The remaining three checkboxes should not be selected (recommended).

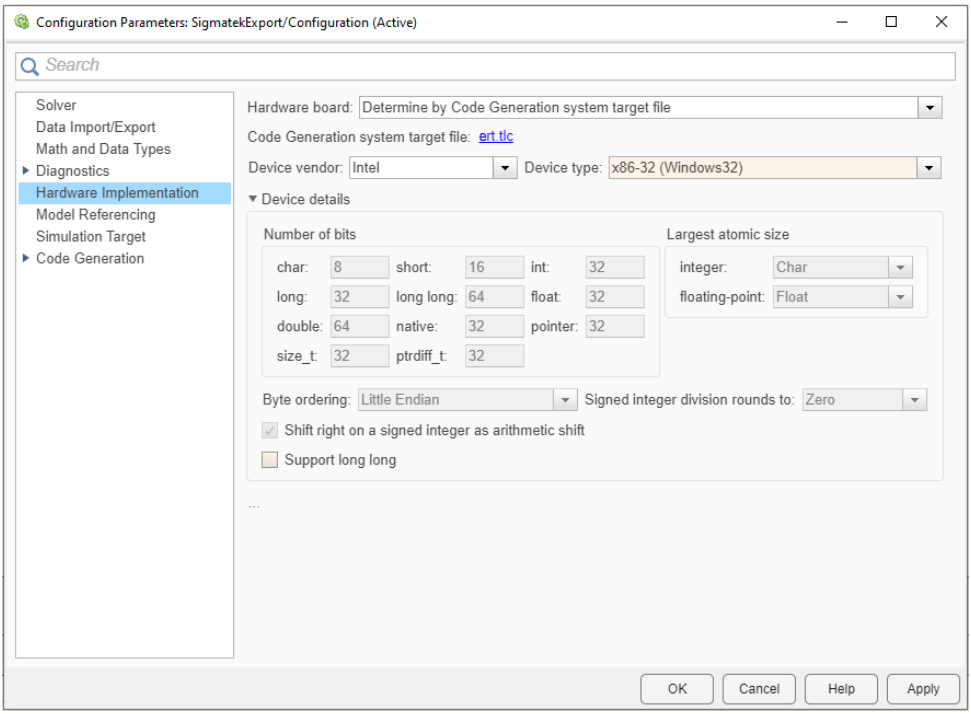
2.1.7.1112 Math and Data Types



Default for underspecified data type:

- Single (recommended)
- Double (supported, elements will not be usable as Lasal Class 2 Server / Client)

2.1.7.1113 Hardware Implementation



Device vendor:

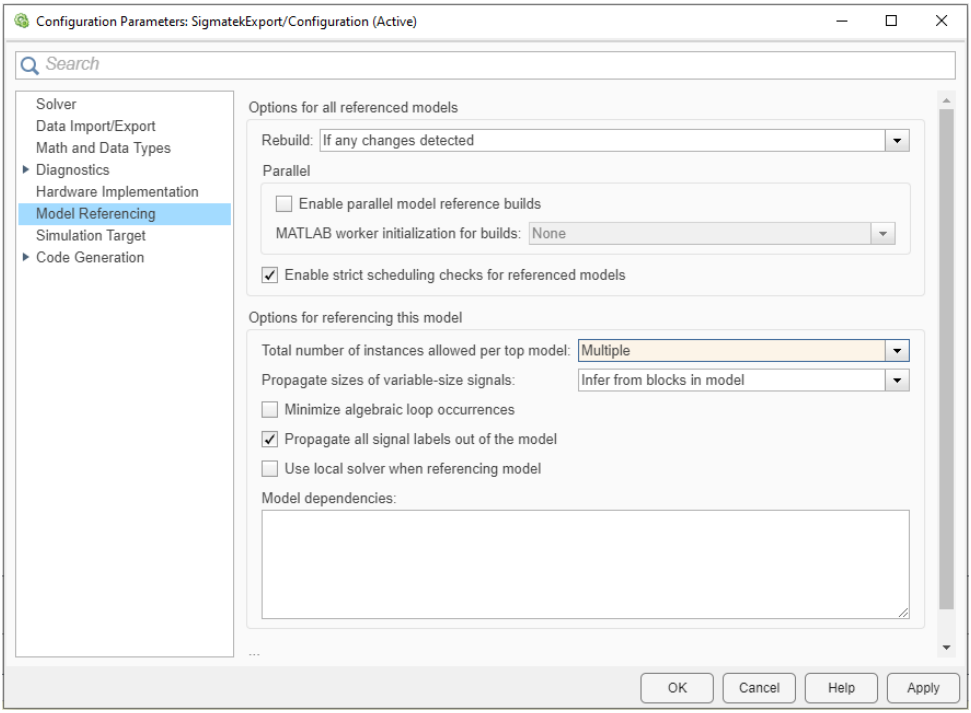
- Intel (mandatory)

Device Type:

- X86-32 (Windows32) (mandatory)

Support long-long:

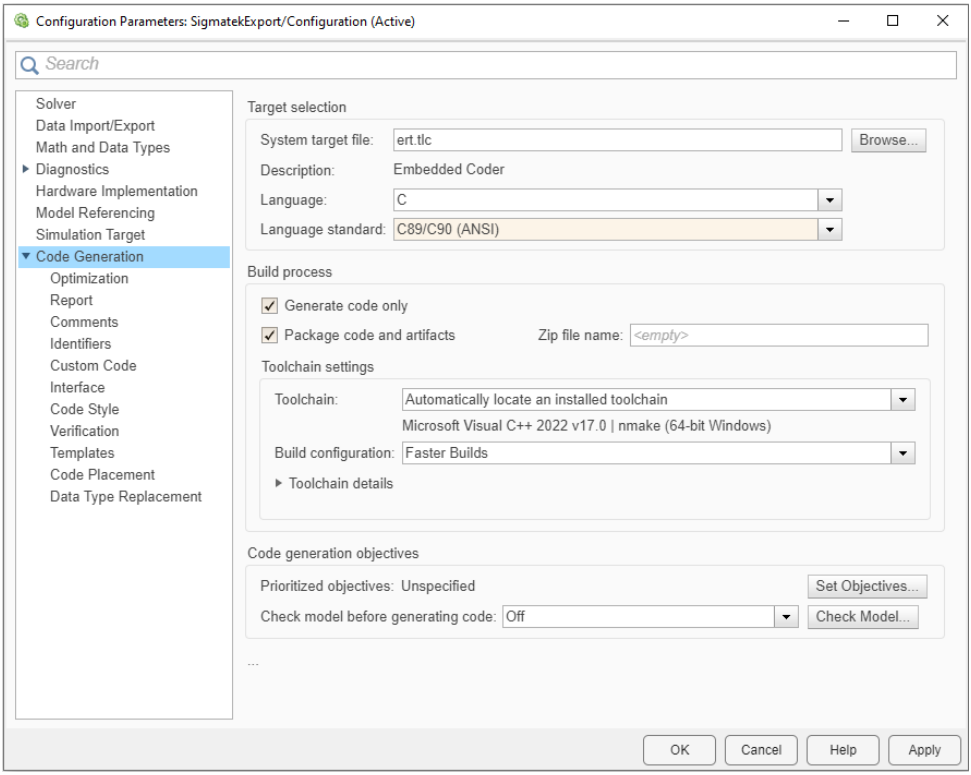
- false (mandatory)



Total number of instances allowed per top model:

- Multiple (recommended)

2.1.7.1114 Code Generation



System target file:

- ert.tlc (mandatory)

Language:

- C (mandatory)

Language standard:

- C89/C90 (ANSI) (recommended)

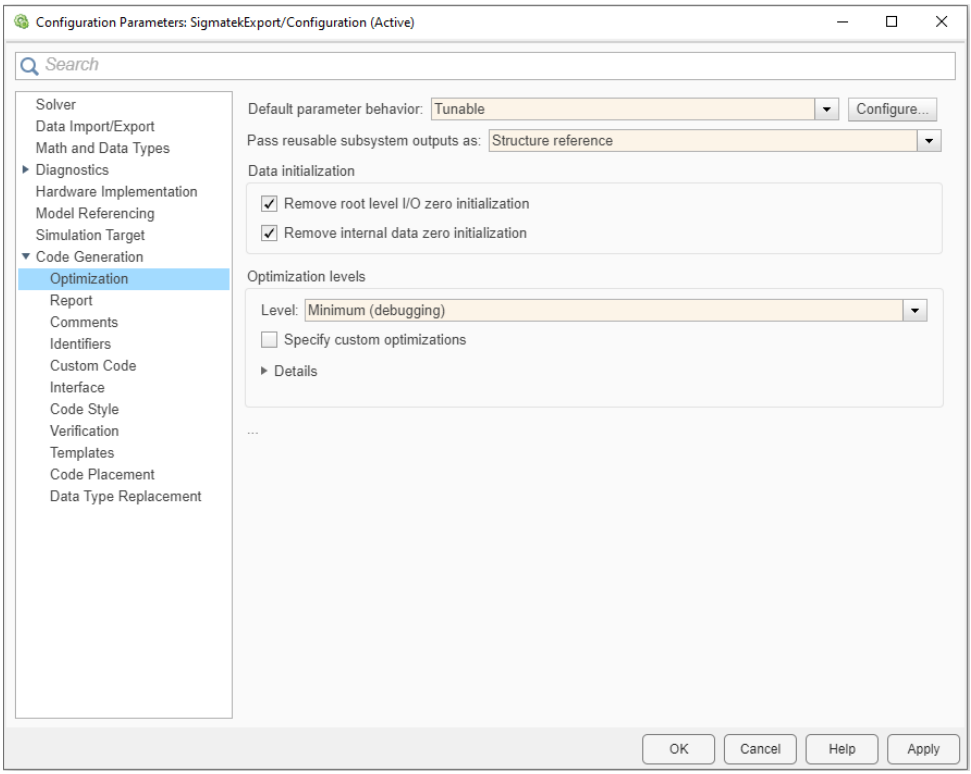
Generate code only:

- true (mandatory)

Package code and artifacts:

- true (mandatory)

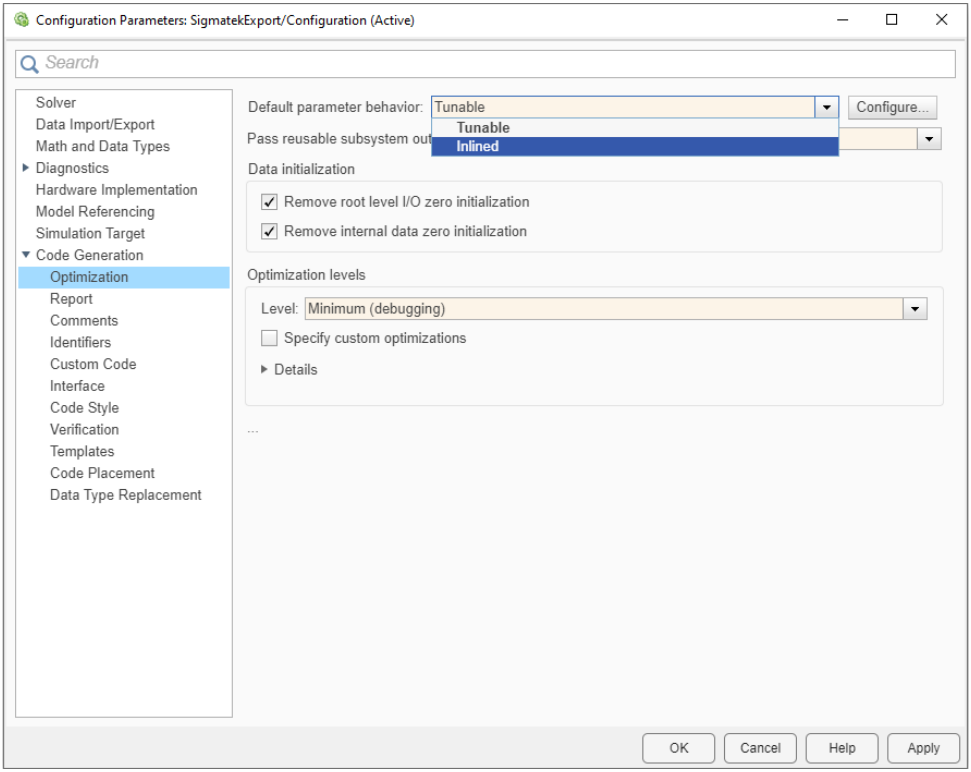
2.1.7.1115 Optimization



Default parameter behavior:

- Inline (recommended for production)

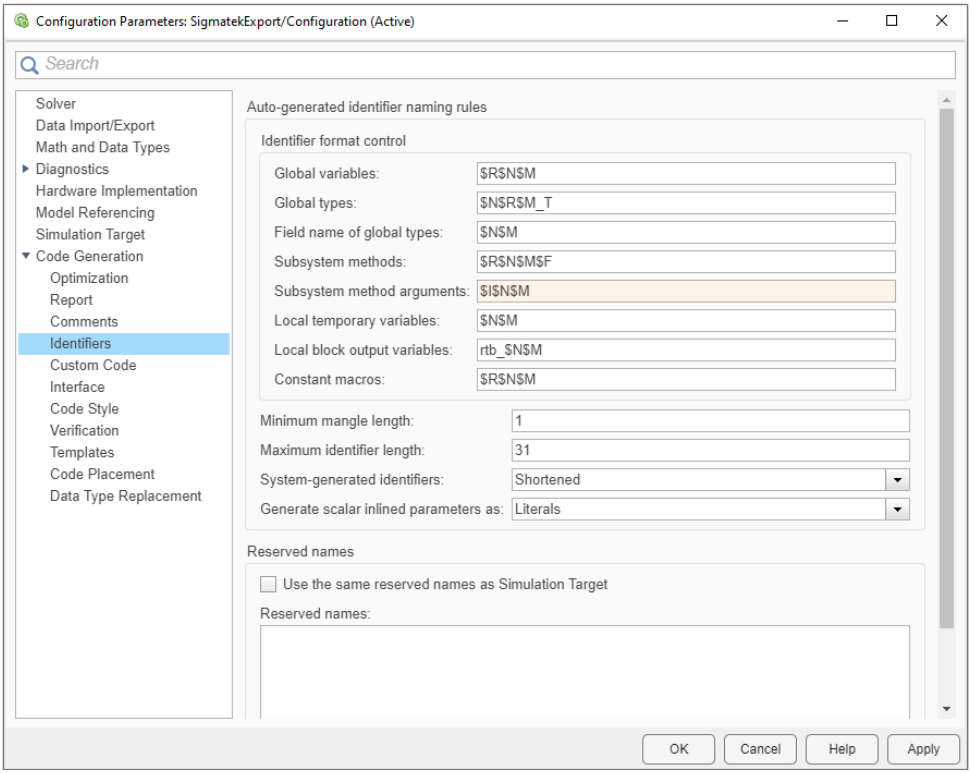
- Tunable (recommended for prototyping and debugging)



Optimization level:

- Balanced with readability (recommended for production)
- Minimum (recommended for debugging)

2.1.7.1116 Identifiers



Global variables:

- Add model name via $R to avoid name collisions between different models

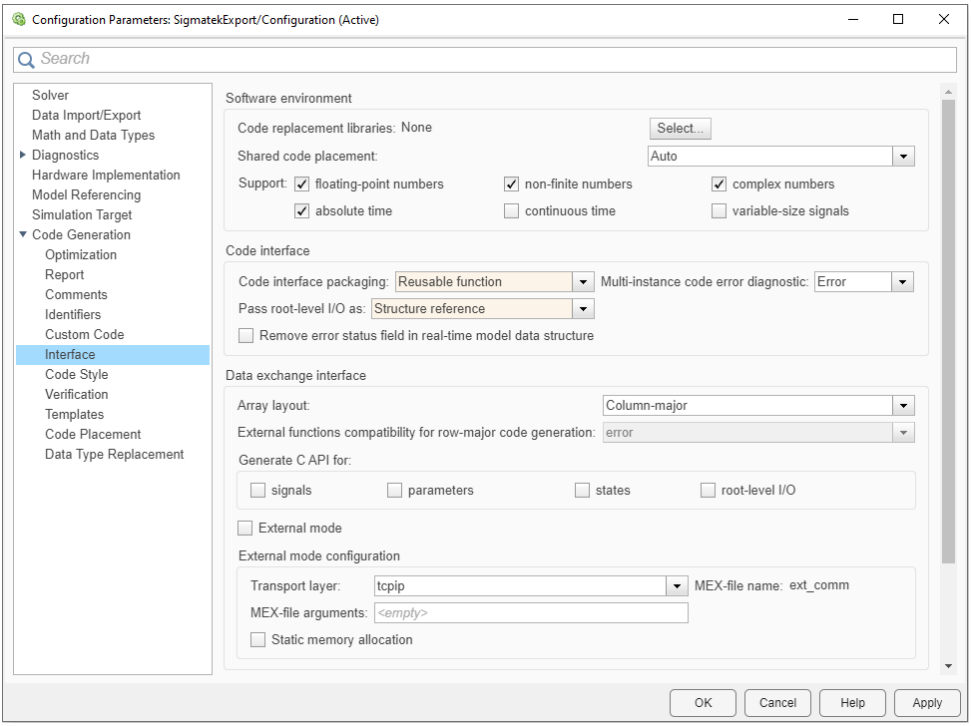
Global types:

- Add model name via $R to avoid name collisions between different models

Subsystem method arguments:

- Start the name with $I so the import in Lasal Class 2 can automatically detect I/O type for reusable models

2.1.7.1117 Interface

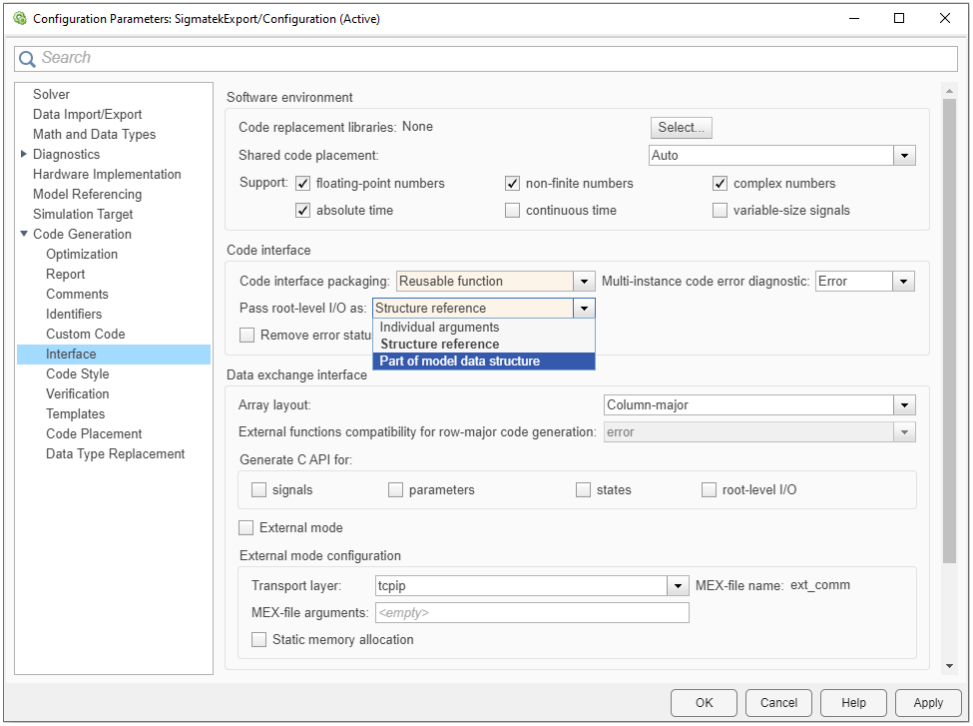


Code interface packaging:

- Reusable function (generally recommended, mandatory for multi instance models)
- Nonreusable function (supported, I/Os will be stored in global variables with naming as defined in Identifiers settings, only one model instance possible)

Pass root-level I/O as (only applies for Reusable function):

- Structure reference (recommended)
- Part of model datastructure (supported, does not support automatic detection of I/O type in Lasal Class 2 import)
- Individual Arguments (not supported if I/Os contain array datatypes)



The "**Advanced parameters**" area can be opened at the very bottom. The following settings have to be made in the checkboxes:

Support non-inlined S-functions:

- **on** (recommended)

Classic call interface:

- **off** (mandatory)

Use dynamic memory allocation for model initialization:

- **off** (mandatory)

Single output/update function:

- **on** (mandatory) - the Sigmatek integration expects a single model_step function as entry point, which is called by the Sigmatek real-time task system

Terminate function required:

- **off** (mandatory)

Combine signal/state structures:

- **off** (recommended)

Generate separate internal data per entry-point function:

- pre-deselected, cannot be changed

MAT-file logging:

- **off** (recommended)

Generate shared constants:

- preselected, cannot be changed

Create prepocessor directive in rwtypes.h:

- **off** (recommended)

Remove disable function:

- **on** (recommended)

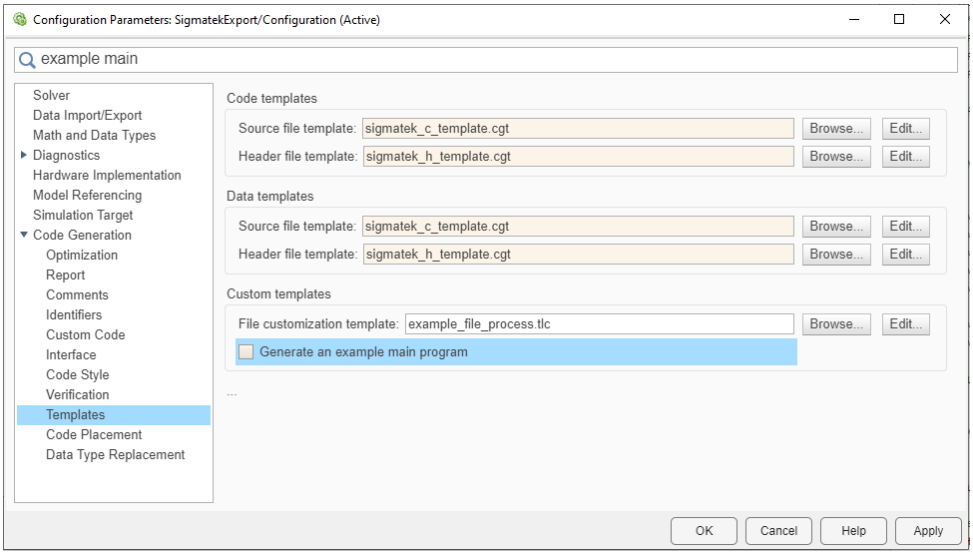
Remove reset function:

- **on** (recommended)

External mode testing:

- **off** (recommended)

2.1.7.1118 Templates



Source file template:

- Select SIGMATEK C template file (mandatory)

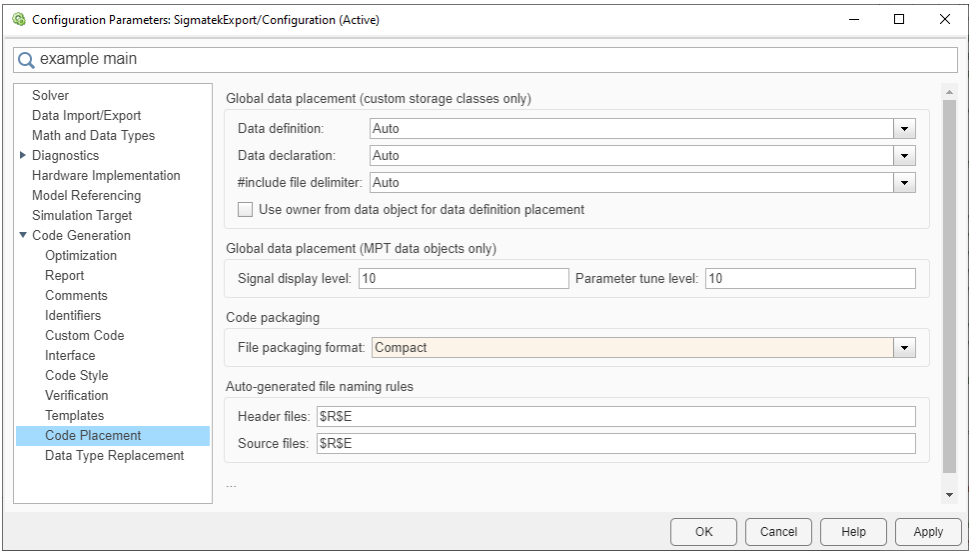
Header file template:

- Select SIGMATEK H template file (mandatory)

Generate an example main program:

- false (mandatory)

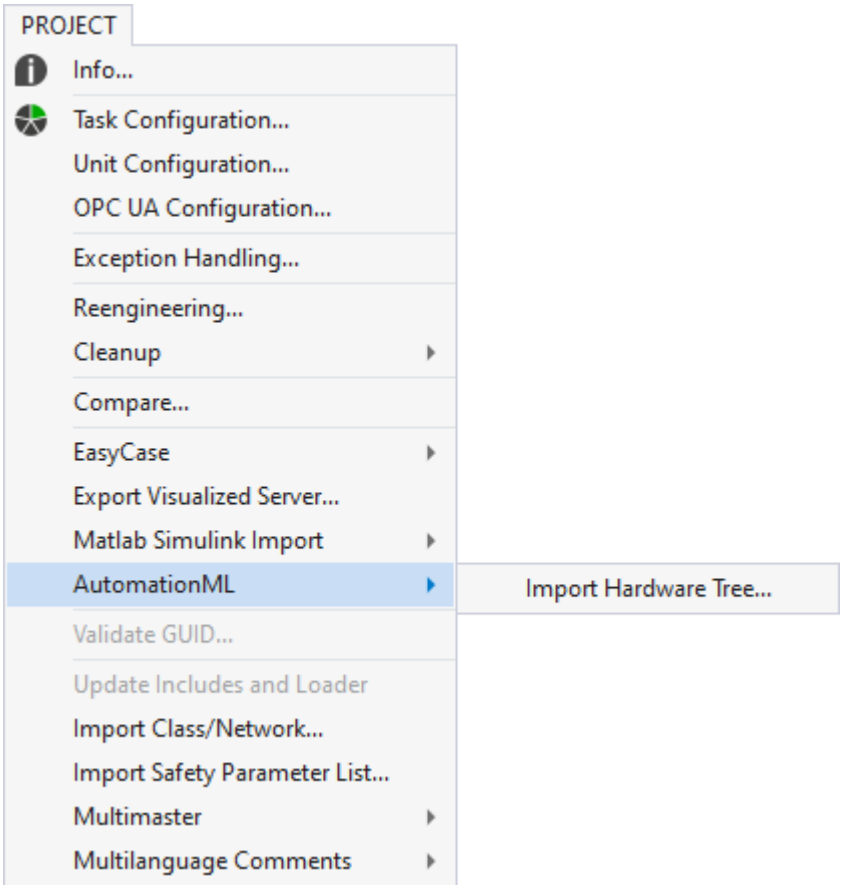
2.1.7.1119 Code Placement



File packaging format:

- Compact (recommended)
- Compact with separate data file (supported)
- Modular (supported)

2.1.7.12 AutomationML



To further simplify the implementation of projects and thus make their process more efficient and customer-friendly, **Lasal CLASS 2 offers the integration of EPLAN-AML**. This creates a **seamless transition** between the **circuit diagram** and the **project planning** in the PLC. Duplicate efforts or errors can thus be reduced to a minimum.

Functionality

Configuration of the hierarchy / topology of the hardware in the ePlan project

Important configuration fields

Import Hardware Tree
Adding modules

2.1.7.121 Functionality

- Overview
- Configuration of the hierarchy / topology of the hardware in the ePlan project
  - Configuration project
  - Station ID
  - Rack ID
  - PLC card is placed on rack ID
  - Position (slot / module)
- Important configuration fields
  - PLC box
  - PLC and bus connection points (bus connections)
  - PLC and bus connections (IO connections)
- Export format

Overview

EPLAN is a tool with which circuit diagrams / hardware structures can be drawn and exported in *.aml format (= AutomationML format - a special XML format). The **EPLAN Data Portal** has already included a large number of SIGMATEK modules for several years.

Projects created in the **EPLAN tool** from SIGMATEK components can be **exported in *.aml format**, and **imported into a LasalClass2 project** using the Import Hardware Tree menu item. This functionality is also accessible via the Hardware Editor.

Configuration of the hierarchy / topology of the hardware in the ePlan project

Die Hierachie/Topologie der Hardware in einem ePlan Projekt muss vom Benutzer durch einige Parameter festgelegt werden, diese Parameter bilden das Grundgerüst eines Hardwareaufbaus in einem ePlan Projekt.

Alle folgenden Einstellungen können im SPS-Kasten unter dem SPS-Strukturdaten-Reiter gefunden werden.

Configuration project

Das Konfigurationsprojekt stellt einen kompletten Hardwareaufbau dar, der im Lasal CLASS 2 importiert werden soll. Ein Konfigurationsprojekt kann mehrere Stations, sollte aber **nicht mehr als eine CPU** beinhalten (SIGMATEK Safety CPUs sind von dieser Regel ausgeschlossen). Bei allen Hardware Modulen, die zu diesem Projekt gehören sollen, muss **dasselbe Projekt als Konfigurationsprojekt** eingestellt sein.

Station ID

Each bus transition that passes through a cable **must create a new station**, i.e. each module that is connected via a Varan cable, for example, must represent a new station. **SDIAS bus transitions**, on the other hand, **should not receive a new station**. Furthermore, **each station may only have one rack** (DIN rail / terminal strip).

Rack ID

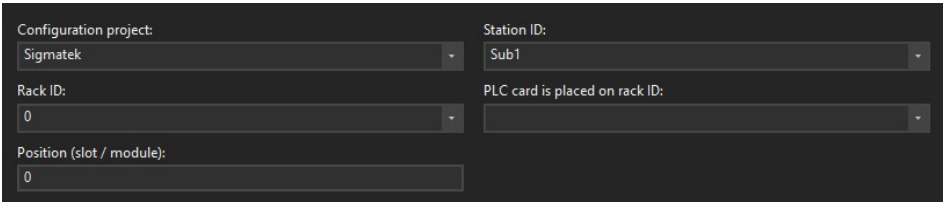
A rack represents a DIN rail / terminal strip on which the individual SDIAS modules are placed. Accordingly, the module housing is also used to detect which SDIAS modules are connected to each other. The CPU or the bus coupler should always be used as rack - however, a dedicated DIN rail may also be used.

PLC card is placed on rack ID

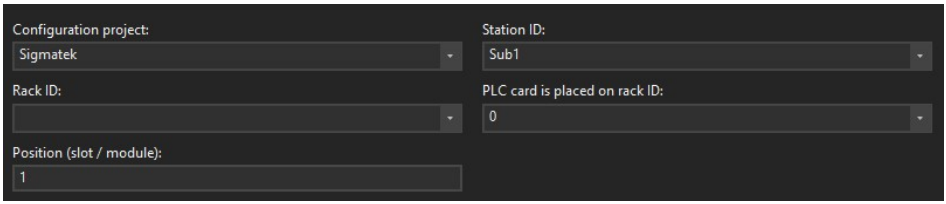
Assignment of the SDIAS cards, **on which rack (ID) a certain card is plugged**. It is important here that the card which itself represents the rack **must not be on a rack**. In this case this field remains empty. The same applies to the cards that are stuck on the carrier, these **must not have a carrier entered**.

Position (slot / module)

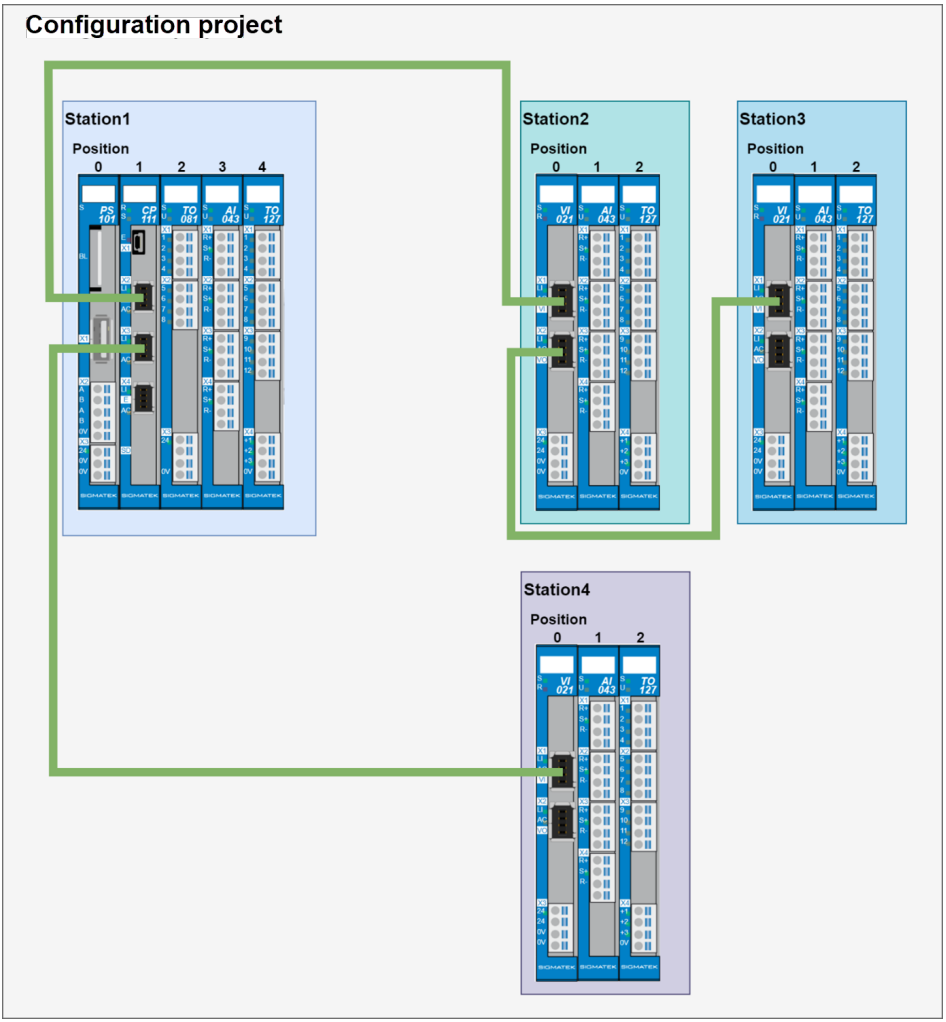
Indicates at which position of the rack the module is plugged. The first module starts with position 0.



Example of a bus coupler (VI021)



Example of an SDIAS card located behind the rack



Example of the topology structure of a project

Important configuration fields

Certain rules must be observed in the EPLAN tool if the import into LASAL CLASS 2 is to be error-free.



Tip

The SIGMATEK function templates from the ePlan Data Portal can be used. These already have preconfigured fields, which reduce the configuration effort in ePlan.

The following configuration fields must be observed in particular:

PLC box

• PLC box tab

PLC type designation	Here the device name of the SIGMATEK module must be specified, e.g. AI088. A class name can also be specified (e.g. AI088_Diag), but this option should only be used if it is a derived hardware class.
PLC card name	Here the later object name in LASAL CLASS 2 must be specified. This should be unique, otherwise complications can arise during import. Furthermore, the name must follow the SIGMATEK naming conventions.
Drive [x]	The drive [x] entries are required if axes exist in the ePlan project. With these entries, axes and their drives can be "connected". To connect the axis with the drive, the same value must be entered in both boxes for the same drive [x] parameter (e.g. "Axis1").

• PLC structure data tab

CPU	Determines whether the module represents a CPU and must be set at the respective CPU of the project.
Bus coupler / head station	Must always be set at bus coupler res. as soon as a bus transition is at this module. At each station at least one module should be a CPU or a bus coupler.
CPU: Name [1]	This parameter shows the affiliation of the hardware to a certain CPU. Here the same name / reference must be entered for the CPU and for all hardware modules that belong to the CPU. If this is not the case, e.g. IO Connections are not correctly assigned to the CPU and therefore not considered during export. The name should be <Configuration project>.<Station name of CPU>.<CPU name> (e.g. Sigmatek.Station1.0).

PLC and bus connection points (bus connections)

• PLC connection point tab

Plug designation	Here, in the case of bus connections, the correct name of the slot on the hardware must be specified. e.g. for a CP111, the name X2 must be specified for the first VARAN port. The plug name

	is required by LASAL CLASS 2 to determine the port number of the VARAN connection.

- **Bus data tab**

Bus system	The bus system used by the bus connection. It is important that the same bus system is set for both bus connections when entering a bus connection, otherwise the connection will not be exported correctly.
Property "Bus interface: Name"	Here the name of the interface must be set. It is to be considered that all ports of a bus system are in the same interface. E.g. with a CP111 the two VARAN ports are in IF1, and the one Ethernet connection in IF2. The interface is required by LASAL CLASS 2 to determine the port number of the VARAN connection (this is the fallback system if connectors are incorrectly labeled or not available).

PLC and bus connections (IO connections)

- **PLC connection tab**

Channel designation	The number of the IOs must be specified here. For example, if output 5 is to be handled, the number 5 must be specified. This number is used by LASAL CLASS 2 to determine the server of the hardware class. Here also several pins can have the same number, in such a case the same HW class server is used for all these pins.
Symbolic address	The symbolic address of the connection is specified here. This field must be filled in so that the connection can be exported from EPLAN. If no value is entered in the "Address" field, the symbolic address is used as the name for the IO connection.
Address / PLC address (in the assignment list)	Here the name of the IO connection is entered, which it is to receive in LASAL Class 2. This name is not allowed to contain any special characters except '.' and '_'. If this field is left blank, the symbolic address is used as the name for the IO connection.
Functional text	A comment for the IO connection can be specified here, this comment is then transferred to the IO Connection Manager for the IO connection.

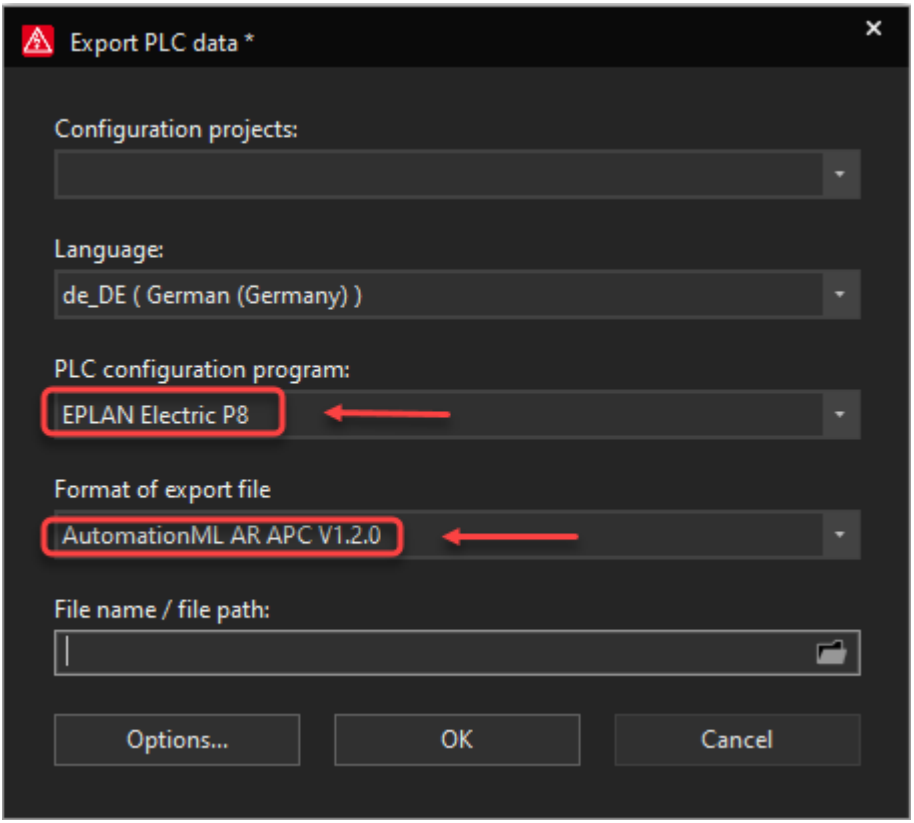


Information

The same setting fields can also be configured via the "Assignment list" dialog in EPLAN!

Export format

In order to use the EPLAN export file in LASAL CLASS 2, certain criteria must be met. If you call the export dialog in the **EPLAN tool**, there are different possibilities to determine the format of the created ***.aml file**:



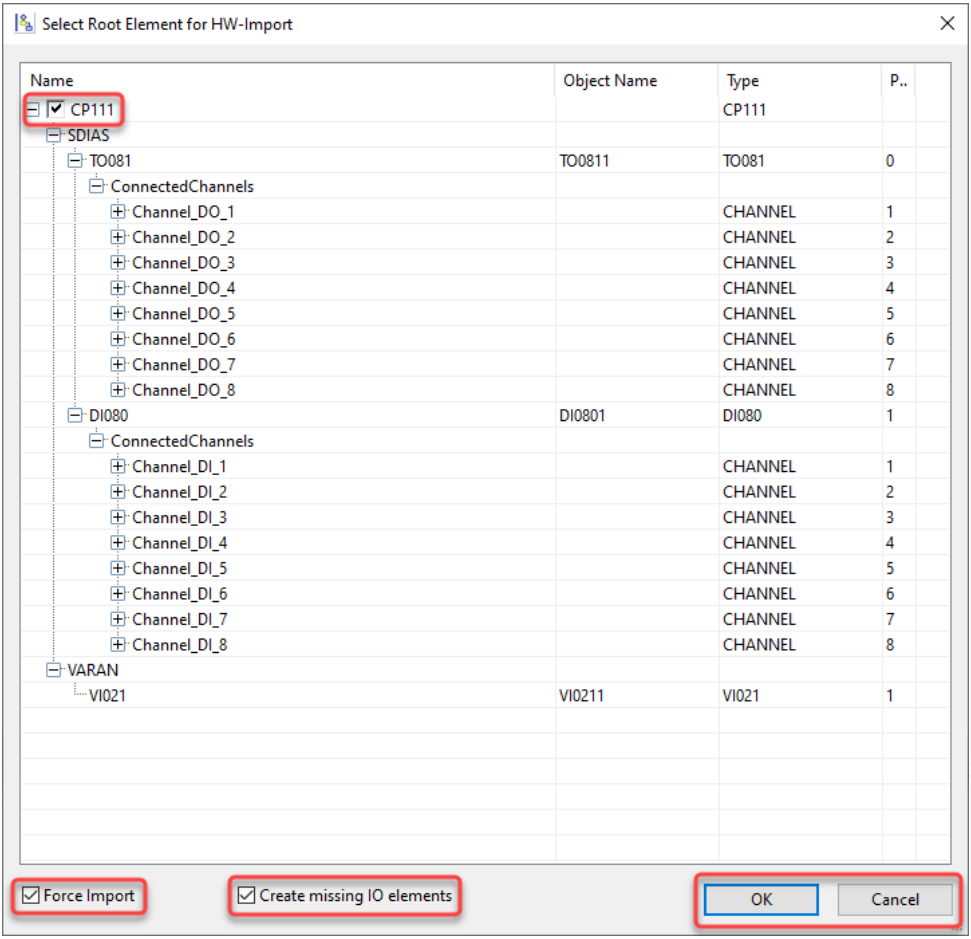
As you can see in the marked places in the picture, both **the PLC configuration program** and **the required format of the export file** must be specified in the export dialog of the **EPLAN tool**. In the picture, the correct entries have already been made. In summary:

PLC configuration program	**EPLAN Electric P8**

Format of export file	**AutomationML AR APC V1.2.0**

2.1.7.122 Import Hardware Tree

With this menu item, a project saved in *.aml format (AutomationML) can be imported into LASAL CLASS 2. After selecting an *.aml file, the **Import dialog** appears with the detected structure of the hardware configuration.



An *.aml project can also contain several CPUs, but in a LASAL CLASS 2 project there can only be one CPU. In such a case there is a selection box in the import dialog for each CPU, **only one CPU can be selected here**.

The **import dialog** contains a table of the hardware structure to be imported, the checkboxes "**Force import**" and "**Create missing IO elements**" as well as the obligatory buttons "**OK**" and "**Cancel**".

In the table you can expand or collapse the corresponding branches in the column "**Name**" with the plus and minus symbols. In front of each **CPU** there is a **checkbox**. Since more than one CPU can be included in an **EPLAN** project, but only one CPU may be present in LASAL CLASS 2, **only one CPU can be selected here**. In this column, **the displayed value cannot be changed**.

The column "**Object Name**" contains the name of the corresponding network object. You can **double-click** on the field and then **change the value**.

The "**Type**" column represents the data type of the object in the row - it can be a hardware class, IO object, connection type. This column is **not editable**.

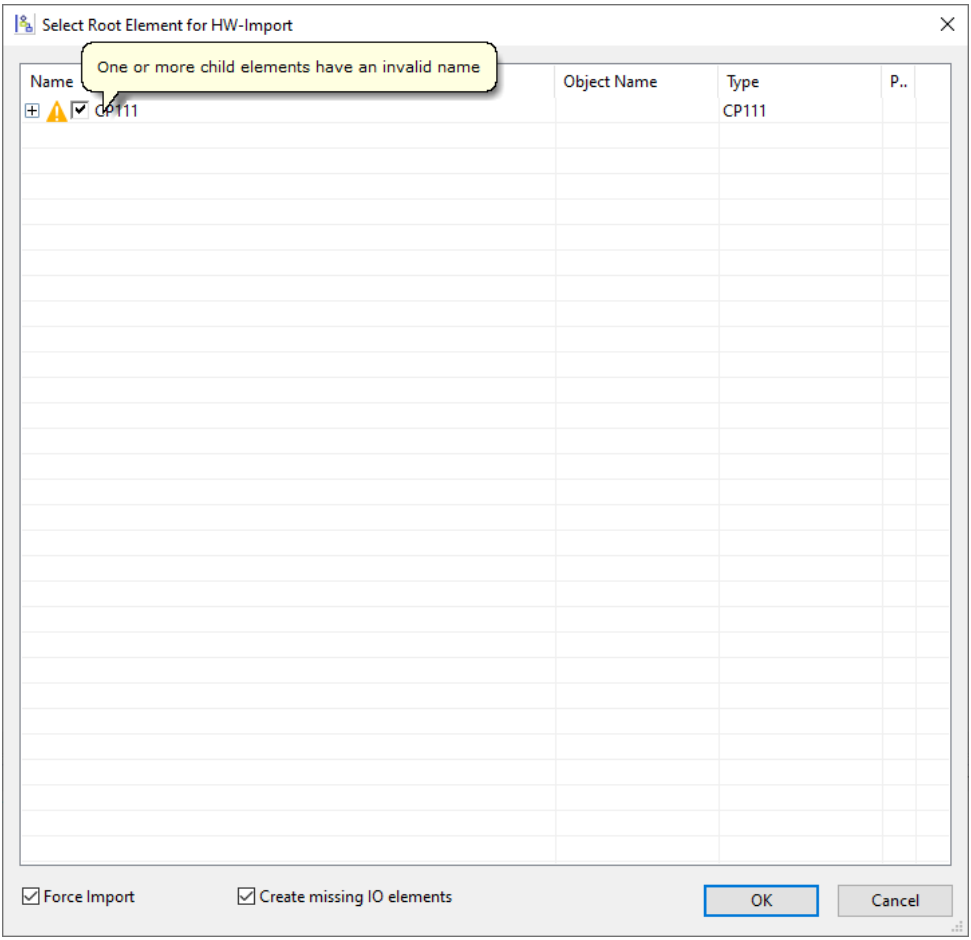
The column "**Pos**" shows the position within the parent node - e.g. the place number at a CDIAS module, or the SDIAS slot. This column is also **not editable**.

The checkbox "**Force import**" causes LASAL CLASS 2 **not to abort the import** if unknown hardware is found.

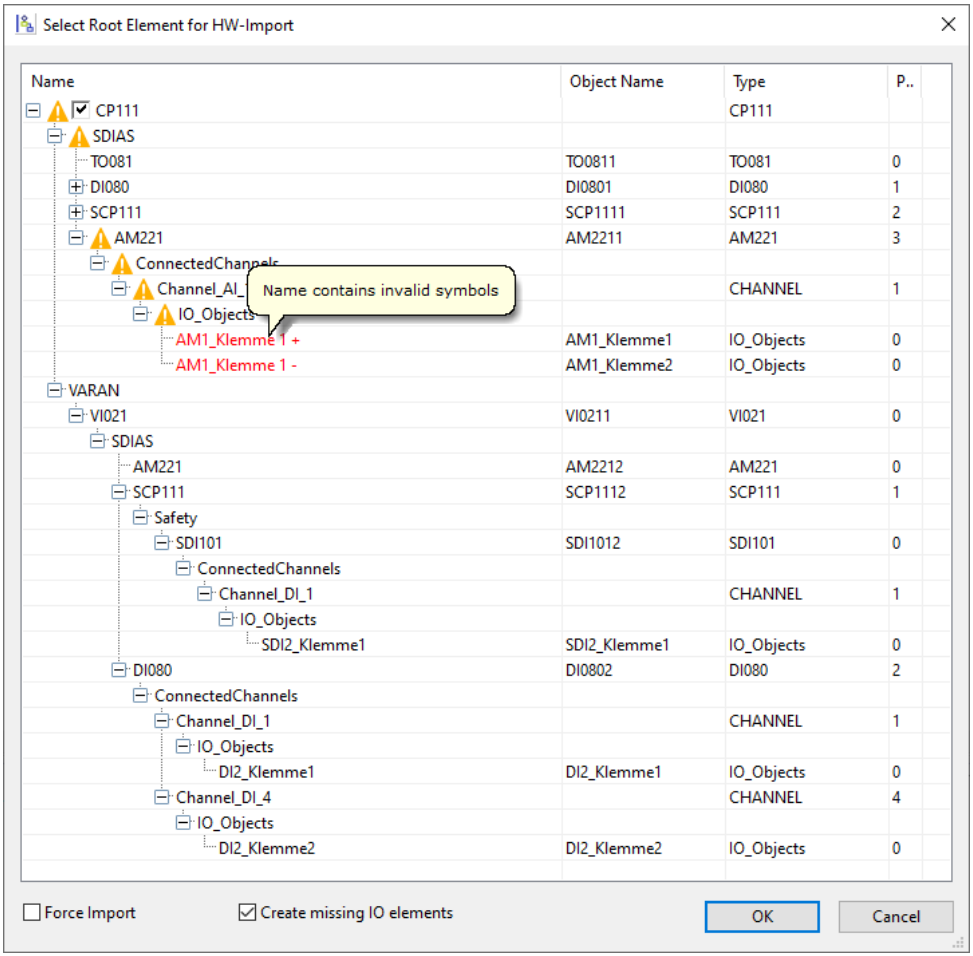
The "**Create missing IO elements**" checkbox controls how to proceed with IO elements that are present in the import file. **The IO elements that already exist** in the project **are overwritten**. Those that **only exist in the import file**, but not in the LASAL CLASS 2 project, **are created if this checkbox is selected**, **otherwise** they are **discarded**.

With the button "**OK**" the import is executed, with "**Cancel**" it is aborted.

If LASAL CLASS 2 encounters names during the import that are **not SIGMATEK compliant**, a **yellow triangle is displayed in front of the name of all parents of the element**, up to the root. If you move the mouse over this sign, you get the explanation as a tooltip:



Now the element can be expanded until the cause of the problem is found. The corresponding name is displayed **in red** and the reason for the error is explained in the tooltip:

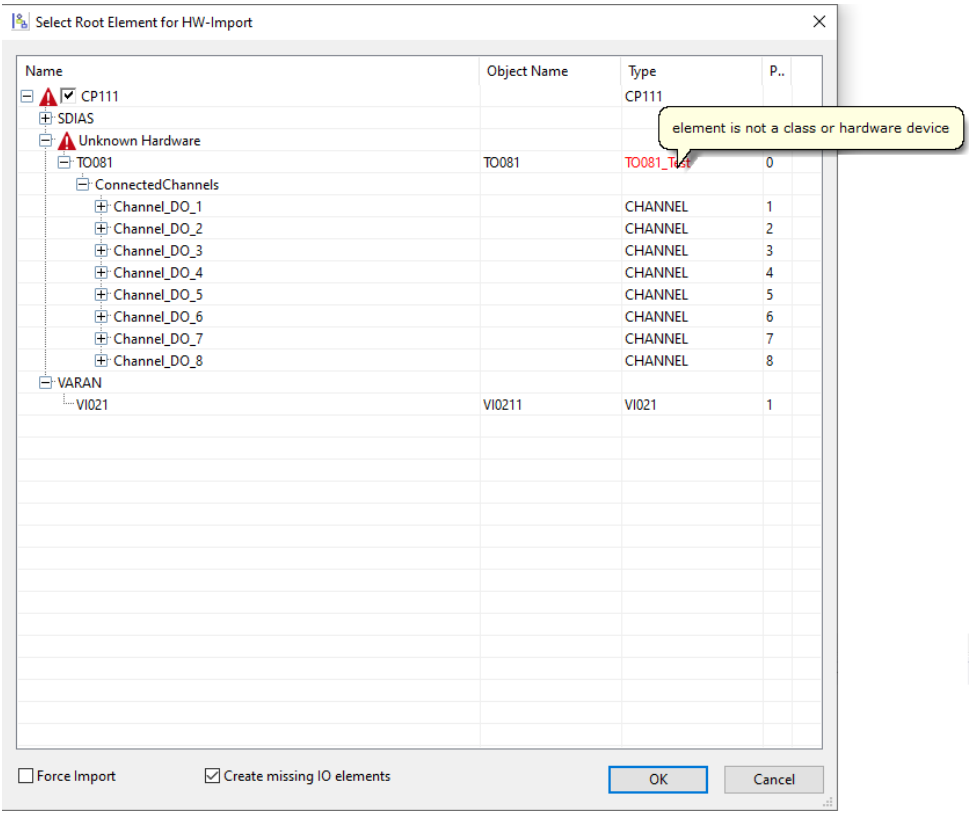


LASAL CLASS 2 **changes**, if necessary, the associated object name **to a SIGMATEK-compliant variant** (with several names of the same name usually with ascending digits at the end), which can be edited with a double-click in the column.

Attention

The originally incorrect designation in the **"Name" column cannot be changed!**

Likewise, it can happen that an unknown type is specified in the *.aml document. In this case, not a yellow but **a red triangle is displayed in front of all parents** of the incorrect type designation:



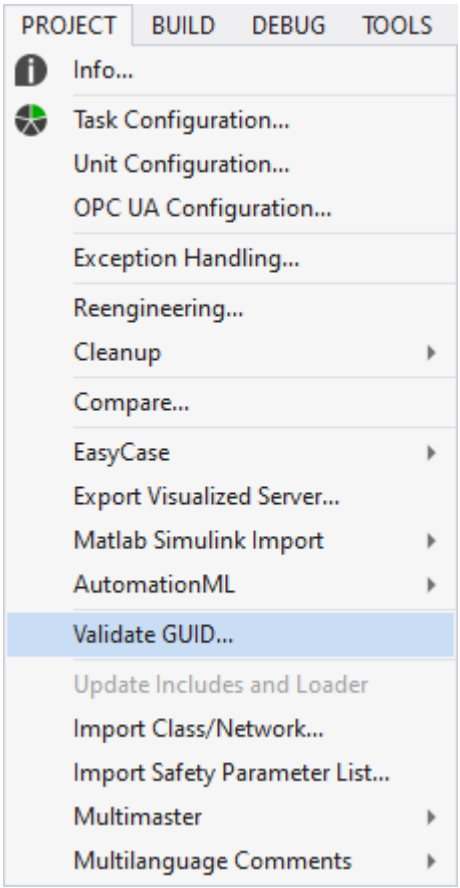
Here, too, the wrong content is marked by red font color as well as the reason is displayed by means of a tooltip when the mouse is moved over it, and here, too, it is not possible to correct the wrong content directly in the dialog. In contrast to non-LASAL-compliant names, however, in this case LASAL CLASS 2 cannot correct the unknown / incorrect type itself. In the displayed tooltip it can be read that no class / type can be associated with the name specified in the type column.

Attention

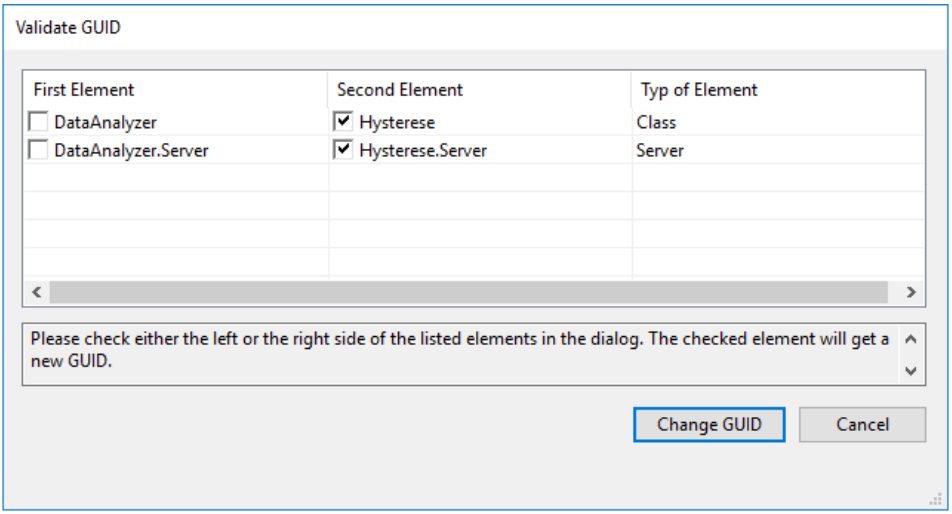
The imported *.aml file **should** therefore **be corrected in the EPLAN tool** as well as **re-exported** and **imported into LASAL CLASS 2!**

About import of branches of an AML hardware structure at a module interface see the chapter Insert modules.

2.1.7.13 Validate GUID



Opens the **Validate GUID window** to correct doubled GUIDs. The menu item is can only be selected when there are doubled GUIDs in the project.



2.1.7.14 Update Includes and Loader

Later activates the option, which can already be selected with **Create New Project** using a checkbox: the loader and all includes are automatically saved with the project.

2.1.7.15 Import Class / Network

With this function, previously exported classes / networks can be imported into the project.

Attention

It is important to note that vertices are ignored during import

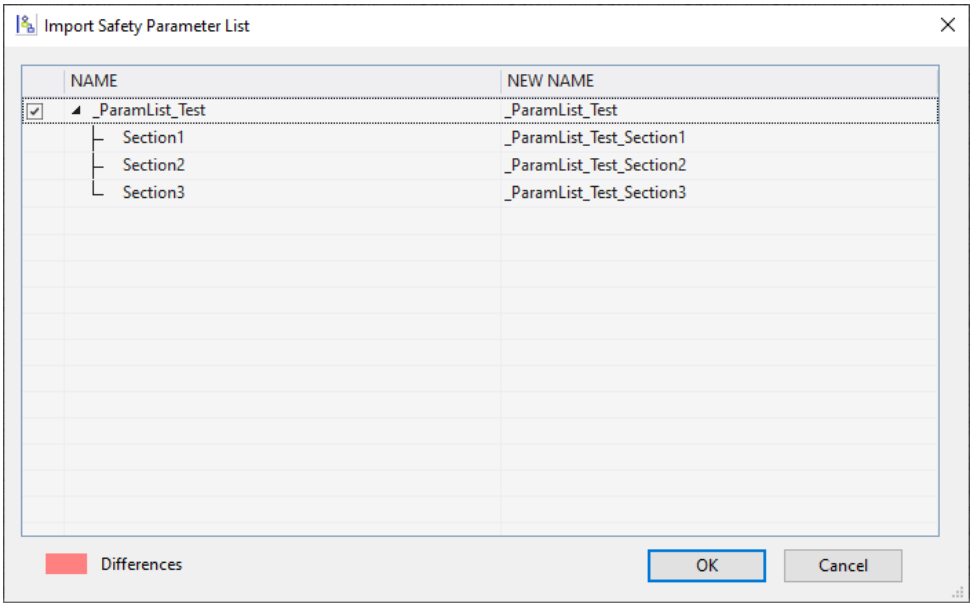
2.1.7.16 Import Safety Parameter List

It is possible to import the **exported files from the Safety Designer** into LASAL CLASS 2. These are special files in *.xml format. Based on the information in the *.xml file, **a global type is created in LASAL CLASS 2**. Through this type a possibility of comparison between project and PLC is created.

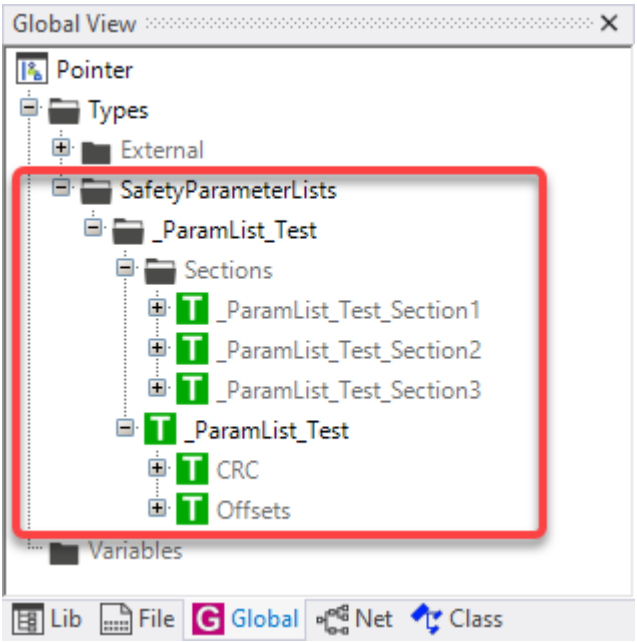
In addition to this menu item, this function can also be called up in the context menu of the project in the GlobalView:



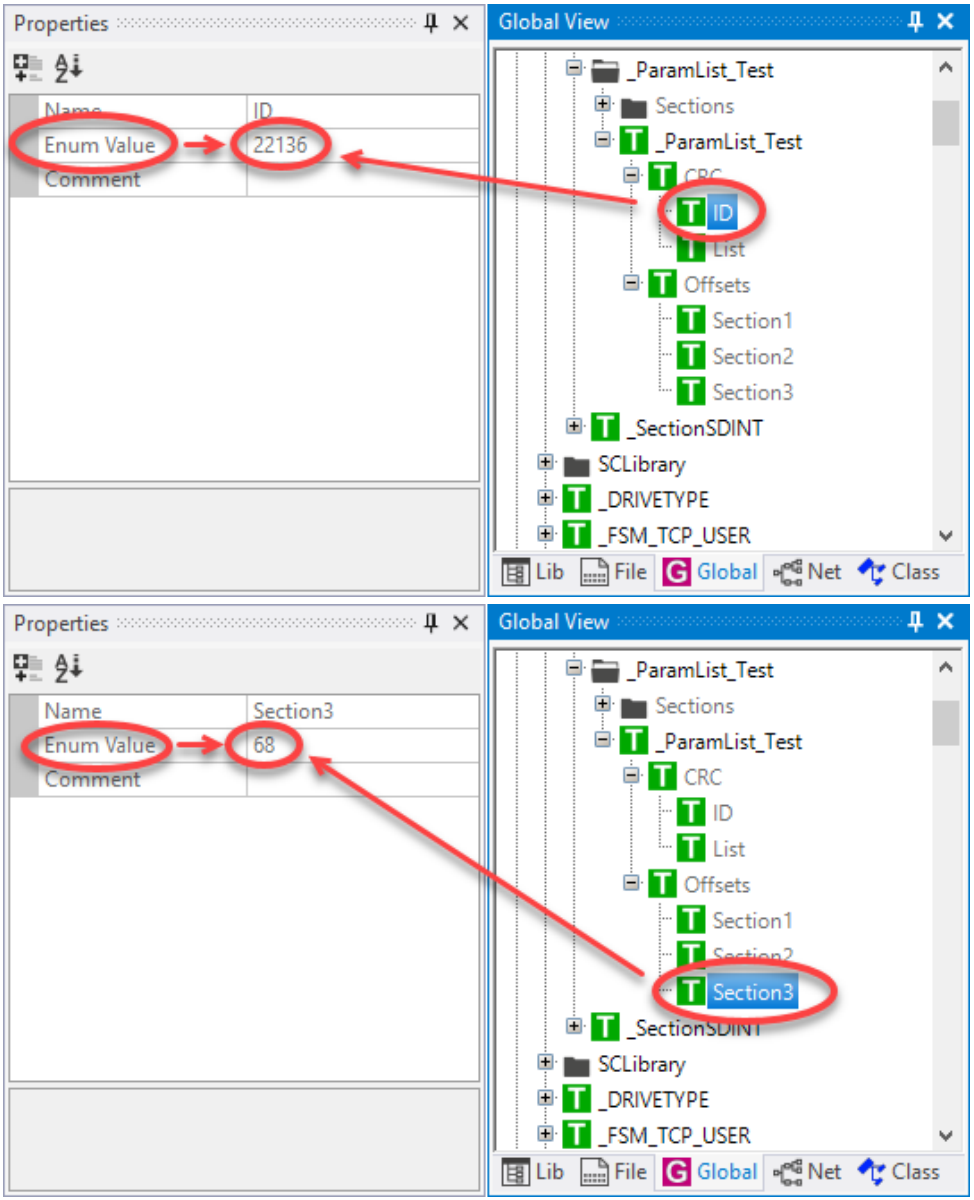
After selecting the "Import Safety Parameter List" menu item, the desired export file is selected in a file selection window. If it is a correct Safety Designer export file, the contents are displayed in the following window:



If confirmed with **OK**, a **global type** (in our example"_ParamList_Test") is created. This has the following structure:

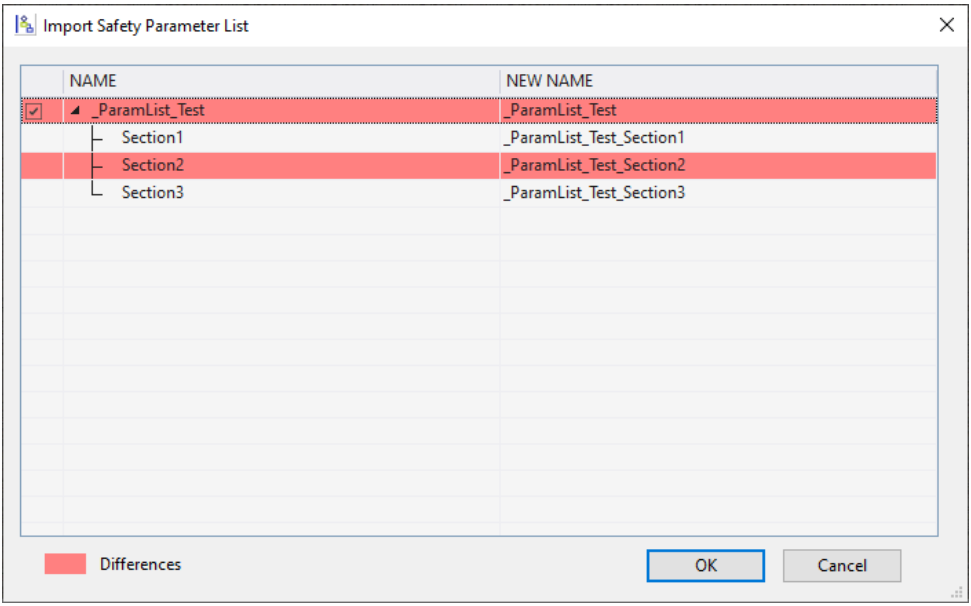


So there is a folder with the name of the parameter list (preceded by underscore) and in it the individual parameter sections as subtype. There is also the parameter list as a type, which has the same name as the folder. The type contains two subelements: the CRC and the offsets for the sections in bytes. Both the CRC and the offsets are **automatically predefined enum variables**:

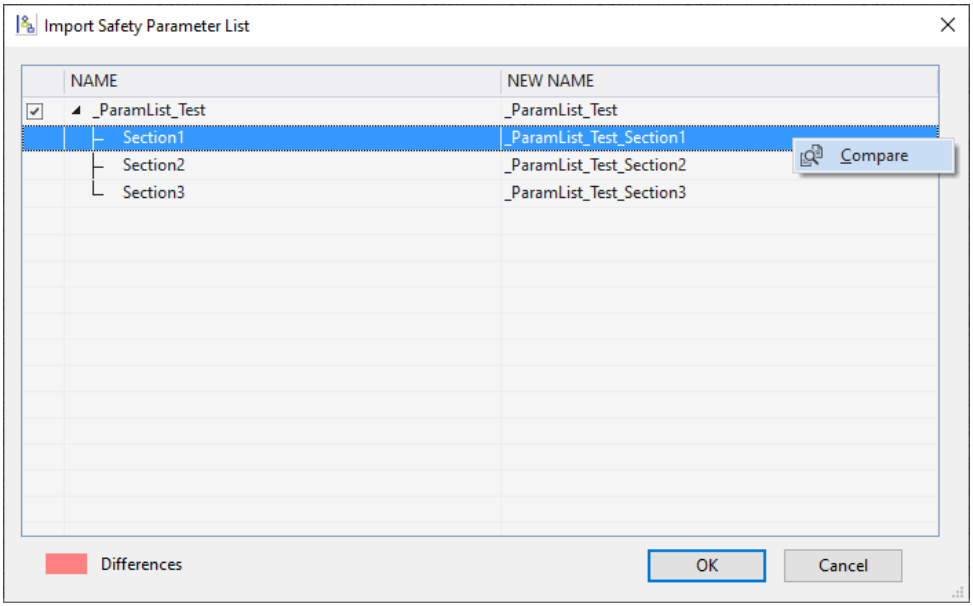

Properties
Name
ID
Enum Value
22136
Comment
Global View
_ParamList_Test
Sections
_ParamList_Test
CRC
ID
List
Offsets
Section1
Section2
Section3
_SectionSDINT
SCLibrary
_DRIVETYPE
_FSM_TCP_USER
Lib
File
Global
Net
Class
Properties
Name
Section3
Enum Value
68
Comment
Global View
_ParamList_Test
Sections
_ParamList_Test
CRC
ID
List
Offsets
Section1
Section3
SCLibrary
_DRIVETYPE
_FSM_TCP_USER
Lib
File
Global
Net
Class

If you **change** the type name, **both the folder name and the section names will be changed** accordingly. If you **delete** the folder or the parameter list type, **the entire folder and its contents will be deleted**.

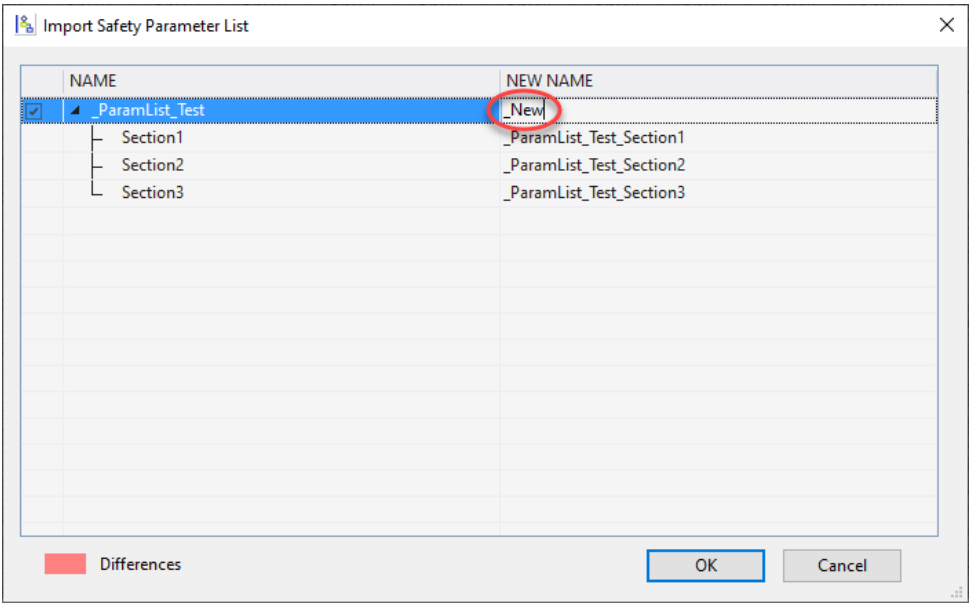
If a Safety Parameter List has already been imported, and it is imported again, any **differences** that may exist **will be highlighted in red**:



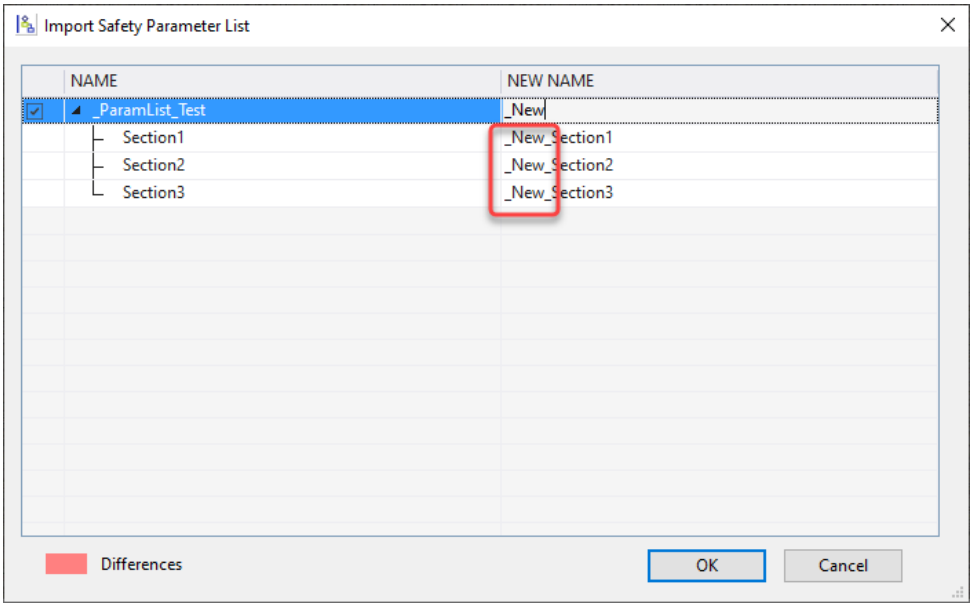
To see the differences, select the "**Compare**" item in the context menu of the parameter list or the section:



It is possible to **change the name of the parameter list type** by double click directly here, in this dialog:

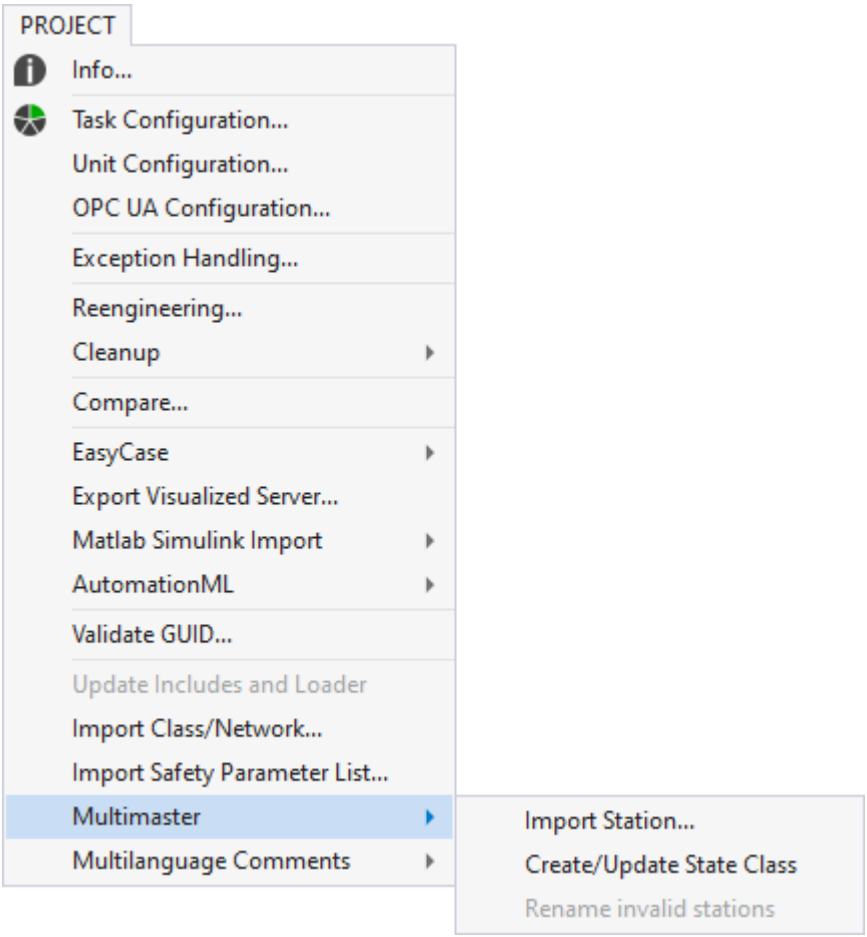


If the name is accepted, the sections will be renamed automatically:



With **OK** the new list is **imported** and the **old one is overwritten**, with **Cancel** (or unchecking the checkbox) **nothing is imported** but the **old list is kept**.

2.1.7.17 Multimaster

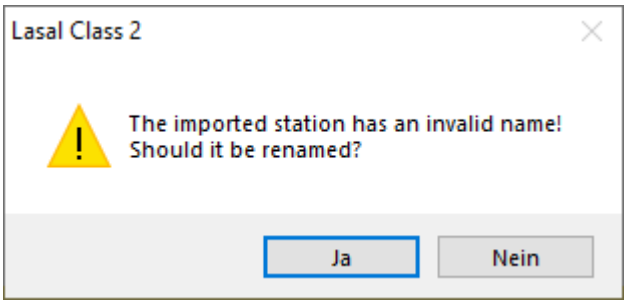


This menu item contains three sub-items, which are important when managing the communication between different controls via **Machine Manager** .

Import Station

By selecting this menu item you can import a station from the **Machine Manager** via a **Multimaster Export File (*.mme)** . If this station has an invalid name (name of the station

starts with a number, contains illegal special characters, etc.), a dialog box is displayed asking whether the name of the station should be changed to a valid name:



Here the change can be made according to desire, or the original name can be used further.

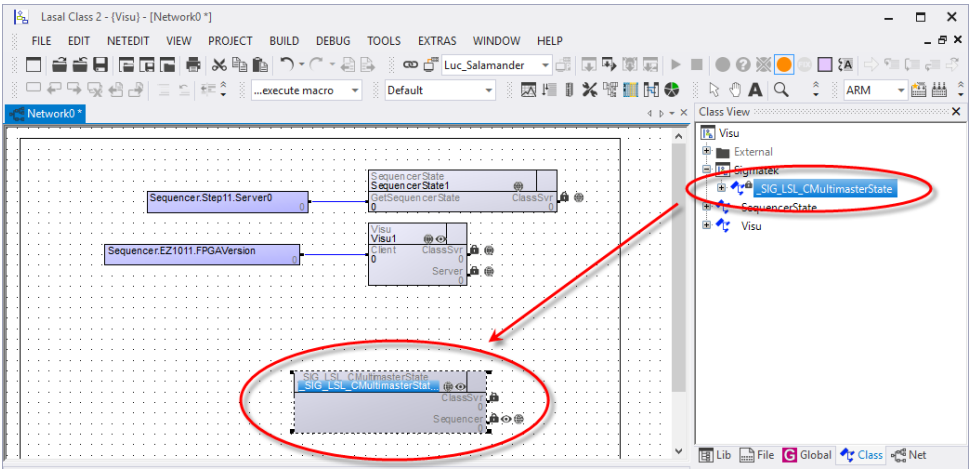
Caution!

With invalid station names, the project can no longer be compiled without errors when using the MultimasterState class!

If you want to rename the invalid station names later, there is the third menu item "**Rename invalid stations**". This is only enabled if LASAL CLASS 2 has detected invalid station names.

Create / Update State Class

Here the class **_SIG_LSL_CMultimasterState** is created or updated.

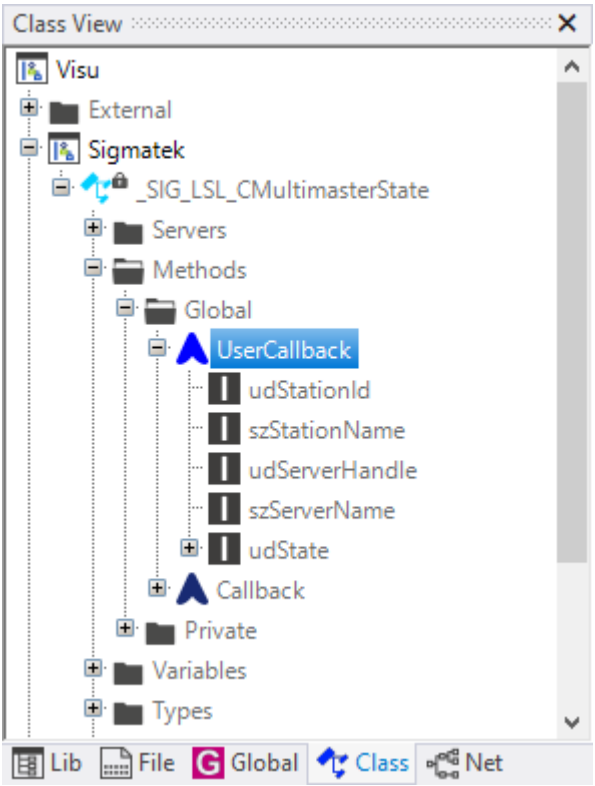


This class is used to retrieve the station status. For each station, a server is added to the class. These servers show whether the station can be reached and if it's servers are initialized. Initialized means that a value has been successfully transmitted at least once.

The following statuses can be displayed:

Offline	Station is not accessible or is not in "Run Ram\Rom" mode.
Online_Initializing	Station can be reached, but the server is not yet initialized.
Online	Station can be reached and all server of the station are available and initialized.
Online_NotAllServerAvailable	Station can be reached, but not all servers are available. The ones that have been located are initialized.

Using the **"UserCallback"** method of the class, status changes can be recognized (to be used like LSL_Exception_Handling_Class).

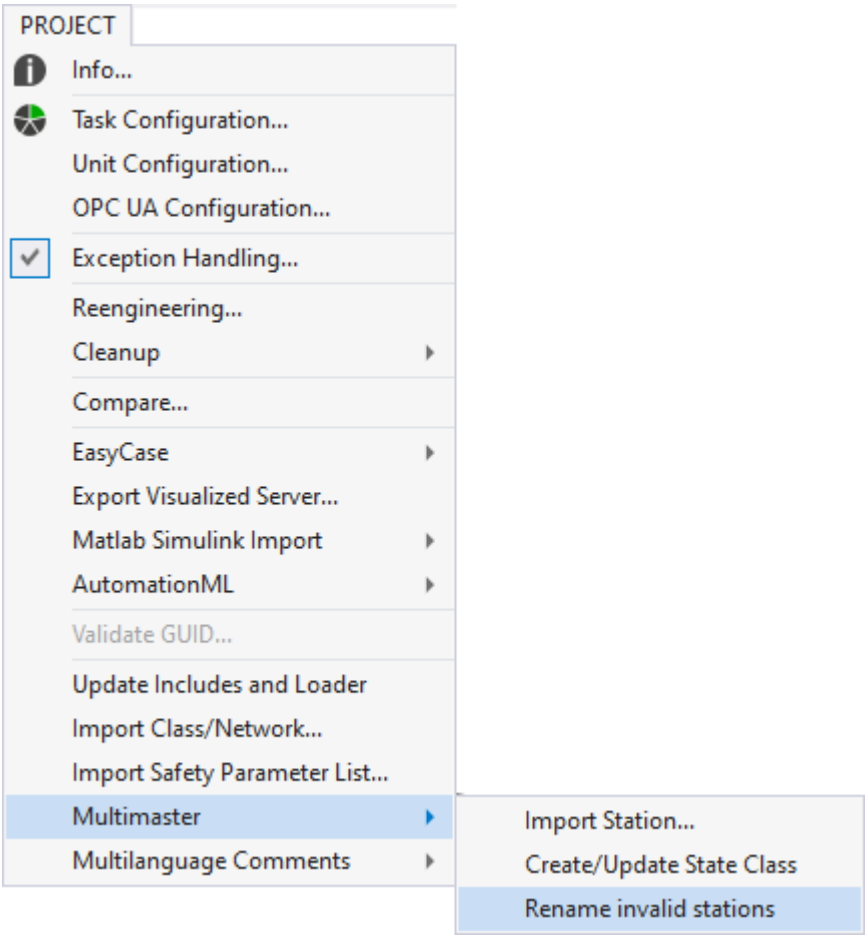


The "UserCallback" is called with each change in the station's status. The state is determined via the **udState** parameter. Which station it is can be read via **udStationId** or **szStationName** .

There is an also an additional callback for each server not found in the station: In this case the parameters **udServerHandle** and **szServerName** are additionally filled and the value of udState is "SvrNotFoundAtStation".

Rename invalid stations

Each time the Multimaster submenu is opened, LASAL CLASS 2 checks for invalid station names. If this is the case, the **Rename invalid stations** menu item becomes selectable:



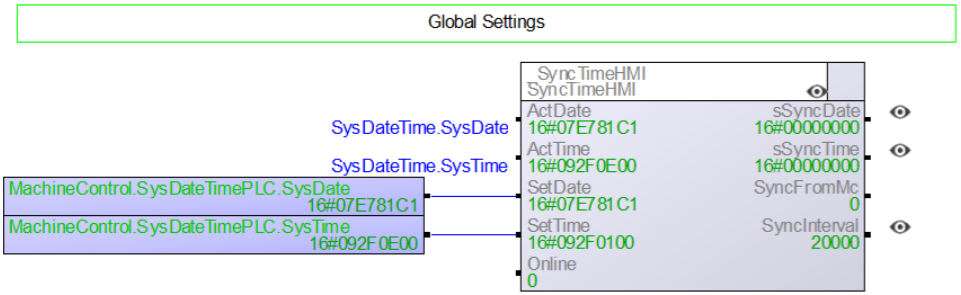
If this menu item is selected, the station names are made LASAL CLASS 2-compliant. This is done according to the following guidelines:

1.) If the invalid name starts with a number, the name is preceded by "Station_".

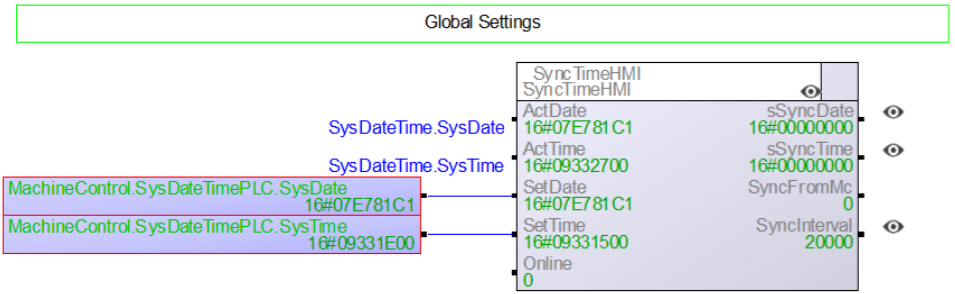
2.) Unallowed special characters (e.g. +, -, &, %, /...) are replaced by "_".

Special case of "red frames"

If the connection between the stations works, the corresponding IO objects are displayed normally (**in a black frame**) in the object network:



If the connection to the remote station is no longer available for some reason (e.g. because the cable was unplugged, the IP address was entered incorrectly, etc.), the corresponding IO objects in the network are **framed in red**:

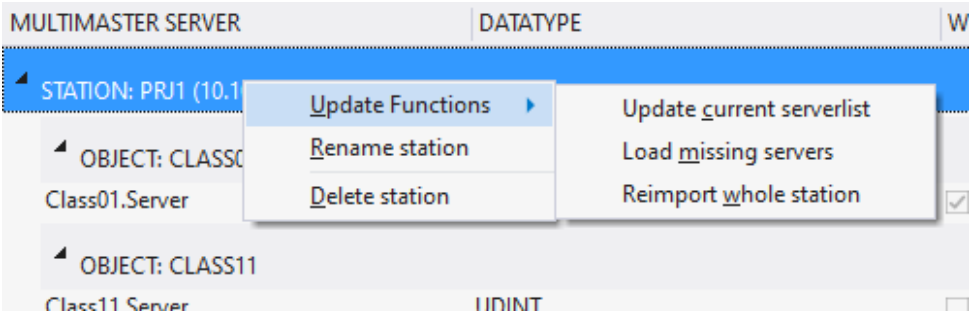


In the same way, **in Machine Manger the corresponding station that cannot be reached is outlined in red**. For more information about the Machine Manager, see the **Machine Manager documentation**.

So if Multimaster IO objects in the object network are displayed with a **red frame**, it means that the **connection between the stations is not in order**. **The cause must be found out and fixed** so that the connection between the stations works properly again.

Context menu of a station

In the Multimaster-Pane the following context menu is displayed after a right click:

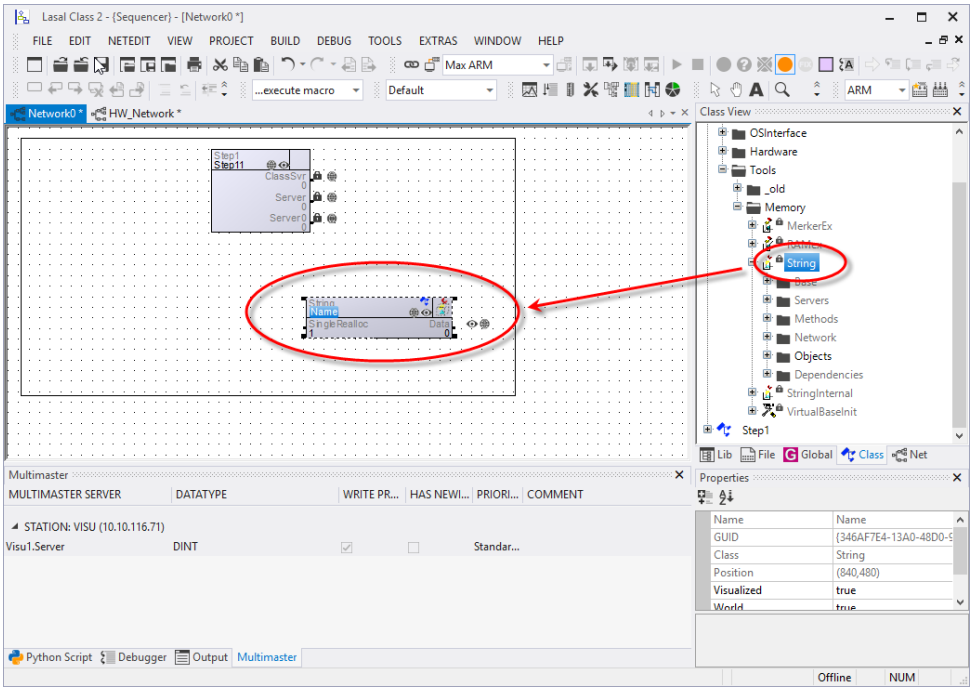


Update Functions	Update Functions contains three menu items for updating:
Update current serverlist	This is used to **update existing servers**.
Load missing servers	This will **only reload servers** from the station that are **not currently in the already imported station**. Already **deleted servers will be imported again**. Already imported **and changed servers will not be overwritten!**
Reimport whole station	This will import **the whole station** (all servers) completely new. **All changes** already made (deleted / changed servers) **are lost**.
Rename station	The station will be **renamed**.
Delete station	The station will be **deleted**.

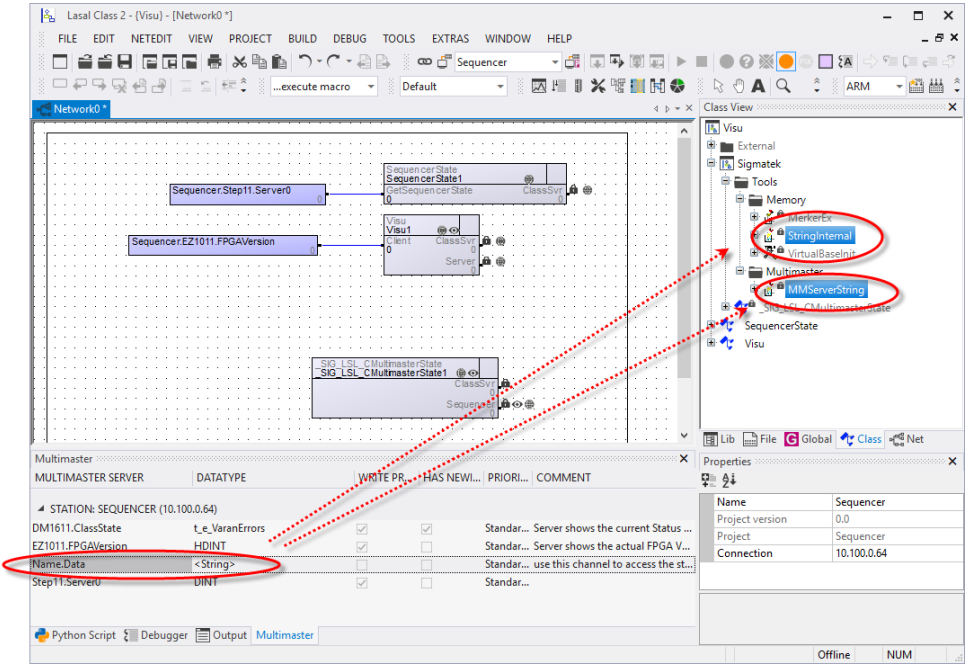
Multi-byte Transfer

Text with String

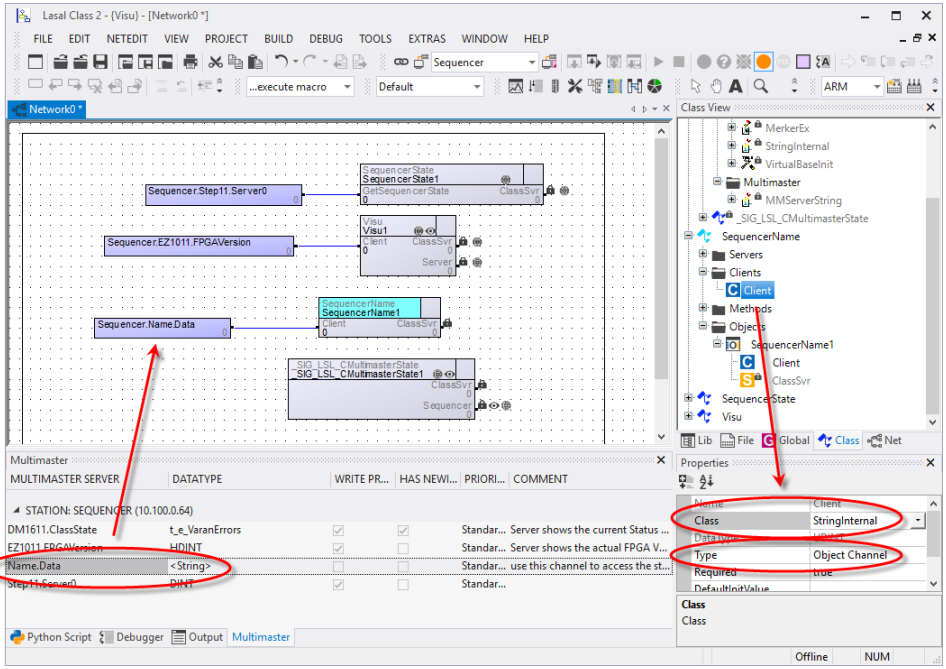
In the host project (“Sequencer”), a string object must first be created so that text can be transferred via the Multimaster variable system.

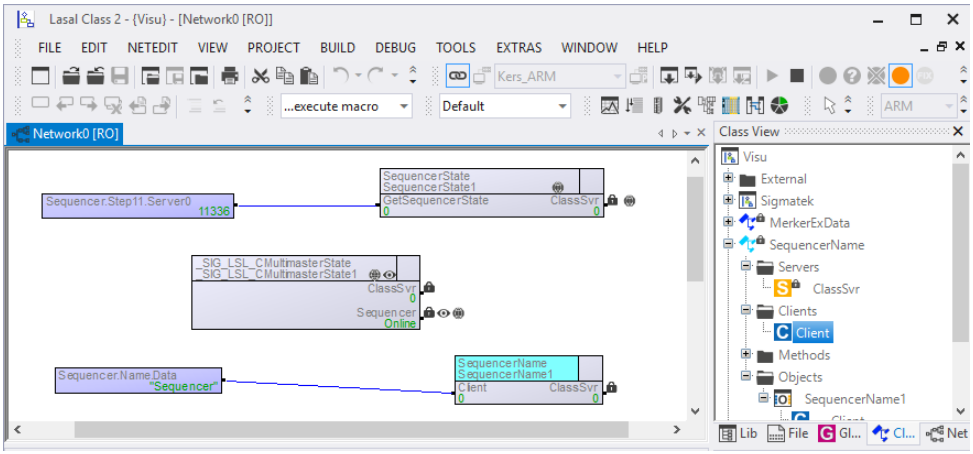


The server is added to the project in the MachineManager e.g. with Drag & Drop, to familiarize it with the consumer project (“Visualization”). If a server of the type string is added to the Multi-master list, the required classes are also copied to the project automatically.



In order to use the Multimaster server like an object channel to the “StringInternal” class, a client with an object channel to “StringInternal” is needed, which is connected with the Multimaster object.


Lasal Class 2 - {Visu} - [Network0 *]
FILE EDIT NETEDIT VIEW PROJECT BUILD DEBUG TOOLS EXTRAS WINDOW HELP
Network0 *
Sequencer.Step11.Server0
Sequencer.EZ1011.FPGAVersion
Sequencer.Name.Data
Class View
Multimaster
MULTIMASTER SERVER
DATATYPE
WRITE PR...
HAS NEWI...
PRIORI...
COMMENT
STATION: SEQUENCER (10.100.0.64)
DM1611.ClassState
t_e_VaranErrors
EZ1011.FPGAVersion
HDINT
Name.Data
<String>
Properties
Class
StringInternal
Type
Object Channel
Required
Python Script
Debugger
Output
Multimaster
Offline
NUM

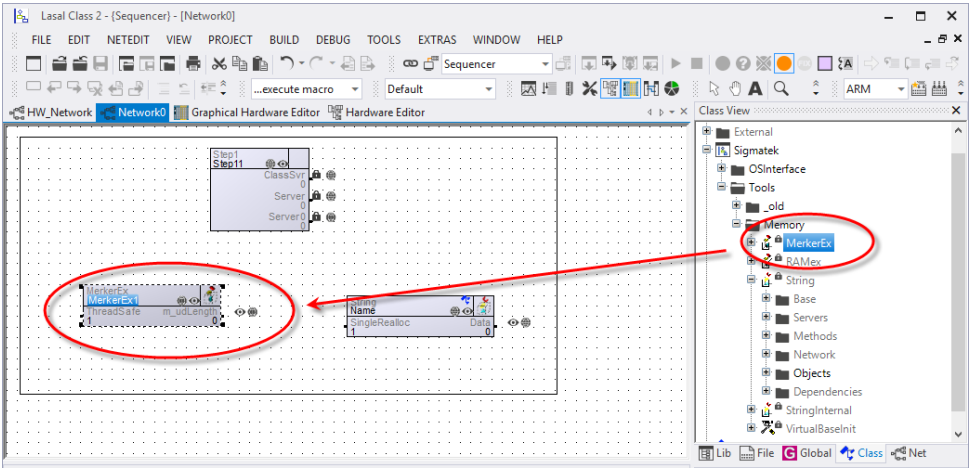

Lasal Class 2 - {Visu} - [Network0 [RO]]
FILE EDIT NETEDIT VIEW PROJECT BUILD DEBUG TOOLS EXTRAS WINDOW HELP
Network0 [RO]
Sequencer.Step11.Server0
11336
Online
"Sequencer"
Class View
Visu
External
Sigmatek
MerkerExData
SequencerName
Servers
ClassSvr
Clients
Client
Methods
Objects
SequencerName1

MerkerEx

Caution!

In order to transfer MerkerEx data consistently, a loader version > 02.02.170 is also required on the host station.

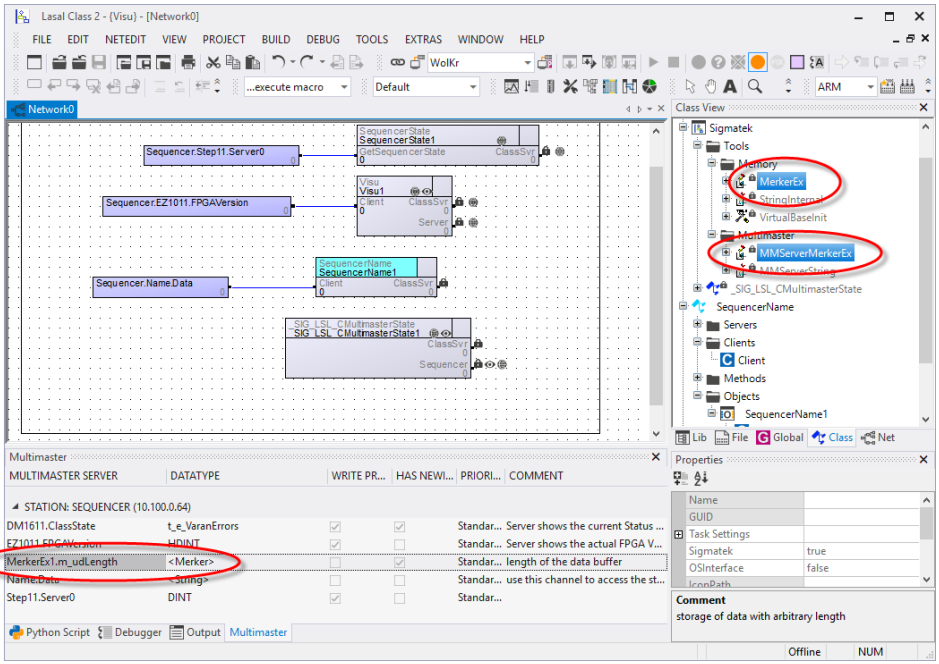
In the host project (“Sequencer”), a MerkerEx object must first be created, so that a data amount greater than 4 bytes can be transferred via the Multimaster variable system.



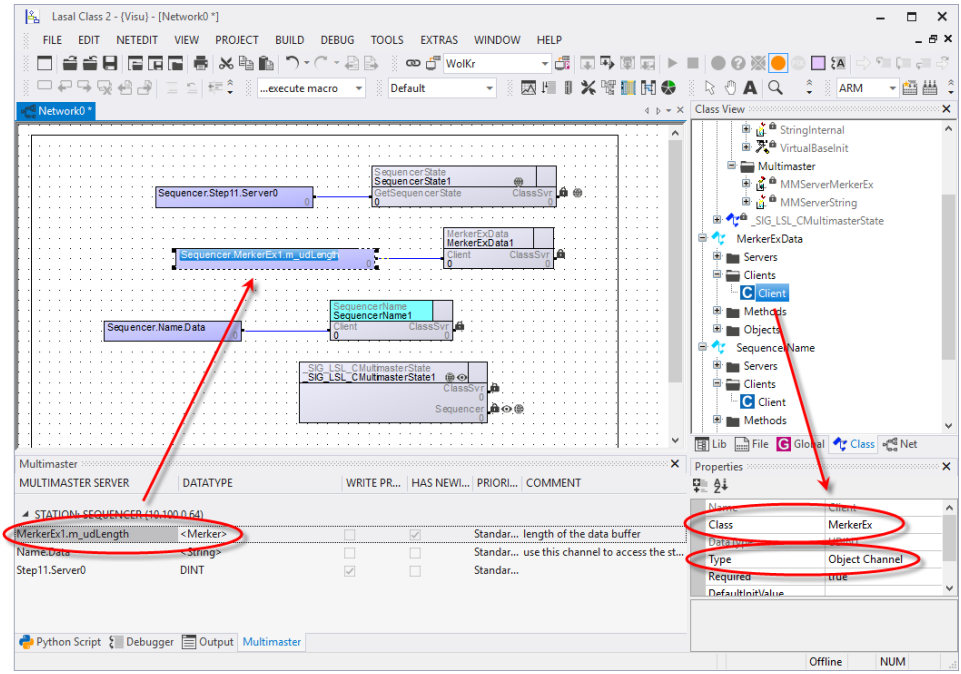
Caution!

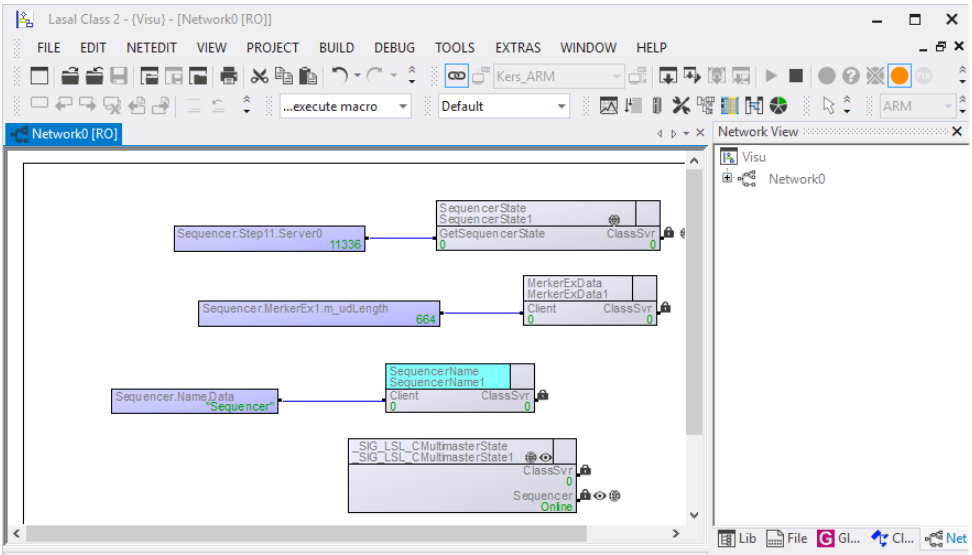
The "ThreadSafe" client of the MerkerEx class should be set to 1 so that access to the memory is protected by means of a lock.

The server is added to the project in the MachineManager, e.g. with Drag&Drop, to make it known to the consumer project (“Visu”). If a server of the type “MerkerEx” is added to the Multimaster list, the required classes are also copied to the project automatically.



In order to use the Multimaster server like an object channel (with limitations, see “Attention” below) to the “MerkerEx” class, a client with an object channel to “MerkerEx” is needed, which is connected with the Multimaster object.


Lasal Class 2 - [Visu] - [Network0 *]
FILE EDIT NETEDIT VIEW PROJECT BUILD DEBUG TOOLS EXTRAS WINDOW HELP
Network0 *
Sequencer.Step11.Server0
SequencerState
SequencerState1
GetSequencerState
ClassSvr
MerkerExData
MerkerExData1
Client
Sequencer.MerkerEx1.m_udLength
Sequencer.Name.Data
SequencerName
SequencerName1
SIG_LSL_CMultimasterState
SIG_LSL_CMultimasterState1
Sequencer
Class View
StringInternal
VirtualBaseInit
Multimaster
MMServerMerkerEx
MMServerString
_SIG_LSL_CMultimasterState
MerkerExData
Servers
Clients
Client
Methods
Objects
SequencerName
Servers
Clients
Client
Methods
Lib File Global Class Net
Multimaster
MULTIMASTER SERVER
DATATYPE
WRITE PR...
HAS NEWI...
PRIORI...
COMMENT
STATION: SEQUENCER (10.100.0.64)
MerkerEx1.m_udLength
<Merker>
Standar...
length of the data buffer
Name.Data
<String>
Standar...
use this channel to access the st...
Step11.Server0
DINT
Standar...
Properties
Class
MerkerEx
Type
Object Channel
Required
true
DefaultInitValue
Python Script
Debugger
Output
Multimaster
Offline
NUM

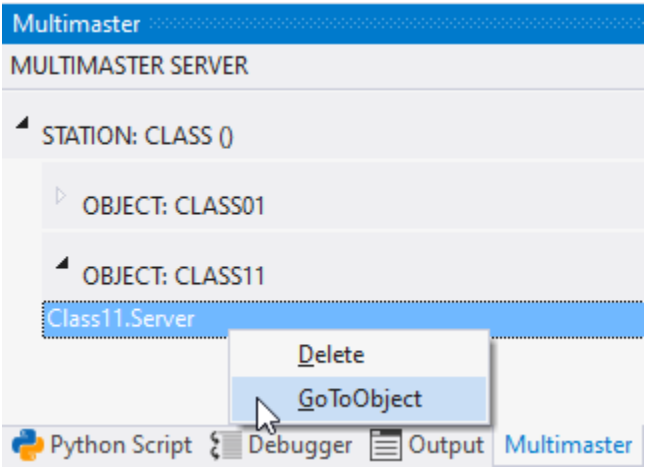


Caution!

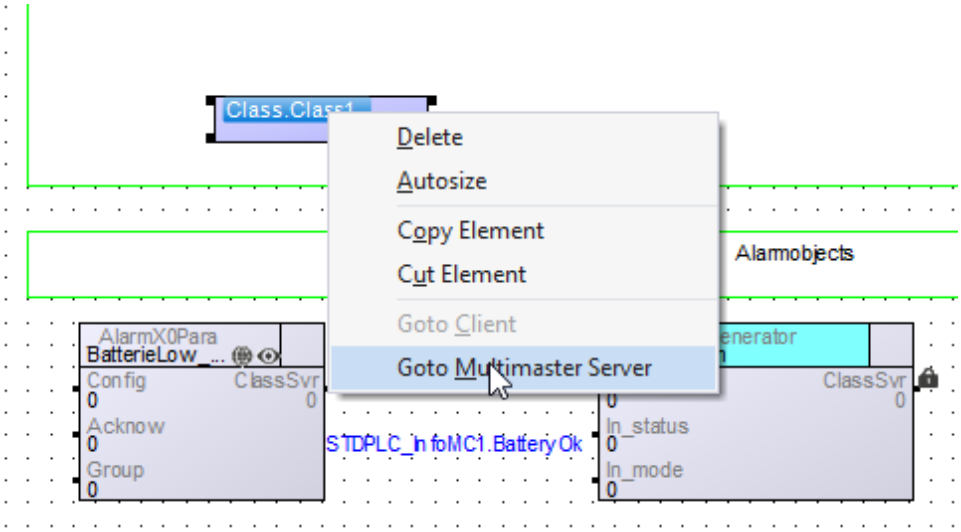
The Multimaster server of type “MerkerEx” only supports functions that do not access the data pointer:

GetData, GetDataAt, GetSize, SetDataAt

It should also be mentioned here that it is possible to jump directly to the corresponding object in the object network by right-clicking on a multimaster server and Go to Object:



You can also jump back to the server from the Multimaster Object. To do this, open the context menu in the object network and select the menu item Goto Multimaster Server:



For a call via Multimaster, the NewInst method on the host must return the status READY. To ensure that the command is recognized as successfully executed, an asynchronous implementation of the NewInst command in connection with cyclic calls of the GetState

method is not possible. The GetState method of the host cannot be called from the consumer side.

Calling the GetState method on the consumer side happens locally and serves for waiting for the answer of the NewInst command, which is executed on the host side. The GetState method on the consumer side returns BUSY, until the the NewInst command on the host side has been processed. For the other return values, see LDR_MM_GetState.

In order to ensure that the a NewInst command is correctly assigned via Multimaster and the corresponding answer received, a unique ID is generated from the NewInst command ID and the addresses of the input parameters. For this reason it is not possible to call several NewInst commands with identical ID and identical transfer parameters in parallel. In this case, after the first NewInst command is called, all further NewInst commands return the status ERROR, until the first command has been completely processed. For further return values see LDR_MM_NewInst.

In order to use the different functions of the _SIG_LSL_CMultimasterState class, the header file LoaderItf.h (located in the Include / Loader directory) must be included.

Functions, concerning a MultimasterStation

Functions, concerning a MultimasterServer

Callback Functions

See also:

Info

Task Configuration

Unit Configuration

OPC UA Configuration

Exception Handling

Reengineering...

Cleanup

Compare

EasyCase

Export Visualized Servers

Matlab Simulink Import

AutomationML

Validate GUID

Update Includes and Loader

Import Class / Network

Import Safety Parameter List

Multilinguage Comments

2.1.7.171 Functions, concerning a MultimasterStation

2.1.7.1711 LDR_MM_StatNrGetName

Determine station Id from the station name

Transfer parameters	Type	Description
udStationId		Id of the station
pszName		Buffer for the name filled as 0-terminated character string
Return parameters	**Type**	**Description**
status		0 Id could be resolved -1 Id has not been found

Example

```
FUNCTION GLOBAL LDR_MM_StatNrGetName
VAR_INPUT
  udStationId : UDINT;
  pszName : ^CHAR;
END_VAR
VAR_OUTPUT
  status : DINT;
END_VAR;
```

2.1.7.1712 LDR_MM_StatNrConnStop

Close connection to a station As the function is executed in a thread, the function can be executed delayed.

Transfer parameters	Type	Description
udStationId		Id of the station between 0 and 255

Return parameters	Type	Description
status		TRUE, if the message has been saved successfully, otherwise an error code is returned

Example

```
FUNCTION GLOBAL LDR_MM_StatNrConnStop
VAR_INPUT
    udStationId : UDINT;
END_VAR
VAR_OUTPUT
    status : UINT;
END_VAR;
```

2.1.7.1713 LDR_MM_StatNrConnStart

Establish connection to a station As the function is executed in a thread, the function can be executed delayed.

Transfer parameters	Type	Description
udStationId		Id of the station between 0 and 255
Return parameters	**Type**	**Description**
status		TRUE, if the message has been saved successfully, otherwise an error code is returned

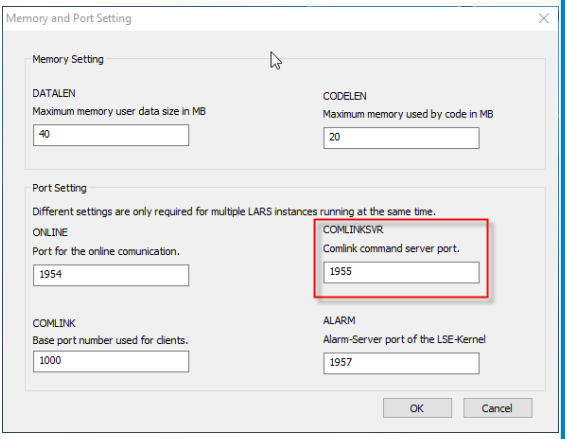
Example

```
FUNCTION GLOBAL LDR_MM_StatNrConnStart
VAR_INPUT
  udStationId : UDINT;
END_VAR
VAR_OUTPUT
  status : DINT;
END_VAR;
```

2.1.7.1714 LDR_MM_StatNrConnSetStr

Temporarily change connection string of a station When again loading the project, the connection string from the Stations.txt is read. Here, the current connection is closed and it is tried to establish connection to the new IP. As the function is executed in a thread, the function can be executed delayed.

Transfer parameters	Type	Description

udStationId		ID of the station between 0 and 255
pszConn		new connection string in the format of the Station.txt file (TCPIP:x.x.x.x). Optionally also the comlink command server port - as a default 1955 - can be given: (TCPIP:x.x.x.x:). This is the port, which can be set in the LARS config tool, e.g. to simulate several stations on a PC. 
Return parameters	**Type**	**Description**
status		TRUE, if the message has been saved successfully, otherwise an error code is returned

Example

```
FUNCTION GLOBAL LDR_MM_StatNrConnSetStr
VAR_INPUT
  udStationId : UDINT;
  pszConn : ^CHAR;
END_VAR
VAR_OUTPUT
  status : DINT;
END_VAR;
```

2.1.7.1715 LDR_MM_StatNrConnGetStr

Returns the connection string for the station with the given Id. Caution! Returns NIL, if the station has not been found!

Transfer parameters	Type	Description

udStationId		Id of the station between 0 and 255
pszConn		Buffer for the connection string
Return parameters	**Type**	**Description**
status		0 Id could not be resolved -1 Id has not been found

Example

```
FUNCTION GLOBAL LDR_MM_StatNrConnGetStr
VAR_INPUT
  udStationId : UDINT;
  pszConn : ^CHAR;
END_VAR
VAR_OUTPUT
  status : DINT;
END_VAR;
```

2.1.7.1716 LDR_MM_StatNrIsConnected

Checks, whether the station with the given Id is connected

Transfer parameters	**Type**	**Description**
udStationId		Id of the station between 0 and 255
Return parameters	**Type**	**Description**
status		-1 Id of the station invalid ≥0 Enum value of _MMState

Example

```
FUNCTION GLOBAL LDR_MM_StatNrIsConnected
VAR_INPUT
  udStationId : UDINT;
END_VAR
VAR_OUTPUT
  status : DINT;
END_VAR;
```

2.1.7.1717 LDR_MM_SetWaitForReqTimeout

Sets the timeout value for a station to wait for the required clients. If this function is not called, the station uses the default value 0, i.e. it does not wait. A value greater than 0 means, that the station waits, until from all clients of this station with set required flag a value has been received or a timeout occurred. Waiting happens in the last Init method (_FirstScan TRUE), so calling this function is only effective, if it happens before the last Init method. This function is available since loader version 02.02.189.

Transfer parameters	Type	Description
udStationId		ID of the station between 0 and 255
waitForReqTimeout_ms		Timeout value in milliseconds
Return parameters	**Type**	**Description**
status		0 ID could be resolved -1 ID could not be found

Example

```
FUNCTION GLOBAL LDR_MM_SetWaitForReqTimeout
VAR_INPUT
  udStationId : UDINT;
  waitForReqTimeout_ms : UDINT;
END_VAR
VAR_OUTPUT
  status    : DINT;
END_VAR;
```

2.1.7.1718 LDR_MM_StatNameGetNr

Determine station Id from the station name

Transfer parameters	Type	Description
pszName		0-terminated character string
Return parameters	**Type**	**Description**
udStationId		Id of the station The return value is only valid, if the value is between 0 and 255, otherwise the station has not been found!

Example

```
FUNCTION GLOBAL LDR_MM_StatNameGetNr
```

```
VAR_INPUT
  pszName : ^CHAR;
END_VAR
VAR_OUTPUT
  udStationId : UDINT;
END_VAR;
```

2.1.7.172 Functions, concerning a MultimasterServer

2.1.7.1721 LDR_MM_SvrIsConnected

Checks, whether the server of the station with the given Id is connected

Transfer parameters	Type	Description
udStationId		Id of the station between 0 and 255
pszSvrName		Name of the server, 0-terminated character string
Return parameters	**Type**	**Description**
status		0 Server is valid and has been read resp. initialized once -1 Server is valid, but has not been read resp. initialized -2 Server invalid

Example

```
FUNCTION GLOBAL LDR_MM_SvrIsConnected
VAR_INPUT
  udStationId : UDINT;
  pszSvrName : ^CHAR;
END_VAR
VAR_OUTPUT
  status : DINT;
END_VAR;
```

2.1.7.1722 LDR_MM_SvrGetHandle

Returns the handle to the server

Transfer parameters	Type	Description
udStationId		Id of the station between 0 and 255
pszSvrName		Name of the server, 0-terminated character string
Return parameters	**Type**	**Description**
udSvrHandle		0 Server handle is invalid, server has not been found

		>0	Valid server handle

Example

```
FUNCTION GLOBAL LDR_MM_SvrGetHandle
VAR_INPUT
  udStationId : UDINT;
  pszSvrName  : ^CHAR;
END_VAR
VAR_OUTPUT
  udSvrHandle : UDINT;
END_VAR;
```

2.1.7.1723 LDR_MM_SvrGetHandleFromCmpName

Returns the handle to the server

Transfer parameters	Type	Description
pszCmpName		Complete name of the server object on the network, 0-terminated character string
Return parameters	**Type**	**Description**
udSvrHandle		0 Server handle is invalid, server has not been found >0 Valid server handle

Example

```
FUNCTION GLOBAL LDR_MM_SvrGetHandleFromCmpName
VAR_INPUT
  pszCmpName  : ^CHAR;
END_VAR
VAR_OUTPUT
  udSvrHandle : UDINT;
END_VAR;
```

2.1.7.1724 LDR_MM_SvrGetNameFromHandle

Determines the station Id (if desired) and the server name from the server handle

Transfer parameters	Type	Description
udSvrHandle		Handle to the server
udStationId		Optional, buffer for the Id of the station
pszSvrName		Buffer for the server name, 0-terminated character string

Return parameters	Type	Description	
bFound		TRUE	Server handle is invalid, buffer has been filled
		FALSE	Server handle invalid, or could not be found

Example

```
FUNCTION GLOBAL LDR_MM_SvrGetNameFromHandle
VAR_INPUT
  udSvrHandle : UDINT;
  udStationId : ^UDINT;
  pszSvrName  : ^CHAR;
END_VAR
VAR_OUTPUT
  bFound : BOOL;
END_VAR;
```

2.1.7.1725 LDR_MM_ReadSvr

Reads the data from MultiMaster-Data-In, for n-byte or string servers the server handle is returned

Transfer parameters	Type	Description	
udStationId		Id of the station between 0 and 255	
pszSvrName		Name of the server, 0-terminated character string	
pBuffer		4-byte buffer for the data	
Return parameters	**Type**	**Description**	
status		-1	Server could not be read
		>0	Time in ms since the last change of the server value

Example

```
FUNCTION GLOBAL LDR_MM_ReadSvr
VAR_INPUT
  udStationId : UDINT;
  pszSvrName : ^CHAR;
  pBuffer : ^UDINT;
END_VAR
VAR_OUTPUT
  status : DINT;
END_VAR;
```

2.1.7.1726 LDR_MM_ReadSvrByHandle

Reads the data from MultiMaster-Data-In, with n-byte or string servers the server handle is returned

Transfer parameters	Type	Description
udSvrHandle		Server handle
pBuffer		4 bytes buffer for the data
Return parameters	**Type**	**Description**
status		-1 server could not be read >0 time in ms since the last change of the server value

Example

```
FUNCTION GLOBAL LDR_MM_ReadSvrByHandle
VAR_INPUT
  udSvrHandle : UDINT;
  pBuffer     : ^UDINT;
END_VAR
VAR_OUTPUT
  status    : DINT;
END_VAR;
```

2.1.7.1727 LDR_MM_SvrNewInst

NewInst SenderRequest is called As the sender request is executed in a thread, the function can be executed delayed.

Transfer parameters	Type	Description
udSvrHandle		Server handle
pPara		Cmd structure for NewInst
pResult		Result of the NewInst
Return parameters	**Type**	**Description**
ret_code		BUSY if NewInst Request has been called ERROR_BUSY NewInst could not be executed ERROR NewInst was not executed because no NewInst server no connection no memory could be allocated for internal parameters

		NewInst with same transfer parameters (command ID an memory addresses of the parametes) in work

Example

```
FUNCTION GLOBAL LDR_MM_SvrNewInst
VAR_INPUT
  udSvrHandle : UDINT;
  pPara : ^CmdStruct;
  pResult : ^Results;
END_VAR
VAR_OUTPUT
  ret_code : IprStates;
END_VAR;
```

2.1.7.1728 LDR_MM_SvrGetState

As NewInst is set as request, the status must be polled with LDR_MM_SvrGetState.

Transfer parameters	Type	Description
udSvrHandle		Server handle
pPara		Cmd structure for NewInst
pResult		Result of the NewInst
Return parameters	**Type**	**Description**
ret_code		BUSY NewInst Request in work READY NewInst successfully executed with return value READY, pResult was filled ERROR_BUSY NewInst was executed, but return parameter not READY ERROR internal parameters not found and GetState could not be determined

Example

```
FUNCTION GLOBAL LDR_MM_SvrGetState
VAR_INPUT
  udSvrHandle : UDINT;
  pPara          : ^CmdStruct;
  pResult                : ^Results;
END_VAR
VAR_OUTPUT
  ret_code       : IprStates;
```

```
END_VAR;
```

2.1.7.173 Callback Functions

2.1.7.1731 LDR_MM_UserCallbackPrototyp

Prototype of the Callback function Is called for every status change regarding Online of the station.

Transfer parameters	Type	Description
pThis		This pointer of the Callback
udStationId		Id of the station
szStationName		Name of the station
udServerHandle		Server handle, can be 0, invalid
szServerName		Server name, can be NIL
udState		Status information _MMState

Example

```
FUNCTION GLOBAL LDR_MM_UserCallbackPrototyp
VAR_INPUT
  pThis : ^void;
  udStationId : UDINT;
  szStationName : ^CHAR;
  udServerHandle : UDINT;
  szServerName : ^CHAR;
  udState : _MMState;
END_VAR;
```

2.1.7.1732 LDR_MM_InstallCallback

“Install” Callback function The Callback with the current status is immediately called once.

Transfer parameters	Type	Description
pCallback		Address of the Callback function, of the type LDR_MM_UserCallbackPrototyp
pThis		This pointer provided in the Callback

Example

```
FUNCTION GLOBAL LDR_MM_InstallCallback
```

```
VAR_INPUT
  pCallback : ^void;
  pThis : ^void;
END_VAR;
```

2.1.7.1733_MMState

Offline	Station cannot be reached or is in "Run RAM\Rom" mode.
Online_Initializing	Station can be reached, but the server is not yet initialized.
Online	Station can be reached and all server of the station are available and initialized.
Online_NotAllServerAvailable	Station can be reached, but not all servers are available. The ones that have been located are initialized.
SvrNotFoundAtStation	Server has not been found on the station and thus has no valid LasalId.

For more information see the help document of the **Machine Manager**.

2.1.7.18 Multilanguage Comments

This menu item can be used to add or manage multilingual comments to the project. Some general remarks about this:

1. **Only comments in the user defined area** are edited! So own classes, objects, networks... **No SIGMATEK** classes / libraries etc.!
2. It is possible **to export** multilingual comments - e.g. for translation purposes - as well as **to import** already exported ones. The format used for this is ***.csv** (with ';' as separator).
3. **All comments** of certain languages **can be deleted** (removed) at once
4. **All IDs** (GUIDs and Language IDs) **can be removed** from all comments **in one step**. (Caution! In case of multilingual comments, the language affiliation can no longer be distinguished afterwards).
5. There is a possibility **to hide** comments in selected languages



Caution!

All points except the last **cannot be undone**!



Caution!

Header files of classes are always handled by the multilanguage feature, **Global header files are not handled by default**. However, it is possible to write the following tag in the file to allow the addition of multilanguage comments: "//#LSL_MULTILANG".

The following areas of LASAL CLASS 2 **are excluded** from comment processing:

a. SFC
b. DataAnalyzer
c. KOP
d. C-Code (*.h resp. *.c files). ***.h files with ST code are supported**!
e. Units
f. DriveIntegration
g. Revision Documentation

Add Multilanguage Identifiers To Current Project

Import / Reimport Multilanguage Comments

Export Multilanguage Comments

Remove Multilanguage Identifiers (GUID + Language ID)

Remove Multilanguage Comments

Set Active Languages

See also:

Info

Task Configuration

Unit Configuration

OPC UA Configuration

Exception Handling

Reengineering...

Cleanup

Compare

EasyCase

Export Visualized Servers

Matlab Simulink Import

AutomationML

Validate GUID

Update Includes and Loader

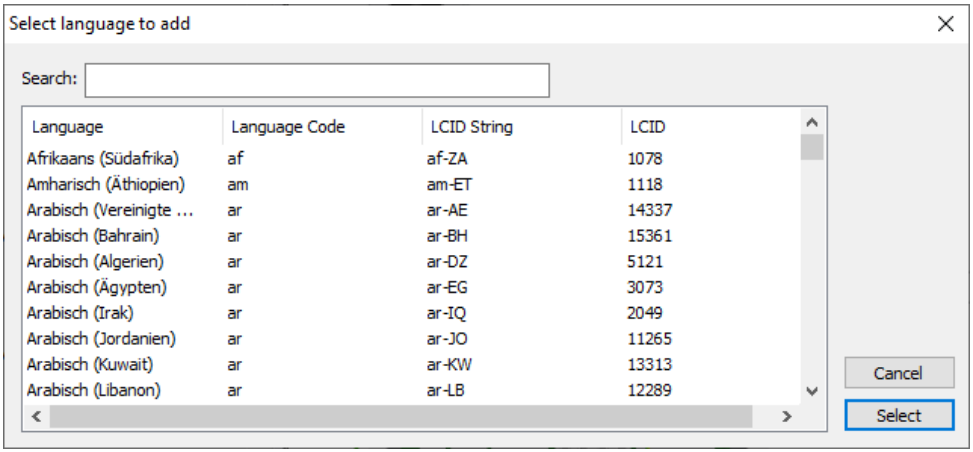
Import Class / Network

Import Safety Parameter List

Multimaster

2.1.7.181 Add Multilanguage Identifiers To Current Project

This menu item allows you to add multilingual comments to the project. If it is selected, the following dialog will be displayed:



Here the language can be selected, which adds a GUID as well as the selected Language-ID to all comments in the source code, object network as well as in the properties. This **only affects user-defined** classes / objects / networks, **not SIGMATEK's own**! So if there are single-line (starting with //), or multi-line (starting with (*) source code comments in the editor, or comments in object networks or in the "Comment" properties of servers, clients etc. in the properties window; these are provided with the corresponding extension.

For example, if the language German [Austria] is selected, it has the language code 3079. As an example, let's take a one-line comment in its own class, meanwhile still without IDs:

```
FUNCTION VIRTUAL GLOBAL Class0::CyWork
  VAR_INPUT
    EAX : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX) : UDINT;
  END_VAR

  // insert your own code here

  state := READY;

END_FUNCTION
```

This comment gets the following IDs after selecting the language "German [Austria]":

```
FUNCTION VIRTUAL GLOBAL Class0::CyWork
  VAR_INPUT
    EAX : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX) : UDINT;
  END_VAR

  // |#{303AABDC-FE2A-439A-9971-E7392FE22C5B}|~3079|insert your own code here

  state := READY;

END_FUNCTION
```

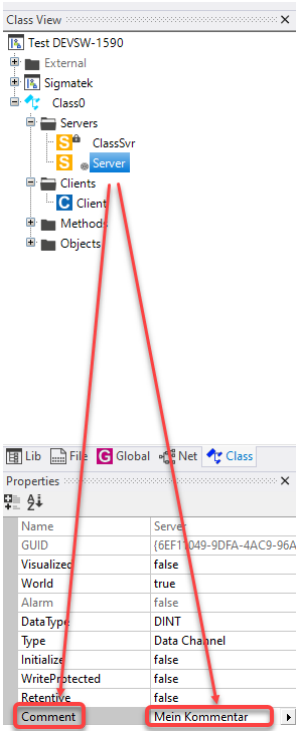
As you can see on the picture, the IDs are inserted **between the comment characters** (here: "//") **and the comment itself** ("Insert your own code here"). This also applies to **multi-line comments**! A comment identification number (GUID) and the country code (LCID) - here 3079 for German [Austria] - are inserted..



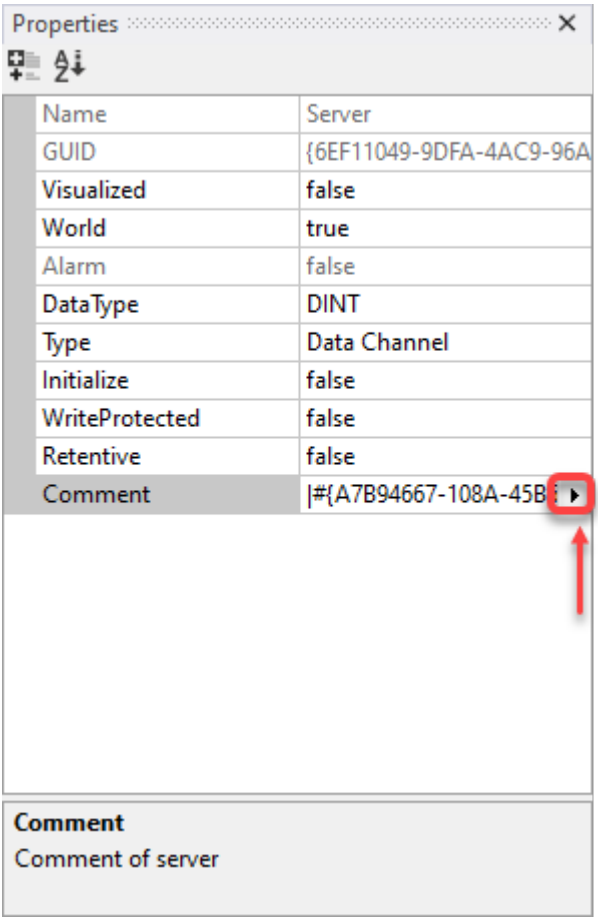
Hint

The colors for GUIDs and Language IDs can be set as desired in the menu item Extras / Options / Code Editor / ST Options. See Extras Menu / Options... / Code Editor

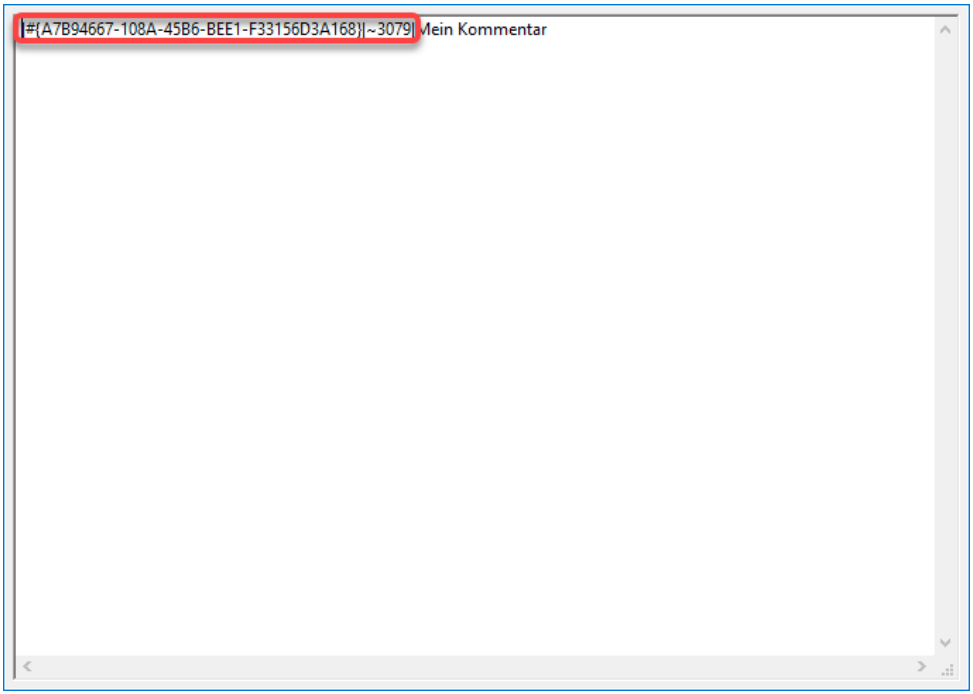
In the properties it looks similar. A property comment without IDs:



The same comment after selecting the language in the "Select language to add" dialog:



Since only the beginning of the comment is visible here, click the More button (outlined in red at the top of the image and marked with the arrow) to display the entire comment text:



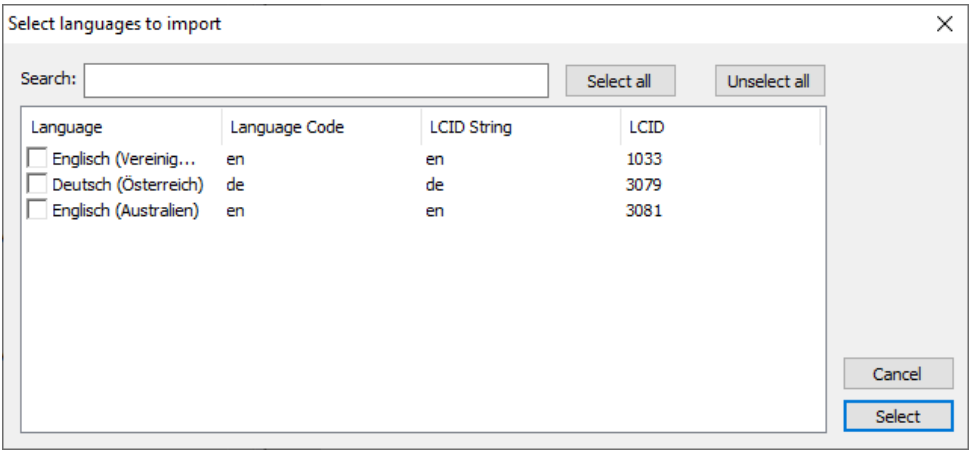
Note:

All comments present at this time, which are without GUID and Language-ID, **will be assigned to the selected language**!

[Back to Multilinguale Comments]
[Import / Reimport Multilanguage Comments]
[Export Multilanguage Comments]
[Remove Multilanguage Identifiers (GUID + Language ID)]
[Remove Multilanguage Comments]
[Set Active Languages]

2.1.7.182 Import / Reimport Multilanguage Comments

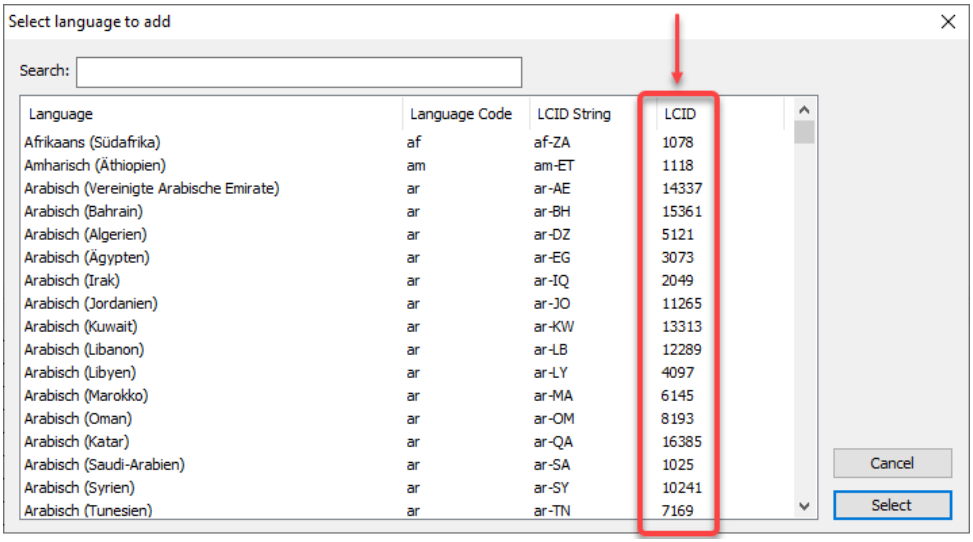
With this menu item, the comments already exported in *.csv format can be imported according to language. After selecting the import file from the file selection dialog, the "Select languages to import" dialog is displayed:



Here you can select the language(s) to be imported. In the dialog exactly the languages, which are entered in the *.csv file, are indicated. Now the desired languages can be selected - **only the comments in these languages** will be imported. This gives the possibility to export several languages at once and to import e.g. only one language. If you want **to import all exported languages**, you can do this with the button "**Select all**".

Structure of the *.csv file

The first row contains the column headers: GUID of the comments and then the individual LCIDs of the available languages. All columns are separated by semicolons. If you want to add a translation to the *.csv document, e.g. into another language that does not yet exist in the project, you can do this by creating a new column with the Language-ID (LCID) in the first line. The LCIDs must be manually selected and entered into the *.csv file. A list of available LCIDs can be found directly in LASAL in the Select language to add dialog (last column),...



...or under LCIDs on the Microsoft dokumentation site.

Back to Multilinguale Comments

Add Multilanguage Identifiers To Current Project

Export Multilanguage Comments

Remove Multilanguage Identifiers (GUID + Language ID)

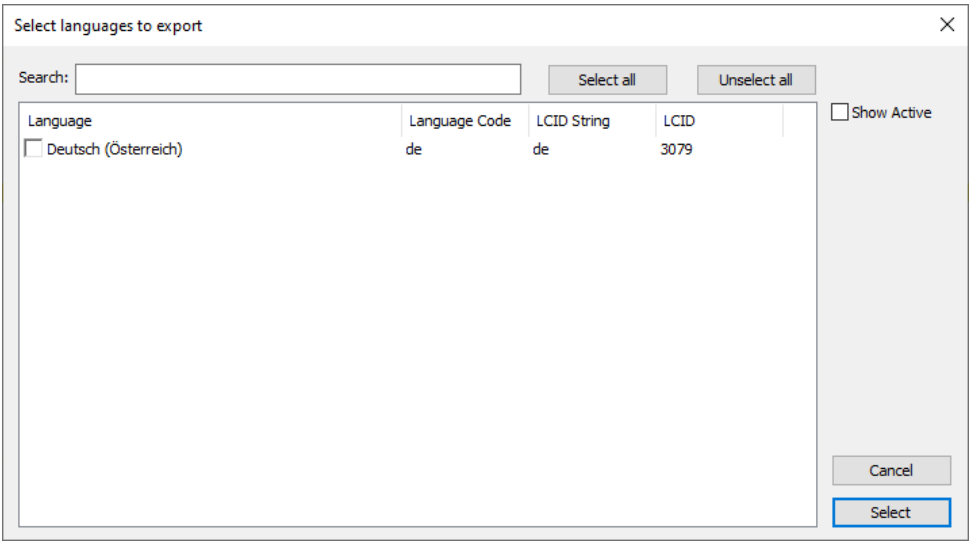
Remove Multilanguage Comments

Set Active Languages

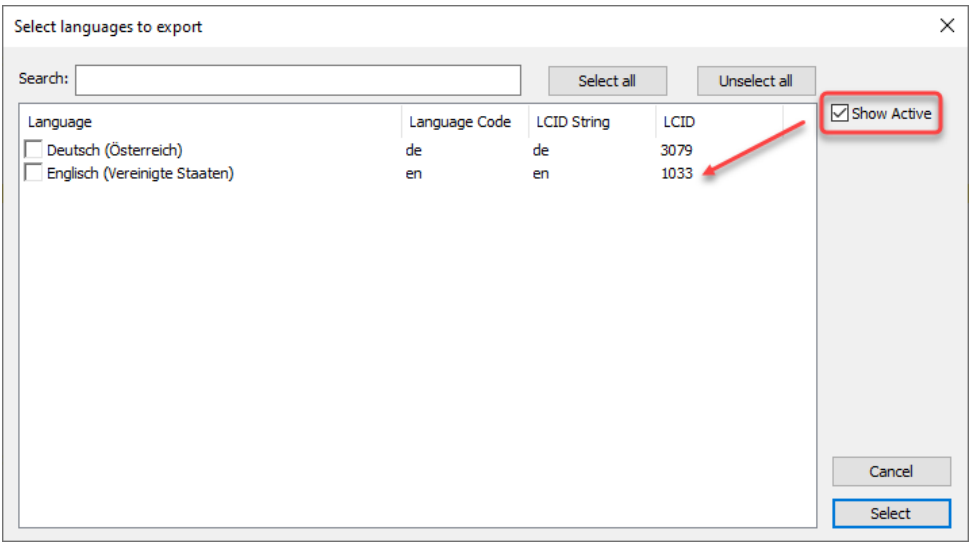
2.1.7.183 Export Multilanguage Comments

Using this menu item, **multilingual comments can be exported in *.csv format (e.g. for Excel)**. Here it is possible to select which language(s) are to be exported if several languages already exist.

When the menu item is selected, the following dialog is displayed:

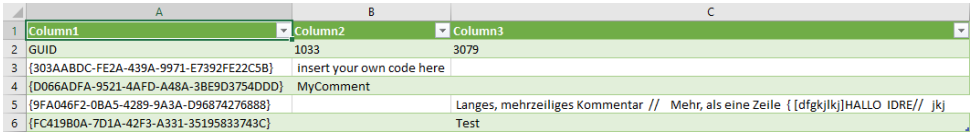


In this example, comments with German Language IDs already exist, and the English Language ID **without comments** exists. Both languages are set to "**Active**". If you now want to see **all active languages**, you have to click the "**Show Active**" checkbox:



In this dialog **the languages can be selected which should be exported**. If you want **to export all languages**, you can do this with the button "**Select all**".

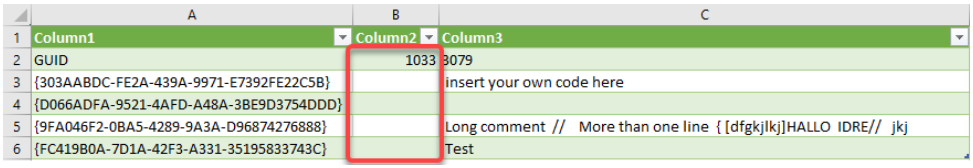
If the languages are exported and then the *.csv list is opened in Excel, for example, it looks like this:



	A	B	C
1	Column1	Column2	Column3
2	GUID	1033	3079
3	{303AABDC-FE2A-439A-9971-E7392FE22C5B}	insert your own code here	
4	{D066ADFA-9521-4AFD-A48A-3BE9D3754DDD}	MyComment	
5	{9FA046F2-0BA5-4289-9A3A-D96874276888}		Langes, mehrzeiliges Kommentar // Mehr, als eine Zeile { [dfgkjlkj]HALLO IDRE// jkj
6	{FC419B0A-7D1A-42F3-A331-35195833743C}		Test

As can be seen, the first line shows "GUID" and all language codes. In the further lines the individual comments are listed: in each case the GUID and - in the column corresponding to the language - the comment text.

If there is a language code, but no comment text in this language, a column is created anyway and the LCID entered in the top line:



	A	B	C
1	Column1	Column2	Column3
2	GUID	1033	3079
3	{303AABDC-FE2A-439A-9971-E7392FE22C5B}		insert your own code here
4	{D066ADFA-9521-4AFD-A48A-3BE9D3754DDD}		
5	{9FA046F2-0BA5-4289-9A3A-D96874276888}		Long comment // More than one line { [dfgkjlkj]HALLO IDRE// jkj
6	{FC419B0A-7D1A-42F3-A331-35195833743C}		Test

Back to Multilinguale Comments

Add Multilanguage Identifiers To Current Project

Import / Reimport Multilanguage Comments

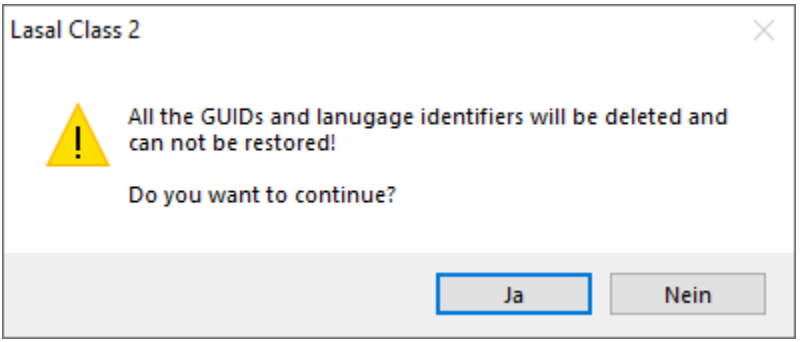
Remove Multilanguage Identifiers (GUID + Language ID)

Remove Multilanguage Comments

Set Active Languages

2.1.7.184 Remove Multilanguage Identifiers (GUID + Language ID)

This menu item **removes all GUIDs and Language IDs from all comments** as the project. **Caution**! In the case of comments that may be multilingual, **the language affiliation can no longer be distinguished afterwards**. This process **cannot be undone**! This is also pointed out in a **warning**:



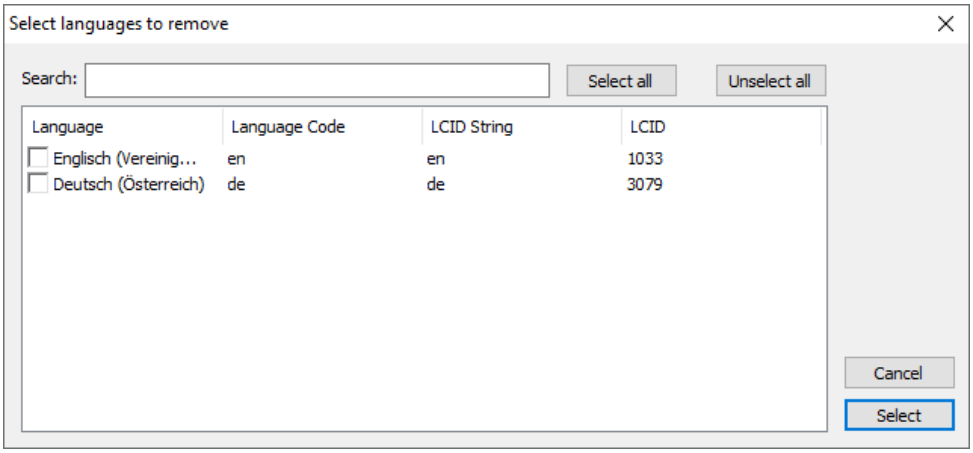
After confirmation all GUIDs and Language IDs are deleted. This process virtually restores the state **without** Multilingual Comments.

Back to Multilinguale Comments
Add Multilanguage Identifiers To Current Project
Import / Reimport Multilanguage Comments
Export Multilanguage Comments
Remove Multilanguage Comments
Set Active Languages

2.1.7.185 Remove Multilanguage Comments

This menu item allows to **completely remove comments in individual languages**.

If you select this menu item, the following dialog is displayed:

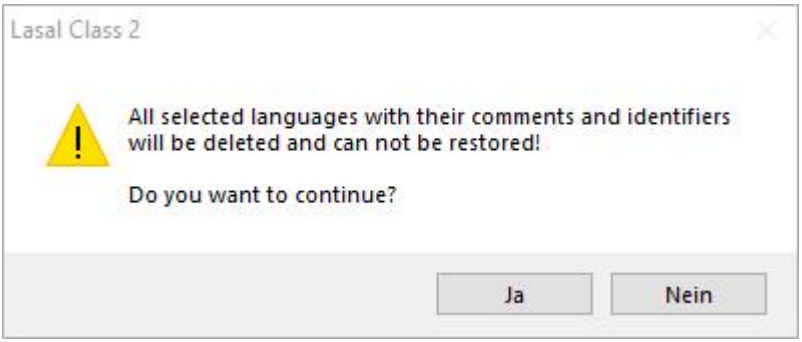


Here you can select the languages for which **all associated comments are to be removed**. Here not only GUID, or Language-ID, but **definitely the whole comment is removed**. If you want **to remove all languages**, you can do this with the button "**Select all**".

Attention!

This process can no longer be reversed!

This is also pointed out in a **warning**:



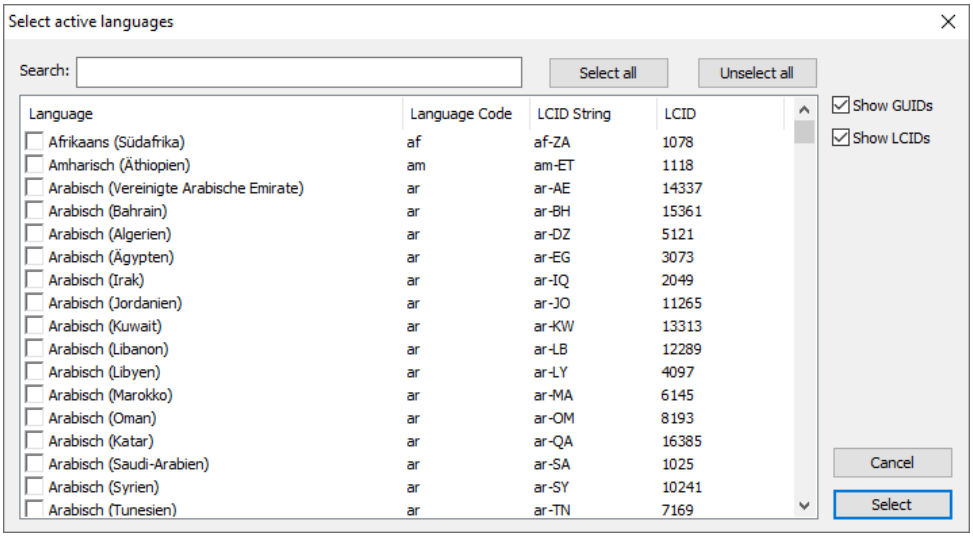
If this is confirmed with **Yes**, **all comments in the selected languages will be deleted**.

Back to Multilinguale Comments
Add Multilanguage Identifiers To Current Project
Import / Reimport Multilanguage Comments
Export Multilanguage Comments
Remove Multilanguage Identifiers (GUID + Language ID)
Set Active Languages

2.1.7.186 Set Active Languages

With this menu item comments in single (or multiple) languages **can be displayed or hidden**. This is very useful, for example, if all comments are available in several languages and should be displayed in only one language, depending on the user / preference.

If the menu item is selected, the following dialog is displayed:



All **languages** in which comments are to be displayed **can be selected** here. If a language is **not selected**, all associated comments **are hidden**.



Hint

If **no** language is selected, **all** languages are displayed!

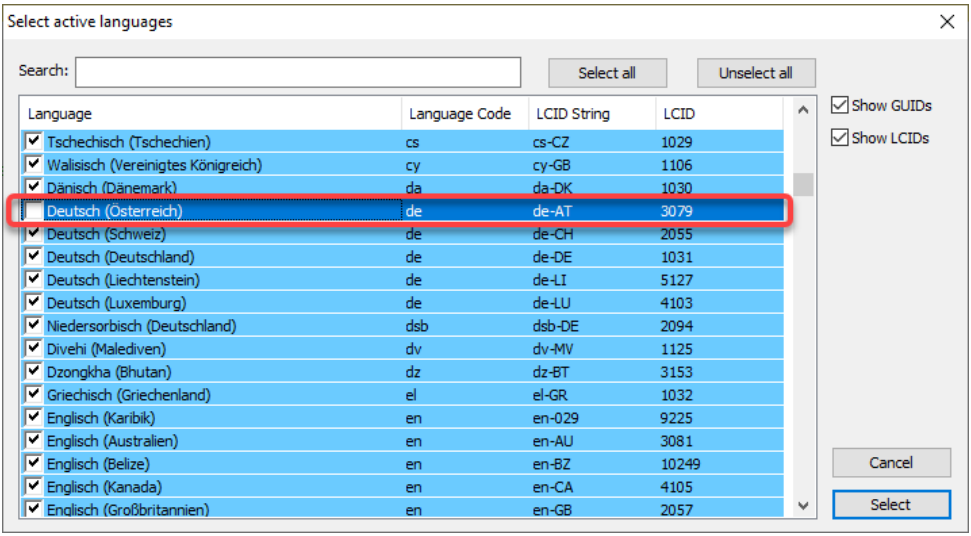
This process is reversible, i.e. hidden comments **can be displayed again at any time** by selecting the appropriate language in this dialog! This is a simple display function. The two checkboxes "**Show GUIDs**" and "**Show LCIDs**" can be used to select whether nothing (no checkbox selected), only the GUIDs or the Language IDs (corresponding checkbox selected), or everything (both checkboxes selected) should be displayed.

Hidden comments

Is the following constellation present:

```
//|#{FC419B0A-7D1A-42F3-A331-35195833743C}|~3079| Test
if a > 1000 then
   a := 0;
end_if;
```

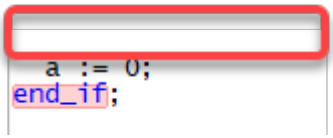
...and in the Set Active Languages dialog the language is hidden...



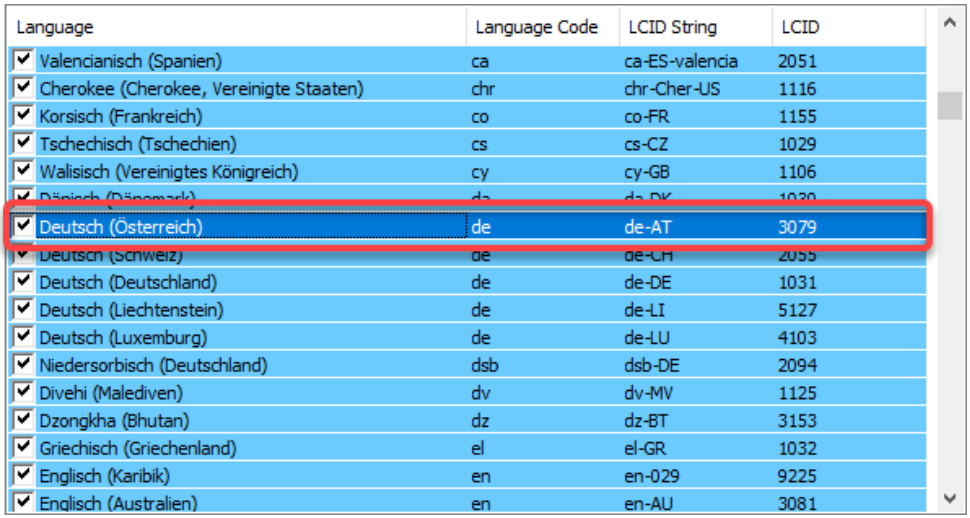
...the comment line disappears:

```
if a > 1000 then
  a := 0;
end_if;
```

If the cursor is placed directly in front of the IF command and the backspace key is pressed - thus deleting the "newLine" character in front of the IF - the whole line of code disappears:



The reason for this is that it is added to the end of the comment and thus becomes invisible. If you show the language in Set Active Languages again...



Language	Language Code	LCID String	LCID
Valencianisch (Spanien)	ca	ca-ES-valencia	2051
Cherokee (Cherokee, Vereinigte Staaten)	chr	chr-Cher-US	1116
Korsisch (Frankreich)	co	co-FR	1155
Tschechisch (Tschechien)	cs	cs-CZ	1029
Walisisch (Vereinigtes Königreich)	cy	cy-GB	1106
Dänisch (Dänemark)	da	da-DK	1030
Deutsch (Österreich)	de	de-AT	3079
Deutsch (Schweiz)	de	de-CH	2055
Deutsch (Deutschland)	de	de-DE	1031
Deutsch (Liechtenstein)	de	de-LI	5127
Deutsch (Luxemburg)	de	de-LU	4103
Niedersorbisch (Deutschland)	dsb	dsb-DE	2094
Divehi (Malediven)	dv	dv-MV	1125
Dzongkha (Bhutan)	dz	dz-BT	3153
Griechisch (Griechenland)	el	el-GR	1032
Englisch (Karibik)	en	en-029	9225
Englisch (Australien)	en	en-AU	3081

...you can see what happened in the background:

```
//|#{FC419B0A-7D1A-42F3-A331-35195833743C}|~3079| Test if a > 1000 then
  a := 0;
end_if;
```

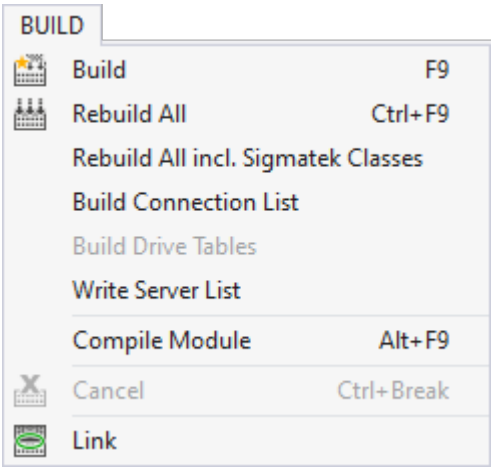


Attention.

If there is something hidden / invisible in the comment (Show LCIDs, or Show GUIDs deselected; language hidden etc.), a comment marked in this way can **only be deleted with the Delete key** and **not with the Backspace key**!

[Back to Multilinguale Comments]
[Add Multilanguage Identifiers To Current Project]
[Import / Reimport Multilanguage Comments]
[Export Multilanguage Comments]
[Remove Multilanguage Identifiers (GUID + Language ID)]
[Remove Multilanguage Comments]

2.1.8 Build Menu



Build	Compiles **the changes of the project**. If there is an online connection, the download window will open afterwards.
Rebuild All	Compiles **the whole project**, but - apart from some exceptions, see text below the table - **without the Sigmatek classes**. If an online connection is established, the download window will be opened afterwards.
Rebuild All incl. Sigmatek Classes	Compiles the entire project **with the Sigmatek classes**, with certain exceptions. For more information see the text below the table.
Build connection list	Compiles the **networks** (ONs) of the project.
Build Drive Tables	Compiles the **parameter files for the drives** created with the DiasDrive environment.
Write Server List	Rewrites the **server export** list (maeexp.txt).
Compile Module	Compiles **only this one module** and no modules that depend on it.
Cancel	**Cancels** a running compilation process.
Link	**Checks** if all **external references** in the source files can be resolved. The link process starts automatically after each error-free compilation process.

It needs a detailed explanation of the differences between **Build**, **Build All** and **Build All incl. Sigmatek Classes**. The compilation speed depends significantly on exactly **how many files** are compiled, therefore there are three different types of building. When compiling, LASAL CLASS 2 checks each file individually. Below you can read the rules for compiling one or more files, depending on the selected compilation type.

Build

This menu item compiles, **as far as possible, only the changed classes**. The rules for this are as follows:

User classes:

1. All source files which do not have resulting object files - **i.e. have never been compiled** - are compiled.
2. **Changed (edited) files**: the timestamp (time of last modification) of the object file is compared with the timestamp of the source file. If it is older, the file is recompiled.
3. **Changed (edited) global files** (all dependencies contained in the file "*global.h"*): In case the timestamp of the source file is older than the file "*global.h"*, a file translation also will be performed.

SIGMATEK-Classes:

1. All source files which do not have resulting object files - **i.e. have never been compiled** - are compiled.
2. **Changed (edited) files**: the timestamp (time of last modification) of the object file is compared with the timestamp of the source file. If it is older, the file is recompiled.
3. **Changed (edited) units** (*Units.h*): In case the timestamp of the source file is older than that of the "*Units.h*", a file translation also will be performed.

Rebuild All

This menu item compiles everything **except SIGMATEK classes**, if possible. The rules for this are as follows:

User classes:

1. All source files which do not have resulting object files - **i.e. have never been compiled** - are compiled.
2. **User classes are always compiled.**

SIGMATEK-Classes:

1. All source files which do not have resulting object files - **i.e. have never been compiled** - are compiled.
2. **Changed (edited) files**: the timestamp (time of last modification) of the object file is compared with the timestamp of the source file. If it is older, the file is recompiled.

3. **Changed (edited) units** ("*Units.h*"): In case the timestamp of the source file is older than that of the "*Units.h*", a file translation also will be performed.

Rebuild All incl. SIGMATEK-Classes

This menu item **recompiles everything**, regardless of the timestamp and whether they are user classes, or SIGMATEK classes.

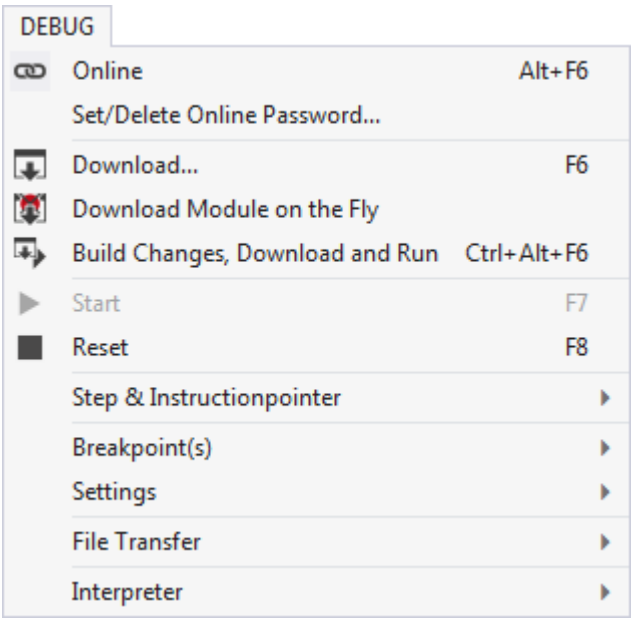
User classes:

Everything will be compiled.

SIGMATEK-Classes:

Everything will be compiled.

2.1.9 Debug Menu

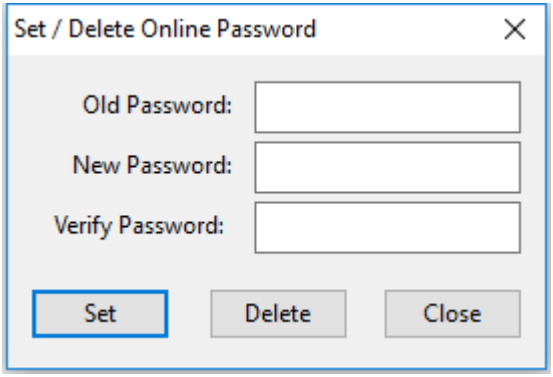


2.1.9.1 Online

Creates an online connection between the PC (project) and the control.

2.1.9.2 Set / Delete online password

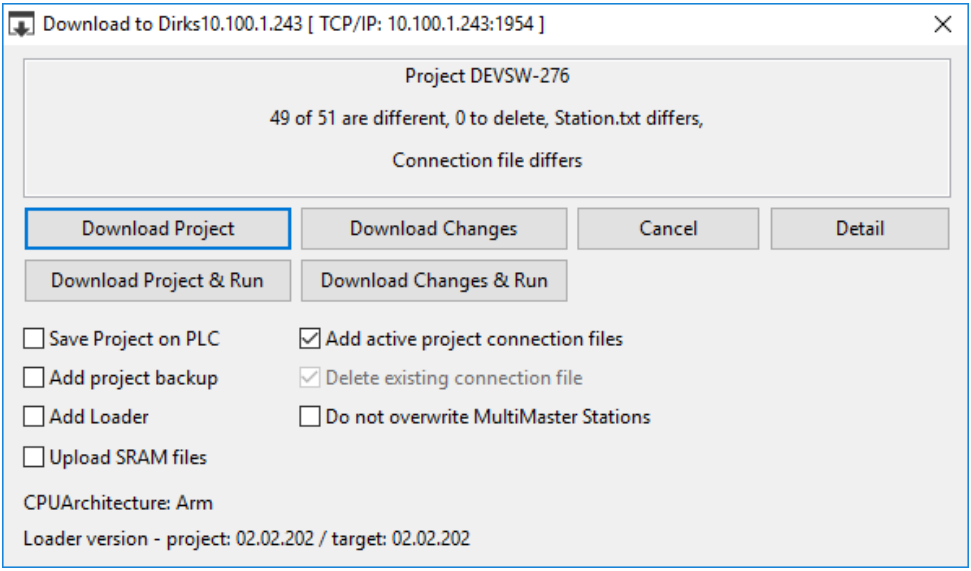
Here, an online password for the PLC can be set or deleted. If a password is set, it must be entered for an online connection. Otherwise no connection is possible. If this menu item is seleted, the following dialog appears:



Old Password	Must only be entered when a password was already entered and it should be changed or deleted.
New Password	The new password must be entered here.
Verify Password	Re-enter the password (confirmation)
Set	Set the new password.
Delete	Deletes the existing password – here, the correct password must be entered in Old Password.

2.1.9.3 Download

Downloads the project to the control.



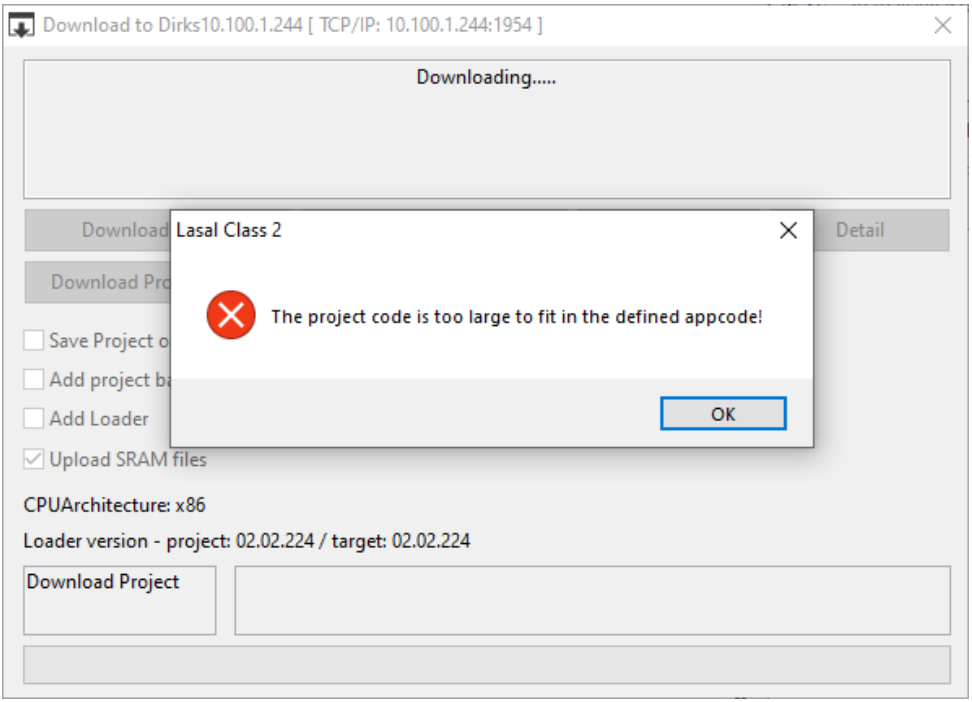
In the status display of the download dialog above, the project calibration data with the target system is found. Here, whether the project name is the same or which modules are different is shown.

The lower status display shows the loader version of the project and in the target system. With the comparison of the Loader versions, the use can decide whether the loader should be stored in the target system.

Download Project	Loads the entire project to the target system.
Download Project & Run	Loads the entire project to the target system and start the program.
Download Changes	Downloads only the changes to the target system. Download Changes can also be run for different project names.
Download Changes & Run	Downloads the changes and starts the program. Download Changes & Run can also be run for different project names.
Cancel	Closes the download dialog and ends the transfer.

Detail	Shows an overview of the changed program modules. Here, each module can be selected individually by setting the checkbox before each module name. By clicking on Detail, the Show All Files checkbox appears. With this option, all program modules (including the unchanged ones) are listed.
Save Project on PLC	By setting this option, the project remains stored in the target system after downloading.
Add project backup	A backup of the project is created in the control.
Add Loader	By setting this option, the loader is included in the download to the target system. If no loader is available in the target system, the option is set automatically.
Upload SRAM files	If SRAM files have already been saved on the control, they are uploaded to the PC before downloading and can be stored there in any location. This checkbox is only visible if SRAM files are actually present on the control!
Add active project connection files	If this option is set, the Project connection file is also saved. This checkbox is only visible if such a file exists!
Delete existing connection file	By setting this option, an existing connection file is deleted from the control. This checkbox is then only seen when such a file is available in the control!
Do not overwrite MultiMaster Stations	If MultiMaster stations are present both in the project and on the control, the latter will not be overwritten during the download if this option is selected. This checkbox is only visible if MultiMaster station(s) are present in the project and on the control!
CPUArchitecture	Shows the CPU architecture of the selected targets (currently Intel x86 and ARAM are supported).

If the project is too large for the AppCode memory in the controller, a corresponding message is displayed:



With the setting SET APPCODE xxxxxx in the file Autoexec.lsl the AppCode memory can be increased.

2.1.9.4 Download Module On the Fly

Transfers changes to the control during runtime without stopping the program.

The "Download on the Fly" function offers the ability to exchange modules in a project during the runtime. However a series of conditions exist that must be met.

A download on the fly is only possible if no changes have been made in the class interface as well as the object network (delete/create connections). The changes can only be made locally within the class.

A change within several classes is possible but under the same restrictions.

So the following is possible:

- Declaration of private methods applied/changed/deleted.

- Variables applied/changed/deleted.
- Changes in code sections of existing methods.
- Type definitions added/changed/deleted.
- Include files added/changed/deleted.
- No global (public) methods added/deleted (for write methods of existing servers also).
- No member variables or global/private variables applied or deleted (a name change is possible in principle, however no data types can be changed).
- No clients/servers added, changed or deleted from a class.
- The class and object names cannot be changed.
- No classes possessing their own tasks or Callback functions can be changed.

In the changed code, functions and interfaces can be accessed without problem.



LASAL does not hinder changes made with a "**Download on the fly**" that can cause problems. The programmer is responsible for debugging changes before they are assumed.



Since the project name is no longer included in the Lob-CRC calculation, it is possible to Download-on-the-fly when the project names are different from those on the control (the previously discussed conditions must first be met).



Linking during the "Download on the Fly" can be accelerated by providing 2 MB AppHeap for a binary search tree. The functionality can be enabled with "**DOFSEARCHTREE ON**" in the **autoexec.lsl** file. **This function is only available on controls with Salamander operating system!**

2.1.9.41 Class Configurations

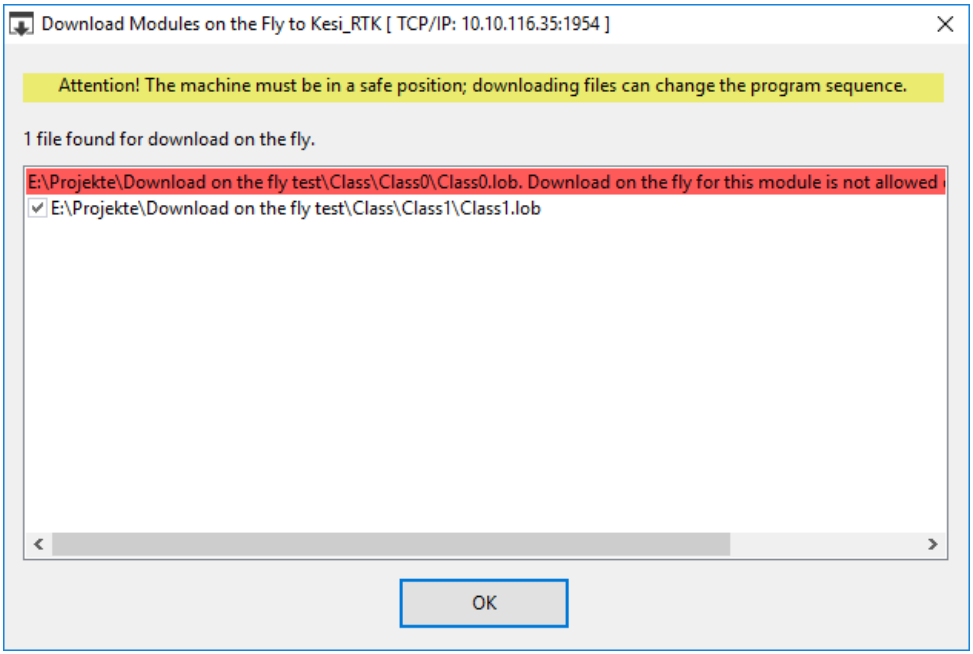
The object task times cannot be changed with for "Download on the fly", however the "**HighPriority**", "**Automatic**" and "**UpdateMode**" flags can be switched. Changing the version information of a class is not allowed.

2.1.9.42 Workarounds

- It is possible that a class must be changed frequently and thus require member variables should this be predefined (possibly as an array to provide sufficient space).
- The same can happen for clients and servers.
- Write methods for servers can be applied without first being assigned a special function.

2.1.9.43 Process

- Go offline
- Make module changes (code, variable declarations,...)
- Go online
- Build
- "Debug" "Download Module on the Fly" (or click on the corresponding Icon ). A window then opens in which all changed classes are listed. If classes are listed, which do not meet the above conditions, they are marked in red. All classes the meet the conditions have checkboxes in front of them (these are preselected). By clicking the checkmarks out of the check boxes, only selected classes can be loaded into the control. The top of the dialog informs about the danger of Download on the Fly. The user is asked to care for a safe position of the machine, as the download can make changes in the program sequence and thus in the behavior of the machine!



With each download, the Realtime, Cyclic and Background tasks are stopped for a short time (several milliseconds).

- In the status bar, "PLC=PC" must appear once all changed modules are downloaded to the control

2.1.9.44 Return Message

After successfully downloading, "PLC=PC" appears in the status bar. If the message remains PLC!=PC, there are other modules to update (with "Download Changes" "Detail", all changed modules are displayed.) or there are classes that have changed such that they no longer meet the conditions above.

If the changed class contains, for example, a callback function (a "Download on the fly" is not safe in this case), "PLC=PC" is still displayed although this process can lead to problems. The "Download on the fly" could require, depending on the number and complexity of objects available in the project, a considerable amount of time. During which, the background task is stopped. The cyclic and realtime tasks are also temporarily stopped to that the changed class can be loaded into the project. After the process is

complete, "PLC=PC" appears in the status bar. Because of the longer continuity a DownloadOnTheFly eventually takes, a dialog informing about the current process and the elapsed time is displayed in LASAL 2 during exchange process.

2.1.9.45 Special Cases

It is possible to assign or change a global **#define function** in a separate header file, which can be added new to the project. When downloading however, caution should be taken to ensure that each module to be linked in this header file is updated individually. A "Download on the fly" for pre-processor instructions is supported so long as no changes to the class declarations have be made (e.g.. **#pragma pack**).

Changes in the Init method are possible but have no effect during a "Download on the fly". This also means that the function pointer set during the Init phase (Callback, interrupt functions…), can't follow possible changes. Therefore, it is not always possible to change classes that use such function and if so the changes are extremely limited. Classes that apply their own tasks beyond the realtime, Cyclic or Background tasks also fall into this category since only these tasks are stopped during a "Download on the fly". Changes to a base class have no Affect on the derived classes. If the same class is used as a base class as well as an object in an object network, the changes in the code are also not assumed. Classes imbedded as objects can be updated during the runtime, even if the imbedded class is used again externally.

2.1.9.5 Build Changes Download and Run

Compiles the project, loads it into the PLC and starts it immediately.

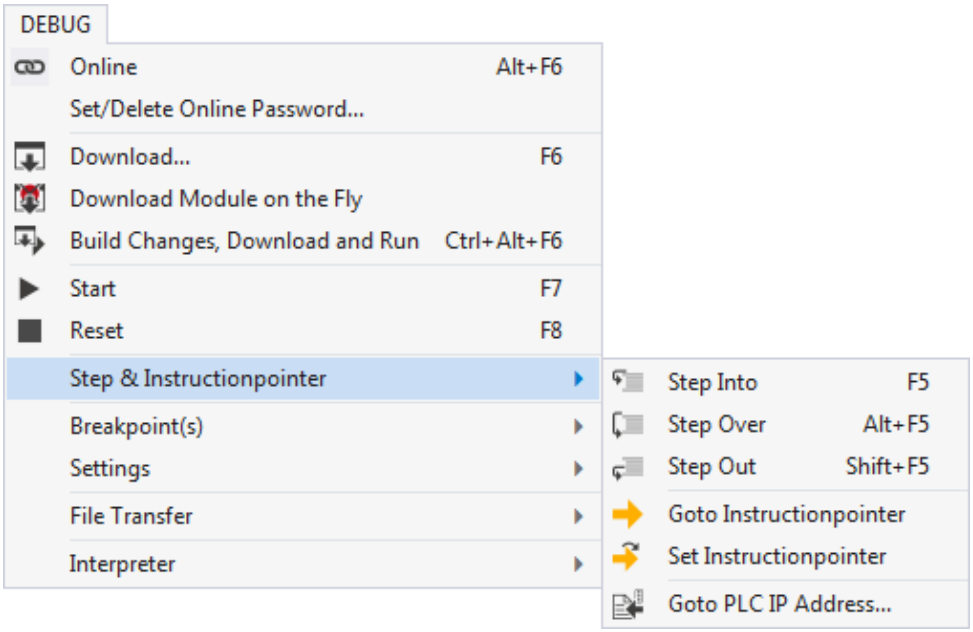
2.1.9.6 Start

Starts the project in the control.

2.1.9.7 Reset

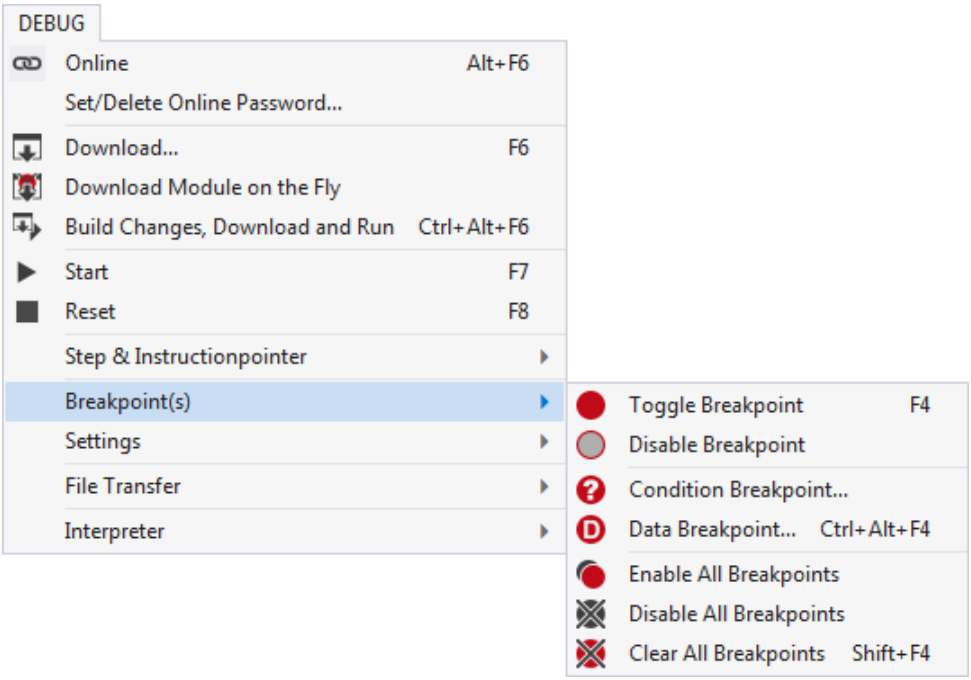
Stops the execution of the project in the control.

2.1.9.8 Step & Instructionpointer



Step Into	Executes the next instruction or jumps to a sub function if provided.
Step Over	Executes the instruction or the entire subvention if provided.
Step Out	Executes the rest of the function. The control is then located at the position where the function was called.
Goto Instructionpointer	Shows each line in the corresponding source file in the CPU.
Set Instructionpointer	The program code pointer is set on the line where the editor cursor is located.
Goto PLC IP Address	The function enables the address of a crashed PLC, which is retrieved from the log file event00.log, to be entered. Crashes that occur can therefore be investigated at a later time.

2.1.9.9 Breakpoint(s)

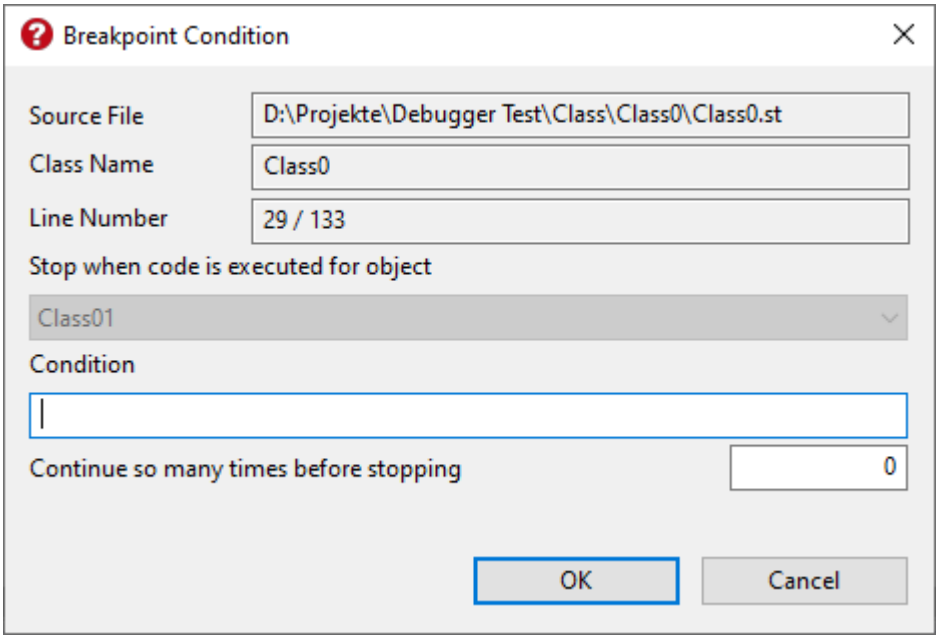


Toogle Breakpoint	Sets and/or deletes a breakpoint at the assigned place in the source code.
Disable Breakpoint	Deactivates a breakpoint without deleting it.
Condition Breakpoint	With this Breakpoint with the data of a certain object can be worked. The conditional breakpoint can be used if more than one object of a class are existent. In addition, almost anything that can be used in an IF can be used as a condition. Below this table there is a more detailed description of condition breakpoints.
Data Breakpoint	Via a data breakpoint the corresponding address of a client, server or data element can be defined in order to debug, if this memory position is accessed.
Enable All Breakpoints	Resets all breakpoints (after Disable).
Clear All Breakpoints	Deletes all breakpoints set in a program.

Disable All Breakpoints	Disables all breakpoints.

Condition Breakpoint

Condition Breakpoints are a powerful tool to trigger a breakpoint only under very specific conditions. Clicking the **Condition Breakpoint** icon will display the following window:



Source File

The ***.st file of the class including path** is specified here. This field is **not ready for input**.

Class Name

The **class name** is specified here. This field is also **not ready for input**.

Line Number

The **line number** of the breakpoint is displayed here. The first number is the line number in the code editor, the second is the line number e.g. in Notepad, if the *.st file is opened with it. This field is also **not ready for input**.

Stop when code is executed for object

This is a **pure selection field** where you can select for which **object of the class** the breakpoint should apply. Will only be enabled if there are multiple objects of a class.

Condition

Here the **condition** can be entered, at the fulfillment of which the breakpoint becomes effective. Almost everything can be entered here that also works as a condition in an IF construction in the program code.

```
iCount >= 200 // The breakpoint becomes effective when the value of the variable iCour
iCount < iPoint // Program execution stops when the value of iCount is below the value
(iCount < iPoint) AND (iCount > 10) // Execution stops when the value of iCount is les
```

...etc. ...



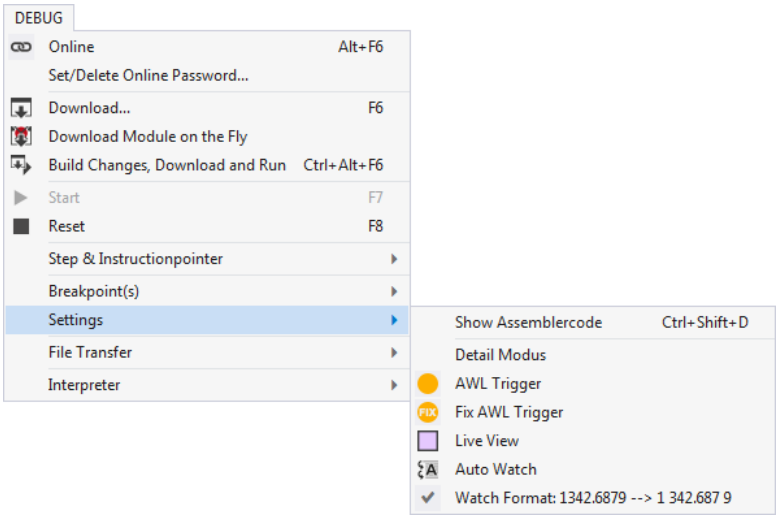
Attention

Global variables do not work in condition breakpoints!

Continue so many times before stopping

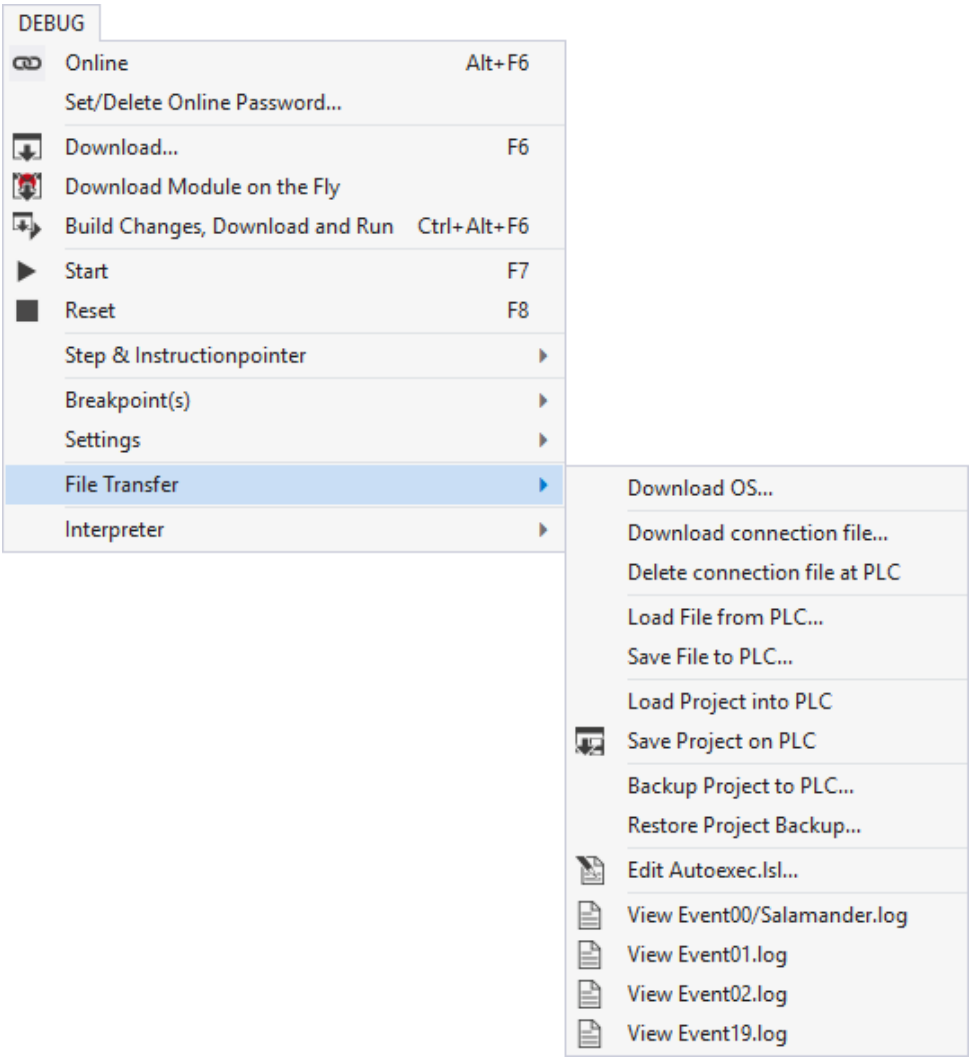
The program execution stops only when the code line has already been passed x times. If, for example, 5 is entered here, the execution stops only after the 6th time the code line has been passed.

2.1.9.10 Settings

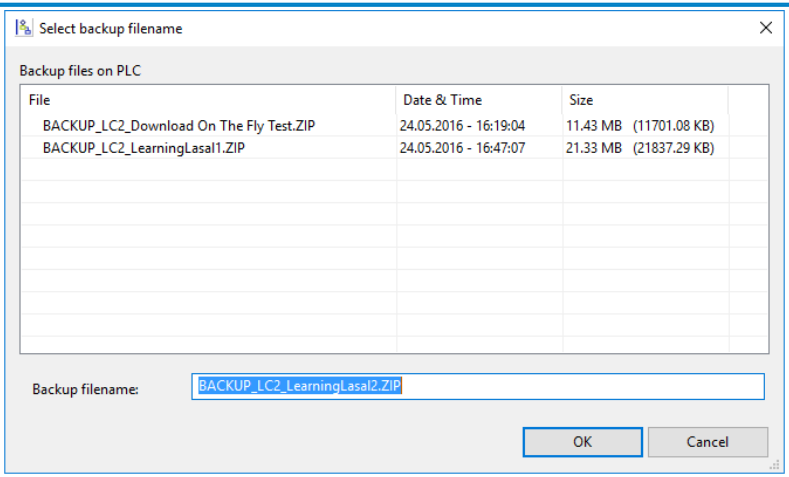


Show Assemblercode	Shows the assembler code (in IL) for each program. The corresponding file must be interpreted without error.
Detail Modus	Enables and/or disables the Detail mode for clients and servers. If the Detail mode is enabled, the entire data structure of the client or server is shown when debugging.
AWL Trigger	Enables/disables the AWL trigger. The AWL trigger supplies the corresponding line of code using a counter on which the cursor is located.
Fix AWL Trigger	Enables/disables the Fix AWL trigger. The Fix AWL Trigger functions similar to the AWL Trigger, except that with further clicking, exiting the editor, etc. it remains at the fixed point in the code while the AWL trigger “follows”.
Live View	Switches on and off the display of all valid variable of the currently visible area in the Code Editor. For more information see chapter The Views Auto and LiveView
Auto Watch	Activates/deactivates the function "Auto Watch". For more information see chapter The Views Auto and LiveView
Watch Format: 1342.6879 --> 1 342.687 9	Turns on/off a space character as thousand tags at the value display in the watch window.

2.1.9.11 File Transfer



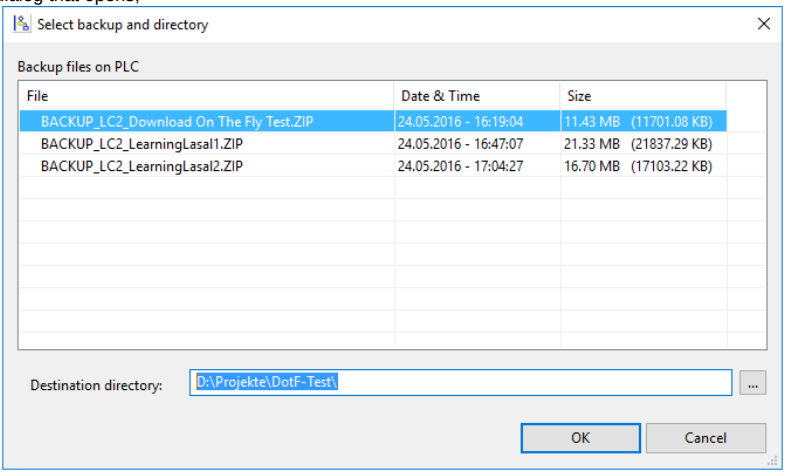
Download OS	Loads a LASAL operating system in the control (opens the Open a LasalOS file window). The following note is only valid for Salamander operating systems. If the version number of the currently installed Salamander operating system is in the range of 09.03.102 to 09.03.143, problems with installed SW packages may occur. There are two possibilities to avoid these problems: Possibility 1: All installed SW packages must be uninstalled with "PACKAGE UNINSTALL" before a new Salamander operating system is installed. After installing the new Salamander operating system, the SW packages must be reinstalled with "PACKAGE INSTALL". For details see the CLI command PACKAGE. Possibility 2: The new operating system must be installed twice in immediate succession.
Download connection file…	Saves the temporary connection from the Connection File Editor to the PLC.
Delete connection file at PLC	Deletes the existing connection data in the PLC, which was loaded using the Download connection file… or through the Connection File Editor.
Load File from PLC	Loads a file from the PLC (opens the PLC Filetransfer window in which can the name and path of the desired file are entered).
Save File to PLC	Saves a file to the PLC.
Load Project into PLC	Loads a project stored in the control to the user memory in the PLC.
Save Project on PLC	Stores a project in the remnant memory of the control (the same function can be performed with the toolbar button  in the main screen).
Backup Project to PLC	Saves a zipped copy of the current project in the PLC. In the dialog the now appears, the backup can be assigned its own name.



The proposed default file name is generated in the following way: BACKUP_LC2_project name.ZIP. Older name formats (BACKUP02.ZIP) still are supported. On saving a backup and if such an old variant already exists, you are prompted whether to old one should be removed.

Restore Project Backup

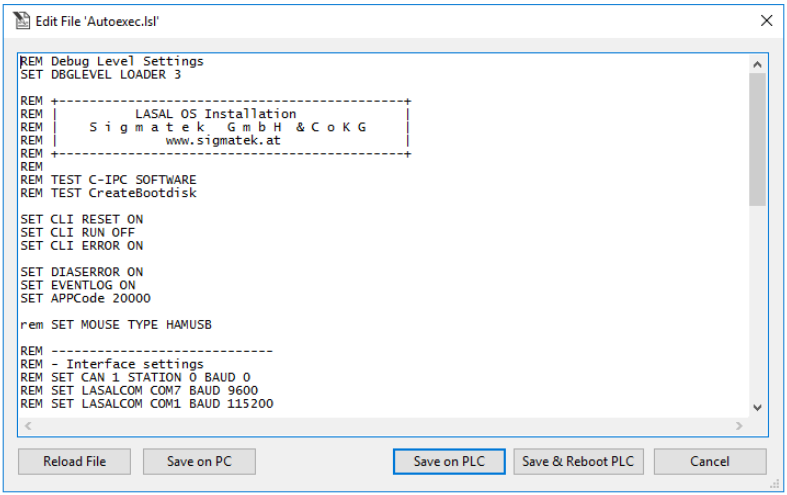
Loads the backup project from the control (zip file) and unpacks it in the assigned directory. In the dialog that opens,



all backup files stored in the PLC are listed. The desired backup file and the target directory in the PC can be entered.

Edit Autoexec.lsl

Allows editing of the Init file Autoexec.lsl.



Reload File	Reloads the file.
Save on PC	Stores the file in the PC.
Save on PLC	Stores the file in the control.
Save & Reboot PLC	Stores the file in the control in the control and reboots it.
Cancel	Aborts editing without saving and closes the window.

View Event00/Salamander.log

Opens the Event00.log file (Rtk-OS) or the Salamander.log (Salamander-OS), into which messages and information on various events of the operating system are entered. This file can be used for further troubleshooting.

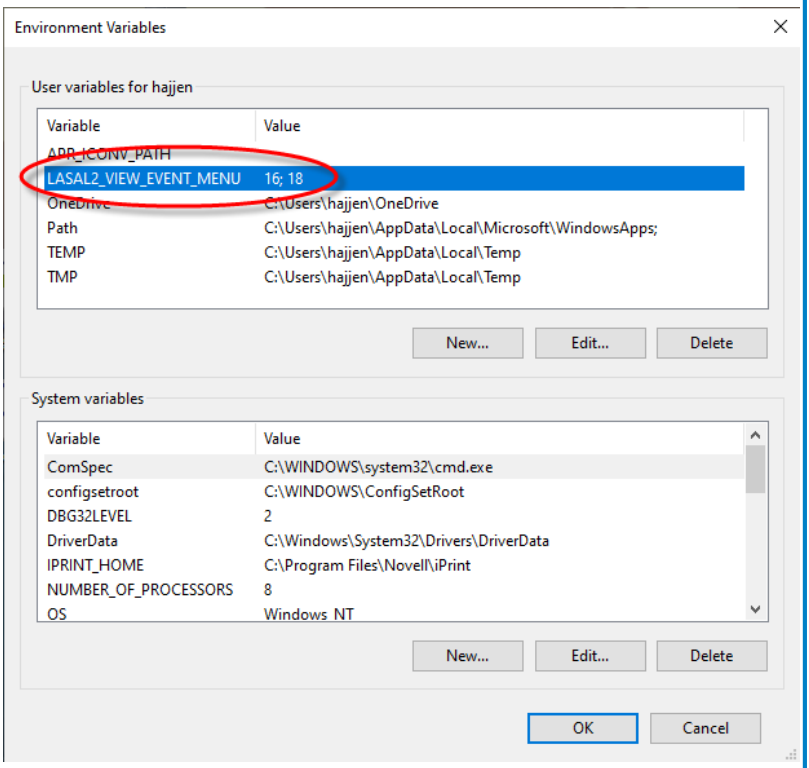
View Event01.log

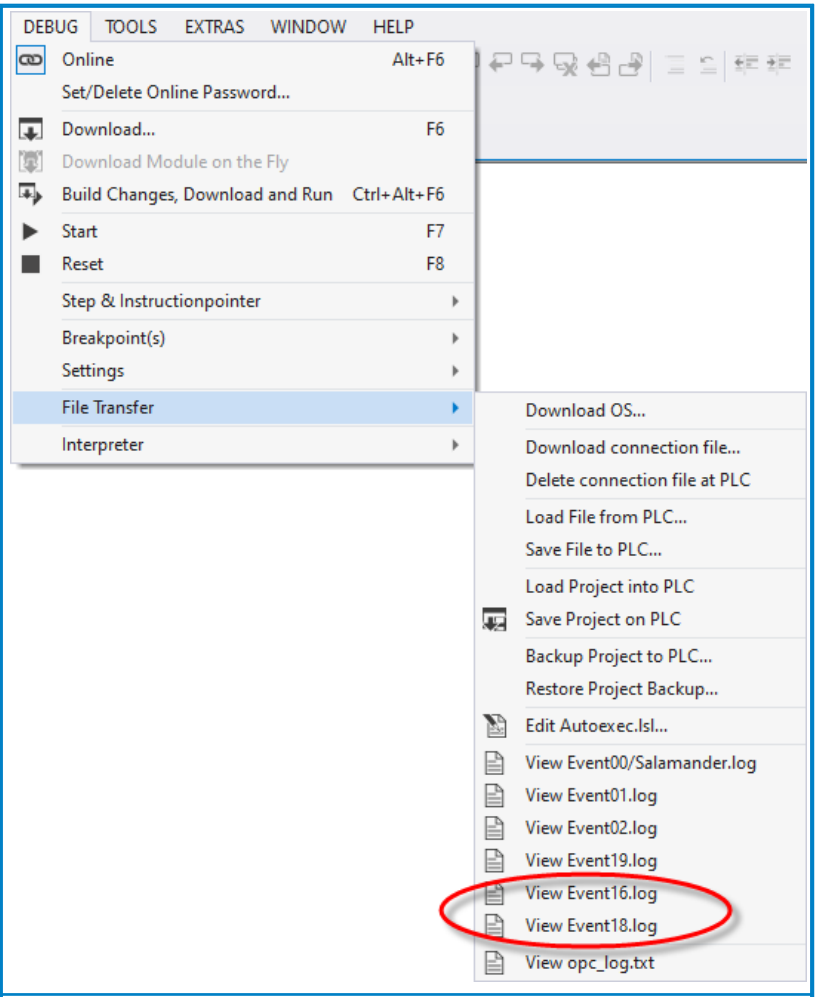
Opens the file, with which the user can also log text. The functions for logging in are contained in the _SysMsg class, which is required in the LASAL CLASS 2 project in order to write in this Log file.

View Event02.log

Opens the file, with which the user can also log events. Here, a maximum of 5 DINT values can be quickly logged (_SysMsg class in the LASAL CLASS 2 project required).

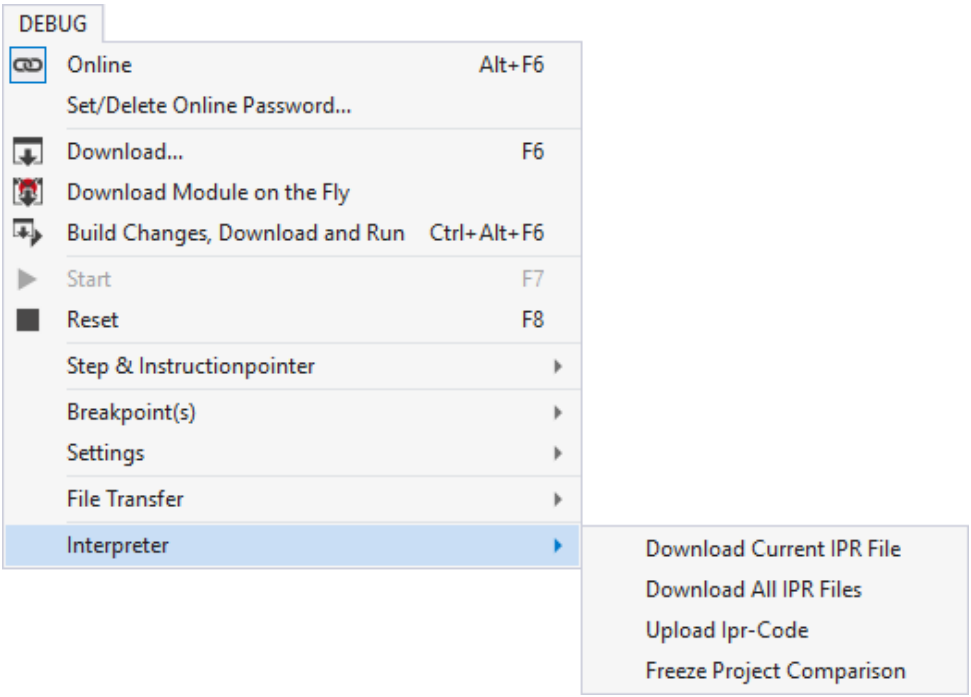
View Event19.log	Opens the file in which the loader messages (part of the application) are logged. These 4 log files are stored in the SYSMSG folder of the PLC.
ViewEventXX.log	If the Windows environment variable LASAL2_VIEW_EVENT_MENU is introduced and value(s) between 0 and 19 are specified (ViewEvent16.log and ViewEvent18.log are specified in the image below), the corresponding ViewEventXX.log files are also listed here and can be opened:





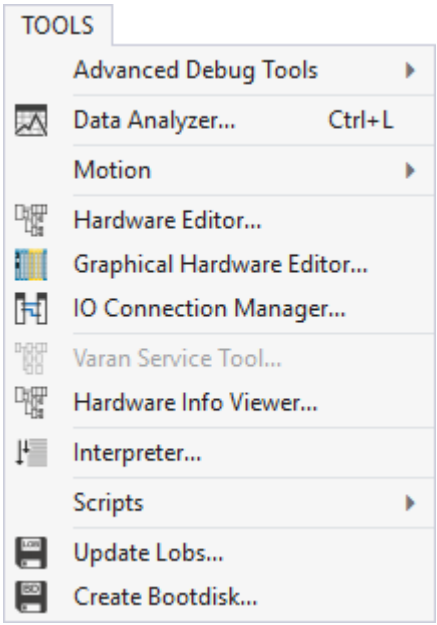
Delete OPC-UA file...	Deletes an existing OPC UA configuration file from the control.

2.1.9.12 Interpreter

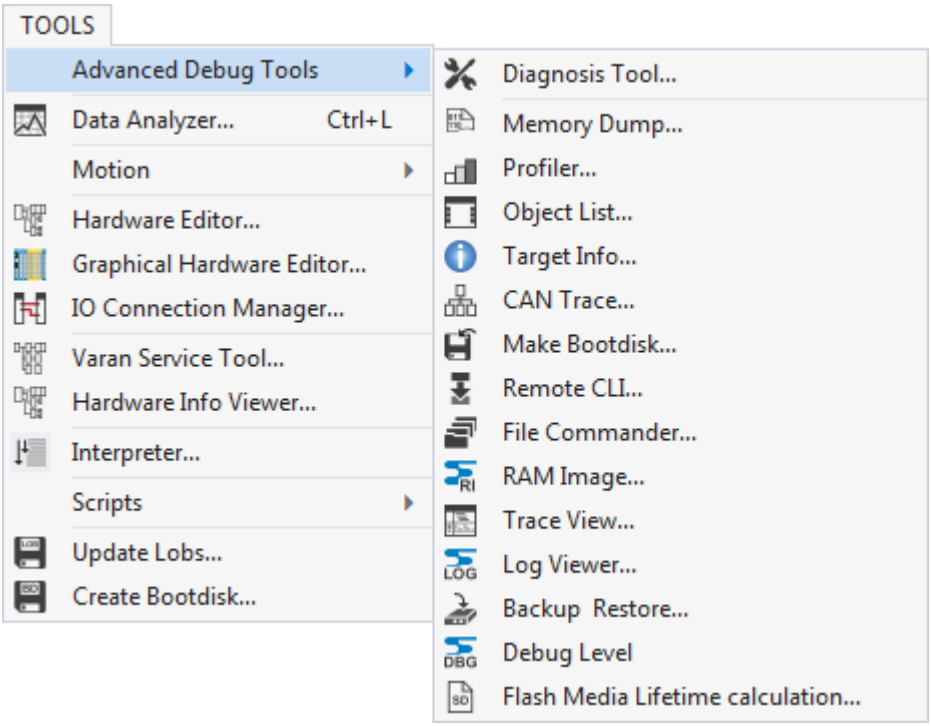


Provides functions for debugging the interpreter. To activate this menu item, under **Tools**, the interpreter tree must be started with the menu item **Interpreter** and an interpreter file created. For further information, see the chapter Interpreter.

2.1.10 Tools Menu



2.1.10.1 Advanced Debug Tools

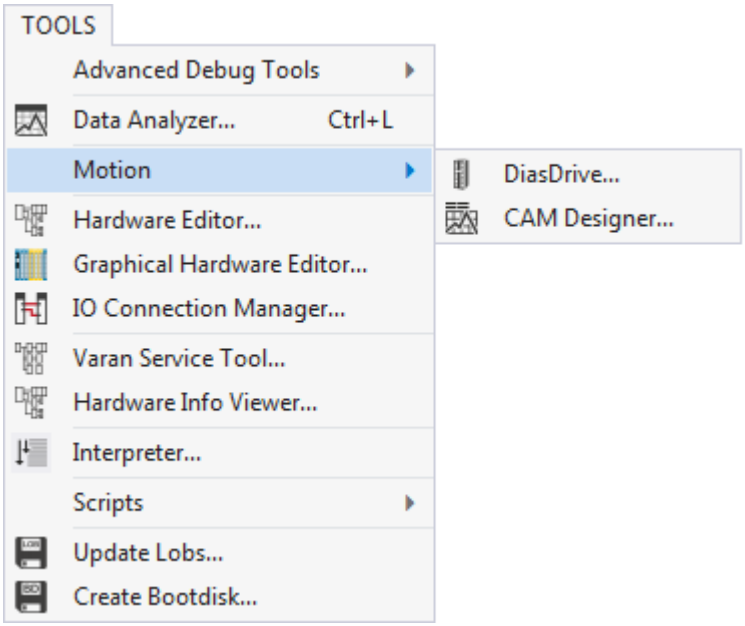


The Advanced Debut Tools menu correspond to the individual items of the PLC Diagnosis program packet, which can be called here under the menu item Diagnosis Tool... as well as by clicking on the PLC Diag icon in the tool bar  . See the chapter Advanced Debug Tools for the individual functions.

2.1.10.2 Data Analyzer...

Opens the surface for data recording (oscilloscope). The same can be accomplished with a click on the DataAnalyzer Icon  in the tool bar (see chapter DataAnalyzer).

2.1.10.3 Motion



DiasDrive...	Opens the DiasDrive Edit window and activates the Dias Drive menu in the menu list. The same can be accomplished with a click on the DiasDrive Icon in the tool bar (see chapter DiasDrive).
CAM Designer...	Opens the electronic CAM editor. More information can be found in the chapter CAM-Designer.

2.1.10.4 Hardware Editor...

Open the hardware editor. This can also be achieved with a click on the hardware editor icon in the tool bar (see chapter Hardware Editor).

2.1.10.5 Grafical Hardware Editor...

Opens the graphic hardware editor. This can also be achieved with a click on the hardware editor icon  in the tool bar (see chapter Grafical Hardware Editor).

2.1.10.6 IO Connection Manager...

Opens the IO ConnectionManager. This can also be achieved with a click on the ConnectionManager  in the toolbar (see chapter IO Connection Manager).

2.1.10.7 Varan Service Tool...

Opens the Varan Service Tool, which provides service and diagnosis functions for Varan devices. This tool is installed apart from LASAL CLASS 2, but can be opened via this menu command. The Varan Service Tool has its own online help including description of all functions.

2.1.10.8 Hardware Info Viewer

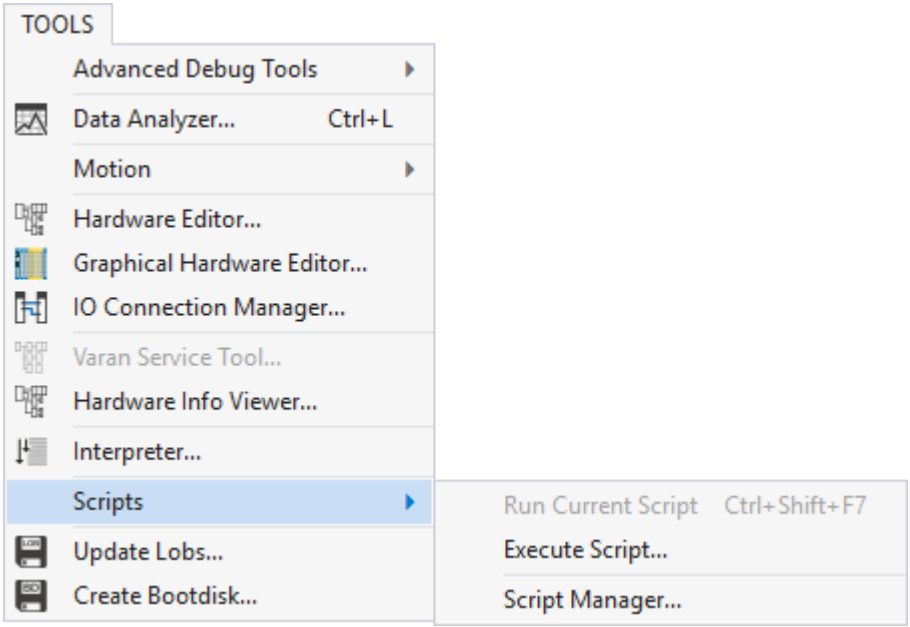
Opens the Hardware Info Viewer, which is principally a display-only form of the VARAN Service Tool.

2.1.10.9 Interpreter...

Here, the interpreter tree is started. This can also be achieved with a click on the interpreter icon  in the tool bar (see chapter Interpreter).

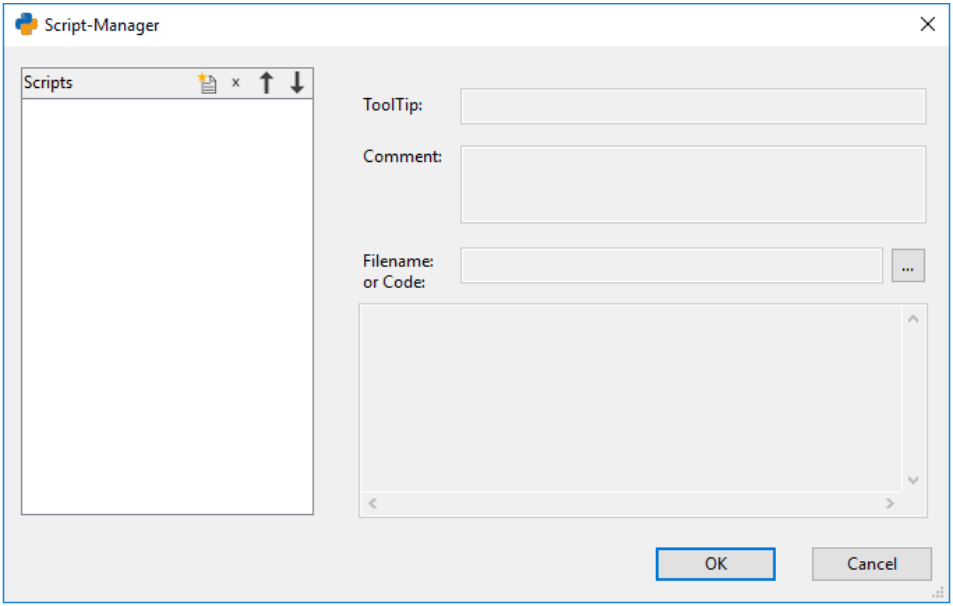
2.1.10.10 Scripts

Allows a Python script to be started.



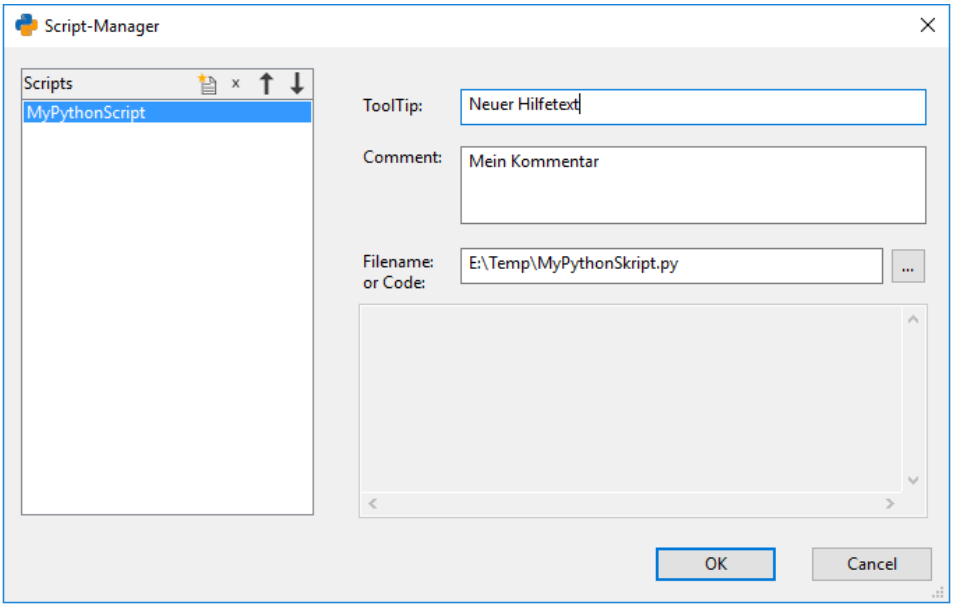
Run Current Script	Start the script file opened source code editor when it is marked.
Execute Script...	A script file can be selected, which is then executed.
Script Manager	Starts the script manager. The **Script Manager window** opens.

The Script Manager

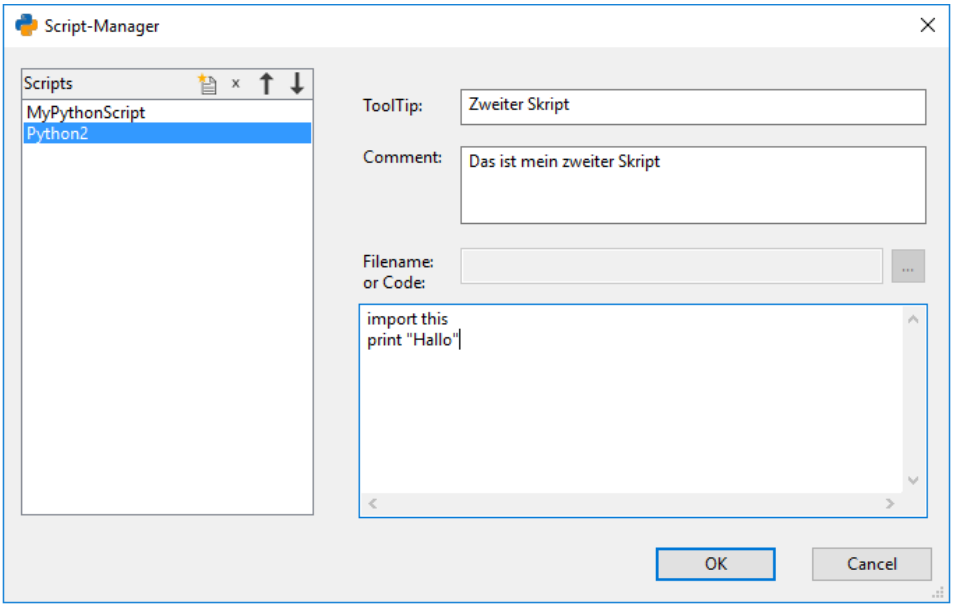


By clicking , a new Python script can be created. The icon deletes the selected script, and moves it in the list. In the **Comment** field, commentary can be entered. Under **ToolTip**, a help text can be entered.

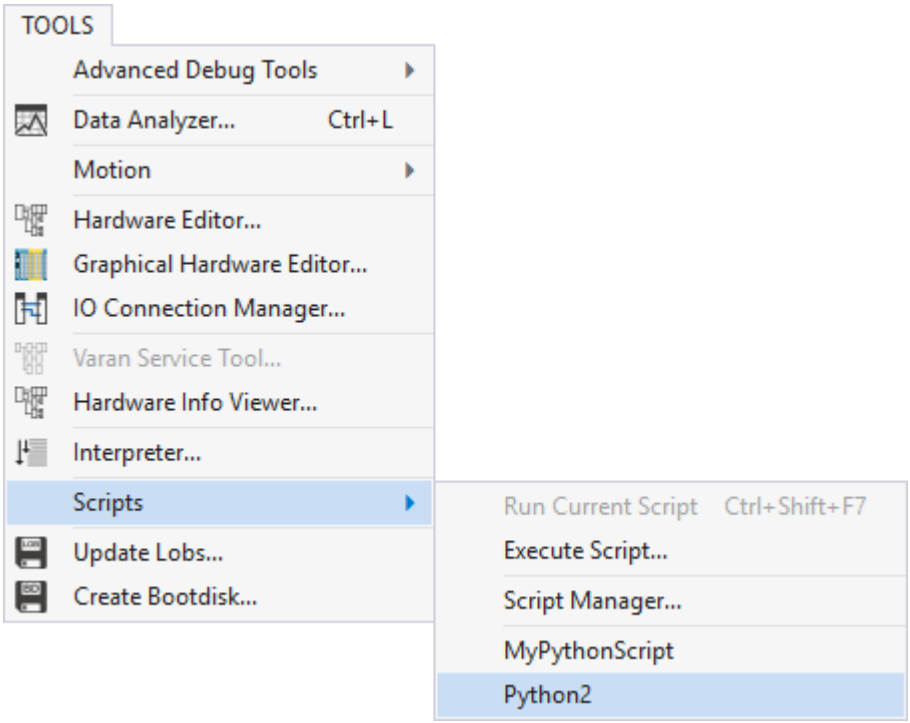
If the File name field is clicked, something entered and confirmed; the **Code** field is then disabled and an existing script can be selected by entering the path and file name or clicking the More button.



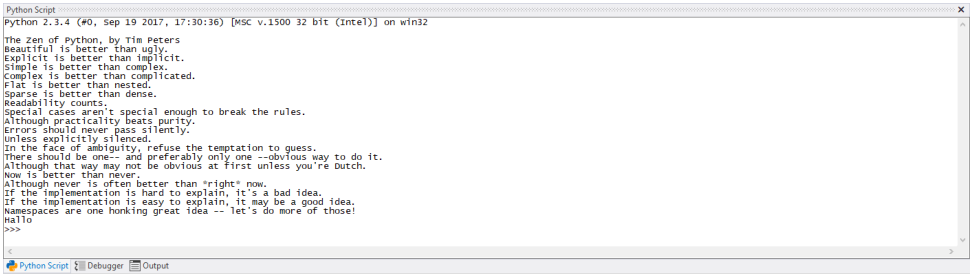
If the **Code** field is clicked, something entered and confirmed; the **File name** is disabled and the Python code can be entered directly into the **Code** field.



A script created in the script manager appears in the **Script menu** with name and can be directly started with a click.



The Python script output can be seen in the Python script window.



For more information about Python see chapter Scripting.

2.1.10.11 Update Lobs...

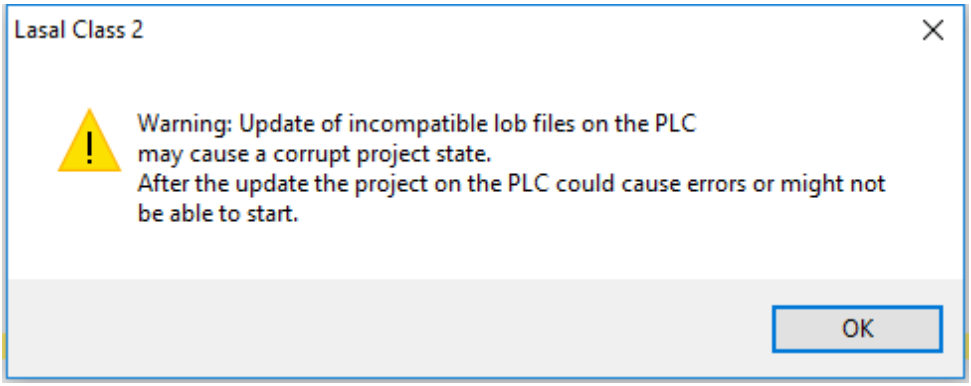
This function allows the updating of a single (or several files) source file (*.lob), without having to recompile and reload a project into the PLC. A boot disk (e.g.: USB stick) is created with the defined lob files. If the Add Update Script checkbox is activated, an additional autoexec.lsl and Rexx script are created that automatically executes the update. If Online is selected, the update is performed online when the IP of the PLC is entered!

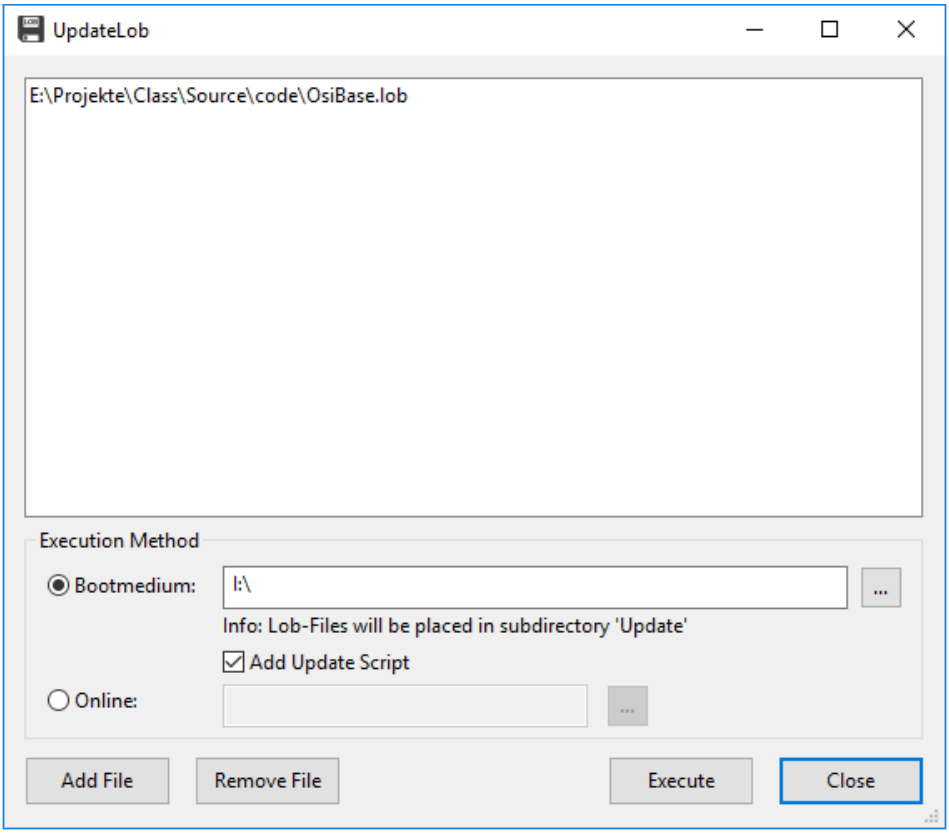
Attention!

An update of the control program over this mechanism is critical, as the compiler cannot check whether the module to be updated harmonizes with the rest of the program.

This must be ensured by the user !

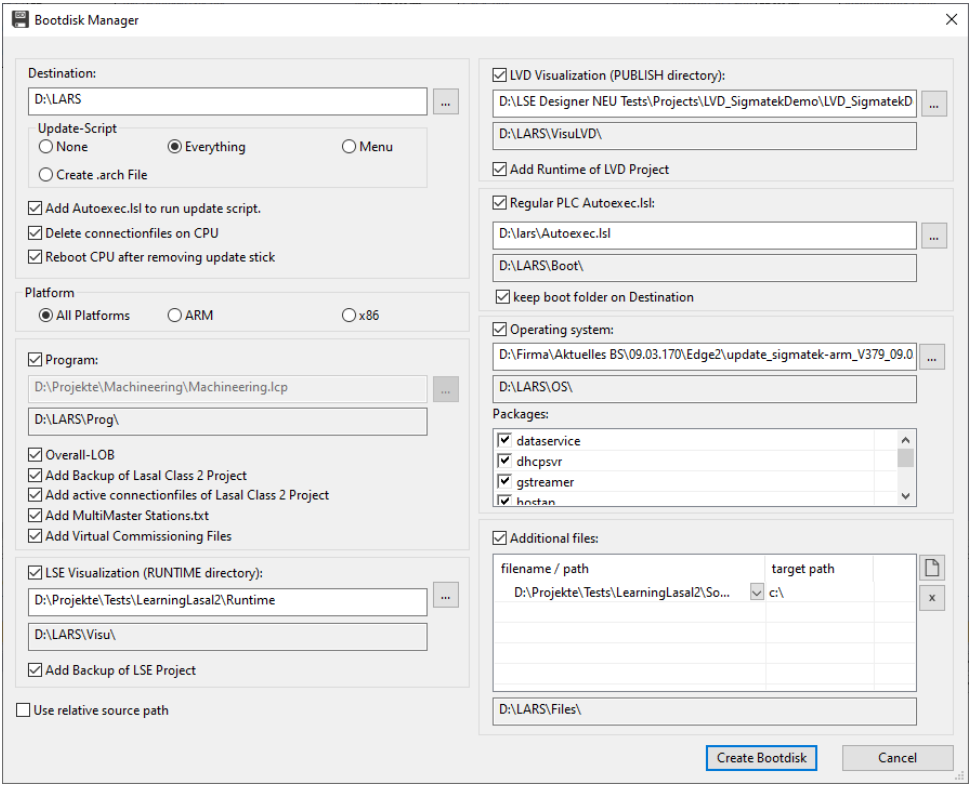
For this reason, a confirmation request first appears when this function is selected:




UpdateLob
E:\Projekte\Class\Source\code\OsiBase.lob
Execution Method
Bootmedium:
I:\
...
Info: Lob-Files will be placed in subdirectory 'Update'
Add Update Script
Online:
Add File
Remove File
Execute
Close

2.1.10.12 Create Bootdisk...

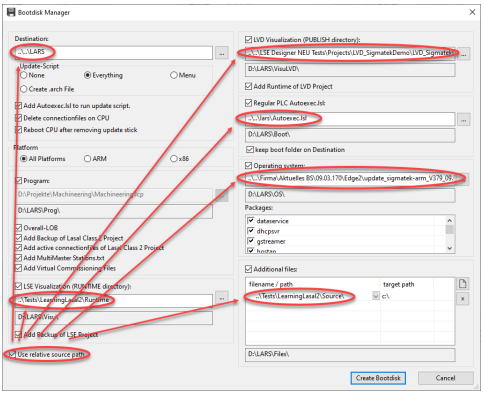
To create a start disk, select **Tools, Create Bootdisk** from the menu list. In the opened dialog you can select a LASAL project, a LSE project an operating system file for your bootdisk. A bootdisk is used to load a program via disc, USB stick etc. to the control.



Destination	The path in which the bootdisk files are stored.
Update-Script	Defines the Rexx script that should be used for the bootdisk.
None	No Rexx script added.
Everything	Everything on the boot disk is updated.

	Menu: A boot menu appears on the target system to select the respective update process. Create .arch File: This is a special case and is used for SIGMATEK diagnostic devices (e.g. HGT). A total of three files are created: • The entire folder structure is packed in one file (bootdisksettings.arch). Hereby, the C:\LARS\lslwork folder is created for the project instead of the C:\LARS\Prog\ and for the visualization, the C:\LARS\mpc folder instead of C:\LARS\visu. • An individual file is created for the CRC checksum (bootdisksettings_crc.cks). This two files are now stored in the control, which is therewith prepared for the diagnosis. • The rexx file REMOTEDIAG.REX, must be loaded into the diagnostic device. Diagnostics are then started automatically (after connecting with an Ethernet cable).
Add Autoexec.lsl to run update script	If this option is set, a standard autoexec.lsl for calling the Rexx script is copied to the boot disk. At deactivation of this option, the autoexec.lsl already existent on the target system has to be used for calling the Rexx script. The autoexec.lsl on the control will be overwritten in no way. This option is not available (grayed out), when the **Create .arch File** checkbox is selected.
Delete connectionfiles on CPU	If this option is set, a *.lcf file (Lasal Connection File) that may already exist on the CPU is deleted. Such a *.lcf file can be created in the Connection Manager via the Download button and downloaded to the CPU. For more information see chapter The Project Connection Files Window.
Reboot CPU after removing update stick	If this option is set, the CPU automatically reboots after the USB stick is removed. This option is not available (grayed out), when the **Create .arch File** checkbox is selected.
Platform	Here you choose for which platform to create a bootdisk. All Platforms: In the folder "Prog" **an ARM as well as an Intel** folder is created. The Rexx script tries to find out the platform of the CPU. If this is not successful, the MENU script asks, whether (if the according folders exist) an update for an ARM or a x86 CPU should be executed. If no folder exists, it will be searched for via the Prog path. **Attention!** If under the topic "**Update-Script**" the option "**Everything**" has been selected, **an error is output** in any case, if the target could not be found. ARM: The bootdisk is created for the ARM platform.

	x86 The bootdisk is created for the Intel (x86) platform.
Program	Specifies the LASAL project. If a LASAL project is started while execution of "Create Bootdisk…" the path cannot be changed (the loaded project is used automatically). If no project is loaded currently, any project can be selected. With the checkbox of course it can be decided whether the LASAL project should be a part of the boot disk or not. Under the defined project the target path is displayed. If only the LASAL project is selected for bootdisk creation, the target path corresponds to path defined in the upper area of the dialog. If several components are selected, also the sub-directory **Prog** is attached to the target path.
Overall-LOB	With this option one single LOB file is created instead of several LOB files of the program modules. Thus booting via USB stick is faster.
Add Backup of Lasal Class2 Project	With this option, a ProjectBackup (SourceCode) is also stored on the bootdisk. This option is not available (grayed out), when the **Create .arch File** checkbox is selected.
Add active connectionfiles of Lasal Class 2 Project	With this option, all active connection files of the project are also saved.
Add MultiManster Stations.txt	With this option the Multimaster stations are also copied to the bootdisk.
Add Virtual Commissioning Files	With this option the Virtual Commissioning Files (Simulation-Configurations) of the project are also copied to the bootdisk.
LSE Visualization (RUNTIME directory)	Specifies the Runtime directory of the LSE project. With the checkbox it can be decided whether the visualization program should be part of the boot disk or not. Under the defined Runtime directory the target path is displayed. The target path defined in the upper area of the dialog is extended by the sub-directory **Visu**.
Add Backup of LSE Project	With this option, an LSE project backup is also stored on the boot disc. This option is not available (grayed out), when the **Create .arch File** checkbox is selected.
LVD Visualisation	Specifies the **Publish directory** of a Lasal VISUDesigner project. Use the checkbox to decide whether the LVD visualization program should be part of the startup disk. The target path is displayed below the specified Publish directory. The target path set in the upper part of the dialog is extended by the subdirectory **VisuLVD. This functionality is only available starting with LVD version number 01.02.001.1600!**
Add Runtime of LVD Project	This option also copying the **Runtime from the LVD project directory** to the boot disk. If this option is **not selected**, the LVD project is copied **without Runtime** and **any existing Runtime is not overwritten / deleted**.
Regular PLC Autoexec.lsl	Specifies the path and the file name of the autoexec.lsl file. The checkbox can be used to decide whether the autoexec.lsl file should be updated or not. If the check box „**keep boot folder on**

	Destination" is selected, the Autoexec.lsl file remains in the boot folder. If the checkbox is not selected, the entire boot folder is overwritten.
Operating system	Specifies the path and file name of an operating system file. With the checkbox it can be decided whether the operating system update should be part of the boot disk or not.
Packages	If packages are present in the folder specified in the "**Operating system**" field, they are displayed in this field and - if selected by the corresponding checkbox - copied to the startup disk together with the operating system. When booting the startup disk in the PLC, first the operating system is installed, and then all selected packages.
Additional Files	Here, the use can specify additional files (*pdf, *ini... etc.) that should also be saved. Thereby, a path or file name can be entered. With the  button, a new file / path can be added; with the  button they can be deleted. If a drive different to C: is entered in the target path, in the boot media under FILES a folder with the drive letter (D, E, etc.) is created. The REX script resolves it, so that the files are copied to the correct target drive. For C: no separate folder is created - everything is handled like before.
Use relative source path	If this check box is marked, the relative path is used in files / directory fields: 

If the bootdisk files are stored on a USB stick and played through this to the PLC, it can be read directly on the hardware - in the case of a C-IPC on the 7-segment display, otherwise on the status LEDs - whether the update is ready / has worked.

7-segment display:

If a 7-segment display (e.g. C-IPC) is available, the number **99** (update worked) or **98** (update failed) is displayed.

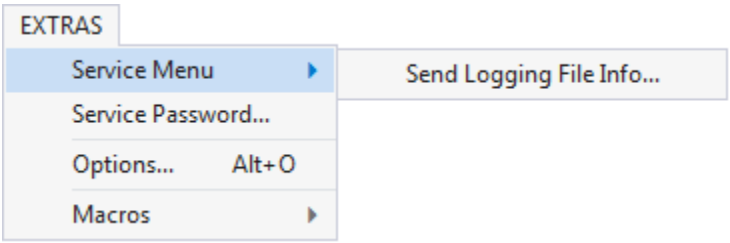
LEDs:

During the update, **the status LED flashes**.

When the update is finished, **the status LED is lit continuously**.

In the event of an error during the update, **the Error LED lights up**.

2.1.11 Extras Menu



2.1.11.1 Service Menü

This menu is used by SIGMATEK software developers to change internal settings in LASAL CLASS 2. The menu can only be accessed with the correct service password.

Send Logging File Info

This is the log file from LASAL 2 (all user actions are logged within) and can be sent with this command.

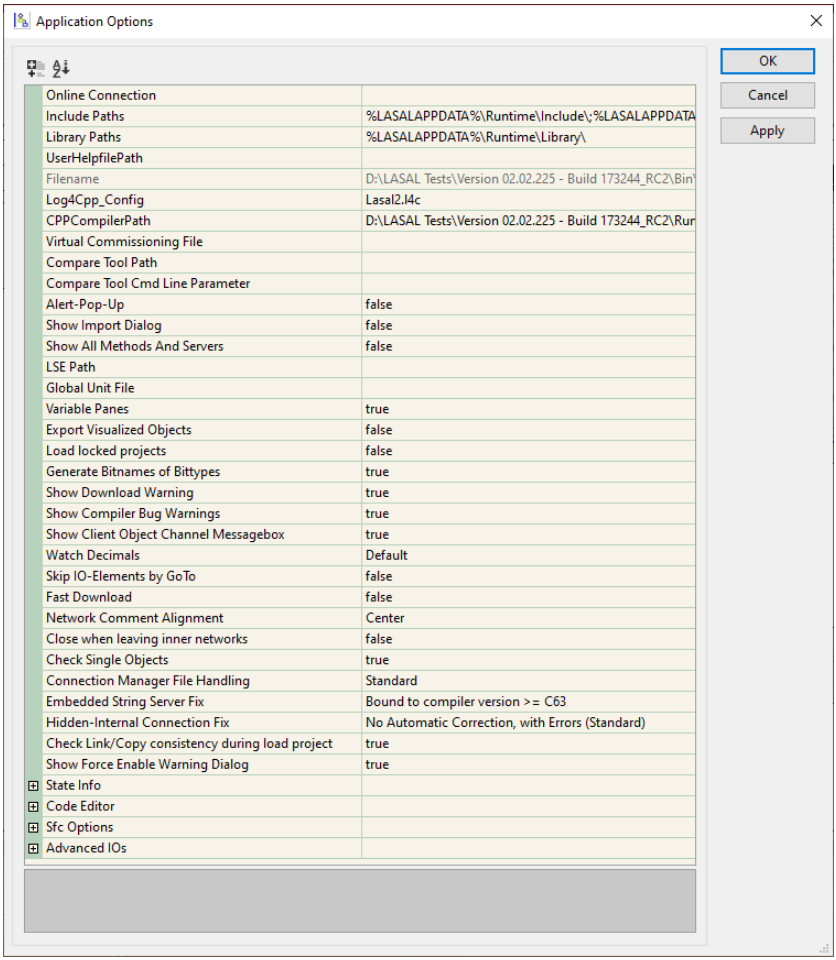
2.1.11.2 Service Password

Here, a password can be entered to change SIGMATEK classes or to perform a firmware upgrade in the DiasDrive.

2.1.11.3 Options

Opens the **Application Options window**, in which various application settings can be made.

With help form this function, various application options can be set.



Online Connection

Define which connection is used to go online. This setting is only active if no project is loaded. Therefore an online connection can also be established without a project. Press the button to open the **LASAL Online Configurations window**.

Include Paths

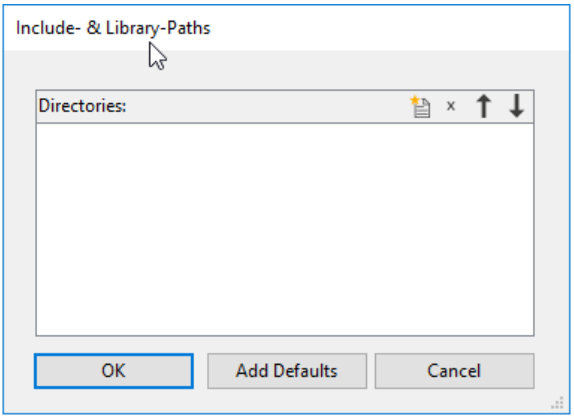
Here, the user enters the include paths, either directly using semicolons or by pressing the button to manage them in the **Include Paths window**. These settings are for the entire LASAL CLASS 2 installation!

Include Paths simplifies the handling of includes in the code. With the instruction **#include <Myheader.h>**, every Include path is searched for the file specified in the project or tool.

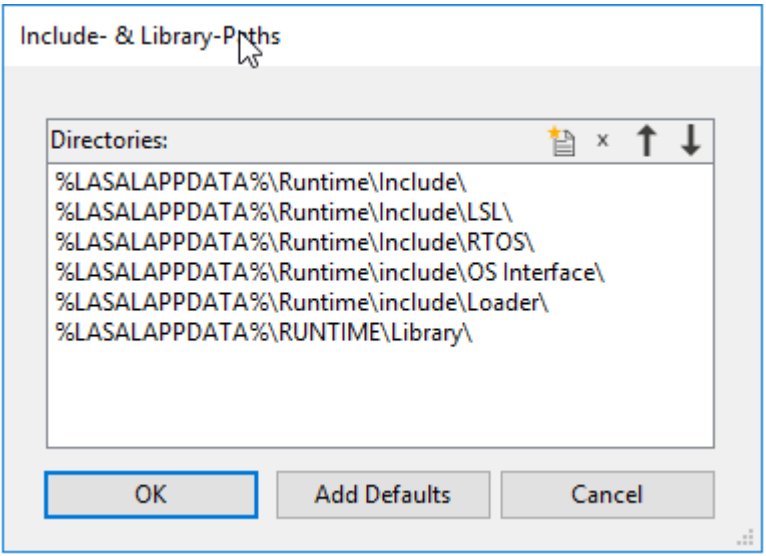
Note:

With the Include path from LASAL 2 (under **Extras**, **Options**), **first no** Include paths are shown.

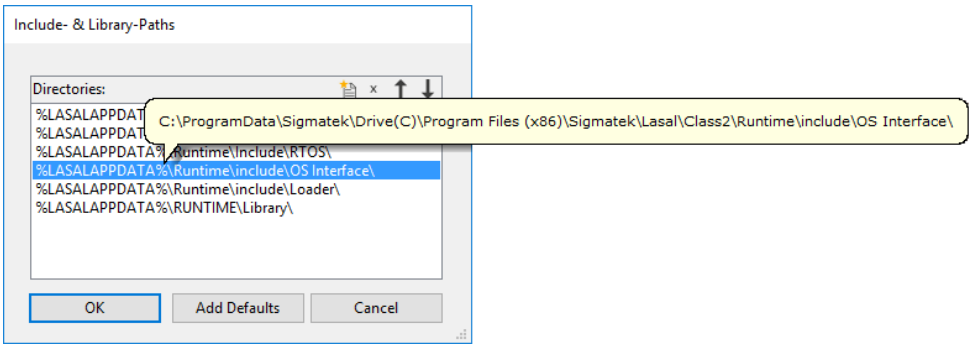
With the More button (), the **Include Paths** dialog is open.



With the button **Add Defaults** default paths can be added, which point to folders with system header files:



The actual path is provided as a Tooltip (the environmental variable %LASALAPPDATA% is replaced by the application path):



With a right click on a selected entry, its context menu is called. If the menu item **Show in Explorer** is selected, an Explorer window opens in which the path is then displayed.

Library Paths

Here, the library path is defined directly using semicolons or by pressing the button to manage them in the Library Paths window. These settings are for the entire LASAL

CLASS 2 installation! If a library in a defined library path is linked, the reference of the library in the project file is saved relative to the library path rather than project.

Analog to **Include Paths**, the option is available to open a dialog with the paths using the More button (); wherein Tooptips contain the actual paths. And with **Show in Explorer**, a selected viewed in a newly opened Explorer window.

A characteristic of ever library path specific to LASAL is that the projects in the library path and their sub folders appear in the library tree through the context menu and can be selected directly.

UserHelpfilePath

The programmer has the option create a help file for self-programmed classes. This help file is then placed on the path defined here. If the class is now marked and the F1 key is pressed for the online help, the user-defined document is opened.

Filename

With this property, the file name (including path) of the project is displayed and cannot be changed.

Log4Cpp_Config

Here, the path for the logging library can be defined by clicking on the  button and selecting the desired path.

CPPCompilerPath

Here, the path of the CPP compiler for the LASAL application is defined. Push the  button to select the desired path.



Attention!

It is not mandatory that this path match that under **Project Properties / Compiler / CPPCompilerPath** (see there).

Compare Tool Path

Specify the path for an external compare tool by pressing the  button and selecting the desired path.

Compare Tool Cmd Line Parameter

Here, the command lines parameter for the external Compare tool can be defined.

Alert-Pop-Up

If an error occurs, a pop-up window opens (in the lower right area of LASAL Tools) if the entry is set to TRUE.

Show Import Dialog

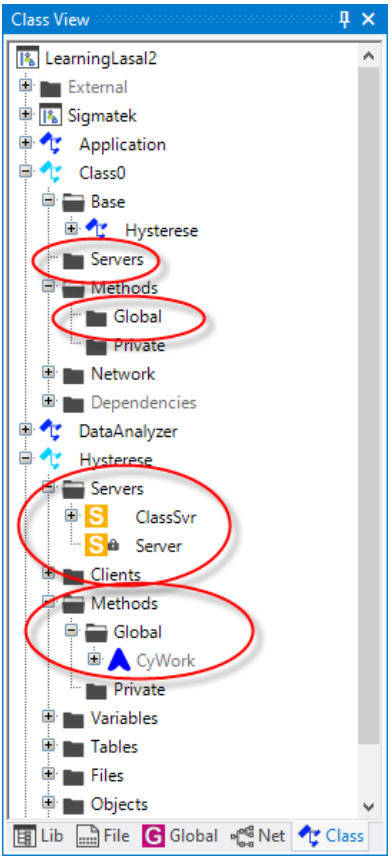
If this flag is set to **true**, this means that each time an import to a hardware class is required, the respective Import Dialog is displayed in the hardware editor for configuration (Link / Copy / Reference, selectable and deselectable components of the class). The default setting is **false**, which means if an import is required, it is run in the background automatically.

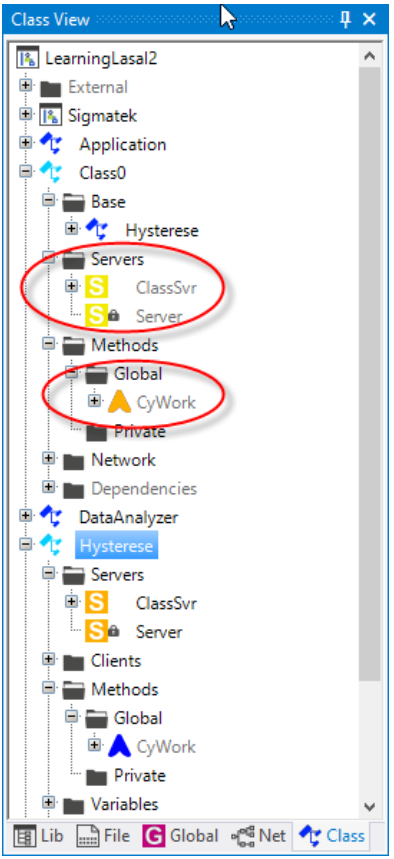
Show All Methods And Servers

If this flag is set to **true**, in the Class View window of a derived class, all servers and methods from a base that have not been overwritten are visible. If the flag is set to **false**, only the overwritten / internal methods are shown.

In the screenshots below, the **Class0** is derived from the **Hysteresis** class. In the first screenshot, the **Show All Methods And Serves** flag is set to **false**. Therefore no servers or methods are visible in **Class0**.

In the second screenshot, the situation is the same except the flag is set to **true**. It can be seen in **Class0** that both servers and the CyWork method of the base class are displayed.





Always Show SFC Detail View

If the flag is set to **true**, the Detail view opened without a double click on the respective element – both views (**Overview and Detail view**) are displayed side by side. For more information see chapter SFC.

LSE Path

Here, the default path for the **LASAL SCREEN Editor** can be entered.

Global Unit File

Here, the path and file name of the global, company-specific unit file can be entered. For further information, see Unit Configuration.

Variable Panes

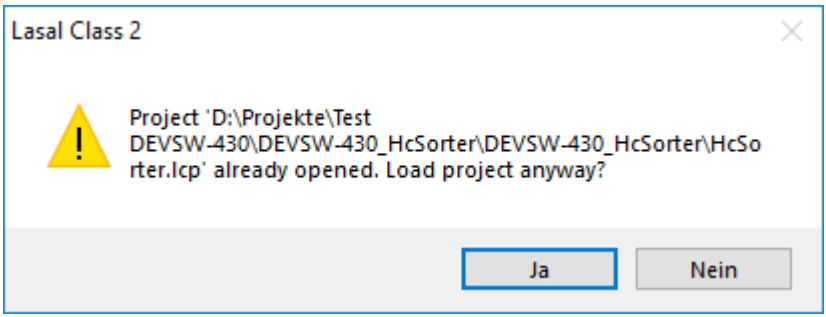
Here, whether the height setting for the individual Tabs window (e.g. Output, Debugger, DiasDriveConfig etc.) should be remembered (**true**) or not (**false**) can be specified.

Export Visualized Objects

Here, whether to export **only** the visualized **servers** (value = **false**) **or also** the visualized **objects** (value = **true**) is defined.

Load locked Projects

Here, whether or not to load locked (opened in another LASAL instance or by another user) projects can be set. If the parameter is set **false** and an attempt is made to open a second instance of a project that is already open, first a message box is displayed.



If Yes is clicked, the project is opened a second time. If no is clicked, the project is not opened, but a corresponding error message is displayed in the Output Pane.

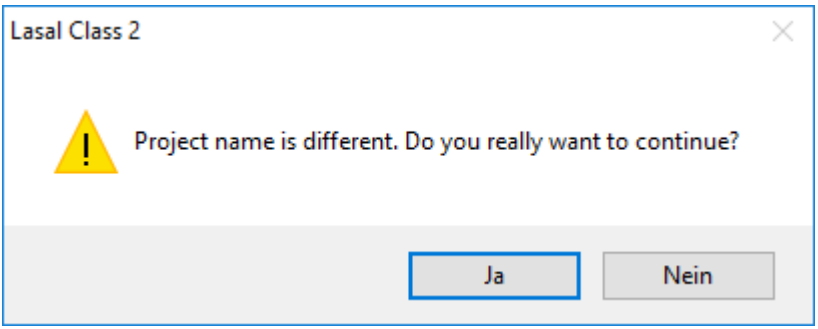


Generate Bitnames of Bitmaps

Here, whether on creation of a variable or a type of an elementary bit type (BINT, BSINT, BDINT), the individual Bits are named "Bit1", "Bit2", "Bit3" automatically (**true**) or not (**false**), can be set.

Show Download Warning

This parameter is **true** as a default and effects, that an additional warning informing about different project names opens during Download Project after the Download dialog, if the project names on the PLC and on the PC are different. This message box must be confirmed by the user to start the download.



If the parameter is **false**, the additional warning is **not** displayed (behavior like before).

Show Compiler Bug Warnings

If a Lasal CLASS 2 project is compiled with an older compiler version, and the compiler encounters an error which is already fixed with a newer compiler version, then the user is informed e.g. via the following warning:

```
[W 0158] Compiler-Bug detected, please update to compiler-version 67 or higher to fix the bu
```

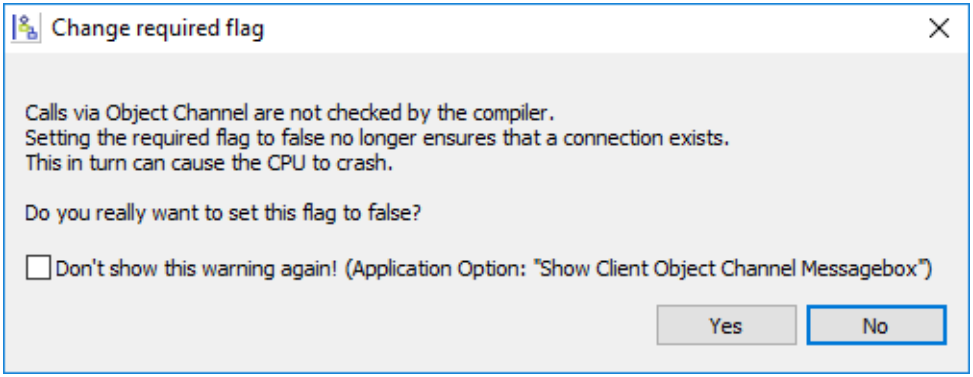
These warnings can now be disabled with the "**Show Compiler Version Warnings**" option.

Attention!

It is expressly pointed out that such warnings should not be deactivated due to the possible risks pointed out by this warning!

Show Client Object Channel Messagebox

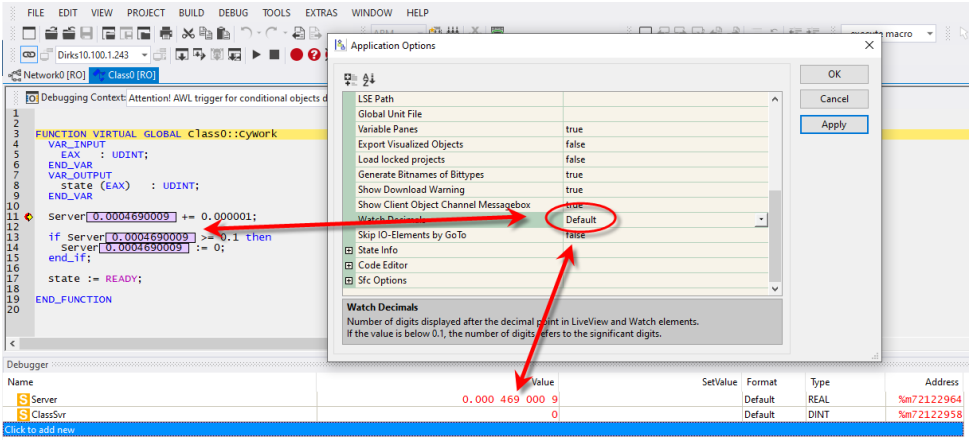
This parameter is set to **true** by default and causes the following message box to appear when a client of the type Object Channel attempts to set the value of **Required** from **true** to **false**:



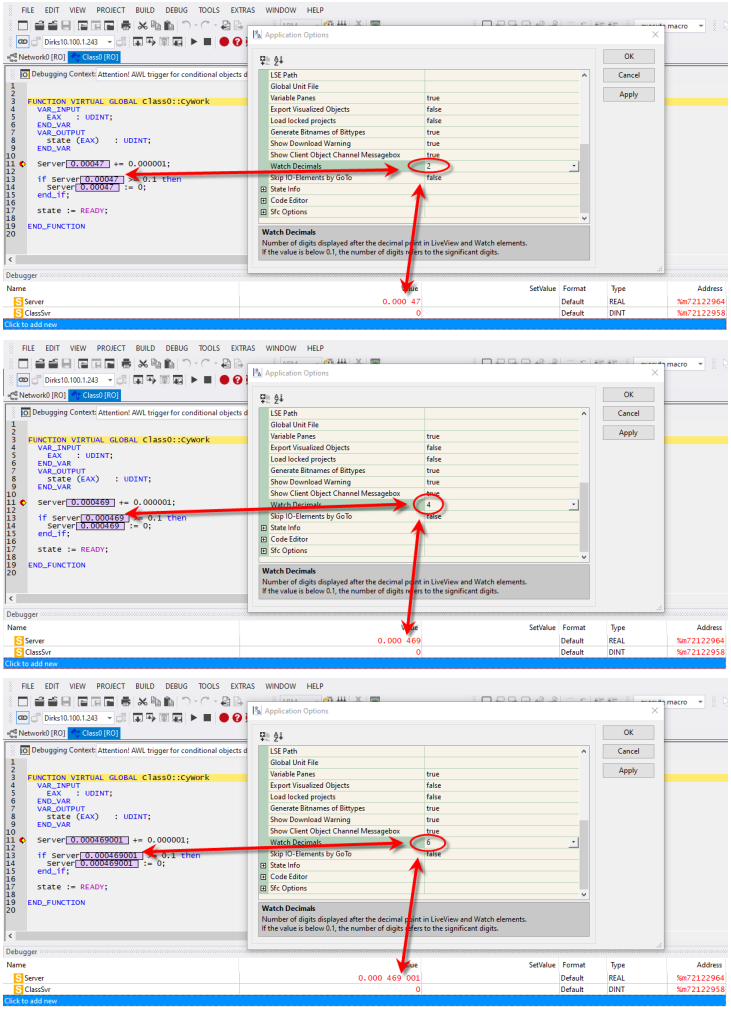
If **Don‘t show this warning again** is selected, the option **Show Client Object Channel Messagebox** is set to **false** and the next time you try to set **Required** from **true** to **false**, no more MessageBox appears.

Watch Decimals

If this parameter is **Default**, the numeric values in the Watch windows (This, Auto and Watch 1-3) and in the Live View are displayed **natively** (i.e. with the according number of decimals):



With quickly changing REAL and LREAL values with different numbers of decimals - e.g. analog inputs - this can lead to an unreadable display. To avoid this a **defined number** of **significant** decimals (**2**, **4** or **6**) can be set here. If the value to be displayed is **less than 0.1**, the fixed number of **significant** decimal places is set to 2, 4, or 6 (this can also mean more decimal places in total!):



If in such a case there are **zeros at the end of the fixed significant digits**, these are **not displayed**! See the picture above for this with **Watch Decimals = 4.** With **Watch Decimals = Default** it is obvious that with the value to be displayed (0.0004690009) there would still be a significant digit further back, but the fourth significant digit is zero,

therefore with **Watch Decimals = 4** only up to **0.000469** - i.e. up to the **3rd** significant digit - is displayed, and **not 0.0004690** !)

Skip IO-Elements by GoTo

This parameter is set to **false** by default and causes the IO elements to be **jumped to** when navigating in the object network using cursor keys. If this parameter is set to **true**, the IO elements are **skipped**. For more information on navigation in the object network, see Navigation Using Cursor Keys.

Fast Download

This parameter is set to **false** by default and causes **no LOB check to take place during project download** - if **true** is set. This means that a complete download is always performed (the **Download Changes** and **Download Changes & Run** buttons are grayed out).If the connection is slow (e.g. VPN with high latency), this parameter should be set to **true**, as this **reduces the required download time**.

Network Comment Alignment

This parameter is set to **Center** by default and specifies the default orientation of the comment texts in the object network. The other two alternatives are **Left** aligned and **Right** aligned.

Close when leaving inner networks

The inner network is closed when returning to the complex object. This parameter is set to **true** by default.

Check Single Objects

This parameter is set to **true** by default. If it is set to **false, error messages** concerning **multiple instantiation** - such as "There can only be one instance of Class-Group DiasMasterC" -> [FATAL] Please correct the previous error(s), save and reload the project" - are no longer displayed, i.e. this error message is **suppressed** when checking the hardware tree.

Connection Manager File Handling

This parameter is set to **Standard** by default. There are three options to choose from: **Standard, Merge with Warnings, Merge without Warnings**. This setting has the following effects:

Standard

- If there are differences between the content of the xml file and the content of the IO Connection Manager in the project (because e.g. the project has changed, but the corresponding IOConnectionManager.xml file has not), **there will be displayed error messages already while loading the project**
- At the next build (also at build changes!) **the project will be adapted according to the xml file** (if there are differences, the connections in the IO Connection Manager will be overwritten with the content of the xml file)

Merge with Warnings

- If there are differences between the content of the xml file and the content of the IO Connection Manager in the project (because e.g. the project has changed, but the corresponding IOConnectionManager.xml file has not), **there will be displayed warnings already while loading the project**
- At the next build (even at build changes!) **the xml file will be adapted according to the project** (if there are differences, **the connections in the xml file will be overwritten with the content of the IO Connection Manager in the project**)

Merge without Warnings

- If there are differences between the content of the xml file and the content of the IO Connection Manager in the project (because e.g. the state of the project has changed, but the corresponding IOConnectionManager.xml file has not), **there will be NO warnings displayed while loading the project**
- At the next build (even at build changes!) **the xml file will be adapted according to the project** (if there are differences, **the connections in the xml file will be overwritten with the content of the IO Connection Manager in the project**)

EmbeddedStringServerFix

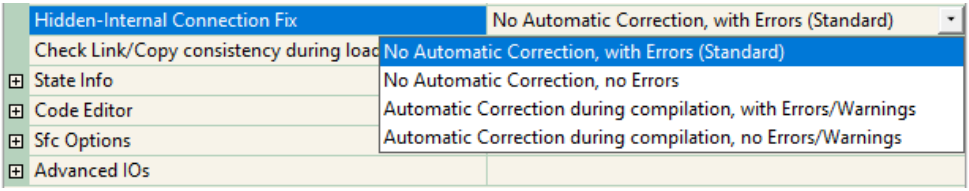
Enables the sorting fix for the CH_REDIRECT_EMB_TAB to be used in earlier compiler versions than C63.

Hidden-Internal Connection Fix

Hidden-Internal Connections are special connections in Lasal CLASS 2, which are used by complex networks to ensure the data flow from the outer (complex object) to the inner layer (embedded object). File mergers or manual changes can cause these connections to be faulty.

Attention!

Faulty Hidden-Internal Connections **should be fixed by all means**, because it can lead to incorrect behavior (wrong values or crash) of the application!



This option is set by default to "No Automatic Correction, with Errors (Standard)" and serves to make the user aware of incorrectly entered hidden internal connections resp. to correct them automatically if necessary.

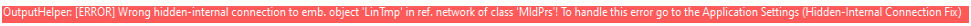
Attention:

It is strongly recommended **not to disable** these warnings/errors!

There are four possible values for **Hidden-Internal Connection Fix**:

- **No Automatic Correction, with Errors (Standard)**

This is the default setting. Error messages are output (see above) as soon as LasalClass2 encounters incorrect internal connections (**also applies to scripting!**). These checks are performed when loading and building the project. An automatic correction is **not** performed. However, the user is informed about this "application option" in the error message.

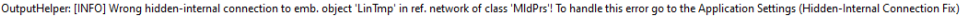


Error message for the option setting "No Automatic Correction, with Errors (Default)"

- **No Automatic Correction, no Errors**

No automatic correction, no error messages. (**not even during scripting!**)

So the user does **not notice in any way** that there are errors hidden in his application.



Information message for the option setting "No Automatic Correction, no Errors"

- **Automatic Correction with Errors/Warnings**

The faulty connections are **reported by LasalClass2 during loading (error message)** and **automatically corrected** during compilation if possible.

The user is **informed about the changed connections via warning**.

OutputHelper: [ERROR] Wrong hidden-internal connection to emb. object 'LinTmp' in ref. network of class 'MIdPrs'! To fix this error please compile the project

Error message for the option setting "Automatic Correction with Errors/Warnings"

- **Automatic Correction without Errors/Warnings**

The faulty connections are **not reported by LasalClass2** during loading and are **automatically corrected** during compilation if possible.

In this case the user is **not informed about the changed connections via warning**.

OutputHelper: [INFO] Wrong hidden-internal connection to emb. object 'LinTmp' in ref. network of class 'MIdPrs'! To fix this error please compile the project

Error message for the option setting "Automatic Correction without Errors/Warnings"

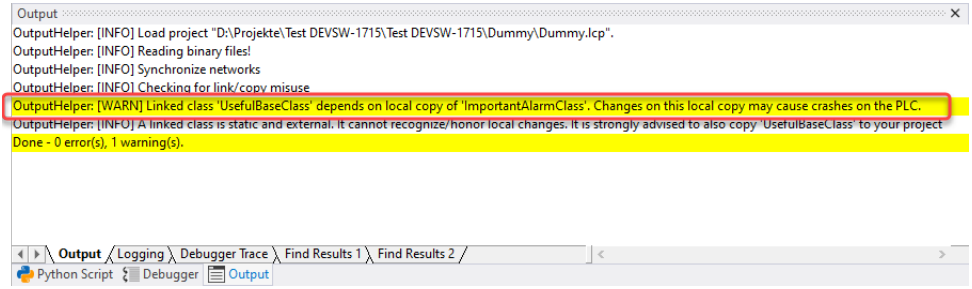


Attention:

However, if a connection **cannot be changed** for some reason, **an error will still be issued** in this particular case!

Check Link/Copy consistency during load project

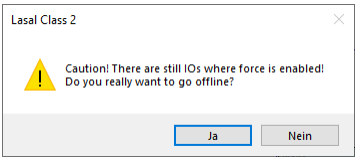
If this parameter is set to **true**, it allows to check during the loading of the project whether the mixing of classes linked and copied from libraries could cause problems due to dependencies of various kinds. If this is the case, appropriate warnings are displayed:



If you set the parameter to **false**, this check will be skipped when loading the project. For more information on the problem of combined use of linked and copied classes see the chapter "mixing copy and link classes"

Show Force Enable Warning Dialog

If this parameter is set to **true** (default), a corresponding warning is issued when trying to go offline while Force Enable is set in an Advanced IO object in the object network:



With the setting **false** this warning message can be switched off.

There are four more sub menus: State Info, Code Editor, Sfc Options and Advanced IOs.

⊞ State Info	
⊞ Code Editor	
⊞ Sfc Options	
⊞ Advanced IOs	

2.1.11.31 State Info

⊟ State Info	
Show Connected Servers	false
Mark visualized flag in tree	true
Mark visualized flag in network	true
Mark writeprotected and retentive servers in netwo	true
Mark world flag in tree	true
Mark world flag in network	true

Show Connected Servers

If this entry is set to TRUE, all servers connected to clients are marked green.

Mark visualized flag in tree

If the entry is set to **true**, all servers where the visualize flag = **true** in the project tree are identified with an eye.

Mark visualized flag in tree

If the entry is set to **true**, all objects where the visualize flag = **true** in the project tree are identified in the network with an eye.

Mark writeprotected and retentive servers in network

If the entry is set to **true**, all write-protected and retentive servers in the network are labeled. Write protection is indicated by a padlock 

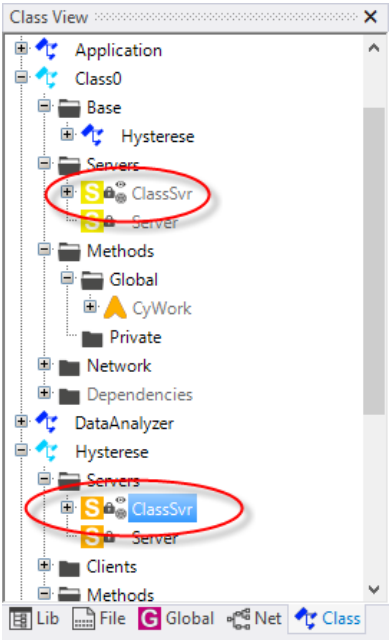
, retentive servers are identified by memory location (Sram 

or file 

).

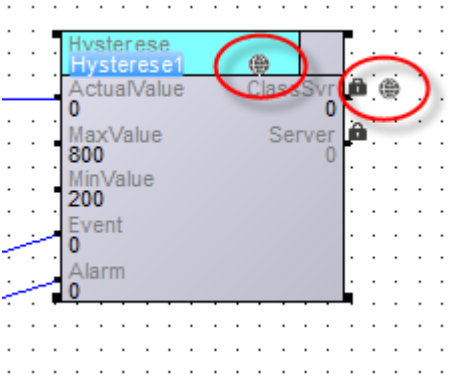
Mark world flag in tree

Setting it to **true**, all objects and servers in the tree are identified which have set the World flag:



Mark world flag in network

Setting it to **true**, all objects and servers in the network are identified which have set the World flag:



2.1.11.32 Code Editor

⊟ Code Editor	
AutoCompletion	true
Keyword Input Mode	Default
⊟ Folding Options	
Enable Folding	false
Fold functions	true
Fold loops	true
Fold conditions	false
Fold comments	true
Fold preprocessor	false
Tab Size	2
Fontname	Lucida Console
Fontheight	10
Constructs Highlighting	true
SmartHighlighting	true
Automatic Line Ending Fix	false
⊟ Marker Options	
Mark Search Color	255; 0; 255
Mark Text Color	155; 255; 255
⊟ Color Options	
⊞ ST Options	
⊞ CPP Options	
⊞ Python Options	
⊞ XML Options	

AutoCompletion

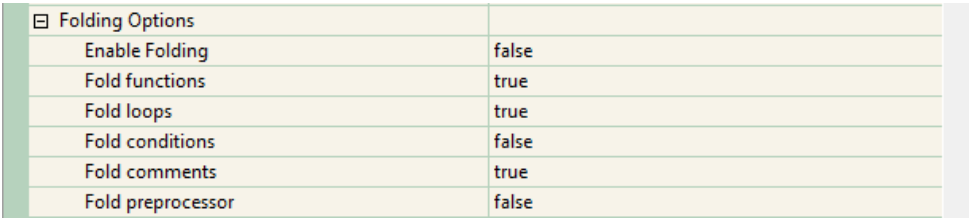
Here, the automatic code extension can be enabled or disabled.

Keyword Input Mode

Here, how the key words are should be entered can be set in the automatic code completion. **Upper** – keyword always always upper case; **Lower** – keywords always lower case; **Default** – upper case keywords capital, mixed or lower case, small letters.

Folding Options

Here, the parameters for expanding and folding can be set in the code editor.



Folding Options	
Enable Folding	false
Fold functions	true
Fold loops	true
Fold conditions	false
Fold comments	true
Fold preprocessor	false

Enable Folding	**true** generally enables folding and expanding with a click on the plus or minus characters at the start of the respective blocks.
Fold Functions	**true** enables the folding of functions.
Fold loops	**true** enables the folding of loops.
Fold Conditions	**true** enables the folding of If and Case blocks.
Fold Comments	**true** allows the folding of commentary.
Fold preprocessor	**true** enables the folding of preprocessor commands such as `#pragma`, `#ifdef` or `#define`

Tab Size

Here, the number of spaces can be entered that are inserted when the tabulator button is pressed.

Fontname

Here, the font can be set.

Fontheight

Here, the point size for the fonts can be set.

Construct Highlightning

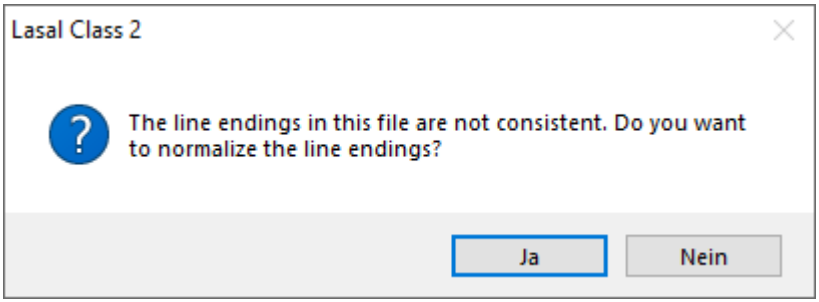
If this parameter is **true**, the keywords are highlighted if the cursor is in a **loop** or an **if-instruction**.

SmartHighlightning

If this parameter is **true**, when double-clicking on a word, it and any other instances will be marked in the opened editor. However, this is only a momentary mark contrary to the marking function in the menu **View => Mark**. Clicking somewhere else, the mark will disappear.

Automatic Line Ending Fix

In LASAL CLASS 2, there is a mechanism that corrects incorrect line endings that occasionally occur in older projects. With this setting you can determine whether the user should be asked before (parameter is set to **false**) or whether the correction should be done automatically without asking (parameter is set to **true**). If the parameter is set to **false** and such a project is loaded, the following dialog is displayed:



If **Yes** is selected, the correction is applied and the project is marked with a star as changed. If saved this way, the error will never occur again when loading the project.

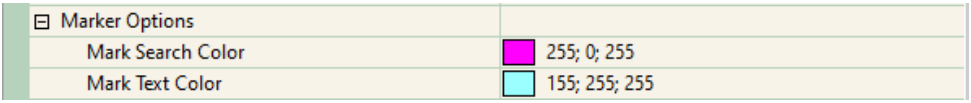
If **No** is selected, the project remains untouched - but during compilation the error message

Compiler: [ERROR] E 0016 "D:\Projekte\Test DEVSW-1442\Firma Lumen\Compiler_Crash\Compiler_Crash_L22_Eis_10\Class\Programme\Programme.st"(800) Not expected input '\CR'

is output.

Marker Options

Here the colors of the markings are set.

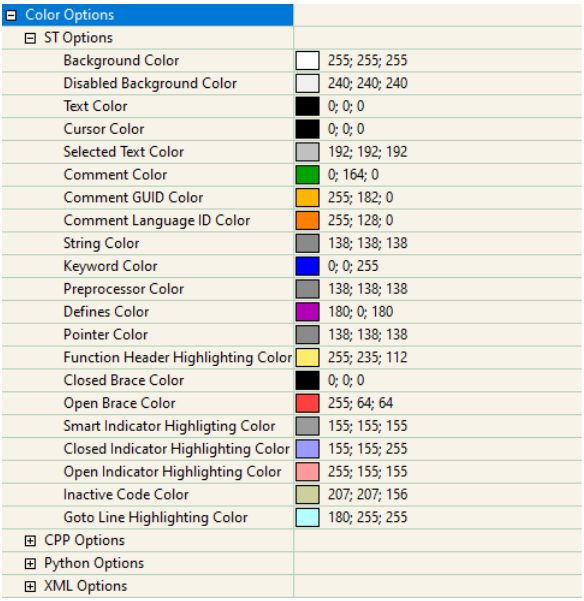


Mark Search Color	Here, the color of the markings, which have been made from the Find dialog via the "**Mark All**" checkbox, can be set.
Mark Text Color	Here, the color of the markings, which have been made via the menu "**View => Mark**", can be set.

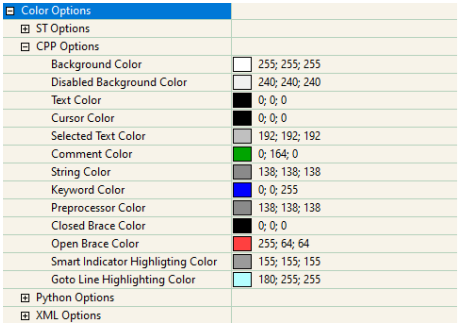
Color Options

Here the colors for ST, CPP, Python and XML files are managed.

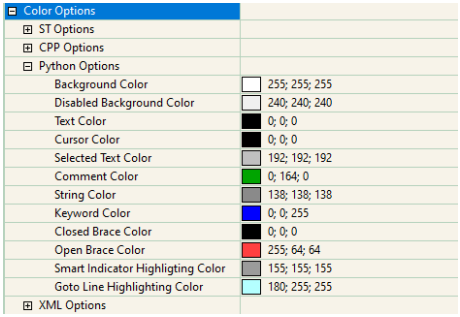
ST Options



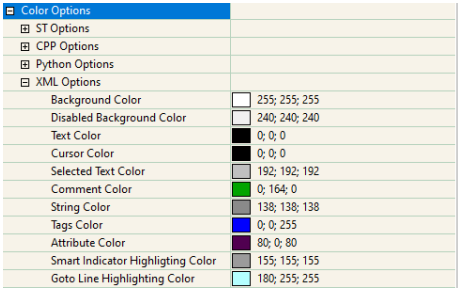
CPP Options



Python Options



XML Options



The Reset Editor Colors menu item in the Edit menu resets all colors to their default settings.

2.1.11.33 Sfc Options

Here the SFC Editor settings are made.

Sfc Options	
Use Sfc Autocompile	false
Always Show SFC Detail View	true
SFC editor raster width	130

Use Sfc Autocompile	After the detail view is exited, the Sfc project is automatically compiled if **true**. For more information see the chapter SFC.
Always Show SFC Detail	The detail view when **true**, is permanently shown next to the SFC overview window.
SFC editor raster width	Here, the width of the SFC editor column is set. The default value is 130.

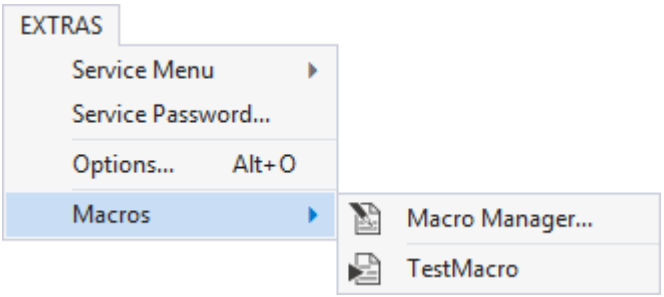
2.1.11.34 Advanced IOs

Advanced IOs	
PLC Path	C:\Isldata\
AdvancedIO	Default Advanced IO
Simulation	iPhysics Simulation
Manager	Default Advanced IO Manager

PLC Path	Path of the Advanced IO configuration file **on the PLC**.
Advanced IO	Here you can make the settings for the **default Advanced IO class**.
Simulation	Here you can make the settings for the **default simulation class**.
Manager	Here you can make the settings for the **default Advanced IO manager class**.

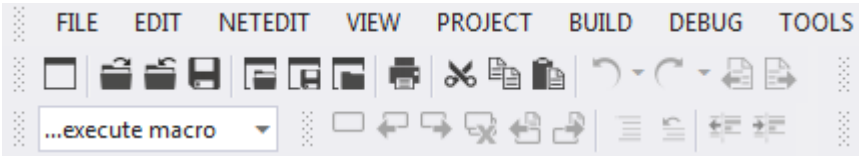
For more detailed information see the chapter Virtual Commissioning Manager.

2.1.11.4 Macros

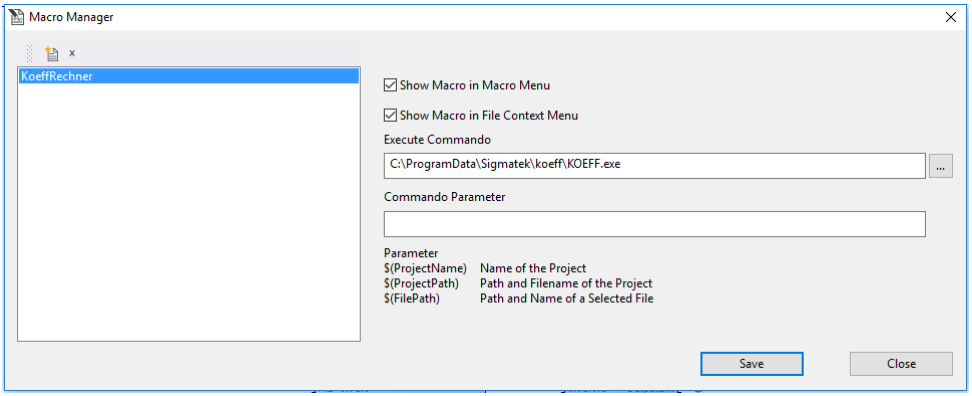


Macro Manager

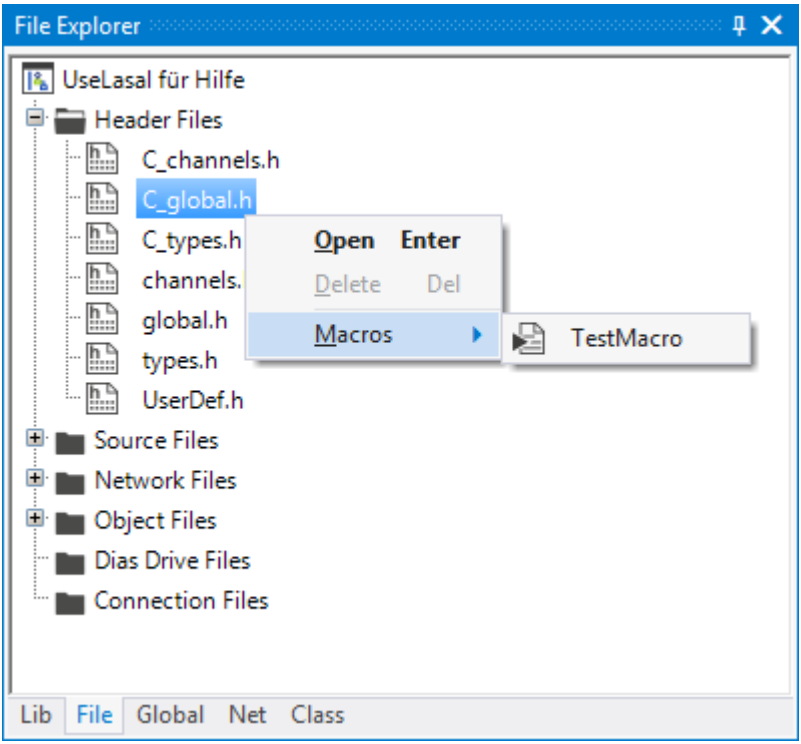
In the toolbar, the **execute macro** field can be found.

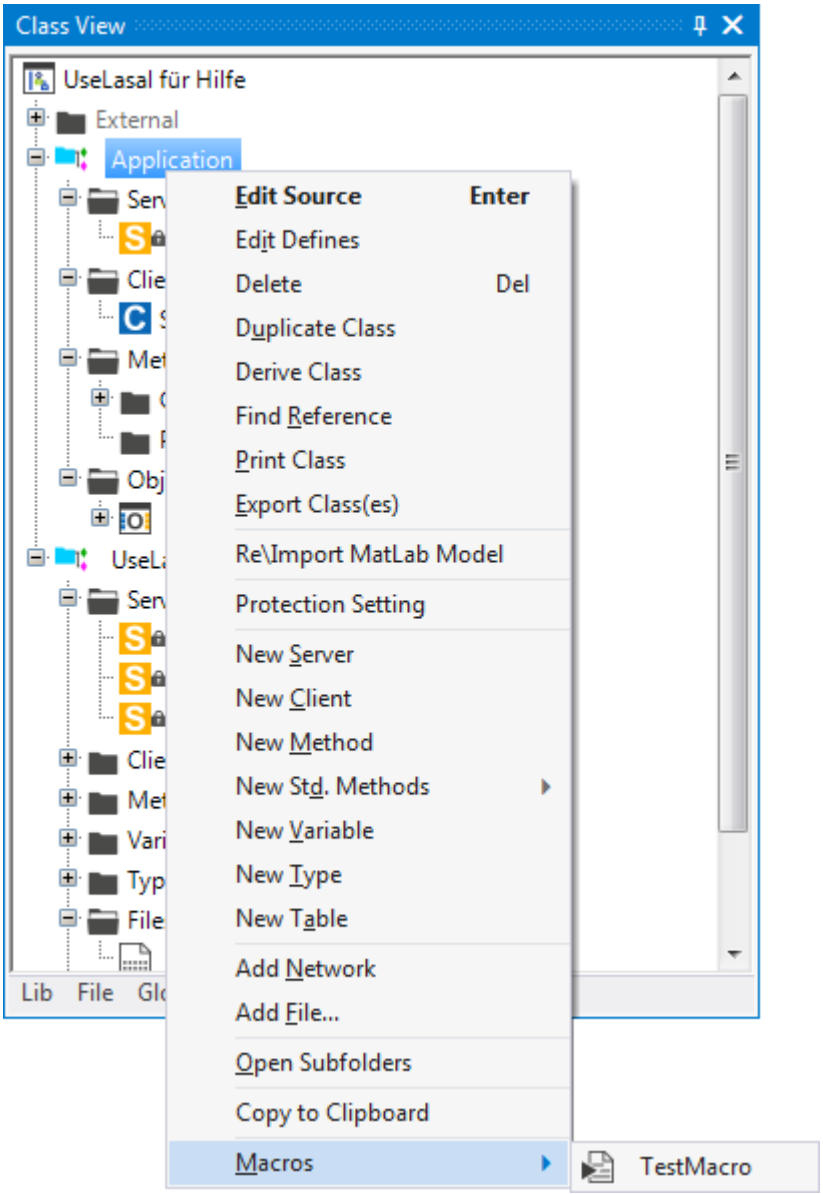


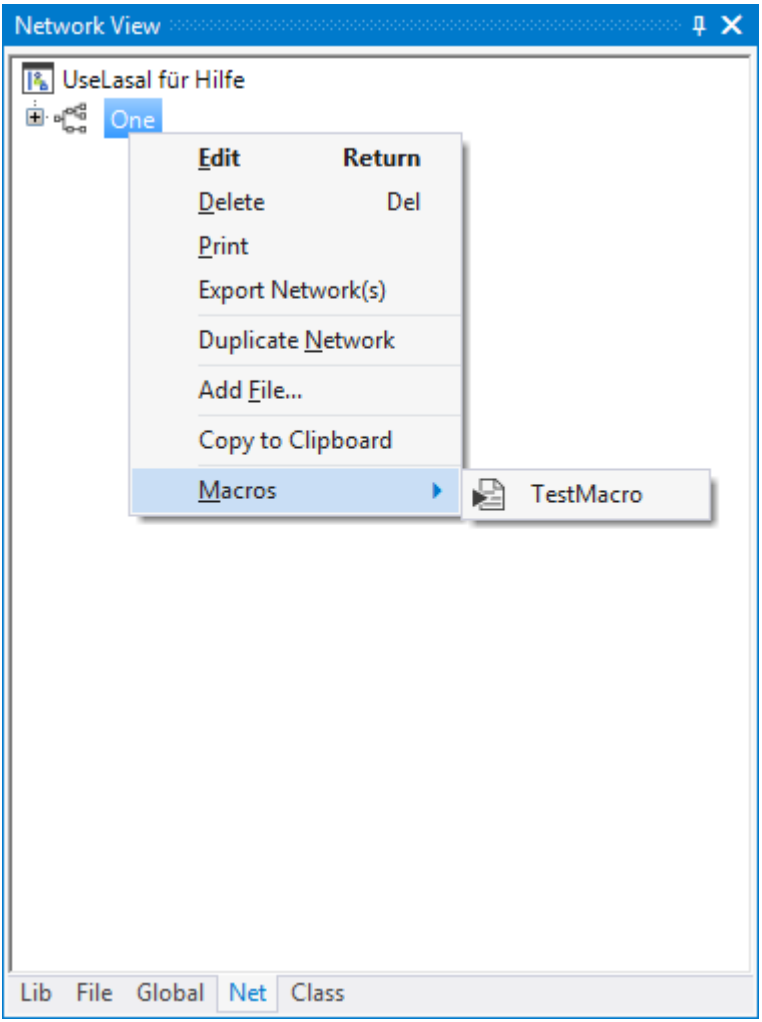
Using a dropdown menu, external programs, which were previously linked through the Macro Manager (in menu Extras/Macros/Macro Manager), can be easily run through the LASAL Tool.



With Macro Manager function, external programs can be easily linked through the LASAL Tool and run without having to exit LASAL. A Macro can also be started from the context menu of the File tree, Network tree or Classes tree:


File Explorer
UseLasal für Hilfe
Header Files
C_channels.h
C_global.h
C_types.h
channels.
global.h
types.h
UserDef.h
Source Files
Network Files
Object Files
Dias Drive Files
Connection Files
Open Enter
Delete Del
Macros
TestMacro
Lib
File
Global
Net
Class

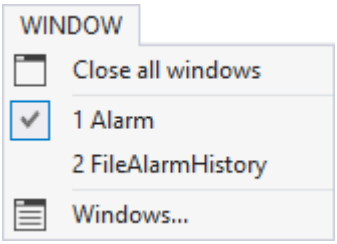

Class View
UseLasal für Hilfe
External
Application
Edit Source Enter
Edit Defines
Delete Del
Duplicate Class
Derive Class
Find Reference
Print Class
Export Class(es)
Re\Import MatLab Model
Protection Setting
New Server
New Client
New Method
New Std. Methods
New Variable
New Type
New Table
Add Network
Add File...
Open Subfolders
Copy to Clipboard
Macros
TestMacro
Lib
File

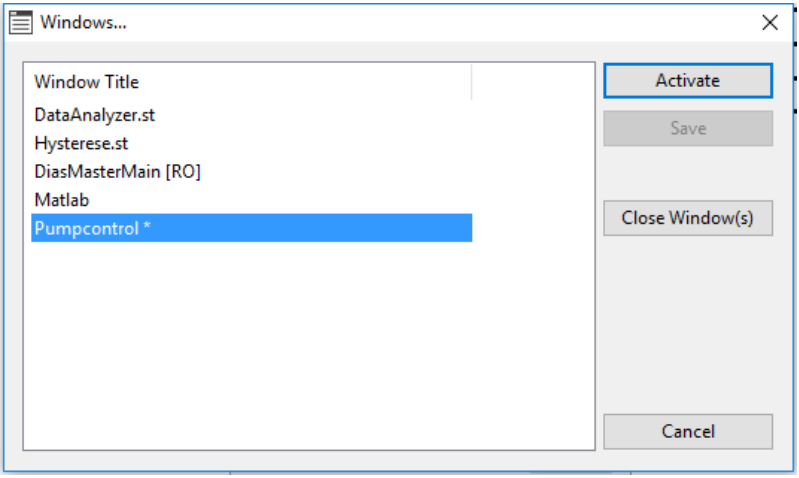


Test Macro (as example)

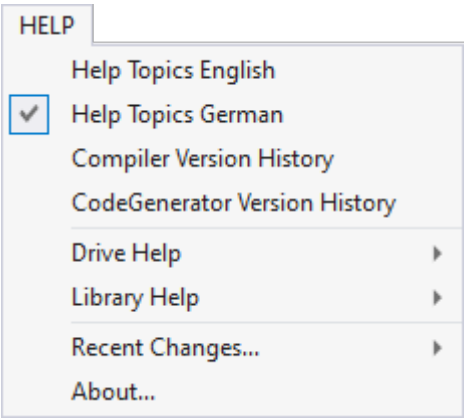
Here, existing Macros are listed and can be started.

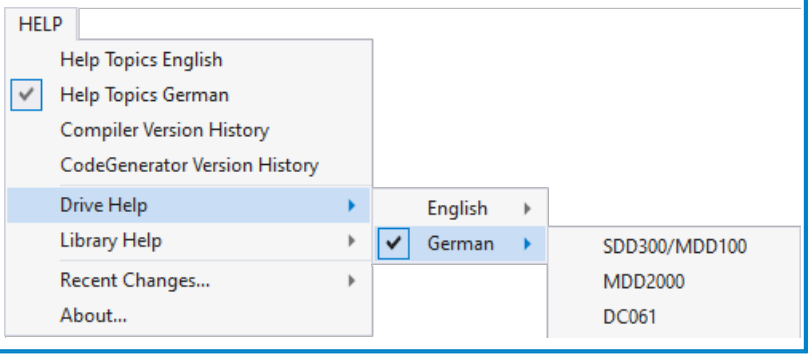
2.1.12 Window Menu

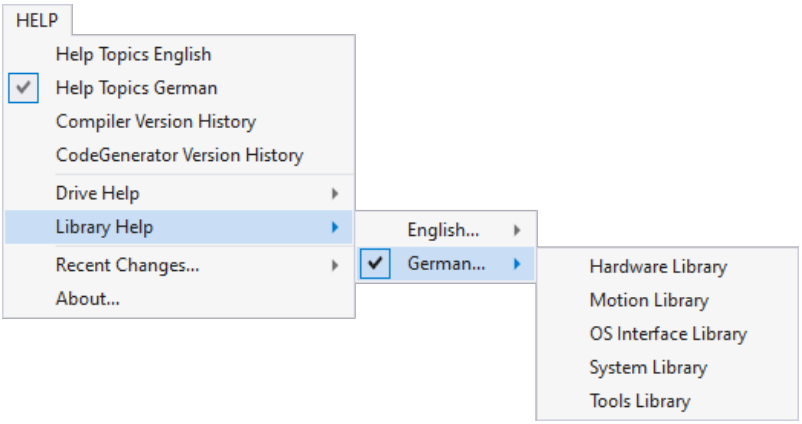


Close all windows	Closes all open windows. All opened windows (here: Class0) that can be activated or deactivated with the check are thereby displayed (Hysteresis is active, Pumpcontrol is inactive).
1 Class0 / 2 ... etc.	Change to the respective window opened in the LASAL Editor.
Windows	Open the window select dialog in which all open windows are listed. If a window is selected with the left mouse button so that it has a blue background, it can be activated (**Activate** button) or closed (**Close Window(s)** button). If several windows are selected, they are closed with the **Close Window(s)** button. 

2.1.13 Help Menu

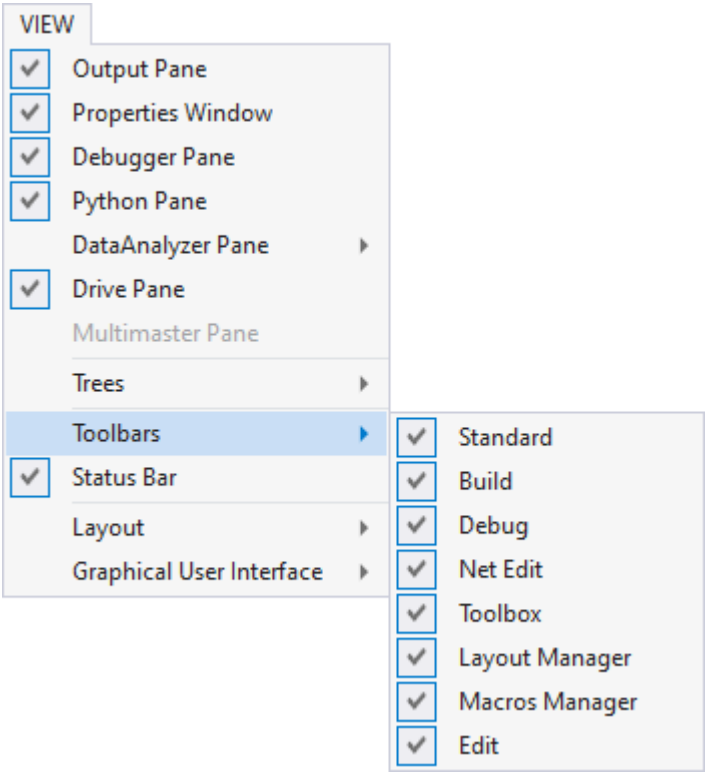


Help Topics English	Starts the English LASAL CLASS 2 – Help.
Help Topics German	Starts the German LASAL CLASS 2 – Help.
Compiler Version History	Displays the Compiler Version History dialog window. The difference between the individual compiler versions is shown.
CodeGenerator Version History	Displays the CodeGenerator Version History dialog window. The difference between the individual CodeGenerator versions is shown.
Drive Help (English, German)	Opens the specific help for Motion / Drive 

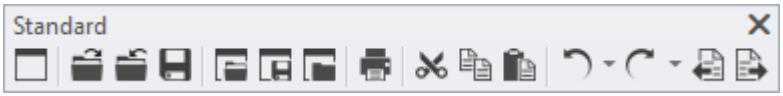
Library Help (English, German)	Starts the library-specific help, if available. 
Recent Changes...	Opens the documents with the last changes in LASAL CLASS 2.
About...	Shows an information window on the LASAL SafetyDesigner containing the data and version.

2.2 Tool Symbol Bars (Overview)

This overview of all toolbars available in LASAL CLASS 2 should make navigating through the LASAL CLASS 2 Help easier. There are 8 toolbars that can be selected under the menu item **View, Toolbars**.

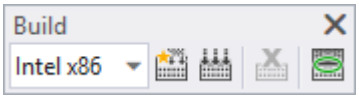


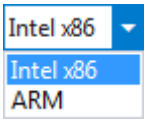
2.2.1 Standard Toolbar



	Create a new project		Cut
	Open a file		Copy
	Close a file		Paste
	Save a file		Undo
	Open a project		Redo
	Save a project		Navigate backward
	Close the actual project		Navigate forward
	Print the active element		

2.2.2 Build Toolbar





Here, the architecture of the target is selected (Intel x86 or ARM).

	Compile the changes in a project		Interrupt the compiler process
	Compile the entire project		Start the link process

2.2.3 Debug Toolbar



	Online / Offline		Opens the breakpoint condition window
	Chance online settings		Delete all breakpoints
IP Lars Local	Selected online connection; quick selection.		Activates/deactivates the AWL trigger (also see chapter What debugging means)
	Sets the online connection for the actual project.		Activates/deactivates the fixed AWL trigger (see also: chapter Debugging Programs)
	Download Project		Show current instruction (Goto Instruction Pointer)
	Several functions at once: Compiles the project changes and goes online with the control, downloads and starts the project:		Activates/deactivates LiveView (Display of the values of all variables currently visible in the Code-Editor and the class elements in online operation, see also chapter The Views Auto and LiveView)
	Download Module on the Fly. Is only available, if the the CPU status is RUN_RAM or RUN_ROM.		Switch on and off Auto Watch (display of the variables of the valid function at the cursor position in an extra Auto-Pane under Debugger)
	Save Project on PLC		Step Into
	Start / Reset		Step Over
	Reset		Step Out
	Toogle Breakpoint		Set current instruction (Set Instruction Pointer)

2.2.4 Net Edit Toolbar



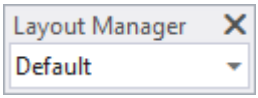
	Select pointer		Zoom
	Hand tool		Zoom all
	Insert text		Zoom area

2.2.5 Toolbox Toolbar



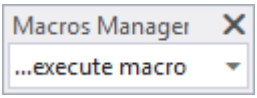
	Open the Data Analyzer		Open the DiasDrive user interface (also see chapter DiasDrive)
	Open the Interpreter user inteface		Open the PLC diagnostic window
	Opens the hardware editor (function currently not available)		Opens the graphical Hardware Editor
	Opens the Connection Manager		Opens the Task Configurator

2.2.6 Layout Manager Toolbar



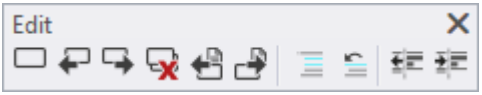
Here, a stored layout can be selected.

2.2.7 Macros Manager Toolbar



Here, a stored Macro can be run.

2.2.8 Edit Toolbar



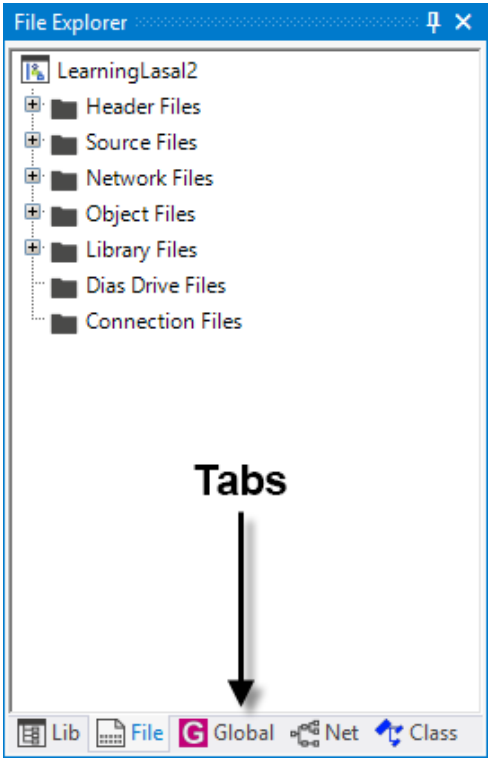
	Sets / clears the selections at the cursor position		Jumps to the previous selection
	Jumps to the next selection		Clears all selections
	Previous mark in this file		Next mark in this file
	Comments the selected area		Removes the comment from the selected area
	Shifts the selected area by one tab stop (to the left)		Shifts the selected area by one tab stop (to the right)

2.3 Project Tree

The project tree is on the right side of the LASAL CLASS 2 user interface and contains 5 areas with various contents (**Files**, **Globals**, **Classes**, **Libraries**, **Networks**). Click on User Interface of the Design Environment To see the position of the project tree in the LASAL CLASS 2 user interface. As shown in the graphic below, each area can be selected through tabs.

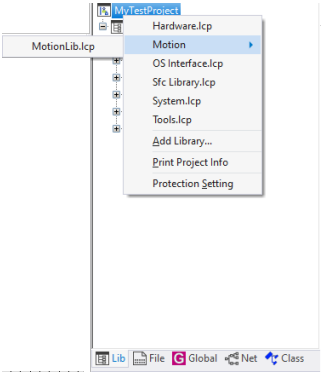
Advantages of a project tree:

- All classes belonging to the project are listed.
- All files required by the project are given.
- New files, global types, classes, libraries and networks can be created or deleted with a mouse click.



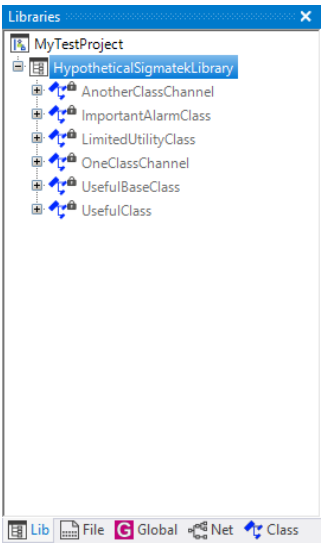
2.3.1 Libraries Tree

The first register tab **Lib** opens the **Libraries window**.

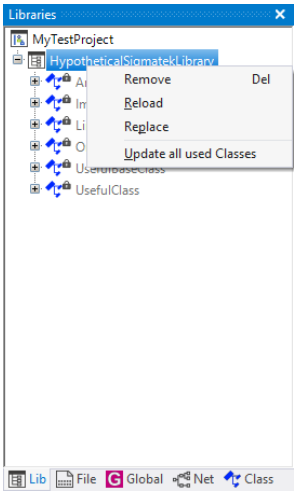


To add a library, the context menu is called with a right click on the project name and a library selected.

The selected library (here: **HypotheticalSigmatekLibrary**) is displayed in the **Libraries window**. A click on the + next to the library, the classes are listed.



The context menu is now activated with a right click on the library name.



Remove

With Remove, the imported library is deleted.

Reload

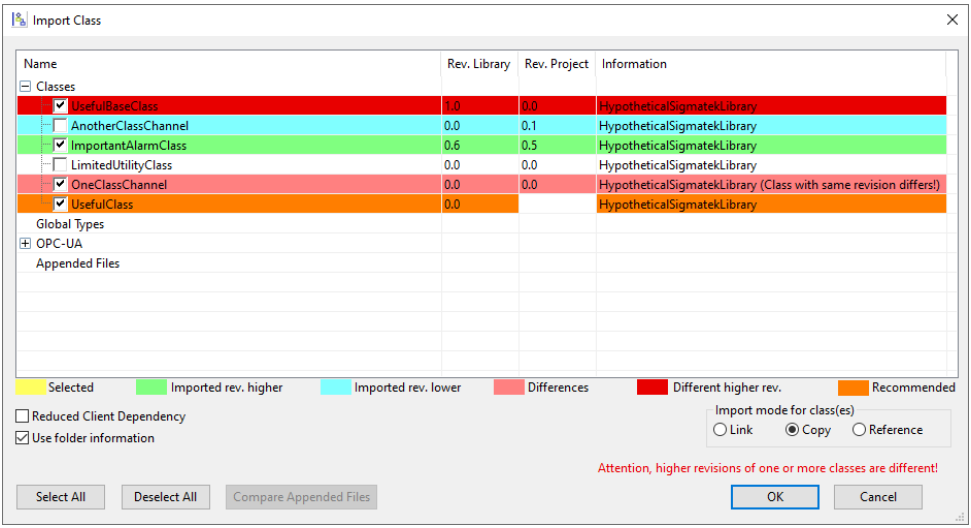
With Reload, the library is reloaded.

Replace

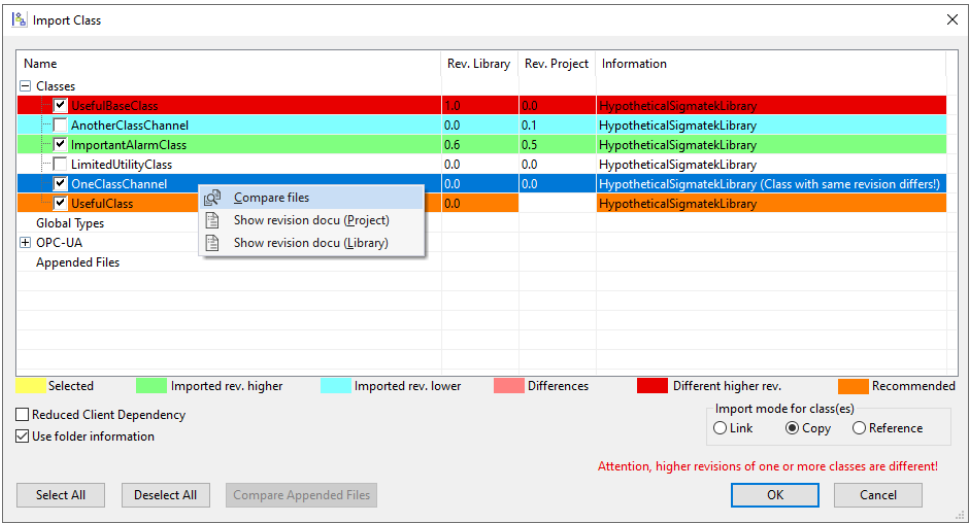
Here, the library can be replaced with another. The desired library is selected in the displayed File Open dialog.

Update all used Classes

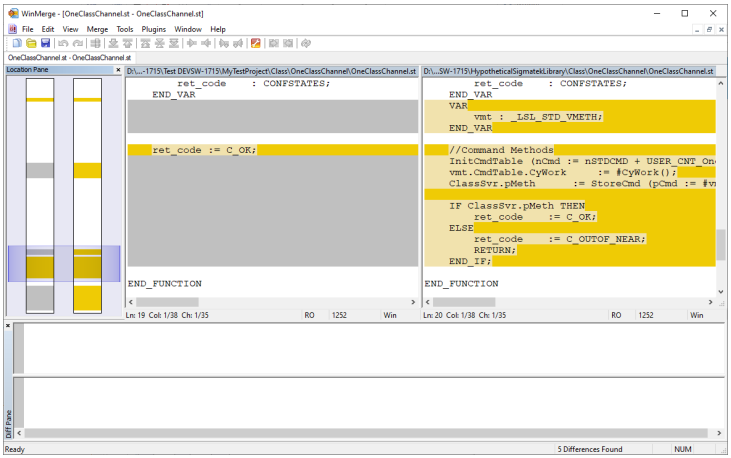
All classes in the current project are updated with the classes from this library. Here, an import class dialog opens, which contains all classes, global types and and added files from the selected library – similar to the ImportClass dialog when adding modules to the hardware editor. There are three options by which classes files and global types can be imported: copying, linking and referencing. The difference of overlapping classes / types are highlighted:



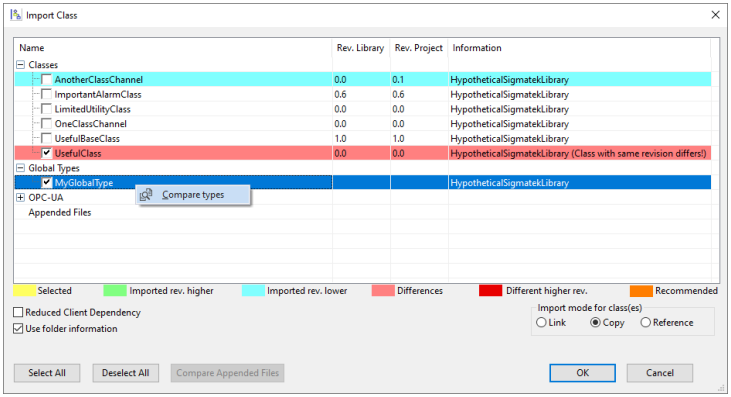
After a right-click on an entry, the context menu is shown in which for classes, contains the menu items “Compare Files”, Show revision docu (Project)” and “Show revision docu (Library)”. For a global type, the context menu contains “Compare Type” and for appended files, “Compare files“.

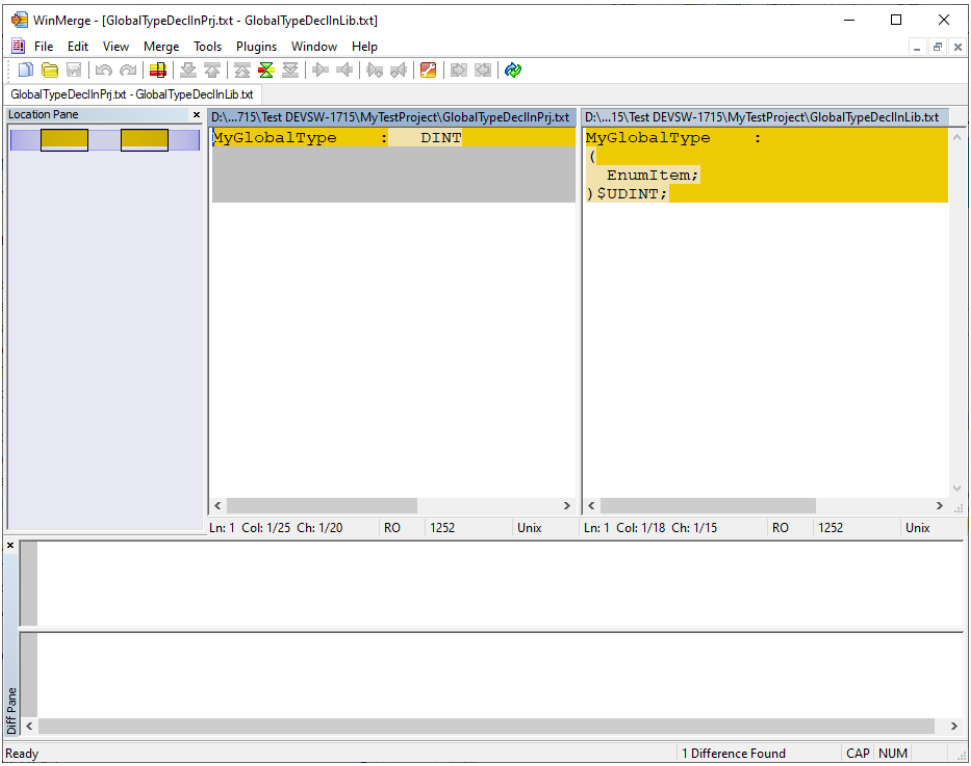


If “Compare files” is selected, WinMerge is started with the two files (Project vs. Libaray) and the result of the comparison is displayed immediately. If differences are found, they are then marked.

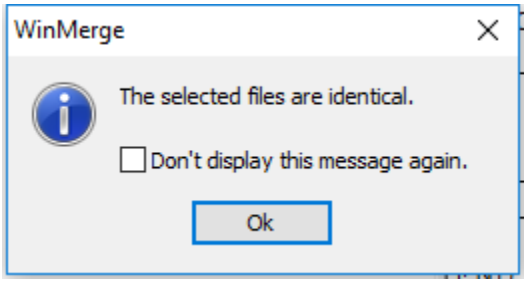


The comparison is also possible with global types:

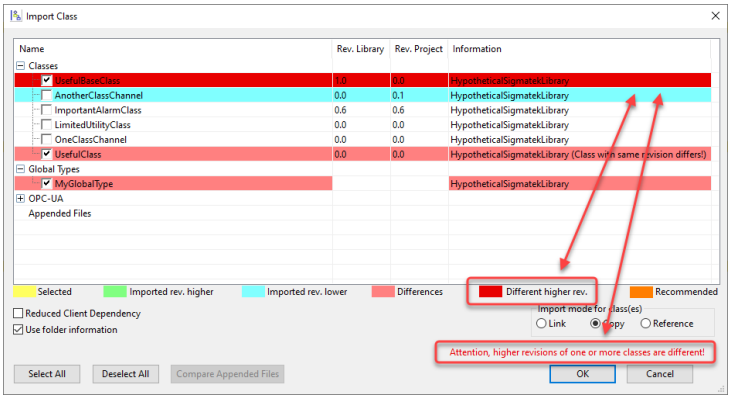




If the files are identical, this is indicated in a small dialog window.

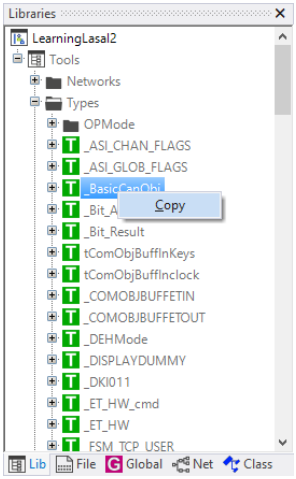


If there are differences in the pre-decimal places of the revision number between the existing class and the class to be imported, this is indicated by the dark red marking of the class and additional text over the OK button:

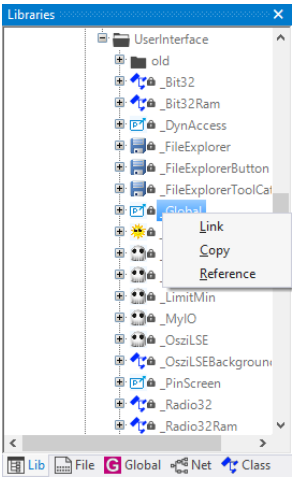


If the “Update all used Classes” dialog is closed, the temporarily files created for WinMerge are deleted.

In the **Types** folder, a list of global types is found. To import one of these types, Copy must be selected from the context menu.

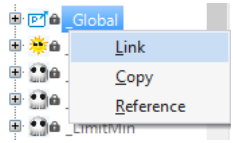


To add a class to the project, open the context menu with a right click.

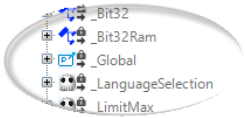


Link

The **Link** instruction is only available for classes. If the instruction is executed, a reference to the library and the linked class entered in the current project. The physical memory location of the class remains in the original project. The affected class is therefore continuously updated when a project containing the linked class is opened. When classes are linked, the corresponding library cannot be removed from the project.

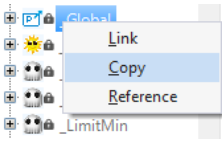


A linked class is identified in the **Class View window** with a black arrow pointing to the right:

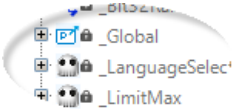


Copy

With the **Copy** instruction, the element is copied with all required files (dependencies) to the current project. With changes however, these classes are not updated.

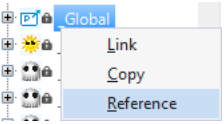


If a class is copied, there is no file next to the class in the Class View window:

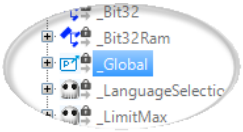


Reference

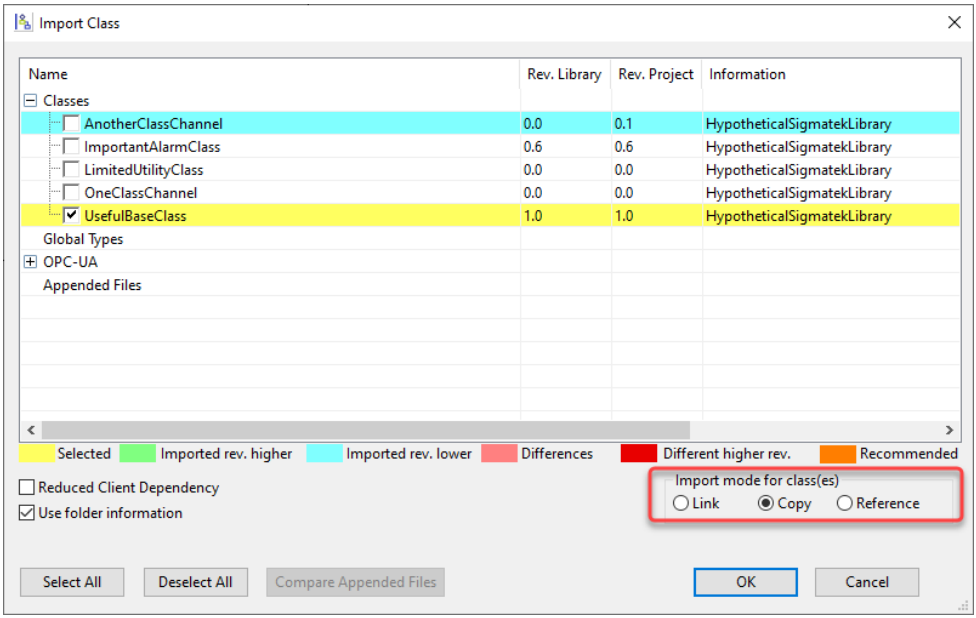
Similar to **Copy**, however, LASAL CLASS 2 remembers the path from where the class was imported. This means that the class can later be updated on command.



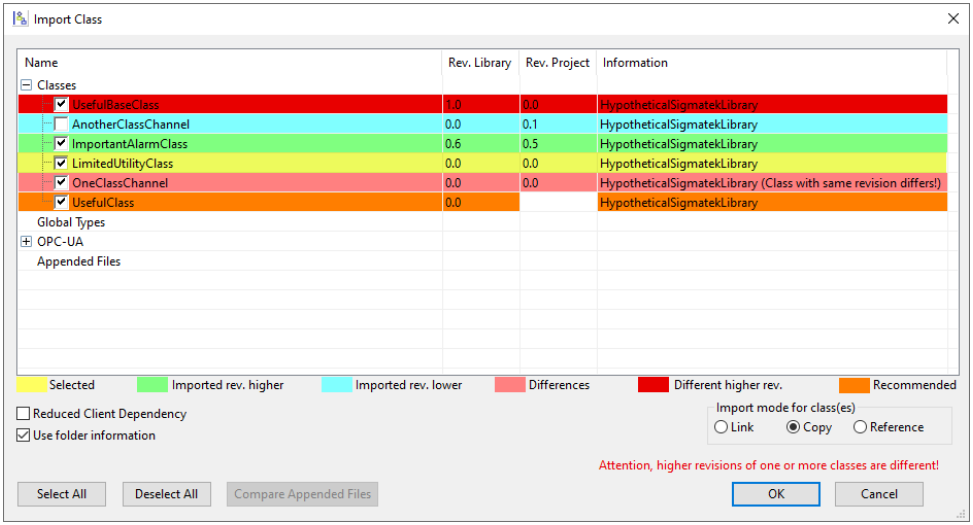
A referenced class is identified in the **Class View window** with a gray arrow pointing to the right:



If one of these three import functions is selected and "**Show Import Dialog**" is set to **true** under Tools / Options, the import dialog opens again. The type of import can also be subsequently changed in this dialog:

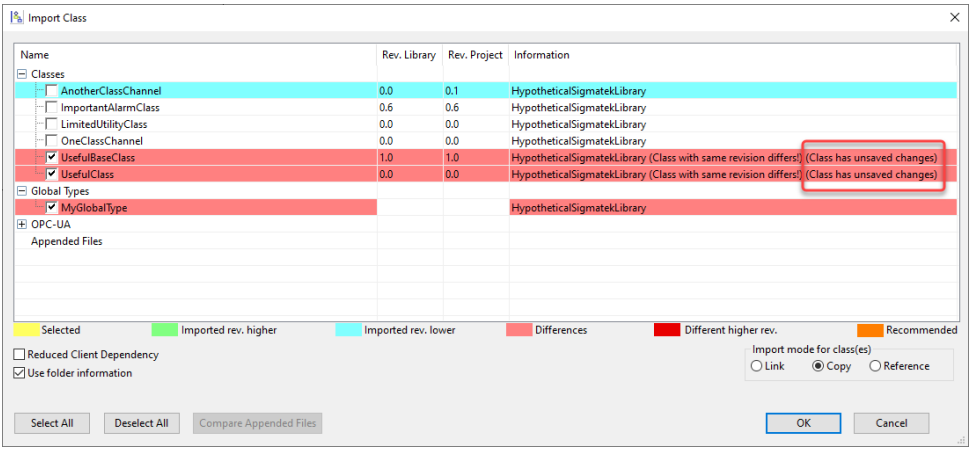


If classes (and/or files) already exist in the project, they can be deselected. Colored markers (see image) indicate those classes that are different:

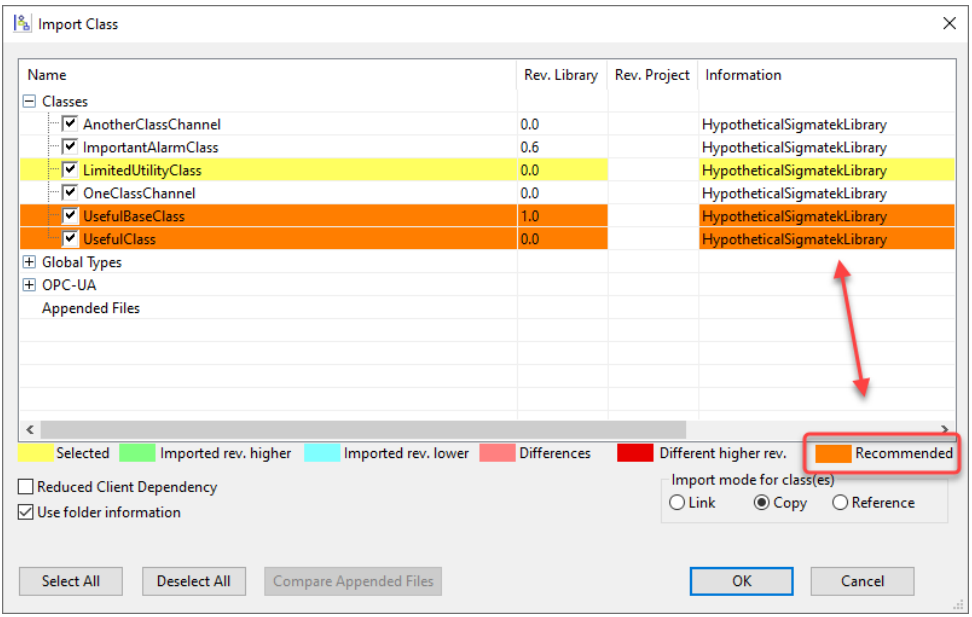


Selected (yellow)	Class was selected for import.
Imported rev. higher (green)	Class in the library has a higher revision number.
Imported rev. lower (blue)	Class in the actual project has a higher revision number (in this case, the class deselected by default).
Differences (red)	Class in the actual project and the library has the same revision, but are different. Class in the actual project and the library has the same revision, but are different. If this difference is because the class in the actual project has changed but has not yet been saved, a corresponding message (**Class has unsaved changes**) is shown in the info column. To see it, the window must be expanded.
Different higher rev. (dark red)	The revision number **before the decimal point** is different in the project or library.
Recommenden (orange)	All classes that could be affected by a local change to the class selected for import (i.e. all superior classes).

The "**Class has unsaved changes**" messages:



Superior classes are colored orange:





Attention!

With the imported classes, the sizes of individual objects as well as the vertices are **not** assumed!

Reduced Client Dependency

With reduced client dependency, a selection can be made as to whether or not classes that have a dependency to the imported class should be included in the importation (dependency e.g. through an object channel).

If this function is activated, only classes are imported that are directly linked to the selected class (no more classes than the 2nd dependency).



Attention!

Reduced Client Dependency should only be used for project structures with partial projects, since by setting this option the project is possibly not operation-capable.

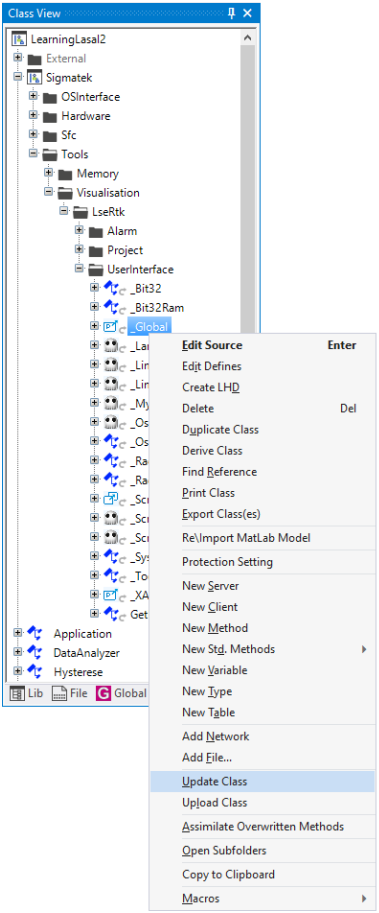
User folder information

With this selection, the user can specify whether project folders should also be imported.

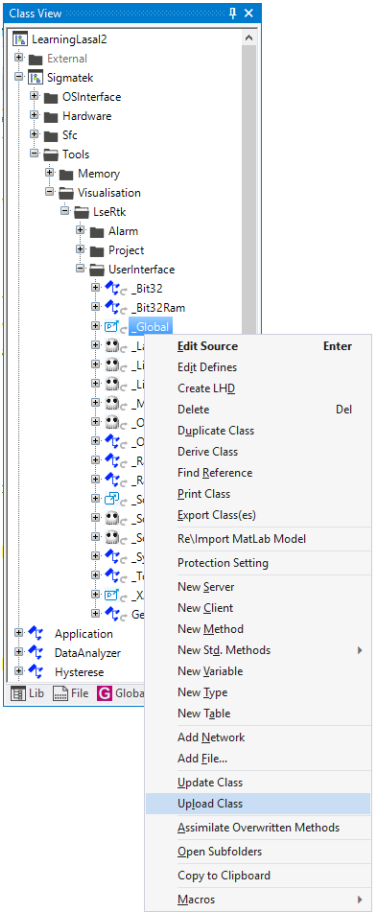
Select All and Deselect All

With this selection all dependency criteria for importing classes can be activated or deactivated.

If a class is referenced, the options **Update Class** and **Upload Class** can be selected from the context menu in the **Class View window**. With Update Class, the corresponding class from the library is updated.

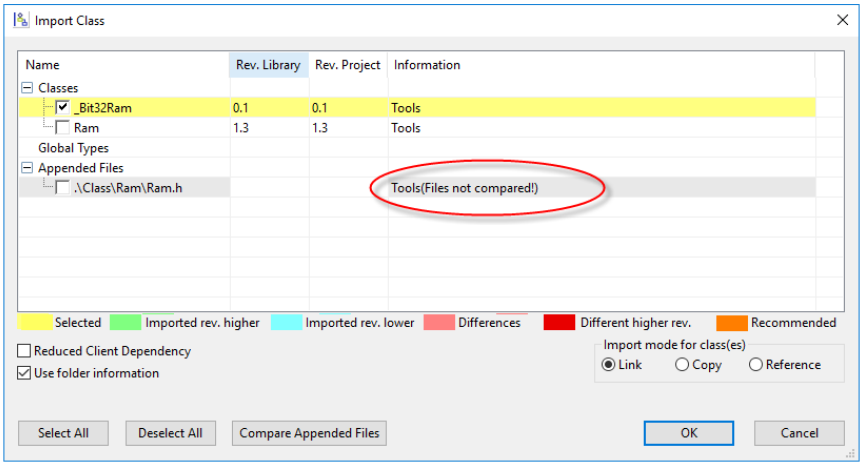


If a changed class is loaded into the library, select Upload Class from the context menu.

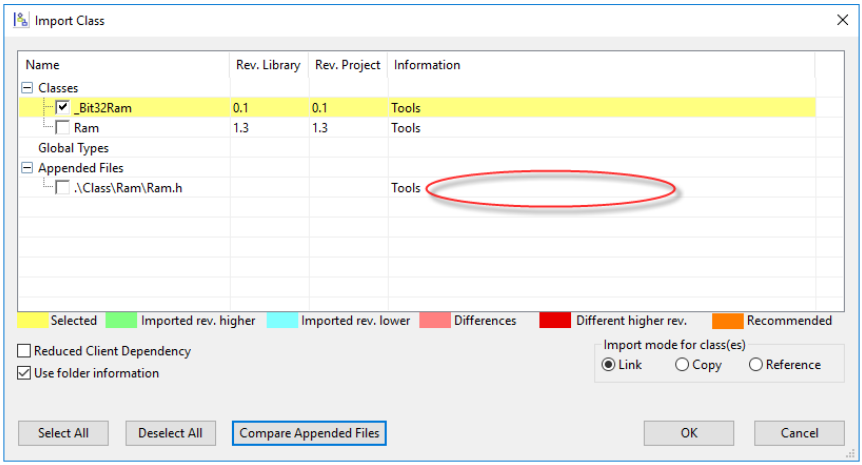


Compare Appended Files

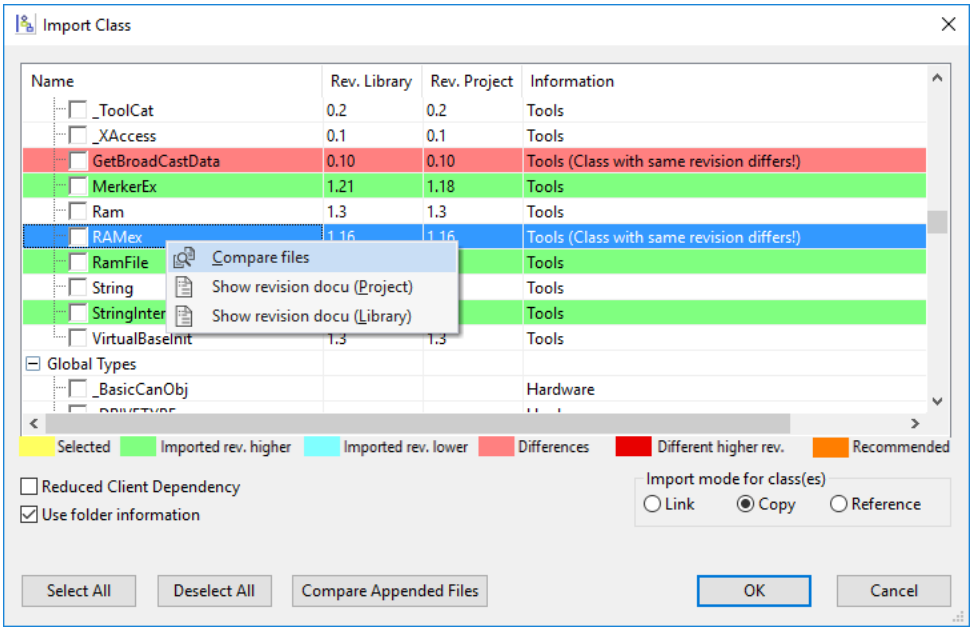
If files are added to the class that should be imported, this is indicated during the importation:



With the "Compare Appended Files" button, these additional files are compared. After being pressed, the warning "Files not compared!" disappeared:



With a right click in the Import Class window, the following context menu is opened:



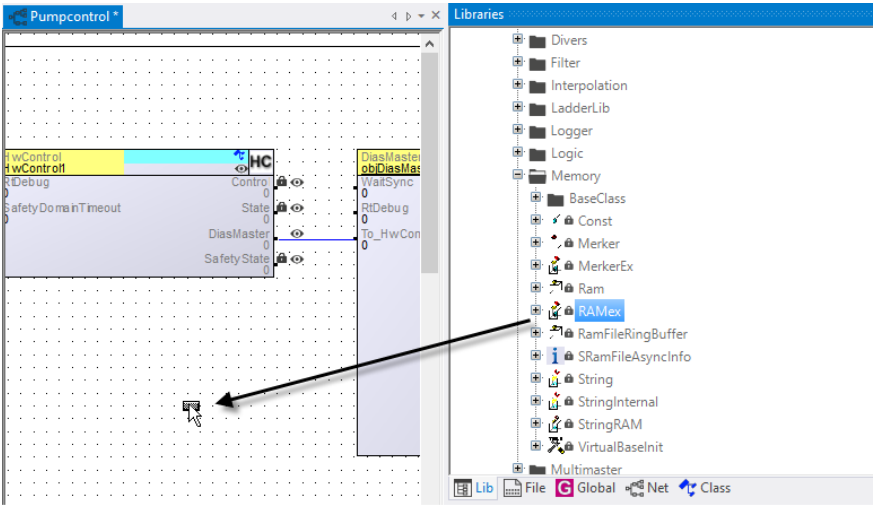
Compare Files

With this menu item, the class versions are compared through the Winmerge program.

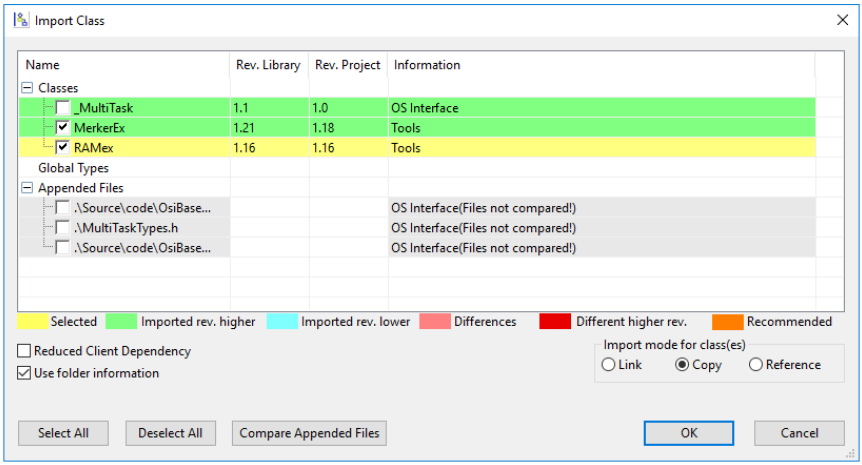
Show revision docu (Project) and Show revision docu (Library)

Open the respective revision documentation.

There is a much more comfortable option to import and position a class from a library. Simply Drag & Drop the desired class into the network window.



The familiar Import Class dialog then appears...



... and when confirmed with OK, the class is imported and positioned in the network!

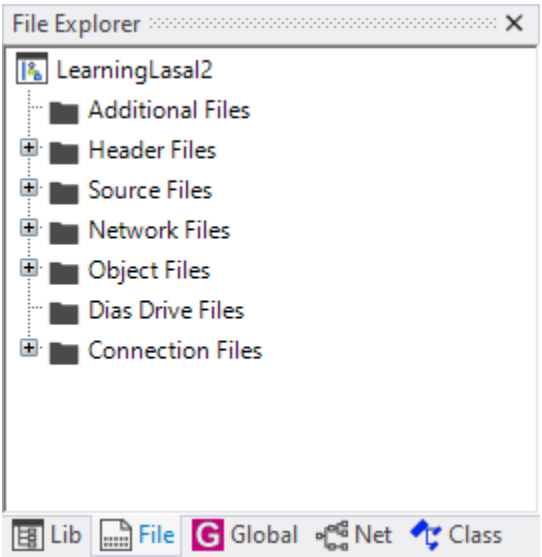
For an overview of the project tree, click on Project tree.

2.3.2 Files Tree

Select the **Files** tab for an overview of all files used by the project in the **File Explorer** window.

The overview is divided into the following parts:

- Additional Files
- Header Files
- Source Files
- Network Files
- Object Files
- Library Files
- Dias Drive Files
- Connection Files



2.3.2.1 Additional Files

This folder is available to the user for storing any project-related files. Click the **+** next to **Additional Files** to open the folder and view the entire list of these files. The **Additional Files** can have **any file extension and type**. To add a new file, right-click on the **Additional Files** folder and select **Add File...** from the context menu.



Note that the files in the **Additional Files** folder are **always copied and never referenced**!



LASAL CLASS 2 also uses this folder to store the **simulation configuration files**, among other things.

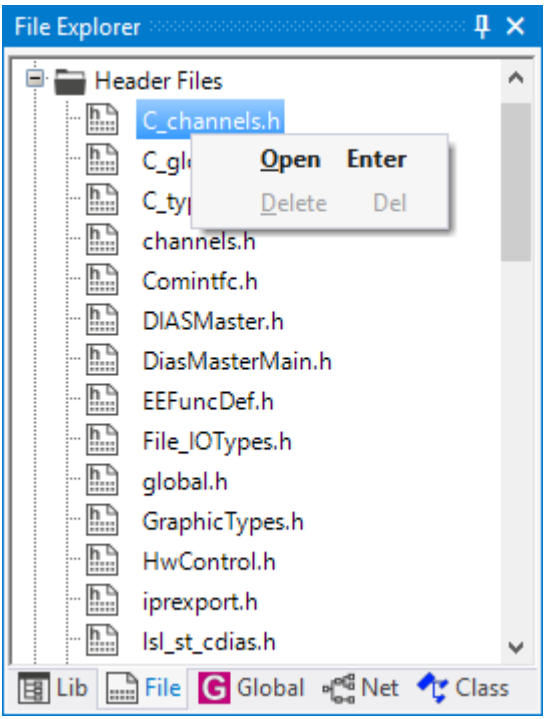
2.3.2.2 Header Files

Click on the + next to **header Files** to open the folder and show the entire list of these files. Header files can be recognize by the **.h** file ending. To add a new file, click on the **header Files** folder with the right mouse button and select **Add File...** from the context menu.



It is better when the header file is assigned directly to the class. To accomplish this, open the context menu of the class with a right click and select **Add File....**

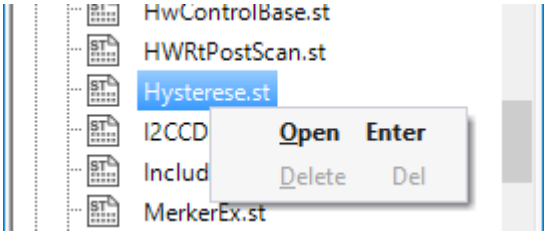
To display the program code of a specific header file in the program code editor, open the context menu with a right click on the file (here: **global.h**) and select open. To delete the selected header file choose **Delete**.



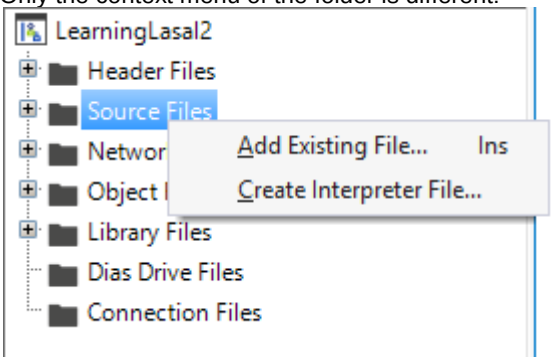
More information on **header Files** is found under *.H.

2.3.2.3 Source Files

These files are recognized by the file ending **.st**, **.c** and **.cpp**. The menu for accessing Source files is similar to it with header files (shown below).



Only the context menu of the folder is different:

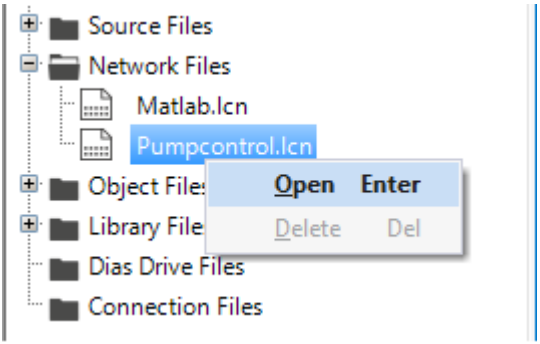


Add Existing File... is the same as in the context menu of the header folder (**Add File...**). **Create Interpreter File...** on the contrary, creates an **interpreter file** - analog to the menu item Tools/Interpreter + right-click on the project in the interpreter tree now displayed (the only menu item here is also called: **Create Interpreter File...**).

More information on Source Files is found under *.ST.

2.3.2.4 Network Files

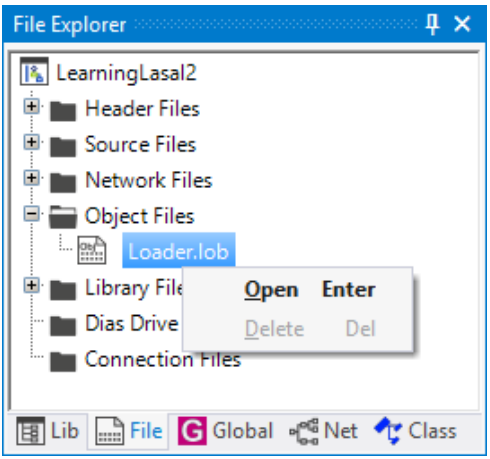
These files are identified by the ending, **.lcn**. The menu for accessing these files is also the same as for the header files as shown below.



More information on **Network Files** is found under *.lcn.

2.3.2.5 Object Files

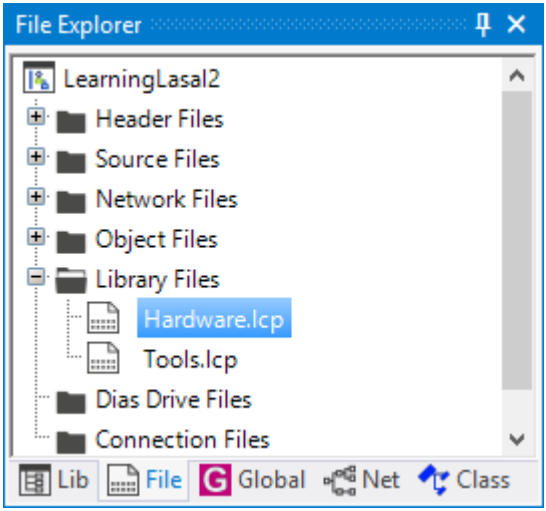
Files ending with **.lob** are object files. The menu for accessing these files can be seen under header files.



More information on **Object Files** can be found under *.lob.

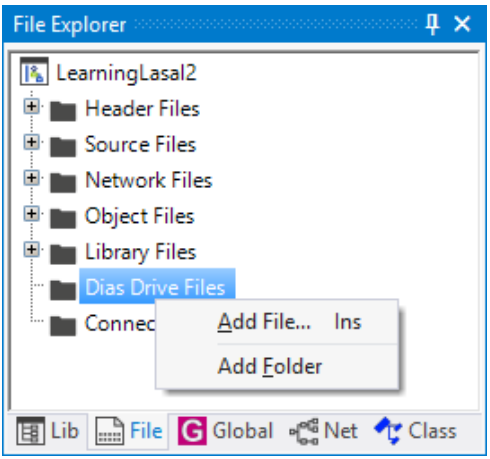
2.3.2.6 Library Files

If at least one library is available in the project, **All libraries** (with the ending ***.lcp**) is displayed here. Here, there is now context menu and the entries **cannot** be edited, expanded or deleted.



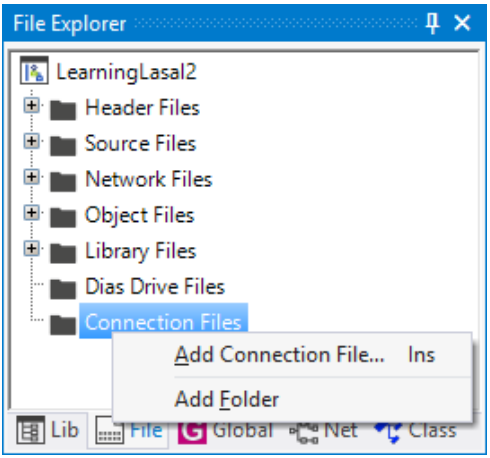
2.3.2.7 Dias Drive Files

When working with the Dias Drive, motor and controller data are found in this folder.



2.3.2.8 Connection Files

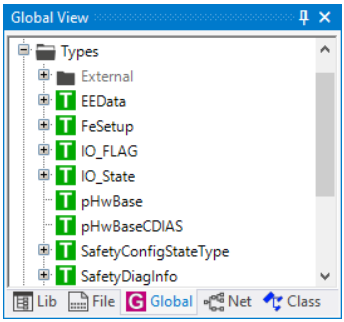
In this folder, the connection files for the project are found (for further information, see the chapter Project Connection Files).



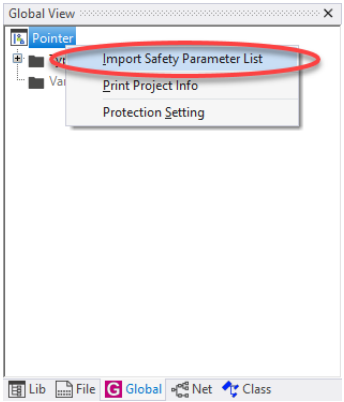
2.3.3 Global Tree

Click on the **Global** tab to open the **Global View window** and view a list of your **global types**.

Here, only **global types** are managed. **Class-related types** can be created and managed in the **Class View window** for the respective class (more information can be found under classes). A class-related type can only be created for a class and can only be used by the class for which it was created. A global type can be used by all classes within the project.



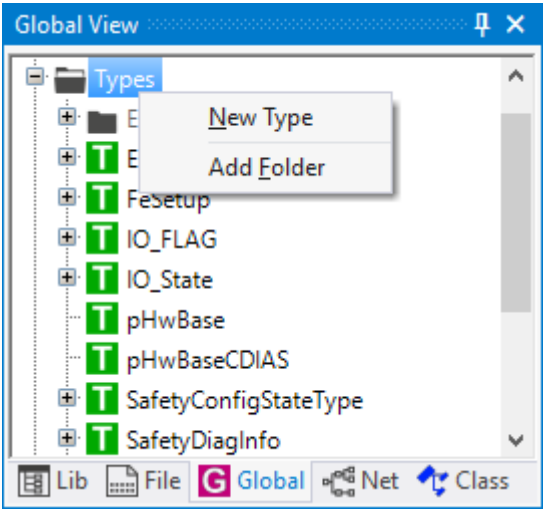
The context menu of the project in this view contains the item "Import Safety Parameter List".



For more information about the functionality see the Import Safety Parameter List chapter in the Project menu.

2.3.3.1 The "Types" Folder

Here, all global types defined directly in the project are listed. A right click on the **Types folder** provides two options:

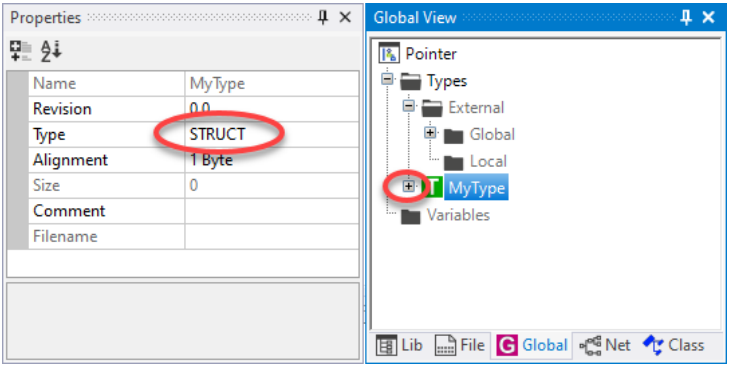


New Type	Here, a new global type can be created.
Add Folder	To better structure the global types, they can be grouped in folders. This menu item creates such a folder.

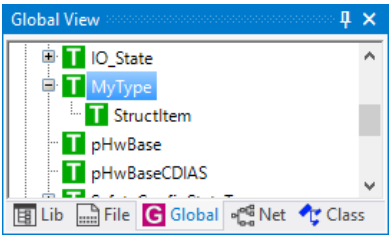
The External folder

In the External folder, all types are listed that are defined in the header file linked to the project.

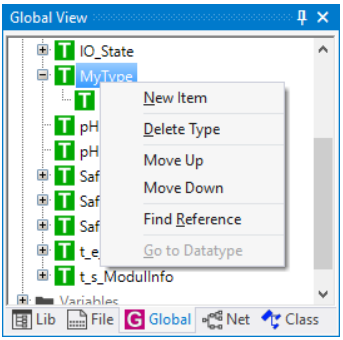
A click on the + next to a specific type opens a list of the subordinate types. To delete a type, select **Delete Type**.



If a type (**MyType** in right graphic) were assigned the data type **STRUCT** in the properties window, a subordinate type is created with the selection of **New Type** in the context menu (**StructItem** shown below).

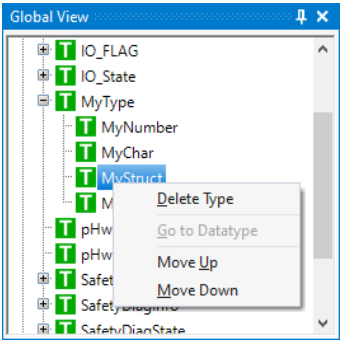


A right click on the new types opens the following options:



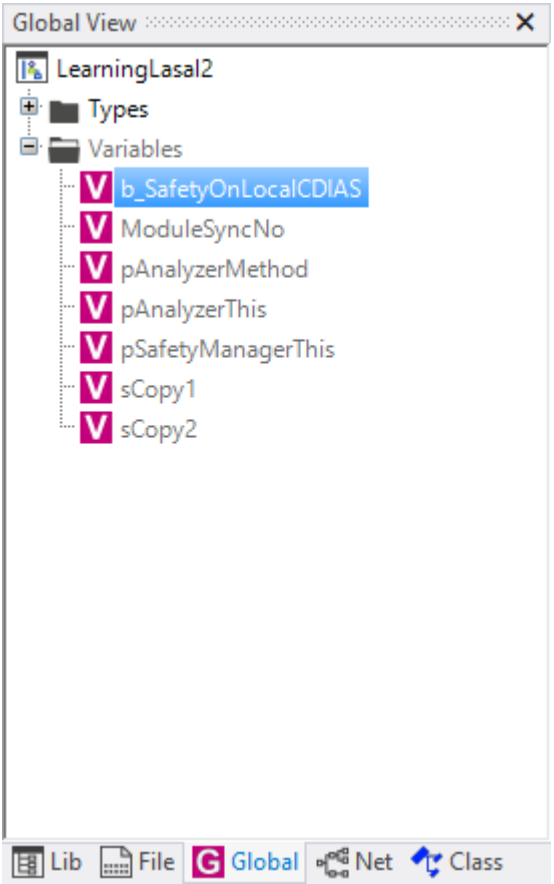
New Item	Adds a new sub-type.
Delete Type	Deletes the type (including sub elements!)
Move Up	Moves the selected type upwards (only enabled, if types above exist).
Move Down	Moves the selected type downwards (only enabled, if types below exist).
Find Reference	Searches the tree and code for the Type reference. The results are found under “Find Results 2” in the Output pane on the bottom of the screen.
Go To Datatype	With Go To Datatype, it is possible to jump to the data type of an element as long as it is not an elementary type (such as DINT).

If the context menu is called with a right click on the sub type, you will find already known entries here. Except for the entries **New Item** and **Find Reference** this menu is similar to the one of the higher level.



2.3.3.2 The "Variables" Folder

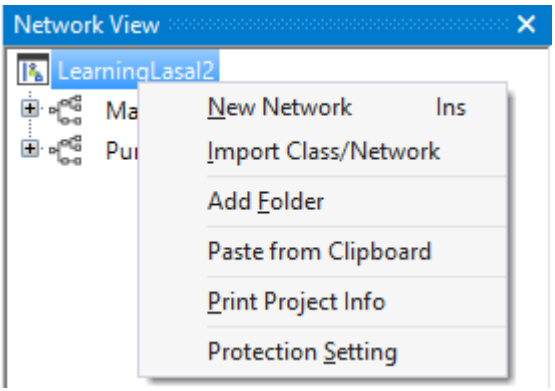
Here, the global variables are shown.



Unlike the global types, global variables cannot be re-created, changed or deleted. This is **strictly a display function only**.

2.3.4 Networks Tree

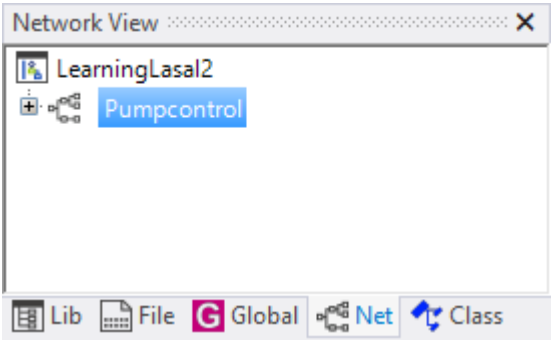
In the register tab, **Networks**, the **Networks View window** is opened.



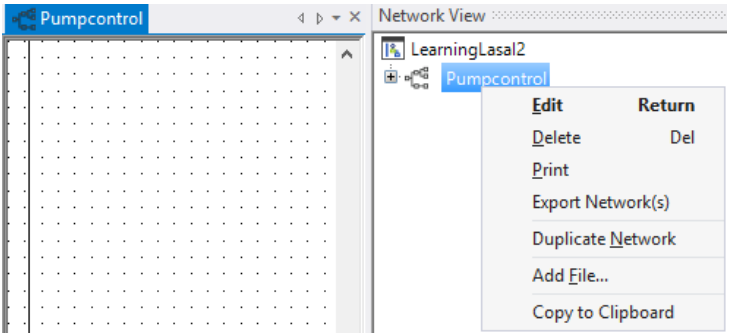
New Network creates a new network. **Import Class/Network** allows the import of a previously exported class or network. **Add Folder** adds a new subfolder, into which networks can be moved. **Paste from Clipboard** allows networks for a different LASAL2 instance to be copied here. **Print Project Info** prints the project information with an installed printer. **Protection Setting** opens the dialog of the same name, in which networks can be protected from changes with a password or encryption.

To add a new network to the project, right click on the project name (here: **LearningLasal2**) and select the New Network from the context menu.

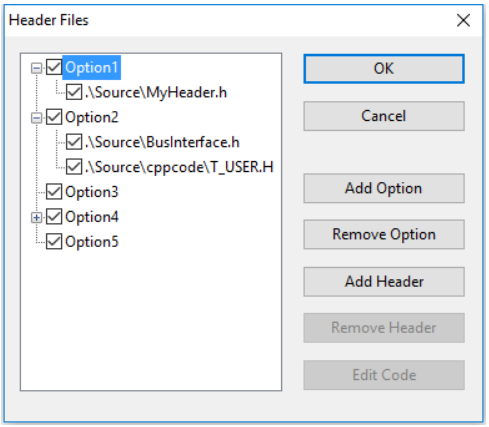
After the name (here: **Pumpcontrol**) has been assigned, confirm with Enter.



With a right click on the network name, **Edit** is selected from the context menu to place the network in the work space. The same is accomplished with a **double click in the network**.



With **Delete**, the network is deleted and with Print, all elements of then network are printed in a graphic display. With **Export Network(s)**, all selected networks (multiple selections possible!) are exported and can then be imported into another project.
Duplicate Network copies the network under a different name with all subelements. By selecting **Add File...**, the Header Files window in which the header file can be linked is opened.



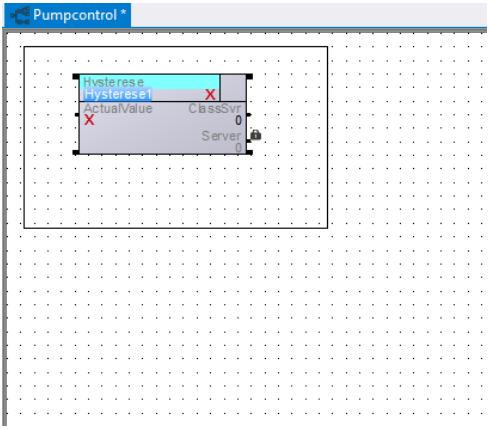
Before a header file can be linked, an option must be assigned. Here, the Add Option button must be clicked.

With **Add Header**, this option can now be assigned a header file and activated/deactivated using the checkbox. With this method, Header files can be activated and deactivated and for example, only certain constants defined in a header file can be used.

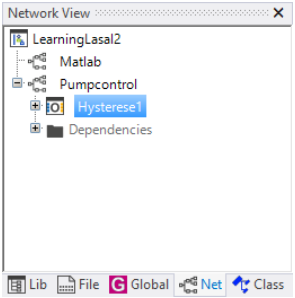
With **Remove Options** or **Remove Header**, the selected option or header file is deleted. With **Edit Code**, the selected header file can be opened.

Copy to Clipboard allows the entire network and all its elements to be copied to the Windows clipboard and inserted into another LASAL 2 instance.

To place the class in a network, change to the Classes register tab and place the desired class (here: **Hysteresis**) in the network using Drag & Drop.



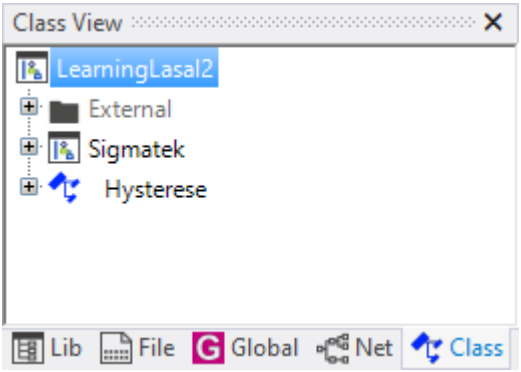
Similar to the image above, the class is now located in the network. When the **Network View window** – where the **Networks** register card should be selected -, the class (here: **Hysteresis1**) placed in the network is displayed.



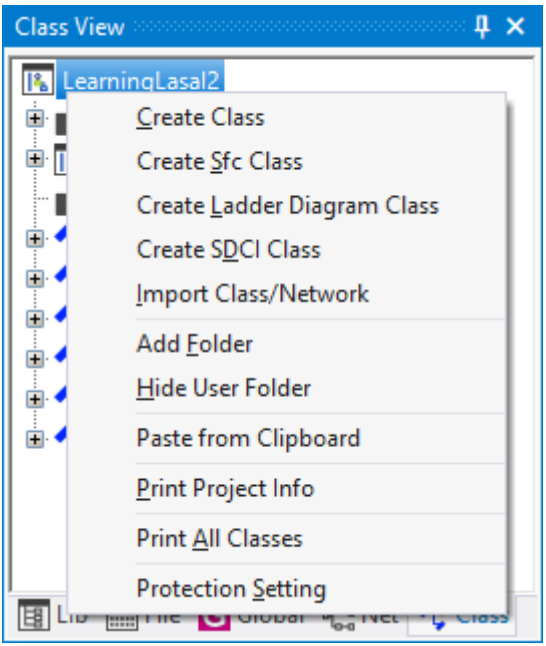
With a right click on the placed class, the context menu is opened. With the single menu item **Show**, the placed class is displayed in the network.

2.3.5 Classes Tree

Select the **Class** tab to open the **Class View window**.

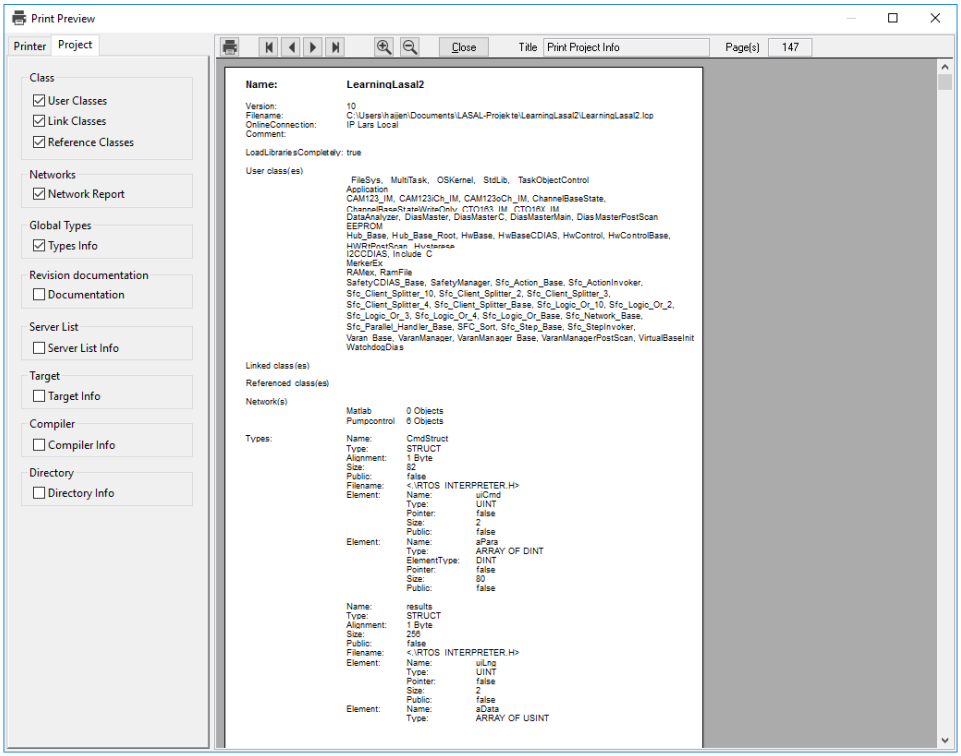


With a right click on the project name, the following context menu appears:



Create Class	Creates a new class.
Create Sfc Class	Creates a new Sfc class (See chapter SFC).
Create Ladder Diagram Class	Creates a new ladder diagram class.
Create SDCI Class	Creates a new SDCI class (See chapter Connecting SDCI Devices).
Import Class/Network	Imports a previously exported class or network.  **Attention!** It is important to note that vertices are ignored during import.
Add Folder	Adds a user-defined folder, into which classes classes can be moved (grouped).
Hide User Folder	Hides the user-defined folder, in order to view all classes in a flat hierarchy.
Paste from Clipboard	Inserts the classes from the Windows clipboard.
Print Project Info	Used to print all project information. Since – depending on the project – this can be a large amount (several hundred pages are not uncommon), the dialog that opens on the left side contains a series of options that can be enables and disabled (see the picture below)
Print All Classes	Unlike with "**Print Class**" – see there – here all classes of a projected can be printed at once. The continued process (preview, etc.) is the same as the other print functions.
Protection Setting	Opens the dialog of the same name, in which classes can be protected from changes using a password or encryption. With the “Autom. Reencrypt Project On Close“ checkbox on the „Encryption“ tab, the project is automatically re-encrypted when closed.

Print Preview



2.3.5.1 Method and server symbols

In the class view of the project tree there are different symbols and colors for methods or servers. Here you can find an overview:

Method icons:

Global methods	
Virtual global methods	

Private method (is superimposed with the method icon)	
CDECL-method (is superimposed with the method icon)	
Overwritten (is superimposed with the method icon)	
Table (is superimposed with the method icon)	
Virtual method in the derived class (can be seen only if "Show All Methods And Servers" is set to true in the Application Options)	
Non-virtual method in the derived class (can be seen only if "Show All Methods And Servers" is set to true in the Application Options)	

The light blue additional symbols can be combined with the method symbols. For example, the symbol  stands for a **virtual, overwritten private** method.

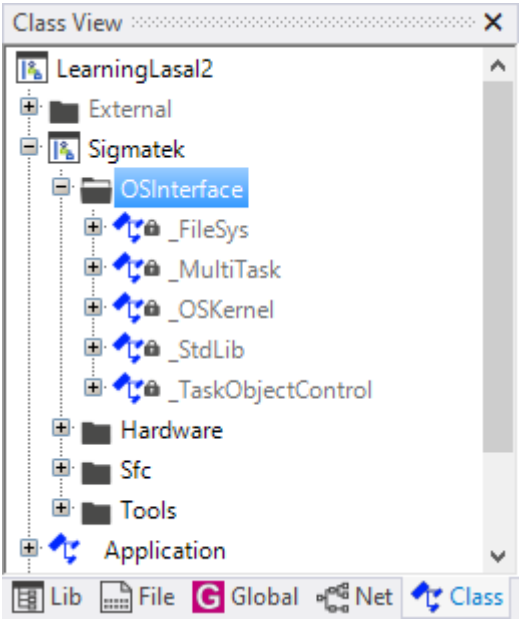
Server icons:

Server of a class, or overwritten server	
Server of the base class that is NOT overwritten (can only be seen if "Show All Methods And Servers" is set to true in the Application Options)	
Write protected (will be added to the server icon on the right)	
Visualized flag is set (will be added to the server icon on the right)	
World flag is set (will be added to the server icon on the right)	

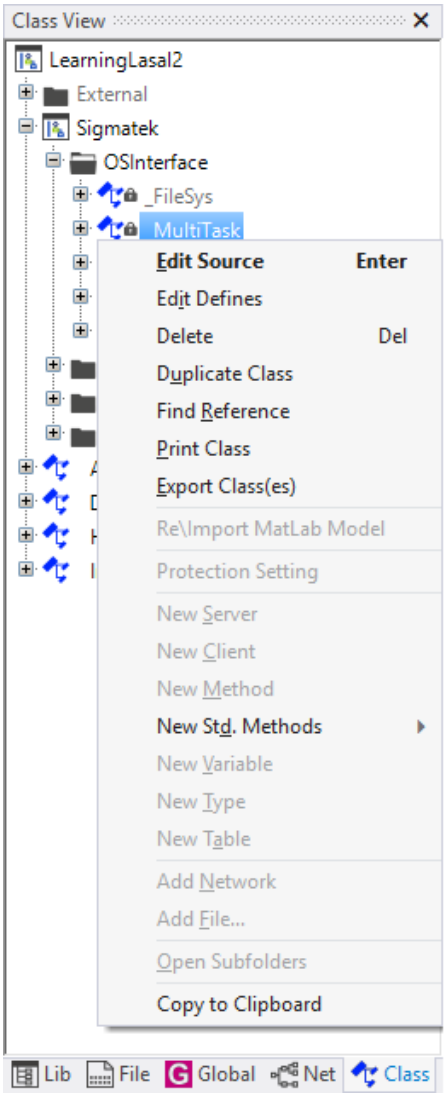
The dark gray additional symbols can be combined with the server symbols. For example, the icon  represents a server with the "**Visualized**" and "**World**" flags set to **true**.

2.3.5.2 Folder External

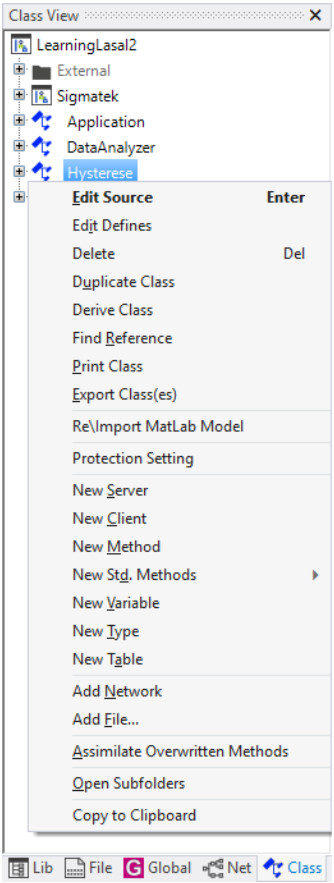
In the **External** folder, all classed in the project are listed that are linked with a header file. Click on the **+** symbol next to "Sigmatek" to display the **SIGMATEK standard classes**.



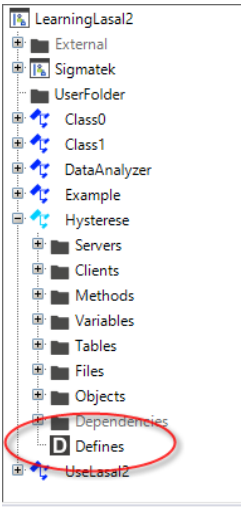
Each class or folder preceded by a **+** symbol has subordinated class specific components available, which contain more information. With a right click on a class (here: **_MultiTask**), the context menu is opened that shows the following entries.

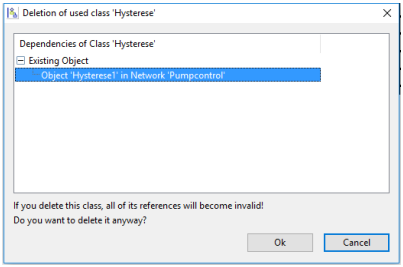
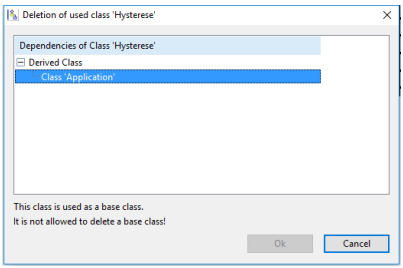

Class View
LearningLasal2
External
Sigmatek
OSInterface
_FileSys
MultiTask
Edit Source Enter
Edit Defines
Delete Del
Duplicate Class
Find Reference
Print Class
Export Class(es)
Re\Import MatLab Model
Protection Setting
New Server
New Client
New Method
New Std. Methods
New Variable
New Type
New Table
Add Network
Add File...
Open Subfolders
Copy to Clipboard
Lib
File
Global
Net
Class

Select **Edit Source**, to show the code from this class in the program code editor. With **Delete** the marked class is deleted. **Edit Defines**, **Duplicate Class**, **Find Reference**, **Print Class**, **Export Class(es)** and **New Std. Methods** (see description below) can be selected. All other entries for the SIGMATEK standard classes are inactive. To create a class (here: **Hysterese**), click on the project name (here: **LearningLASAL2**) and select **Create Class** from the context menu. Activate the context menu in the new class with a right click.



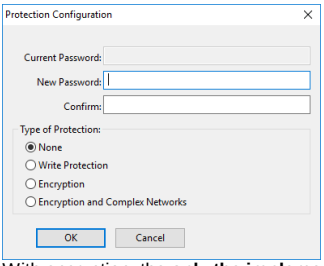
In this menu, all entries are now active and are listed as follows:

Edit Source	Shows the class code in the program code editor.
Edit Defines	With the Edit Defines command, a special section in the header of the St file is opened in the program code. Here, constants can be entered in the ST-Syntax (#define MyKonst 3) that can be used within the class. **The advantage**: an extra header file does not have to be created and linked for several less class-specific constants. If something is already entered here, there is a new sub element of the class ("Defines"). 

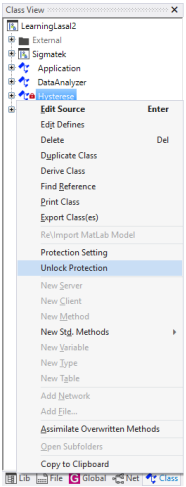
Delete	Deletes the class. If objects or references (e.g. object channel) exist, a dialog box with a list appears for he user to decide which to delete.  If the class to be deleted is a base class, LASAL does not allow the deletion. 
Duplicate Class	Duplicates the current class. It should be noted here that all duplicated classes still contain **all public types from the original class**. The reason for this is that otherwise, a class whose client is set to a server connection with the original server can no longer connect to the server of the duplicated class, since the fully resolved name of the (public) type, contains the name of the original class.
Derive Class	Creates a derivation of the selected class.
Find Reference	In the FindResults window, all calls to methods in other classes are listed.
Print Class	Opens the print menu.
Export Classes	Exports the selected class.
RE \ Import Matlab Model	Imports a C file exported from MatLab Simulink.

Protection setting

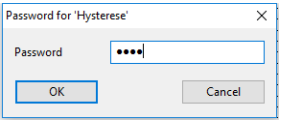
Enables a class to be write protected or encoded.



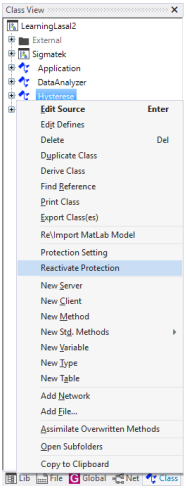
With encryption, the **only the implementation** is encrypted and **not the entire file**! In addition, **Encryption and Complex Networks** prevents the possibility to click into the class networks of complex classes in the object network; also complex classes cannot be expanded in the Watch window. If a project is password protected, a new menu item, "**Unlock Protection**" is listed in the context menu.



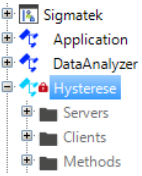
When this menu item is selected, the encryption can be temporarily disabled after entering the password



From this point, the menu item "**Reactivate Protection**" is included under "**Protection Setting**". With this option, the password can be re-entered to activate the encryption.

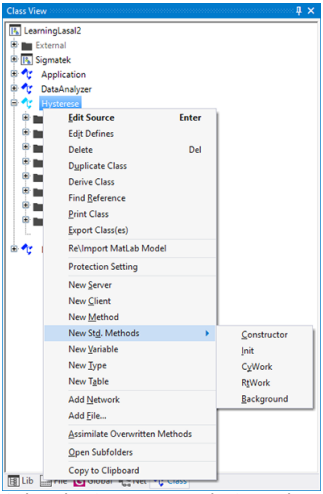


An encrypted / password protected class appears in the class view as follows:



To permanently disable the protection, the **Protection Settings** must be opened and after entering the current password, the fields "**New Password**" and "**Confirm**" must be cleared.

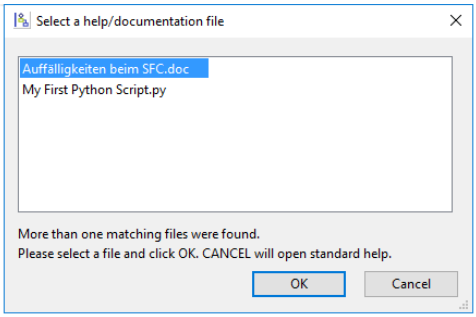
New Server	Creates a new server for a class.
New Client	Creates a new client for a class.
New Method	Creates a new method for a class.

New Std. Methods	 Select the Constructor, Init, CyWork, RtWork or Background to create these methods.
New Variable	Creates a new variable for a class.
New Type	Creates a new class related type.
New Table	Creates a new table for a class.
Add Network	Creates a reference network for a class. With this instruction, the class is made into a complex class. In the properties window, the class is then defined as a base class or it could be placed in objects embedded in the reference network. A double click on the reference network of the class in the tree opens the network editor. In the editor bar, the character "[**RN**]" is inserted before the class name to indicate the **R**eference **N**etwork of a class is opened and not that of an object.
Add File...	Open the explorer to select and add a new source file, header file, *.VOV file or help file. More information can be found under Source code files.
Open Subfolders	Opens all subfolders in a class. All members of this class (servers, clients, methods, variables, …) are then visible at a glance.
Copy to Clipboard	Copies the class into the Windows clipboard.

Macros

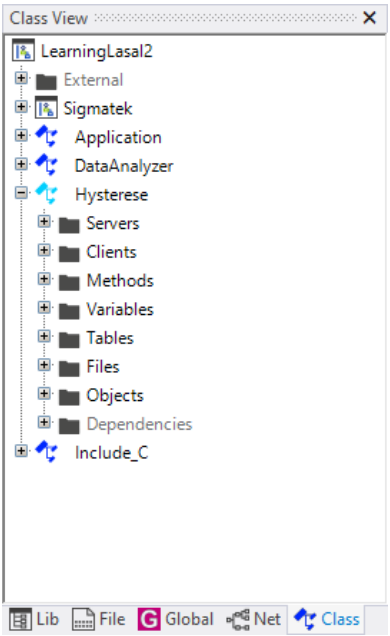
Runs the created Macros with the Macro manager. The menu item is only visible when Macros are actually available!

With a selected class, the **library help** opens with the **F1** key, unless a help file was added with Add File. In this case, LASAL asks the user whether this file or the standard (library) help should be opened.

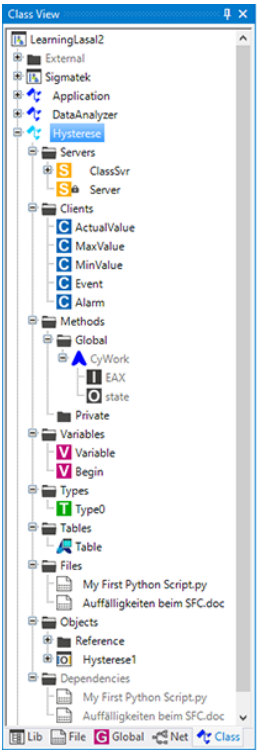


Once the components just described are added to the class, they can be displayed by clicking on the + symbol before the class name (here: **Hysterese**).

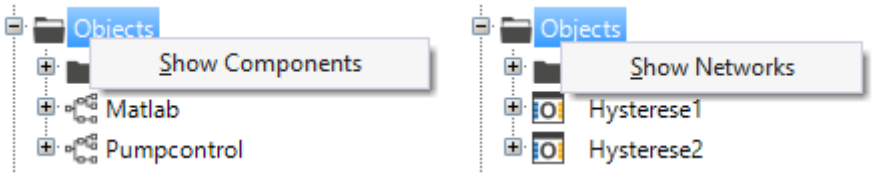
Depending on which components were added (**Servers**, **Clients**, **Methods**,...), they are listed in the Class View window as shown below.



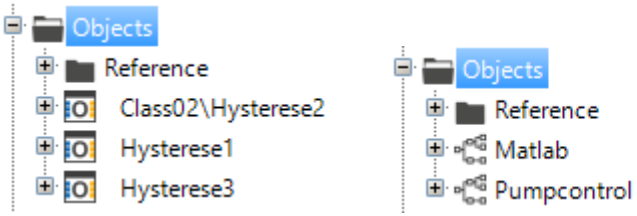
Each of these folders can also be opened with a click on the **+** symbol.



In addition to the subfolders of class mentioned above, the Objects folder is listed as soon as the class is used in a network or complex class. Via the context menu of the Objects folder it can be selected between two views:

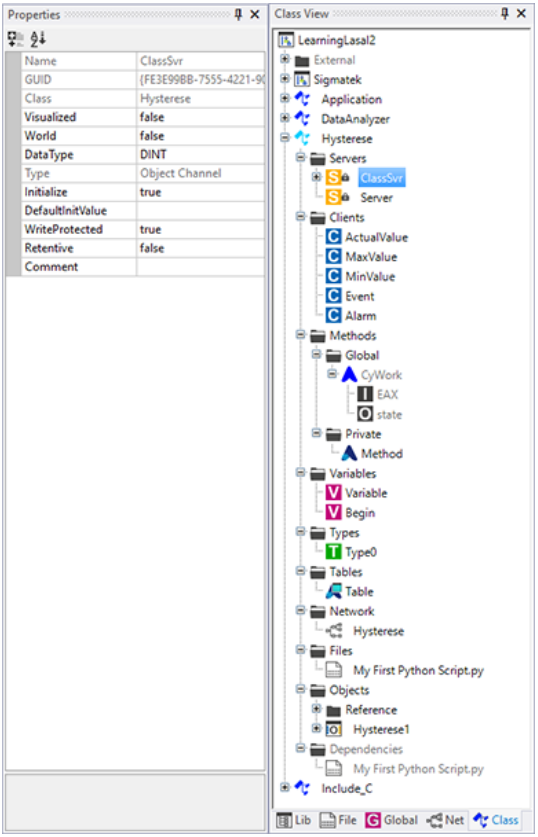


- Alphabetical listing of all instanced objects
- Grouped listing according to the name of the network in which the object is placed.



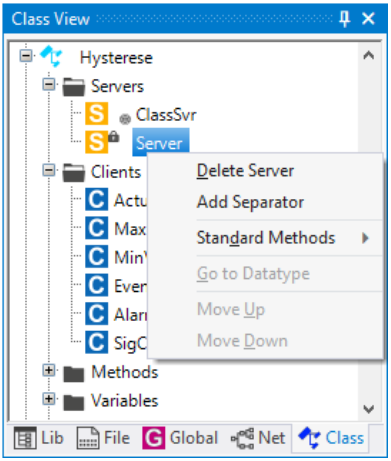
Use in a complex class is always displayed with the subfolder **"Reference"**, independently to the view type.

To display the properties of a server, for example, select it (here: **ClassSvr**) and make the desired settings in the properties window (see illustration below).

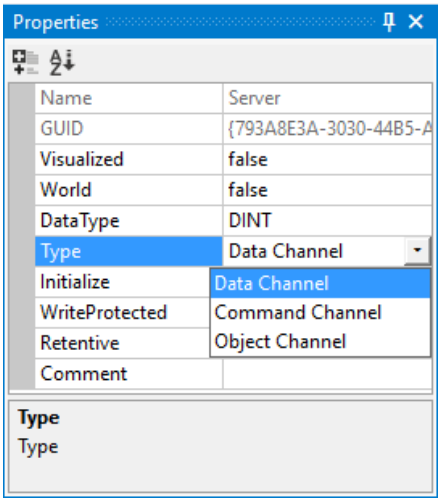


The properties for **Clients**, **Methods**, **Variables**, **Types**, Tables or **Networks** can also be changes as described above. More information on "editing properties" can be found under Properties window.

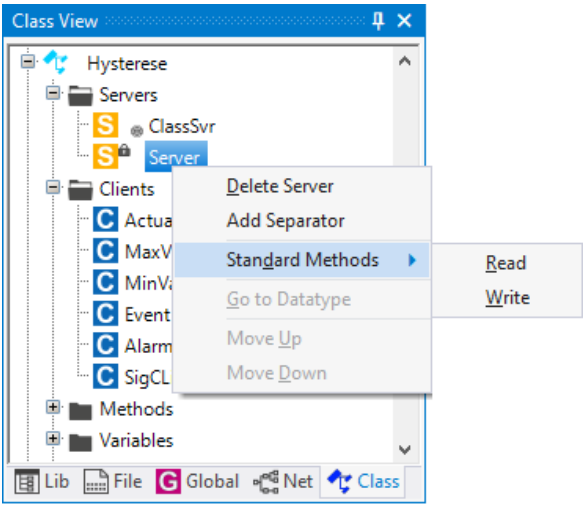
Click on a server (here: **Server**) with the right mouse button to open the context menu:



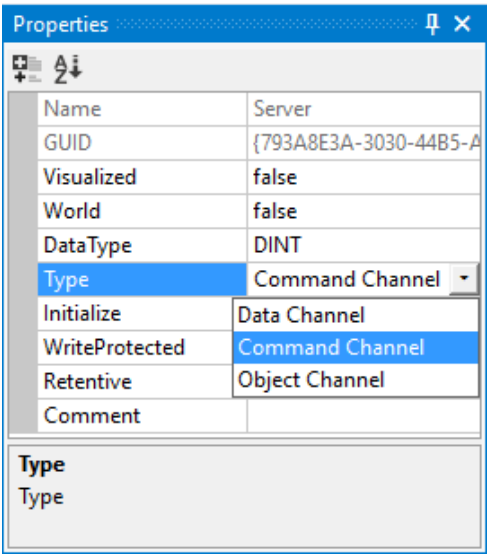
The context menu for **Standard Methods** depends on the channel in the **Properties window**.



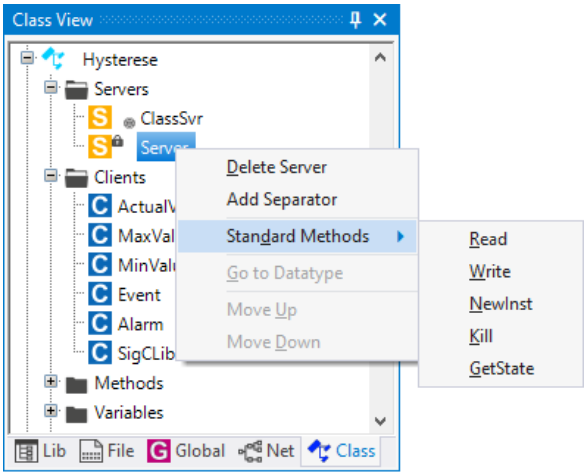
If the **Data Channel** is selected as shown above, the standard methods context appears menu as follows:



The context menu contains the **Read** and **Write** only.



If the **Command** or **Object Channel** were selected, the standard methods context menu would list three additional methods as shown below:

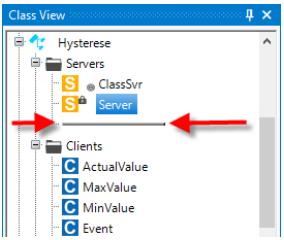


Delete Server

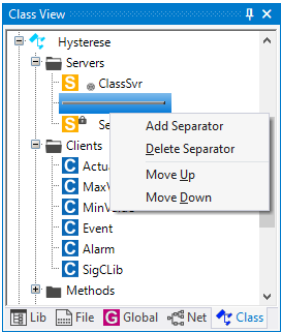
The selected server is deleted.

Add Separator

Below the selected server, a separator - similar to those in menus - is added for logically grouping the servers:

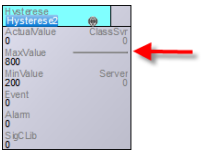


Right-clicking it, opens the following context menu:



Add Separator	Adds another separator
Delete Separator	Deletes the selected separator
Move Up / Move Down	Moves the separator

The separator is also visible at the according place in the object network:



Also the clients provide the possibility to set separators.

Standard Methods	Read	Returns the contents of a channel (**4-byte DINT**).
	Write	The contents of a channel are overwritten with the assigned value.
	NewInst	The **NewInst** function is called from the operating system when the interpreter code is processed. This function should execute the existing code and signals the interpreter to what extent the process was successful. If **NewInst** is selected, the "**Use Hidden NewIn**st" property can be selected in the Properties window (default = true), which specifies whether the **NewInst** function template in the *.st file should be created in the Hidden area.
	Kill	The Kill method is called by the interpreter when it is ended (for longer NewInst instructions).
	GetState	The GetState method is called by the interpreter if lengthier **NewInst instructions** are processed. This instruction decides whether the next instruction should be executed.
Move Up	Shifts the selected server to a higher position in the server list.	
Move Down	Shifts the selected server to a lower position in the server list.	

2.3.6 Tool-dependent Temporary Trees

Interpreter	Here, new interpreters are created and the existing ones are managed. For more information, see the chapter Interpreter in LASAL CLASS 2.
Data Analyzer	The Data Analyzer can be used to load and analyze data from the controller on the PC. For more detailed information see the chapter Data Analyzer.
Dias Drive	The DiasDrive section in LASAL CLASS 2 handles the configuration and tuning of servo drive hardware modules. For more information see the chapter DiasDrive.

2.4 Properties Window

The available settings in the **Properties window** vary depending on the selection made in the **project tree**.

2.4.1 Project Properties

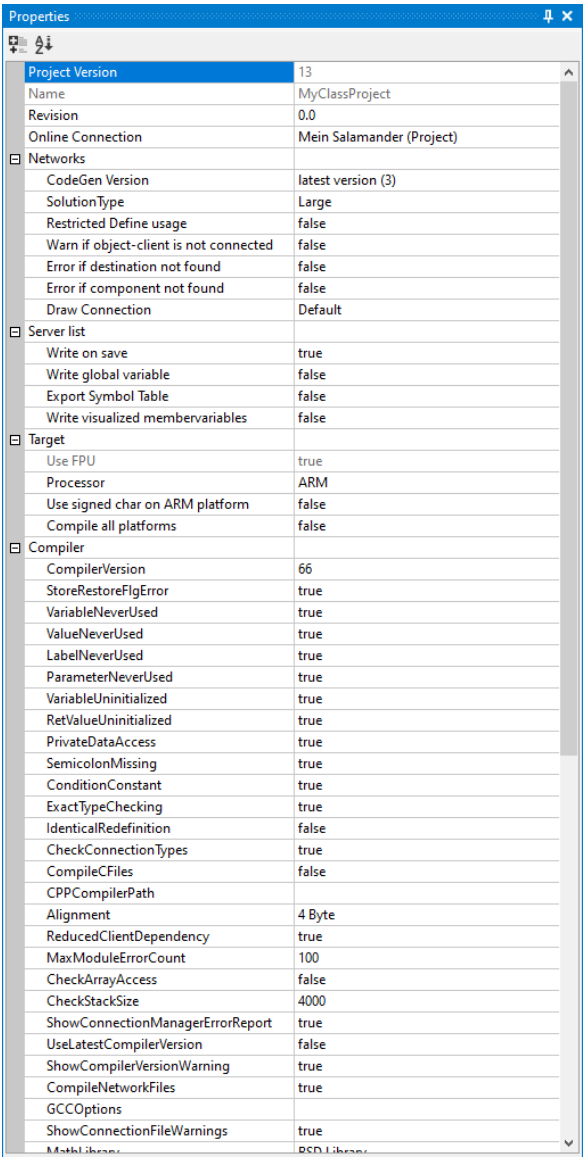
2.4.1.1 Properties of a project

To display and set the project properties in the **Properties window**, select the register tab **Classes** in the project tree and click on the project name.



Note

The first entry in the project tree always opens the **Project Properties** in the **Properties window**.


Properties
Project Version 13
Name MyClassProject
Revision 0.0
Online Connection Mein Salamander (Project)
Networks
CodeGen Version latest version (3)
SolutionType Large
Restricted Define usage false
Warn if object-client is not connected false
Error if destination not found false
Error if component not found false
Draw Connection Default
Server list
Write on save true
Write global variable false
Export Symbol Table false
Write visualized membervariables false
Target
Use FPU true
Processor ARM
Use signed char on ARM platform false
Compile all platforms false
Compiler
CompilerVersion 66
StoreRestoreFlgError true
VariableNeverUsed true
ValueNeverUsed true
LabelNeverUsed true
ParameterNeverUsed true
VariableUninitialized true
RetValueUninitialized true
PrivateDataAccess true
SemicolonMissing true
ConditionConstant true
ExactTypeChecking true
IdenticalRedefinition false
CheckConnectionTypes true
CompileCFiles false
CPPCompilerPath
Alignment 4 Byte
ReducedClientDependency true
MaxModuleErrorCount 100
CheckArrayAccess false
CheckStackSize 4000
ShowConnectionManagerErrorReport true
UseLatestCompilerVersion false
ShowCompilerVersionWarning true
CompileNetworkFiles true
GCCOptions
ShowConnectionFileWarnings true

Name

Name of the LASAL CLASS 2 project (here: **LearningLasal2**).

Revision

Project revision number. Press the  button to call the **“Revision Documentation of LearningLasal2”**.

Online Connection

Used to define the type of online setting. Press the  button to open the **LASAL Online Configurations window**. For a more detailed explanation see chapter Online Settings.

Networks

Here, the settings for the object networks are made.

CodeGen Version

CodeGenerator version. This is responsible for the **status of the code after saving** (not to be confused with the compiler!). If an invalid value is entered here, the latest CodeGen Version is set. It is possible to select a specific CodeGenerator version or the "latest version". In the latter case, the latest valid CodeGenerator version is shown in brackets at the end (e.g. "latest version(3)"). For more detailed information on the respective versions, see the Help menu chapter.

SolutionType

Use the  button to determine **Solution-Type** as **large** or **small**.

This setting determines the configuration of the object network tables. Here, 2 options are available:

With **Large Solution**, connections and Init values are entered through the client names, servers and objects. The table is thereby somewhat longer (st file of an object network), however, it provides the option to later exchange individual classes in a machine.

With **Small Solution**, connections and Init values are entered through the memory offset. The table and the boot time of the loader are somewhat shorter, however, changing the memory size of a class results in the network containing the objects of this class changing. Individual classes cannot be loaded.

Restricted Define Usage

Default: **False**

If a redefintion compiler warning is given due to a multiple #define, the property can be set to **true**. With **False**, all defines of the class placed in a network are categorized the same and made available for I.e. the initialization values of servers. If the properties are set to **True**, a define of class A cannot be used as an initialization value in a server of class B.

Warn if object-client is not connected

If the property is set to **True**, an error message is also given when an object-client is connected that is **not** required. The default setting is **False**. This property can be toggled using Script in the project property "`WarnNotConnectedClient`".

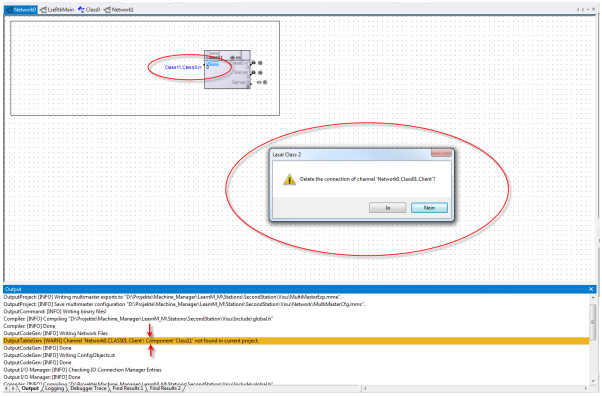
Error if destination not found

If the property is set to **True**, an error message is also given when client is connected to a **server**, which does **not exist**.

Error if component not found

If the property is set to **True**, an error message is also given when client is connected to a server, whose **object does not exist**.

If you doubleclick after the build and showing the error message / warning, you jump to the client in the object network. Additionally a dialog opens asking the user, whether this connection should immediately be deleted or not.



If the user answers with "Yes", the connection is deleted.

Draw Connection

This property determines how the network connections in objects should be displayed on the client side - with connection lines (**Line**), or with the display of the server name on the client, as is always the case with external network connections - i.e. connections to another network (**Text**). The third option is (**Default**) - the standard value here is **Line**.

This property concerns the display of connections **at all network objects** in the project. If you want to change the display **only at a certain object**, you can set this individually in the properties of the respective object.

Server list

Here, the server list settings are made.

Write on save

This setting specifies whether the server export list (maeexp.txt) is written to with each write procedure. If this setting is **false**, the server list is rewritten during a Rebuild All or when the Build \ Write Server List menu command is run.

Write Global variable

If this property is set to **True**, a global variable is also written to the Maeexp.txt file.

Export Symbol Table

If the property is set to **True**, a file named LSL_SymTabExport.st is stored in the Source folder of the project in addition to the server list. This file can be used through programming to access project data elements during runtime.

Write visualized membervariables

If this property is set to **True**, all member variables are also assigned the property **Visualized**. If this property is set to true, the variable is written to the MaeExp.txt file and visualized in LSE. The corresponding variable is displayed in the tree next to an **eye symbol**.

There are several **limitations** to visualizing member variables:

Variables **cannot be changed through the visualization**, only displayed. They must be **1, 2 or 4-byte variables, structure or array elements with a scalable data type - LREAL** is **not supported**. With pointer variables, **only the address can be visualized** but not the data behind it.

Member variables are only available in the LSE Kernel, not in LseEasy, LseLight, Visual Object View and DotNet Kernel Array elements **cannot be accessed via index**. In addition, arrays also **cannot be directly interpreted as strings**.



Caution!

Testing the offset for correctness during run time is not possible – if the offset values are wrong, values could be incorrectly displayed!

LASAL CLASS 2 can only export correct offsets for compiled projects. **After each class change** in LASAL CLASS (new client, server, member variable; change of a type used by member variables etc.), **the LASAL CLASS project must be compiled** as well as **Reference to Variables** run in LASAL SCREEN EDITOR to ensure that the offsets are correct. Both projects must then be **loaded into the target system**.

Additionally in the header file "**T_ipc.h**" belonging to the class "**_Lse**", the line "//#define UC_MEMBERVARIABLE" must be **added**, and the project has to be **compiled** with the compiler setting "**Compile C Files**" **= true**!

Target

Here, the settings are made via the control.

Use FPU

This setting can be set to **true** or **false**. When set to **true** the instruction set uses the optional mathematic coprocessor for REAL or LREAL calculations. If set to is set to **false**, the calculation is emulated. The emulation is slower than the coprocessor but is available in every target system.

CAUTION! Setting this property to true without a coprocessor in the target system causes the control to crash!

Processor

The target architecture is determined here (Intel x86, or ARM).

Use signed char on ARM platform

The ARM GNU C Compiler uses the "char" data type as unsigned type different to the Intel compiler. The reason is historical. if this option is activated, **both compilers behave in the same way**.

Compile all platforms

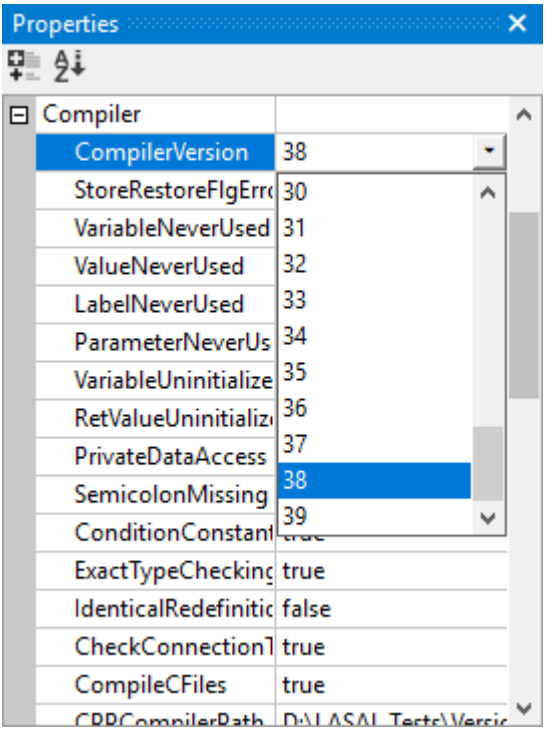
If this option is set to **true**, the project is **translated for all supported architectures**. If the user goes online in an existing project with a selected online connection to a control with a different architecture, the display of the selected target architecture in the toolbar is automatically converted.

Compiler

For this reason, all settings required for the compiler are made. Most settings are set to TRUE by default.

CompilerVersion

Indicates the compiler version. The compiler version can be selected:



StoreRestoreFlgError

With **false**, the compiler error messages are disabled with incorrect use of the Store/Restore flags. It is stongly recommended that this switch be always set to **true**!

VariableNeverUsed

Displays a message (**WARN**) in the compiler window when a local variable in a method is not used.

E.g. 'i' declared but never used

ValueNeverUsed

This message appears when a variable is assigned a value that is never used.

E.g. 'i' assigned a value that is never used

LabelNeverUsed

A message stating that a jump point is defined which is never used.

E.g. Label 'step20' is declared but never used

ParameterNeverUsed

Generates a message (**WARN**) in the compiler window that an unused parameter is in the program.

VariableUninitialized

Means that a variable has not been initialized.

E.g. Possible use of 'a' before definition

RetValueUninitialized

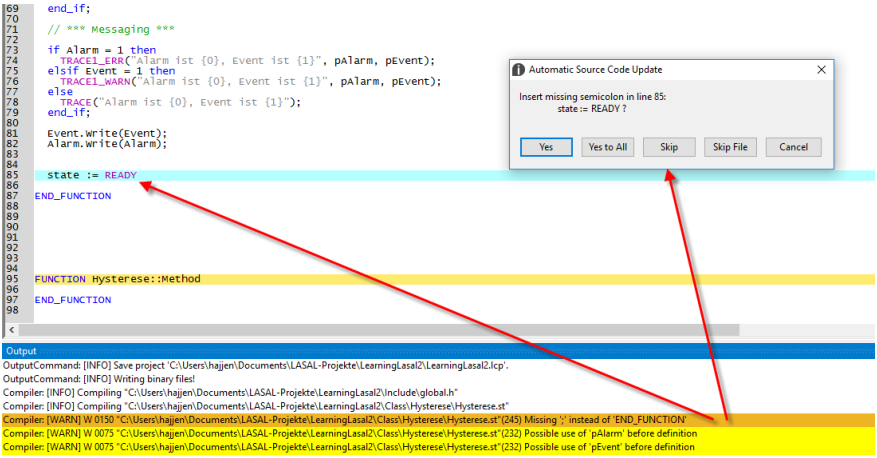
This message is generated when the return value of a method has not been initialized.
E.g. Return value 'OutParam' is not defined

PrivateDataAccess

Generates a message when private variables are accessed by other objects.
e.g. (Private) data element 'var' of class 'Class0' cannot be accessed here

SemicolonMissing

This message appears when the semicolon at the end of an instruction is missing (message). With a double-click on the message, not only is a jump made to the location in the code, a dialog opens in which the error(s) can be comfortably corrected.



This only functions however, when no errors (only warnings) occur during compilation.

ConditionConstant

If a condition is always **true** or **false**, a message is triggered

(E.g. infinite loop).

E.g. Condition is always TRUE

ExactTypeChecking

Displays an error message when using different data types.

E.g. Different types: using 'DINT' instead of 'UDINT'

CheckConnectionType

Checks the connection between a client and server to ensure that they are the same data type.

CompileCFiles

This instruction used to determine whether to compile C-files or not. With the detail debuginfo value, whether the C-files can also be debugged or not is defined.

CPPCompilePath

Here, a path can be defined deviating from the application option path, where the C compiler should be located for this project. If both locations are defined, the project path wins. If it is illegal or empty, the path in the Application Options is used. Only if none of the two paths has a C compiler, the error message

```
Compiler: [ERROR] No C / C ++ compiler Given. Skipping modules ("D:
\ projects \ DS4615 \ Class \ Class0 \ add_file.c")
```

is issued by the compiler when trying to compile C files.

Alignment

Serves as the data alignment for the entire project. Combined projects and/or libraries to which classes are linked must be compiled with the same alignment (this is also **true** for the loader).

More information on Alignment can be found in the #pragma pack chapter under Pragma instructions.

ReducedClientDependency

The compiler setting ReducedClientDependency (RCD) significantly increases compilation speed for larger projects and ensures that the classes are encapsulated. In the setting is defined, an object client of class A is recognized over the object client referenced by class B. Classes referenced over the object channel of class B are not recognized within Class A. Instructions such as Client.Client.Read() are not allowed through this setting.

Definition of the compiler setting when loading projects:

When loading projects in LASAL CLASS 2, synchronization between the project and library is established automatically by which new class information is assumed by the project. After determining whether RCD is being used with the project, the automatic importing of unnecessary classes in prohibited (a simple synchronization is executed however). If the RCD setting is deactivated during the working with LASAL CLASS 2, the project must be reloaded to include the missing classes.

MaxModuleErrorCounter

With this value, the maximum number of compilation errors can be output. If more errors occur in a module during compilation of a project than the **MaxModuleErrorCounter** value, only the specified number of errors is shown in the Compiler window. The errors from the next module are then shown in the Compiler window.

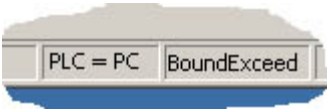
CheckArrayAccess

If an index variable is accessed in an array during the runtime, it is tested against the array limits when this function is set to **True**. Array access is tested by the operation system during runtime. If the array limits are violated, the program processing stops.

The instruction pointer is set to the corresponding line…

```
1
2  FUNCTION VIRTUAL GLOBAL Class0::CyWork
3    VAR_INPUT
4      EAX    : UDINT;
5    END_VAR
6    VAR_OUTPUT
7      state (EAX)    : UDINT;
8    END_VAR
9    VAR
10     myArray :array[0..9] of DINT;
11     i :DINT;
12   END_VAR
13
14
15
16    for i:= 0 to 20 do
17       myArray[i] := i;
18    end_for;    i : 10 : DINT
19
20
21   state := READY;
22
23 END_FUNCTION
24
```

... and in the status line in Lasal, the BoundExceed warning is given:



CheckStackSize

If this option is set to a value greater than 0 (0 is disabled), the stack is tested during compilation for the defined size. If the value (default: 4000 bytes) is exceeded, an error message is given.

Show Connection Manager Error Report

If this option is set to **false** (default is **true**), ConnectionManager error messages such as „Server side ... of IO connection is incomplete (channel name missing)“ are **not** displayed.

UseLatestCompilerVersion

If this option is set to **true**, the newes compiler version is used automatically.

ShowCompilerVersionWarning

If this option is **true**, a compiler warning is triggered when a compiler version is used that is older as the latest version. A double click on this warning opens a dialog box, in which the user is asked whether the newest compiler version should be used. When OK is selected, the newest compiler version is set automatically. Otherwise, the user is asked whether the warning should continue to be displayed. If answered no, the **ShowCompilerVersionWarning** is set to **false**.

CompileNetworkFiles

If this option set to **true** everything - as always - is compiles, including the networks. If the flag is set to **false**, network files are neither written, compiles or linked. This is especially advantageous when processing several partial projects (e.g. via Python script) and searching for errors in the ST files, since the corresponding error messages regarding the network compilation are eliminated.

GCCOptions

In this field the user can set **user-defined compiler options**. If a value is set here, the **internal compiler options** are **extended**.

List of compiler option is huge, and can be found in details on https://gcc.gnu.org/onlinedocs/gcc-3.3.6/gcc/Invoking-GCC.html. Here is a quick summary of the options in LASAL CLASS 2:

Code optimization

Option	Function
-O0	Optimization Off. Good for debugging.
-O **-O1**	Optimize code using most safe options.
-O2	Optimize even more.
-O3	Optimization with loop unrolling.
-Os	Optimize to produce most compact form.
-ffast -math	Improve math speed by relaxing strict IEEE/ISO compatibility.

X86 specific options

Option	Function
-march=core2	Uses modern CPU features (newer PLC models only)
-mfpmath=sse, 387	Optimizes math for newer PLC models

Note that if you specify `-march=core2`, you don't have to specify `-mfpmath=sse,387`. Newer CPUs have support for vector instruction, which may speedup most complex math algorithms.

Tips

- Test your code, before choosing an optimization type. CPUs have cache memory that is many times faster than RAM memory. A compact code that fits into the cache will probably outperform a loop-unrolled code that has to be read sequentially from the RAM chip.
- The ideal option depends mostly on the nature of the executed code.
- It is not uncommon that code compiled with the `-O3` option crashes, because it uses riskier optimization algorithms that may fail in some cases.
- A good compromise regarding stability size and performance is the `-Os` and it is the recommended starting point, though some compiler do not support it. In this case, try `-O2` as default.
- As a general rule, optimized code cannot be debugged.
- gcc has a linker which is not used in LasalClass2. Therefore, the linker options described on the gcc website cannot be used.

Example

GCCOptions: `-O2 -march=core2`

ShowConnectionFileWarnings

If this flag is activated, the connection files are checked for validity (do servers and clients have a valid value, do the connections exist). If necessary, warnings are displayed.

MathLibrary

This setting can be set to **Legacy Library** or **BSD Library** and influences, among other things, the accuracy of mathematical calculations. For more information, see The MathBSD Library chapter.

ThisCallScope

This setting is set to **true** by default and causes the warning **W 0172** to be output (see chapter **W 0172: Cannot resolve 'this' pointer for method 'NotMyBaseClass::MyVirtual'. Non-static methods will crash your program**). If this value is set to **false**, the warning is not output.

Directory

In **Directory**, the location of classes, sources, networks etc.. in their respective project folders is shown. Display only function - no changes can be made!

Advanced IOs

Options for the Advanced IO functionality.



Attention!

This item is only visible if the Use Advanced-IO property is set to **true**!

If you open this item by clicking on the plus sign, the following properties will become visible:

Default Network

This is a read-only field. The default network for the generated Advanced IO objects is displayed here.

PLC Path

Path on the PLC for the Advanced IO configuration.

Advanced IO

Setting for the **default Advance IO class**.

Simulation

Setting for the **default simulation class**.

Manager

Setting for the **default manager class**.

Load all libraries

True = all classes in the specified libraries are loaded, **False** = only the required classes are loaded.

Switching off (= **False**) this option can bring a considerable speed advantage, mainly for scripting, since the loading times are considerably reduced. However, **it should be used with caution**, since with complex dependencies the project may no longer be compilable. **The responsibility for using this optimization lies with the user**.

Use Unit System

If this flag is set to **true**, the LASAL CLASS 2 Unit System is used.

Include Paths

Here, the user enters the include paths, either directly using semicolons or by pressing the  button to manage them in the **Include Paths window**. This path is used in the current project. Settings for the complete LASAL CLASS 2 installation can be made under the menu item **Extras, Options** in the dialog box Application Options.

Include Paths simplifies the handling of includes in the code. With the instruction **#include <Myheader.h>**, every Include path is searched for the file specified in the project or tool.

Library Paths

Here, the library path is defined directly using semicolons or by pressing the  button to manage them in the **Library Paths window**. This path is used in the current project.

Settings for the complete LASAL CLASS 2 installation can be made under the menu item **Extras, Options** in the dialog box Application Options.

If a library in a defined library path is linked, the reference of the library in the project file is saved relative to the library path rather than project. A characteristic of ever library path specific to LASAL is that the projects in the library path and their sub folders appear in the library tree through the context menu and can be selected directly.

Backup Includes and Loader

If this option is set to **true**, the Includes and the loader are also saved during the project backup.

Ignore at cleanup

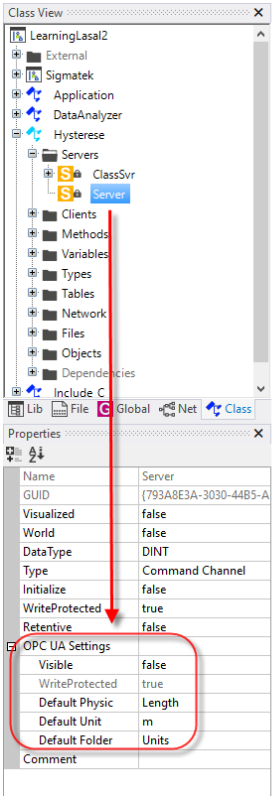
Here, files or entire folders can be entered that should be ignored with Project / CleanUp.

Reencrypt Project On Close

If this property is set to **true** (= default), the project is automatically encrypted when closed. The field cannot be edited. In the Protection Setting dialog, the value is controlled with the **Autom. Reencrypt Project on Close** checkbox.

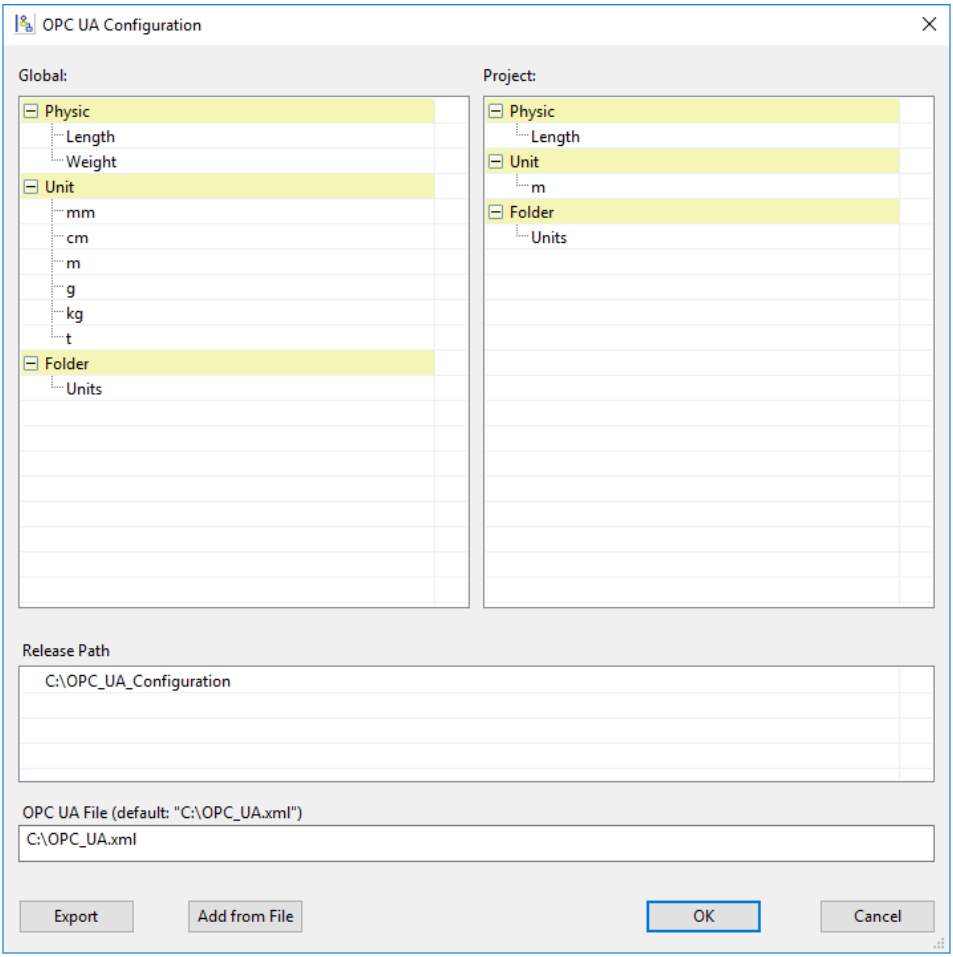
Enable OPC UA

If this value is set to **true**, the servers are assigned the properties group **OPC Settings**:



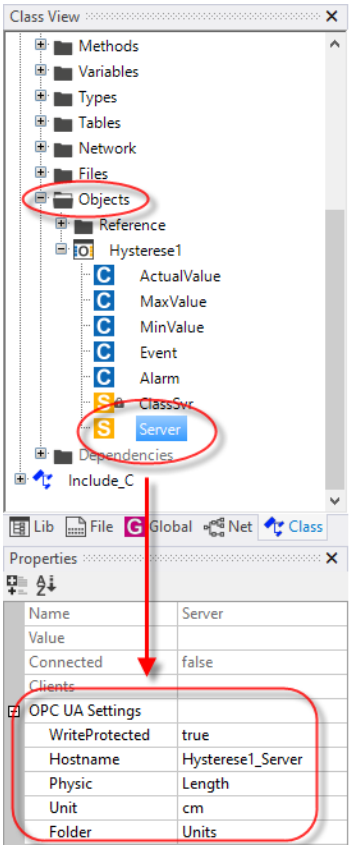
The OPC UA parameters are thereby downloaded to the control as their own *.xml file. If the **OPC UA Settings-Visible** server is set to **true**, the parameters of tis server are written in the *xml file. If this setting is false however, the server parameters are not included in the *xml file!

The values displayed here from **Default Physic**, **Default Unit** and **Default Folder** cannot be entered, but selected only. In the **Project menu**, there is a separate **OPC UA Configuration** menu item. The possible entries **can be** edited and managed in a separate dialog box.

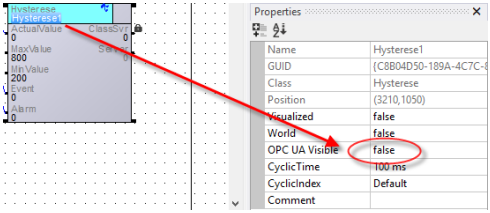


(For more information on this dialog box, see **Project Menu** – OPC UA Configuration).

The servers in the class (as is generally at the case in with classes and objects) only contain default values used for instancing new objects. **In an instance object, the default values** of the class (**Default Physic**, **Default Unit** and **Default Folder**) be overwritten.

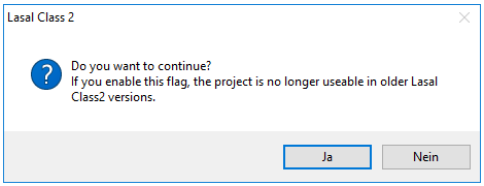


With **OPC UA Visible** = **false** for the **class object**, **all servers** of the object can be **made invisible** for the OPC server at the same time; regardless of Visualized settings in the individual servers. In this case, **nothing is written to the *.xml file** from these servers.

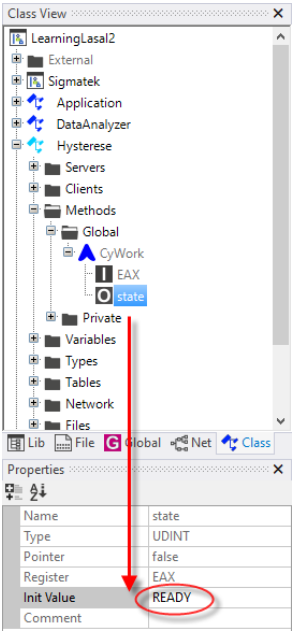


Enable initvalues for output parameters

Allows initialization of output parameters in standard methods. If the value is set to **true**, a warning is first given that older versions of LASAL can no longer be complied thereafter.



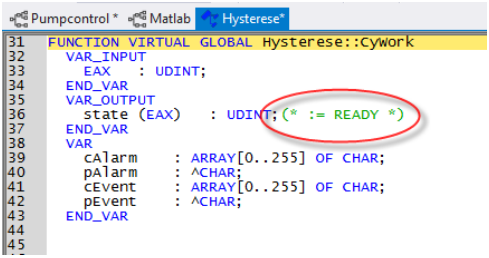
The output parameters of standard methods can then be initialized:



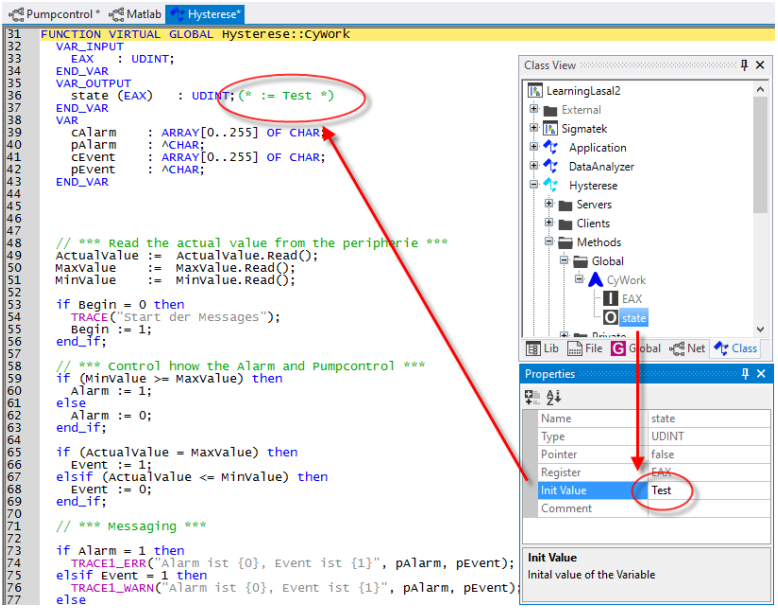
During the creation of a standard method, the code line required until now has been dropped.

```
state := READY;
```

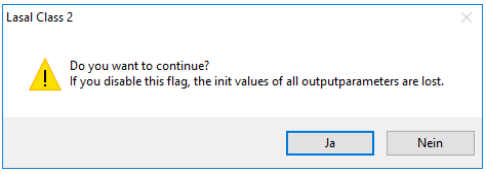
Instead, the Init variable is added to the declaration of the output variables as commentary:



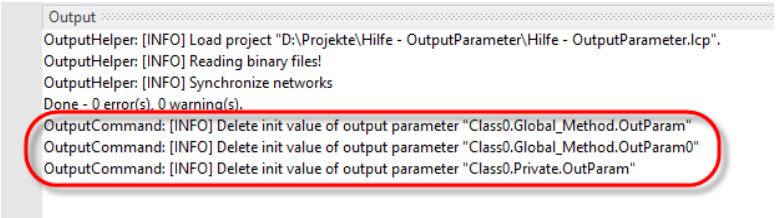
If the initial value of the output parameter has changed, the comment also changes automatically:



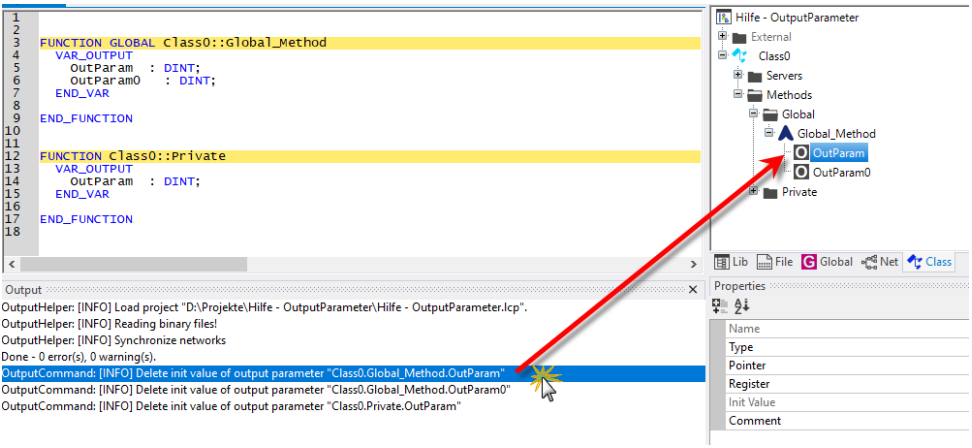
If the value is reset to **false**, a warning is triggered before the loss of user-defined Init values for the output parameters:



If you answer with **Yes**, not only are all self-entered init values for out-put parameters deleted, but all output parameters with deleted init values are listed in the output window.



If you double-click on such a line, the system jumps to the corresponding position in the tree:

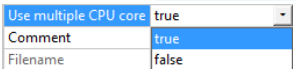


Use multiple CPU core

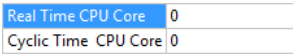
For use with 2 CPU cores for the realtime or cyclic tasks, this property must be enables in the property browser of the project

Attention!

This property is only displayed if the project **already contains a multi-core capable CPU** (e.g. CP731)!



After activation, the CPU core in objects with a realtime or cyclic task can be set to 1 or 0.

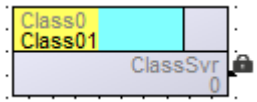


Attention!

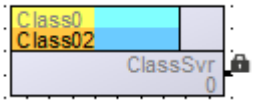
In objects of HW classes, the conversion is not possible since the CPU core used is automatically assigned through the hardware classes!

In the network view, objects with tasks in CPU core 1 are displayed in a different (darker) color

CPU core 0:



CPU core 1:



Attention!

It should be noted that objects whose tasks run on different CPU cores have no dependencies, since the tasks run parallel!

Use Advanced-IO

If this setting is **true**, the Advanced IOs item with all subitems becomes visible. By switching on this property **Advanced IOs including the Force functionality can be used** in the project. With the **true** setting, the conversion starts after a query, whereby the required classes as well as the corresponding Advanced IO or simulation objects are created in an Advanced IO network. With the **false** setting, the project is converted back in the same way after a query, and the Advanced IO network together with all objects and classes will be deleted - only simple IO objects can be used in the project. For more detailed information see the chapter Advanced IO functionality

Comment

Short commentary can be written directly in the line. For longer or structured commentary, it is recommended that the commentary editor be activated using the  symbol. Here the structured commentary can be entered with word wrap or tabulation. If commentary is available for an element, the structured commentary appears in the commentary window, which is always located at the lower margin of the property window. The size of the window can also be changed.



Note:

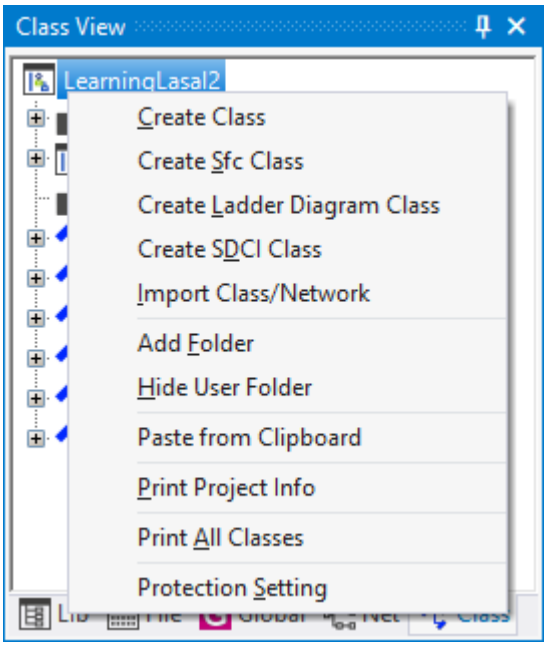
Contrary to LASAL 1, the Tooltip is no longer shown in the object network editor as a single line. Instead, it is available in multi line structuring.

Filename

With this property, the file name (including path) of the project is displayed and cannot be changed.

2.4.1.2 Context menu of a project

With a right click on project name, the following context menu appears:



Create Class

Creates a new class.

Create Sfc Class

Creates a new Sfc class (See chapter SFC).

Create Ladder Diagram Class

Creates a new ladder diagram class.

Create SDCI Class

Creates a new SDCI class. (see chapter Connection of SDCI Devices)

Import Class/Network

Imports a previously exported class or network.



Attention!

It is important to note that vertices are ignored during import.

Add Folder

Adds a user-defined folder, into which classes classes can be moved (grouped).

Hide User Folder

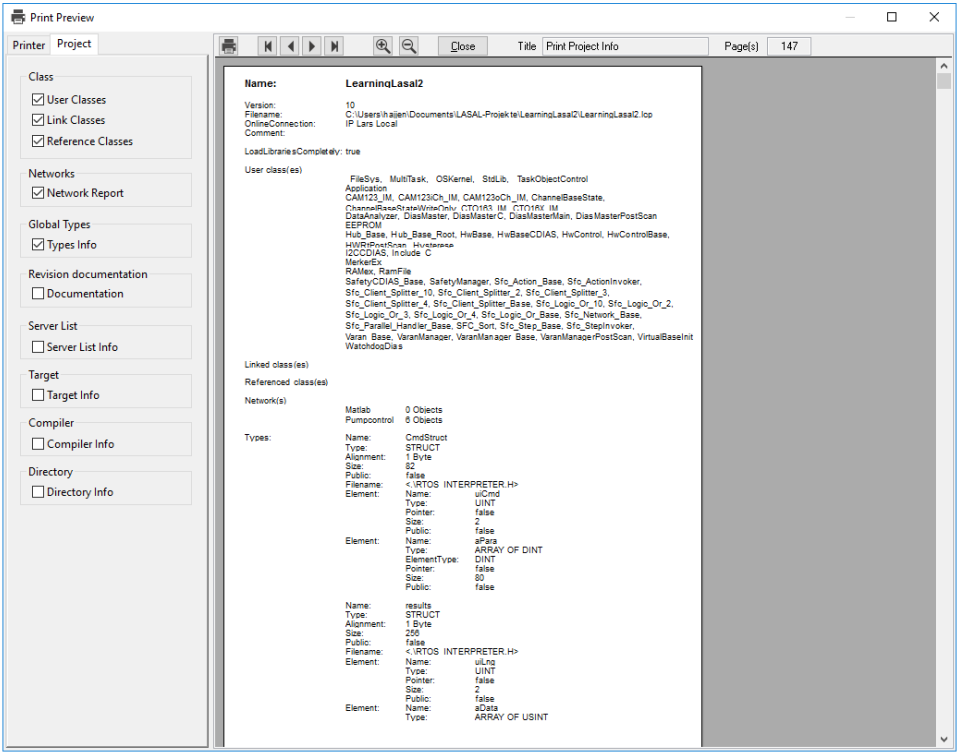
Hides the user-defined folder, in order to view all classes in a flat hierarchy. Displayed only if those folders are also available.

Paste from Clipboard

Inserts the classes from the Windows clipboard.

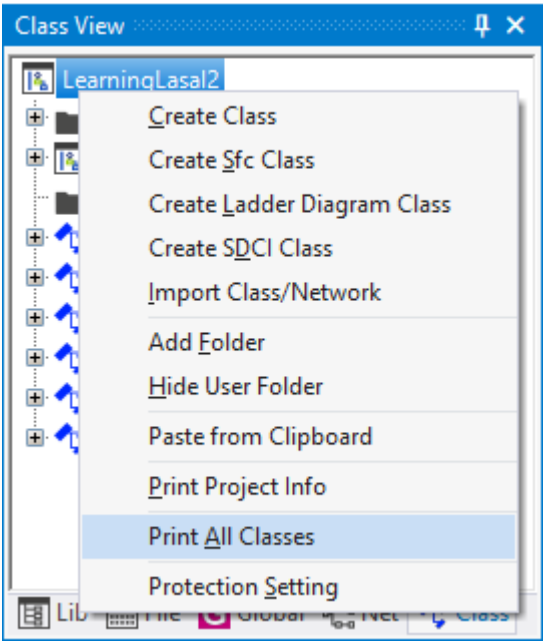
Print Project Info

Used to print all project information. Since – depending on the project – this can be a large amount (several hundred pages are not uncommon), the dialog that opens on the left side contains a series of options that can be enables and disabled:



Print All Classes

Unlike with **"Print Class"** – see menu item – here all classes of a projected can be printed at once.



The continued process (preview, etc.) is the same as the other print functions.

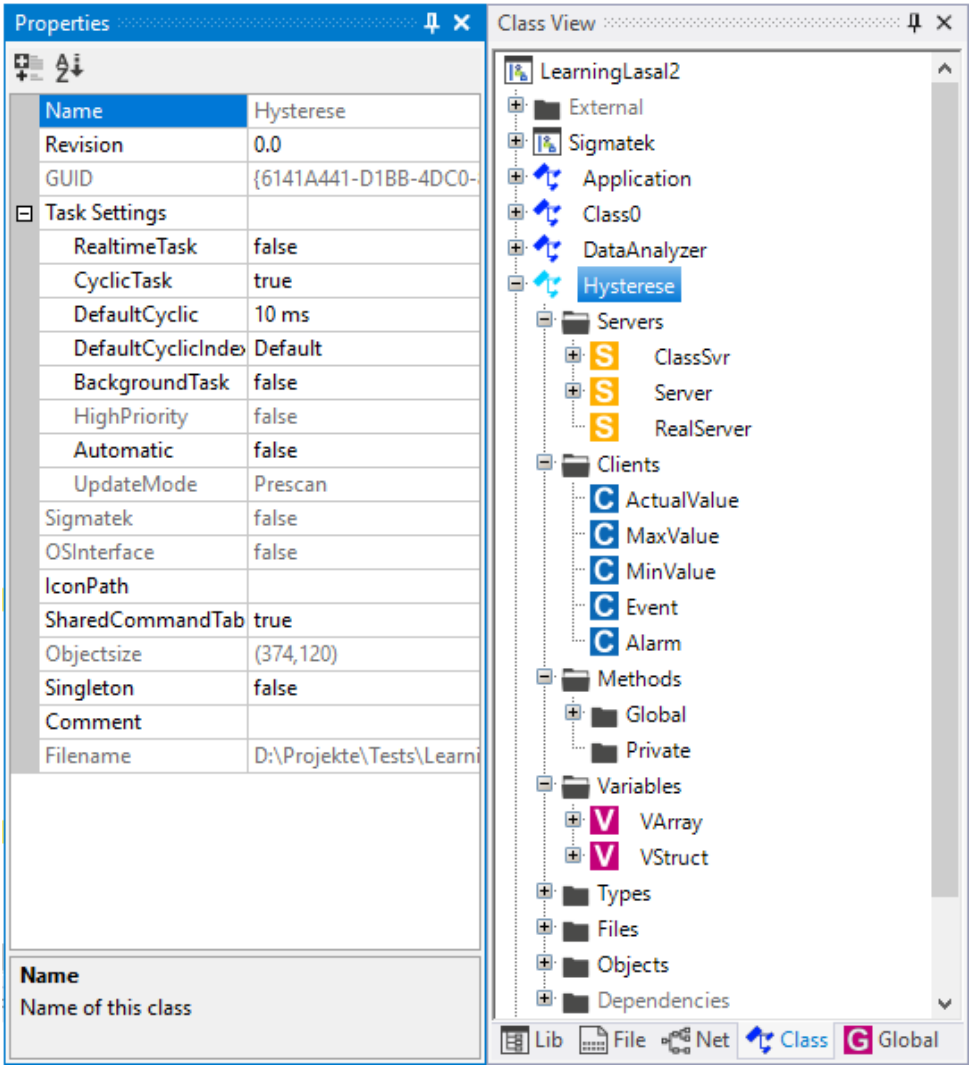
Protection Setting

Opens the dialog of the same name, in which classes can be protected from changes using a password or encryption. With the “Autom. Reencrypt Project On Close“ checkbox on the „Encryption“ tab, the project is automatically re-encrypted when closed.

2.4.2 Class Properties

2.4.2.1 Properties of a class

To view and change the properties of a class in the properties window, click on the class (here: **Hysterese**) in the **Class View window**.



Name

Name of the class (here: **Hysterese**).

Revision

Here, a revision number can be assigned, which for example, increases each time your class is changed. When the revision field is clicked, the More arrow is then visible.



Here, the **Revisions Documentation** can be accessed. For more information, see the chapter Revision.

Base Class

If the class is derived from another, the base class is displayed here.

Task Settings

RealtimeTask

Define a **Realtime Task** as **True** or **False** using the button. When **True** is selected the **DefaultRealtime** field is opened in which a predefined time for the RealtimeWork method can be entered. If an object is created, this default time is used.

More information on **Realtime tasks** can be found under Task configuration.

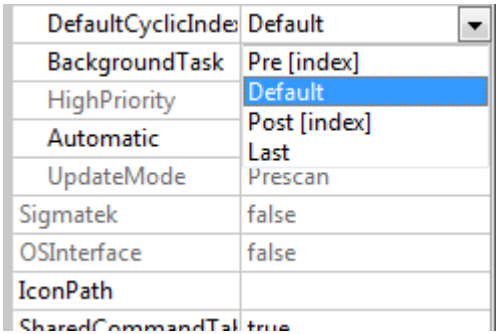
CyclicTask

This property is used to activate a **Cyclic Task** that can be set to **True** or **False** using the button. When **True** is selected the **DefaultCyclic** field is opened in which a predefined time for the CyclicWork method can be entered. If an object is created, this default time is used.

More information on **Cyclic tasks** can be found under Task configuration.

Default Cyclic Index

Indicates which cyclic index (Pre, Default, Post, Last) is used when creating an object.



BackgroundTask

This property is used to activate a **Background Task** that can be set to **True** or **False** using the button. When **True** is selected the **DefaultBackground** field is opened in which a predefined time for the Background method can be entered. If an object is created, this default time is used.

More information on **Background tasks** can be found under Task configuration.

HighPriority

If the class has a high priority, **True** should be selected instead of **False**. The button is used for this purpose.

This function should be set to **True** when the Automatic function is used. If HighPriority is activated, the CyWork method is called for all objects of this class with each kernel tick. With **False**, the interval between two CyWork method calls is 10 kernel ticks.

Automatic

This function is use, for example, with a hardware class. The Cy-Work method of this class is then called periodically by the operating system. When and how often it is called must be specified with the PreScan-, PostScan- and HighPriority functions. The is used to set the Automatic function to **True** or **False**.

UpdateMode

Automatic must first be set to **True** in order to use the **PreScan** or **PostScan** function. The button is used to select **PreScan**. The **CyWork-Method** is now called during the

PreScan phase. The **PreScan** phase is called before all other **CyWork** methods. If **PostScan** is selected, the **CyWork** method is called during the **PostScan** phase. The **PostScan** phase is called after all other **CyWork** methods.

Sigmatek

Flag indicating a SIGMATEK class.

OSInterface

Flag indicating an operating system class.

IconPath

Here you can assign your class a symbol. You can insert the symbol path directly or just select it via the  button from the explorer.

SharedCommandTable

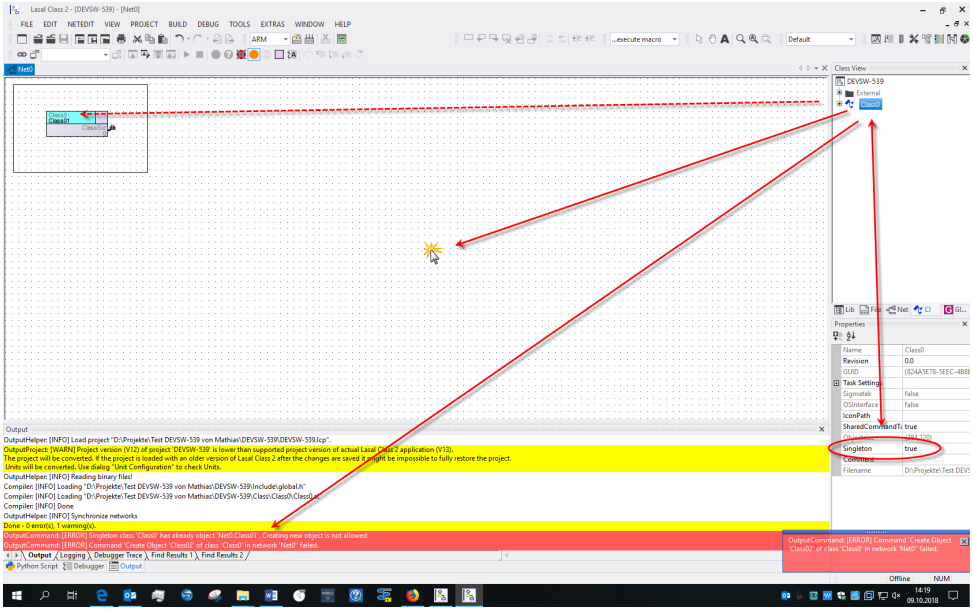
When this function is set to **True** using the  button, the operating system creates a **functions table** for all objects of a class. For special applications, it could be more sensible to call the **various functions** each object. In this case select **False**.

Objectsize

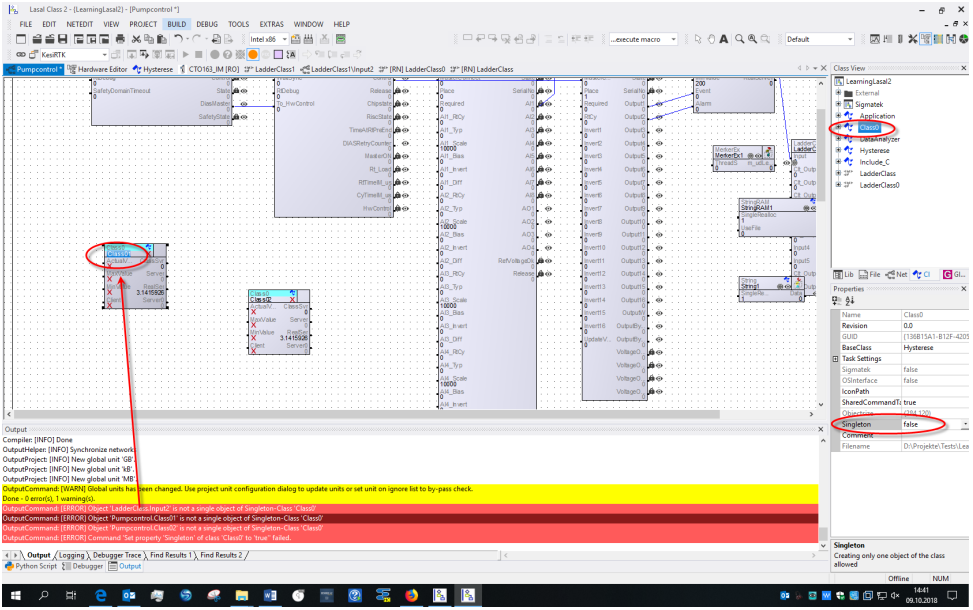
Size of the class object.

Singleton

This term comes from the development patterns and describes whether it is allowed to create multiple instances (objects) of a class (=**false**), or not (=**true**). If you still try to create a second object from a class with **Singleton** = **true**, the following error message is displayed:



If an attempt is made to set **Singleton** to **true** for a class that already has several instances (objects), **Singleton** is reset to **false** and an error message is issued for each existing object. If you double-click on one of these error messages, the object is marked in the object network and brought into the field of view:



Comment

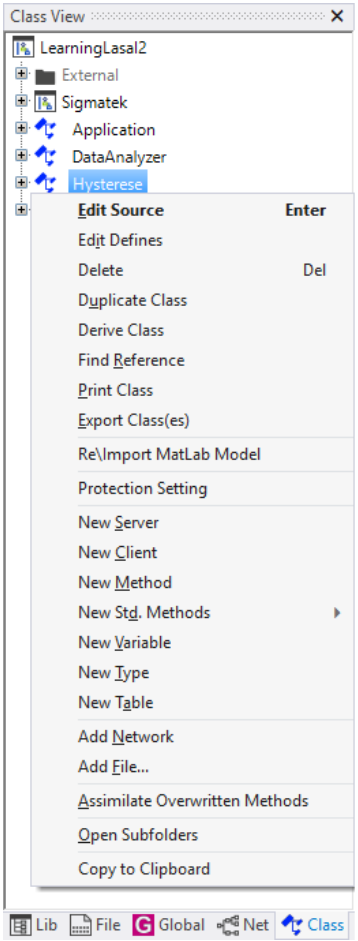
Use the button to open the input window and enter commentary.

Filename

Here, the file name (including path) of the project is displayed but cannot be changed.

2.4.2.2 Context menu of a class

Activate the context menu in the new class with a right click.



In this menu, all entries are now active and are listed as follows:

Edit Source

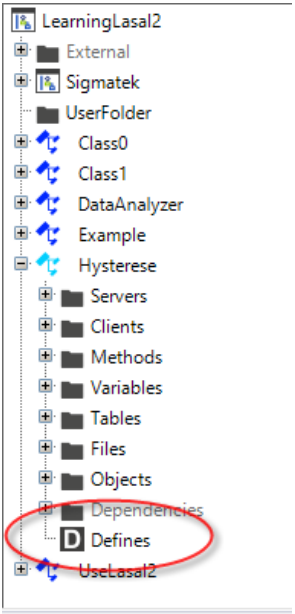
Shows the class code in the program code editor.

Edit Defines

With the Edit Defines command, a special section in the header of the St file is opened in the program code. Here, constants can be entered in the ST-Syntax (#define MyKonst 3) that can be used within the class.

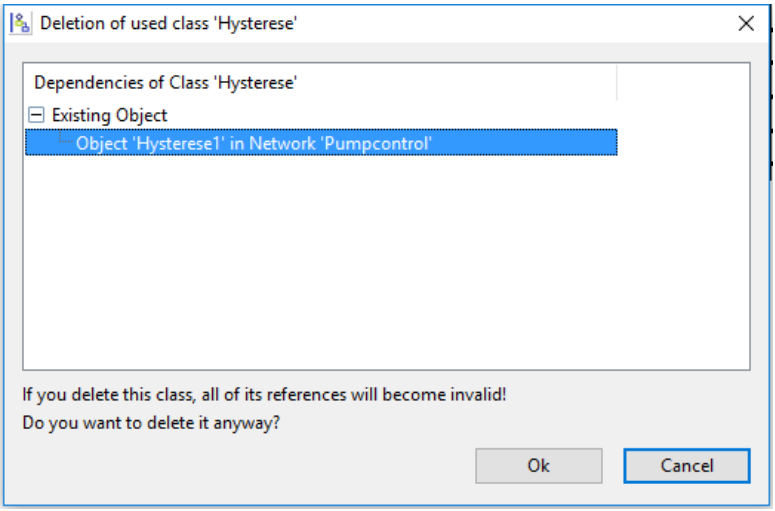
The advantage: an extra header file does not have to be created and linked for several less class-specific constants.

If something is already entered here, there is a new sub element of the class ("Defines").

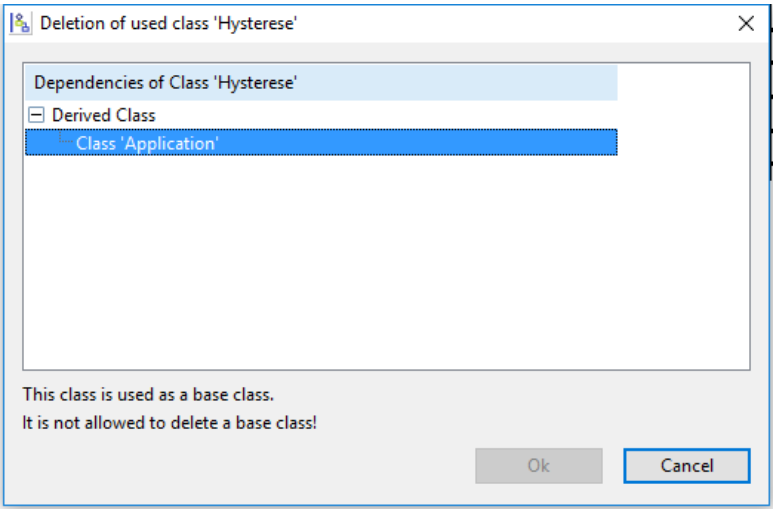


Delete

Deletes the class. If objects or references (e.g. object channel) exist, a dialog box with a list appears for he user to decide which to delete.



If the class to be deleted is a base class, LASAL does not allow the deletion.



Duplicate Class

Duplicates the current class.

Derive Class

Creates a derivation of the selected class.

Find Reference

In the FindResults window, all calls to methods in other classes are listed.

Print Class

Opens the print menu.

Export Classes

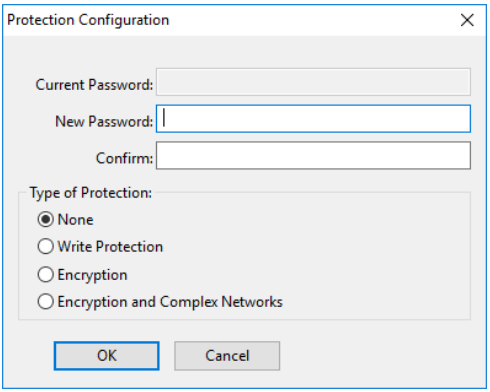
Exports the selected class.

RE / Import Matlab Model

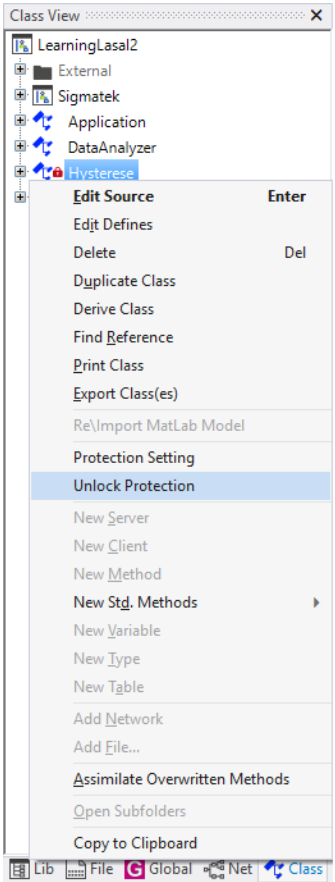
Imports a C file exported from MatLab Simulink.

Protection setting

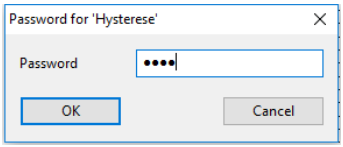
Enables a class to be write protected or encoded.



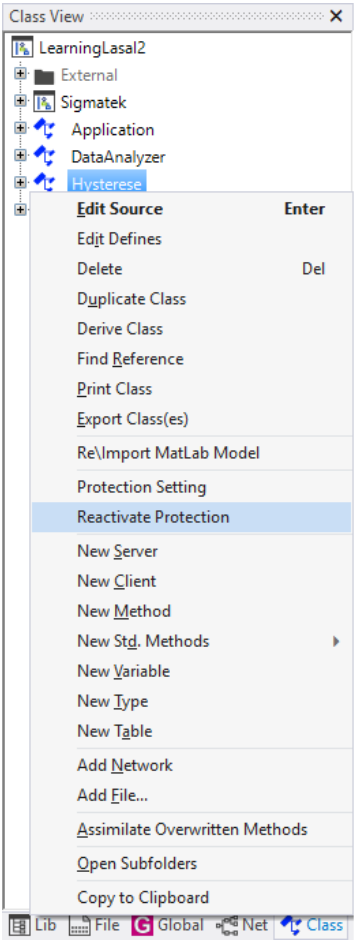
With encryption, the **only the implementation** is encrypted and **not the entire file!** If a project is password protected, a new menu item, "**Unlock Protection**" is listed in the context menu.



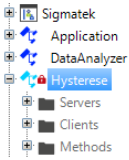
When this menu item is selected, the encryption can be temporarily disabled after entering the password.



From this point, the menu item "**Reactivate Protection**" is included under "**Protection Setting**". With this option, the password can be re-entered to activate the encryption.



An encrypted / password protected class appears in the class view as follows:



To permanently disable the protection, the **Protection Settings** must be opened and after entering the current password, the fields "**New Password**" and "**Confirm**" must be cleared.

New Server

Creates a new server for a class.

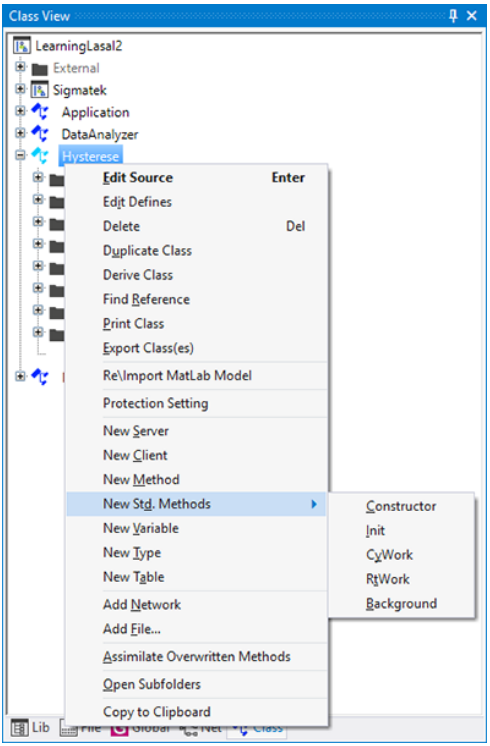
New Client

Creates a new class for a class.

New Method

Creates a new method for a class.

New Std. Methods



Select the **Constructor**, **Init**, **CyWork**, **RtWork** or **Background** to create these methods. Just click on Methods of a class to get more information.

New Variable

Creates a new variable for a class.

New Type

Creates a new class related type.

New Table

Creates a new table for a class.

Add Network

Creates a reference network for a class. With this instruction, the class is made into a complex class. In the properties window, the class is then defined as a base class or it could be placed in objects embedded in the reference network. A double click on the reference network of the class in the tree opens the network editor. In the editor bar, the character "**[RN]**" is inserted before the class name to indicate the **R**eference **N**etwork of a class is opened and not that of an object.

Add File...

Open the explorer to select and add a new source file, header file, *.VOV file or help file. More information can be found under Source code files.

Assimilate Overvritten Methods

This menu entry is only available for **derived classes**.

If the declaration of a virtual method changes (new input variable etc.), the overwritten methods are changed automatically in the derivations. But in some cases this is not possible, because e.g. the base class is linked.

For such cases, the method can be adjusted manually with this menu entry.

Open Subfolders

Opens all subfolders in a class. All members of this class (servers, clients, methods, variables, …) are then visible at a glance.

Copy to Clipboard

Copies the class into the Windows clipboard.

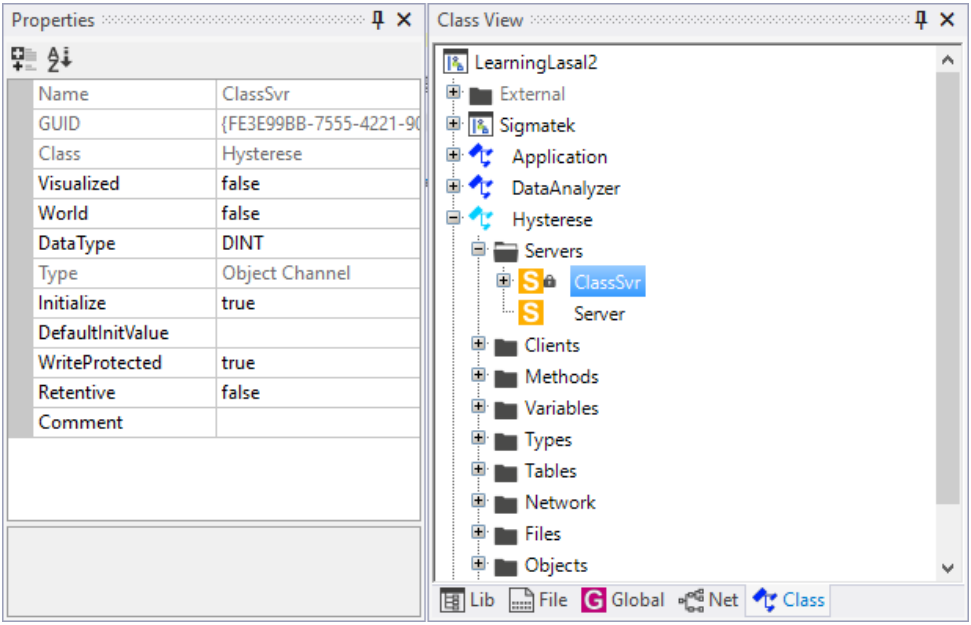
Macros

Runs the created Macros with the Macro manager Only visible when Macros are actually available!

2.4.3 Server Properties

2.4.3.1 Properties of a server

Click on the server to display the server properties in the **Properties window** in the **Class View window**.



Name

Server name (here: **ClassSvr**).

Class

Use the  button to select the **sever** of the corresponding **class**.

Visualized

With **True**, the server is displayed in the **LSE Data Server list**.

World

With **True**, this server is visible in the **Machine Manager**. But here it is necessary, that an object of the class exists in the network. In this object the property **World** also must be **true**.

Alarm

Only **SIGMATEK classes** have this property and as a default it always is **false**. It can only be set in the classes **AlarmX0Para** and **AlarmXBuffer** (can be found under Tools\Visualisation\LseRtk\Alarm\PLC\). If it is **true**, the server is written to the Maeexp file as an alarm and exported in the LSE.

DataType

Click on the  button to select the correct data type. A detailed list of all LASAL CLASS 2 data types can be found under Data types.

Type

Define the type from on of the **3 server types**.

Data channel

The server type is used to access the data of a channel (object) over a Read or Write method.

Command channel

This function is used to execute instructions of a channel (object) through accessing standard methods. The first server of a class is a **Command Server**.

Object channel

The server must be linked with an embedded object of the class displayed. The first server of a class is always an object channel of its own class.

Initialize

Using the  button, set this function to **True** to open the **DefaultInitValue field**. In this field, the **initial value** for the server can be assigned. When an object is created, the server will contain this default value.

WriteProtected

Set the write protection to **True** or **False** using the  button.

Retentive

This property determines whether the server should be stored retentively (meaing zero-voltage protected) or not. There are 2 options for saving available: **Sram** or **File**. If **Sram** is selected, the Server is saved in the battery buffered SRAM. If **File** is selected, the server is saved in a file in the root directory on the Compact Flash card of the PLC. Saving in Sram has an advantage of speed. This entry is selected when one or more servers should be saved frequently. Because the lifespan of a CF card can be greatly reduced through frequent writing, **File** should be selected when the server(s) must be saved seldom or only once (i.e. for inits).

To accelerate the retentive function, whether the actual value in the server is different from the value that should be written with the write method is internally compared. This means the when the server value is changed BEFORE it is written, no change is detected and the server value is therefore NOT saved!

Example for **NOT** correct programming:

```
MyServer := MyServer + 1;
MyServer.Write(MyServer);
```

Example for **CORRECT** programming:

```
myValue := MyServer + 1;
MyServer.Write(myValue);
```

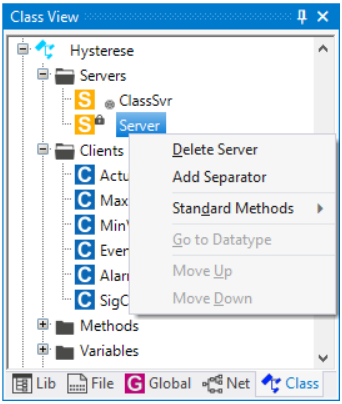
In this case, a change is detected in the server value is stored.

Comment

Use the  button to open the input window and enter commentary.

2.4.3.2 Context menu of a server

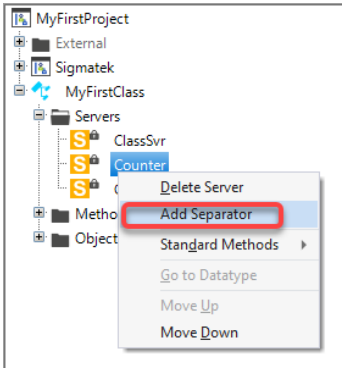
Click on a server (here: **Server**) with the right mouse button to open the context menu:



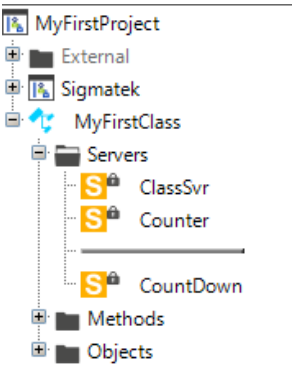
The first server of a class (ClassSvr by default) **cannot be deleted**! At this server, the menu item **Delete server** is therefore grayed out.

Add Separator

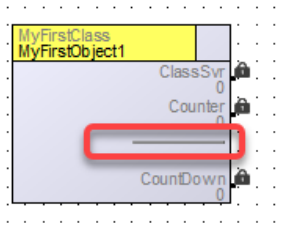
The "**Add Separator**" menu item...



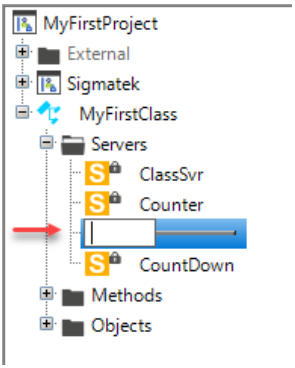
inserts, as the name suggests, a separator line at the marked position:



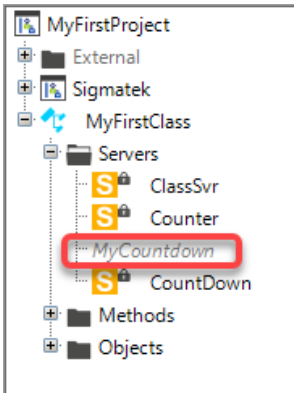
If there are already objects of this class in the network, one is also inserted there at the corresponding position:



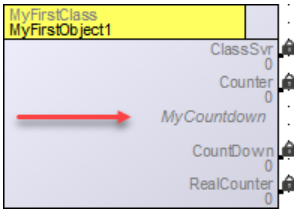
If you select a separator line in the project tree and press the F2 key, you can enter text in a field...



...which replaces the separator line:

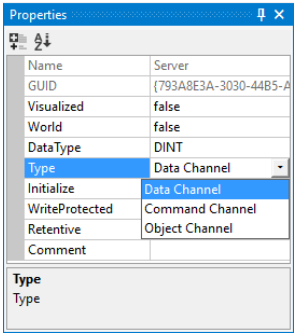


Similarly, you can see the name in the network object:

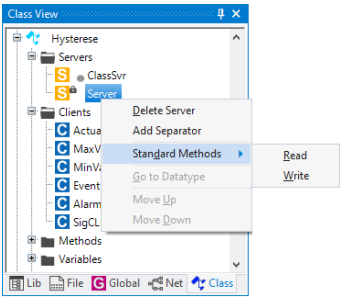


If you edit the name in the project tree again and delete it, the separator line is displayed again.

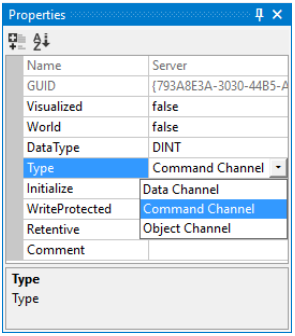
The context menu for **Standard Methods** depends on the channel in the **Properties window**.



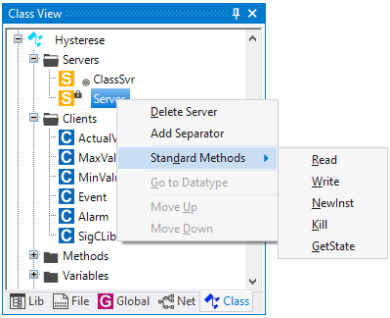
If the **Data Channel** is selected as shown above, the standard methods context appears menu as follows:



The context menu contains the **Read** and **Write** only.



If the **Command** or **Object Channel** were selected, the standard methods context menu would list three additional methods as shown below:

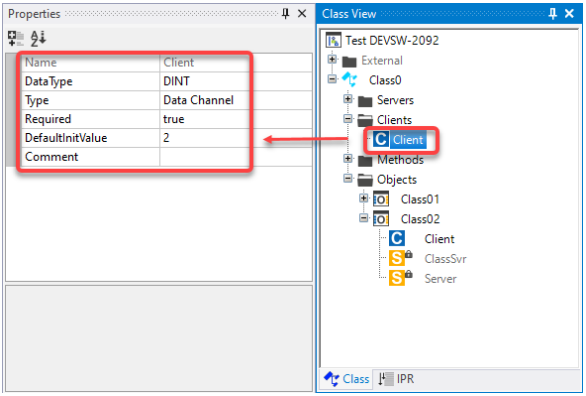


Delete Server	The selected server is deleted.	
Add Separator	Inserts a separator line at the current position. See above for description: AddSeparator	
Standard Methods	Read	Returns the contents of a channel (**4-byte DINT**).
	Write	The contents of a channel are **overwritten** with the assigned value.
	NewInst	The **NewInst** function is called from the operating system when the interpreter code is processed. This function should execute the existing code and signals the interpreter to what extent the process was successful. If **NewInst** is selected, the "**Use Hidden NewInst**" property can be selected in the Properties window (default = **true**), which specifies whether the NewInst function template in the *.st file should be created in the Hidden area.
	Kill	The **Kill method** is called by the interpreter when it is ended (for longer NewInst instructions).
	GetState	The GetState method is called by the interpreter if lengthier **NewInst instructions** are processed. This instruction decides whether the next instruction should be executed.
Move Up	Shifts the selected server to a higher position in the server list.	
Move Down	Shifts the selected server to a lower position in the server list.	

2.4.4 Client Properties

2.4.4.1 Properties of a client

To display and change the properties of a client in the **Properties window**, click on the client (here: **ActualValue**) in the **Class View window**.

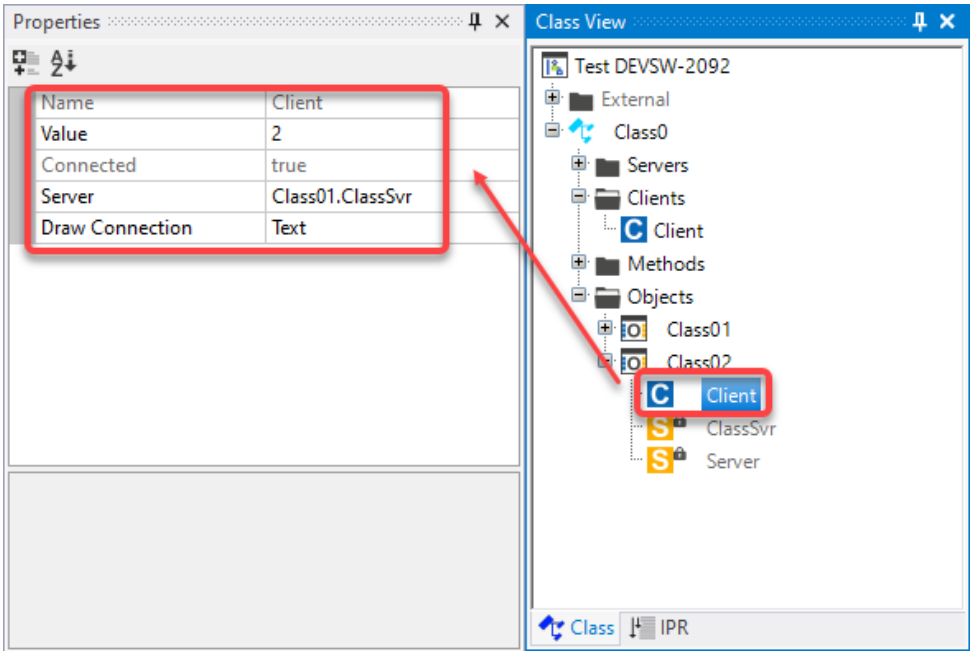


Name	Client Name (here: **Client**).
DataType	Click on the button to select the appropriate data type. A detailed listing of all LASAL CLASS 2 data type can be found under Data types.
Type	Select one of the **three client types** using the button.

Data channel	Clients of this type can be connected with any sever. They support calls to both read and write methods of a server, which are possible in 4-byte data transfers.
Command channel	These clients can only be connected to command serves. The support standard method calls. Connected servers can be initialized (INIT), send an instruction (NewInst) or hold (CyWork).
Object channel	These clients can only be connected to an object server of a class (or the derived class). This class can and/or must be selected under Class. Each public method of the class (including the standard methods) can be called. In addition, the public data (channels) of the connected object can be accessed.

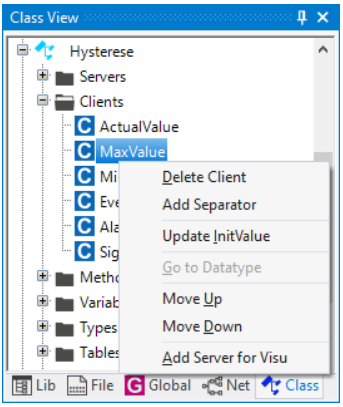
Required	Select True using the button to specify that a connection must be made to this client. If no connection exists or a client is not initialized, an error messaged appears in the output window.
DefaultInitValue	Here, an initial value is defined for the client.
Comment	Use the button to open the input window and enter commentary.

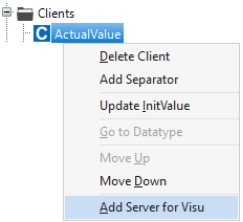
Other properties can be found by clicking on the **client of an already placed object** in the project tree



Name	Client Name (here: **Client**).
Value	Value of the Client.
Server	If the client is connected, here is the name of the connected server.
Draw Connection	The representation form of the network connection **to this single client in this network object**. There are three possibilities here: **Line** (connection lines), **Text** (name of the connected server as for connections to another object network) and **Default** (this is the standard value. Default is: **Line**).

2.4.4.2 Context menu of a client

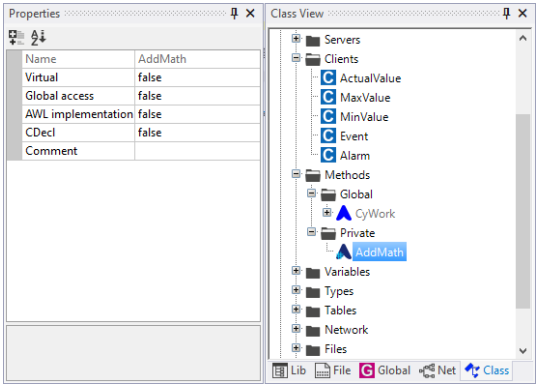


Delete Client	Deletes the client from the project.
Add Separator	Inserts a separator line, which can later be replaced by a text. For more information see Add Separator in the chapter Context menu of a server.
Update InitValue	If a default init value has been assigned for the client in the class, it will be subsequently applied to all existing instances (objects) of the class.
Go to Datatype	If the client is of a non-standard type, this menu item can be selected. It is used to jump to the type in the object tree.
Move Up	Moves the client one place upwards (can also be done with the mouse).
Move Down	Moves the client one place downward (can also be done with the mouse).
Add Server for Visu	Adds a server with the label ClientnameVISU. The READ method of the server is also transcribed and programmed This is an option to quickly add a server for the visualization that mirrors a client.The same context menu entry is also available for variables. 

2.4.5 Method Properties

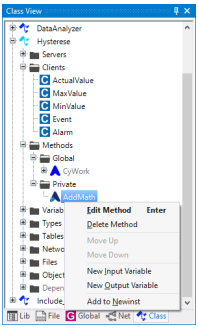
2.4.5.1 Properties of a method

Click on the method (here: **AddMath**) in the **Class View window** to display the method properties in the *Properties window*.



Name	Method Name (here: AddMath).
Virtual	Select **True** using the button if the method is virtual or **false** if not. Methods that are to be overwritten must be assigned as virtual (**VIRTUAL GLOBAL**) methods.
Global access	If a method should be accessed globally over an object channel, select **True** using the button. Select **False** if the method should be accessible to the class only.
Use Hidden NewInst	If this flag is set to **true**, it is provided as basic equipment in the Hidden area when creating the NewInst method.
Implement GetState	If this flag is set to **true**, a predefined GetState method is implemented in the hidden area with the NewInst function. (only in the properties of NewInst Methods)
AWL Implementation	Select **True** with the button if the method is to be programmed in AWL.
CDecl	Select **True** using the button, to call methods from a C function. You can read more about the interrelationships between the C programming language and ST code especially for Intel x86 CPUs in the chapter Compatibility between LASAL CLASS 2 and CDECL functions with REAL return values
Comment	Use the button to open the input window and write commentary.

2.4.5.2 Context menu of a method



Edit Method

Opens the method for editing in the code editor.

Delete Method

Deletes the method.

Move Up

Moves the method one step upward in the method list.

Move Down

Moves the method one step downward in the method list.

New Input Variable

Creates a new input variable.

New Output Variable

Creates a new output variable. Note: only one output variable is possible. If one has already been inserted, this menu item is disabled.

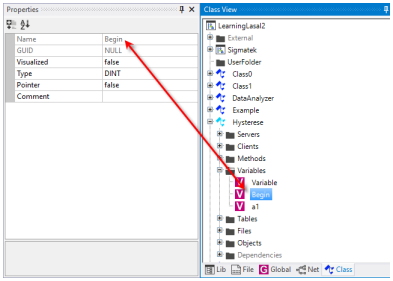
Add To NewInst

Adds the methods for the NewInst function of the ClassServer.

2.4.6 Variable Properties

2.4.6.1 Properties of a variable

To display and change a variable in the **Properties window**, Click on the Variable (here: **Begin**) in the **Class View window**.



Name

Variable name (here: **Begin**).

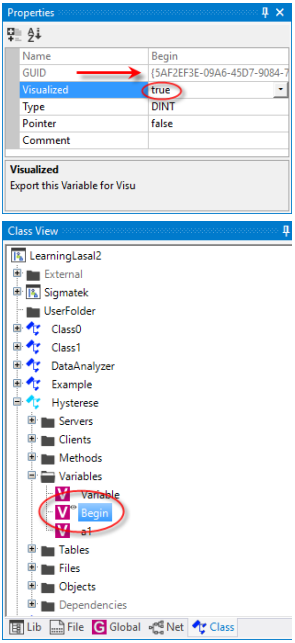
GUID

If the variable should be visualized (see next property), it gets a GUID, which is shown here. As a default Visualized is **false** and thus GUID is **NULL**.

Visualized

If this property, as well as the **project property Write visualized member variables** are set to true, the member variable is visualized. This also means, that they are exported via the Maeexp.txt file and read in the LASAL SCREEN Editor, where they are displayed. Variables, that are visualized, get an **eye symbol** in the tree.

There are limitations to the visualization of member variables – these are listed under the project property **Write visualized member variables**.



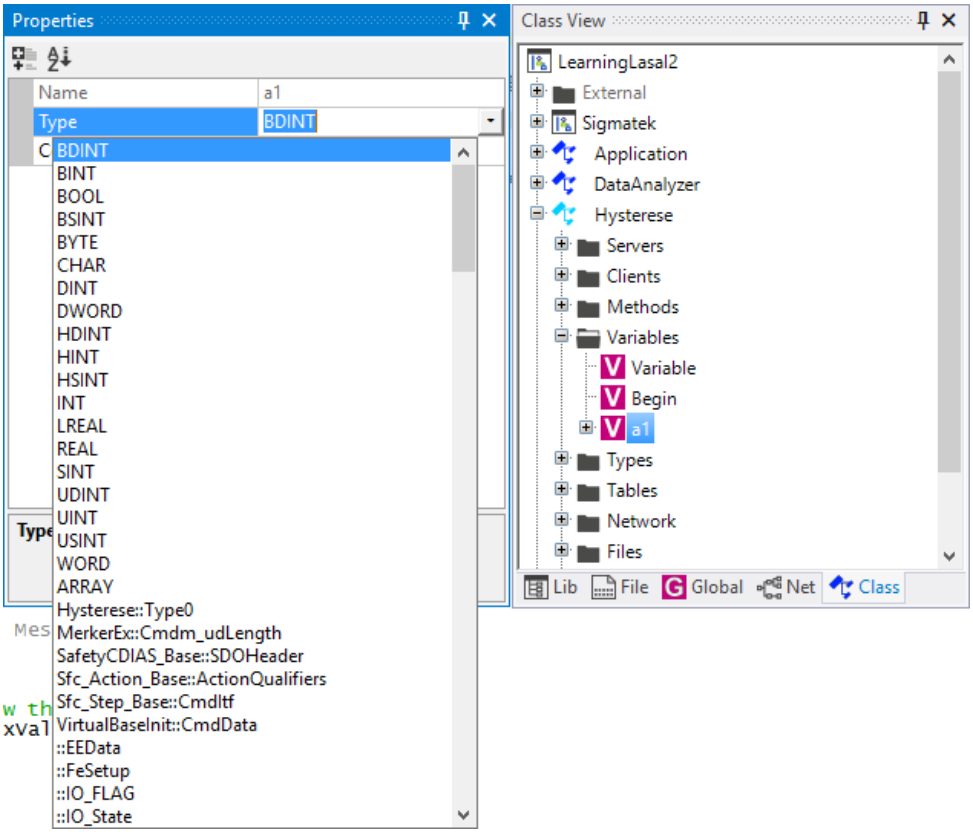
This property is only visible, if **Write visualized membervariables** is set to **true!**

Type

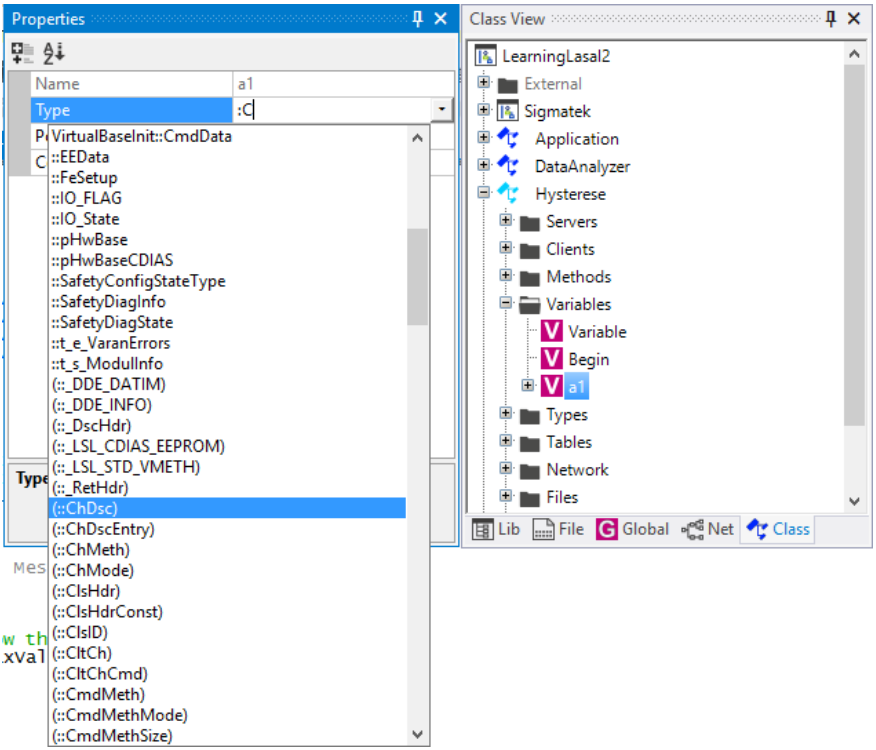
Click on the button to select the appropriate data type.

A detailed list of all data types in LASAL CLASS 2 can be found under Data types.

In the input field of the property, **Type**, it is enough to enter the first letter of the data type to select it.



When a letter of a data type is entered which occurs in the middle of the desired type (here: **C**), the combination **Ctrl+ Up/Down** every type containing the corresponding letter can be selected.



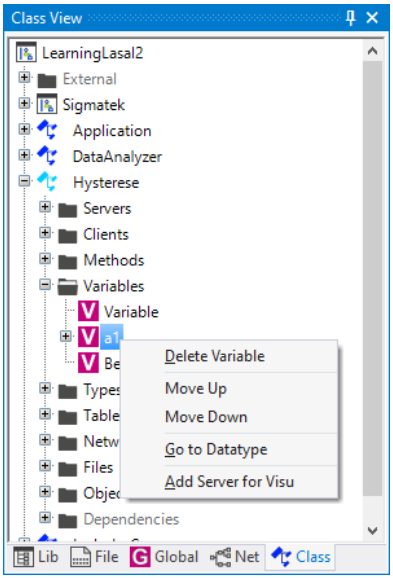
Pointer

Select **True** using the  to use a pointer or **False** when no pointer is to be defined. Pointers are variables that point to memory contents. Every Pointer requires **4-Bytes (Linear, Offset)**.

Comment

Use the  button to open the input window and write commentary.

2.4.6.2 Context menu of a variable



Delete Variable

Deletes the variable from the project

Move Up

Moves the variable one place upward (can also be done with the mouse).

Move Down

Moves the variable one place downward (can also be done with the mouse).

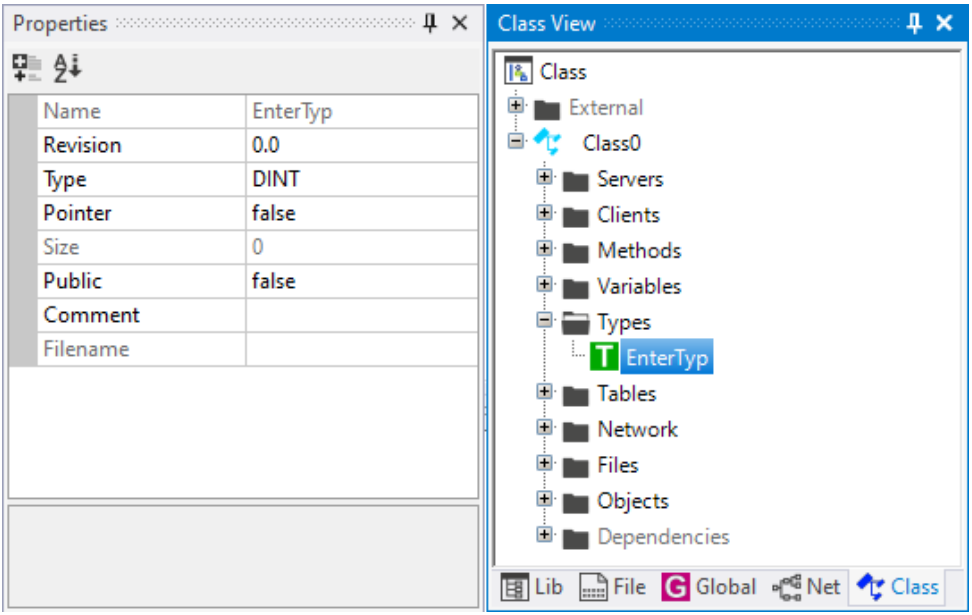
Add Server for Visu

Adds a server with the labelVariablennameVISU. The READ method of the server is also transcribed and programmed. This is an option to quickly add a server for the visualization that mirrors the value of a variable.

2.4.7 Type Properties

2.4.7.1 Properties of a type (class oriented type)

To display and change the properties of a class related type in the **properties window**, click on the type (here: **EnterTyp**) in the **Class View window**.



Name

Name of the type (here: **EnterTyp**).

Revision

Type revision number. Press the button to call the **"Revision Documentation of EnterTyp"**.

Type

Click on die  button to select the Data type. A detailed list of all data types in LASAL CLASS 2 can be found under Data types. Additionally to this data types it is also possible to enter a class name. After confirmation a pointer to this class is created automatically.

Pointer

Select **True** using the  to use a pointer or **False** when no pointer is to be defined. Pointers are variables that point to memory contents. Every Pointer requires 4-Bytes (**Linear, Offset**).

Size

Shows the size of a type in bytes but cannot be changed. This information is updated after a compiler process.

Public

If Public is set to **True**, this data type can also be used in other classes, whereby the class automatically creates a dependency to the class with the type. The data type is part of the class. That means, if the class is copied to another project, then also the data type is copied.

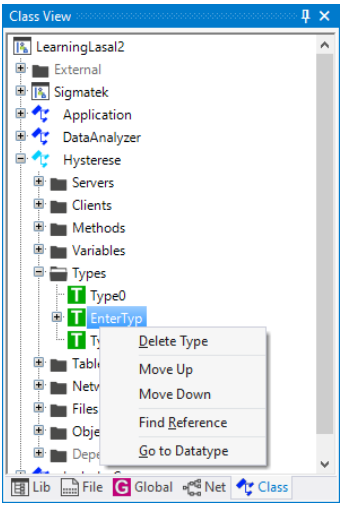
Comment

Use the  button to open the input window and write commentary.

Filename

With a class-related type, this field is type is always empty.

2.4.7.2 Context menu of a type



Delete Type

Deletes the type.

Move Up

Moves the type upwards.

Move Down

Moves the type downwards.

Find Reference

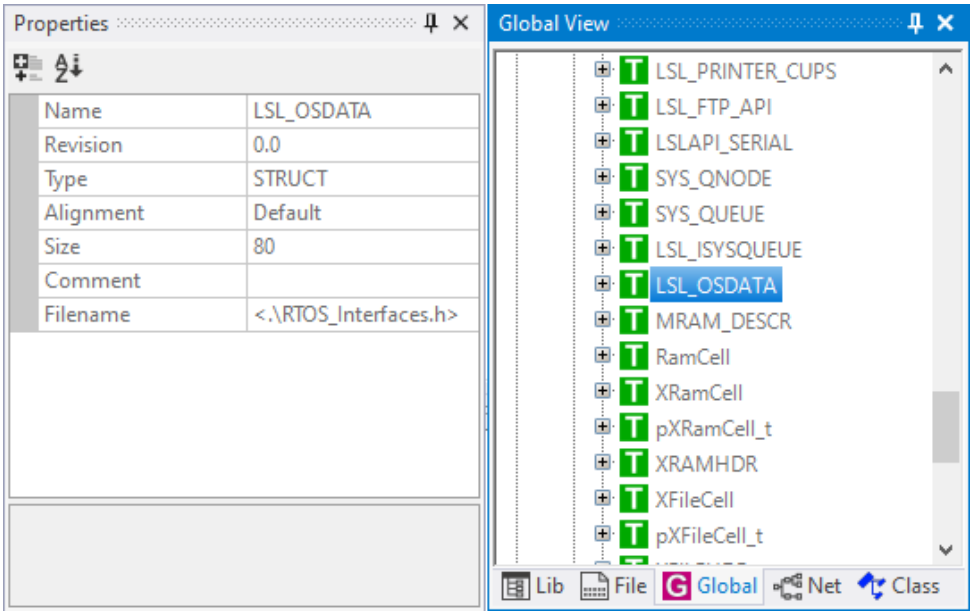
Finds occurrences of the type and displays them in the Find Results 2 window.

Go to Datatype

If the type is user-defined, this menu item can be selected and with a click, the marker jumps to the original type in the tree.

2.4.7.3 Properties of a type (global type)

To display the properties of a **global type**, the **Globals** tab must first be selected. Next click the global type (here: **LSL_OSDATA**) to define the settings in the **Properties window**.



Name

Name of the global Type (here: **LSL_OSDATA**).

Revision

Global Type revision number. Press the button to call the **"Revision Documentation of LSL_OSDATA"**.

Type

Click on die  button to select the appropriate data type. A detailed list of all data types in LASAL CLASS 2 can be found under Data types.

Pointer (not possible with every type)

Select **True** using the  to use a pointer or **False** when no pointer is to be defined. Pointers are variables that point to memory contents. Every Pointer requires **4-Bytes (Linear, Offset)**.

Size

Shows the size of a type in bytes but cannot be changed. This information is updated after a compiler process.

Comment

Use the  button to open the input window and write commentary.

Filename

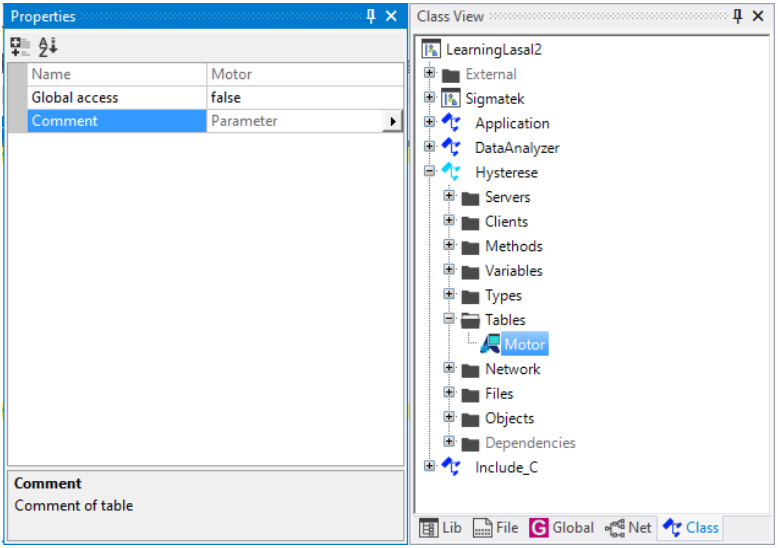
If the type is defined in a different (header) file, it is shown here.

The Context menu is identical with that of the local type.

2.4.8 Table Properties

2.4.8.1 Properties of a table

To display or change the properties of a Table in the **Properties window**, click on the Table (here: **Motor**) in the **Class View window**.



Name

Name of the Table (here: **Motor**).

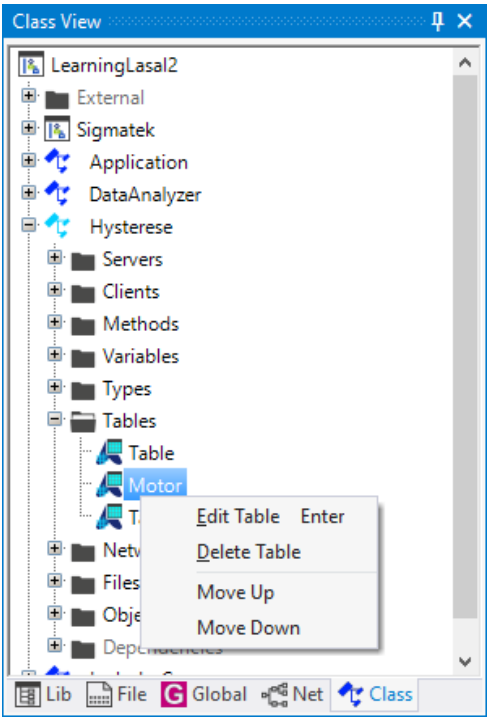
Global access

If the table should be accessed globally over an object channel, set this property to **True** using the button. Select **False**, when the table should be local to the class.

Comment

Use the button to open the input window and write commentary.

2.4.8.2 Context menu of a table



Edit Table

Here, the editor is opened and the table can be modified.

Delete Table

Here, the table is deleted.

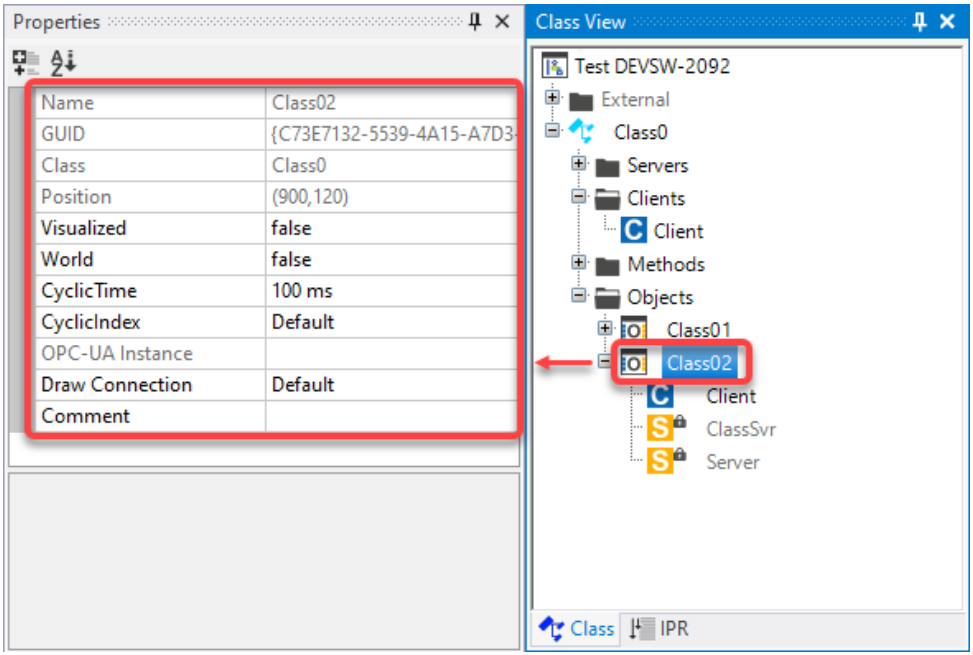
Move Up / Move Down

With this function the table, when several are exist, can be moved up or down.

2.4.9 Object Properties

2.4.9.1 Properties of an object

To view and change the properties of an object in the **Properties window**, click on the object (here: **Class02**) in the **Class View window**.



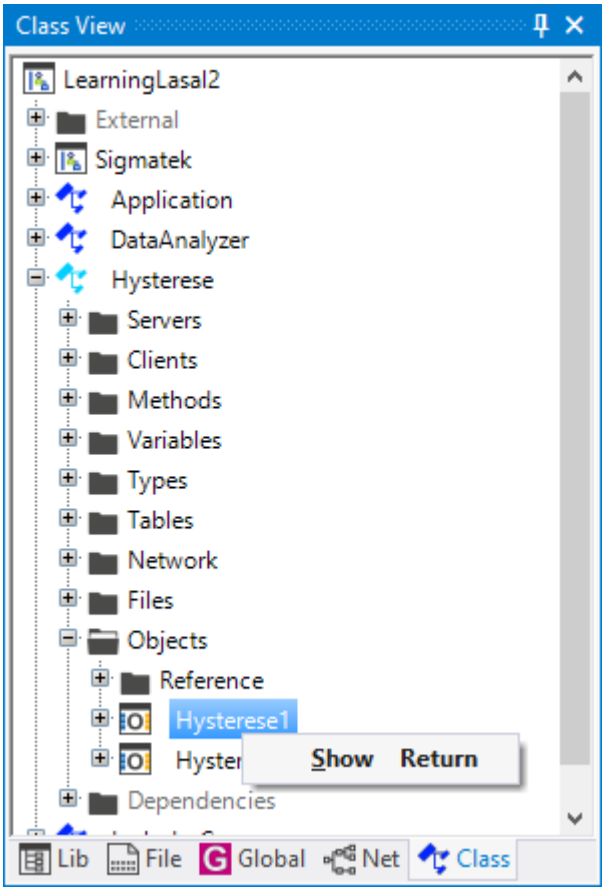
Name	Object Name (here: **Class02**).
Class	Object Class.
Position	Here the Position of the object in the workspace is shown.
Visualized	When set to **True**, this object is displayed in the **LSE Data Server list**.
World	With **True**, all the servers of this object, where the property **World** is also **true**, are visible in the **Machine Manager**.
CyclicTime	Here, the **CyclicTime** is set in ms.
CyclicIndex	In this field, the sequence of the call is defined. A CyWork method with an index of 1 is called before a CyWork method with an index of 2.
Draw Connection	This property determines how the network connections of the object should be displayed on the client side - with connection lines (**Line**), or with the display of the server name on the client, as is always the case with external network connections - i.e. connections to another network (**Text**). The third option is **Default** - the standard value is **Line**. This property affects the display of connections **at all** clients of this object. If you want to change the display **only on a certain client**, you can set this individually **after expanding the object** in the properties of the respective client. **Attention**: these are not to be confused with those of the client in the class! There the **Draw Connection** property does not exist, because **the representation of a connection** is not about the client of a class in general, but about that of a single (network) object.
Comment	Use the  button to open the input window and write commentary.



Attention!

Make sure that the background task has a minimum time of **2 ms**. The time for the background task cannot be set to less than 2 ms.

2.4.9.2 Context menu of an object



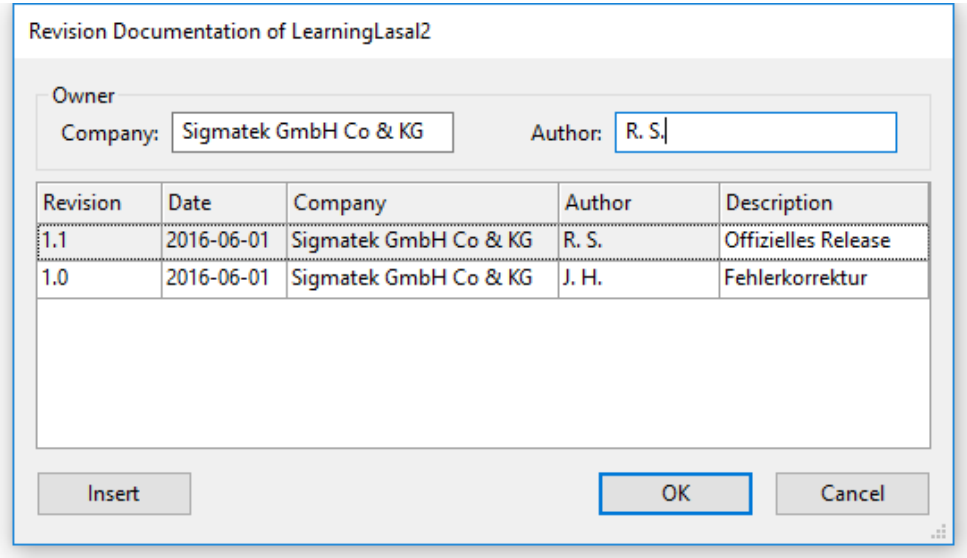
Show Return

If closed, this function opens the **network editor** and jumps to the object

2.4.10 Revision

For classes, object networks and the project itself, it is possible to define a Revision history. For each of the elements mentioned before the "**Revision**" property exists. Here, the revision number can be set. This number functions as a preset for new entries in the revision history (see below).

Via the More button in the respective property window the following dialog can be opened to edit the Revision history:



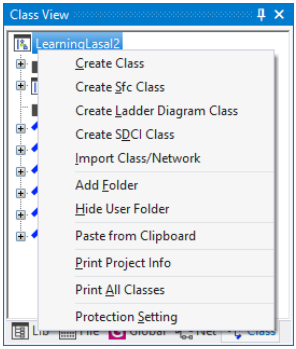
In the upper area of the dialog the owner of the software can be entered via company and author name. Additionally, a new Revision entry can be created with the **Insert** button. Thereby the latest changing stands on the beginning of the list. Revision number, Company and Author are automatically filled in based on the preset. Via the **Del key** on the keyboard the highlighted Revision entry can be deleted.

Properties to be entered are:

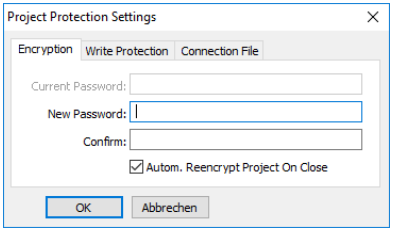
- Revision number
- Date of the changing
- Company and Name of the developer who made the changing
- Description of the changing (also possible over several lines)

2.4.11 Password Protection and Encryption of a Project / a Class

To protect or encrypt a project with a password, select **Protection Setting** from the context menu:



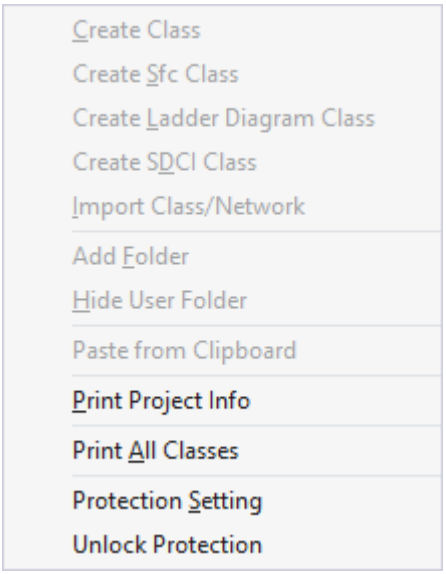
The following window appears:



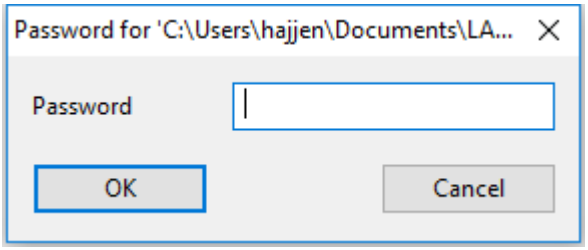
Here select **Encryption, Write Protection** or **Connection File** - depending on what to encrypt -, enter a password, and repeat it with **Confirm:** then press **OK**. Through the context menu, this window is opened to remove the security settings.

If the checkbox **Autom. Reencrypt Project On Close** is clicked (default), the project is automatically encrypted when closed.

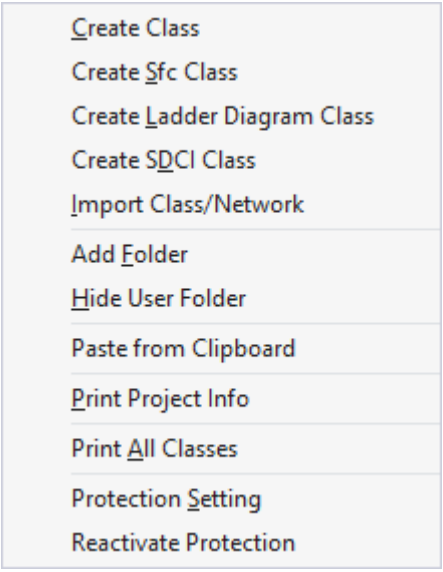
If a project is password protected, a new menu item, "**Unlock Protection**" is listed in the context menu.



When this menu item is selected, the encryption can be temporarily disabled after entering the password.



From this point, the menu item "**Reactivate Protection**" is included under "**Protection Setting**".



The protection can then be activated at any time without entering a password. To permanently remove the protection (no password), in the menu item "**Protection Settings**" under "**Current Password**", the current password must be entered and the "**New Password**" or "**Confirm**" fields must be cleared.

Individual classes, however, can still be protected / encrypted with a password. The only difference here is the visibility: If a class is encrypted, red padlock is shown at the start the class name. For the encryption of a project, no visible sign is given.

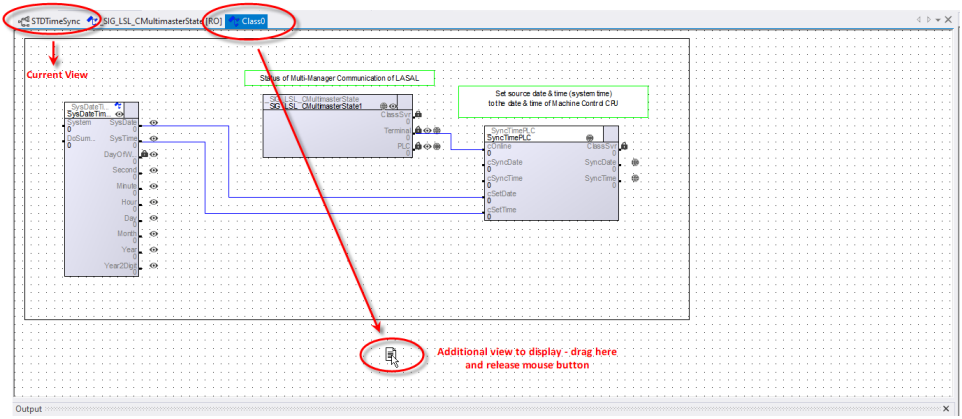
Using the third tab you can encrypt the connection file (*.lcf). It is also downloaded to the control in encrypted form. For more information on the encryption of Connection Files see the chapter The Project Connection Files Window.

If a project or class is password protected, the name is displayed with a gray background.

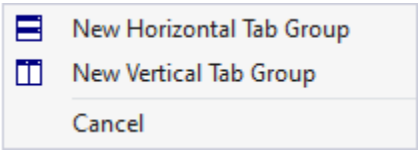
2.5 Work Space

In the workspace, the network view, hardware editor, one of the additional tools (such as the Data Analyzer or DIAS Drive) or an editor – graphic or text based, depending on the programming language – for one the programming languages (structured text, C LD, SFC...) can be displayed as desired. Here, the actual programming is done.

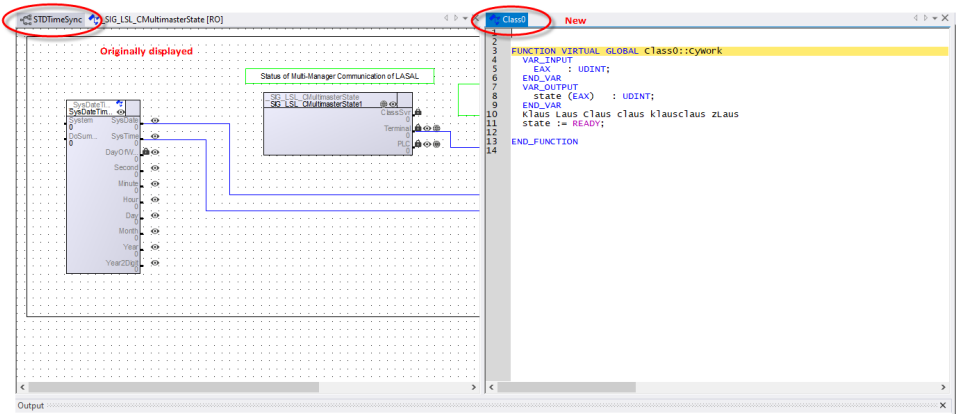
It is also possible to divide the workspace horizontally or vertically into groups so that several views (e.g. Code Editor and Network Editor) are visible next to or below each other. To do this, click the tab of the view to be displayed simultaneously with the right mouse button and drag it to the workspace while holding down the mouse button (Drag & Drop):



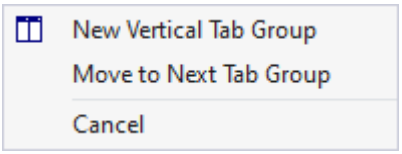
When the mouse button is released, a new context menu appears, in which you can decide whether the available area should be divided horizontally or vertically:



When you select Vertical Group, the desktop now looks like this:



If an additional tab is dragged onto the area (in our case e.g. the _SIG_LSL_CMultimasterState tab), a changed context menu appears, in which you can select whether the tab should be displayed as a new group (in our case as a third column), or moved to the already existing next group (in our case this is the group with Class0):



2.5.1 Code editor

2.5.1.1 Comfort functions

In the program code editor, the actual code is written. Here, the methods of the classes and objects, as well as the structured text code or integrated C files can be displayed and edited. Programming is simplified and/or supported by various comfort functions.

```
31  FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
32    VAR_INPUT
33      EAX    : UDINT;
34    END_VAR
35    VAR_OUTPUT
36      state (EAX)    : UDINT;(* := Test *)
37    END_VAR
38    VAR
39      cAlarm     : ARRAY[0..255] OF CHAR;
40      pAlarm     : ^CHAR;
41      cEvent     : ARRAY[0..255] OF CHAR;
42      pEvent     : ^CHAR;
43    END_VAR
44
45
46
47
48    // *** Read the actual value from the peripherie ***
49    ActualValue :=  ActualValue.Read();
50    MaxValue    :=  MaxValue.Read();
51    MinValue    :=  MinValue.Read();
52
53    if Begin = 0 then
57
58    // *** Control hnow the Alarm and Pumpcontrol ***
59    if (MinValue >= MaxValue) then
60      Alarm := 1;
61    else
62      Alarm := 0;
63    end_if;
64
65    if (ActualValue = MaxValue) then
66      Event := 1;
67    elsif (ActualValue <= MinValue) then
68      Event := 0;
69    end_if;
70
71    // *** Messaging ***
72
73    if Alarm = 1 then
80
81    Event.Write(Event);
82    Alarm.Write(Alarm);
83
84
85    state := READY;
86
87  END_FUNCTION
```

Folding (organically contracting and expanding related program components).

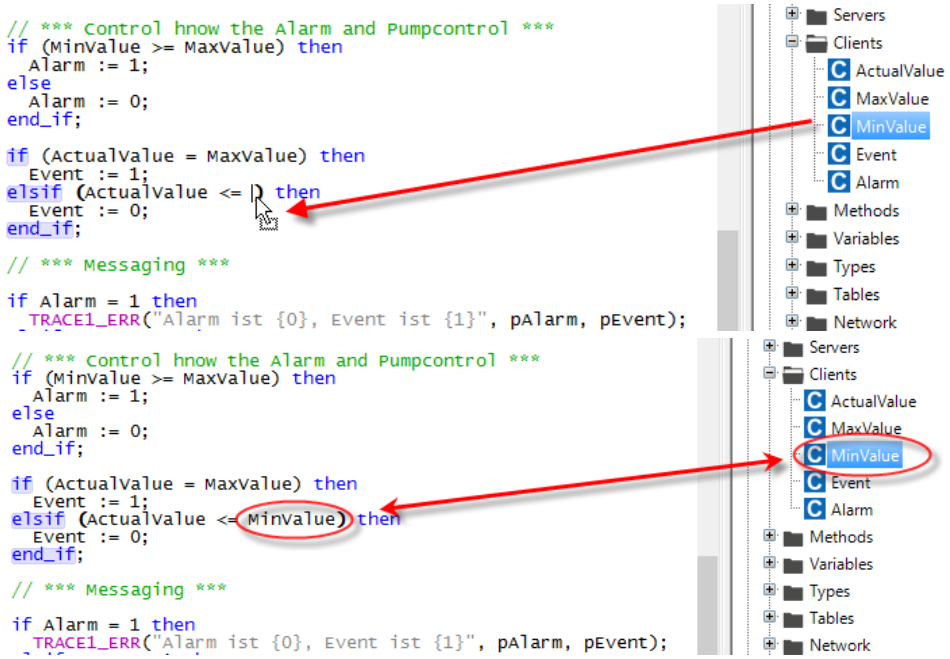
```
// *** Messaging ***

if Alarm = 1 then
  TRACE1_ERR("Alarm ist {0}, Event ist {1}", pAlarm, pEvent);
elsif Event = 1 then
  TRACE1_WARN("Alarm ist {0}, Event ist {1}", pAlarm, pEvent);
else
  TRACE("Alarm ist {0}, Event ist {1}");
end_if;

Event.Write(Event);
Alarm.Write(Alarm);

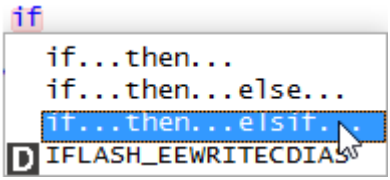
state := READY;
```

Syntax highlighting (color highlighting of keywords, commentary, constants etc.)



Drag and Drop from the class tree (helps to avoid typing errors)

2.5.1.2 Code completion

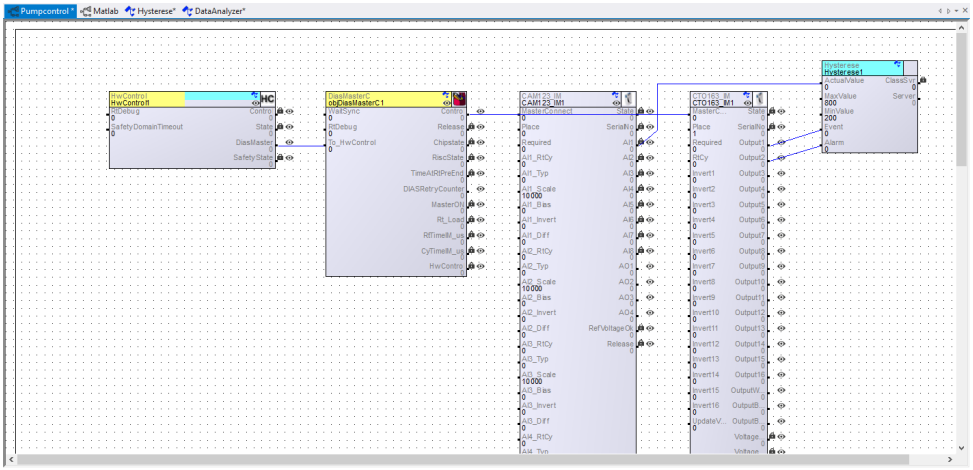


An especially helpful function of the program code editing is the code auto completion. When inputting code, the first two letters are sufficient and the LASAL Code Editor displays a list of keywords, functions, objects and variable names from which the desired entry can be selected with the cursor keys. If this includes a keyword such as "if" or "case", the multi-line framework of the instruction is created after confirmation – The user must only insert the missing parts (variables, code).

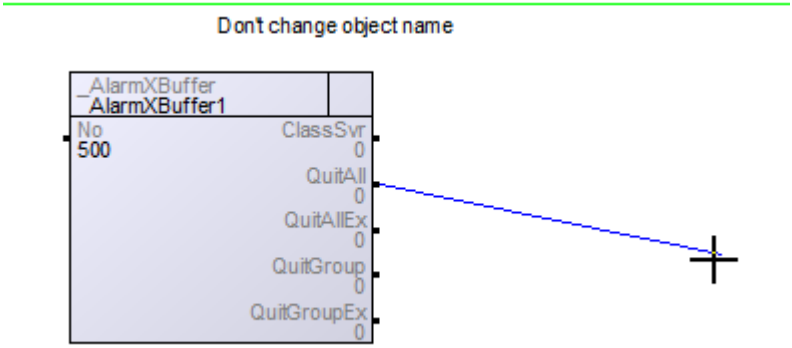
```
if  then
elsif then
end_if;
```

2.5.2 Network view

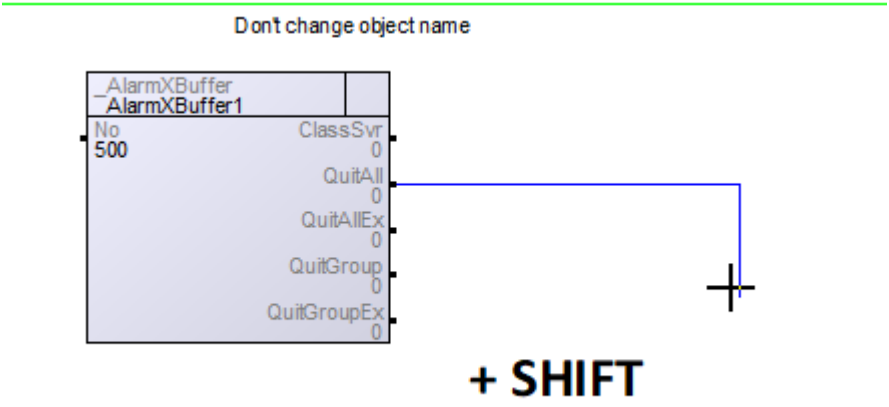
An essential component of working in LASAL CLASS is designing and managing object networks. The network view is the optimal tool for this purpose. Here, objects can be placed, moved, deleted or replaced, as well as connections made, changed and deleted.



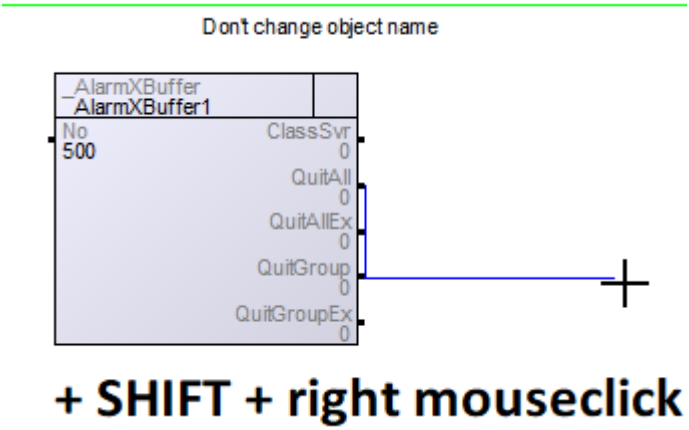
When drawing a connection, the shortest line is always drawn from the starting point to the next click:



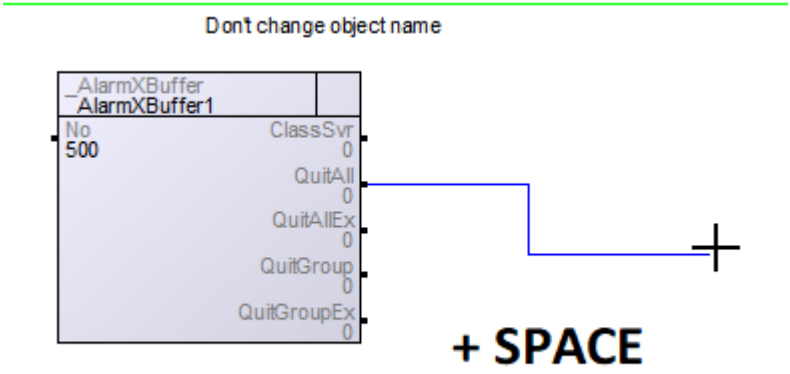
If you press the SHIFT key, the rectangle is drawn upwards:



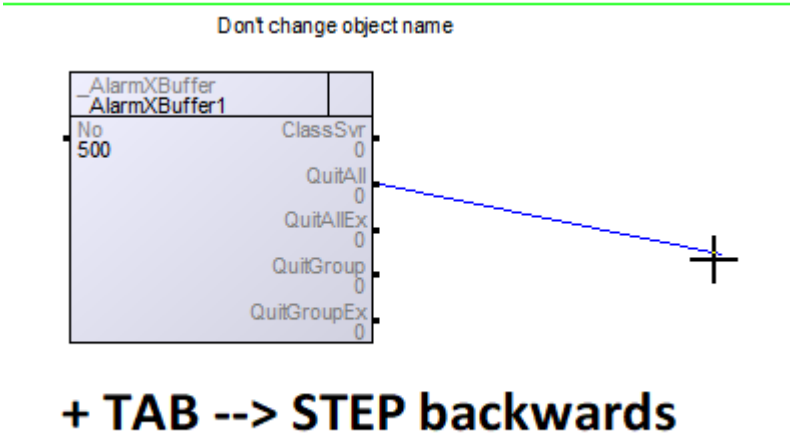
If you want to reverse the direction in which the corner was drawn, you must also press the right mouse button:



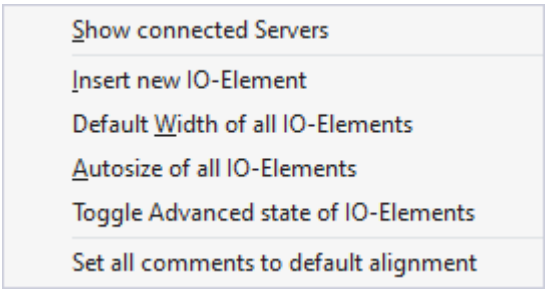
If you want to draw two rectangles at once, use the space bar:



After each change, the last step can be cancelled with the TAB key:



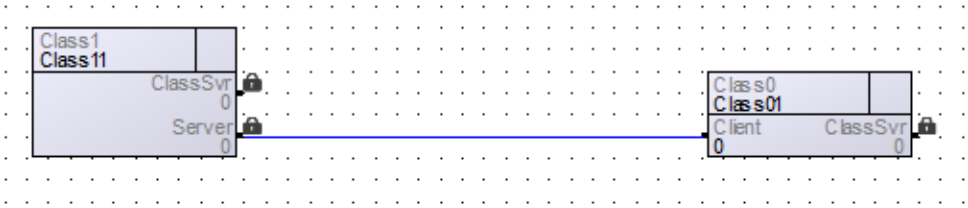
With a right-click on an empty space, an IO object can be created at that location, all IO objects can be set to standard width or autosize, or the connected servers identified by color (see context menu of network view).



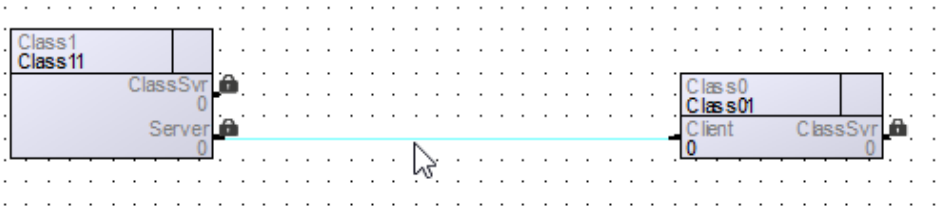
2.5.2.1 Vertices

Vertices

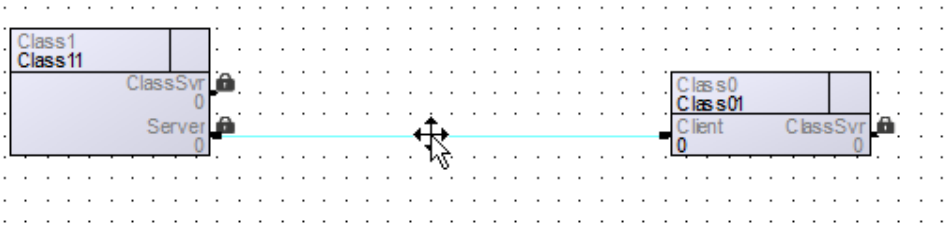
If a connection line between a client and a server already exists, it can be edited in many ways and as desired **using the mouse and additional keys**.



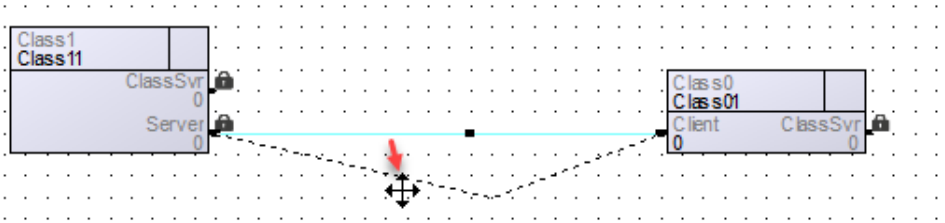
When you **move the mouse over a connection**, it turns **light blue**,



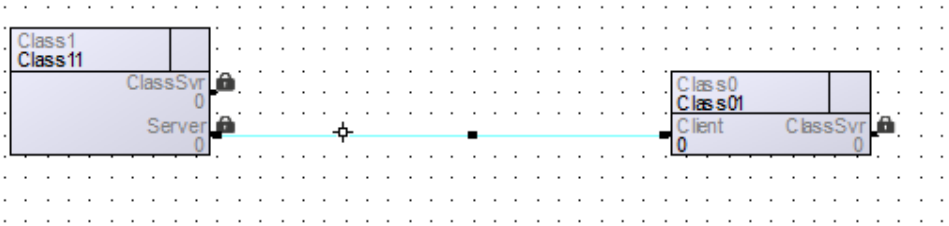
and after a left click the mouse pointer changes to the Move sign:



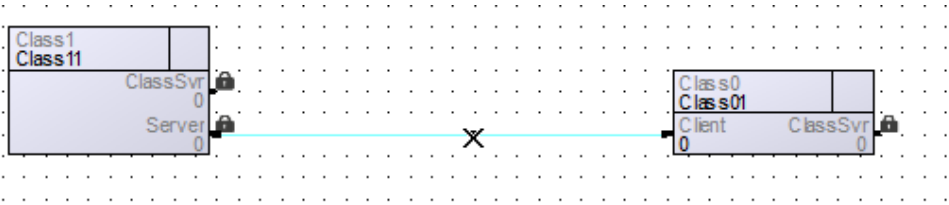
With the **left mouse button held down**, the entire link **can be moved** - provided there is at least one additional corner point.



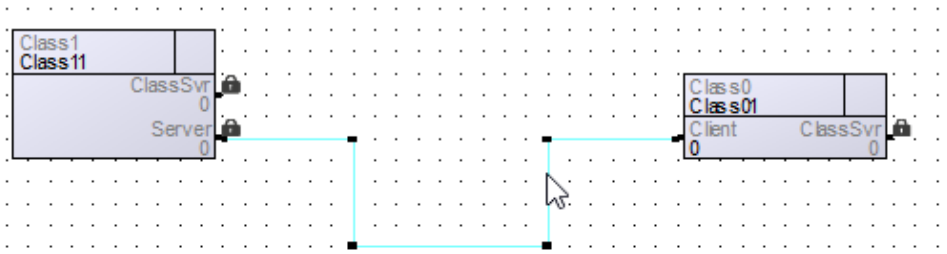
To **add such a point**, you should hold the **Ctrl key** while the connection is selected, then the mouse cursor will change to the sign for "insert a new point":



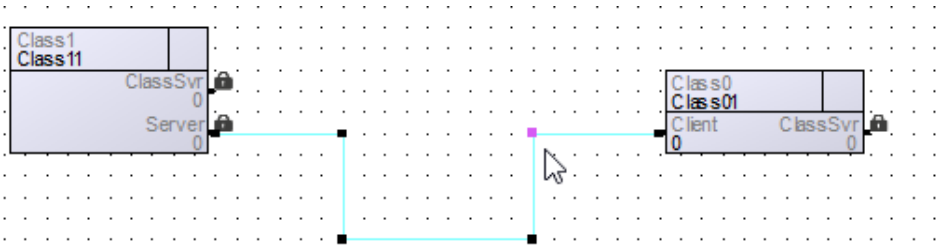
with an additional **left click** on the light blue connecting line a new corner point is **added**. If you **move the mouse over such an inserted corner point while holding the Ctrl key**, the mouse pointer changes to an X and a **left click deletes** this corner point.



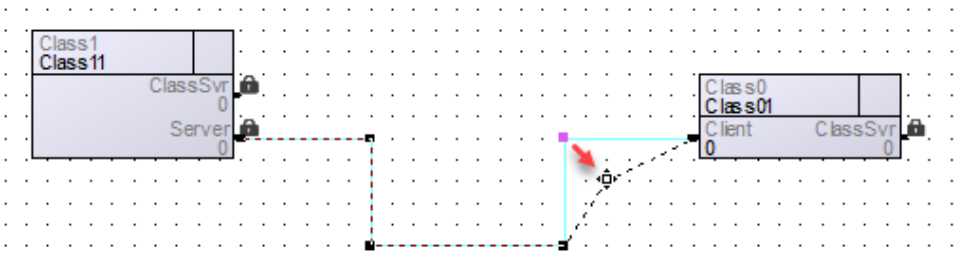
If the mouse is moved over a connecting line with **corner points**, these are displayed as **black squares**.



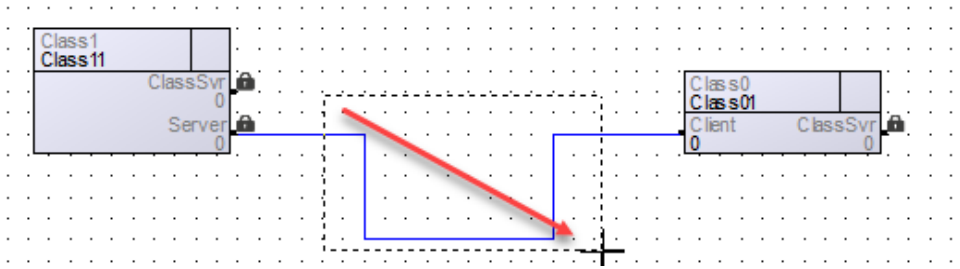
If you **left-click** on such a corner point, it will **be selected** and **colored pink** for identification.



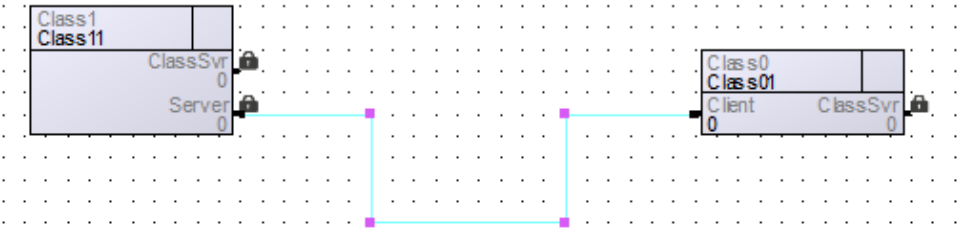
By **clicking and holding the left mouse button**, a corner point can now be **moved** in this way.



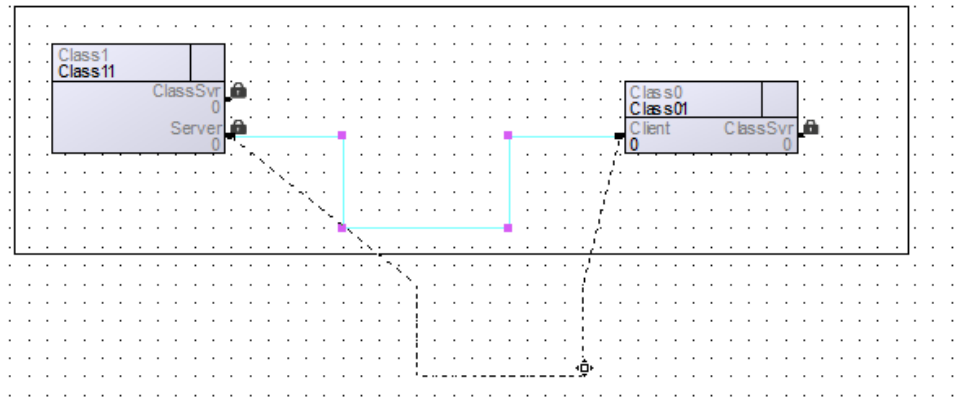
With the **drawing of a frame across several corner points**



these are **all selected**:

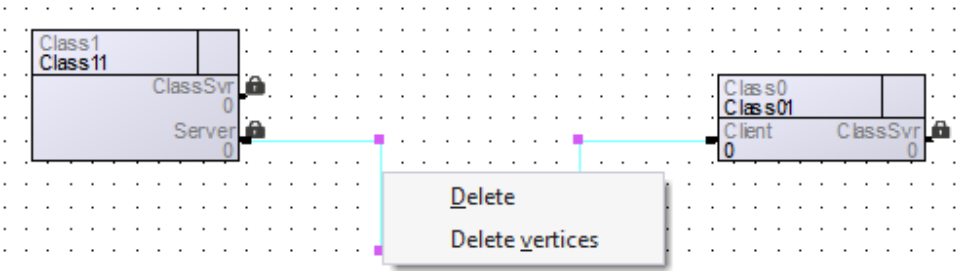


If you click with the left mouse button on one of the selected points or on the connection to which they belong, you can move or **all the selected points at once while holding the left mouse button**.



Likewise, **deleting all selected points** is possible with the **DEL** key.

Right-clicking on a point, or on the connection, opens a **context menu** with the ***Delete*** and ***Delete vertices*** items.

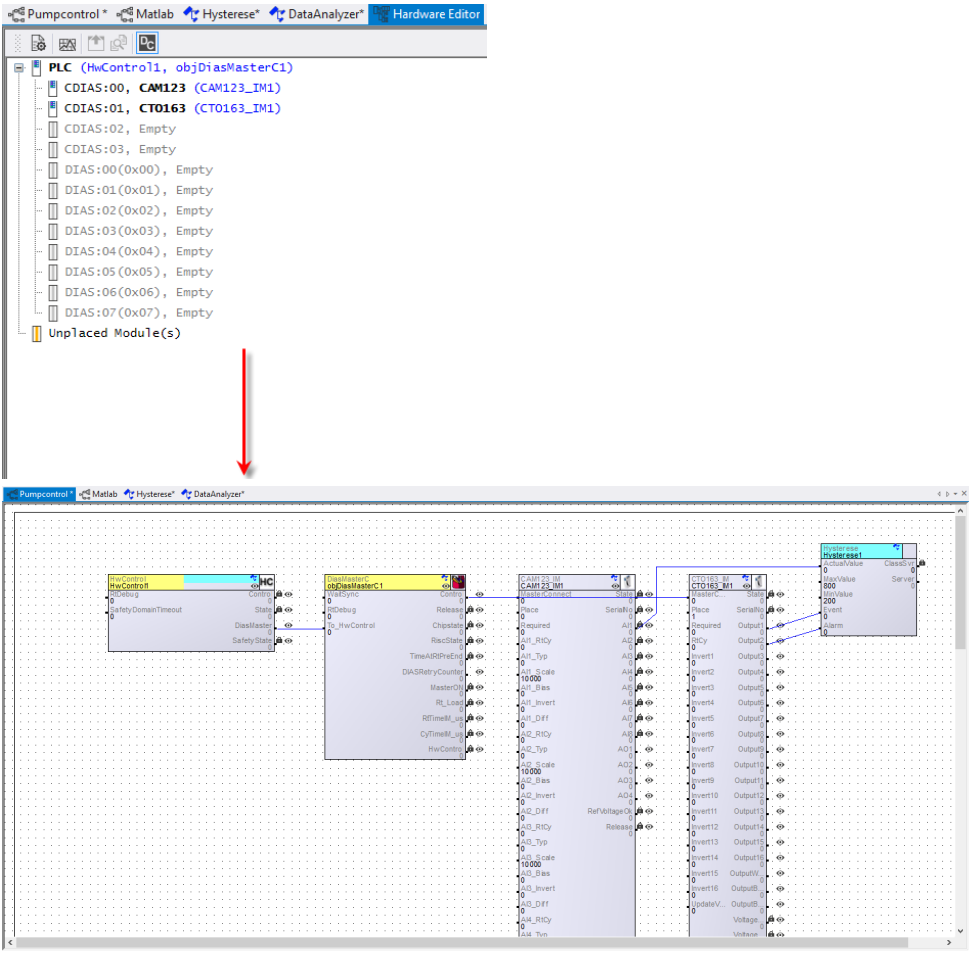


Delete **deletes the selected** point, or the connection, depending on what is clicked. ***Delete vertices*** **deletes all vertices** between the beginning and the end of the connection, making it (again) a single straight line.

2.5.3 Other Displays

2.5.3.1 The Hardware Editor

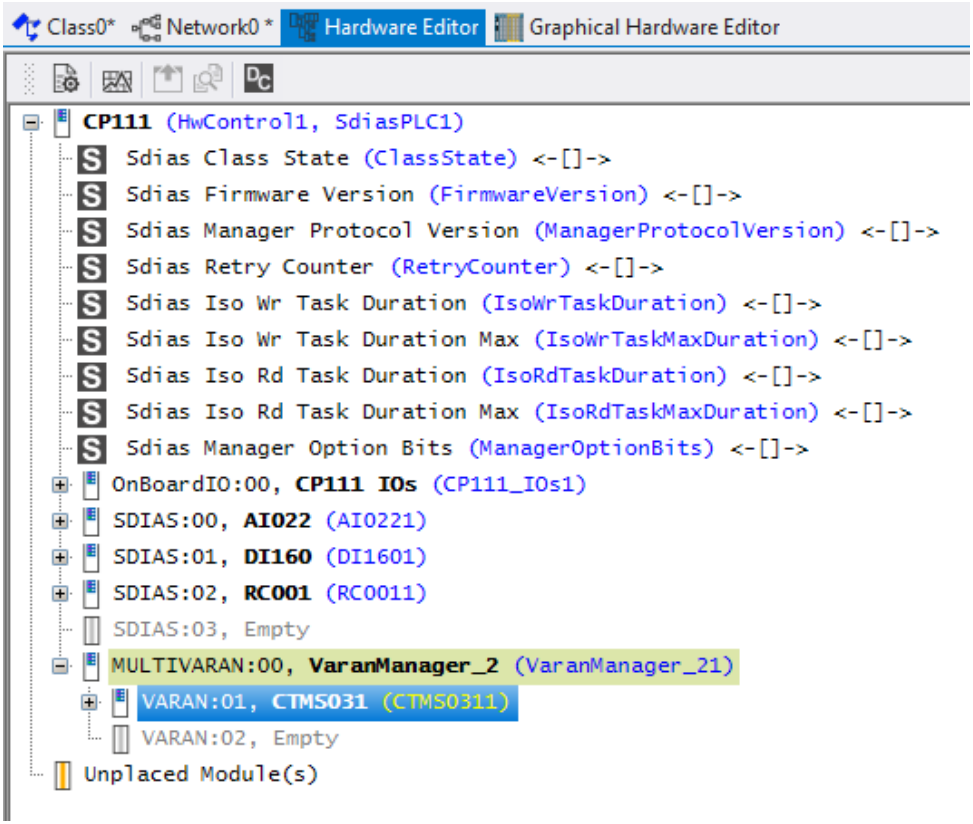
A much faster option for adding SIGMATEK hardware, in comparison with "manual work" in the network view, is entry into the hardware editor. The SIGMATEK classes stored in the library are not only automatically added to the object network – all required connection are also made automatically. The end result can then be observed in the network view and adjusted or expanded if necessary.



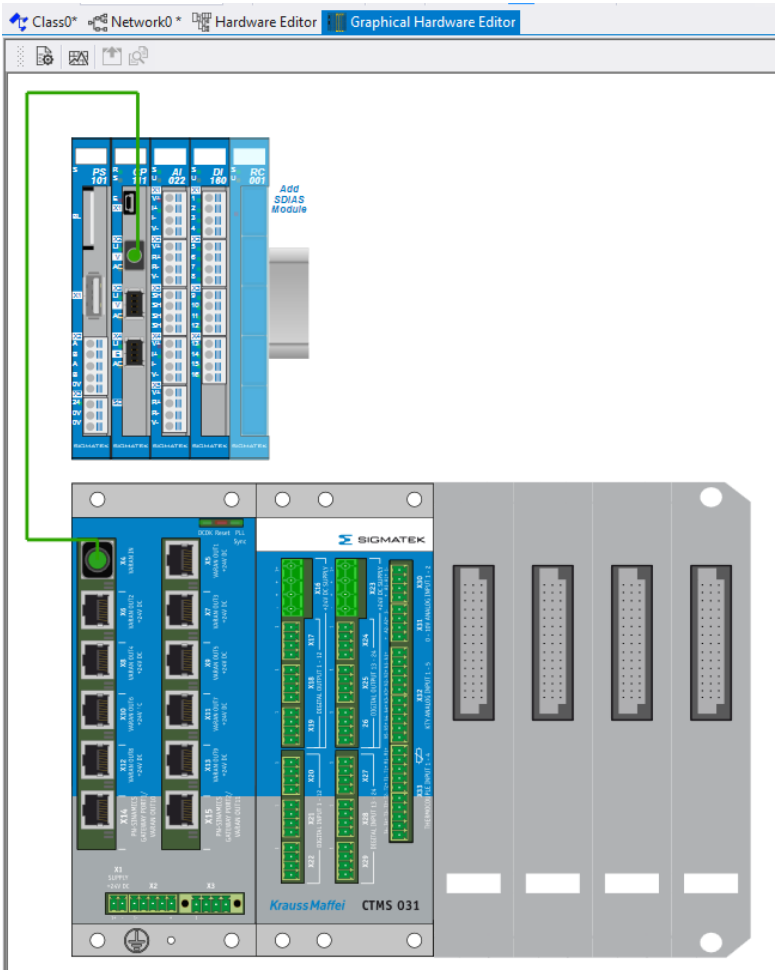
For further information, see the chapter Hardware Editor.

2.5.3.2 The Graphical Hardware Editor

An even more realistic form of presentation is the **Graphical Hardware Editor** (GHE). The hardware configuration is displayed as a tree in the hardware editor,



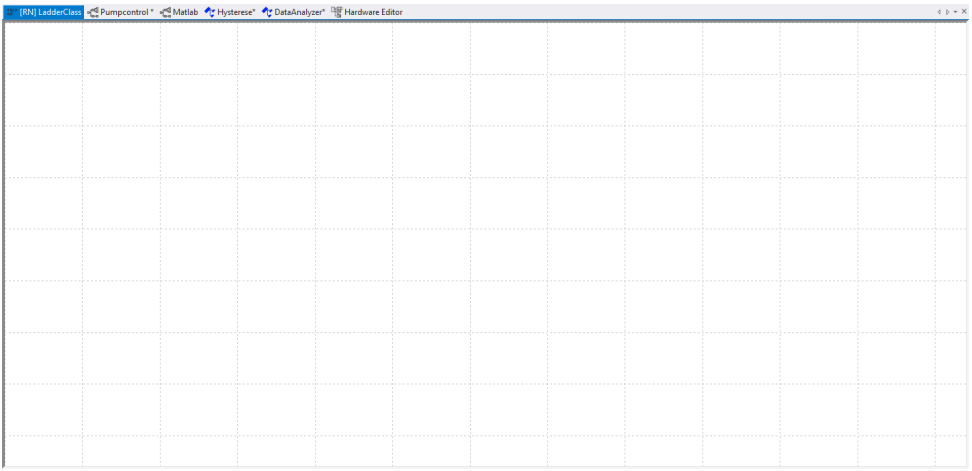
the hardware available in the project is realistically represented in the GHE.



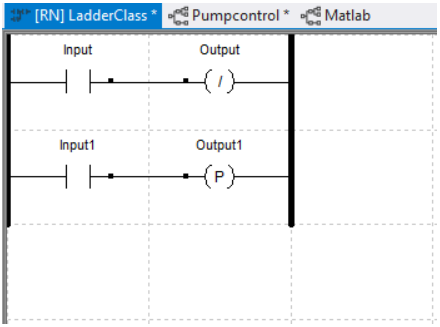
More information about the Graphical Hardware Editor can be found in chapter Graphical Hardware Editor.

2.5.3.3 KOP Class

If a ladder diagram class is added to the project and it is double-clicked in the class tree, an empty ladder diagram is displayed in the workspace.



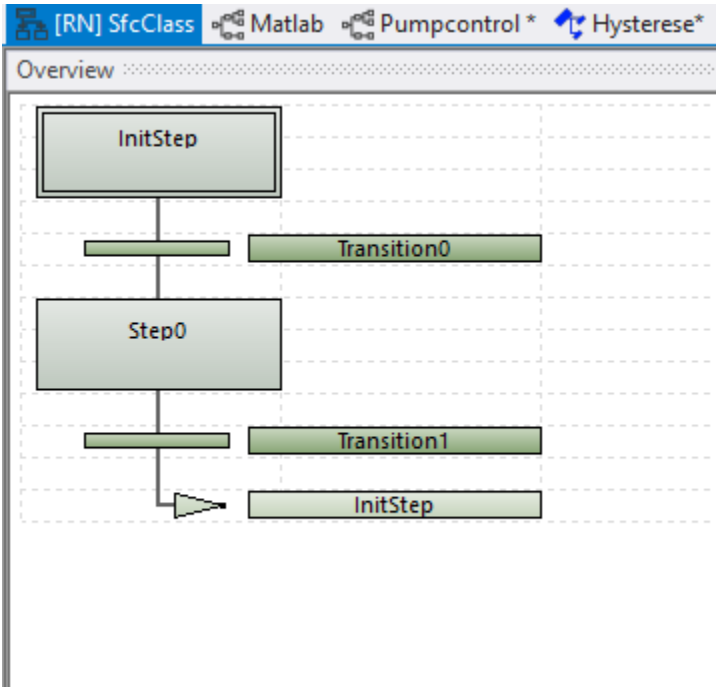
With a right mouse click, the components of the ladder diagram are added and the required connections then made.



For more information, see the chapter Ladder Diagram (LD) in LASAL CLASS 2.

2.5.3.4 SFC Class

If an SFC (Sequential Function Chart) class was added to the project and it is double-clicked, the SFC editor is displayed in the workspace.



For more information see chapter SFC.

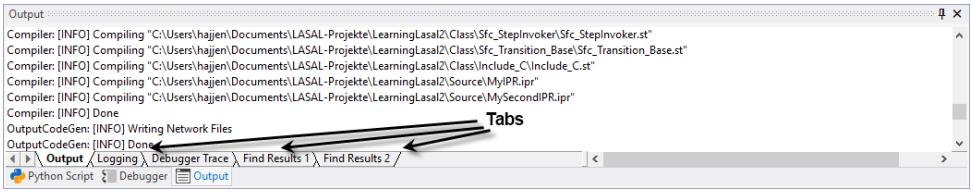
2.6 Information Window

2.6.1 Output Window

The Output window is located at the lower middle border of the LASAL CLASS 2 user interface and contains 5 different output windows. These are opened using tabs located in the lower left corner of the output window.

The Output window contains the following windows:

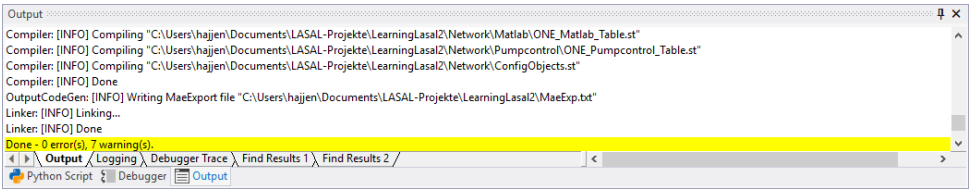
- Output
- Logging
- Debugger Trace
- Find Results 1-2





If the output window is not displayed in the LASAL CLASS 2 user interface, it can be activated by selecting **View, Output pane** from the menu bar.

2.6.1.1 Output

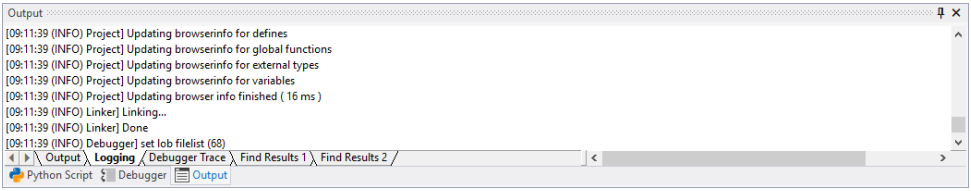


The **Compiler** interprets the high level code to machine language. If an error or warning occurs, they are displayed in the summary at the end of the process.



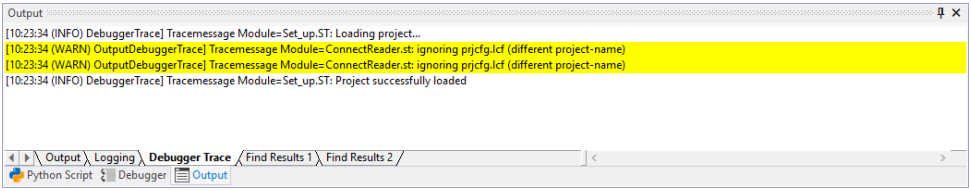
If **errors or warnings** are shown, after selecting the according line you can directly jump **to the description of the according error/warning** in the help with **pressing F1**.

2.6.1.2 Logging



Here, instructions and actions in LASAL CLASS 2 are logged.

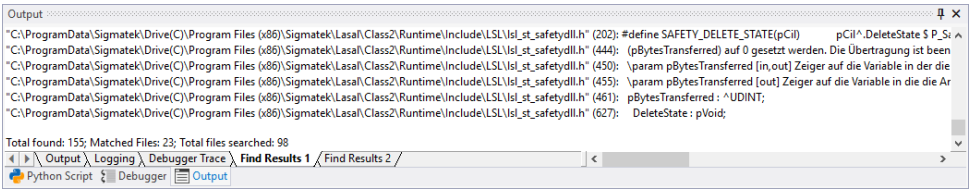
2.6.1.3 Debugger Trace



In this window, **Trace messages** sent by the control are displayed.

Via the context menu (right mouse click in the Trace window) the current Trace messages can be deleted. Through this activity the messages are also deleted on the target system.

2.6.1.4 Find Results 1



In this window, the search results for the **Find in Files...** functions are displayed and can be selected under the menu item **Edit**. The **results of a „Find Reference“ search** are also shown in the **Find Results 2** window.

2.6.1.5 Find Results 2

The result of a **“Find Reference search”** is always output in the **Find Results 2** window. This can also be found in the **Find in Files dialog** or simply enter the sub window in the **Find** dialog as the destination.

2.6.2 Debugger Window

Like the Output window, the debugger window is located at the bottom of the LASAL CLASS 2 user interface and contains 9 different output windows, which are selected using the tabs at the lower left of the debugger window.

The debugger window is comprised of the following output windows:

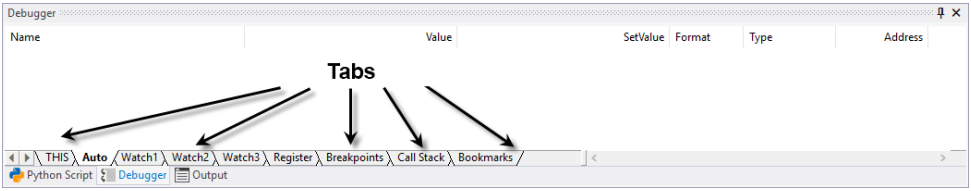
- THIS
- Auto
- Watch 1-3
- Register
- Breakpoints
- Call Stack
- Bookmarks



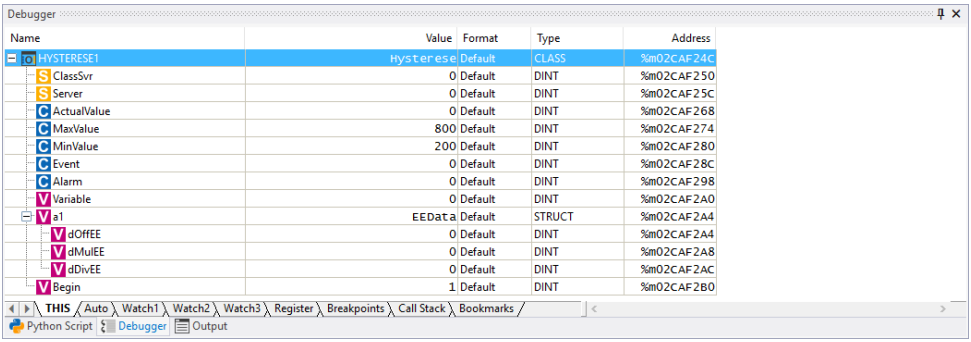
If the debugger window is not displayed in the LASAL CLASS 2 user interface, select **View** and **Debugger pane** from the menu bar.



If an SFC class is debugged, the corresponding elements are displayed in the **THIS** or **Watch Window**. More information can be found in the chapter **THIS and Watch Window in the SFC**.



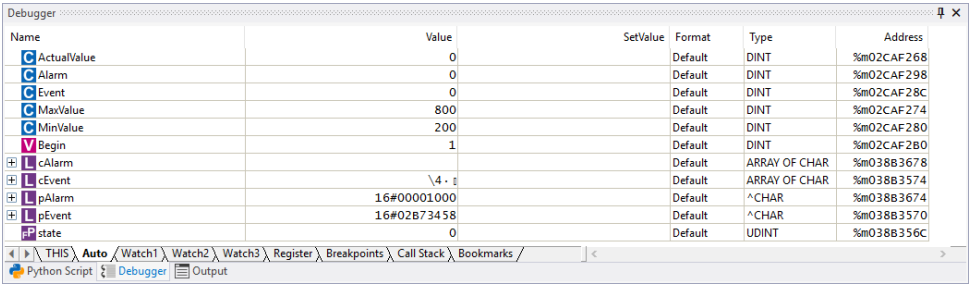
2.6.2.1 THIS



The **THIS window** is used to display all variables, clients and servers of an object. When the name of an object in the object network is clicked, its components are displayed in the window above.

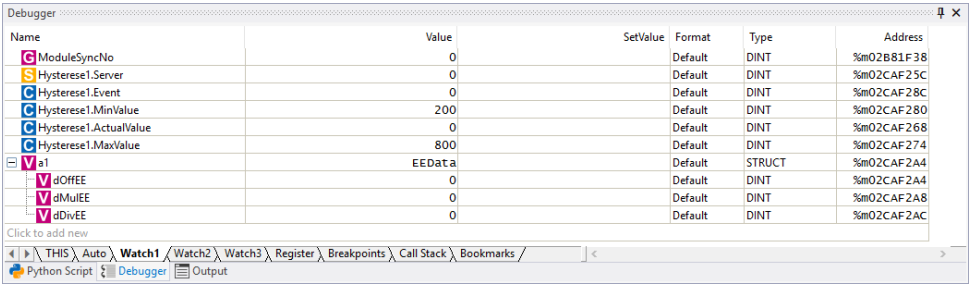
2.6.2.2 Auto

To use the **Auto window**, click on the AutoWatch icon in the toolbar.



The **Auto window** shows the elements that are valid in the function where the cursor is. If the program flow has been interrupted (e.g. because of a breakpoint), also the transfer parameters and the local variables are shown.

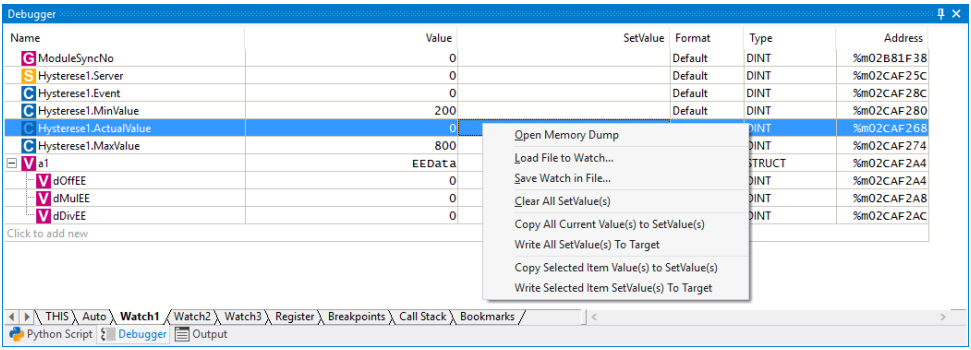
2.6.2.3 Watch 1-3



To observe several values from different objects, select a variable and insert it into one **Watch window** with the Drag & Drop method. To display the value, the variable name can be also be entered by selecting **Click to add new**. The **Watch windows** can only be used when debugging.

2.6.2.31 The SetValue column and its context menu

The Watch windows contain the **SetValue** column. This is used to set the variable in the PLC rather than in the PC. The entered values do not change content of the **Value** column. Any value can be entered into the **SetValue** column. If an online connection now exists and when for instance after a multi-selection this column is clicked on the right in the Watch window, the following context menu is opened:



With **Save Watch in File...** the complete **Watch list** can be **saved** as *.lwl (Lasal Watchlist Files) for further use.

Load File to Watch... loads such a Watch list.

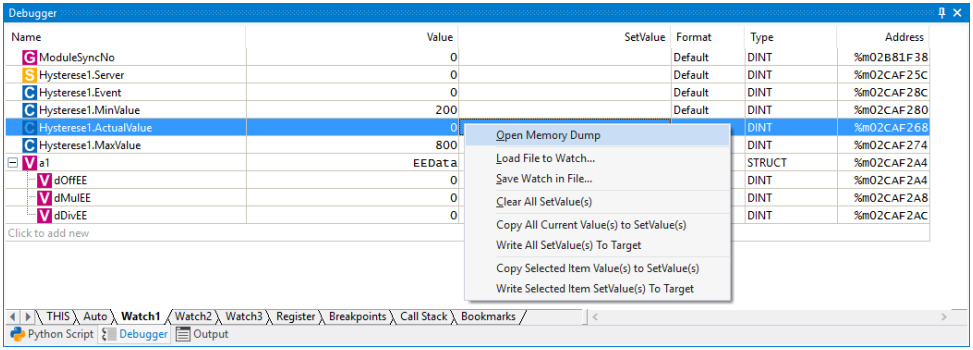
With **Clear All SetValue(s**), the entire column is cleared.

Copy all current Value(s) to SetValue(s) copies the actual values to the **SetValue** column (and **not** in the PLC!).

Write All Set Value(s) to Target writes all values available in the column to the PLC.

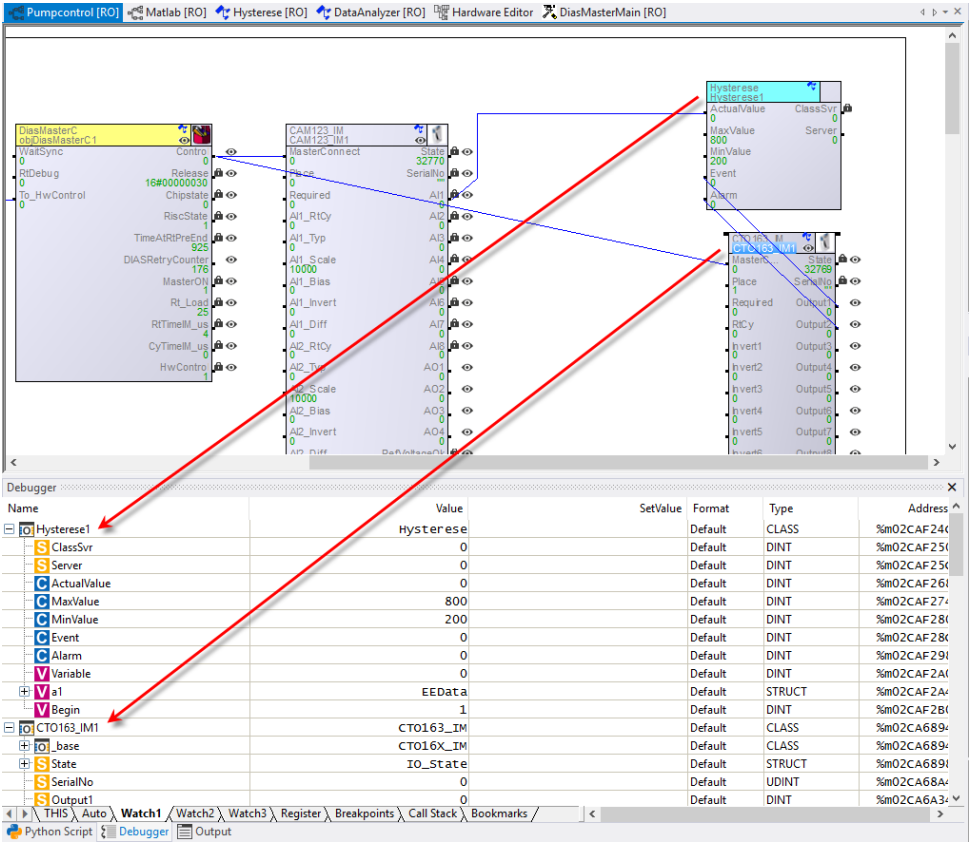
The last two menu items - **Copy all current Value(s) to SetValue(s)** and **Write All Set Value(s) to Target** - are the same as the two previous ones with the exception that here, only selected values are copied to the **SetValue** column or written to the **PLC**. Since multiple selection is possible in this window, these are diverse functions.

If only a single variable has been selected, the context menu gets an additional entry at the top: **Open Memory Dump**. If it is selected, Advanced **Debug Tool / Memory Dump** automatically **starts with** the (changeable) **start address of the selected variable**.

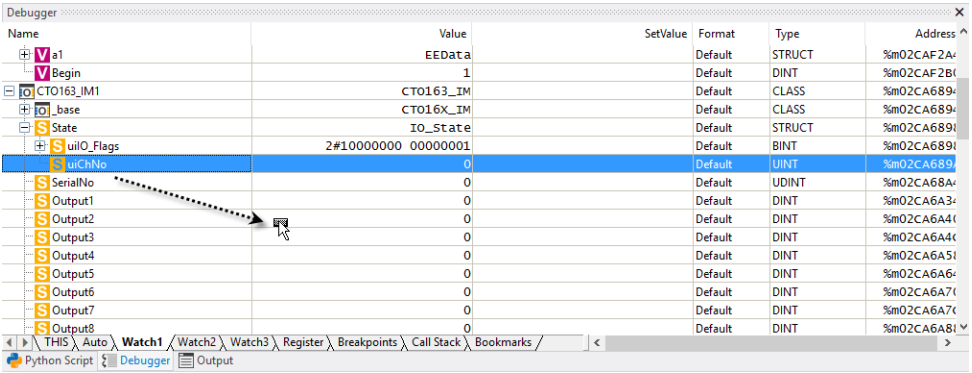


2.6.2.32 Drag & Drop in the Watch Window

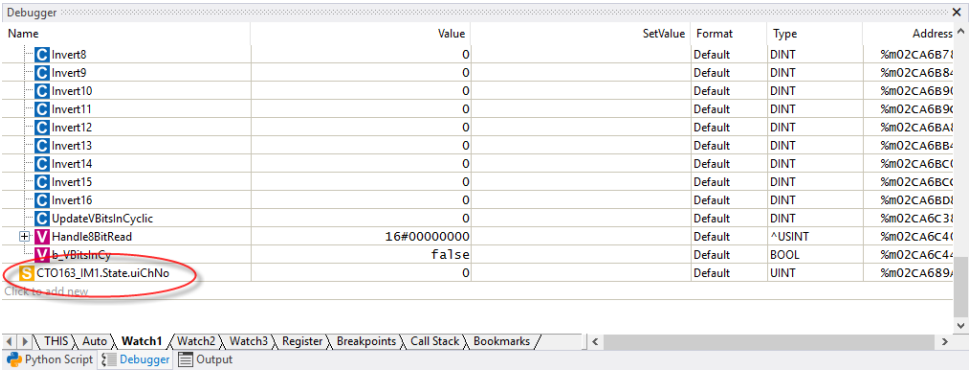
During **online operation** in LASAL CLASS 2, all objects placed in a network can be inserted in a Watch window by Drag & Drop. In this window, all components of an object such as clients, servers and variables are displayed.



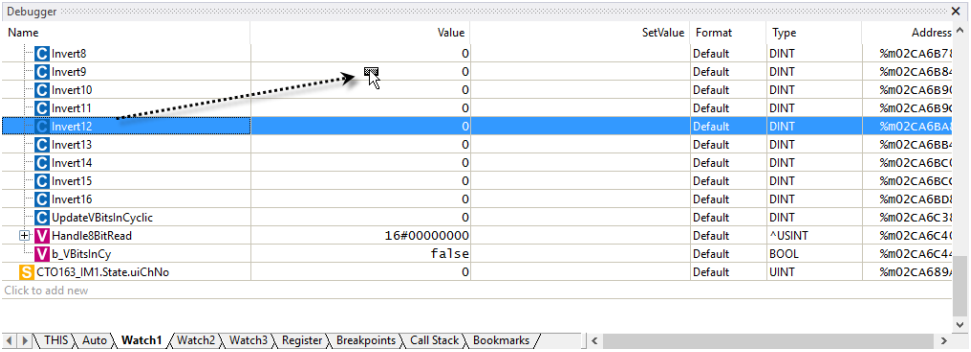
The option is available to simply **Drag & Drop** a desired (sub) element from the Watch window anywhere in the Watch window. Here, the element with the object name is place at the end of the table. This makes sense, especially for large objects, when only specific elements should be monitored simultaneously that are located far apart from one another in the object.



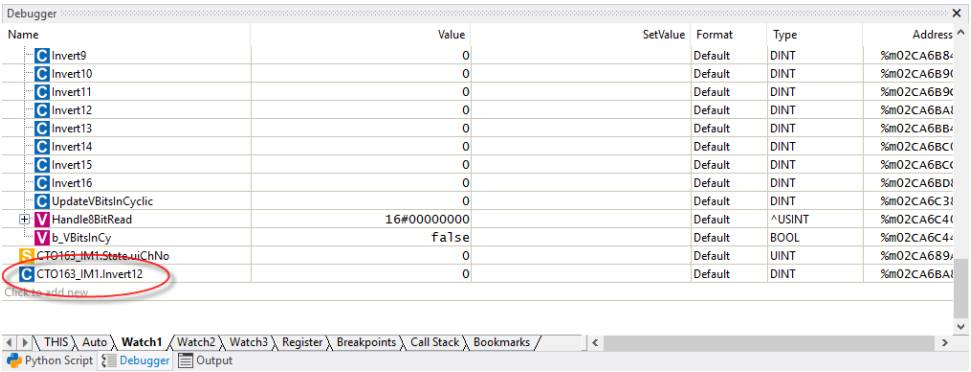
The server **uiChNo** is thereby added below.



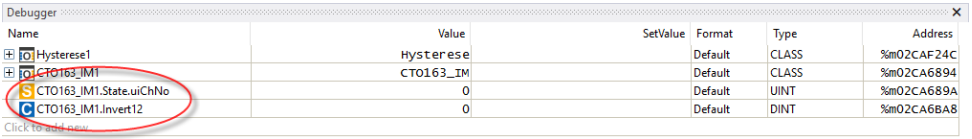
Now the client **Invert12** is dropped into the Watch window with Drag & Drop.



... and now is also added below.



If all sections are closed now, you have a good possibility to clearly view all selected elements at the same time.



The option to also move a server or object from the HW editor into the Watch window is available.

2.6.2.33 Casting in the Watch window

It is possible to **cast** the displayed variables **directly in the Watch window**. To do this, **double-click in the first column** (where the variable name is) and enter a **$ character and the data type** directly behind the variable name. For example, if the variable uiInput is of the data type UINT, it is - if possible - casted to the data types UDINT using the command

```
uiInput$UDINT
```

If it is a pointer, the syntax is as follows:

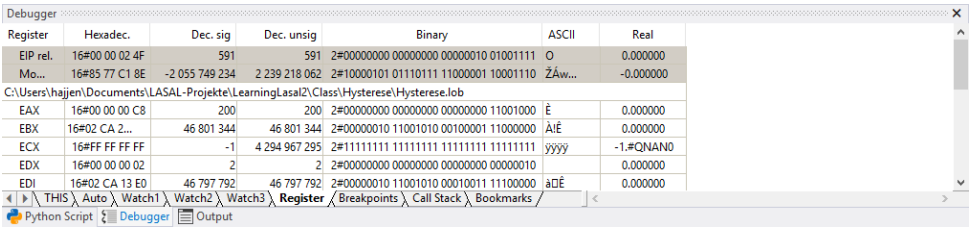
```
pTest^$UDINT
```

This command returns a UDINT value from the address the pointer pTest points to. You can also specify here how many additional bytes are to be read:

```
pTest^$UDINT, 20
```

returns the value of the pointer as UDINT and another 20 bytes.

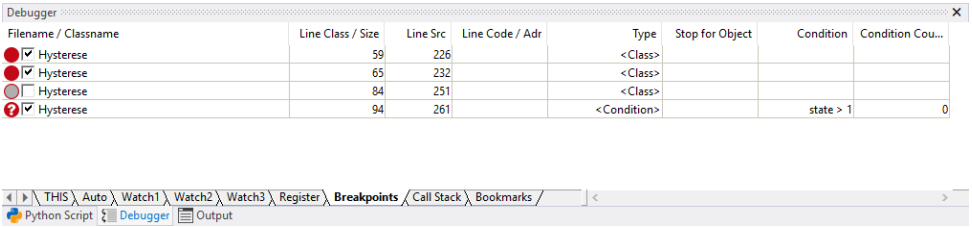
2.6.2.4 Register



This output window is mainly use to debug AWL programs. The **Register window** allows for observing the process registers.

To select additional output windows, select Debugger window.

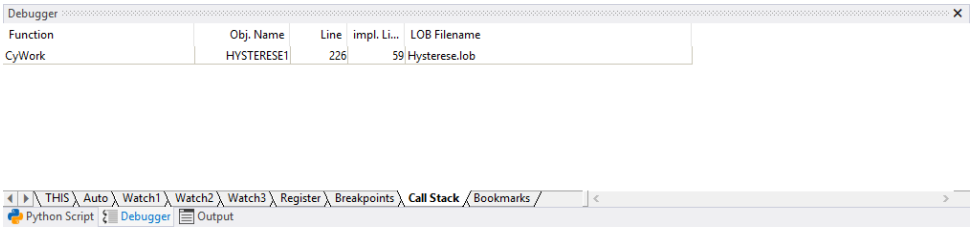
2.6.2.5 Breakpoints



In this window, all active and inactive breakpoints are shown. All breakpoints in the program code as well as the source code are shown here.

To select additional output windows, select Debugger window.

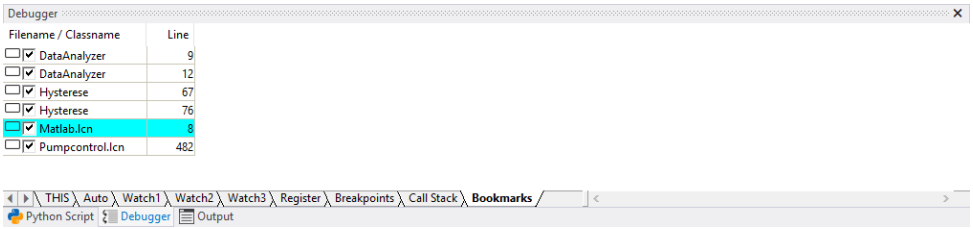
2.6.2.6 Call Stack



Here, the actual program line being processes is shown. If the program were to jump to a method, the address is displayed at which the program continues after ending the method.

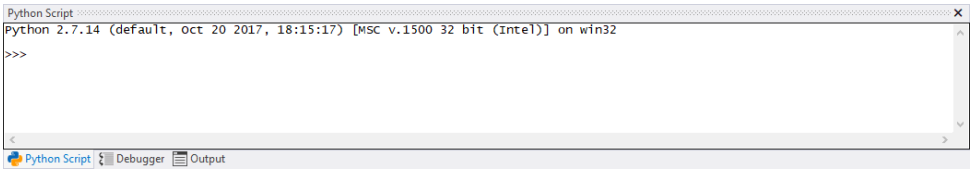
To select additional output windows, select Debugger window.

2.6.2.7 Bookmarks



Here - similar to the breakpoints - the current bookmarks are managed. The file or class name as well as the line number are displayed. The last bookmark you jumped to is indicated in cyan. Similar to the breakpoints, it is possible to enable / disable bookmarks with a checkbox.

2.6.3 Python Script window



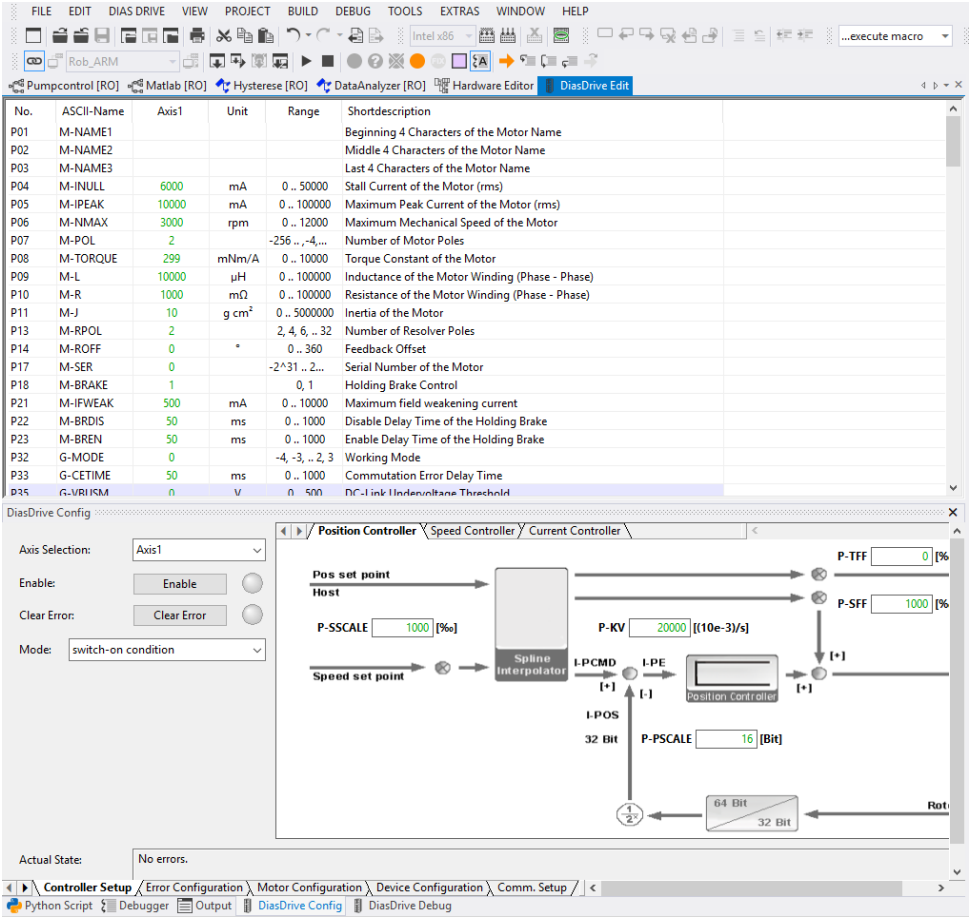
This window shows the in and output windows of the Python interpreter. Here, Python instructions are entered and executed.

For more information on Python, see chapter Scripting.

2.6.4 Tool-dependent temporary window

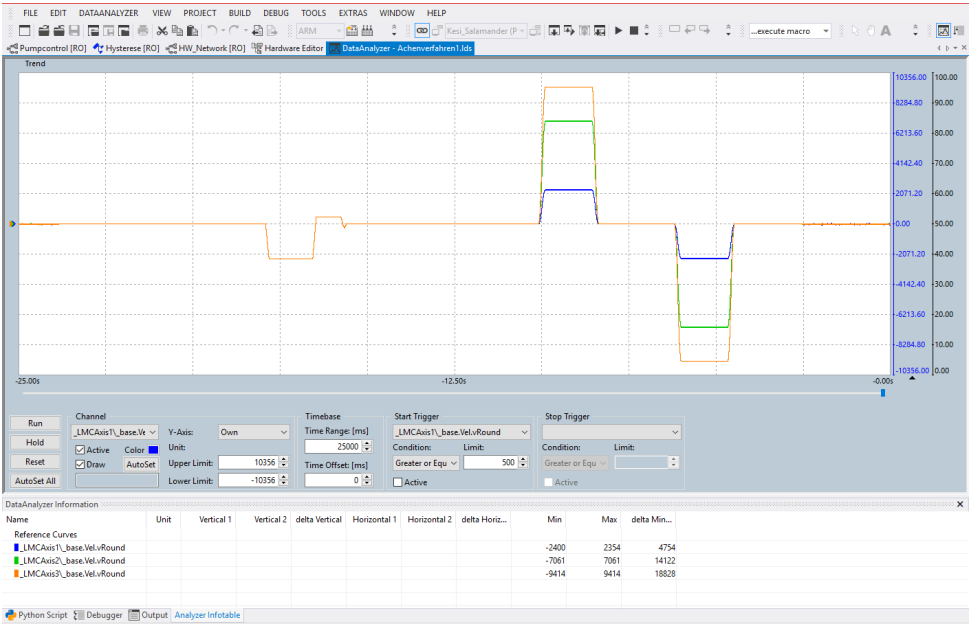
If special tools or views in use, additional information windows can be displayed.

With the Dias Drive for example:



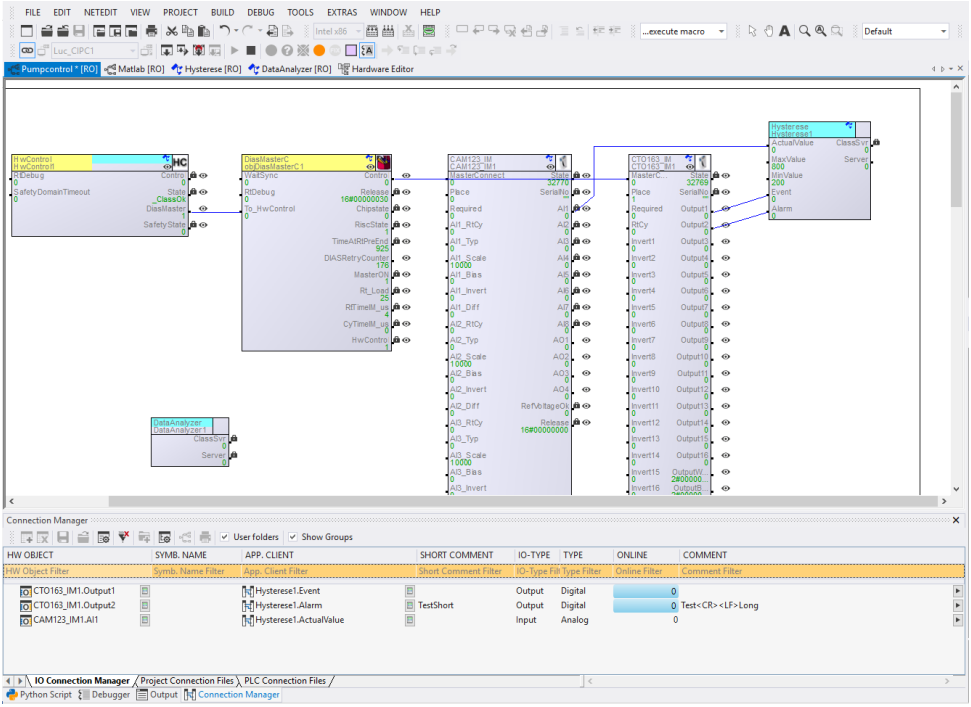
For further information, see the chapter DiasDrive.

Displays are also available in the information window area of the Data Analyzer:



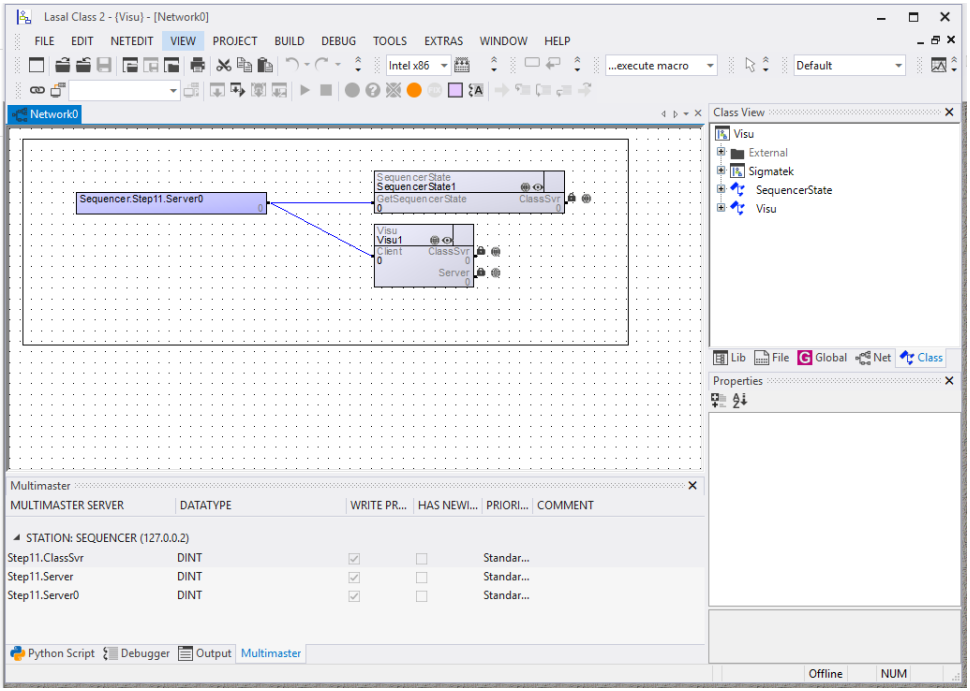
For further information, see the chapter Data Analyzer.

The IO Connection Manager is shown in the information window as well:



For further information, see the chapter IO Connection Manager.

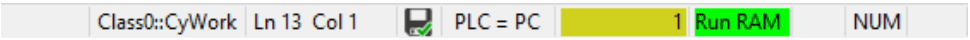
If LASAL has been started from the Machine Manager, so possible stations and servers are shown in the Multimaster server pane of the information window.



It is also possible to display the Multimaster Pane directly from the LASAL. In the menu select View / Multimaster Pane. For further information see the help documentation of the **Machine Manager**,

2.7 Status bar

The status bar is located on the lower margin of the LASAL CLASS 2 user interface provides information about the current process status (here: Run RAM) and shows keyboard commands (here e.g. NUM) as well as the value of the IL trigger in debug mode (here 1) and in step mode the name and class of the current function at the position of the command pointer (here Class0::CyWork). In online mode, there is also an icon that shows whether the project - as shown here in the picture - is already saved () on the control, or not () .





Attention

This only **works reliably** if the **loader version** on the PLC **is identical** to the current LASAL version! Background: When saving a project, the loader CRC is added to the project CRC and thus the project is not recognized as the same, even if only the loader is different.

3 PROGRAMMING

3.1 Classes and Objects in LASAL CLASS 2

3.1.1 Introduction to Object Orientation

3.1.1.1 Object Oriented Programming (OOP) Basics

To begin with OOP, it's best to consider the read world. The real world consists of objects. For example, houses, machines, people, plants and animals, etc.

Each of these objects has certain characteristics. They have a certain condition and serve a specific task. These objects have internal characteristics that user does not have to fully understand in order to use them.t is enough, at first, to be familiar only with the outer components of an object and to know what task it performs. To understand the world externally, one must be familiar with the functions of various objects and know how they communicate. It would seem logical then, to apply the object oriented concept to programs.

A program is comprised of objects that are independent parts responsible for specific tasks.

Each object contains specific information, which represents its current condition. This data is represented using variables. The tasks and actions of an object are represented by functions. These are described in OOP as methods.

One distinguishes:

- The internal construction of an object

- The externally visible properties (connections) of an object as well as the external function

It is only necessary to understand the total function of an object last. How an object performs its tasks is an internal issue (Implementation, nesting).

3.1.1.2 Advantages of OOP

The purpose of OOP is to make it easier for the programmer. An object with all its prosperities and functions is easier to imagine and to work with than a long string of instructions. The program is also clearer.

The methods and properties of an object in conjunction with variables and functions are much easier to understand.

The possibility for structuring in manageable and easy to maintain programs is a significant advantage of OOP.

A class can be tested as a single, independent unit. The multiple testing in several user programs is not required, thus increasing the efficiency of the programming.

The use of classes increases the reusability of the software.

3.1.1.3 Important Concepts in OOP

3.1.1.31 Classes

A Class represents a user defined data type and is the most significant language element of object-oriented programming. A class is the description of an object.

When developing classes, the following questions are raised: what is needed to describe an object exactly and which properties must the object exhibit? What does it do? To meet these requirements, classes with status describing properties (attributes) and methods are provided.

3.1.1.32 Objects

An object in the world of object-oriented concepts is nothing more than a variable of the type from a specific class. It is practically a realization of a class description, which possesses attributes and methods. In order to work with an object in program, it must be instantiated.

3.1.1.33 Instantiation

A specific object is generated with the instantiation of a class. That means, for a class to be usable, an object of this class must be created. Each object is an instance of a specific class.

3.1.1.34 Attributes

Also called properties, attributes are basically nothing more than variables than contain a value and describe the status of the object.

3.1.1.35 Methods

Methods execute actions and status changes of objects. Methods specify the "behavior" of a class object.

3.1.1.36 Data Encapsulation

Objects cannot read or change the internal status of other objects. Only the method of an object should be able to access its own internal status. The access of an object externally is only possible using public methods (global methods). A class therefore, insulates data from direct access.

3.1.1.37 Inheritance

Objects, which are by nature from different sorts, can have common features. Imagine, for example, that you must implement three classes (students, workers, pensioners) in a project.

All classes, among other things, have the following properties in common:

- Gender
- Address
- Nationality
- Place of birth

It seems logical to define a single class, People, which contains only the common characteristics. A student would be a special type from the class "people" as well as workers and pensioners.

Classes can also be defined based on classes already available. A class, whose properties are available to other classes, is identified as a **Base Class**. A class the takes on the properties of a base class is identified as a Derived Classes. **Derived classes** take on the properties and methods of the base class but expand the inherited characteristics with type specific properties.

The advantages of inheritance are:

The reduction of program code as the common properties and methods must be implemented once in the base class.

Common properties of classes only exist in one place in the program and must only be tested or changed once if required.

3.1.1.38 Polymorphism

The ability of program elements to reference objects of various classes during runtime is called polymorphism. In other words: during the runtime of a program, an operation is executed in the object with the corresponding instructions. The means the same method can be used in different objects.

3.1.2 Class Definition

3.1.2.1 Basic Information on Methods (Privat, Global, Virtual Global)

To strengthen the modularity of a class, methods can be defined for an explicit area of validity. A method can be defined as **global** or **virtual global** as well as **private**.

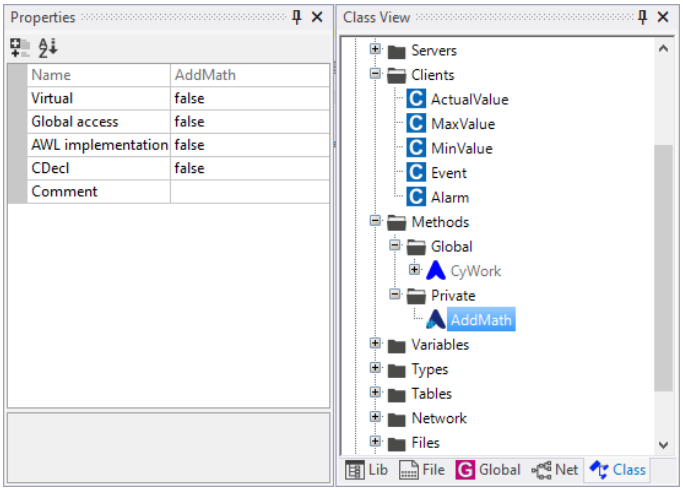
3.1.2.11 Private Methods

The methods defined as private can only be accessed within the class in which it's defined. That means, private methods are valid only with the class and cannot be called by another class.

In the Properties window, the setting for Global access was not made.

```
FUNCTION Hysterese::AddMath

END_FUNCTION
```





The  symbol means that this method is **private**. For a summary of the possible method and server symbols see the chapter "Method and server symbols"

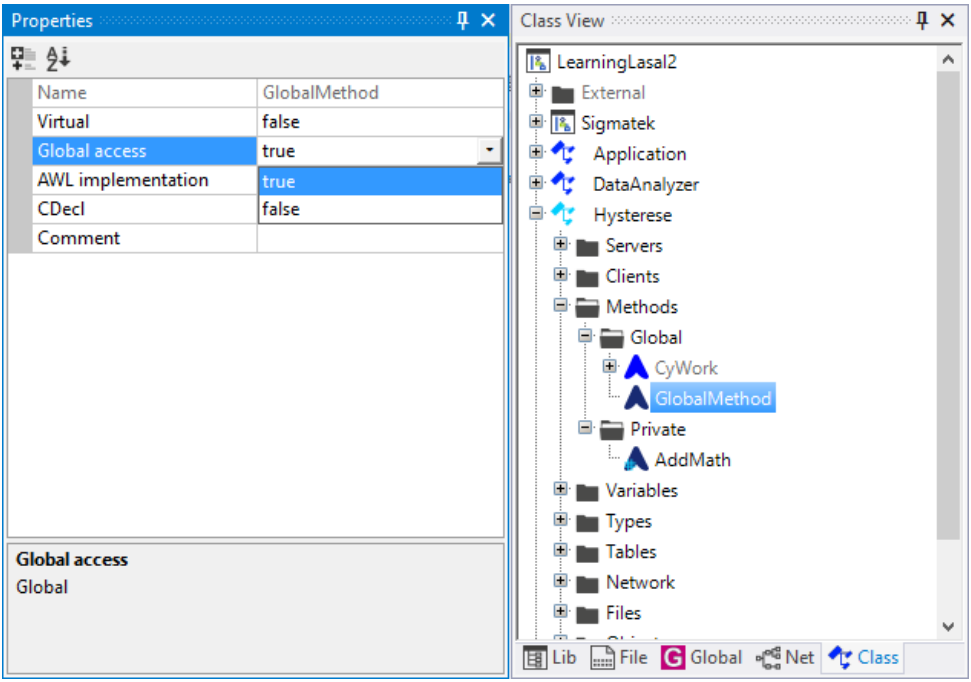
3.1.2.12 Global Methods

Global methods of a class are valid with in the entire class and can also be called by other classes.

Global methods are set by selecting the methods name and in the properties window, setting **Global Access** to **True**.

```
FUNCTION GLOBAL Class0::GlobalMethod

END_FUNCTION
```



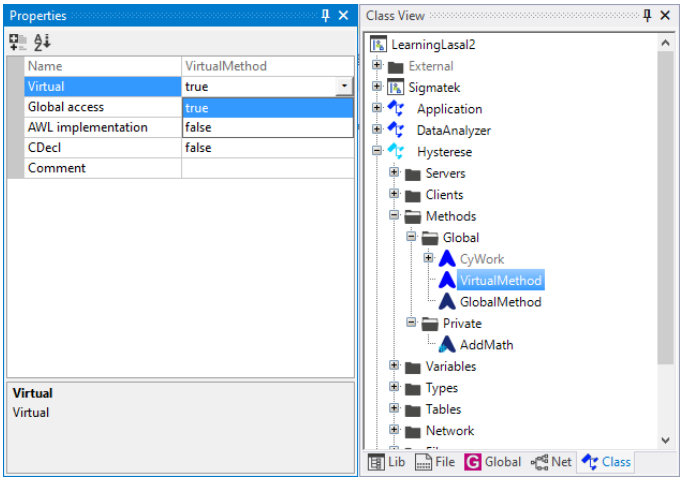
The symbol means the method is **global**. For a summary of the possible method and server symbols see the chapter "Method and server symbols"

3.1.2.13 Virtual global Methods

Virtual global methods of a class are valid with in the entire class and can also be called by other classes. To assign virtual methods, both **Virtual** and **Global access** must be set to **True** in the **Properties window**.

```
FUNCTION VIRTUAL GLOBAL Class0::VirtualMethod

END_FUNCTION
```



Virtual global methods can be overwritten in derived class. With virtual methods, the methods of the base or derived class are called using a methods table, but only if they were overwritten.

If the global method is set to **Virtual**, the color of the symbol is lighter. For a summary of the possible method and server symbols see the chapter "Method and server symbols"

If the declaration of a virtual method is changed (new input variable or the like), then the overwritten methods in possibly existent derivations are changed automatically. However, in some cases this is not possible, because, for example, the base class is linked. If so with a right click on an overwritten method the **Assimilate Overwritten Method** command can be selected from the context menu in order to manually adapt the method.

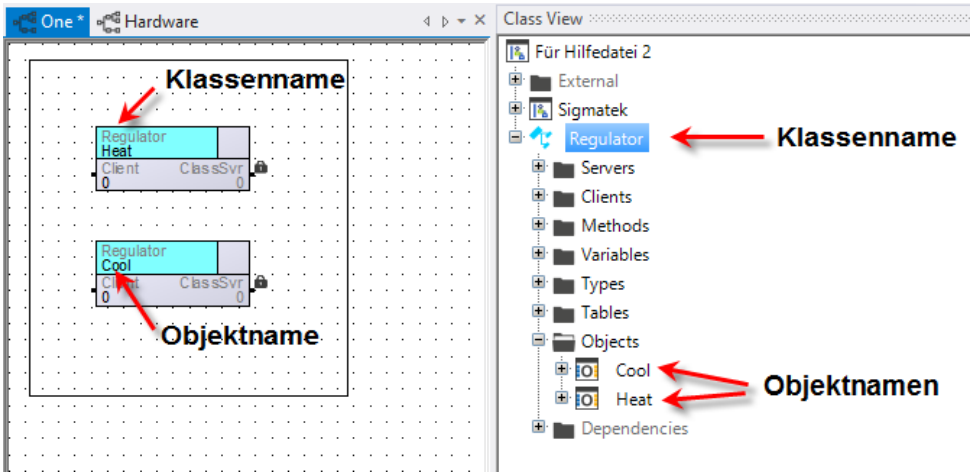
3.1.2.14 Virtual private Methods

Virtual private methods can be written in a derivation; an external call through an object channel is not possible.

For a summary of the possible method and server symbols see the chapter "Method and server symbols"

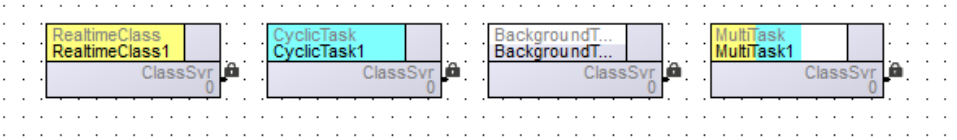
3.1.2.2 Classes and Objects in LASAL Class 2

A class created in LASAL CLASS 2 (e.g.: class name **Regulator**) can only be used once with the class name. However, several objects of a class can be placed in an object network. Each object must have its own name (e.g.: Object name "Heat" or "Cool").

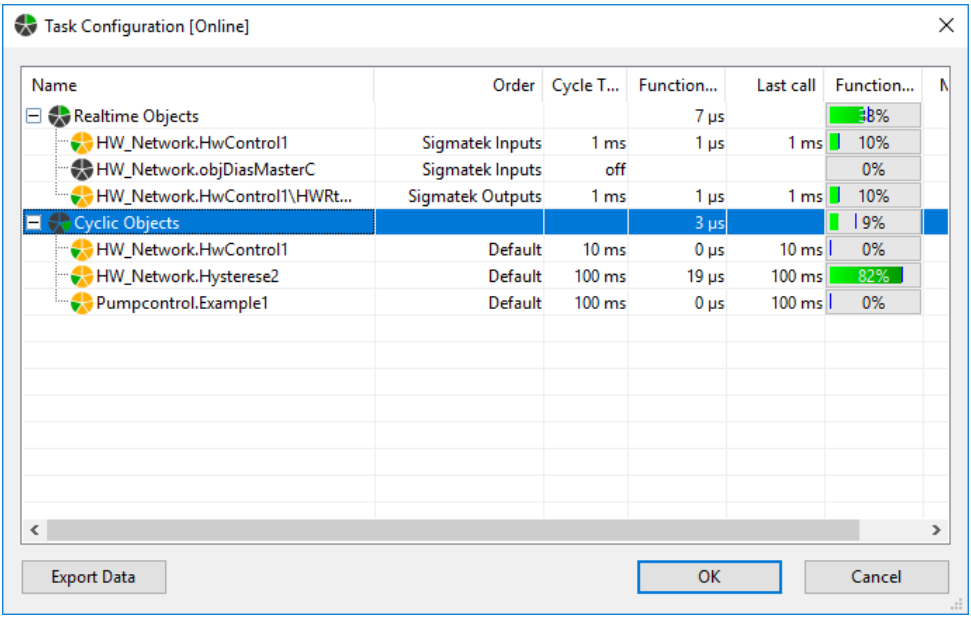


An additional distinction of classes and objects is the **Task configuration**. The possibility exists to let objects run in various tasks. Here, Realtime work, Cyclic work and Background are available.

In the following screen, the class headings can be seen in varying colors. Each color indicates a different type of task (Realtime, Cyclictime and Background). If more than one task is activated in a class, the area containing the class name will have a multicolored background (up to 3 colors). The background of the object name displays a color only when the time is set for the selected task. A gray color means that the respective task is deactivated.



The **Task Configuration window** shows in which category the individual objects run as well as the cycle time set for each object. The properties folder gives the order of the tasks according to their priority.



For task times setting also constants can be used.

3.1.2.3 The Client and Server Model

An essential characteristic of LASAL CLASS 2 is the object oriented programming with graphic representation, project construction and debugging.

Objects connected in an object network can exchange and process data with one another. An overview of the project data flow and interfaces is available at a glance.

As in a PC network, the client (**Client**) requires services, which are provided by a corresponding server (**ClassSvr**).

Communication between Server and Client:

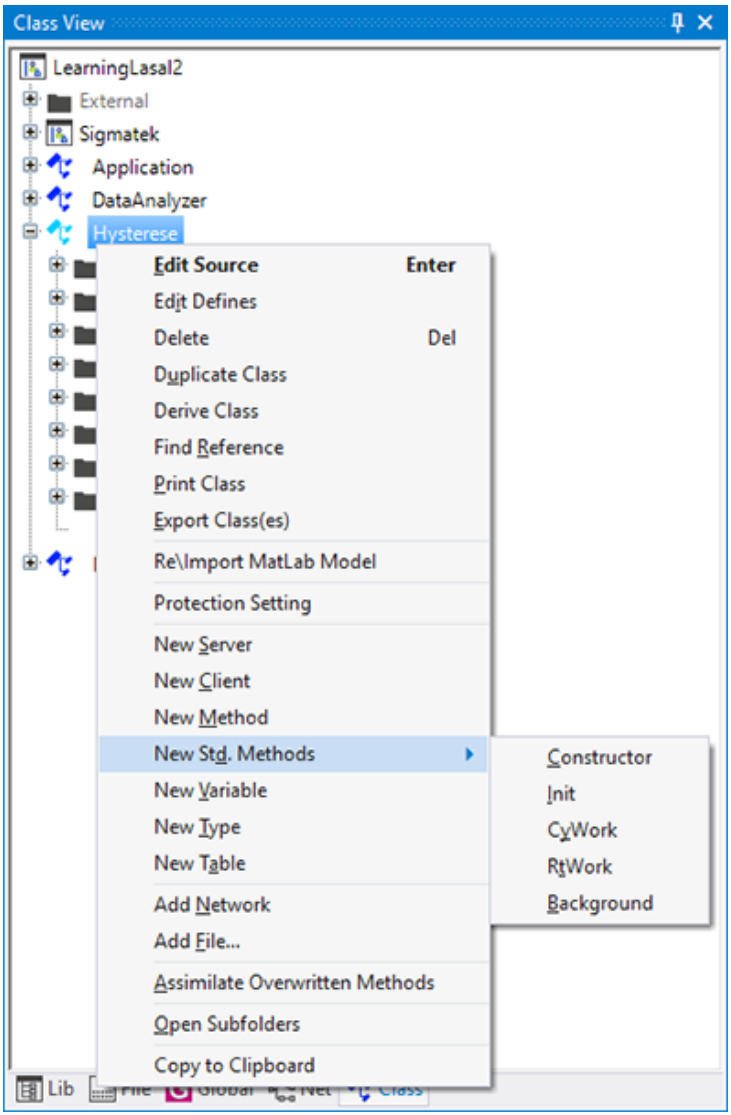


3.1.3 Methods of a Class

A method is another description for an operation, which can be executed based on the data of an object. A method is an element of a class. The description, method, is an object-oriented name for a function.

3.1.3.1 The Standard methods of a class

The following standard methods of a class are available:



3.1.3.11 Constructor

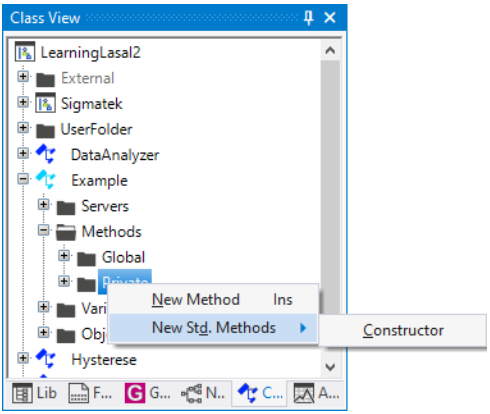
The constructor has the same name as its class and is a special method of a class used to initialize the properties of an object.

The constructor is called by the operating system once directly after the generation of memory for the data. This method serves to initialize the private variables of a class.

With derived classes there is a rule for the call sequence of the constructors: they are processed from the inner most base class to the outside.

Base();

Derivation();



The standard constructor for a class is always called once. If a constructor is assigned for possible initialization, it is called once additionally.

The init method is the called 12 times.

In the startup phase, all constructors are called first. Once this process is completed, all Init methods are called 12 times in sequence.

```
FUNCTION Application::Application
  VAR_OUTPUT
    ret_code  : ConfStates;
  END_VAR

      //  Insert your Code
      //  for Initialization

  ret_code := C_OK;

END_FUNCTION
```



Up to this point, no connections exist to other objects. Therefore access to another object in the constructor is not possible (e.g. calling the read method of another object).

3.1.3.12 Init

The Init method can also be used to initialize private variables in a class. At the time the Init method is called by the operating system, connections to other objects already exist. This means they can be accessed using the **Read()** and **Write()** methods.

This method is called by the operating system 12 times. If several objects in a project are called by the operating system, the Init method of the objects is called sequentially starting with the first. Calling this method **12 times** is necessary so that the Init method in several objects can be accessed simultaneously.

```
FUNCTION VIRTUAL GLOBAL Hysterese::Init
END_FUNCTION
```



The operating system variable _**firstscan** is use to run the Init method in the program code once.

```
FUNCTION VIRTUAL GLOBAL Hysterese::Init
  if _FirstScan = TRUE then
    // Program code
  end_if;
END_FUNCTION
```

After the 11th run of the Init method, the operating system variable _**firstscan** is set to **True** (=1). The methods RtWork, CyWork or Background are called cyclically.

3.1.3.13 CyWork (Cyclic Work)

Each CyWork method of an object is called by the operating system when the cyclic time is set to a value greater the null. The priority of the CyWork method lies between the RtWork and Background methods.

```
FUNCTION VIRTUAL GLOBAL DataAnalyzer::CyWork
  VAR_INPUT
    EAX    : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)    : UDINT;
  END_VAR

  state := READY;

END_FUNCTION
```



Used, for example, for flow control in various tasks.

3.1.3.14 RtWork (Realtime Work)

Each RtWork method of an object is called by the operating system when the Realtime of the object is set to a value greater than null. This method has the highest priority of all tasks.

```
FUNCTION VIRTUAL GLOBAL DataAnalyzer::RtWork
  VAR_INPUT
    EAX    : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)    : UDINT;
  END_VAR

  state := READY;

END_FUNCTION
```



Used, for example, for axle control.

3.1.3.15 Background

Each Background method of an object is called by the operating system when the background time on an object is set to a value greater than null. The Background method has the lowest priority of all tasks.

```
FUNCTION VIRTUAL GLOBAL DataAnalyzer::Background
  VAR_INPUT
    EAX    : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)    : UDINT;
  END_VAR

  state := READY;

END_FUNCTION
```



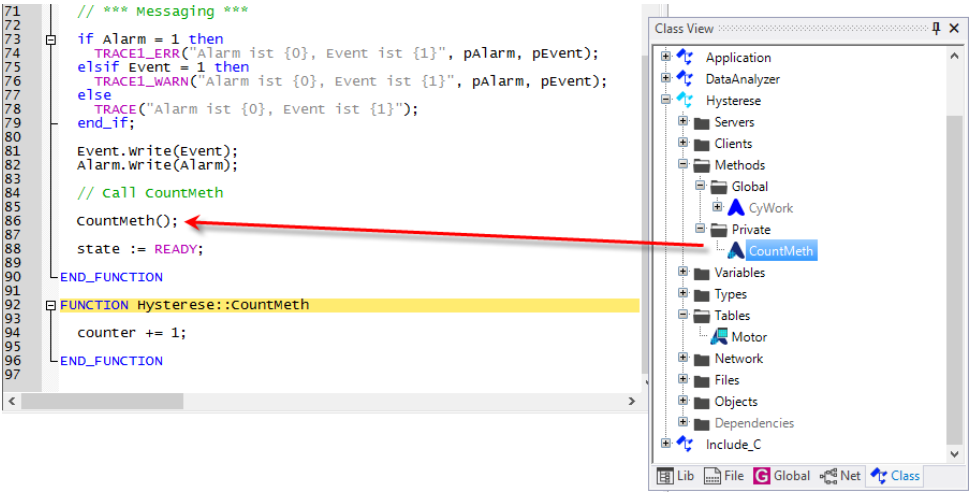
The visualization object, _Lse, runs in the Background task.

Used, for example, to query an export process, or the status of the visualization.

3.1.3.2 Calling a Method

With the following syntax, a function called **CountMeth** is called in the CyWork method.

During processing of the CyWork method, the program will jump to MyMethod and execute the instructions therein, then return to the CyWork method. When writing the program code, it is recommended that the method, **CountMeth**, be inserted in the editor via Drag & Drop from the project tree (**Class View window**).



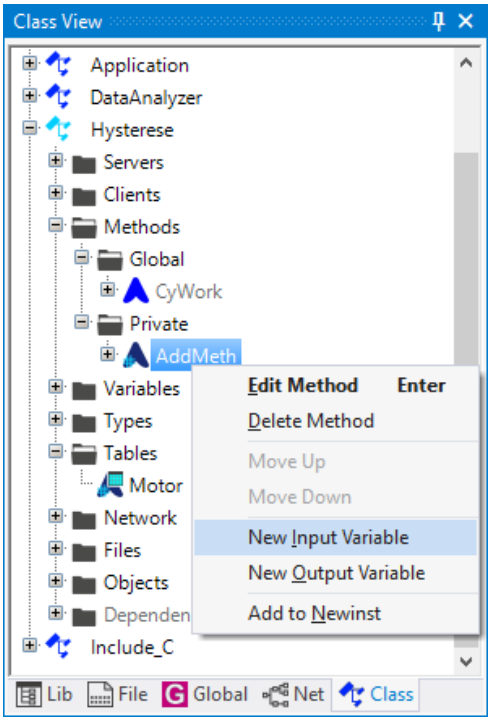
By doing so, the method is placed correctly with the parenthesis and typing errors are avoided. The **counter** variable should also be placed in the same manner.



It is recommended that all components required in the program editor be place using Drag & Drop.

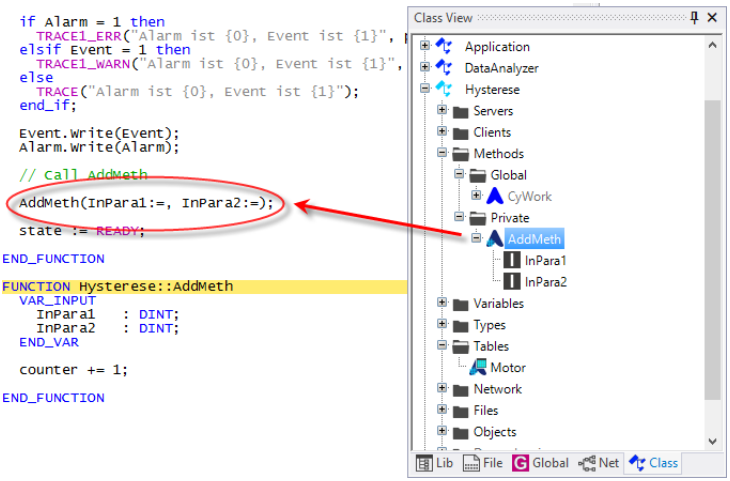
3.1.3.3 Calling Methods with Transfer parameters

To transfer parameters of a method, they must be defined in the method (**AddMeth**) to be called.

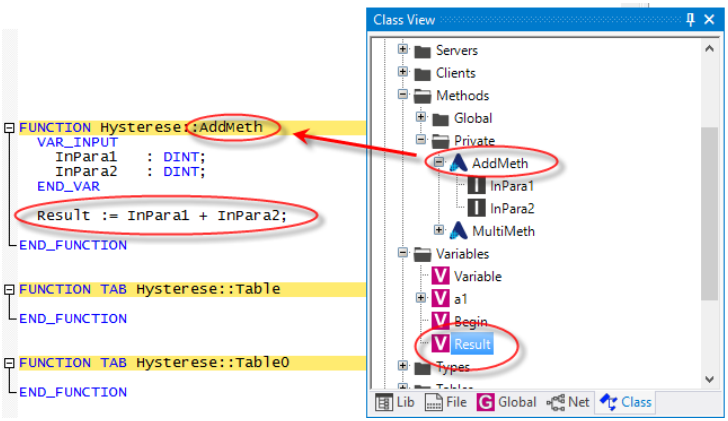


The **AddMeth method** should transfer two values, add them then write them to the **Result** variable.

Place the method in the program code editor using Drag & Drop. The program code for the method should appear as shown in the following illustration. The parameters, which the method should transfer, are displayed between the parentheses.



The **AddMeth** must then send the values assigned to the parameters. The result of the addition is written to the Result variable.



Methods can transfer values in the following ways:

The parameters can be assigned to a constant or variable:

```
AddMeth(InPara1 := 45, InPara2 := 10);
```

A constant or variable can be transferred to a method directly:

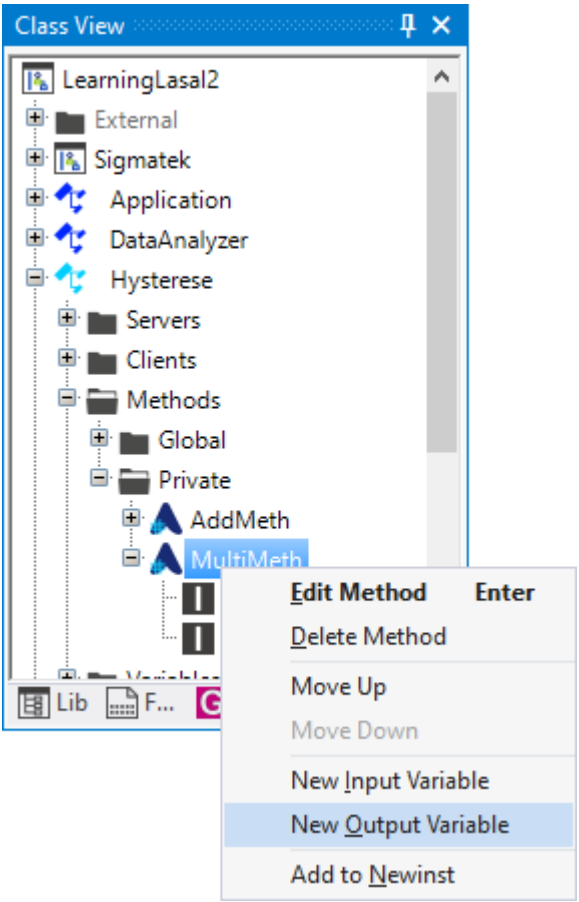
```
AddMeth(45,10);
```

A client or server can also be transferred to a method:

```
AddMeth(Client1, Client2);
```

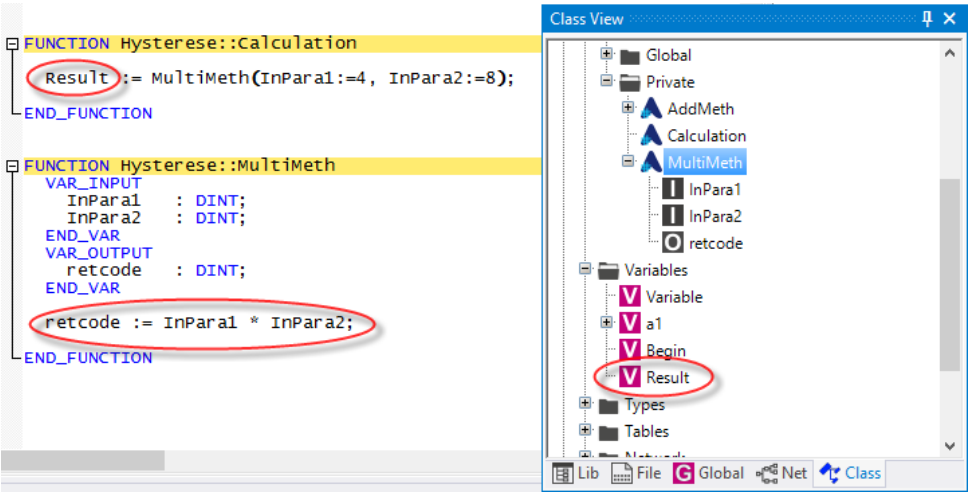
3.1.3.4 Calling a Method with Transfer and Return Parameter

In order for a method to retrieve a value, a return parameter must be defined in the method.



When creating the program, it is again recommended that the necessary components be inserted into the program code using Drag & Drop.

The **MulMeth method** should transfer two values, multiply them and over the return parameter, write the result to the **Result** variable.

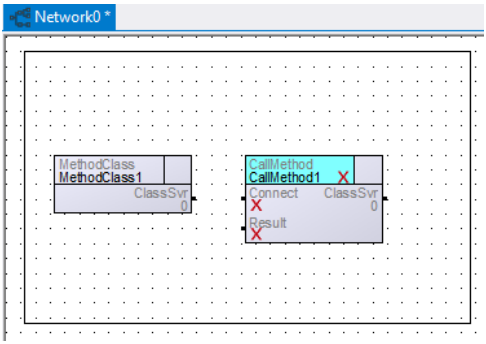


The return parameter contains a valid number after the program line, retcode := InPara1 * InPara2, has been executed. Before this, the return parameter contains an arbitrary value.

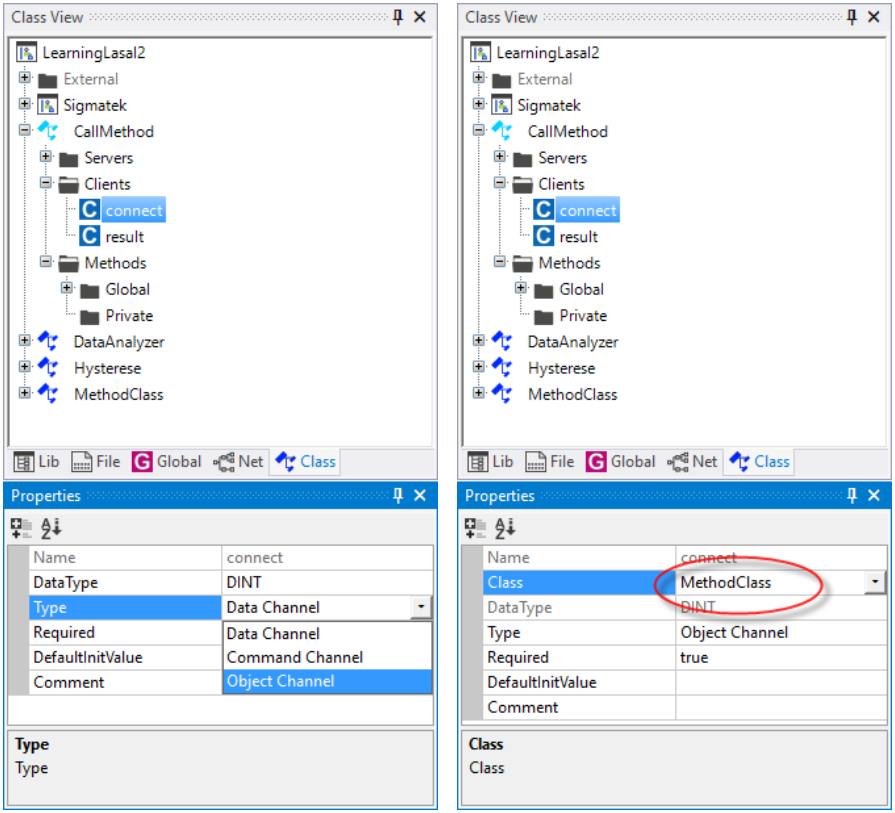
The value in the return parameter is transferred after the method is exited. In this case the **Result** variable contains the result of the calculation. The processes can be observed in the debugging window.

3.1.3.41 Calling a Method via the object channel

Using object channels, methods can be called from another class. This is only possible when the method in the other class was defined as a global method. For the following example, two classes (**MethodClass** and **CallMethod**) are defined.



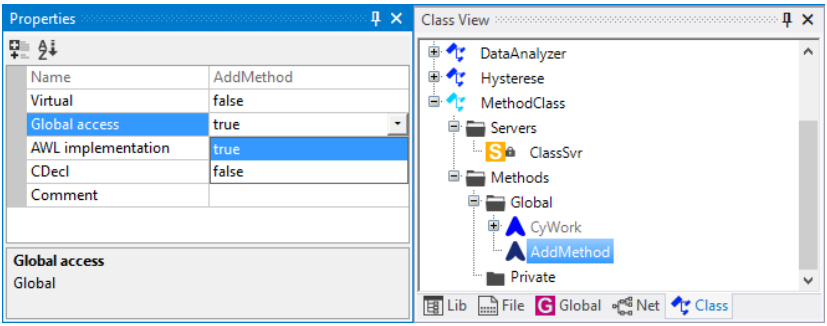
In the **CallMethod** class, only the **CyWork method** is programmed. In this class the **connect** client must be assigned as an **object channel** to the **MethodClass**. The **result** client is assigned as normal to display the return value.



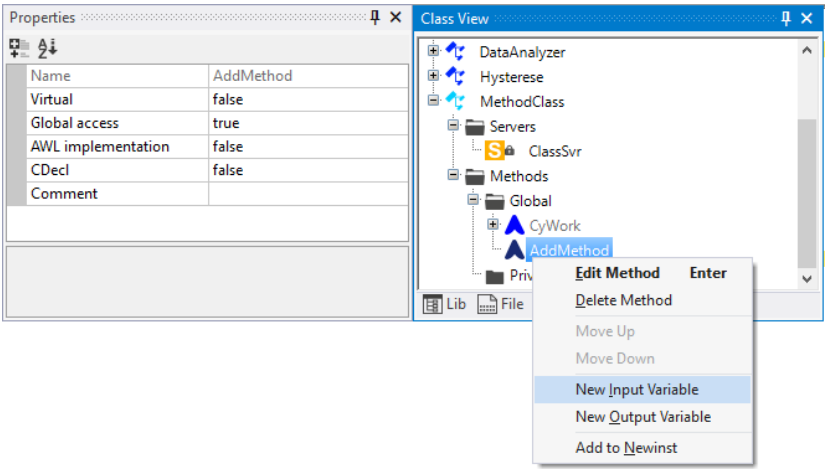
With these settings, the relationship between both classes is established. Now the **connect** client can be connected with the first server (**ClassSvr**) of the **MethodClass**.

Since it is now possible to call methods from the MethodClass, a method, **AddMethod**, is created in this class.

Only global methods can be called over an **object channel.**



Because values should be transferred to this method, parameters must be assigned (**Value1** and **Value2**). In addition a return parameter is required (**sum**).



Insert the following code for the **AddMethod**:

```
FUNCTION GLOBAL MethodClass::AddMethod
  VAR_INPUT
    Value1  : DINT;
    Value2  : DINT;
  END_VAR

  Sum := Value1 + Value2;

END_FUNCTION
```

This method will be give two value, which will be added. The call for this method occurs in the **CyWork** method of the **CallMethod** class. Insert the code below.

```
FUNCTION VIRTUAL GLOBAL CallMethod::CyWork
  VAR_INPUT
    EAX  : UDINT;
  END_VAR
  VAR_OUTPUT
    state(EAX)    : UDINT;
  END_VAR

  state := READY;

  result := connect.AddMethod(50, 100);
END_FUNCTION
```

By writing **AddMethod(50, 100)**, the method from **MethodClass** is called. The word, **connect** is a keyword and means the **connect** client can call "global methods". If the connect client had another name (e.g. **set**), the instruction would be written as **Set.AddMethod(50, 100);**.

3.1.3.5 Local Variables of a Method

If a variable is defined in a method as local, it exists only until the method is ended.

That means, so long as the program is running the method, the local variable contains the values. Once the method is processed, the values are lost. Caution must be take to initialize all local variables before use since these variables contain whatever value that happens to be at the memory address at the time. The address of local variables is not defined so they can be located anywhere in memory.



All local variables must be initialized before being used, as the values of the variables can be arbitrary when the function is started.

```
FUNCTION VIRTUAL GLOBAL Methoden::CyWork
  VAR_INPUT
    EAX   : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)   : UDINT;
  END_VAR
  VAR
    Number1   : DINT;
    Number2   : DINT;
    Sum       : DINT;
  END_VAR

    Number1 := 1;
    Number2 := 4;

    Sum := Number1 + Number2;

  state := READY;

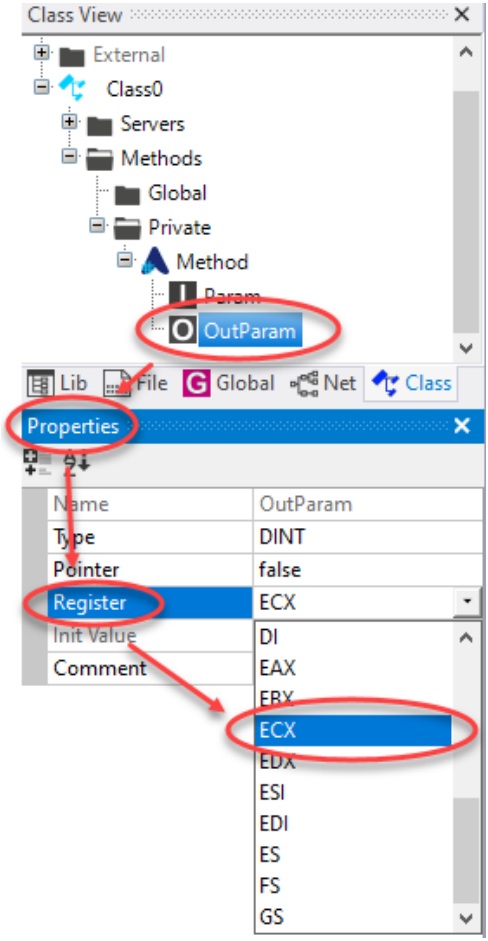
END_FUNCTION
```

As mentioned, the variables in this method are first initialized.

3.1.3.6 Register settings for IO variables

Overview

The properties of input or output variables are similar to the properties of a "normal" variable - with one significant difference: there is the additional property "**Register**" here:



This can be used to determine how the return values for this particular function call should be handled. By default the **EAX** register is used for returning values. Since this register can handle almost all internal CPU operations, it is generally the best way to pass data. However, it is now possible to select a different register. This has advantages and disadvantages.

Advantages

- This LASAL CLASS 2 function is intended **for programmers** who write code in AWL (also known as **assembler**), which allows finer tuning and control over where the data is stored in a given function call

If you **do not develop an AWL function**, there is **no reason to change the default behavior of the compiler**, because you have to accept at least the following disadvantages:

Disadvantages

- Note that for most programming languages there is a standard ABI (Application Binary Interface), which regulates how information is passed on. Among others, it is used for programming languages such as C. Since this is a standard, changing the register that is used to return a value according to this standard can lead to an impairment of interoperability with other programming languages.
- Another drawback: LASAL CLASS 2 extends optimization to the limit where a program can still be debugged. Complex optimizations impair debugging quality. Since the LASAL CLASS 2 compiler does not produce very optimized code for this reason, this option will not be of much use most of the time.
- The third point, as mentioned earlier, is that EAX is a special register in the CPU architecture, and it may be the shortest way to transfer the information to other places. Only in very special cases one could benefit from a different behavior

Register subdivision

Another complex concept in an Intel CPU is that a 32-bit register can be split into two 16-bit registers or four 8-bit registers. This is due to the development history of these processors, starting with the 8080 8-bit processor in April 1974, through the 8086 in June 1978, which was a 16-bit evolution of the 8080, to the introduction of the 80386 in October 1985, which evolved into a 32-bit processor. The following figure shows how these registers are grouped:

i386 registers

^{31} ... ^{15} ... ^{07} ... ^{00} *(bit position)*			
Main registers *(8/16/32 bits)*			
EAX	AX	AL	Accumulator register
ECX	CX	CL	Count register
EDX	DX	DL	Data register
EBX	BX	BL	Base register
Index registers *(16/32 bits)*			
ESP	SP		Stack Pointer
EBP	BP		Base Pointer
ESI	SI		Source Index
EDI	DI		Destination Index
Program counter *(16/32 bits)*			
EIP	IP		Instruction Pointer
Segment selectors *(16 bits)*			
	CS		Code Segment
	DS		Data Segment
	ES		Extra Segment
	FS		F Segment
	GS		G Segment
	SS		Stack Segment

Status register

^{17} ^{16} ^{15} ^{14} ^{13} ^{12} ^{11} ^{10} ^{09} ^{08} ^{07} ^{06} ^{05} ^{04} ^{03} ^{02} ^{01} ^{00} *(bit position)*																	
V	R	0	N	IOPL	O	D	I	T	S	Z	0	A	0	P	1	C	EFlags

For more information on the i386 processors, see: [i386 on Wikipedia]



Attention

Note that when using partial registers, a corrresponding data type (SINT, INT, USINT or UINT) must be used!

Differences to ARM compiler

The register directory shown **in LASAL CLASS 2 is based on the Intel CPU**. The registers of the ARM CPU are **always 32 bits wide** and have other names, like R0, R1, R2,...

If the ARM compiler is used, the Intel universal registers are assigned according to the table below:

Intel CPU-Register	ARM CPU-Register
EAX	R0
EBX	R1
ECX	R2
EDX	R3

Note that the ARM CPU has a completely different architecture than an Intel CPU, but the advantages and disadvantages are roughly identical.



Hint

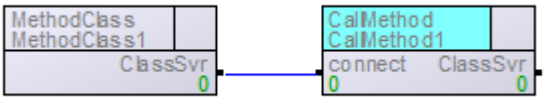
When you use this function, you are at an Assembly Language Level (AWL). You can check the disassembled output by entering the disassembly mode in the editor with the key combination [CTRL]+[SHIFT]+[D]

3.1.3.7 The Read and Write Methods of Server

The **Read** and **Write methods** are server-based methods.

3.1.3.71 Read methods for servers

Using the Read method, the value of a server can be read by another object. To demonstrate this, the class CallMethod is first created with the program code shown below. The **Write Protected** setting of **ClassSvr** in the **CallMethod class** is set to **FALSE**.




```
FUNCTION VIRTUAL GLOBAL CallMethod::CyWork
  VAR_INPUT
    EAX  : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)  :UDINT;
  END_VAR

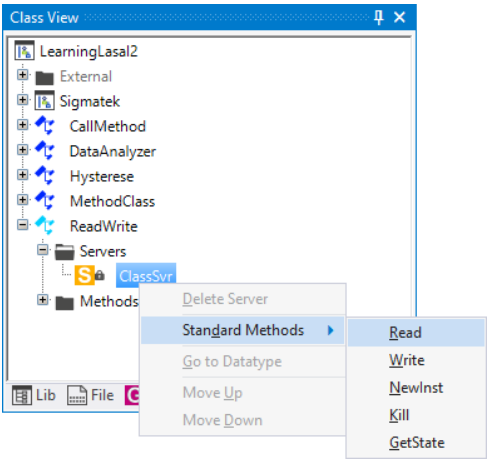
  if ClassSvr <> 0 then
    Call      := Call.Read();
    ClassSvr  := 0;
  end_if;

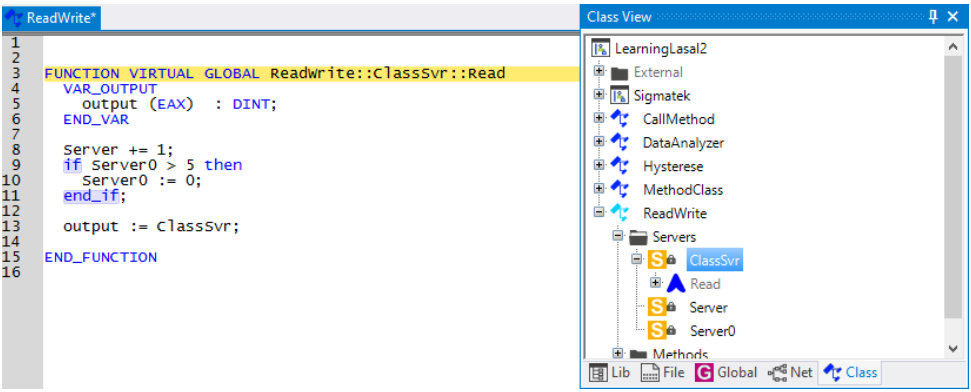
  state := READY;

END_FUNCTION
```

Next, create the ReadWrite class is assign the Read method from the **ClassSvr** then insert the code shown in the following illustration. With these steps, the Read method of the ClassSvr is programmed in the Readwrite class. This class does not require a cyclic task.

The Read method is assigned as follows:

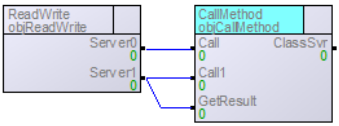




Download the project to the control and test the function in which a value unlike 0 is entered in the ClassSvr of the **objCallMethod object**. After each entry the Call client from the **objReadWrite** increments by 1 and is reset to 0 when the number 5 is reached. After programming, the Read method transfers the result through the return parameter **output**.

3.1.3.72 Write methods for servers

To explain the Write method, both classes are expanded as shown below. First, begin with the **CallMethod class**.



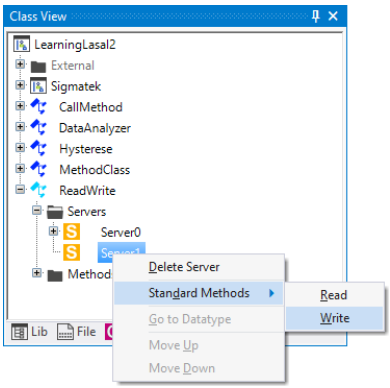

```
FUNCTION VIRTUAL GLOBAL CallMethod::CyWork
  VAR_INPUT
    EAX   : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)  : UDINT;
  END_VAR

  if ClassSvr <> 0 then
    Call      := Call.Read();
    Call1.Write(460);
    GetResult := GetResult.Read();
    ClassSvr  := 0;
  end_if;

  state := READY;

END_FUNCTION
```

The **Write method** for server1 of the ReadWrite class must now be programmed. The Write method of the server is assigned as follows:



Insert the following code for the **Write method**:

```
FUNCTION VIRTUAL GLOBAL ReadWrite::Server0:Read
  VAR_OUTPUT
    output (EAX)  : UDINT;
  END_VAR

  Server1 += 1;
  if Server0 > 5 then
    Server0 := 0;
  end_if;

  output := Server0;

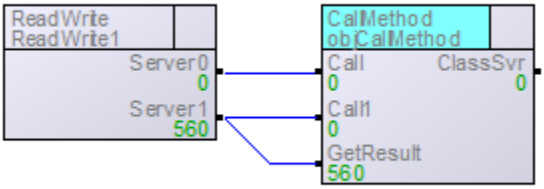
END_FUNCTION

FUNCTION VIRTUAL GLOBAL ReadWrite::Server1::Write
  VAR_INPUT
    input (EAX)  : DINT;
  END_VAR
  VAR_OUTPUT
    result (EAX)  : DINT;
  END_VAR

  Server1 := input + 100;
  result := Server1;

END_FUNCTION
```

Download the project to the control again and enter a number unlike 0 in the **ClassSvr** of the **ObjCallMethod object** to test the function. The Write method of the ReadWrite class is called over the **Call1 client** and the value is transferred. In the Write method for **Server1** of the **ReadWrite** class, the value is changed (100) and written to **Server1** through the return parameter **result**. Through the **Read method** of the **CallMethod class**, the new value is retrieved by the **GetResult client**.



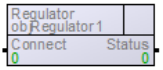
3.1.4 Objects

3.1.4.1 Complex Objects general

Complex objects can be used to combine several objects in a new class, giving it the functions of many classes.

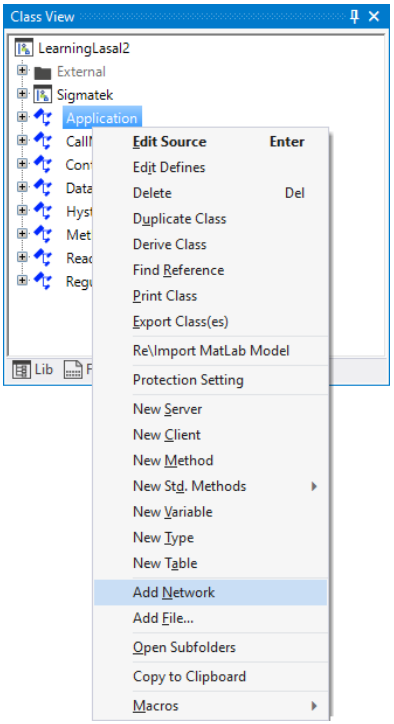
To illustrate, two classes are created that appear as follows. The code for these classes is programmable as required for the individual application:





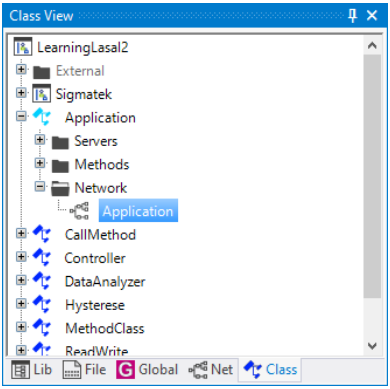
To implement a complex object with the function of both classes (**Controller** and **Regulator**), a class (**Application**) must be created. This process is called aggregation.

The actual complex class is defined in a new network created for it as shown below.



This network serves as the reference network of the complex class, in which standard initial values can be entered.

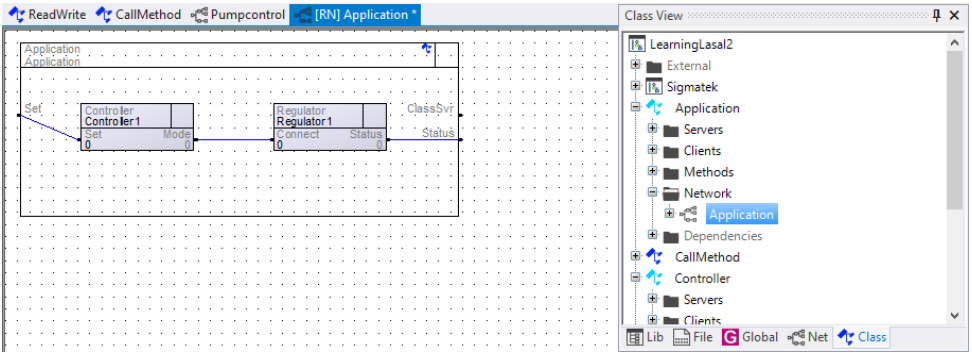
With a double click on **Application**, the network for adding objects to the complex class is opened.



The classes designated for the complex class can now be inserted in the network via Drag & Drop.

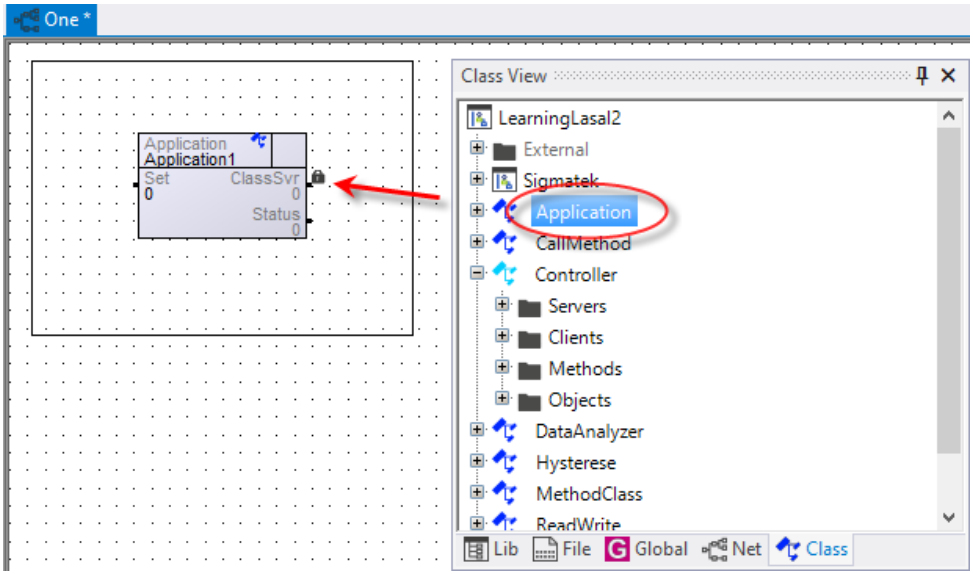
In order to distinguish between a reference network of the complex class and the interior of an object at first glance, "[RN]" is inserted before the class name in the editor bar.

This is the actual definition of the complex class:

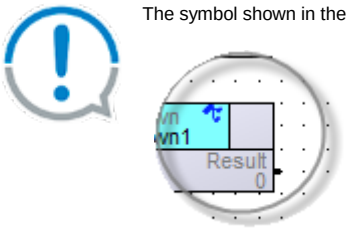


The finished complex object can now be placed in another network.

The newly created complex object has the properties of both the **Controller** and **Regulator classes**.



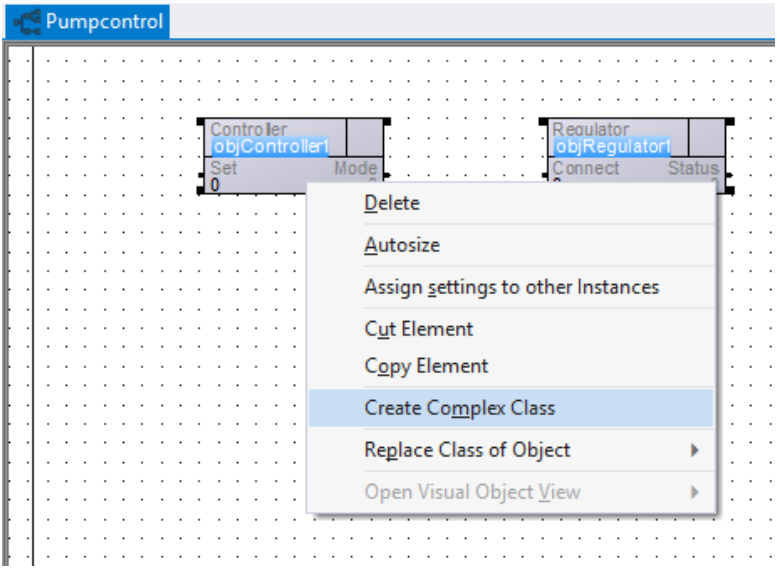
The symbol shown in the class below identifies a **Complex Object**.



Another possibility to create a complex class is to mark one or more objects in the network editor and to select **"Create Complex Class"** in the context menu of the right mouse button. Thereby the marked objects are automatically overtaken to the inner life of the complex class.

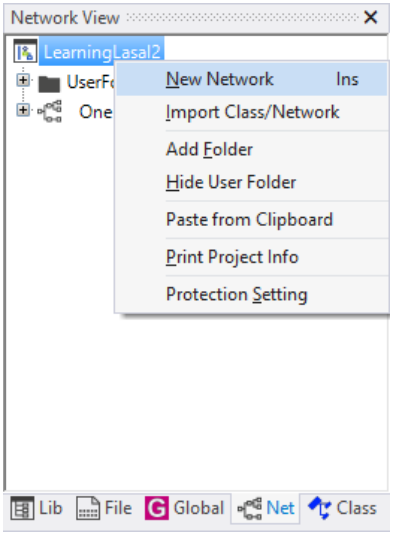
Attention!

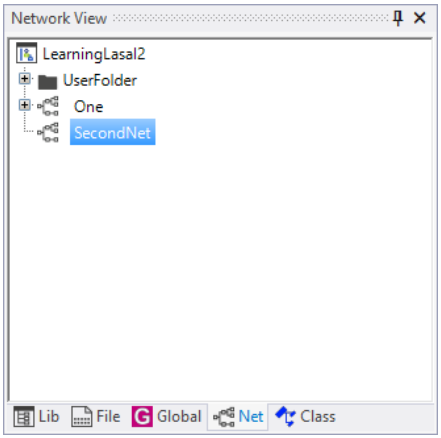
To see the menu entry **"Create Complex Class"**, in the selected objects there must be **no clients connected to IO objects!** If a connection of the clients to IO objects is necessary, the according objects **first must be inserted in a complex class**, the according **clients dragged to the outside**, and **then the IO objects can be connected to the clients of the complex class object!**



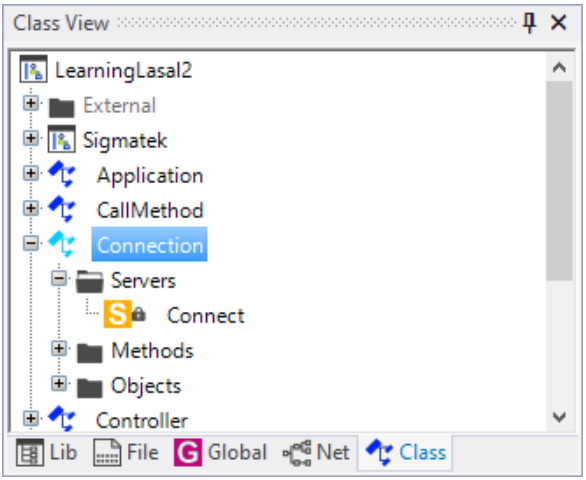
3.1.4.2 Connecting Objects in different networks

To make the construction of a project clearer, objects can be distributed over several networks. To accomplish this, create a new network (**SecondNet**) in the **Network View window**.

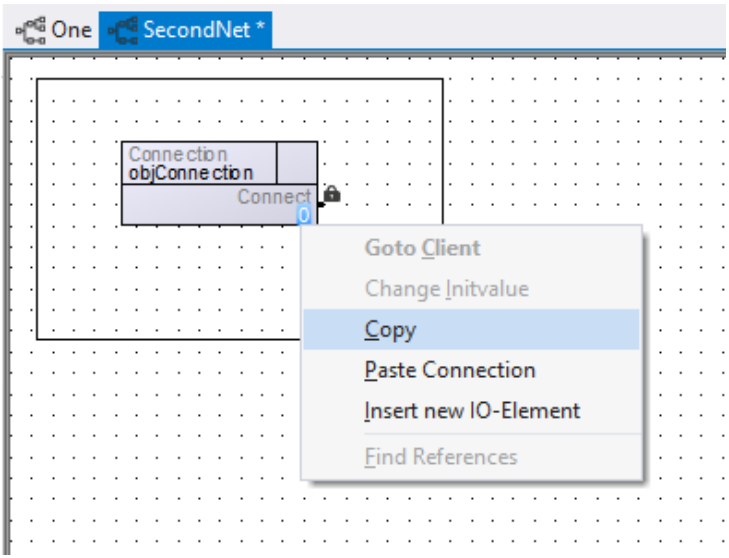




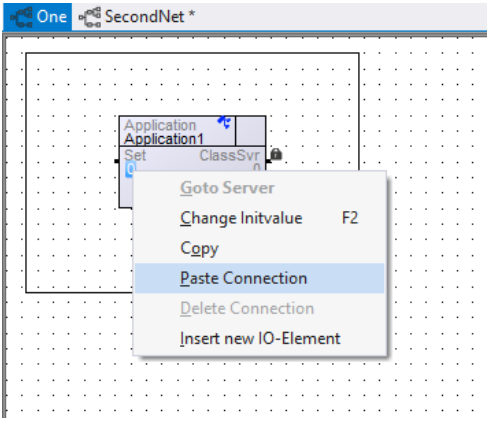
Also define a new class (**Connection**) and place it in the network, **SecondNet**.



With a right click on the Connect server, open the context menu and select **Copy**.

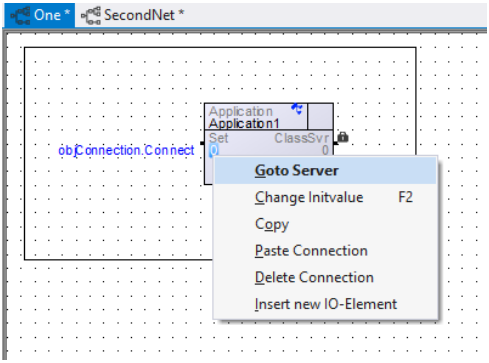


Switch to the **One network** and select **Paste Connection** from the context menu of the **Set client** to establish the connection.

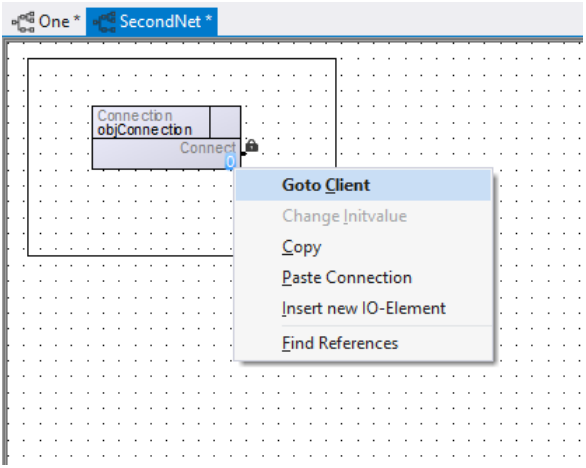


The connection between the objects in both networks is complete. The Copy/Paste mechanism for cross-network connections is reversible. This means, that connections to the **Copy client** and the **Paste Connection** server can be made.

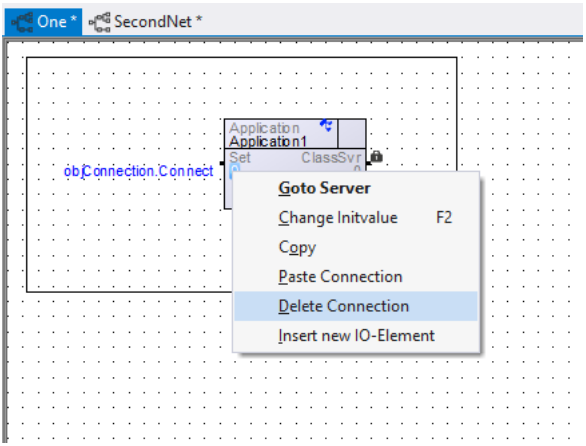
To switch from the **Set** client to the connected server, **Connect**, select **Goto Server** from the context menu of the client.



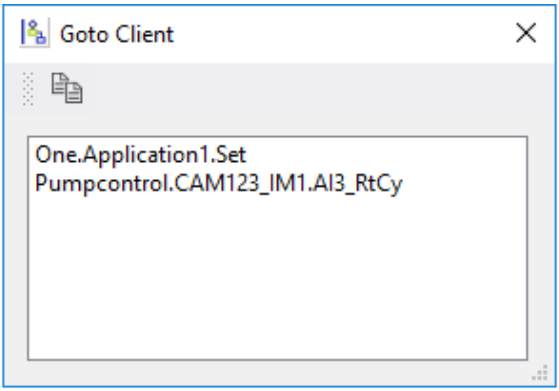
The network, **SecondNet** then appears on screen and the server (**Connect**) connected with the client (**Set**) is marked. It is also possible to switch from the **Connect server** to the **Set Client**. Simply select **Goto Client** in the context menu of the server.



On the client side of a global connection, the latter can be deleted with **Delete Connection**.



If several clients are connected to a server and **Goto Client** is selected, a selection window appears in which the client of both objects can be chosen.



In the lower right corner of this window, the size can be changed. The size remains constant once it's set.

3.1.4.3 Moving and copying of objects

The moving and copying of one or more objects between two networks in LASAL CLASS 2 is possible via the clipboard.

First the corresponding network elements (objects, comments) are selected in the network. Then the commands **"Cut Element"** (for moving), respectively, **"Copy Element"** (for copying) can be selected via the context menu of the right mouse button. The key combination **"Ctrl + X"** (moving) and **"Ctrl + C"** (copying) has the same effect like these commands.

Afterwards via the context menu of the right mouse button and the commands **"Paste Element(s)"**, respectively, **"Ctrl + V"** the network elements are insert from the clipboard. The mouse position defines the paste position at insertion via the command **"Paste Elements"**. At insertion with **"Ctrl + V"** the position has to be defined before via mouse click, otherwise the position of the source network is used.

The functionality for moving and copying of objects via clipboard offers the following possibilities:

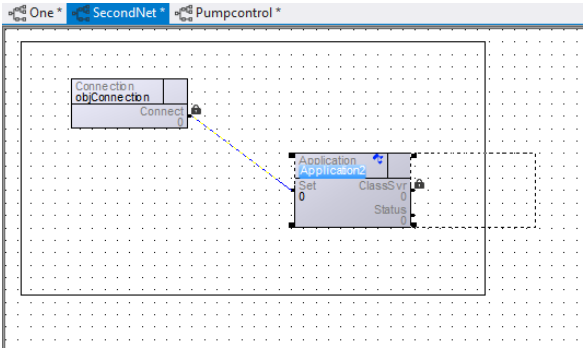
- Copying of objects in the same network.
- Copying / moving of objects between two objects.
- Copying / moving of objects from and to networks of complex classes.
- Copying of objects from Library networks.
- Copy objects between two open LASAL instances

Object properties and connections remain at copying and moving.

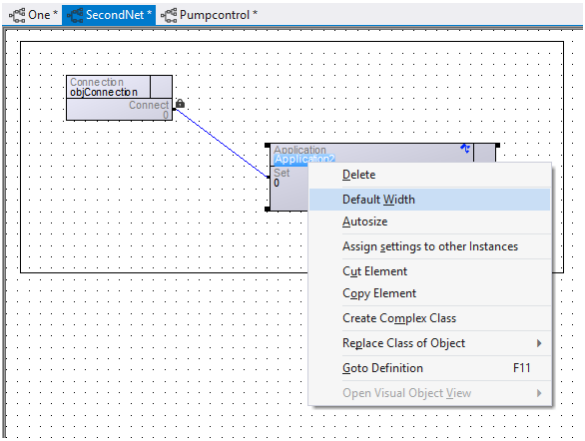
In addition, classes, networks and objects can also be copied between two open LASAL instances by Drag & Drop. Classes and networks can be simply pulled into the other instance while pressing and holding the left mouse button. When copying objects between two LASAL instances, the Alt key – to move within the network – must also be held.

3.1.4.4 The Default Width and Auto size Functions

The size of an object (**objApplication**) can be changed if, for example, the object name is cut off because the window is too small. The size of an object can be changed on the left or right side by pulling on the selection boxes. It is important that, in doing so, the setting is assumed by all objects of the class.

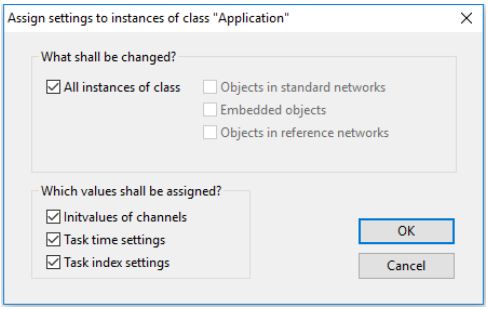


Simply select **Default Width** from the context menu of the desired object to reset the origin length of the object window.



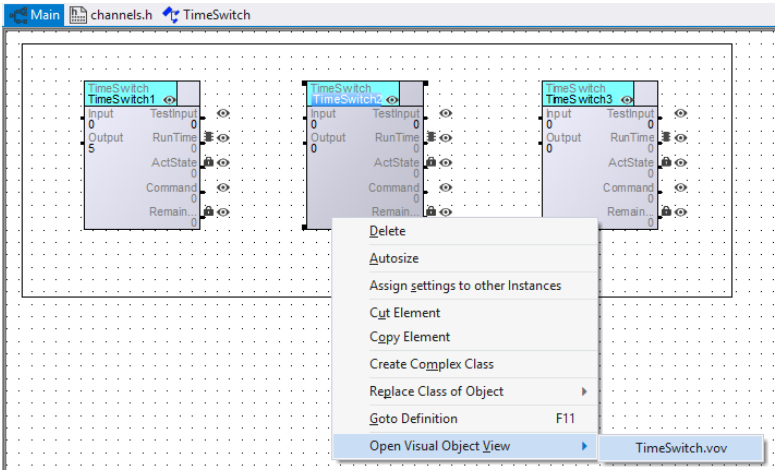
With the selection of **Autosize**, the object window automatically adjusts to the length of the object name.

Selecting **Assign settings to other instances** allows initial values, task times and task indexes of an object to be transferred to other instances of the class.A dialog box opens that enables the user to specify which properties are transferred to which objects.



With **Go To Definition** (or the **F11** key), the program jumps directly to the object in the class tree.

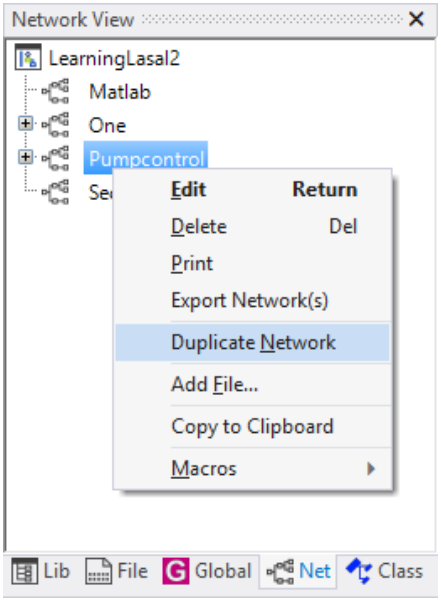
If a *.VOV file (Visual Object View) has already been created with the LASAL SCREEN Editor and a class was added with **Add File**, the menu item **Open Visual Object View** can be selected.



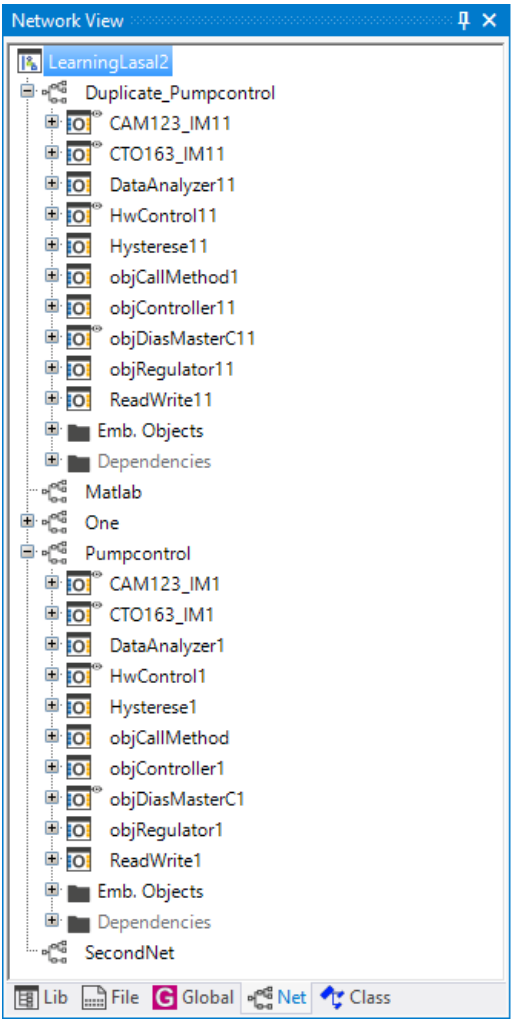
If the item is selected, the LSE kernel opens and the screen created in LSE as well as the exported Visual Object View are displayed. If offline, the default server and client values are set. If online these values are read directly from the control.

3.1.4.5 Duplicating Object Networks

To duplicate an existing object network, in the **Network View window** of the desired network, open **Duplicate Network**.



Assign a name to the duplicated network (**Duplicate_Pumpcontrol**) and open it. As can be seen, the duplicate network contains the same objects as the original network.



3.1.4.6 Search and Goto Definition in object networks

In LASAL CLASS 2 it is possible to search for elements within an object network. Via the menu command Edit\Find…, respectively, the key combination **Ctrl + F** objects, classes, server, clients and initialization values can be searched for in an opened object network.

Through a right mouse click on an element the **Goto Definition** command can be selected in the context menu. This command marks the concerned element in the tree and can also be called via **F11**.

3.1.5 Complex Classes

3.1.5.1 Important Information on Deriving Classes

Deriving means that a class derived from another class (base class) in order to assume its characteristics and processes.

Only classes are derived, not methods.

Methods are overwritten.

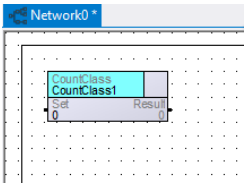
Methods, which are to be overwritten, must be defined as virtual methods (**VIRTUAL GLOBAL**).

With virtual methods, LASAL chooses automatically whether the method of the base class or the derived class should be called using a method table as long as it was overwritten.

The advantage of deriving classes is that the methods of the base class can be used and possible changes must only be made once. By deriving a class, the new class inherits the methods of the base class.

3.1.5.2 Deriving Classes and Overwriting Methods

To illustrate this exercise, a class (**CountClass**) with the following program code is created:



The method **CountMethod** is defined as a virtual method!

```
FUNCTION VIRTUAL GLOBAL CountClass::CyWork
  VAR_INPUT
    EAX  : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)  : UDINT;
  END_VAR

  if Set = 1 then
    Result  := CountMethod();
  else
    Result  := 0;
  end_if;

  state := READY;

END_FUNCTION

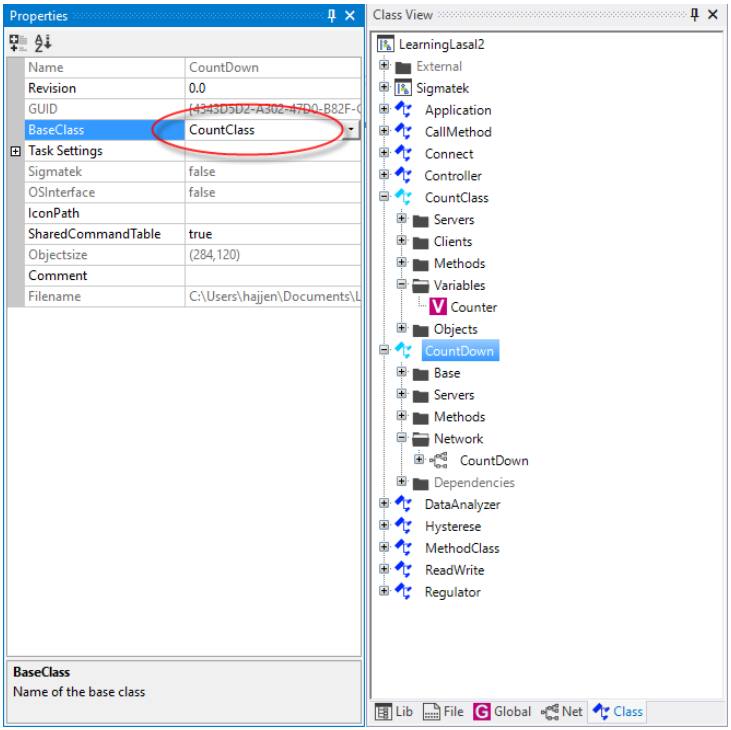
FUNCTION VIRTUAL GLOBAL CountClass::CountMethod
  VAR_INPUT
    retcode  : DINT;
  END_VAR

  Counter += 1;
  retcode := Counter;

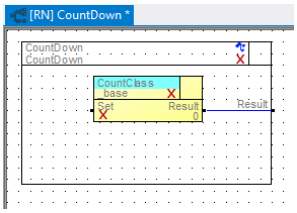
END_FUNCTION
```

Note:

Test whether the value in the **Result** server actually counts upward by writing to the **Set** client. In the next step, a new class (**CountDown**) is created in a new network as previously mention with the complex object. By selecting the **CountDown** class, the base class can be defined in the **Properties window**.

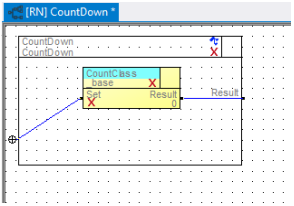


With a double click on the new network containing the **CountDown class**, the network with the **base class** is opened.

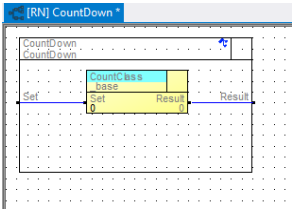


Moreover, the **Set** client must be connected to the **CountDown** class.

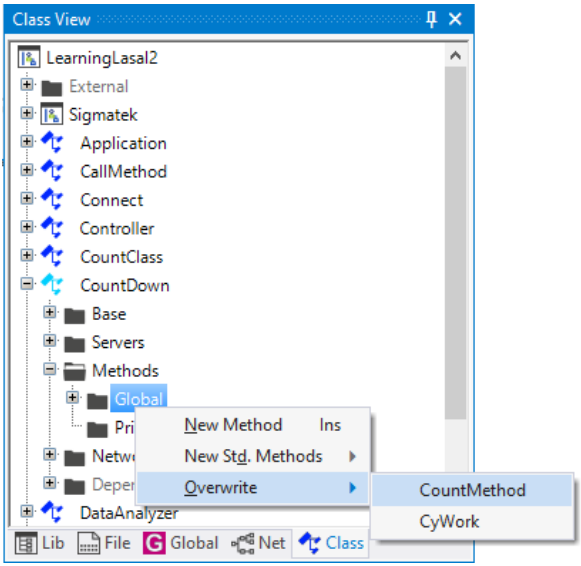
Create the connection as shown in the next graphic by clicking outside of the left border of the class.



This graphic shows the completed **CountDown class** with the correct connections.



With a right click on the **method** of the class (**CountDown**) in the context menu under **Overwrite**, select a method (**CountMethod**) of the base class (**CountClass**).



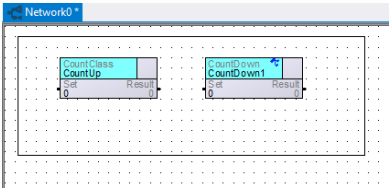
Insert the following program code in the new **CountMethod** if the method of the base class was over written.

```
FUNCTION VIRTUAL GLOBAL CountDown::CountMethod
  VAR_INPUT
    retcode  : DINT;
  END_VAR

  Counter -= 1;
  retcode := Counter;

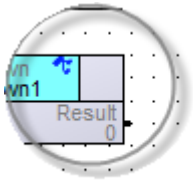
END_FUNCTION
```

After placing the new object of the **CountDown** class, download the project to the control and test the function.



Test the function by setting both **Set** clients. In the **CountUp object**, the value in the **Result server** should increment and the value in the **CountDown object** should decrement.

The symbol shown in the class below identifies a **Complex Object.**



3.1.6 Processing Tasks

3.1.6.1 General Information about the Different Task Types

The SIGMATEK operating system for LASAL is capable of multitasking, which is of great significance in control technology. The operating system is responsible for the processing of various tasks in objects assigned by the user and must ensure that all assigned objects are processed within their specified period.

3.1.6.11 RealTime

This task has the highest priority. Only time-critical tasks (e.g. controlling axes) should be defined as real-time tasks.

3.1.6.12 Cyclic

This task is middle priority. In the settable time slot (= cyclic time), the task is processed cyclically (= repeated). Generally, it is used for process control.

3.1.6.13 Background

This task has the lowest priority. The background is processed in time remaining after the RealTime and Cyclic tasks have been completed. It is used for example, to query an export process or status of the visualization.

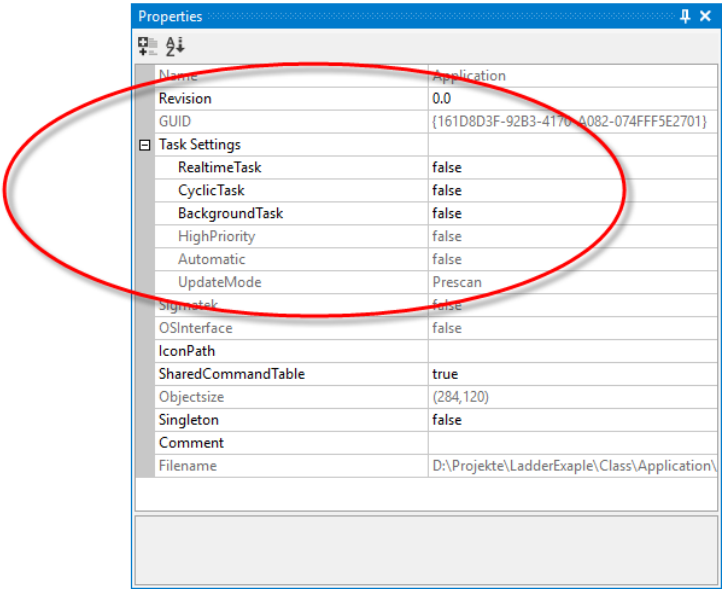


With the use of background tasks, the derived class cannot contain any virtual methods!

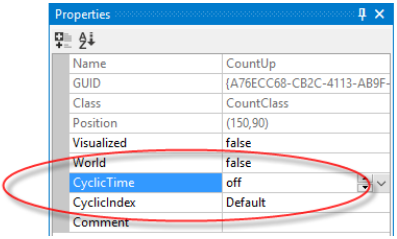
3.1.6.2 Configuration of the Tasks

3.1.6.21 In the class

In the properties window of the class, the property group "Task Settings" can be found. When creating a new class, all three tasks are set to **false**:



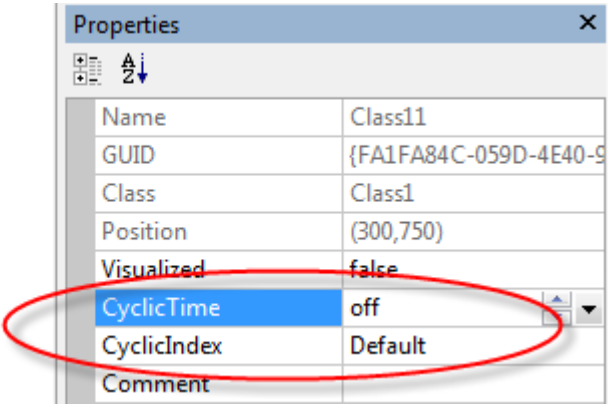
If a property (here: CyclicTask) is now set to **true**, the new property appears with which the corresponding time can be set.



Each project of this class placed later in the object network, will have this property as a default setting.

3.1.6.22 In object

If an object is placed in the object network, no task settings group can be found at the properties of the Properties window, but only the fields of the defined task types with the initial values set there:



These can then be overridden for each object.



Only when the appropriate time in the **object** is greater than zero (= off), the task is assigned the object!

3.1.6.3 Processing the tasks by the CPU

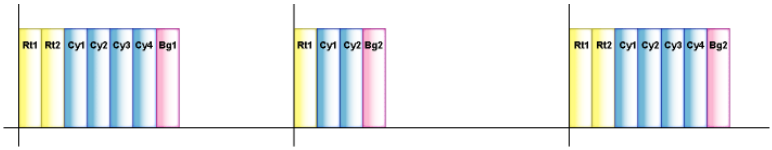
Brief description of the graphic

Rt => Real time work

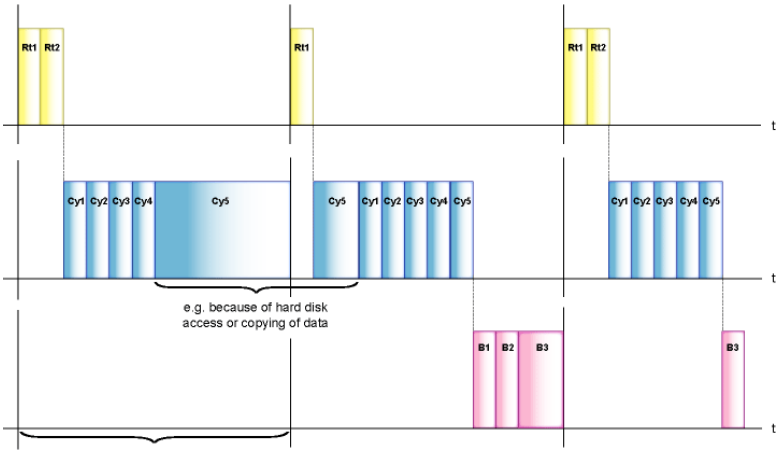
Cy => Cyclic work

Bg => Background

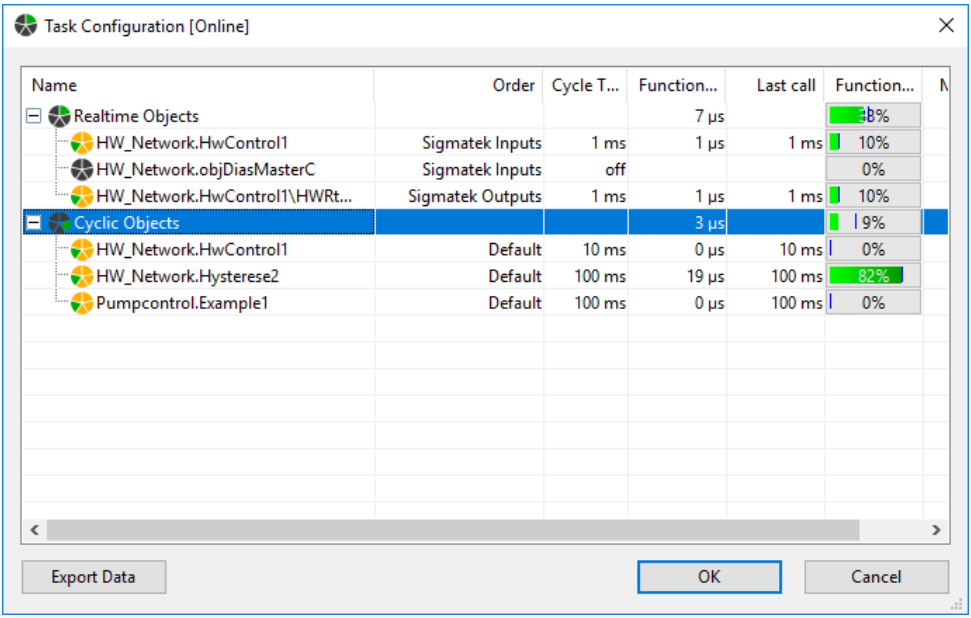
In the following graphic, the priority of the different tasks is clearly shown. Real time tasks have the highest priority. Cyclic tasks are processed next. With the remaining time, the operating system calls the background tasks.



According to this diagram, Rt1, Cy1 and Cy2 are processed every cycle while Rt2, Cy3 and Cy4 are processed every second cycle. The next graphic shows when a process, in the event a task with a long running program, must be interrupted. The task Cy5, can require more time when copying data. Here the task is interrupted and the Rt1 task is processed. Cy5 is then continued after Rt1 is complete.



3.1.6.4 Online configuration of the tasks

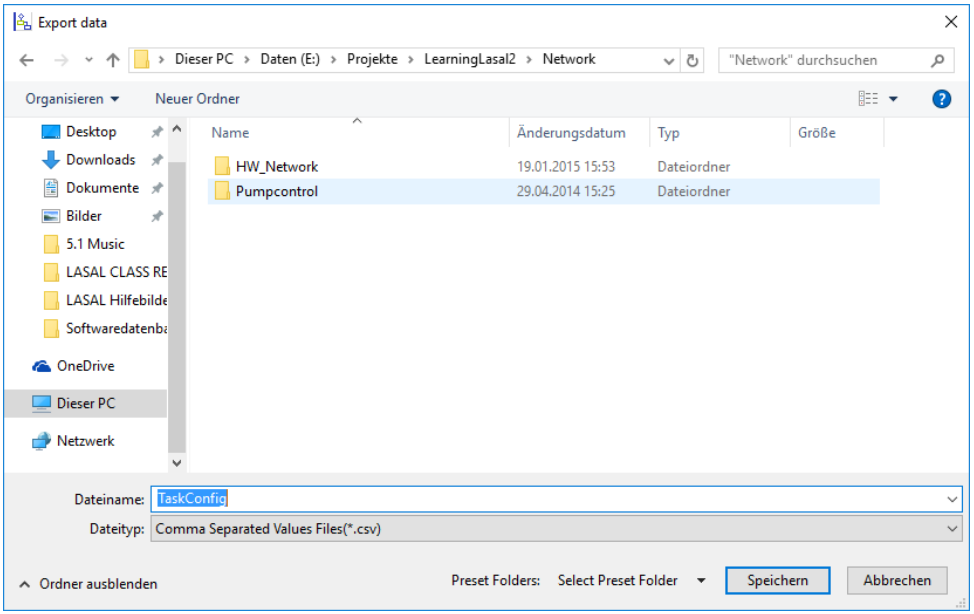


The function (**Realtime-, Cyclic** and **Background-Function**) called by the operating system must be defined in the Properties window for each object. The time span for each must also be specified in milliseconds.

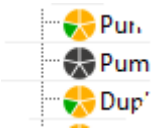
The Order and Cycle Time fields can be changed. The properties for each task object can be changed (activated/deactivated, change priorities). In doing so, it is important whether target system is online with LASAL 2 or not. In offline status, the project is changed. In online status, the changes are assumed by the target system temporarily (temporary disconnection of individual tasks).

In the group lines (Realtime Objects, Cyclic Objects, Background Objects), the utility variables (Cyclic Average, Cyclic Maximum... etc) are shown.

By clicking the Export Data button, the File Save dialog appears in which the task configuration data is exported as a *.csv file (Comma Separated Value Files) and can be edited – imported in an Excel file for example.



In the Task Configuration window, an object with a **gray background** is not cyclically called by the operating system.



If the context menu of a task is called in online mode, there are two more entries:

Reset measurement...

Here all measuring data of the the selected task are reset.

Reset all measurements

Here all measuring data of the complete group (Realtime, Cyclic oder Background) is reset.

If a group is selected (e.g. Realtime taskes), then with Reset measurement, only those utility variables of the group are reset. With Reset all measurements the variables of the underlying individual tasks are also reset.

The Task Configuration Columns:

Name

Here the cyclic objects are arranged by its task priority. The name for each object consists of "Network name.Object name".

Real Time Work

This method is called by the LASAL operating system and has the highest priority.

Cyclic Work

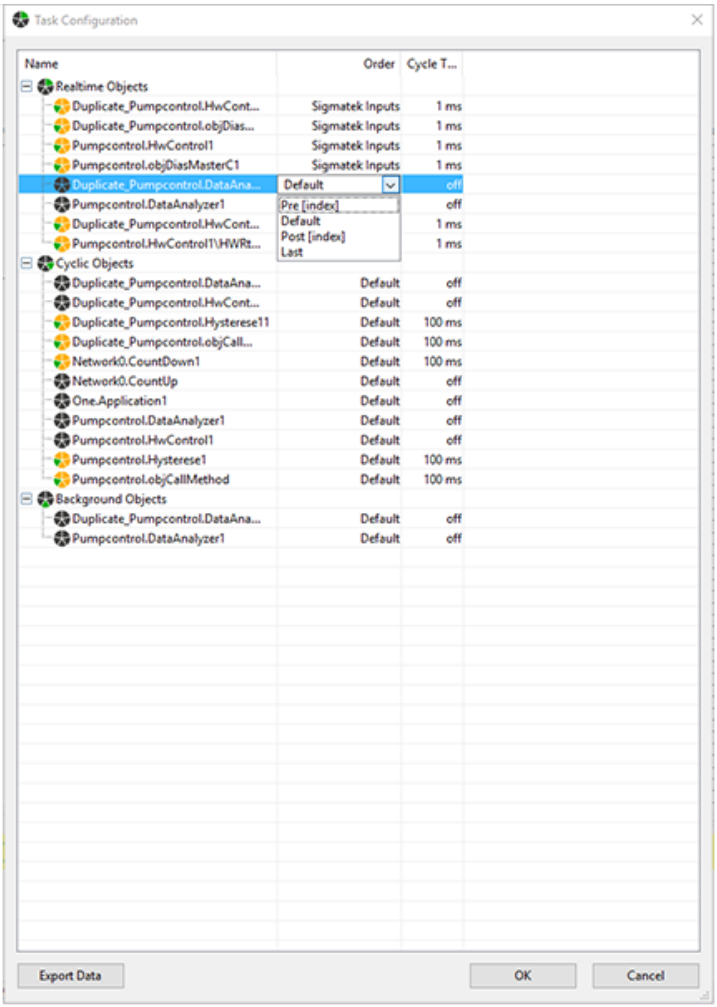
This method is called by the LASAL operating system and has middle priority.

Background

This method is the lowest priority method, called by the operating system and is used preferably for visualization.

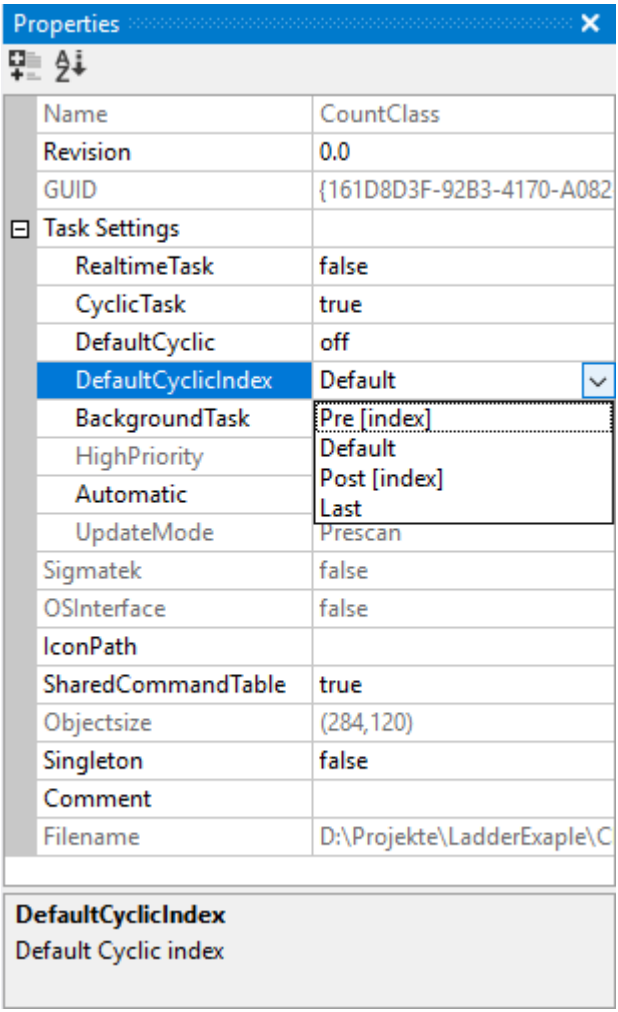
Order

Here the order of execution within a task priority is defined. There are four options: Pre (with an additional number), Default, Post (with an additional number) and Last. The tasks are processed exactly in this sequence. Where it is possible to enter an additional number - Pre and Post - the task with the lowest number is process first for tasks. For tasks that have the same priority, the sequence shown in the dialog box does not indicate the actual processing sequence. In such a case, the user cannot count on a specific sequence.



The task sequence can also be changed by Drag & Drop.This must be done offline. The task can only be moved to the very front (it is then pre) or to the very end (it is then post). Within an order (e.g. the default area), the sequence cannot be changed: it remains sorted alphabetically.

The same selection can also be made in the class/object.



Cycle Time

Here the cyclic time for the respective task is defined.

Function time

Here, the actual time the task required to last run the method can be seen.

Last call

This value shows the time span between the last and the current call of the object. An object with a cyclic time of 100ms has here the value 100ms in normal case.

If the value is higher than 100ms, the task couldn't be called immediately after expiration of 100ms because

- tasks with a higher priority allocate the system,
- the own object took longer than the defined cyclic time,
- the task time has been set to a value, which is not supported by the control (e.g. 1ms although a minimum of 2ms is possible).

Function time of max. time

This illustration shows the current time consumption of the object in proportion to the maximum reached time consumption since program start. The complete width of the "Function time of max. time" category is scaled to the "Max. time" value. The bar shows the "Function time" value. Moreover a drag indicator is implemented in this display, which displays the maximum cycle of the last seconds.

Max. time

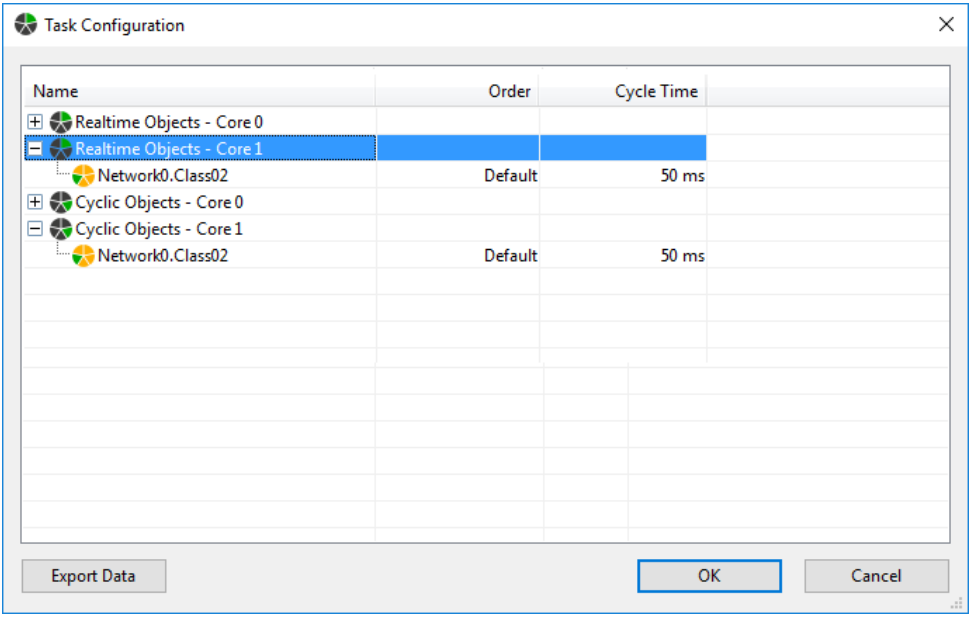
This value shows the maximum cycle since program start.

Export Data (Button)

With this button, it is possible to export the measured value as a *.csv file. It appears in the Select File dialog, in which the file name and the memory location can be defined.

Displaying the tasks when using 2 CPU cores

If in the project options, the property Use multiple CPU Core is set to true, the tasks times fort he CPU cores are shown separately in the Task Configuration window.



Additional information on the task sequence can be found in the chapters Task Sequence without RtPostScan (up to Lasal 02.02.052), Task Sequence with RtPostScan (starting from Lasal 02.02.054) and Effect of RtPostScan.

3.2 Structured Text

The structured text (abbreviation: ST) is a programming language for programmable logic controls (PLC).

ST offers more structuring possibilities than **IL** and so more and more replaces it. Complex algorithms and mathematical functions can be programmed clearer and faster in ST.

3.2.1 Structured Text Basics

3.2.1.1 Basic ST Syntax

3.2.1.11 Upper and Lower Case writing

With Structured Text programming in LASAL CLASS 2, no difference is made between upper and lower case letters (**not case sensitive**). It is recommended however, to use the same writing technique as the definition for clients, servers, variables or methods in the program code.

When these components are placed in the program code from the project tree using the drag and drop method, they are identical. With the upper and lower case writing style, the clarity and readability of the program is increased.

3.2.1.12 The Semicolon

Programs consist of a number of instructions, which are executed in sequence. Instructions specify what the program has to do and how they must occur as well as the instruction type. In the **LearningLASAL** example, the following program line appears as follows:

```
ActualValue := ActualValue.Read();
```

Each instruction requires a marking that signals the **end of the instruction**. In Structured Text, a semicolon is used. If the semicolon is forgotten, compiler warning occurs.

Although it has no meaning, several semicolons can be written on after the other without an explicit instruction between them:

```
;;;;
```

Because an instruction is clearly ended by a semicolon, several instructions can be written in a line. Conversely, instructions can be divided over several lines without a compiler problem.

```
ActualValue := ActualValue.Read(); MaxValue := MaxValue.Read();
```

Because the program code would be difficult to read, the above writing technique should be avoided.

3.2.1.13 Program Structure

The program structure in LASAL CLASS 2 is class oriented, whereas the classes are further divided (or can be divided) into methods. Each method can contain any number of instructions.

The program code in a method is always written between **FUNCTION** and **END_FUNCTION**:

```
FUNCTION ClassName::NewMethode

  // <-- Programcode

END_FUNCTION
```

If a method possesses transfer and return parameters, the program is written between **END_VAR** and **END_FUNCTION**:

```
FUNCTION ClassName::NewMethode
VAR_INPUT
  Param1       : DINT;
  Param2       : DINT;
END_VAR
VAR_OUTPUT
  OutParam     : DINT;
END_VAR

  // <-- Programcode

END_FUNCTION
```

3.2.1.14 Commentary

One should not be too generous with commentary. Commentary helps the user to better understand the program code. You not only do others a favor but yourself as well when dealing with your code. What characterizes this variable, which function is behind each method? Spare yourself and others the unnecessary and time-consuming search for the answer.

It is possible to write commentary in the program code to serve as information for the programmer, which is not seen by the compiler. This information must be commented out as follows. There are two possibilities:

```
//
```

All text in a line following these characters is commentary.

```
(* *)
```

All text written between these characters (also over several lines) is irrelevant to the function of the program.

Example:

```
#define MAXTEMP 90 //maximum temperature in the housing
#define MINTEMP 10
(*
Minimum ambient temperature that must be maintained to ensure fault-free operation.
*)
```



Tip:

Select the lines you want to set under comments or remove from comments and press **Ctrl-Shift-C** to set and **Ctrl-Shift-X** to remove.

3.2.1.2 Variables and Data Types

3.2.1.21 Data Types

In the following table, the binary, hexadecimal and decimal numbers from 0 to 255 are listed:

Binary	Hexadecimal	Decimal		Binary	Hexadecimal	Decimal
0000 0000	0	0		0010 0001	21	33
0000 0001	1	1		0010 0010	22	34
0000 0010	2	2		0010 0011	23	35
0000 0011	3	3		0010 0100	24	36
0000 0100	4	4		0010 0101	25	37

0000 0101	5	5		0010 0110	26	38
0000 0110	6	6		0010 0111	27	39
0000 0111	7	7		0010 1000	28	40
0000 1000	8	8		0010 1001	29	41
0000 1001	9	9		0010 1010	2A	42
0000 1010	A	10		0010 1011	2B	43
0000 1011	B	11		0010 1100	2C	44
0000 1100	C	12		0010 1101	2D	45
0000 1101	D	13		0010 1110	2E	46
0000 1110	E	14		0010 1111	2F	47
0000 1111	F	15		0011 0000	30	48
0001 0000	10	16		0011 0001	31	49
0001 0001	11	17		0011 0010	32	50
0001 0010	12	18		0011 0011	33	51
0001 0011	13	19		0011 0100	34	52
0001 0100	14	20		0011 0101	35	53
0001 0101	15	21		0011 0110	36	54
0001 0110	16	22		0011 0111	37	55
0001 0111	17	23		0011 1000	38	56
0001 1000	18	24		0011 1001	39	57
0001 1001	19	25		.	.	.
0001 1010	1A	26		.	.	.
0001 1011	1B	27		.	.	.
0001 1100	1C	28		1111 1011	FB	251
0001 1101	1D	29		1111 1100	FC	252
0001 1110	1E	30		1111 1101	FD	253
0001 1111	1F	31		1111 1110	FE	254
0010 0000	20	32		1111 1111	FF	255

Bit

A binary character or bit (short for binary digit) can only contain two values and is the smallest organizational unit or data. Bit values are mostly represented by the symbols "L" (binary 0) and "H" (binary 1).

Nibble

A nibble represents a half byte; a hex digit is a nibble!

Byte

A byte is a sequence of eight bits, which are processed together in a computer. A byte is also an 8-bit quantity with a defined memory address and is the smallest addressable memory unit.

The description and value range for each variable type can be seen in the following table.

Data type	Lower limit	Upper limit	Description
SINT	-128	127	Short Integer
INT	-32768	32767	Integer
DINT	-2147483648	2147483647	Double Integer
USINT	0	255	Unsigned Short Integer
UINT	0	65535	Unsigned Integer
UDINT	0	4294967295	Unsigned Double Integer
REAL	-3,4 E+38	+3,4 E+38	Real number, floating point number
LREAL	-1.79769313486231 E+308	1.79769313486232 E+308	Long-Real, 64 Bit-floating point value
BOOL	0 (FALSE)	1 (TRUE)	Bool
CHAR	0	255	Character (ASCII character)
BSINT	0	255	8 Bit Short Integer
BINT	0	65535	16-Bit Integer
BDINT	0	4294967295	32 Bit Double Integer
HSINT	0	255	Hex Short Integer
HINT	0	65535	Hex Integer

HDINT	0	4294967295	Hex Double Integer
ARRAY			Data field
STRUCT			Data structure
ENUM			Enumeration type

In the following table, the various data types are explained:

Type	Memory	Description
SINT	1 Byte	The data type, Short Integer, consists of both a positive and negative value range from –128 to +127 and is used with whole number values.
INT	2 Byte	The data type, Integer, consists of both a positive and negative value range from –32768 to +32767 and is used with whole number values.
DINT	4 Byte	The data type, Double Integer, consists of both a positive and negative value range from –2147483648 to +2147483647 and is used with whole number values.
USINT	1 Byte	The data type, Unsigned Short, Integer consists of a positive value range from 0 to 255 and is used with whole number values.
UINT	2 Byte	The data type, Unsigned Integer, consists of a positive value range from 0 to 65535 and is used with whole number values.
UDINT	4 Byte	The data type, Unsigned Double, Integer consists of a positive value range from 0 to 4294967295 and is used with whole number values.
REAL	4 Byte	The data type REAL is a floating-point data type with 1 bit for signs, 8 bits for exponents and 23 bits for number values. Calculations like those with a pocket calculator can be made using this data type. The disadvantage is that calculations are slower with this data type than with others. Numbers with places after the decimal point are also possible.
LREAL	8 Byte	The data type LREAL or Long-REAL, is a floating-point data type with a 1-bit sign, 11-bit exponent and 52-bit number value. With this data type, calculations can be performed as with a pocket calculator. This disadvantage these operations a slower than with other data types. Numbers with decimal values are also possible. The data type LREAL is **not thread-safe!**
BOOL	1 Byte	With this data type, only the condition 1 (TRUE) and 0 (FASLSE) are possible. This data type is used for logic conditions and can be linked to logic operations.
CHAR	1 Byte	This data type can be written to with ASCII characters or numbers between 0 and 255. This data type displays ASCII characters only.
BSINT	1 Byte	This data type can be loaded with a number value between 0 and 255. In addition, each of the 8 bits in this variable can be identified (provided with a name) and can be used as a Bool variable in the program. The variable display always occurs in the binary number system.

BINT	2 Byte	This data type can be loaded with a number value between 0 and 65535. In addition, each of the 16 bits in this variable can be identified (provided with a name) and can be used as a Bool variable in the program. The variable display always occurs in the binary number system.
BDINT	4 Byte	This data type can be loaded with a number value between 0 and 42949967295. In addition, each of the 32bits in this variable can be identified (provided with a name) and can be used as a Bool variable in the program. The variable display always occurs in the binary number system.
HSINT	1 Byte	The data type, Hex Short Integer, consists of a positive value range from 0 to 16#FF (255) and is used with whole number values. The variable type is always displayed in hexadecimal.
HINT	2 Byte	The data type Hex Integer consists of a positive value range from 0 to 16#FFFF (65535) and is used with whole number values. The variable type is always displayed in hexadecimal.
HDINT	4 Byte	The data type Hex DoubleInteger consists of a positive value range from 0 to 16#FFFFFFFF (4294967295)and is used with whole number values. The variable type is always displayed in hexadecimal.
ARRAY	Field dependant	This data type consists of a one or more dimensional fields of a different data types
STRUCT	Structure dependant	This data type enables the creation of a data structure of different data types.
ENUM	Data type dependant	These data types a named allocation of numbers (0 to.).

The data types REAL and LREAL are implemented in LASAL in accordance with the IEEE-754 standard. The tables below illustrates the limits of both data types according to IEEE-754:

Type	Size (1+r+p)	Exponent (r)	Mantissa (p)	Exponenten values (e)	Bias value (B)
single	32 bits	8 bits	23 bits	-126 <= e <= 127	127
double	64 bits	11 bits	52 bits	-1022 <= e <= 1023	1023

Type	?	Decimal places	(according to amount) smallest number (standardized)	(according to amount) smallest number (non-standardized)	Largest number
single	$2^{-(23+1)}$ ≈ 5,960·10−8	7 … 8	2^{-126} ≈ 1·10−38	$2^{-23} \times 2^{-126} = 2^{-149}$ ≈ 1·10−45	$(1-2^{-24}) \times 2^{128}$ ≈ 3·1038
double	$2^{-(52+1)}$ ≈ 1,110·10−16	15 … 16	2^{-1022} ≈ 2·10−308	$2^{-52} \times 2^{-1022} = 2^{-1074}$ ≈ 5·10−324	$(1-2^{-53}) \times 21024$ ≈ 2·10308

3.2.1.22 Alignment for REAL/LREAL Numbers

With REAL and LREAL (long real) numbers, special attention must be paid to alignment. REAL or LREAL values must always be on a memory address that can be divided by 4 in order to be processed correctly. If you now manage a memory area containing REAL or LREAL data, you must ensure that the data pointer is located in a memory area whose address can be divided by 4 before you cast the data. The following figure illustrates the problem:

```
// Example: (Memory start-address: pPointer := 0x09000000)
//|____|____|____|____|____|____|____|____|____|____|____|____|____|____|____|____|
//|SINT|REAL.____.____.____|INT_.____|LREAL____.____.____.____.____.____.____.____|
//|    V                   |         V
//V    0x09000001          V         0x09000007
//0x09000000               0x09000005
//
// pPointer += 7;
```

In the case shown above, the following access to the LREAL value ...

```
MyLREAL := (pPointer)$^LREAL^
```

...would result in the error message UNALIGNMENT ACCESS. (memory address 0x09000007 --> not divisible by 4). A possible remedy is to first copy the data into a local variable:



```
VAR
 locLREAL : LREAL;
END_VAR

if (pPointer MOD 4) <> 0 then
 _memcpy(#locLREAL, pPointer, sizeof(LREAL));
 MyLREAL := locLREAL;
else
 MyLREAL := (pPointer)$^LREAL^;
end_if;
```

This guarantees that there is no UNALIGNMENT ACCESS when accessing the LREAL value.

3.2.1.23 The Difference Between Conversion and Casting

Theoretical

Type conversion and **casting** are different ways **to change the data type of an expression**. An example would be the conversion of an integer value into a floating point value or its textual representation as a string and vice versa. When **converting**, certain characteristics of type hierarchies or data representations can be used. Two **important aspects** of a conversion are **whether it is implicit (automatic) or explicit**, **and whether** the underlying data representation is **converted** from one representation to another, **or** whether a particular representation is simply **reinterpreted** as the representation of another data type. In general, **both primitive and composite data types can be converted**.

An **implicit conversion** can take place **either during compilation or at runtime**. For example, in an expression that mixes integers and floating-point numbers (such as 5 + 0.1), the compiler **automatically** converts the integer representation to floating-point representation so that fractions are not lost. **Explicit conversion** is indicated **either by writing additional code** (e.g. by adding type identifiers or calling built-in routines) **or by coding conversion routines** that the compiler can use if it would otherwise stop in case of a type error.

Conversion and casting are distinctly different concepts.

Conversion refers to **implicitly or explicitly changing a value** from one data type storage format to another, e.g. from a 16-bit integer to a 32-bit integer. **The memory requirement may change as a result of the conversion**, including a **possible loss of accuracy, or even shortening of the value**.

Casting, on the other hand, means an **explicit change in the interpretation of the bit pattern,** which represents a value of a certain type. For example, 32 contiguous bits can be treated as an array of 32 booleans, a 4-byte string, an unsigned 32-bit integer, or a single precision floating-point number according to IEEE. Since **the stored bits are never changed,** the programmer must know basic details, such as the display format, byte order, or alignment requirements, to be able to cast meaningfully.

It can also be simplified in this way:

- When **converting**, the user tells the compiler the following: the **bit sequence** representing the source variable **may not fit into the target data format - the compiler may** take care of it / **adapt it**

- When **casting,** the user informs the compiler that the **bit sequence,** which is located on a certain memory location (or in a variable), **should have a certain form.**

The result is that the compiler is very "economical" (efficient) when casting. It can even happen that the compiler does nothing at all - if the existing data fits into the target type without further ado. There is **only one exception**: the **pointer or a reference** - because they are an **indirect assignment** - may **cut off bits** or result in a **sign extension**. In these cases **Casting "behaves" like a conversion**.

To avoid this, the reference / pointer can be bypassed **by a cast into an intermediate result.**

Example:

```
1.  myInt1 := (myDint1 + myDint2)$INT * myInt2; // Cast
2.  myInt1 := TO_INT(myDint1 + myDint2) * myInt2; // Conversion
```

In the first case (**cast**), we inform the compiler that the intermediate result (myDint1 + myDint2) actually fits into an INT type without any problems. Thus the compiler will **not issue an error message** (DINT is larger than INT), but **simply multiply** this intermediate result **by** `myInt2`.

In the second case (**conversion**), we instruct the compiler **to convert the intermediate result** (myDint1 + myDint2) **of type DINT to INT**. In this case, you may have to reckon with **truncation of bits or sign extension!**

Depending on the method, we will have **quite different results** with the **overflow**!

Practice

In a project, it is possible that global and local variables, clients and servers of various data types are assigned. If the value of a client of type **DINT** (double integer) were assigned to a variable of type **INT** (integer), the compiler would generate an error message in the output window. To avoid this, the client, the client must be changed to the same type as the variable. The compiler will also not accept an assignment between to unlike types with the same data width (e.g. **BINT** to **INT**).

For assigning larger types, the conversion is done by the compiler automatically.

```
a : SINT;
b : DINT;

b := a; // Allocation to a larger data type
```

In the next example, how variable "**b**" of type **DINT** is converted to the same data width as variable "a" of **SINT**.

```
a : SINT;
b : INT;

a := TO_SINT(b); // convert
```

The data element to be converted, the local variable, etc. must always be enclosed in ().

The following instructions are available for conversion of types:

`TO_INT, TO_SINT, TO_DINT, TO_UINT, TO_USINT, TO_UDINT, TO_REAL, TO_LREAL`



Before continuing, a word about the rounding behavior of ST functions. There is a setting in Autoexec.lsl that determines the general rounding behavior in the ST language. The setting is:

```
SET FPUROUNDCONTROL X
```

X can take integer values between 0 and 3. The values have the following meaning:

0 The values are rounded up from 0.5, rounded down below to the next integer.

1 Always round down to the next integer.

2 It is always rounded up to the next integer.

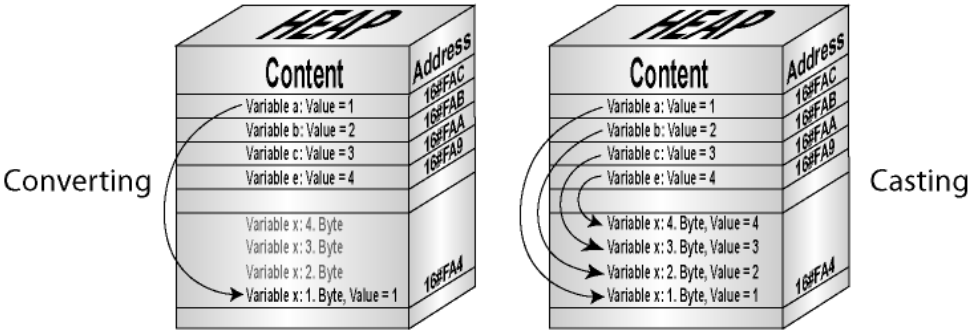
3 The decimal part of the number is scrapped, i.e. the integer part remains.

In the case of negative numbers, the rounding behavior must be taken into account. An example: Value 1, i.e. "always round off" would result in -2.0 for -1.4; but +1.0 for +1.4!

By default - if there is no SET FPUROUNDCONTROL command in the Autoexec.lsl file - 0 is assumed, i.e. it is rounded up from 0.5 and down below. The SET FPUROUNDCONTROL setting applies to all ST functions where rounding may occur! If you want each function to have its own well-defined rounding behavior, you should use the MathBSD library. See the corresponding chapter "The MathBSD Library" in the appendix.

The difference between conversion and Casting

The following graphic shows the difference between **casting** and **conversion**. Variable **a**, of type **USINT**, is assigned to variable **x** of type **UDINT**.



With **converting**, the value of variable **a** is written to the **first** byte of variable **x**.

With **casting**, the value of variable **a** is not only written to **byte 1** of variable **x**, the contents of variables **b**, **c** and **e** are also written to bytes **2**, **3** and **4** of the **x** variable.

Application of Casting

Only pointers or self defined types (types that can be converted) can be casted.



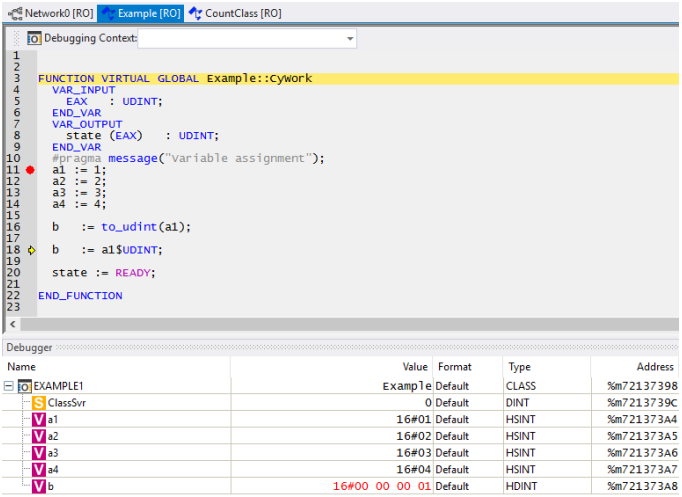
With casting, only variables with the same data width should be used. Otherwise, part of the desired value will be lost.

In the following example, the difference between casting and converting is explained by debugging the executed program lines:

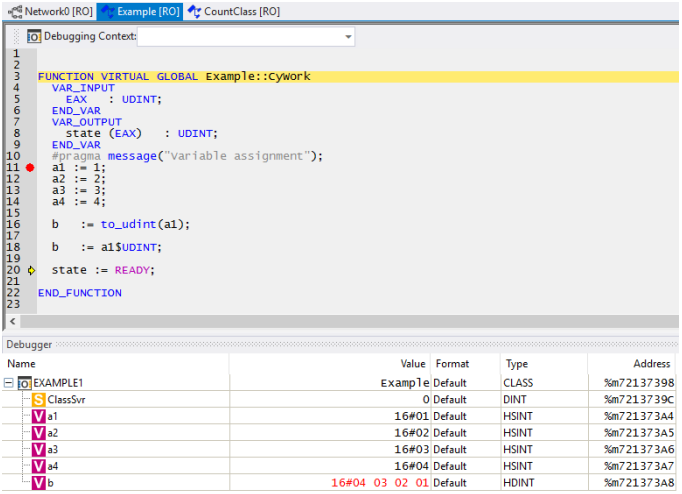
a1, a2, a3, a4 Assigned as a Data element of type **HSINT**

b Assigned as a Data element of type **HDINT**

The graphic below shows the value of the data element (yellow arrow) after the program line **b := TO_UDINT(a1);** was processed in single step.



In the next step of the debugging process, the value of the data element after the program line **b := a1$UDINT;**.



3.2.1.24 Variable Declaration

In each every program, user defined data is required to complete specific tasks. Variables are used to represent values. A variable represents an address in the main memory of the computer. In this address, a specific number of bytes are reserved. What a variable represents can be any number of things: a simple number, a large floating-point number, a single character or a character string.

The variable name, also call an **identifier**, serves to identify the memory address in the program code with a descriptive name. A variable is nothing more than a synonym or placeholder of a specific memory location.

Variables must be declared. Under **variable declaration**, the name of the variable as well as the data type is entered.

3.2.1.25 The Variable Identifier

A variable name can name can be assigned arbitrarily, but special rules apply.

- An identifier can only be comprised of alphanumeric characters, the Underscore and a number. Spaces and other special characters such as #, §, $ for example are not allowed.
- A number at the beginning of an identifier is also not allowed.
- The identifier must be unique; it cannot be the same as a Keyword, method, class or object name.
- If a variable name appears in read after being assigned, it is either a keyword or a name that is already in use.

An addition Tip for name assignments: Select descriptive names so that the code is more readable. Simple x or y identifiers are too vague. Only number variables for loops are given short names.

3.2.1.26 Validity period (Lifespan) of variables

There are 3 validity periods:

Global

When a variable or constant is declared with the subsequent key word external to a function or class definition, the variable is global and its value can be accessed and changed in the entire program.

Example of a declaration:

```
VAR_GLOBAL
  MyGlobalVar :DINT;
END_VAR
```

If this global variable is accessed in another class, it must be declared in this class. This is done as follows:

```
VAR_EXTERNAL
  MyGlobalVar :DINT;
END_VAR
```

Local

If a variable is declared within a function definition, it is local. It is created when entering the function and destroyed when the function is exited. Accessing the variable from outside the function is not possible.

```
VAR
  MylocalVar :INT;
END_VAR
```

Class (member variable)

If the variable is declared within a class definition (through the class tree), it is available in the class and direct global access is not possible.

3.2.1.27 Variable Types

In a program, there are many changeable number values that can contain specific machine or equipment information (position, temperature, etc.) or intermediate results. These number values are called variables. For variables several things must be defined that are relevant to the variable properties.

Variables

Before a variable can be assigned in a system, the user should consider the number range in which the variable moves. Therefore several things must be considered which influence the processor speed and memory requirements of the variable. Basically, all variables are either signed or unsigned. An unsigned variable cannot contain a negative value. In the variable type declaration, a **U** (unsigned) is placed before the type name.

An additional question is the size of the number range assigned to the variable. This decision is whether 1,2 or 4 bytes of memory should be used. Special forms are the number variables **REAL** and **LREAL**. (Floating-point arithmetic variables). **REAL** uses 4 bytes of memory and consists of a sign bit, an 8-bit exponent and a 23-bit number value. **LREAL** uses 8 bytes of memory and consists of a sign bit, an 11-bit exponent and a 52-bit number value. With REAL number variables, almost all number ranges are covered. The disadvantage of this variable type is the relatively long calculation time in comparison to other variable types.It should also be noted that LREAL is **not thread-safe**.

The example below shows the declaration of a variable:

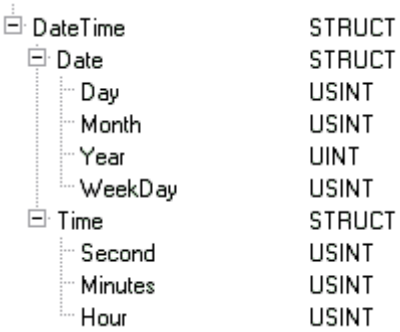
Acceleration	INT
Deceleration	INT
Speed	DINT

The assignment of the variable, **Speed**, appears as follows:

```
Speed := 2435;
```

3.2.1.28 Structures

Variables can also consist of data structures. Data structures are individual variables nested in other variables. For example, the time and date is processed as a structure of a variable.



An assignment of the variable, **Year**, appears as follows:

```
DateTime.Date.Year := 9;
```

To assign a value to the variable, **Minutes**, the following syntax is used:

```
DateTime.Time.Minutes := 34;
```

3.2.1.29 Special Variable forms

ENUM

This variable is an enumeration type and enables the allocation of names to number values from 0 to …

This type of variable is used mostly as a step control for a CASE instruction. Using this type a clear step process can be created.



The following example shows the general structure of a CASE instruction:

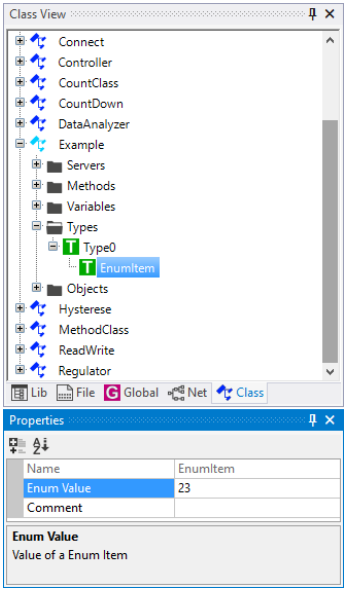
```
case Motion of
  ControllerOn:  //(0)
  MoveLeft:      //(1)
  MoveRight:     //(2)
  StopMove:      //(3)
```

```
  ControllerOff: //(4)
end_case;
```

Using this construction, the step variable is set to MoveRight (step 2):

```
Motion := MoveRight;
```

It is also possible to assign values to ENUM entries in the property window:



CHAR

A variable of data type CHAR is always displayed as ASCII characters. Either the ASCII characters or the corresponding number variables between 0 and 255 can be written to this variable.

ASCII is a standardized code for written and control characters and stands for **A**merican **S**tandard **C**ode of **I**nformation **I**nterchange.

An example of the current ASCII code is shown in the table below.

000	NUL	033	!	066	B	099	c	132	ä	165	Ñ	198	ã	231	þ				
001	Start Of Header (SOH)	034	"	067	C	100	d	133	à	166	ª	199	Ã	232	Þ				
002	Start Of Text (STX)	035	#	068	D	101	e	134	å	167	º	200	╚	233	Ú				
003	End Of Text (ETX)	036	$	069	E	102	f	135	ç	168	¿	201	╔	234	Û				
004	End Of Transmission (EOT)	037	%	070	F	103	g	136	ê	169	®	202	╩	235	Ù				
005	Enquiry	038	&	071	G	104	h	137	ë	170	¬	203	╦	236	ý				
006	Acknowledge (ACK)	039	'	072	H	105	i	138	è	171	½	204	╠	237	Ý				
007	Bell	040	(	073	I	106	j	139	ï	172	¼	205	═	238	¯				
008	Backspace (BS)	041	)	074	J	107	k	140	î	173	¡	206	╬	239	´				
009	Horizontal Tab	042	*	075	K	108	l	141	ì	174	«	207	¤	240	-				
010	Line Feed **(LF)**	043	+	076	L	109	m	142	Ä	175	»	208	ð	241	±				
011	Vertical Tab	044	,	077	M	110	n	143	Å	176	▒	209	Ð	242	_				
012	Form Feed **(FF)**	045	-	078	N	111	o	144	É	177	▓	210	Ê	243	¾				
013	Carriage Return **(CR)**	046	.	079	O	112	p	145	æ	178	▓	211	Ë	244	¶				
014	Shift Out	047	/	080	P	113	q	146	Æ	179	│	212	È	245	§				
015	Shift In	048	0	081	Q	114	r	147	ô	180	┤	213	ı	246	÷				
016	Dataline Escape (DLE)	049	1	082	R	115	s	148	ö	181	Á	214	Í	247	¸				
017	DC 1 (XON)	050	2	083	S	116	t	149	ò	182	Â	215	Î	248	°				
018	DC 2	051	3	084	T	117	u	150	û	183	À	216	Ï	249	¨				
019	DC 3 (XOFF)	052	4	085	U	118	v	151	ù	184	©	217	┘	250	·				
020	DC 4	053	5	086	V	119	w	152	ÿ	185	╣	218	┌	251	¹				
021	Negative Acknowledge (NAK)	054	6	087	W	120	x	153	Ö	186	║	219	█	252	³				
022	Synchronous Idle	055	7	088	X	121	y	154	Ü	187	╗	220	▄	253	²				
023	End Of Transmission Block	056	8	089	Y	122	z	155	ø	188	╝	221	¦	254	■				
024	Cancel	057	9	090	Z	123	{	156	£	189	¢	222	Ì	255					
025	End Of Medium	058	:	091	[	124	|	157	Ø	190	¥	223	▀						
026	Substitude	059	;	092	\	125	}	158	×	191	┐	224	Ó						
027	Escape **(ESC)**	060	<	093	]	126	~	159	ƒ	192	└	225	ß						
028	File Seperator	061	=	094	^	127 (DEL)	⌂	160	á	193	┴	226	Ô						
029	Group Seperator	062	>	095	_	128	Ç	161	í	194	┬	227	Ò						
030	Record Seperator	063	?	096	`	129	ü	162	ó	195	├	228	õ						
031	Unit Seperator	064	@	097	a	130	é	163	ú	196	─	229	Õ						
032	*SPACE* (SP)	065	A	098	b	131	â	164	ñ	197	┼	230	µ						

With an Array of CHAR type Strings can be displayed. The following combinations with the '$' character can be used as control character within a String.

- L or l($L or $l corresponds to LineFeed)
- N or n($N or $n corresponds to NewLine)
- P or p($P or $p corresponds to FormFeed)
- R or r($R or $r corresponds to CarriageReturn)
- T or t($T or $t corresponds to Tabulator)
- $ ($$ corresponds to a Dollar symbol within a String)
- 22 ($22 corresponds to QuotationMark “). After the $ character not only the 22, but any hexadecimal number between 0 and FF (decimal between 0 and 255) can be used to get the corresponding character from the table above. $22 means 34 in decimal notation; in the above table, 034 is the value of the quotation mark.

Bitvariable (BSINT, BINT, BDINT)

If a variable in the system is assigned as type BSINT, BINT or BDINT, the individual bits can be allocated a name and addressed.

Hexvariablen (HSINT, HINT, HDINT)

If a variable in the system is assigned as type HSINT, HINT of HDINT, the contents are shown as a hexadecimal value.

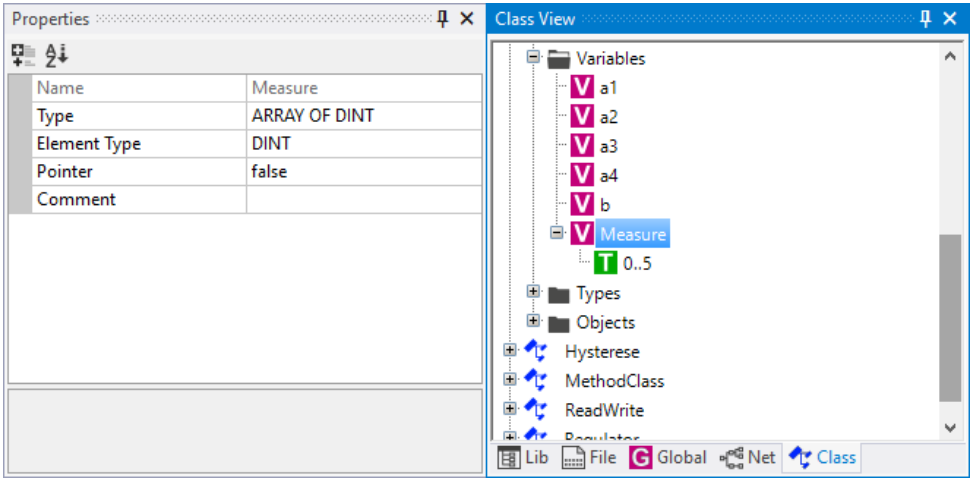
3.2.1.3 Data fields (Arrays)

In programming, data fields are also called arrays. Arrays are a row of variables, arranged sequentially in memory. When initializing an array it is advantageous to start the array index from 0. The field depth can be set according to user requirements.

Arrays can also be nested in several dimensions.

3.2.1.31 Declaration and Initialization of an Array

In the following example, the declaration of the array, **Measure**, is shown with a field depth of 6 entries.



The limits for the maximum values of possible field entries are between –32768 and 32767 since the type INT was assigned in the **Properties window** under **Element Type**.

With the following instruction, the first field, **Measure[0]**, is assigned the negative value –3456 and the second, **Measure[1]**, is loaded with 3456:

```
Measure[0] := -3456;
Measure[1] := 3456;
```

The initialization of the array **Measure** should be done using a For-loop. More information can be found under For-Loop.

3.2.1.32 Accessing Array Elements

Accessing a specific element in order to analyze the contents or assign a new value is done over the index, which is written in brackets at the end of the array name.

For clarification, the six-element array, **Measure**, is reviewed. To assign the number 55, the following instruction must be implemented:

```
Measure[0] := 55;
```

The evaluation of the contents occurs analogous with the assignment of the index. Below, the contents of the first element is written to the variable, **MeasureValue**:

```
MeasureValue := Measure[0];
```

3.2.1.33 Memory Image of an Array

An array describes a specific number of elements from the same type, which are stored sequentially similar to a string. The array, **MyArray**, of type **SINT** is used to show the individual field elements:

```
MyArray[0] := 72;
MyArray[1] := 24;
MyArray[2] := 10;
MyArray[3] := 33;
MyArray[4] := 33;
MyArray[5] := 67;
MyArray[6] := 45;
```

Let's assume that for the first element, **MyArray[0]**, the memory address 4000 was reserved. The image of the memory area under observation could then appear as shown below:

Address	Content
4009	–
4008	–
4007	–
4006	MyArray[6] := 45;
4005	MyArray[5] := 67;
4004	MyArray[4] := 33;
4003	MyArray[3] := 33;
4002	MyArray[2] := 10;
4001	MyArray[1] := 24;
4000	MyArray[0] := 72;
3999	–
3998	–

Reserved memory for the array

3.2.1.34 Multidimensional Array

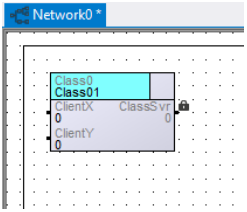
The one-dimensional array discussed up to this point can be thought of as simple row of data on a track.

One-dimensional fields aren't sufficient to represent more complex data structures. Therefore two or more dimensional arrays are frequently used in practical applications.

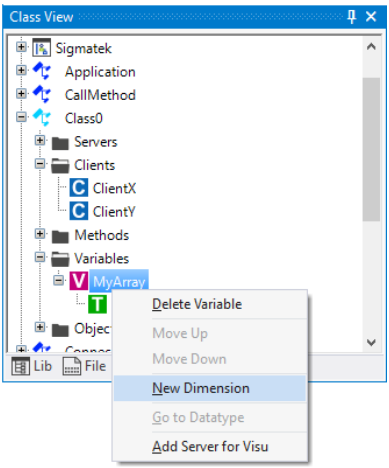
A two dimensional array can be thought of as a matrix or table.

As commonly known, each cell in a table is identifiable by its position in a row and column. To describe the contents of a cell in a table in a specific array element, a two-dimensional offers one dimension to describe the row and another for the column.

The following example clarify the used of multidimensional arrays. Here, a class is created with two clients, **ClientX** and **ClientY**.

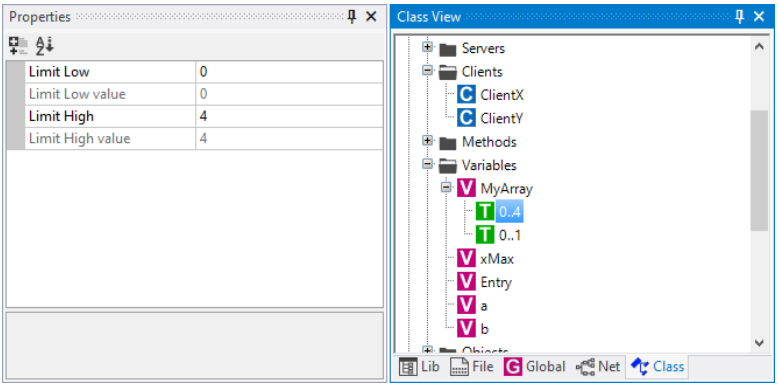


Create a variable (**MyArray**) of type **ARRAY**. For a multidimensional array, click on the variable (**MyArray**) with the right mouse button and select **New Dimension** from the context menu.



In the **Properties window** enter the array size (**0..4**) for the newly created dimension. The dimensions are then displayed in the **Class View window** under **MyArray**.

Next, create the variables **xMax**, **Entry**, **a** and **b**.



Create the **CyWork** Method and insert the following code.

```
FUNCTION VIRTUAL GLOBAL Class0::CyWork
  VAR_INPUT
    EAX  : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)  : UDINT;;
  END_VAR

  ClientX := ClientX.Read();
  ClientY := ClientY.Read();

  if < 4 then

    if ClientX > Xmax then
      xMax            := ClientX;
      myArray[a][b]   := ClientX;
      myArray[a][b+1] := ClientY;
      a     += 1;
      Entry += 1;
    end_if;

  else

    Entry := 0;
    a     := 0;

  end_if;

  state := READY;
```

```
END_FUNCTION
 MyLREAL := (pPointer)$^LREAL^;
end_if;
```

Because the array has a limited number of values, caution must be taken that values are not written beyond the limit. The variable, **Entry** serves to reset the variable after five entries.

To secure the last largest value read client, **ClientX**, the variable **xMax** is used. In this example, **ClientY** is used only to store a value and plays no role when values are being entered into the array.

The variables **a** and **b** define the array entry.

If variable b = 0, the value of ClientX is entered.

If variable b = 1, the value of ClientY is entered.

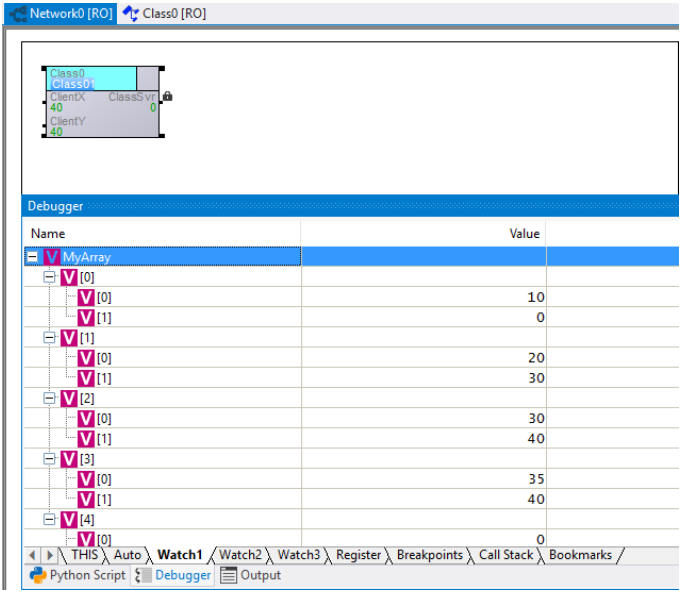
Because the array size column **b** is only **0..1**, it is not incremented per entry.

Column **a** however should be incremented with ever entry.

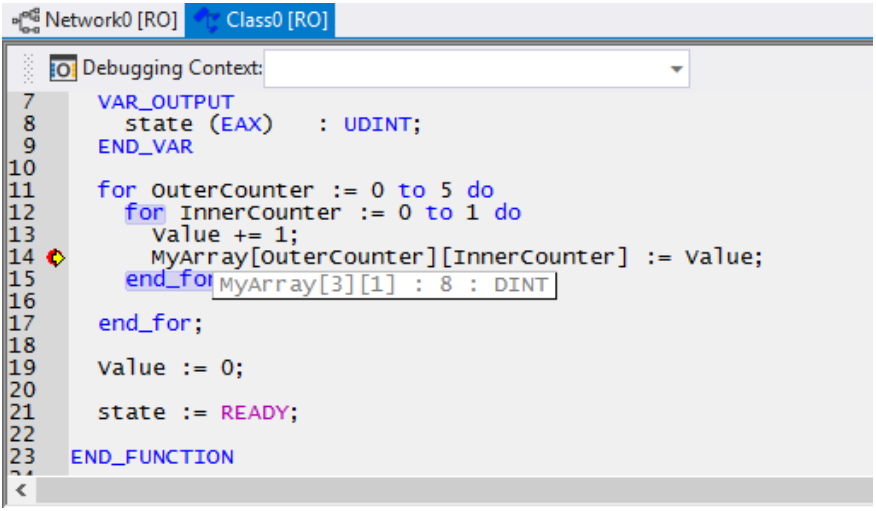
Once all five entries are made, the **Entry** counter and variable a are reset to 0 and the entries start from 0.

Download the project to the control for testing. The program will check whether the client value (**ClientX**) is larger than the previously read value. If this is **true**, the values from **ClientX** and **ClientY** are then stored in the array.

Try to change the value of the clients and observe the array in the Watch window.



As can be seen in the previous screenshot, several options are available for accessing an array element. In addition to the previously discussed form MyArray[0][1], variables can be assigned as the index instead of digits (MyArray[Index1][Index2]). This not only resolved correctly in the debugger, but also in the code editor using Tooltip while debugging.



In the graphic below, a representation of the two dimensional array is shown.

X	Y
10	30
20	40

3.2.1.4 Instructions

3.2.1.41 The If-Instruction

The If-instruction is used when specific sections of a program are executed only when certain conditions are met. The if-instruction serves, depending on a condition, to execute instruction1 or instruction2. If the condition is true, instruction 1 is executed. Otherwise instruction2 after the else-branch is executed if one is assigned, since the else-branch is optional.

```
if (Bedingung) then
    // Instruction1
else
    // Instruction2
end_if;
```

After the word **if**, a condition such as a comparison follows. This comparison results in either TRUE or FALSE. If the comparison is TRUE, the instruction after the word **then** is executed. If it is FALSE, the instruction after **else** is executed (and/or **elsif**) if provided. The IF- instruction must be ended with **end_if**.

Comparisons can also be linked with one another:

„**&**" stands for a logic „**and**",

„**|**" stands for a logic „**or**".



If the variable, **Temp** exceeds the value range of 10° - 90°, a step control should be set to **EMERGENCY**. It the temperature is between 10 and 40 or 60 and 90 then the step control is set to **DANGER**.

If the temperature is between 40 and 60, the setting is **OKAY**.

```
if (Temp < 10) | (Temp > 90 ) then
  ssw := EMERGENCY;
elsif (Temp < 40) | (Temp > 60) then
  ssw := DANGER;
else
  ssw := OKAY;
end_if;
```

3.2.1.42 The Case Instruction

The Case instruction represents a multi selection of values from a variable (comparable to an If-instruction with a large number of elsif branches). After the word **case**, the variable is entered. When none of the steps are **true**, the instructions after the word **else**, if provided, are processed.

The statement **end_case** signals the end of the case instruction.



The principle construction of a Case instruction:

```
CASE ssw OF
     EMERGENCY:
           iStatus :=  2;
           retcode := 1;
     DANGER:
           iStatus :=  3;
           retcode := 1;
     OKAY:
           iStatus := 1;
           retcode := 0;
END_CASE
```

ELSE can also be used in a Case instruction:

```
CASE ssw OF
     EMERGENCY:
           iStatus := 2;
           retcode := 1;
     DANGER:
           iStatus :=  3;
           retcode := 1;
ELSE
     iStatus := 1;
     retcode := 0;
END_CASE
```

If several conditions are listed in sequence and the same code should be executed in all of these cases, it is sufficient to call the code with the last of the conditions.

```
CASE ssw OF
      EMERGENCY,
      DANGER:
           iStatus := 5;
           retcode := 1; // Danger commands
      NORMAL_CASE:
           iStatus := 1;
           retcode := 0; // Commands in normal case
END_CASE
```

In the "EMERGENCY" and "DANGER" cases, the "danger instructions" are called, in "NORMAL CASE", the "normal case instructions" are executed.

3.2.1.5 Program Loops

3.2.1.51 For-Loop

If the number of times an instruction should be repeated is known, the for-loop is used. The instructions are executed for the range specified in the running variable between (including) the start and end values. The increment between executions can be determined by the user with the addition of **by** in the instruction. If **by** is not included, the increment value is 1 by default. The for-instruction is ended with **end_for**.



An array (field index = 5) should be initialized with a specific value.

```
for i := 0 to 4 by 1 do
  Array[Var] := 0;
end_for;
```

3.2.1.52 While-Loop

In this instruction, the run condition is at the beginning of the loop. The run condition, the expression between **while** and **do**, specifies how long the instruction following it is repeated. The value of the expression is tested before the execution of the instruction; therefore it is possible that loop is never processed. The comparison operators are identical to those in the if-instruction. The instruction, **end_while** terminates the loop.



As long as variable X is less the 100, it should be incremented by 10 and the counter is incremented by 1.

```
while (x<100) do
  x +=10;
  counter +=1;
end_while;
```

3.2.1.53 Repeat-Loop

With the repeat-loop, the exit condition is at the end of the loop. With the words **repeat** and **until**, the repeated instructions are combined in a group. This instruction group is repeated until the exit condition, the expression after **until** is met. The repeat-loop must be closed with **end_repeat**.



Repeat-loops are executed at least once, unlike the while-loop.



The x variable should be doubled until it is greater than 500. If it is greater than 500 at the first loop, it will still be doubled once (unlike the while-loop).

```
repeat
  x *=2;
until (x>500) end_repeat;
```

3.2.1.6 Interrupt instructions for Loops

3.2.1.61 Exit

With the help of the Exit instruction, a loop (for, while and repeat) can be exited early.



```
for i := 0 to 100 do
  if s >= 50 then
    exit;
  end_if;
  s := sum(s1 := s; s2 := 1);
end_for;
```

3.2.1.62 Return

With help from the Return instruction, a method can be exited early. Without Return, the method will run until the END_FUNCTION and/or END_AWL instructions are read.



```
FUNCTION ...

  if end = 0 then
    return;
  end_if;
  CallMethode();

END_FUNCTION
```

3.2.1.63 GoTo

The Goto instruction serves to continue at jump destination assigned by a local label. The label must be closed by a colon.



```
if (condition) then
  goto MyLabel;
end_if;
:::
MyLabel:
:::
```



The use of this instruction doesn't necessarily help the programmer. Functions should be programmed without this instruction whenever possible.

3.2.1.7 ST instructions

3.2.1.71 SEG, OFF

With library functions OFF and SEG, the offset or segment part of a pointer is calculated.



```
segment := SEG(ptr); always 0
offset := OFF(ptr); always ptr
```



This function is used for compatibility only! Because of the memory structure (flat memory), these instruction have no meaning in LASAL.

3.2.1.72 MK_FP

Creates a pointer from segment and offset.

```
ptr := MK_FP(seg, off);  always off
```





Because of the memory structure (flat memory), these instruction have no meaning in LASAL.

3.2.1.73 STI

This instruction cancels the CLI command and releases the interrupt.



The user cannot use this instruction!

3.2.1.74 CLI

The instruction hinders interrupts from stopping the program.



The user cannot use this instruction!

3.2.1.75 SAVE_FLG

The CPUs **flag register** is pushed to the stack.

PUSHF pushes **2 Byte** and **PUSHFD** pushes **4 Byte** from the **Flag-Register**.

3.2.1.76 RESTORE_FLG

The flag or Eflag register is Load from the stack.

POPF pops **2 Byte** and **POPFD** pops **4 Byte** in the **Flag-Register**.

3.2.1.77 SAVE_REG

The register is Load to the stack when **PUSHAD** is called.

32 Bit Mode: **EDI, ESI, EBP, ESP, EBX, EDX, ECX** and **EAX**

3.2.1.78 RESTORE_REG

The register is Load from the stack with **POPAD**.

32 Bit Mode: **EDI, ESI, EBP, ESP, EBX, EDX, ECX** and **EAX**

3.2.1.79 TO_BOOL

Converts the specified type to a variable of type **BOOL**.

Syntax: Variable := **TO_BOOL**(Variable);

Example

```
VAR
 a : INT;
 b : BOOL;
END_VAR

a := 1;
b := TO_BOOL(a);  // --> b now contains TRUE
a := 0;
b := TO_BOOL(a);  // --> b now contains FALSE
a := 7;
b := TO_BOOL(a);  // --> b contains TRUE, since anything that is not 0 is interpreted as TRUE
```

For more information about the the difference between Conversion and Casting see chapter The Difference Between Conversion and Casting

3.2.1.710 TO_SINT

Converts the specified type to a variable of type **SINT**.

Syntax: Variable := **TO_SINT**(Variable);

Example

```
VAR
 a : INT;
 c : SINT;
END_VAR

a := 102;
c := TO_SINT(a);  // c contains 102
a := 308;
c := TO_SINT(a);  // c contains 52 (due to overflow, 308 – 256 = 52)
```

For more information about the the difference between Conversion and Casting see chapter The Difference Between Conversion and Casting

3.2.1.711 TO_INT

Converts the specified type to a variable of type **INT**.

Syntax: Variable := **TO_INT**(Variable);

Example

```
VAR
 a : INT;
 d : DINT;
END_VAR

d := 12435;
a := TO_INT(d);  // a contains 12435
d := 32999;
a := TO_INT(d);  // a contains -32537 (due to overflow)
```

For more information about the the difference between Conversion and Casting see chapter The Difference Between Conversion and Casting

3.2.1.712 TO_DINT

Converts the specified type to a variable of type **DINT**.

Syntax: Variable := **TO_DINT**(Variable);

Example

```
VAR
```

```
 a : INT;
 d : DINT;
END_VAR

a := 12345;
d := TO_DINT(a);  // d contains 12345
a := -1520;
d := TO_DINT(a);  // d contains -1520
```

For more information about the the difference between Conversion and Casting see chapter The Difference Between Conversion and Casting

3.2.1.713 TO_USINT

Converts the specified type to a variable of type **USINT**.

Syntax: Variable := **TO_USINT**(Variable);

Example

```
VAR
 a : INT;
 e : USINT;
END_VAR

a := 123;
e := TO_USINT(a);  // e contians 123
a := -123;
e := TO_USINT(a);  // e contians 133 (since no negative values)
a := 12345;
e := TO_USINT(a);  // e contians 57  (overflow, a=48 x 256 + 57)
```

For more information about the the difference between Conversion and Casting see chapter The Difference Between Conversion and Casting

3.2.1.714 TO_UINT

Converts the specified type to a variable of type **UINT**.

Syntax: Variable := **TO_UINT**(Variable);

Example

```
VAR
 a : INT;
 d : DINT;
 g : UINT;
END_VAR

a := 12345;
g := TO_UINT(a);  // g contains 12345
```

```
a := -12;
g := TO_UINT(a);  // g contains 65524 (since no negative numbers)
d := 1234567;
g := TO_UINT(d);  // g contains 54919 (since overflow)
```

For more information about the the difference between Conversion and Casting see chapter The Difference Between Conversion and Casting

3.2.1.715 TO_UDINT

Converts the specified type to a variable of type **UDINT**.

Syntax: Variable := **TO_UDINT**(Variable);

Example

```
VAR
 d : DINT;
 h : UDINT;
END_VAR

d := 2347657;
h := TO_UDINT(d);  // h contains 2347657
d := -8;
h := TO_UDINT(d);  // h contains 4294967288 (since no negative numbers)
```

For more information about the the difference between Conversion and Casting see chapter The Difference Between Conversion and Casting

3.2.1.716 TO_REAL

Converts the specified type to a variable of type **REAL**.

Syntax: Variable := **TO_REAL**(Variable);

Example

```
VAR
 a : INT;
 d : DINT;
 i : REAL;
 j : REAL;
 k : REAL;
END_VAR

a := 3;
d := 1000;
i := TO_REAL(d / a);  // i contains 333.0 (since with integer division, there are no decimal
j := 3;
k := 1000;
i := TO_REAL(k / j);  // i contains 333.333344 (small rounding errors on the end)
```

For more information on the difference between conversion and casting see the chapter The Difference Between Conversion and Casting

3.2.1.717 TO_LREAL

Covert the entered type in a variable type **Real**.

Syntax: Variable := **TO_LREAL**(Variable);

Example

```
VAR
 a : INT;
 d : DINT;
 i : LREAL;
 j : LREAL;
 k : LREAL;
END_VAR

a := 3;
d := 1000;
i := TO_REAL(d / a);  // i contains 333.0 (since with integer division, there are no decimal p
j := 3;
k := 1000;
i := TO_LREAL(k / j);  // i contains 333.33333333333...
```

For more information on the difference between conversion and casting see the chapter The Difference Between Conversion and Casting

3.2.1.8 Operators

For operators order, see Operator sequence.

3.2.1.81 Arithmetic Operators

**	Exponent (result of the input type; **REAL** or **LREAL**) **ATTENTION - does not work with negative numbers!**
*	Multiplication
/	Division
MOD	Modulo (remainder, see example below)
+	Addition
-	Subtraction

Example for MOD

```
a := 15 MOD 4; // a:= 3;
```

3.2.1.82 Bit Operators

-	Negation (2's complement)
NOT, !	Complement (1's complement)
AND	Bit wise AND function
OR	Bit wise OR function
XOR	Bit wise XOR- function
SHL	Bit wise Shift Left operation
SHR	Bit wise Shift Right – operation
SAR	Arithmetic Shift Right – operation
ROL	Bit wise Left rotation

ROR	Bit wise Right rotation

3.2.1.821 NOT

Inverts (negates) all Bits of an Operand.

Truth Table

Operand 1	0	1
Resultant	1	0

Application

- Unconditional Toggle for an output.
- Inverts all bits in a Byte, Word, or DWord.

Changes in the **FL register**: None

```
value := NOT value;
```



3.2.1.822 AND

Performs a bit wise AND operation with two operands.
The result is then stored in the left operand.

Truth Table

Operand1	0	1	0	1
Operand2	0	0	1	1
Resultant	0	0	0	1

Application

Setting an output when all input conditions have been met (=1).
Clearing a bit in a Byte, Word or Double word.



```
A.B b_var ,  2#11110000 // lower 4 bits to 0
```



```
value := value AND 16#FFFFFF00
```

3.2.1.823 OR

Creates a bit wise OR connection between two operands. The result is stored in the left operand.

Truth Table

Operand1	0	1	0	1
Operand2	0	0	1	1
Resultant	0	1	1	1

Application

- Setting an output if one or more input conditions are met (=1).
- Setting bits in a byte or word.



```
O.B % D 0100 , 2#00001111 //Lower 4 bits to 1
```



```
value := value OR 16#0000FFFF
```

3.2.1.824 XOR

Establishes a bit wise **XOR connection** between two operands.

The result is stored in the left operand.

Truth Table

Operand1	0	1	0	1
Operand2	0	0	1	1
Resultant	0	1	1	0

Application

- Set an output if 2 input conditions are unequal.
- Invert individual bits in a byte or word.

3.2.1.825 SHL

The operand is shifted to the left by the specified number of positions. With each shift process, the released bits to the right are filled with 0.

After this instruction has been executed, CF contains the last "shifted" bit.

Before	1	1	0	0	0	0	1	1	
After	1	0	0	0	0	1	1	0	=> CF=1

```
Value := value SHL amount;
```



3.2.1.826 SHR

The operand is shifted to the right by the specified number of positions. With each shift process, the released bits to the left are filled with 0. After this instruction has been executed, CF contains the last "shifted" bit.

Before	1	1	0	0	0	0	1	1	

After	0	1	1	0	0	0	0	1	=> CF=1



```
value := value SHL 2;
```

3.2.1.827 ROL

The operand is shifted by the specified number of positions to the left. With each shift process, the released bits to the right are filled with the bit shifted from the left.

After this instruction has been executed, CF contains the last "shifted" bit.



```
value := value ROL amount;
```



```
ROL.AX  002
// AX is rotated 2 times
ROL CL
// AL rotates CL once to the left
ROL.W <DI>,CL
// The word addressed by DI will rotate
// CL to the left one time
```

3.2.1.828 ROR

The operand is shifted by the specified number of positions to the right. With each shift process, the released bits to the left are filled with the bit shifted from the right.

After this instruction has been executed, CF contains the last "shifted" bit.



```
value := value ROR amount;
```



```
ROR.AX 002
// AX is rotated right 2 times
ROR CL
// AL rotates CL once to the right
ROR.W <DI>,CL
// The word addressed by DI will rotate
// CL to the right one time
```

3.2.1.83 Logic Operators

!!	Logic Inverting operator, Test on <> 0
&	Logic AND operator
|	Logic OR operator



```
IF (MinTemp1<40) & (MinTemp2>30) THEN
    // Temperature normal
END_IF;
```

3.2.1.84 Assignment operators

:=	Simple assignment: x := 12.5;
+=	Abbreviation for: x := x + y;
-=	Abbreviation for: x := x - y;
*=	Abbreviation for: x := x * y;
/=	Abbreviation for: x := x / y;

Example

```
x += y;
```

short for

```
x = x + y;
```

3.2.1.85 Comparison Operators

<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
=	Equal
<>	Not equal

3.2.1.86 Component Selection Operator

.	Direct Component Selection Operator e.g.: control.actvalue
^.	Indirect Component Selection Operator e.g.: control_ptr^.actvalue

The '.' operator is used to address components from data elements or variables. When this operator is used after type or class names, the expression transmits the offset of the specified component to the complete structure. In IL, CLASS.METHOD supplies the offset of a method from the method list of a class. This expression is valid for virtual methods only.

Example

```
L.ESIObjChannel.pCh       // Load pointer to a different channel.
L.EDI(ESI)                // Load pointer to the method table of another channel.
CALL(EDI+OtherClass.MyVirtualMethod)
```

In BitField.Bit of AWL, the bit mask or the bit number (starting with 0) is output according to the command, in ST always the bit mask is output.

Example

```
a := bob.bit3;   // gives 4, Bit mask
L.EAX bob.bit3   // gives 4
BT.D EAX, bob.bit3        // gives 2, Bit number
```

3.2.1.87 Address Operators

#	Address Operator

Example

```
control_ptr := #control
```

3.2.1.88 Indirect Operator

^	Operator used to determine the contents of a pointer

Example

```
value := ptr^;
```

3.2.1.89 Operator sequence

An expression is a sequence of operators, operands and punctuation characters, which help define a calculation. Expressions are defined **recursively**. This means that is no limit to the amount of levels in which they can be nested.

The sequence of operands are listed from highest to lowest priority in the following table:

1.	-	Negation (2's complement)
	!!	Logic Negation (= 0, <> 0)
2.	**	Exponential (only for REAL and LREAL)
	NOT	Complement
3.	*	Multiplication
	/	Division
	MOD	Modulo-Division
4.	+	Addition
	-	Subtraction
5.	SHL	Logic Shift Left Operator

	SHR	Logic Shift Right Operator
	SAR	Logic Arithmetic Shift Right Operator
	ROL	Rotate to the left
	ROR	Rotate to the right
6.	<	Is smaller than
	>	Is larger than
	<=	Is smaller than or equal
	>=	Is larger than or equal
	=	Is equal
	<>	Is unequal
7.	AND	Bitwise AND-Operator
8.	XOR	Bitwise XOR-Operator
9.	OR	Bitwise OR-Operator
10.	&	Logic AND-Operator
11.	|	Logic OR-Operator

Because of the priority, the expression **a AND b < c AND d** reacts differently as first expected. One would expect **a, b** and **c, d** to be logically connected and the resultant compared. According to the table above, the **less than character** has a higher priority than the AND operand and is evaluated first. To process this calculation in the proper sequence, parentheses can be used as follows:

```
(a AND b) < (c AND d)
```

3.2.1.9 Standard functions

ABS	Absolute
SQRT	Square Root
LN	Natural logarithm

LOG	Log10
EXP	Exponential function
SIN	Sine
COS	Cosine
TAN	Tangent
ASIN	Arc Sine
ACOS	Arc Cosine
ATAN	Arc Tangent
SIZEOF	Delivers the size of a data element or data type in bytes

3.2.1.91 ABS

Returns the absolute value of a REAL, LREAL, DINT, SINT or INT number.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.92 SQRT

Calculates the square root of a REAL or LREAL number.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.93 LN

Calculates the natural logarithm of a REAL or LREAL number.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.94 LOG

Calculates the base 10 logarithm of a REAL or LREAL number.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.95 EXP

Calculates the exponential value of a REAL or LREAL number.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.96 SIN

Returns the sine of a Real or LREAL number.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.97 COS

Returns the cosine of a REAL or LREAL number.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.98 TAN

Returns the tangent of a REAL or LREAL number.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.99 ASIN

Calculates the arc sine of a REAL or LREAL number and returns the corresponding value.
The result is the **input variable** type.
All results of **angle functions** are in RADIANT values!

3.2.1.910 ACOS

Calculates the arc cosine of a REAL or LREAL number and returns the corresponding value.

The result is the **input variable** type.

All results of **angle functions** are in RADIANT values!

3.2.1.911 ATAN

Calculates the arc tangent of a REAL or LREAL number and returns the corresponding value.

The result is the **input variable** type

All results of **angle functions** are in RADIANT values!

3.2.1.912 SIZEOF

Delivers the size of a data element or date type in bytes.

3.2.1.10 Examples of Data Types

3.2.1.101 BOOL

Boolean values

(0) FALSE and

(1) TRUE

3.2.1.102 SINT

Short integer

Value range: -128 to +127

3.2.1.103 INT

Integer

Value range: -32768 to +32767

3.2.1.104 DINT

Double integer

Value range: -2147483648 to +2147483647

3.2.1.105 USINT

Unsigned short integer

Value range: 0 to 255

3.2.1.106 UINT

Unsigned integer

Value range: 0 to 65535

3.2.1.107 UDINT

Unsigned double integer

Value range: 0 to 4294967295

3.2.1.108 REAL

Real numbers

Value range: 3.4E-38 to 3.4E+38

Additional information on possible calculation errors when converting real numbers can be found in the chapter Real Numbers – Conversion Errors.

The Loader help functions are available for this data type A detailed description can be found under Loader.

3.2.1.109 LREAL

Long real numbers

Value range: -1.79769313486231 E+308 to 1.79769313486232 E+308

Additional information on possible calculation errors when converting real numbers can be found in the chapter Real.

When using LREAL it should be noted that this number type is **not thread-safe**!

3.2.1.1010 CHAR

Character

Value range: 0 to 255

3.2.1.1011 BSINT

Bit field with 8 entries, shown in the debug list in binary format.

Range: 16#00 to16#FF.



```
VAR_GLOBAL
        bob1 : BSINT;
// Byte access or bit: bob1.5
        bob2 : BSINT;
// Byte access: bob2 := 0; or
        bit1, bit2,bit5…;
// Bit access: bob2.bit5 := bob2.3;
END_VAR
```

3.2.1.1012 BINT

Bit field with 16 entries, shown in the debug list in binary format.

Range: 16#0000 to16#FFFF

3.2.1.1013 BDINT

Bit field with 32 entries, shown in the debug list in binary format.

Range: 16#00000000 to16#FFFFFFFF

3.2.1.1014 HSINT

Hexadecimal values

Value range: 16#00 to16#FF

3.2.1.1015 HINT

Hexadecimal values

Value range: 16#0000 to16#FFFF

3.2.1.1016 HDINT

Hexadecimal values

Value range: 16#00000000 to16#FFFFFFFF

3.2.1.1017 ARRAY

An ARRAY is be used to define an optional amount of same data types.



```
TEXT : ARRAY[0..19] OF CHAR;
// Text field with 20 characters
```



Arrays of type CHAR are displayed in the debugger as ASCII strings.

3.2.1.1018 STRUCT

Data, which has different data types, can be defined using a **STRUCTURE**.

```
DATE : STRUCT
day : USINT;
month : USINT;
year : UINT;
END_STRUCT;
```

Elements for self-defined data types are not limited to the basic data types. In addition to predefined data types, data types can be defined later as well.



```
DATA   : STRUCT
 date   : DATE;
 text   : TEXT;
 amount : UINT;
END_STRUCT;
```

The name and data type are inseparable in further processing.

3.2.1.1019 ENUM

Values in their respective variables can be assigned names. If nothing is specified, symbolic constants are assigned in **ascending order** starting from 0.

To use the same name as the ENUM entry for multiple variables of type ENUM, the TypeScope can be defined:



```
VAR
EnumVar1: (Start, Ready, Step1, End);
MyVar1: (Start, NewLevel, Proof, Return);
END_VAR

state := EnumVar1::Start;
drain := MyVar1::Start;
GlobalState := ::Start; //Global ENUM entry!
```

If a global ENUM entry is available, as well as one in the class, **no** TypeScope is assigned. The ENUM from the class is always used.

ENUM entries can **also be used as an array index**.

This data type is extremely useful when programming a **sequence of conditional steps**. Each section or step can be defined with its own unmistakable name, which hinders the possibility of confusing the process.



```
SSW : (CUT, DRILL, SAND := 4);
```

Preset size: **4 Bytes**.

Casting possible: **E.g.: SSW: (…) $[USINT|UINT|UDINT]**

3.2.1.1020 Type Conversion

Assignments (a := b;) are only permitted for data types that are same or compatible. This means, data type for a must be bigger than or equal to data type for b.



When this condition is not met and a link is needed, the compiler must be informed.

Type Conversion

The **compiler** makes conversions to **larger** data types automatically. When converting a data type smaller or unlike types, an assignment provided by the compiler, must first be set.

TO_INT, TO_SINT, TO_DINT, TO_UINT, TO_USINT, TO_UDINT, TO_REAL, TO_LREAL

```
Syntax : Operand := TO_SINT (Operand);
```



```
a := TO_SINT (b);
```

Any defined data type can be used for **casting**:

```
Syntax : [Operand] := Operand $ Type;
a := b $ SINT;
```



```
//'a' is of type INT;
//'b' is of type SINT;
b := -1;
// Converting
a := TO_INT (b);
// a := -1
// Casting
a := b $ INT;
// a := 'undefined';
```

During the conversion, a byte is read and extended over **2 bytes**. For Casting however, **2 bytes** are read (even though only **1 byte** is initialized) and assigned to **a**.

Casting is often used for **reading pointers**. A **4 byte unsigned value** is read from a table that is addressed with a pointer to **SINT [pts]**.

```
pts  := #table();  // Assignment to the table
e := pts^$UDINT;   // Determine the table length
```

In **STL** an additional type of casting exists with the addition of an **offset** to the address:

```
L.AX (ESI+Struct.Element+0) // Data type of Struct.Element is arbitrary
```

3.2.1.11 Pointer

3.2.1.111 Definition of a pointer

Pointers are variables that point to the contents of a memory address.

They are always comprised of **4-bytes (Linear, Offset)**.

Example of pointer definitions in the type editor:



```
PTS : ^SINT;  // Pointer to SINT
PTI : ^INT;   // Pointer to INT
STR : STRUCT  // Structure containing elements:
        a : INT;
        b : SINT;
        p : PTS;
END_STRUCT;   // Pointer to PTS pointer
PST : ^STR;   // Pointer to STR structure
```



A special pointer is the **this** pointer, which can be used within the method of a class to address the object that processes the code is executed. The data type is, for example, ^MyClass.



A pointer to the base class of a complex class can always be used to address an object of the complex class. Using a pointer, all the **base class features** can be used.

3.2.1.112 Pointer Contents - NIL

The contents of a pointer are addressed with the ^ character

NIL is equal to a Dword zero, which means the pointer is not initialized.

Application in the Functions text:



```
FUNCTION MAIN
VAR
        pti : ^INT;
        point_i : PTI;
// Pointer to INT
        poi_poi : ^PTI;
// Pointer to Pointer to INT
        point_s : PST;
```

```
// Pointer to Structure
        a : INT;
        s : STR;
END_VAR
        a := pti^;
// Contents of Pointer to INT
        a := (pti+2)^;
// Contents from address + 2
        point_s := #S
// (Byte)
        a := point_s^.a;
// Contents from structure entry a
        point_i := poi_poi^;
// Contents of pointer to pointer
        a := poi_poi^^;
// Corresponds with point_i^!
END_FUNCTION
```

3.2.1.113 Variable Addresses

In structured text, the address of a variable is specified with the address operator #

In AWL, the **LEA** instruction is used.

Example of a pointer definition in the Typeditor:



```
PTI : ^INT;
// Pointer to INT
STR : STRUCT
// Structure with the following elements:
        a : INT;
        b : SINT;
        p : PTI;
END_STRUCT;
PST : ^STR;
// Pointer to the STR element
```

Application in the functions text:



```
FUNCTION main
VAR
        point_i : PTI;
// Pointer to INT
        poi_poi : ^PTI;
// Pointer to pointer to INT
        point_s : PST;
// Pointer to structure
        a : INT;
        adr : STR;
        fld : ARRAY[0..4] of STR;
// Field of structures
END_VAR
        point_i := #a;
// Variable point_i points to a
```

```
        point_s := #adr;
// Pointer to structure adr
        poi_poi := #point_i;
// Pointer to Pointer to a field of structures with Pointer
        fld[0].p := #fld[1];
// Determine Pointer
        a := fld[0].p^.b;
// Contents of Pointer (Offset b)
        a := fld[0].p^.a;
// Contents of Pointer (Offset a)
        fld[1].p := #fld[2];
        a := fld[0].p^.p^.a;
// !!!
END_FUNCTION
```

3.2.1.114 Table Contents

The **Table name** is also the pointer to the table and can be assigned to pointers of any type. The first four bytes of every table correspond to the **table length** but are **not included** in the actual length!

A Table that contains the text, **"This is text!"**:



```
FUNCTION TAB TABLE
"This is text!"
END_FUNCTION
lng := #table()^;
pts := #table()+sizeof(UDINT);
// pointer to text
```

Elements supported in tables are numbers (123, 16#EF, 2#0101) and character strings ("hello"). Symbolic names can also be assigned (e.g. # define MAX_COUNT 5). The LASAL compiler always uses the smallest data-width possible (e.g. 1 byte for 16#0012). To change this, cast operators such as (TO_UINT(16#0012) or 16#0034$ UDINT) can be used.

The table and OFF operator are used to determine the length of a part of the table:

```
START:
//supplies no code
"String with unknown length",
END:
OFF(END)-OFF(START),
//LASAL sets the String Length here
```

Line and block comments as well as pre-processor commands can also be used.

3.2.2 Additional Components of Structured Text Programming

3.2.2.1 Working with Pointers

3.2.2.11 What is a Pointer?

As the name implies, **Pointers** point to a specific area memory. Pointers are always 4-byte variables, which point to the memory location and not to the actual value. A pointer can assigned as any data structure. Therefore a pointer can be defined to point to a single integer value or complex structures, arrays, etc. Behind the name of a data element (variable) is an address that defines its location in memory; the pointer points to this address. If the contents of the pointer are accessed, the data element is accessed directly.

- A pointer is a number. It describes the exact address in memory where a certain value is located.
- It always points to the first byte of a type
- A pointer contains the variable location but not its contents.

3.2.2.12 Definition of a Pointer

The size of a memory area is assigned in bytes. The numbers of these areas are described as addresses. To easier address memory space names (variable) are used instead of numbers. Variables also represent addresses in memory. Pointers are thus variables, which point to specific addresses in memory.

Important information on working with Pointers

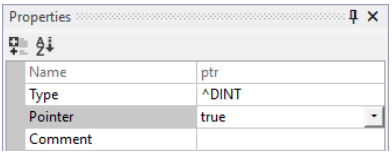


The address is given through the address operator #

The contents of a pointer is addressed through the ^ character (e.g.: ptr^)

Declaration of a pointers e.g.: ^DINT



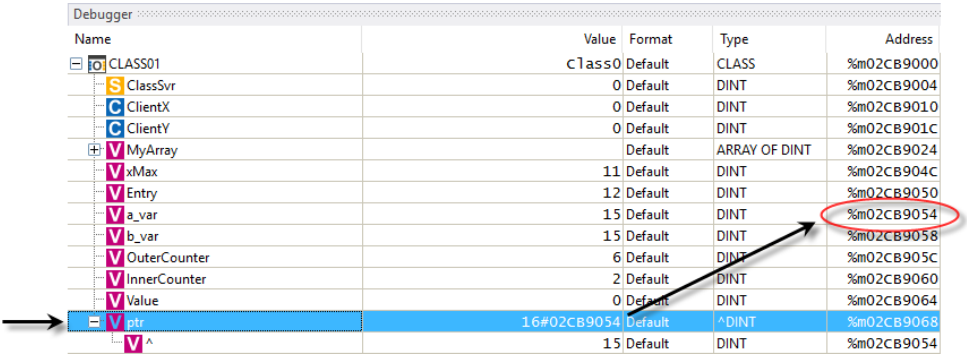


For testing, local variables can be used instead of data elements.

```
a_var := 15; // the data element a_var is assigned the value 15.
ptr := #a_var; // ptr points to the address of a_var.
b_var := ptr^; // the contents of ptr are written to b_var
```

The executed program lines also correspond to the assignment b_var := a_var;. The following illustration shows the variable window (THIS window) in which the assignment of the data elements can be observed. In the Address column, the address of the respective client and data elements are shown.

In the following lines (small arrow) how the pointer ptr points to the address of the data element a_var (long arrow). In the line below, the content of the pointer can be seen (here: 15).



For testing, local variables can be used instead of data elements.

3.2.2.121 THIS-Pointer

The **this**-pointer is a special pointer, which points to the structure of an object. In the variable window (Online connection to the CPU required) under the **THIS** tab, the object structure can be seen.

```
VAR
  pointer :^void; //lokal definierter Pointer
END_VAR
```

```
pointer := this;
```

3.2.2.122 NIL-Pointer

As previously mentioned, pointer requires 4 bytes of memory. The NIL (Null) pointer always points to the address 16#00000000 and serves for the identification of unused resp. invalid pointers.

Example for NIL-Pointer:

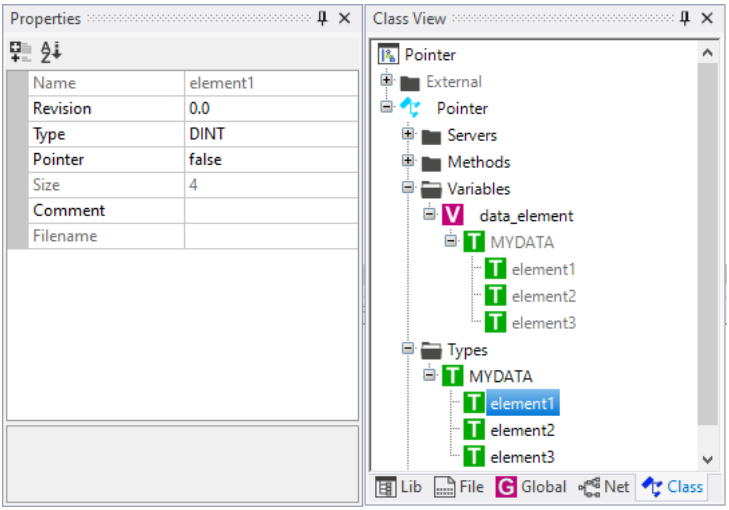
```
a_var := 15;
ptr:= #a_var;

if (ptr <> NIL) then
  b_var := ptr^;
end_if;
```

3.2.2.13 Transferring a Pointer to a Method

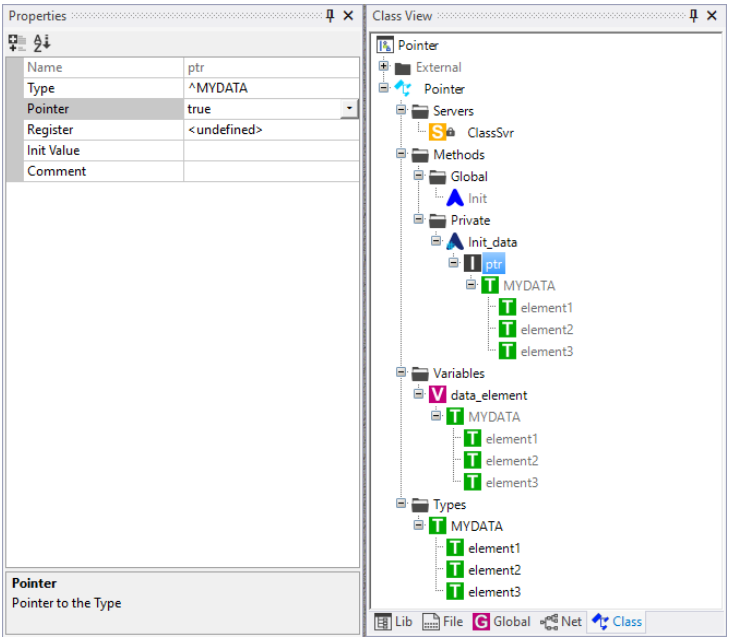
To show how pointers are used, a class is created in which the **Init-Method** and an additional method called **init_data** are assigned. In this example, the address of a variable is transferred to the init_data method and initialized using a pointer.

The graphic below show the type and variable required for this example.



To assign a parameter for the method, click the desired method with the right mouse button. Select **New Input Variable** or **New Output Variable** to create parameter. After

assigning a name, the type can be assigned in the properties window. In this case, it is a pointer of type **MYDATA**.



In this illustration, how the **init_data** is called can be seen. The address of the variable is transferred through the address operator #.

```
FUNCTION VIRTUAL GLOBAL Printer::Init

  if _firstcan = TRUE then
    init_data(#data_element);
  end_if;

END_FUNCTION

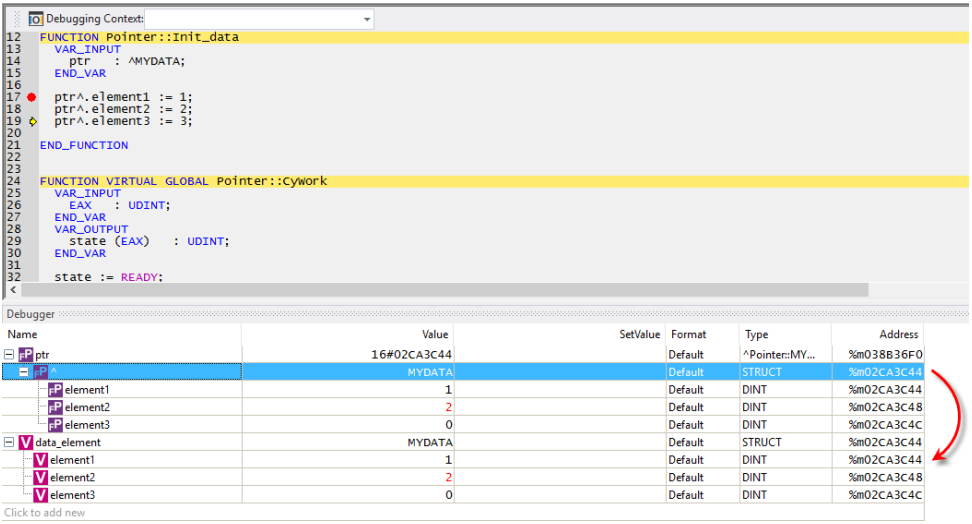
FUNCTION Pointer::init_data
  VAR_INPUT
    ptr  :^MYDATA;
  END_VAR

  ptr^.element1 := 1;
  ptr^.element2 := 2;
  ptr^.element3 := 3;

END_FUNCTION
```

The individual elements are accessed in the **init_data method** through the pointer, ptr^ and initialized with a value.

The following graphic shows the pointer pointing to the address of the variable during debugging (yellow arrow). In the **Debugger window**, the individual values can be observed.



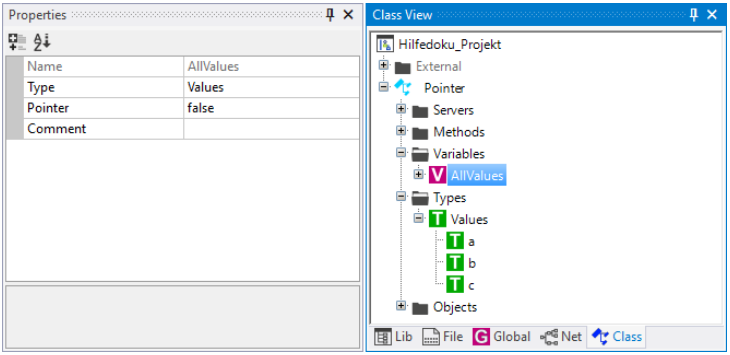


This example should definitely run so that the pointer can be seen in the **Watch** window and the variable in the **THIS window**.

3.2.2.14 Pointers Used as Return Parameters of a Method

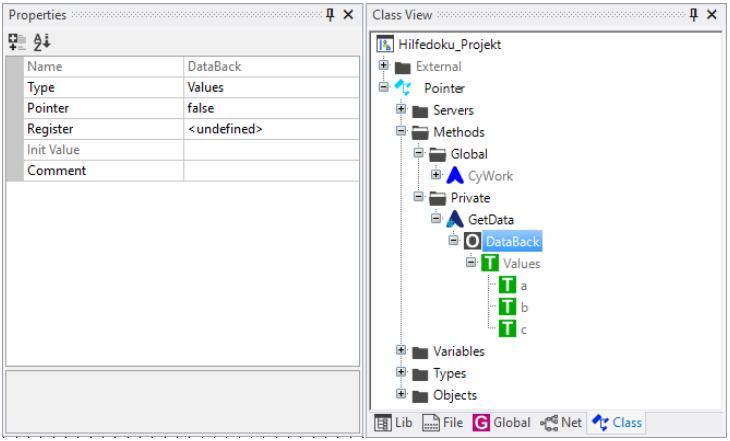
To receive more the one value from a method, pointers are used that point to an array and/or structure. By assigning a **New Output Variable**, several values can be returned.

The graphic below shows the variable and types used for this example.



Two methods are programmed in this example. In practice, the second method could be located in another class. Thus the status of several variables can be transferred with only one pointer.

Assign a method (**GetData**) with an output variable (**DataBack**). The method should read three clients (**Value1, Value2, Value3**) and transfer them to the predefined variable, **AllValues**.



Insert the following code into the program code editor.

```
FUNCTION VIRTUAL GLOBAL Class0::CyWork
  VAR_INPUT
    EAX  : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)  : UDINT;
  END_VAR
  VAR
    GetAdr := ^Values;
  END_VAR

  GetAdr := GetData();

  state := READY;

END_FUNCTION

FUNCTION Class0::GetData
  VAR_OUTPUT
    DataBack  : ^Values;
  END_VAR

  Value1 := Value1.read();
  Value2 := Value2.read();
  Value3 := Value3.read();

  AllValues.a := Value1;
  AllValues.b := Value2;
  AllValues.c := Value3;

  DataBack := #AllValues;

END_FUNCTION
```

In the **CyWork**, the method is called and the **GetAdr pointer** supplies the address. In **GetData**, the clients are read and written to the structure. The address is returned to the CyWork over the output variable. When debugging the project, one can see that 3 values are transferred with one variable.

Debugging Context:

```
1
2
3   FUNCTION VIRTUAL GLOBAL Pointer::Cywork
4     VAR_INPUT
5       EAX    : UDINT;
6     END_VAR
7     VAR_OUTPUT
8       state (EAX)    : UDINT;
9     END_VAR
10    VAR
11      GetAdr   :   ^Values;
12    END_VAR
13
14    GetAdr := GetData();
15
16    state := READY;
17
18  END_FUNCTION
19
20
21  FUNCTION Pointer::GetData
22    VAR_OUTPUT
23      DataBack   : ^Values;
24    END_VAR
25
26    Value1   := Value1.read();
27    Value2   := Value2.read();
28    Value3   := Value3.read();
29
30    AllValues.a := Value1;
31    AllValues.b := Value2;
32    AllValues.c := Value3;
33
34    DataBack := #AllValues;
35
```

Debugger

Name	Value	SetValue	Format	Type	Address
GetAdr	16#02CA3C80		Default	^Pointer::Values	%m038B3774
^	Values		Default	STRUCT	%m02CA3C80
a	3		Default	DINT	%m02CA3C80
b	4		Default	DINT	%m02CA3C84
c	5		Default	DINT	%m02CA3C88
AllValues	Values		Default	STRUCT	%m02CA3C80
a	3		Default	DINT	%m02CA3C80
b	4		Default	DINT	%m02CA3C84
c	5		Default	DINT	%m02CA3C88

Click to add new

3.2.2.15 Summarization of Pointers

Through the use of pointers, data is processed more flexibly since it is easier for them to access structures. In certain cases, pointers can accelerate the program flow. Only pointers or self defined types (types that cannot be converted) must be casted.

3.2.2.2 Working with Tables

3.2.2.21 What are Tables Used for?

Tables are used to define values or text in user programs. Tables can contain text, numbers and/or constants used for interpolation of temperatures or axle positioning. The contents can be read from the table according to specified criteria and subsequently used in various applications.

3.2.2.22 General Information on Tables

Tables can only contain constant data, which cannot be changed during the run time of control program. Also, the values contained in a table are predefined. The name of a table also serves as the **pointer** to the table. The first 4 bytes of every table contain the length of the table but are not included in the actual table length.

3.2.2.23 Syntax for Texts and Numbers in Tables

The following elements can be used in tables:

Numbers (123, 16#EF, 2#0101)

As well as characters („this is text").

The possibility also exists to use pre-processor commands in tables. With help from these commands, constants can be defined and symbolic names entered in a table. The definition of these constants can be made in a header file or in the table directly.

The various methods for writing tables are as follows:

```
FUNCTION TAB Table::Table0
  "TEXT OF MY TABLE"
END_FUNCTION
```

or

```
FUNCTION TAB Table::Table0
#define MAX_COUNT 150
#define MIN_COUNT  50
  123,
  16#EF
```

```
  2#0101
  START_VALUE,
  STOP_VALUE,
END_FUNCTION
```

Writing text in Tables:

MAX_COUNT and **MIN_COUNT**

are defined directly in the Table.

(Pre-Processor instruction)

Numbers (123, 16#EF, 2#0101)

START_VALUE and **STOP_VALUE**

would then be defined in a Header file.

Further information can be found under Pre-Processor Instructions.

The LASAL CLASS 2 compiler always uses the smallest data type possible, which is 1 byte. For example, for 16#00A2, here this data type can also be converted or casted (E.g.: TO_UINT(16#00A2) or 16#00A12$UINT).

With a **LABEL** and the **OFF operator**, the length of a specific part of a table can be sent (see below).

```
FUNCTION TAB Table::Table0
START:
  "STRING WITH UNKNOWN LENGTH",
END:
  OFF(END) - OFF(START),
END_FUNCTION
```

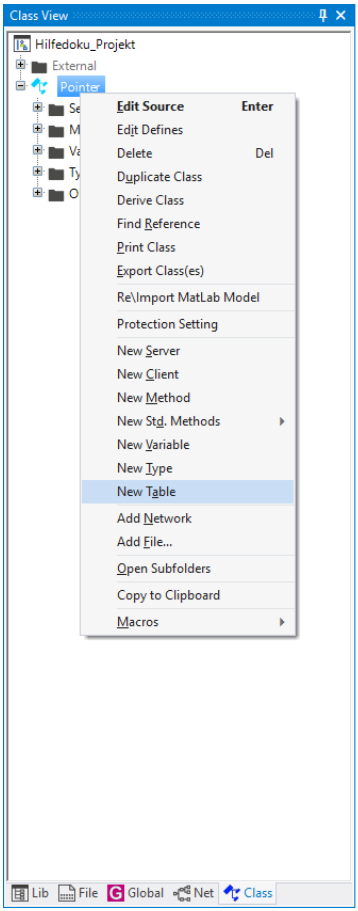
The labels **START** and **END** do not generate code in the table. Here, LASAL adds the string length:

```
FUNCTION TAB Table::Table0
(*
  #define MAX_COUNT  150
  #define MIN_COUNT   50
MAX_COUNT,
MINCOUNT,
*)
  123,
  16#EF,
  2#0101
  // START_VALUE,
  // STOP_VALUE,
END_FUNCTION
```

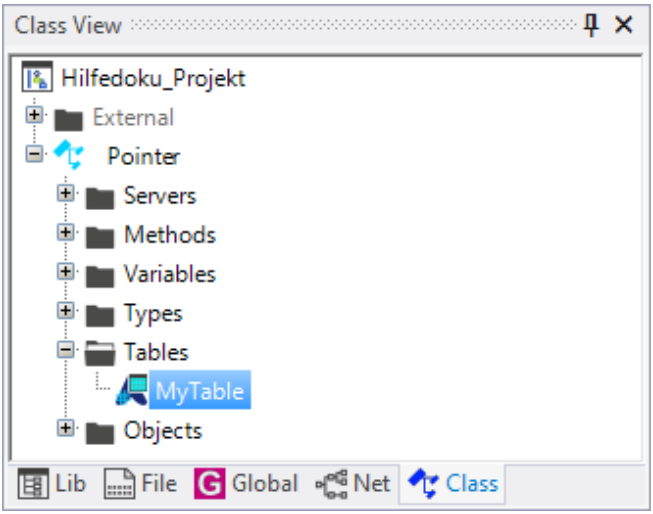
Lines and block commentary can also be used in tables.

3.2.2.24 Table Content with Text

The following example shows how the data length of a table is determined and addition a pointer accesses the first character of a text string. To implement this example, a class is created with the methods shown below:



With the assignment of **MyTable** as table, caution must be taken to select **New Table** and not **New Method**.

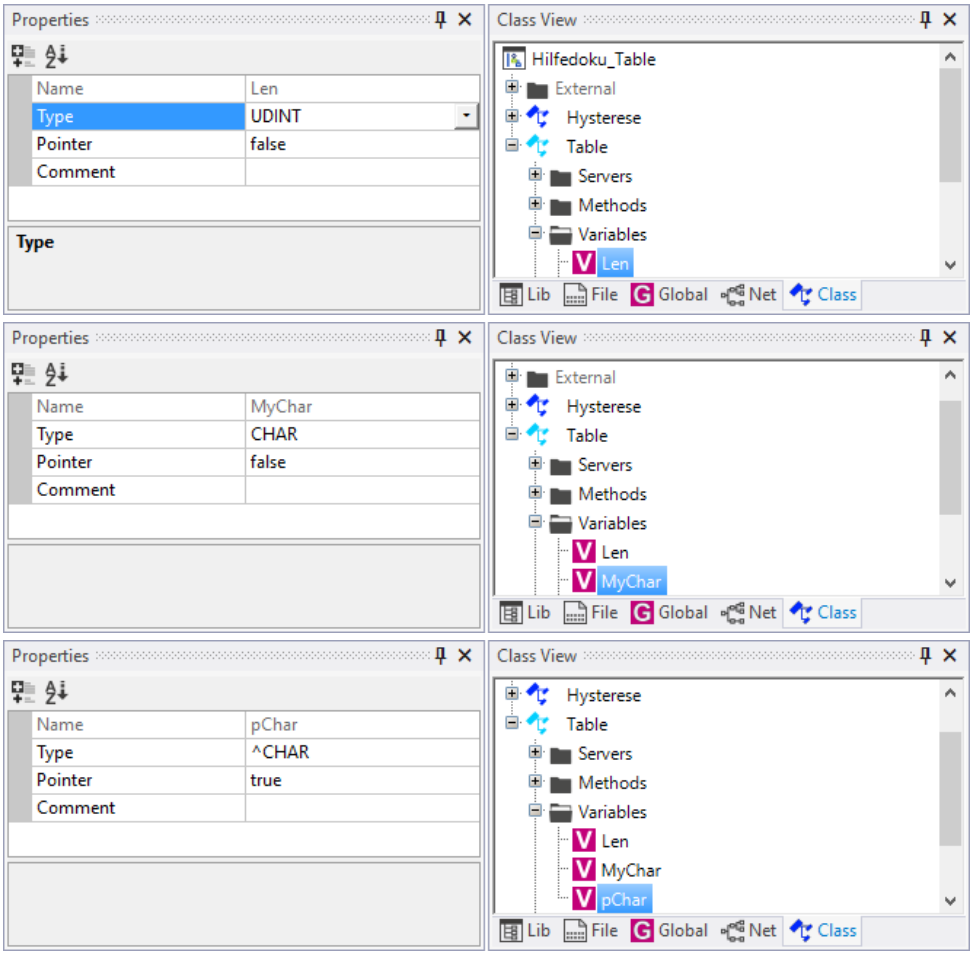






This symbol identifies the method as a table.

Create the required variables as shown in the graphic below.



```
FUNCTION VIRTUAL GLOBAL Table::CyWork
  VAR_INPUT
    EAX  : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)  : UDINT;
  END_VAR

  Len := #MyTable()^;
  pChar := #MyTable() + sizeof(UDINT);
  MyChar := pChar^;

  state := READY;

END_FUNCTION

FUNCTION TAB Table::MyTable

  "Text of MyTable"

END_FUNCTION
```

Insert the above program code in the program.



Important tip!

After generation of the program code with "Edit Code" in the properties window, **TAB** will automatically appear after FUNCTION.

```
FUNCTION TAB Table::MyTable
```

Explanation of the Program Code

```
Len := #MyTable()^;  // Len must be a 4-Byte variable.
```

This line defines the data length of the table. As previously mentioned, the table name is also the pointer to the table. The address of the table is assigned through the pointer (table name) and the contents are written to the Len variable through the ^ character.

```
pChar := #MyTable() + SIZEOF(UDINT);
```

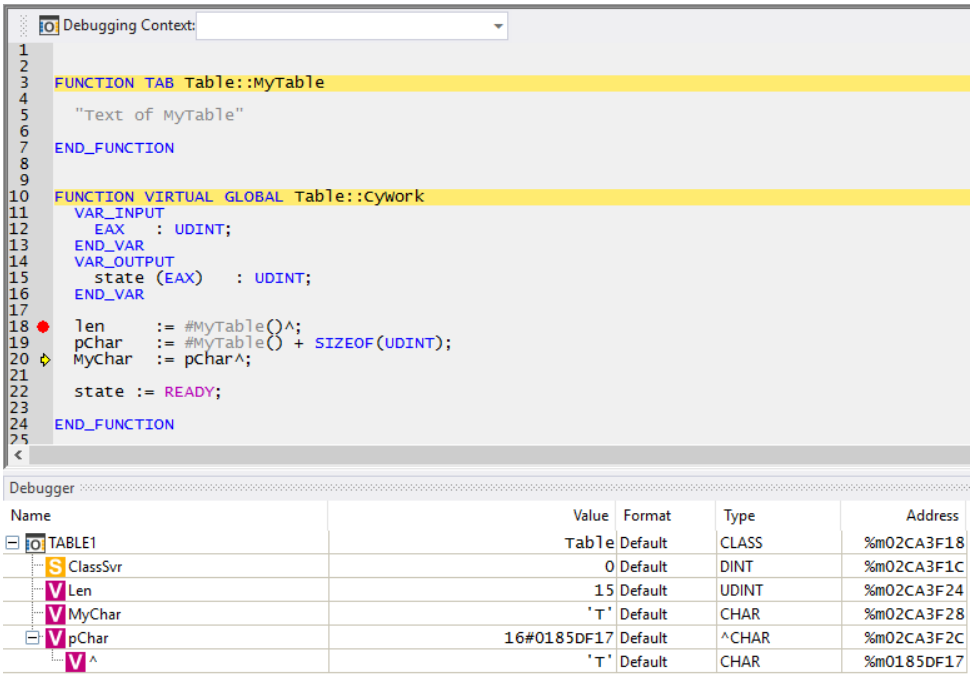
The pointer is positioned on the first letter in the table.

+ SIZEOF(UDINT) is the first 4 bytes of the table and is added for the table length. The pointer is then positioned on the first letter **"T"**.

```
MyChar := pChar^;
```

Through the ^ character, the contents of the pointer is written to **MyChar**.

This example should definitely be debugged to see how the corresponding variables are written to (following graphic). Because the text in the table has 15 characters (with spaces), after debugging the value 15 will be written to **Len**.



To access the second character in the text, the second line must appear as follows:

`(pChar := #MyTable() + SIZEOF(UDINT) + 1;).`

Since the LASAL compiler always uses the data type, 1 byte, for tables, one must be added.

3.2.2.25 Table Content with Numbers (Constants)

In this section, various values are read from a table. The following example should familiarize the user with the construction of such tables and how the individual values are accessed.

To illustrate this example, a class is created with the methods shown below.



To implement this task, the variable, NextValue of type **USINT** and the ActualVal variable of type **UDINT** are assigned to the class TableValue. Sever0 is renamed **Result**.

```
FUNCTION VIRTUAL GLOBAL TableValue::CyWork
  VAR_INPUT
    EAX  : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)  : UDINT;
  END_VAR

  NextValue += 1;
  if NextValue > 17 then
    NetValue := 0;
  end_if;

  ActualVal := ((#MyTable() + SIZEOF(UDINT) + NextValue)^);

  Result_DINT := TO_USINT(ActualVal);

  state := READY;

END_FUNCTION

FUNCTION TAB TableValue::MyTable
0      $USINT;
1      $USINT;
3      $USINT;
7      $USINT;
15     $USINT;
31     $USINT;
63     $USINT;
127    $USINT;
255    $USINT;
127    $USINT;
63     $USINT;
31     $USINT;
15     $USINT;
7      $USINT;
3      $USINT;
1      $USINT;
END_FUNCTION
```

Explanation of the Program Code

The following program line is necessary to store the individual values in the table.

```
ActualVal := ((#MyTable() + SIZEOF(UDINT) + NextValue)^);
```

The actual value is then written to the **Result server**.

```
Result := TO_USINT(ActualVal);
```

```
#MyTable()
```

This line supplies the address of the pointer to the table.

```
#MyTable()+SIZEOF(UDINT)
```

To access the first entry of the table, 4 bytes are added.

```
(#MyTable()+SIZEOF(UDINT)+NextValue)
```

Since the NextValue is incremented by one with each scan, the offset for each value is obtained.

```
((#MyTable()+SIZEOF(UDINT)+NextValue)^)
```

Transfers the contents of each offset from the table.

```
Result := TO_USINT(ActualVal);
```

To write result in the Result server (**DINT**), the whole thing must be converted to DINT.

3.3 Instruction List (IL) in LASAL CLASS 2

LASAL CLASS 2 offers the possibility to create programs or functions using **IL**. The programming language, Instruction List (germ. **A**n**W**eisungs**L**iste), is a type of machine language.

In the following **only the INTEL instruction sets** are explained.

Because the original instruction syntax in INTEL processor is not ideal for a PLC, minor modifications were made to make programming easier. The logic instructions that occur in a program were shortened to letters.

There are no Load or Save instructions in the original instruction set. Instead the **MOV** instruction is used and destination is written on to the left of the data source.

The IEC 61131-3 standard uses the instruction **L** (load) for all data loaded to the processor (except data loaded directly to a memory area) and **S** (save) for all data saved from the processor.



Norm IEC 61131-3		INTEL	
L	Number	MOV	AL, Number
S	Result	MOV	Result, AL

Byte and/or Word operations that don't use a processor register need an instruction that identifies it as an 8-bit or 16-bit operation. For this reason, a **BYTE PTR** for 8-bits or **WORD PTR** for 16-bits is included before the instruction in the original syntax.

SIGMATEK uses the following abbreviations:

B => Byte = 8-Bit

W => Word = 16-Bit

D => Doubleword = 32-Bit

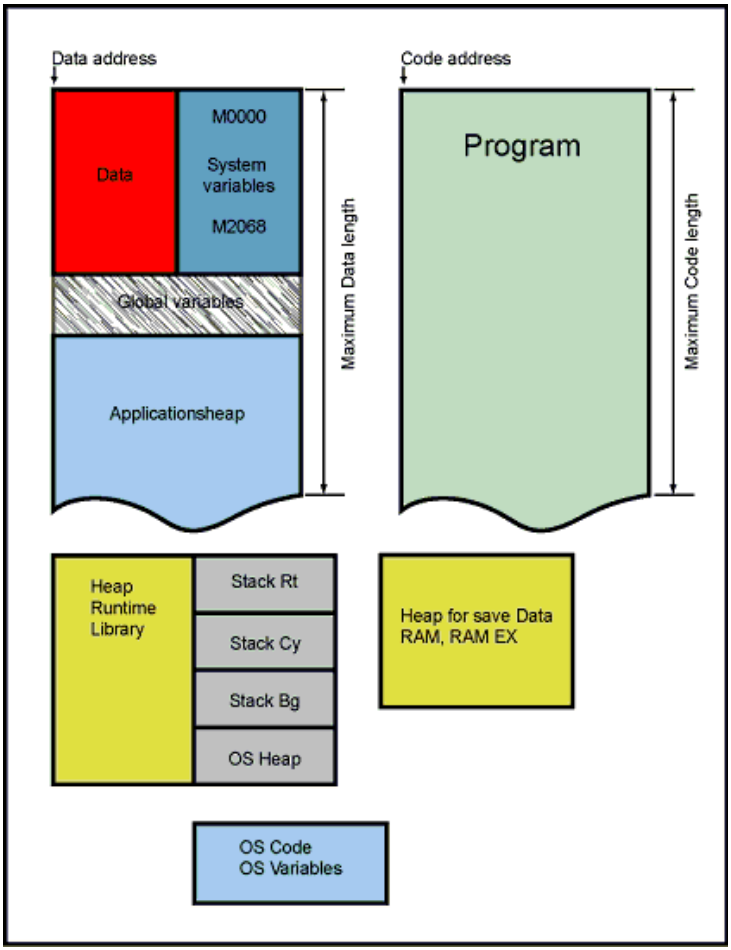


Norm IEC 61131-3	INTEL
CMP.**B** Hand, 001	CMP BYTE PTR Hand, 1

3.3.1 Principle Construction of CPU Memory

In the following diagram, the schematic construction of a LASAL CPU is shown. The components with which the memory is constructed can be seen here:

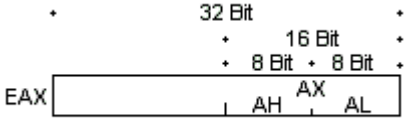
3.3.1.1 Schematic Memory Distribution of a LASAL CPU



3.3.2 The Most Important CPU Register

For Intermediate storage of data during the program runtime, memory units are available in the processor. In these so-called registers, individual data words can be stored. Since these registers are located in the CPU, intermediate results can be stored and recalled faster.

In the diagram below, the 32-bit expanded AX register is shown.



A listing of the available registers can be found in the Register chapter.

3.3.3 Instruction Groups for Programming IL

The Instructions for IL Programming are divided into the following groups:

Data Transfer Instructions

Logic Instructions

Comparison Instructions

Shift Instructions

Jump Instructions

Arithmetic Instructions

Status Register Manipulation

Special Instructions

3.3.4 IL Instructions

3.3.4.1 Introduction

The description of the instruction explains the architecture of the CPU, the types of addressing and the corresponding instructions.



It is recommend that individual instructions be tested in smaller example programs before being used in a larger, more important program. This avoids the frustration of searching for errors in a real program, which could have been prevented through testing.

The **DIAS CPUs** have an **INTEL 80386** processor and no bit processor. The **IPC C300** has an **INTEL CELERON** processor and no bit processor.

Since the original instruction syntax is not ideal for use in a PLC, a few small modifications have been made to make your programming easier. These changes are described in the following section.

Instruction List

The logic instructions that occur predominantly in programming have been shortened to a single letter. In the original instruction syntax, there are no **load** or **save instructions**. Instead, **MOV (Move)** is used. We use the instruction **L (Load)**:

```
L.dst   scr
L       scr
```

and **S (Store)** is used:

```
S.scr   dst
S       dst
```

dst = destination

scr = source

```
INTEL    : MOV AL,PANIC
SIGMATEK: L PANIC
INTEL    : MOV MOTOR,AL
SIGMATEK: S MOTOR
```

For byte, word and double word operations without the use of a processor register, the width must be designated as **8, 16** or **32-bits**. In the original syntax, the addition of **"BYTE PTR"** for **8-Bit**, **"WORD PTR"** for **16-Bit** and **„DWORD PTR"** for **32-Bit** are placed before the address.

```
INTEL    : CMP BYTE PTR HAND,1
SIGMATEK : CMP.B HAND , 001
```

In addition, the display type identification is set **before** the constant (with Intel it's set **after** the constant). To clearly separate the **instructions and address fields**, the register name entry is set in the **instruction field**.

```
INTEL    : ADD AX,D 1000
SIGMATEK : ADD.AX % D 1000
```

The **iAPX 86/88,186/188 User's Manual, Programmer's Reference** contains more useful information regarding the manufacturing of **INTEL components**.

3.3.4.2 STL instructions overview

Instruction	Designation
AAA	ASCII comparison for addition
AAD	ASCII comparison for Division
AAM	ASCII comparison for Multiplication
AAS	ASCII comparison for Subtraction
ADC	Add with carry
ADD	Addition
AND	Logic AND
CALL	Call of a function
CBW	Convert Byte to Word
CLC	Clear Carry-Flag
CLAD	Clear Direction-Flag
CLI	Clear Interrupt-Flag
CMC	Amend Carry-Flag
CMP	Compare destination with source
CMPS	Compare string
CWD	Convert Word to DWord
DAA	Decimal comparison for Addition
DAS	Decimal comparison for Subtraction
DEC	Decrement by 1
DIV	Unsigned division

ENTER	Process entry
ESC	Escape
HLT	Halt
IDIV	Integer Division
IMUL	Integer Multiplication
IN	Input Byte or Word
INC	incremented at 1
INT	Interrupt
INTR+	Interrupt, when INTER and IF = 1
INTO	Interrupt, when Overflow
IRET	Interrupt answer
JA/JNBE	Jump if upper half / Jump if lower half
JAE/JNB	Jump if upper half or equal / Jump if not lower half
JB/JNAE	Jump if lower half / Jump if not upper half
JBE/JNA	Jump if lower half or equal /Jump, if not upper half
JC	Jump if carry
JECXZ	Jump if ECX Null
JE/JZ	Jump if equal/Jump, if Null
JG/JNLE	Jump if greater than / Jump, if not less
JGE/JNL	Jump if greater than or equal/Jump, if not less
JL/JNGE	Jump if less than/ Jump, if not greater
JLE/JNG	Jump if less than or equal/Jump, if not greater
JMP	Jump
JNC	Jump if not carried
JNE/JNZ	Jump if not equal/Jump, if not Null
JNO	Jump if no Overflow
JNP/JPO	Jump if No even parity/Jump, if odd parity
JNS	Jump if not signed
JO	Jump if overflow

JP/JPE	Jump if parity/Jump, if even parity
JS	Jump if character
LAHF	Load AH von Flags
LDS	Load Pointer with DS
LEA	Load effective Address
LES	Lord Pointer with ES
LOCK	Lock Bus
LODS	Load String
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/Loop, if Null
LOOPNE/LOOPNZ	Loop, if not equal/Loop, if not Null
NMI+	Interrupt, if NMI = 1
MOV	Move
MOVS	Move String
MOVSB/MOVSW	Move String (Byte/Word)
MUL	Multiplication, unsigned
NEG	Negate
NOP	No operation
NOT	Logic NOT
OR	Logic inclusive OR
OUT	Byte or Word output
POP	Load register contents of data from the Stack
POPF	Retrieves all registers
PUSH	Shift Word to the Stack
PUSHF	Shift Flags to the Stack
RCL	Rotate left through carry
RCR	Rotate right through carry

REP	Repeat String Operation
REPE/REPZ	Repeat String Operation although equal/ Null
REPNE/REPNZ	Repeat String Operation although not equal/ Null
RET	Return from process
ROL	Rotate left
ROR	Rotate right
SAHF	Store AH in Flags
SAL/SHL	Arithmetic / logic link shift
SAR	Arithmetic shift right
SBB	Subtract with carry
SCAS	Scan String
SEGMENT+	Write to special segment
SHR	Shift Logic right
SINGLE STEP++	Interrupt, if TF = 1
STC	Set Carry-Flag
STD	Set Direction-Flag
STI	Set Interrupt
STOS	Save Byte or Word String
SUB	Subtraction
TEST	Tests or interrupt free Logic AND
WAIT	Waits during TEST-Pin
XCHG	Exchange
XLAT	Translate
XOR	Logic exclusive OR

3.3.4.3 Processor

Address range

32 bit linear addresses

Protected memory

Addressing Types

To access a memory location, the memory address must be defined in the program.

Defining the address can be done in different ways:

Variable names

```
L Variable1
// The contents of variable 1 are loaded to AL.
```

or

```
L.B Variable1,20
// Data byte Variable1 is loaded with the value 20.
```

This definition of the address is easy to understand and is a good solution in may cases.

If, for example, **10 variables in continuous memory** must be loaded with the number 20, the instruction shown below must be programmed **10** times.

```
L.W Variable1,  00020
L.W Variable2,  00020
L.W Variable10, 00020
```

This technique is not comfortable.

In a situation such as this, it is better to program a memory location that stores the value **20**. The memory area is defined as a variable and uses the same instruction **10 times**; each time with a higher address. This is called **indexed addressing**.

Effective Address Loading

```
LEA.EBX, #Variable1     //Word with the Address of Variable1 loaded in the EBX register
```

The registers below may be used as index registers:

EBX, ESI, EDI



Exercise caution with applications with the **ESI register**!

Address defined as sum of 2 index registers

```
S.AX (EBX+ESI) // Contents of register AX written to word with address defined
               // by the sum of registers EBX+ESI.
```

Address defined with index register + constant offset

```
S.AX (ESI+360) // Contents of register AX written to word with address defined by contents
               // of ESI register + the constant 360.
```

Every number between 16#80000000 and 7FFFFFFF can be used as an offset. Register combinations are also allowed.

```
S.AX (EBX+ESI+360)
```

Value defined as constant

```
L.AX - 20000 // AX register is Loaded with number - 20000.
```

A constant is defined, depending on the application, using one of the following methods:

- **Binary:** 2#01010101

For (8, 6, 32) - digit 1/0 sequence.

- **Hex:** 16#0AFD

For (2, 4, 8) digit hexadecimal number.

- **ASCII:** Si

For (1, 2) digit sequence of ASCII characters.

- **Decimal:** -123

For numbers between 0 and 255 or -128 and +127(or 0 to 65535 or -32768 to +32767)

16 bit values (words) (or 0 to 4294967295 or -2147483648 ro +2147483647 32-Bit values (DWords)

Defining the address of a memory location:

```
L.ESI #Variable1
// SI-Reg loaded with address
L.ESI #Variable1
// of location Variable1;
LEA.ESI Variable1
// e.g. character "#" next to the register means that it points to defined memory location
```



The register points to the variable; it is **NOT** loaded with the **CONTENTS** of the memory location!

If the address operator (#) is forgotten, the **contents** of the memory location are completely different but can appear as the normal code.

This error type is frequently the cause of hard to find program errors since the processor calculates strangely, randomly overwrites memory and possibly crashes.

Example:



```
L.DX BX          // Load register DX with the contents of BX.
L.DX (EBX+0)     // Load DX with the contents of the memory location of t
L.SI 100         // Load SI with the number 100.
L.ESI #control   // Load ESI with the address of variable 'control'.
L.ESI cnt        // Load ESI with the contents of the variable 'cnt'.
L.ESI #TABLE()   // Makes ESI point to the start of the table "TABLE".
```

3.3.4.4 Register

IPC and **DIAS** both have 8 free 32 bit registers.

Most **STL instructions** can be used with **any** register. For some instructions, the processor expects the operands to be located in **defined** registers.

The first **4 registers** can be used as **2 separate 8 bit registers (xL /xH)**, where **L** is defined as the **lower 8 bits** and **H** as the **higher 8 bits**. To address the **lower word** of the register, exclude proceeding **E** (e.g. **AX** instead of **EAX**)

Register name	Special functions
EAX, AX (AH,AL)	Standard calculation register
EBX, BX (BH,BL)	Universal register; can also be used as index register.
ECX, CX (CH,CL)	Used as a counting register for all instructions with automatic repeat functions (shifting, blockwise data transfer, loops). Cannot be used as index.
EDX, DX (DH,DL)	Must contain the port address for all accesses to the interrupt controller, system timer or serial interface. Cannot be used as index
ESI	Index register: Sets up the pointer to the data source (source index) for block transfer instructions.
EDI	Index register: Sets up the pointer to the data source (source index) for block transfer instructions.
EBP	Index register for stack manipulation, addressing of local variables and function parameters.
ESP	Stack Pointer: Points to lowest memory location on the stack. Manages the subroutine processing.

3.3.4.41 Special Registers

DS	Data segment: Base address for the data area.
CS	Code segment: Base address for the code area (Program)
ES	Extra segment: Base address for the extra segment
SS	Processor stack: Base address for the processor stack
EIP	Instruction Pointer: Program counter. Points to the byte read from the application program. Moves continuously in the direction of program end as long as it is not re-routed by a jump instruction.
FS	Additional register
GS	Additional register
FL	Processor Status Register: This register contains bits that indicate the status of the last operation, which creates the conditions for t conditional jumps in the program; Other bits in this word are used for influencing internal processor functions. Some bits are reserved for future expansion and are not mentioned here.

3.3.4.42 Processor Status Register

Bit No:	Name	Function / Description
0	CF	This flag is set when the result of the last operation has exceeded the 8 (16) bit range and is required for addition and subtraction of more bytes (words) as a carry. After shift instructions, the last bit shifted from a byte (word) is put in CF (Carry Flag).
2	PF	This flag is set when the lower 8 bits of the last operation contain an even number of 1 bit. (Parity Flag)
4	AF	This flag is set when the last operation has caused a bit carry over between bits 3 and 4 (7 and 8). This is used for decimal arithmetic (Auxiliary Carry).
6	ZF	This flag is set when the result of the last operation = 0. (Zero Flag)
7	SF	This flag is set when the highest bit of the result from the last operation = 1 (this bit is actually the sign for negative numbers in 2's complement; Sign Flag).
8	TF	When this bit is 1, the processor creates a single step interrupt after every instruction. For system test only! (Trap Flag)
9	IF	If this bit is 0, all mask interrupts are cleared. If this bit is set to 0 in the application program for more than 0.9ms, online operation ceases to function. If it remains set to 0 for more than 30ms, the internal watchdog logic is activated! (Interrupt Enable Flag)

10	**DF**	If this bit is 1, the block transfer instruction copies downwards. This means that the register ESI and/or EDI are decremented after every byte (word) transferred. Normally = 0 (Direction Flag)
11	**OF**	This is set when the result of an arithmetic operation doesn't fit the defined data length (Overflow Flag).
12,13	**IO PL**	(I/O Privilege Level)
14	**NT**	(Nested Task Flag)
16	**RF**	(Resume Flag)
17	**VM**	(Virtual 8086 Mode)

The status of individual bits can be evaluated with **conditional jumps**. Although it is possible (and allowed) to set individual bits, it is recommend that bits not be directly manipulated (with the exception of **CF**) as it could cause the **system a crash**!

3.3.4.5 Data Transfer Instructions

L	Load
S	Memory
SET	Set to 1
CLR	Set to 0
XCH	Exchange
LODSB	Load string element (1 Byte to AL)
LODSW	Load string element (Word to AX)
LODSD	String-Element laden (Dword to EAX)
STOSB	Store string element (AL to 1 Byte in EDI)
STOSW	Store string element (AX to Word in EDI)
STOSD	Store string element (EAX to Dword in EDI)
MOVSB	Copy data (Byte)
MOVSW	Copy data (Word)
MOVSD	Copy data (DWord)
INB	Read byte/word from port address (Byte)
INW	Read byte/word from port address (Word)
IND	Read byte/word from port address (DWord)
INSB	Read string from a port (Byte)
INSW	Read string from port (Word)
INSD	Read string from port (Word)
OUTB	Save AL to port address (Byte)
OUTW	Save AX to port address (Word)
OUTD	Save EAX to a port address (DWord)
OUTSB	Write string to port (Byte)
OUTSW	Write string to port (Word)
OUTSD	Write string to port (DWord)
PSH	Save register contents or data to stack (Byte, Word or DWord).

PUSH	Save register contents or data to stack (Byte, Word or DWord).
PSHA	Save register contents to stack (all 2-Byte Registers: AX, CX, DX, BX, SP, BP, SI, DI).
PSHF	Save register contents to stack (1-Byte Flag-Register)
PUSHAD	Save register contents to stack (all 4-Byte Register: EAX, ECX, EDX, EBX, original ESP, EBP, ESI, EDI).
PSHFD	Save register contents to stack (4-Byte Flag-Register).
POP	Load register contents or data from stack (Byte, Word or Dword).
POPA	Load register contents or data from stack (all 2-Byte-Register: AX, CX, DX, BX, SP, BP, SI, DI).
POPAD	Load register contents or data from stack (all 4-Byte Register: EAX, ECX, EDX, EBX, original ESP, EBP, ESI, EDI).
POPF	Load register contents or data from stack (1-Byte Flag Register).
POPFD	(4-Byte Flag Register)
LEA	Load effective address of a memory location.
LDS	Load pointer to DS register
LES	Load pointer to ES register
LAHF	Load contents of FL register to AH register
SAHF	Store contents of AH register in FL register
RST	Conditional reset of memory space
PRS	Conditional setting of memory space
LSS	Segment Load/offset pointer (SS-Register)
LGS	Segment Load/offset pointer (GS-Register)
LFS	Segment Load/offset pointer (FS-Register)
SETA	Set byte, if greater than (A = Above)
SETAE	Set byte, if greater than or equal (AE = Above/ Equal).
SETB	Set byte if les than (B = Below)
SETBE	Set byte if less than or equal (BE = Below/ Equal).
SETC	Set byte if carry (C = Carry)
SETE	Set byte if equal (E = Equal)
SETG	Set byte if greater than (Signed, G = Greater).

SETGE	Set byte if greater than or equal (signed, GE = Greater/ Equal)
SETL	Set byte if less than (Signed, L = Less).
SETLE	Set byte if less than or equal (signed, LE = Less/ Equal)
SETNA	Set byte if not greater than (NA = Not Above)
SETNAE	Set byte if not greater than or equal (NAE = Not Above/ Equal).
SETNB	Set byte if not less than (NB = Not Below)
SETNBE	Set byte if not less than or equal (NBE = Not Below/ Equal).
SETNC	Set byte if no carry (NC = Not Carry)
SETNE	Set byte if unequal (NE = Not Equal)
SETNG	Set byte if not greater than (Signed, NG = Not Greater).
SETNGE	Set byte if not greater than or equal (signed, NGE = Not Greater/ Equal)
SETNL	Set byte if not less than (Signed, NL = Not Less).
SETNLE	Set byte if not less than or equal (signed, NLE = Not Less/ Equal)
SETNO	Set byte if no Overflow (NO = No Overflow)
SETNP	Set byte if no parity (NP = Not Parity)
SETNS	Set byte if no sign (NS = Not Signal)
SETNZ	Set byte if not Null (NZ = Not Zero)
SETO	Set byte if Overflow (O = Overflow)
SETP	Set byte if parity (P = Parity)
SETPE	Set byte if even Parity (PE = Parity Even)
SETPO	Set byte if odd Parity (PO = Parity Odd)
SETS	Set byte if negative (S = Signed)
SETZ	Set byte if Null (Z = Zero)
LSXB	Copy with sign extension (SXB = Xtend Signed Byte)
LSXW	Copy with sign extension (SXW = Xtend Signed Word)
LZXB	Copy with sign extension (ZXB = Xtend Byte with Zero)
LZXW	Copy with sign extension (ZXW = Xtend Word with Zero)

3.3.4.51 L - LOAD

With this instruction, the defined register or memory space is loaded with value. The Load instruction must always be at the beginning of a link.

```
L <Target> <Source>
```

Changes in the **FL-Register:** none.



```
L Varible1            // AL <-- Variable1
L rabbit              // AL <-- Contents of 'rabbit'
L.ESI #Variable1      // SI <-- Addresses Variable1
L.BX AX               // BX <-- Contents of AX
L.CX (ESI+100)        // CX <-- (ESI+100) assigned to contents of address
L.B rabbit , 010      // 'rabbit' <-- Number 10
L.W (EBX) , -00001    // EBX contains the value -1 and assigned to word.
L.D var32, 0000000001 // Loading a doubleword variable
L.EAX #Function()     // Function address loaded
```

3.3.4.52 S - SAVE

This instruction saves the contents of a register to a defined memory location.

```
S <Source> <Destination>
```

Changes in the FL register: none



```
S Variable1          // Variable1 <-- AL
S Variable1          // Variable1 <-- AL
S.CX Variable2       // Variable2 <-- CX
                     // (Variable2 <-- CL
                     // Variable2:H
                     // <-- CH)
S.AX BX              // BX <-- AX
S.AX (EBX+ESI+0)     // The address specified by the sum of the values
                     // in EBX+ESI are stored in AX.
S.EBX var32          //var32 <-- EBX
```

3.3.4.53 SET

The defined memory location is set to **1**.

This instruction is actually the abbreviation for:

```
L.B <Address>, 001
```



This instruction is not permitted for indexed address, Word addresses and registers! It is for global and 1-byte variables only!

```
SET (BX+002) // <= not allowed!
```

Changes in the **FL** register: None



```
SET Motor // Must be a global variable
```

3.3.4.54 CLR

The defined memory location is set to **0**.

This instruction is actually the abbreviation for:

```
L.B <Address>, 000
```



This instruction is not allowed for indexed and word addresses!

Changes in the **FL** register: None

The **CLR AX** or **CLR AL** instructions are **SPECIAL CASES**.



Because the CLR instruction is not a processor instruction, the function must be converted so that it can be used in the AX/AL registers.

```
CLR AX –> X.AX AX
```
or
```
CLR AL –> X.AL AL
```
For inexperienced program, it is difficult to understand why **X.AX AX** clears the **AX** register. Therefore the **CLR AX** instruction was created.



This instruction is not allowed for segment registers!

Changes in the **FL** register: CF, ZF, SF, PF, OF and AF are defined.



```
CLR Lamp
CLR AX
CLR SI
CLR BL
```

3.3.4.55 XCH

The contents of a register are exchanged with those of another register (or memory location).

Changes in the **FL** register: None



```
XCH.AX BX
XCH.DX Variable1
XCH.BL (EBX+EDI)
```

3.3.4.56 LODSB

Load String Element

The **AL** register, (**LODSB**), **AX**-register (**LODSW**) or the **EAX**-Register (**LODSD**) are Loaded with the contents of a memory location, whose address was entered in the ESI register and ESI is then raised by **1**, **2** (**LODSW**) or **4** (**LODSD**). If the **DF** bit is already set to 1 with the **STD instruction**, ESI is reduced by **1, 2** or **4**. Since the **ESI** is required for this instruction in order to point to the memory location, it must be loaded before this instruction is executed.

Instruction	LODSB	LODSW
Replaced instruction	L (ESI) INC ESI	L.AX (ESI) ADD.ESI 2

Changes in the **FL** register: None



```
LODSB  // AL <= ESI If DF=0: ESI=ESI+1 then ESI=ESI-1
LODSW  // AX <= ESI If DF=0: ESI=ESI+2 then ESI=ESI-2
LODSD  // EAX <= ESI If DF=0: ESI=ESI+4 then ESI=ESI-4
```

3.3.4.57 LODSW

Load String Element

The **AL** register, (**LODSB**), **AX**-register (**LODSW**) or the **EAX**-Register (**LODSD**) are Loaded with the contents of a memory location, whose address was entered in the ESI register and ESI is then raised by **1**, **2** (**LODSW**) or **4** (**LODSD**). If the **DF** bit is already set to 1 with the **STD instruction**, ESI is reduced by **1,2** or **4**. Since the **ESI** is required for this instruction in order to point to the memory location, it must be loaded before this instruction is executed.

Instruction	LODSB	LODSW
Replaced instruction	L (ESI) INC ESI	L.AX (ESI) ADD.ESI 2

Changes in the **FL** register: None



```
LODSB  // AL <=  ESI If DF=0: ESI=ESI+1 then ESI=ESI-1
LODSW  // AX <=  ESI If DF=0: ESI=ESI+2 then ESI=ESI-2
LODSD  // EAX <= ESI If DF=0: ESI=ESI+4 then ESI=ESI-4
```

3.3.4.58 LODSD

The **EAX** register is Loaded with the contents of a memory location, which was addressed in the **ESI register**. **ESI** is then raised by for. If the **DF** bit is already set to **1** with the **STD** instruction, **ESI** is reduced by **4**.

At the time the instruction is created, **ESI** must have a valid address.

Instruction	LODSD
Replace instruction	L.EAX (ESI) ADD.ESI 004

Changes in the **FL** register: None



```
LODSB  // AL <= ESI If DF=0: ESI=ESI+1 then ESI=ESI-1
LODSW  // AX <= ESI If DF=0: ESI=ESI+2 then ESI=ESI-2
LODSD  // EAX <= ESI If DF=0: ESI=ESI+4 then ESI=ESI-4
```

3.3.4.59 STOSB

Store String Element

The contents of the **AL**-Registers (**STOSB**), **AX**-Registers (**STOSW**) of the **EAX**-Registers (**STOSD**) are copied to the memory location addressed in **EDI**. **EDI** is then raised by **1**, **2** or **4**. If the **DF** bit has already been set to 1 with the **STD** instruction, **EDI** is reduced by **1, 2** or **4**. Since **EDI** is needed for this instruction to point to a valid location in memory, it must have a valid address before the instruction is executed.

Instruction	STOSB	STOSW
Replaced instruction	S (EDI) INC EDI	S.AX (EDI) ADD.EDI 2

Changes in the **FL** register: None



```
STOSB  // AL <= EDI If DF=0: EDI=EDI+1 else EDI=EDI-1
STOSW  // AX <= EDI If DF=0: EDI=EDI+2 else EDI=EDI-2
STOSD  // EAX <= EDI If DF=0: EDI=EDI+4 then EDI=EDI-4
```

3.3.4.510 STOSW

Store String Element

The contents of the **AL**-Registers (**STOSB**), **AX**-Registers (**STOSW**) of the **EAX**-Registers (**STOSD**) are copied to the memory location addressed in **EDI**. **EDI** is then raised by **1**, **2** or **4**. If the **DF** bit has already been set to 1 with the **STD** instruction, **EDI** is reduced by **1, 2** or **4**. Since **EDI** is needed for this instruction to point to a valid location in memory, it must have a valid address before the instruction is executed.

Instruction	STOSB	STOSW
Replaced instruction	S (EDI) INC EDI	S.AX (EDI) ADD.EDI 2

Changes in the **FL** register: None



```
STOSB  // AL <= EDI If DF=0: EDI=EDI+1 else EDI=EDI-1
STOSW  // AX <= EDI If DF=0: EDI=EDI+2 else EDI=EDI-2
STOSD  // EAX <= EDI If DF=0: EDI=EDI+4 then EDI=EDI-4
```

3.3.4.511 STOSD

The contents of the **EAX** register stored in the memory location, whose address is defined in the **EDI** register. EDI is the raised by **4**.

If the **DF** bit has already been set to 1 with the **STD** instruction, **EDI** is reduced by **4**. Since **EDI** is needed for this instruction to point to a valid location in memory, it must have a valid address before the instruction is executed.

Replaced instruction:

```
S.EAX (EDI)
```

```
ADD.EDI 00004
```

Changes in the **FL** register: None



```
STOSB  // AL <= EDI If DF=0: EDI=EDI+1 else EDI=EDI-1
STOSW  // AX <= EDI If DF=0: EDI=EDI+2 else EDI=EDI-2
STOSD  // EAX <= EDI If DF=0: EDI=EDI+4 then EDI=EDI-4
```

3.3.4.512 MOVSB

Copy data (Byte/Word/Dword)

The content of the memory location, whose address is stored in the **ESI** register, is copied to the memory location whose address is defined in the **EDI** register. The **ESI** and **EDI** registers are then raised by **1** or **2**. **(If DF equals 1, they are reduced, see LODSB)**.

When using this instruction, the following should be observed:

The **ESI** and **EDI** registers must contain valid addresses.

The data source addressed with **ESI** and the destination with **EDI**.

If the **REP** (**repeat**) instruction is used, the **MOVSB** (**MOVSW**) is repeated as many times as was defined in the **ECX** register. Since the **ECX** register is decremented by 1, it will contain 0 once the **REP** instruction is complete.

If a data block has to be copied to an overlapping memory area and the target address is **higher** than the **source address**, the data block must be copied from the **higher** to the **lower address**. Otherwise data could be overwritten before it's copied.

In this case, **DF** must be set with **STD** before copying. If the copy was ended, **DF** must be cleared using **CLD**.

Changes in the **FL** register: None



Character strings (Character string **source**) with a length of 10 bytes should be copied to the **dest** character string. Since these areas don't overlap, no problems occur during the copy process.

```
L.ESI #source
L.EDI #dest
L.ECX 00005 // 5 * sizeof(UINT) = 10
```

```
REP
MOVSW
```



Characters strings (Character string **source**) should be moved upward with a Word (**D 0011** to **D 0015**).

The procedure is the same as shown in the previous example providing the value in the **source** was not overwritten by the value first copied (**source[9]**), which means that it wasn't destroyed before is was copied. This procedure must **start** at the beginning and not at end.

```
L.ESI #source[8]  // start with the last word
L.EDI #source[10]
L.ECX 00005       // 5 Words
STD               // Direction Flag to "Downwards"
REP
MOVSW
CLAD
```

The content from **DWORD**, whose address is defined in the **ESI** register, is copied to the memory location defined in **EDI**. The **ESI** and **EDI** registers are then raise or reduced by one depending on the contents in **DF** (0 or 1).

Changes in the **FL** register: None

3.3.4.513 MOVSW

Copy data (Byte/Word/Dword)

The content of the memory location, whose address is stored in the **ESI** register, is copied to the memory location whose address is defined in the **EDI** register. The **ESI** and **EDI** registers are then raised by **1** or **2**. **(If DF equals 1, they are reduced, see LODSB)**.

When using this instruction, the following should be observed:

The **ESI** and **EDI** registers must contain valid addresses.

The data source addressed with **ESI** and the destination with **EDI**.

If the **REP** (**repeat**) instruction is used, the **MOVSB** (**MOVSW**) is repeated as many times as was defined in the **ECX** register. Since the **ECX** register is decremented by 1, it will contain 0 once the **REP** instruction is complete.

If a data block has to be copied to an overlapping memory area and the target address is **higher** than the **source address**, the data block must be copied from the **higher** to the **lower address**. Otherwise data could be overwritten before it's copied.

In this case, **DF** must be set with **STD** before copying. If the copy was ended, **DF** must be cleared using **CLD**.

Changes in the **FL** register: None



Character strings (Character string **source**) with a length of 10 bytes should be copied to the **dest** character string. Since these areas don't overlap, no problems occur during the copy process.

```
L.ESI #source
L.EDI #dest
L.ECX 00005 // 5 * sizeof(UINT) = 10
REP
MOVSW
```



Characters strings (Character string **source**) should be moved upward with a Word (**D 0011** to **D 0015**).

The procedure is the same as shown in the previous example providing the value in the **source** was not overwritten by the value first copied (**source[9]**), which means that it wasn't destroyed before is was copied. This procedure must **start** at the beginning and not at end.

```
L.ESI #source[8]  // start with the last word
L.EDI #source[10]
L.ECX 00005       // 5 Words
STD               // Direction Flag to "Downwards"
REP
MOVSW
CLAD
```

The content from **DWORD**, whose address is defined in the **ESI** register, is copied to the memory location defined in **EDI**. The **ESI** and **EDI** registers are then raise or reduced by one depending on the contents in **DF** (0 or 1).

Changes in the **FL** register: None

3.3.4.514 MOVSD

Copy data (Byte/Word/Dword)

The content of the memory location, whose address is stored in the **ESI** register, is copied to the memory location whose address is defined in the **EDI** register. The **ESI** and **EDI** registers are then raised by **1** or **2**. **(If DF equals 1, they are reduced, see LODSB)**.

When using this instruction, the following should be observed:

The **ESI** and **EDI** registers must contain valid addresses.

The data source addressed with **ESI** and the destination with **EDI**.

If the **REP** (**repeat**) instruction is used, the **MOVSB** (**MOVSW**) is repeated as many times as was defined in the **ECX** register. Since the **ECX** register is decremented by 1, it will contain 0 once the **REP** instruction is complete.

If a data block has to be copied to an overlapping memory area and the target address is **higher** than the **source address**, the data block must be copied from the **higher** to the **lower address**. Otherwise data could be overwritten before it's copied.

In this case, **DF** must be set with **STD** before copying. If the copy was ended, **DF** must be cleared using **CLD**.

Changes in the **FL** register: None



Character strings (Character string **source**) with a length of 10 bytes should be copied to the **dest** character string. Since these areas don't overlap, no problems occur during the copy process.

```
L.ESI #source
L.EDI #dest
L.ECX 00005 // 5 * sizeof(UINT) = 10
REP
MOVSW
```



Characters strings (Character string **source**) should be moved upward with a Word (**D 0011** to **D 0015**).

The procedure is the same as shown in the previous example providing the value in the **source** was not overwritten by the value first copied (**source[9]**), which means that it

wasn't destroyed before is was copied. This procedure must **start** at the beginning and not at end.

```
L.ESI #source[8]  // start with the last word
L.EDI #source[10]
L.ECX 00005       // 5 Words
STD               // Direction Flag to "Downwards"
REP
MOVSW
CLAD
```

The content from **DWORD**, whose address is defined in the **ESI** register, is copied to the memory location defined in **EDI**. The **ESI** and **EDI** registers are then raise or reduced by one depending on the contents in **DF** (0 or 1).

Changes in the **FL** register: None

3.3.4.515 INB

Read Byte/Word/DWord from the Port Address

The process port, which is addressed through the contents of the **EDX** register, is transferred to the **AL** (INB), **AX** (INW), or **EAX** (IND) registers.

This instruction is used in the control only when the system time is read or the V.24 interface integrated in the CPU is addressed.

Hardware knowledge is required for use with this application.



This instruction should be used only in conjunction with the application examples available at SIGMATEK.

Changes in the **FL** register: None

```
INB
INW
IND
```

3.3.4.516 INW

See INB.

3.3.4.517 IND

See INB.

3.3.4.518 INSB

Read String form Port

The **INSB** instruction transmits data from the input port, which is numbered by the EDX register, to the **Byte**, **Word** or **Dword** memory of the **EDI**.

After data is transmitted, the **EDI** register expands automatically.

If the **DF** flag is **0** (a **CLAD** instruction was executed), the **EDI** register expands. If the **DF** flag is **1**, an **STD** instruction was executed and the **EDI** is reduced by **1** if a byte was entered, **2** if a word was entered and **4** for a **DWord**.

Affected Flags: None

3.3.4.519 INSW

Read String form Port

The **INSW** instruction transmits data from the input port, which is numbered by the EDX register, to the **Byte**, **Word** or **Dword** memory of he **EDI**.

After data is transmitted, the **EDI** register expands automatically.

If the **DF** flag is **0** (a **CLAD** instruction was executed), the **EDI** register expands. If the **DF** flag is **1**, an **STD** instruction was executed and the **EDI** is reduced by **1** if a **byte** was entered, **2** if a word was entered and 4 for a **DWord**.

Affected Flags: None

3.3.4.520 INSD

Read String form Port

The **INSB** instruction transmits data from the input port, which is numbered by the EDX register, to the **Byte**, **Word** or **Dword** memory of the **EDI**.

After data is transmitted, the **EDI** register expands automatically.

If the **DF** flag is **0** (a **CLAD** instruction was executed), the **EDI** register expands. If the **DF** flag is **1**, an **STD** instruction was executed and the **EDI** is reduced by **1** if a byte was entered, **2** if a word was entered and **4** for a **DWord**.

Affected Flags: None

3.3.4.521 OUTB

Save AL/AX/EAX to port address

The **AL** (OUTB), **AX** (OUTW), or **EAX** (OUTD) registers is transferred to the processor port defined by the contents of the **EDX** register.

This instruction is used in the control only when the system time or **interrupt controller** was changed or the **V.24 interface** in the CPU is accessed by from the application program.

For this instruction **precise Hardware knowledge** is required.



These commands should be used in conjunction with one of the application examples provided by SIGMATEK engineers.

Changes in the **FL** register: None

```
OUTB
OUTW
OUTD
```

3.3.4.522 OUTW

See OUTB.

3.3.4.523 OUTD

See OUTB.

3.3.4.524 OUTSB

Write to String to Port

The **OUTSB** instruction transfers data from the **Byte, Word** or **DWord** memory in the **source index** register to the output port addressed through the **EDX** register. The ESI register is used as the **source index** register. After the transfer, the EDI register expands automatically.

If the DF flag is **0** (a **CLAD** instruction was executed), the **EDI** register expands. If the DF flag is **1**, an STD instruction was executed and the EDI is reduced by 1 if a byte was entered, **2** if a word was entered and **4** for a **Word**.

Affected Flags: None

3.3.4.525 OUTSW

Write to String to Port

The **OUTSW** instruction transfers data from the **Byte, Word** or **DWord** memory in the **source index** register to the output port addressed through the **EDX** register. The ESI register is used as the **source index** register. After the transfer, the EDI register expands automatically.

If the DF flag is **0** (a **CLAD** instruction was executed), the **EDI** register expands. If the DF flag is **1**, an STD instruction was executed and the EDI is reduced by 1 if a byte was entered, **2** if a word was entered and **4** for a **Word**.

Affected Flags: None

3.3.4.526 OUTSD

Write string to port

The **OUTSD** instruction transfers data from the **Byte, Word** or **DWord** memory in the **source index** register to the output port addressed through the **EDX** register. The ESI register is used as the **source index** register. After the transfer, the EDI register expands automatically.

If the DF flag is **0** (a **CLAD** instruction was executed), the **EDI** register expands. If the DF flag is **1**, an STD instruction was executed and the EDI is reduced by 1 if a byte was entered, 2 if a word was entered and **4** for a **DWord**.

Affected Flags: None

3.3.4.527 PUSH

Store Register Contents or Data in the Stack.

The defined register of contents of a memory address is put in the **processor stack**. The stack pointer is previously decremented by **2** or **4**. After the instruction is executed, the stack pointer points to the **WORD** or **DWORD** stored in the **stack This-instruction** is used **store register** or **memory contents**. The **POP** instruction can then be used to restore the data if destroyed.

This instruction is for **16** or **32-bit information** (**Words/Dwords**). One byte can be pushed however. E.g.: PUSH.B MyByte.

This instruction cannot be used with **AL, AH, BL, BH, CL, CH, DL, DH** registers.

If the instruction is used for data memory, **2** or **4 sequential bytes** are always transferred. 1 Byte is only allowed for registers.



The amount of PSH instructions must be the same as the amount of POP instructions in a subroutine called with the CALL instruction. The CALL instruction stores the return addresses in the stack.

With the **PUSH** and **POP** instructions, the pointer changes to the next free memory location on the **STACK**, which means that the **RET instruction** receives the wrong return address and the program jumps back to the wrong line number. This usually causes a **system crash**.

Special Forms

PSHA

All registers (with exception of the **SEG** register and the **FL** register) are pushed in the following order: **AX, CX, DX, BX, SP, BP, SI, DI**.

PHSA

Utilized in subroutines if all registers called or excluded in a function call have the same value.

PSHF

The **FL**-register is stored in the stack.

Changes in the **FL** register: None



```
PSH AX
PSH ES
PSH Variable1
PSH (EBX)
PSH 022
PSHA
PSHF
PSHHD
PSHFD
```

PUSHAD stores all **32 bit general registers** on the stack. **PUSHAD** decrements the stack pointer (**ESP**) by **32** to contain eight **DWORDS**. The registers are sent to the stack in the following order:

EAX, ECX, EDX, EBX, original ESP, EBP, ESI, EDI

Changes in the **FL** register: None

PUSHFD decrements the stack pointer by **4** and copies the **80386 EFLAGS register** to the new top of the stack.

3.3.4.528 PSH

See PUSH.

3.3.4.529 PSHA

See PUSH.

3.3.4.530 PSHF

See PUSH.

3.3.4.531 PUSHAD

See PUSH.

3.3.4.532 PSHFD

See PUSH.

3.3.4.533 POP

Load Register Contents or Data from the Stack

The defined register (or the addressed memory location) is loaded with the **last word** stored in the **STACK**. The stack pointer is then increased by **the data size**.

The instruction is used to Load registers (or memory contents) with values that have been put on the stack with **PSH instructions**.

This instruction is for **16** or **32 Bit Information** (**Words/Dwords**) only.

This instruction cannot be used with **AL, AH, BL, BH, CL, CH, DL, DH** registers.

If this instruction is used, **2** or **4 sequential bytes** are transferred. Only at registers 1 Byte is not allowed.

3.3.4.534 POPA

All registers (with exception of the **SEG** register and the **FL** register) are loaded in the following sequence: **DI, SI, BP, SP, BX, DX,** and **CX, AX**.

3.3.4.535 POPAD

Extended registers with **POPAD DIAS** used to restore registers that are temporarily stored with PUSHAD.

3.3.4.536 POPF

Only the FL register is loaded from the stack.

The registers are loaded in the following order:

EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX

Changes in the FL register: None

3.3.4.537 POPFD

POPFD stores the **Dword** at the beginning of the stack and saves the value in the **EFLAGS** register. This is instruction has no influence on bits 16 and 17 (**Resume Flag** and **Virtual 8086 Mode**).



Since the stack is constructed like a stack of plates, the data placed on the stack FIRST is at the BOTTOM. Therefore, all information must be accessed in the same order in which it was placed (last in => first out).

All PSH instructions values must at some time be removed by a POP instruction, otherwise the stack will **overflow**. Which means that the amount of PSH and POP instructions must be the same.

Data transfers with PSH in a subroutine are not easy to execute because the **CALL** instruction puts the return address **on top** of the stack. The first POP instruction in the subroutine reads the return address instead of the actual data.

Changes in the **FL** register: None (except: **POPF**)



```
POP AX
POP ES
POP Variable1
POP EBX)
POPA
POPF
```

3.3.4.538 LEA

Load the actual address to a memory location.

The actual address of a memory location or bit position is loaded in the defined **32-bit register**.

Changes in the **FL** register: None



```
LEA.EAX #Variable1 // EAX <= Address of Variable1
```

3.3.4.539 LDS

The data segment register (**DS**), extra segment register (**ES**) and the assigned register are Loaded with value Loaded from the assigned address.

```
Reg    // <= (Adr)
DS/ES // <= (Adr+2)
```

This instruction assumes the segment and offset are in the correct sequence at the assigned address.

Changes in the **FL** register: None

3.3.4.540 LES

See LDS.

3.3.4.541 LAHF

Contents Loaded from the **FL** registers to the **AH** registers.

The **AH** register is Loaded with the contents of the **processor status registers**.

7	6	5	4	3	2	1	0
SF	ZF		AF		PF		CF

Changes in the **FL** register: None

3.3.4.542 SAHF

The contents of the **AH** registers are Loaded to the **FL** registers.

The process status register is Loaded with the contents of the **AH** register.

7	6	5	4	3	2	1	0
SF	ZF		AF		PF		CF

Changes in the **FL** register: **SF, ZF, AF, PF** and **CF**

3.3.4.543 RST

Conditional reset of a memory location. The assigned register is tested; if the value is 1 the memory address is set to 0.

Affected Flags: **AF, CF = 0, OF = 0, PF, SF, ZF**



```
RST.CL a1 // Set a1 to 0 if CL = 1
RST a1    // Set a1 to 0 if AL = 1
```

Syntax in Structured Text:

```
RST(a1) := e1; // Set a1 to 0 if e1 = 1
```

3.3.4.544 PRS

Conditional reset of a memory location. The assigned register is tested; if the value is 1 the memory address is set to 0.

Affected Flags: **AF, CF = 0, OF = 0, PF, SF, ZF**



```
PRS.CL  a1  // Set a1 to 1 if CL = 1
PRS     a1  // Set a1 to 1 if AL = 1
```

Syntax in Structured Text:

```
PRS(a1) := e1;   // Set a1 to 1 if e1 = 1
```

3.3.4.545 LSS

The **(GS/SS/FS)** registers and the defined registers are loaded with the stored value starting at the defined address.

```
Reg       // <= (Adr.)
GS/SS/FS  // <= (Adr. + 2)
```

This instruction requires that the segment and the offset be in the defined location in the correct sequence.

Changes in the **FL** register: None



```
LSS.EDI Variable1  // DI  Value in Variable1
                   // SS  Value in Variable2
```

3.3.4.546 LGS

The **(GS/SS/FS)** registers and the defined registers are loaded with the stored value starting at the defined address.

```
Reg      // <= (Adr.)
GS/SS/FS // <= (Adr. + 2)
```

This instruction requires that the segment and the offset be in the defined location in the correct sequence.

Changes in the **FL** register: None



```
LGS.EDI Variable1  // DI  Value in Variable1
                   // SS  Value in Variable2
```

3.3.4.547 LFS

The **(GS/SS/FS)** registers and the defined registers are loaded with the stored value starting at the defined address.

```
Reg      // <= (Adr.)
GS/SS/FS // <= (Adr. + 2)
```

This instruction requires that the segment and the offset be in the defined location in the correct sequence.

Changes in the **FL** register: None



```
LFS.EDI Variable1  // DI  Value in Variable1
                   // SS  Value in Variable2
```

3.3.4.548 SETA

SETA saves **1** as a byte value in the destination when **CF = 0** and **ZF = 0** or a **0** when the conditions aren't met.

Changes in the **FL** register: None



```
SETA BL
SETA Var1
```

3.3.4.549 SETAE

SETAE saves **1** as a byte value in the destination when **CF = 0** or a **0** when the conditions aren't met.

Changes in the **FL** register: None



```
SETAE BL
SETAE Var1
```

3.3.4.550 SETB

SETB saves a **1** as a byte value in the destination or a **0** when the condition isn't met.

Changes in the **FL** register: None



```
SETB BL
SETB Var1
```

3.3.4.551 SETBE

SETBE saves a **1** as a byte value in the destination when **CF = 1** or **ZF = 1** or a **0** if conditions aren't met.

Changes in the **FL** register: None



```
SETBE BL
SETB Var1
```

3.3.4.552 SETC

SETC saves a **1** as a byte value in the destination, if **CF = 1**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETC BL
SETC Var1
```

3.3.4.553 SETE

SETE saves a **1** as a byte value in the destination when **ZF = 1**,or a **0**, when conditions aren't met.

Changes in the **FL** register: None



```
SETE BL
SETE Var1
```

3.3.4.554 SETG

SETG saves a **1** as a byte value in the destination when **ZF = 0** or when **SF = OF**. A **0** is saved when conditions aren't met.

Changes in the **FL** register: None



```
SETG BL
SETG Var1
```

3.3.4.555 SETGE

SETGE saves a **1** as a byte value in the destination when **SF = OF** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETGE BL
SETGE Var1
```

3.3.4.556 SETL

SETL saves a **1** as a byte value in the destination when **SF <> OF**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETL BL
SETL Var1
```

3.3.4.557 SETLE

SETLE saves a **1** as a byte value in the destination when **ZF = 1** or **SF <> OF**. A **0** is saved when conditions aren't met.

Changes in the **FL** register: None



```
SETLE BL
SETLE Var1
```

3.3.4.558 SETNA

SETNA saves a **1** as a byte value in the destination when **CF = 1** or **ZF = 1**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNA BL
SETNA Var1
```

3.3.4.559 SETNAE

SETNAE saves a **1** as a byte value in the destination when **CF = 1** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNAE BL
SETNAE Var1
```

3.3.4.560 SETNB

SETNB saves a **1** as a byte value in the destination when **CF = 0** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNB BL
SETNB Var1
```

3.3.4.561 SETNBE

SETNBE saves a **1** as a byte value in the destination when **CF = 0** and **ZF = 0** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNBE BL
SETNBE Var1
```

3.3.4.562 SETNC

SETNC saves a **1** as a byte value in the destination when **CF = 0** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNC BL
SETNC Var1
```

3.3.4.563 SETNE

SETNE saves a **1** as a byte value in the destination when **ZF = 0** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNE BL
SETNE Var1
```

3.3.4.564 SETNG

SETNG saves a **1** as a byte value in the destination when **ZF = 1** or **SF <> OF**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNG BL
SETNG Var1
```

3.3.4.565 SETNGE

SETNGE saves a **1** as a byte value in the destination when **SF <> OF** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNGE BL
SETNGE Var1
```

3.3.4.566 SETNL

SETNL saves a **1** as a byte value in the destination when **SF = OF** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNL BL
SETNL Var1
```

3.3.4.567 SETNLE

SETNLE saves a **1** as a byte value in the destination when **ZF = 0** and **SF = OF** or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNLE BL
SETNLE Var1
```

3.3.4.568 SETNO

SETNO saves a **1** as a byte value in the destination when **OF = 0**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNO BL
SETNO Var1
```

3.3.4.569 SETNP

SETNP saves a **1** as a byte value in the destination when **PF = 0**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNP BL
SETNP Var1
```

3.3.4.570 SETNS

SETNS saves a **1** as a byte value in the destination when **SF = 0**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNS BL
SETNS Var1
```

3.3.4.571 SETNZ

SETNZ saves a **1** as a byte value in the destination when **ZF = 0**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETNZ BL
SETNZ Var1
```

3.3.4.572 SETO

SETO saves a **1** as a byte value in the destination when **OF = 1**, or when conditions aren't met.

Changes in the **FL** register: None



```
SETO BL
SETO Var1
```

3.3.4.573 SETP

SETP saves a **1** as a byte value in the destination when **PF = 1**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETP BL
SETP Var1
```

3.3.4.574 SETPE

SETPE saves a **1** as a byte value in the destination when **PF = 1**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETPE BL
SETPE Var1
```

3.3.4.575 SETPO

SETPO saves a **1** as a byte value in the destination when **PF = 0**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETPO BL
SETPO Var1
```

3.3.4.576 SETS

SETS saves a **1** as a byte value in the destination when **SF = 1**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETS BL
SETS Var1
```

3.3.4.577 SETZ

SETZ saves a **1** as a byte value in the destination when if **ZF = 1**, or **0** when conditions aren't met.

Changes in the **FL** register: None



```
SETZ BL
SETZ Var1
```

3.3.4.578 LSXB

LSXB reads the contents of the second operand as bytes then expands the character value to the size of the instruction operand attributes (**16** or **32 Bits**) and saves the result to the first operand.

Changes in the **FL** register: None



```
LSXB.BX Var1     // BL  Variable1
                 // BH  16#00 or 16#FF
LSXB.EBX Var1    // BL  Variable1
                 // higher 3 Bytes of EBX
                 // are filled with either 0
                 // or 1
```

3.3.4.579 LSXW

LSXW reads the contents of the second operand as Words then expands the character value to the size of the instruction operand attributes (**32 Bit**) and saves the result to the first operand.

Changes in the **FL** register: None



```
LSXW.EBX Var1   // BX  Variable1
                // Upper 2 Bytes of EBX are filled with either 0 or 1
```

3.3.4.580 LZXB

LZXB reads the contents of the second operand as a byte, executes an expansion by filling the new incoming bytes with zeros to a 4 byte value (**16** or **32** Bits) and saves the result to the first operand.

Changes in the **FL** register: None



```
LZXB.BX Var1    // BL  Variable1
                // BH  16#00
LZXB.EBX Var1   // BL  Variable1
                // Upper 3 Bytes of EBX
                // are filled with 0
```

3.3.4.581 LZXW

LZXW reads the contents of the second operand as a word, executes an expansion by filling the new incoming bytes with zeros to a 4 byte value (**32 Bits**) and saves the result to the first operand.

Changes in the **FL** register: None



```
LZXW.EBX Var1   // EBX  Variable1
                // Upper 2 Bytes of EBX
                // are filled with 0
```

3.3.4.6 Logic Instructions

O	bitwise OR
OR	bitwise OR
A	bitwise AND
AND	bitwise AND
X	bitwise EXCLUSIVE OR
XOR	bitwise EXCLUSIVE OR
NOT	bitwise Invert

3.3.4.61 O

Represents a bit wise **OR gate** between two operands.
The result is stored in the left operand.

Truth Table

Operand1	0	1	0	1
Operand2	0	0	1	1
Result	0	1	1	1

Application

- Sets an output when on or more input conditions are met (= 1).
- Sets bits in a Byte or Word.



```
O.B % D 0100 , 2#00001111 //Lower 4 bits to 1
```



```
value := value OR 16#0000FFFF
```

3.3.4.62 OR

See O.

3.3.4.63 A

Represents a bit wise **AND gate** between two operands.
The result is stored in the left operand.

Truth Table

Operand1	0	1	0	1
Operand2	0	0	1	1
Result	0	0	0	1

Application

- Sets an output when all input conditions are met (= 1).
- Clears a bit in a byte, Word or DWord.



```
A.B     b_var ,  2#11110000 // lower 4 bits to 0
```



```
value := value AND 16#FFFFFF00
```

3.3.4.64 AND

See A.

3.3.4.65 X (Abbreviation for XOR)

Represents a Bit wise **XOR gate** between two operands.

The result is stored in the left operand.

Truth Table

Operand1	0	1	0	1
Operand2	0	0	1	1
Result	0	1	1	0

Application

- Sets an output when two inputs are unlike
- Inverts individual bits in a Byte / Word.

Special Instructions

Clearing a register:



```
X.BX BX // Set BX register to 0.
```

If this instruction is used, the entry is accepted but is converted immediately to CLR BX.

This doesn't mean that there is cause for worry. It is simply a different syntax, since inexperienced programmers recognize CLR BX easier than the instruction X.BX BX.



```
value := value X 16#FFFFFFFF;
```

3.3.4.66 XOR

Represents a Bit wise **XOR gate** between two operands.

The result is stored in the left operand.

Truth Table

Operand1	0	1	0	1
Operand2	0	0	1	1
Result	0	1	1	0

Application

- Sets an output when two inputs are unlike
- Inverts individual bits in a Byte / Word.

Special Instructions

Clearing a register.



```
XOR.BX BX // Set BX register to 0.
```

If this instruction is used, the entry is accepted but is converted immediately to CLR BX.

This doesn't mean that there is cause for worry. It is simply a different syntax, since inexperienced programmers recognize CLR BX easier than the instruction X.BX BX.



```
value := value XOR 16#FFFFFFFF;
```

3.3.4.67 NOT

Inverts (negates) all bits in an Operand.

Truth Table

Operand1	0	1
Result	1	0

Application

- Unconditional output toggle.
- Inverts all bits in a Byte/Word/DWord.

Changes in the **FL** register: None



Exception: 1 toggled to 0 and 0 toggled to 1



```
value := NOT value;
```

3.3.4.7 Comparison Instructions

CMP	Compare
CMP0	Byte wise comparison with 0
CMP1	Byte wise comparison with 1
CMPSB	Compare two character strings
CMPSW	Compare two character strings

CMPSD	Compare two character strings
SCASB	Find an element in a character string
SCASW	Find an element in a character string
SCASD	Find an element in a character string
TEST	Test bits

3.3.4.71 CMP

Compares the left operands with the right operator. The **Right** operand is subtracted from the **Left** operand but the **result** from this function is not saved.

This instruction is used in preparation for a conditional jump instruction.

The instruction sequence is processed as follows:

```
CMP.AX 00100
JA MAXI
```

A program references the line with the name **MAXI** if **AX** is greater than **100**.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**



```
CMP.AX BX
JBE MAXI                // Jumps to MAXI if AX <= BX
CMP.B (ESI) , 020       // Jumps to ELVIS if the Byte addressed with ES
JA ELVIS
CMP.W Variable1, 1234   // Jumps to EQUAL if the content of Variable1 =
JE EQUAL
CMP Variable2           // Jumps to UNEQUAL if AL is not equal to bit V
JNZ UNEQUAL
```



Caution!

If the CMP command is used with a bit from a bit variable (e.g.: BitVar is of type BDINT; Code: „CMP.B BitVar.MyBit, 0“), the compiler must execute sever instructions internally. In the process, **the value of the EAX register can change!**

3.3.4.72 CMP0

The assigned memory location is compared with the value 0. The defined address is compared with the value 0 by bytes. This function subtracts the value from the address but does not save the result of the function. This instruction is used in preparation for a subsequent conditional jump instruction.

This instruction is the short form for:

```
CMP.B <Address>, 000
```



Not allowed for indexed and Word addresses!

The instruction sequence is processed as follows:

```
CMP0 Variable1
JE EQUAL
```

The program jumps to the line called **EQUAL** if variable1 contains the value **0**. This instruction can be used for data **memory locations**.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**

```
CMP0 Variable1  // Jumps to UNEQUAL if the contents
JNE UNEQUAL     // of Variable1 dont't equal 0
```

3.3.4.73 CMP1

The assigned memory location is compared with the value **1**. The defined address is compared with the value **1** by bytes. This function subtracts the value from the address but does **not save** the **result** of the function.

This instruction is used in preparation for a subsequent conditional jump instruction.

This instruction is the short form for:

```
CMP.B <Address>, 001
```



Not allowed for indexed and Word addresses!

The instruction sequence is processed as follows:

```
CMP1 Variable1
JE EQUAL
```

The program jumps to the line called **EQUAL** if variable1 contains the value **1**. This instruction can be used for data **memory locations**.

Changes in the FL register: **CF, ZF, SF, PF, OF, AF**



```
CMP1 Variable1
JE ON           // Jumps to ON if Variable2 = 1
CMP1 Variable2 // Jumps to EQUAL if the contents of Variable1 aren't
JNZ EQUAL       // equal to 1.
```

3.3.4.74 CMPSB

This instruction compares the operands addressed with **ESI** that were addressed through **EDI** and can be comprised of **bytes** (**CMPSB**), Words (**CMPSW**) or **Dwords** (**CMPSD**). Depending on the contents of the **DF** flag, the **ESI** and **EDI** registers are incremented (**DF = 0**) by **1** (**Byte**), **2** (**Word**) or **4** (**Dword**) or reduced if **DF = 1**.

The instruction sets the flags in the **FL 4** register to enable a conditional jump with the comparison.

Repeat functions

1. Preparatory instruction REP (repeats while unequal)

If the repeat **REP** is set before an instruction then a character string is compared with the defined length in the **ECX** register. The comparison is repeated until **ECX = 0** (the string contains no corresponding elements) or the first matching element is found. When a matching element is found, the **ESI** and **EDI** registers point to the very next element. The **ECX** register contains the number of the elements that are not matching.

2. Preparatory instruction REPZ (repeats while equal)

If the **REPZ** is set before an instruction, a character string is compared with the length defined in the **EXC** register. The comparison is repeated until **EXC = 0** (all contents were compared) or the first unlike element was found. When this occurs, the **ESI** and **EDI** registers point to the first element after the unequal string element and EXC contains the number of non-matching elements.



```
String 1: source : 1,2,3,4,5,6,7
String 2: dest   : 1,2,3,9,5,6,7
                                    // Elements in No. 3 are not equal (
L.ECX 007                           // Strings are 7 Bytes long
L.ESI source
L.EDI dest
REP
```

```
CMPSB                          // Search for the first compared ele
                               // ESI now points to #source[1], EDI
                               // ECX=6 since the first element is
L.ECX 007                      // Strings are 7 Bytes long
L.ESI source
L.EDI dest
REPZ
CMPSB                          // Search for the first non-matching
                               // ESI now points to #source[4], EDI
                               // ECX=3 since the fourth element do
```

3.3.4.75 CMPSW

See CMPSB.

3.3.4.76 CMPSD

See CMPSB.

3.3.4.77 SCASB

Compares **AL** (**SCASB**) with operand addressed with **EDI**.

Depending on the contents of **DF**, the **EDI** register is raised (**DF = 0**) or reduced (**DF = 1**) **1** (**Byte**), **2** (**Word**) or **4** (**DWord**). The instruction sets the flags in the **FL** register to enable a conditional jump with the comparison.

Repeat functions

1. Preparatory instruction REP (repeats while unequal)

If **REP** is placed before an instruction, an **array** of **bytes** or Words whose condition is compared with the contents defined in the **ECX** register. The comparison is repeated until **ECX=0** (the contents of **AL/AX/EAX** no longer exist in the string) or the first match is found. When a match is found the **EDI** points to the first element **after** the matching string element. The **ECX** register contains the number of elements that no longer match.

2. Preparatory instruction REPZ (repeats while equal)

If **REPZ** is placed before an instruction, an **array** of bytes or **Words** whose condition is compared with the contents defined the **ECX** register.

The comparison is repeated until **ECX = 0** (the strings match the contents of **AL/AX/EAX**) or the first unequal element is found. When this occurs, the **EDI** points to the first element after the non-matching string element. The **ECX** register contains the number of elements that **no** longer match.



```
String: Source: 1,2,3,4,5,6,  // This should determine whether value 3
L.ECX 00007                   // Strings are 7 Bytes long
L.EDI #source
L 003                         // Search for number 3
REP
SCASB                         // Search for the 1st match
                              // EDI now points to #source[3], ECX=4 s

L.ECX 00007                   // Strings are 7 Bytes long
L.EDI #source[0]
L 003                         // Search for number 3
REPZ
SCASB                         // Search for the 1st non-match
                              // EDI now points to #source[1], ECX=6 s
```

3.3.4.78 SCASW

See SCASB.

3.3.4.79 SCASD

See SCASB.

3.3.4.710 TEST

Executes a logic **AND** operation between two operands. No result is given; the status bits in the FL register are changed instead.

This instruction is used in preparation for conditional program jump depending on whether certain bits in the operand are equal or unequal.

Changes in the **FL** register: **CF=0, ZF, SF, PF, OF=0, AF=?**



```
TEST.B Variable1, 1  // Branches to ONE if bit 0 in Byte
JNZ ONE              // Variable1 = 1, because the result of the AND c
TEST.AX (ESI)        // Branches to NIX if Word addressed with
JE NIX               // ESI does not contain a bit that is 1 at the sa
                     // bit in AX
```

3.3.4.8 Shift Instructions

SHL	Shift left
SHR	Shift right
SHLD	Shift left
SHRD	Shift right
ROL	Rotate left
ROR	Rotate right
RCL	Rotate carry left
RCR	Rotate carry right
SAR	Arithmetic shift right
SAL	Arithmetic shift left

3.3.4.81 SHLD

SHLD shifts the second operand as far left as defined in the **counter operand**. The first operand, **a register**, contains the bits that are shifted from the right beginning with **31**. The result is store in the second operand and the first operand remains unchanged.

The **counter operand** is defined either as a **byte constant** or through the **CL register**. These operands are first processed with **MODULO 32** to find a number between **0** and **31**. Because the bits that should be shifted are located in a defined register, this operation is more suited for **exact shifts (64 bits or more)**. **CF** contains the value of the last shifted bit.

Changes in the **FL** register: **AF, CF, OF, PF, SF, ZF**



```
SHLD.EBX EDX, 004  // EDX is shifted 4 places to the left.
                   // The Bits shifted from the right are taken from
                   // the EBX register. EBX is not changed.
```

3.3.4.82 SHRD

SHRD shifts the second operand as far right as defined in the **counter operand**. The first operand, **a register**, contains the bits that are shifted from the left beginning with **0**. The result is stored in the second operand and the first operand remains unchanged.

The **counter operand** is defined either as a **byte constant** or through the **CL register**. These operands are first processed with **MODULO 32** to find a number between **0** and **31**. Because the bits that should be shifted are located in a defined register, this operation is more suited for **exact shifts** (**64 bits or more**). **CF** contains the value of the last shifted bit.

Changes in the **FL** register: **AF, CF, OF, PF, SF, ZF**



```
SHRD.EBX EDX, 004  // EDX is shifted 4 places to the right.
                   // The Bits shifted from the left are taken from
                   // the EBX register. EBX is not changed.
```

3.3.4.83 ROL

The operand is shifted to the left by the desired setting. With each shift process, the bit released from the right is filled with the bit shifted to the left. **CF** contains the last bit shifted.



```
value := value ROL amount;
```

3.3.4.84 ROR

The operand is shifted to the right by the desired setting. With each shift process, the bit released from the left is filled with the bit shifted to the right. **CF** contains the last bit shifted.



```
value := value ROR amount;
```

3.3.4.85 RCL

The operand is shifted to the left by the defined number. With each shift, the bits released from the right are filled with the contents of CF. The bits shifted to the left are then Loaded to CF.

Before	1	1	0	0	0	0	1	1	E.g.: CF = 0
After: CF = 1	1	0	0	0	0	1	1	0	Filled with old CF

Changes in the **FL** register: **CF, OF=0**



```
RCL.AX 002              // AX is rotated left two times
RCL CL                  // AL is rotated left through CL
RCL.W (EDI), CL         // The Word addressed with EDI is shifted lef
                        // the number in CL

RCL.B Variable1, 001    // The Data byte Variable1 is rotated to the
RCL.D var32, 0000000002
```

3.3.4.86 RCR

The operand is shifted to the right by the defined number. With each shift, the bits released from the left are filled with the contents of **CF**. The bits, shifted to the left are then Loaded to **CF**. after this instruction has been executed, **CF** contains the last shifted bit.

Before	1	1	0	0	0	0	1	1	E.g.: CF = 0
After: CF = 1	0	1	1	0	0	0	0	1	Filled with old CF

Changes in the **FL** register: **CF, OF=0**



```
RCR.AX 002              // AX is rotated right two times
RCR CL                  // AL is rotated right through CL
RCR.W (DI), CL          // The Word addressed with DI is shifted right
                        // the number in CL
RCR.B Variable1, 001    // The Data byte Variable1 is rotated to the r
RCR.D var32, 0000000001
```

3.3.4.87 SAL

The **SAL** instruction shifts the bits of the operands upward, but leaves the highest bit as it is. Signed calculations are thereby possible. The bit shifted higher is moved to the **CF** flag and the lower shifted bit is cleared.

Changes in the **FL** register: The **OF** flag is affected with each shift and is undefined with multiple shifts. The **CF, ZF, PF** and **SF** Flags are set according to the result.



```
SAL.AX 002      // AX is shifted left 2 times
SAL CL          // AL is shifted left CL times
SAL.W <DI>, CL  // The Word addressed through DI is shifted left CL t
```

3.3.4.88 SAR

The **SAR** instruction shifts the bits of the operands downward, but leaves the highest bit as it is. Signed calculations are thereby possible. The lowest bit is moved to the **CF**-Flag. This results in the (signed) division of the operands by 2.

Changes in the **FL** register: The **OF** flag is affected with each shift and is undefined with multiple shifts. The **CF, ZF, PF** and **SF** Flags are set according to the result.



```
SAR.AX 002      // AX is shifted right 2 times
SAR CL          // AL is shifted right CL times
SAR.W <DI>, CL  // The Word addressed through DI is shifted right CL times
```

3.3.4.89 SHL

The **SHL** instruction shifts the bits of an operand upward, the bit shifted higher is moved to the **CF** flag and the lower shifted bit is cleared.

Changes in the **FL** register: The **OF** flag is affected with each shift and is undefined with multiple shifts. The **CF, ZF, PF** and **SF** Flags are set according to the result.



```
SHL.AX 002      // AX is shifted left 2 times
SHL CL          // AL is shifted left CL times
SHL.W <DI>, CL  // The Word addressed through DI is shifted left CL times
```

3.3.4.810 SHR

The **SHR** instructions shift bits downward. The lower shifted bit is moved to the **CF** flag, dividing the operand by 2.

Changes in the **FL** register: The **OF** flag is affected with each shift and is undefined with multiple shifts. The **CF, ZF, PF** and **SF** Flags are set according to the result.



```
SHR.AX 002       // AX is shifted right 2 times
SHR CL           // AL is shifted right CL times
SHR.W <DI>, CL   // The Word addressed through DI is shifted right CL times
```

3.3.4.9 Jump Instructions

JZ	Jump if 0
JE	Jump if equal
JNZ	Jump if not 0
JNE	Jump if not equal
JNBE	Jump if greater than (ZF = 0 and CF = 0)
JA	Jump if greater than (ZF = 0 and CF = 0)
JC	Jump if less than (CF = 1)

JNAE	Jump if less than (CF = 1)
JB	Jump if less than (CF = 1)
JNC	Jump if greater than or equal (ZF = 1 and CF = 0)
JNB	Jump if greater than or equal (ZF = 1 and CF = 0)
JAE	Jump if greater than or equal (ZF = 1 and CF = 0)
JBE	Jump if less than or equal (CF = 0 or ZF = 1)
JNA	Jump if less than or equal (CF = 0 or ZF = 1)
JS	Jump if negative (SF = 1)
JNLE	Jump if greater than
JG	Jump if greater than
JNL	Jump if greater than or equal
JGE	Jump if greater than or equal
JNGE	Jump if less than
JL	Jump if less than
JLE	Jump less than or equal
JNG	Jump less than or equal
JO	Jump if overflow
JNO	Jump if no overflow
JPE	Jump if even parity (PF = 1)
JP	Jump if even parity (PF = 1)
JPO	Jump if odd parity (PF = 0)
JNP	Jump if odd parity (PF = 0)
JNS	Jump, if highest is Bit 0
JCXZ	Jump if cx = 0
JECXZ	Jump if the ECX registers = 0
LOOP	Repeat loop to ECX = 0
LOOPE	If ZF = 1 repeat loop CX = 0
LOOPNE	If ZF = 0 repeat loop CX = 0
JMP	Conditional jump

CALL	Call subroutine
RET	End subroutine
IRET	End interrupt program

3.3.4.91 JE

The jump is executed if **ZF** is set.

This occurs if:

- The result of a logic, arithmetic or shift instruction is **0**.
- Both operands for a **CMP** instruction equal.
- No matching bits were found in either of the operands of a **TEST** instruction.



```
CMP.AX BX              // Jump to LAB1, if AX=BX
JE LAB1                // Variable1 = 1, because the result of the AND
ADD.AX Variable1       // Jump to LAB1, if AX+Variable1 = 0
JE LAB2                // if an carry occurs is not evaluated.
TEST.B Variable2, 001  // Jump if Bit Variable2 = 0
JE LAB3
```

3.3.4.92 JNZ

The jump is executed if the **ZF** is not set.

This occurs when:

- The result of a logic, arithmetic or shift instruction does not equal **0**.
- Both operands of **CMP** instruction are unequal.
- At least one bit in each operand of a **TEST** instruction is 1.



```
CMP.AX BX            // Jump to LAB1, if AX<>BX
JNZ LAB1
ADD.AX Variable1     // Jump to LAB1, if AX+Variable1 <> 0
JNZ LAB2             // if an carry occurs it is not evaluated.
TEST.B Variable2, 1  // Jump if Bit0 in Variable2 = 1
JNZ LAB3
```

3.3.4.93 JNE

See JNZ.

3.3.4.94 JNBE

See JA.

3.3.4.95 JA

The jump is executed if neither **ZF** nor **CF**.

This occurs when:

- The result of a logic, arithmetic, or shift instruction is <> 0 and no carry exists.
- The left operand in a **CMP** instruction is greater than the right (unsigned variable)



```
CMP.AX BX          // Jump to LAB1, if AX > BX
JA LAB1
ADD.AX Variable1   // Jump to LAB2, if AX+Variable1 <> 0
JA LAB2
```

3.3.4.96 JC

See also JB.

3.3.4.97 JNAE

See JB.

3.3.4.98 JB

The jump is executed if **CF** is set.

This occurs if:

- A carry exists after an arithmetic instruction.
- The left operand is less than the right in a **CMP** instruction
- CF is set after a shift instruction.



```
CMP.AX BX          // Jump to LAB1, if AX < BX
JB LAB1
SUB.AX Variable1   // Jump to LAB2, if AX-Variable1 < 0
JB LAB2
SHR.AX 001         // Jump if Bit 0 was in AX 1
JB LAB3
```

3.3.4.99 JNC

See JAE.

3.3.4.910 JNB

See JAE.

3.3.4.911 JAE

The jump is executed if **CF** is not set.

This occurs if:

- No carry exists after an arithmetic instruction.
- After a shift instruction, **CF** contains a 0
- The left operand is >= the right in a **CMP**-instruction.



```
CMP.AX BX   // Jump to LAB1, if AX >= BX
JAE LAB1
SHL.AX 001  // Jump to LAB2, if Bit 15 in AX=0
JAE LAB2
```

3.3.4.912 JBE

See JNA.

3.3.4.913 JNA

The jump is executed if the **CF** or **ZF** is set.

This occurs when:

- After an arithmetic operation, a carry exists or the result is **0**.

-The left operand is less then or equal to the right in a **CMP** instruction.



```
CMP.AX BX          // Jump to LAB1, if AX <= BX
JNA LAB1
SUB.AX Variable 1  // Jump to LAB2, if AX - Variable1 <= 0
JNA LAB2
```

3.3.4.914 JS

Jump if the **sign Flag** = 1.

The jump is executed when the highest bit in the result is **1** after an operation.

This occurs when working with signed numbers and the result is **negative**.



```
SUB.AX Variable1 // Jump if the result is a
JS LBL1          // a negative number.
SHL.W (EBX) , 1  // Jump if the Word addressed with EBX Bit 15
JS LBL2A         // Equals 1 after a Shift instruction
```

3.3.4.915 JNLE

See JG.

3.3.4.916 JG

ZF = 0 and **SF = 0**

The jump is executed if two signed Operands are compared and the left operand is greater than the right.



```
CMP.AX Variable1  // Jump if AX greater than
JG LBL1           // the value in Variable1
```

3.3.4.917 JNL

See also JGE.

3.3.4.918 JGE

SF = 0

The jump is executed if two signed Operands are compared and the left operand is greater than or equal to the right.



```
CMP.AX Variable1  // Jump if AX is greater than or equal to the value
JGE LBL1          // in Variable1.
```

3.3.4.919 JNGE

See also JL.

3.3.4.920 JL

SF <> 0

The jump is executed if two signed Operands are compared and the left operand is less than the right.



```
CMP.AX Variable1  // Jump, if AX is less than the value in
JL LBL1           // in Variable1.
```

3.3.4.921 JLE

ZF = 1 or SF <> 0

The jump is executed if two signed Operands are compared and the left operand is less than or equal to the right.



```
CMP.AX Variable1  // Jump if AX smaller or equal to the value
JLE LBL1          // in Variable1.
```

3.3.4.922 JNG

See also JLE.

3.3.4.923 JO

OF=1

The jump is executed if a carry from **Bit 3** to **Bit 4** occurred in a previous arithmetic operation.

3.3.4.924 JNO

OF = 0

The jump is executed if no carry from **Bit 3** to **Bit 4** occurred in a previous arithmetic operation.

3.3.4.925 JPE

PF = 1

The jump is executed if the number of 1-bits in the lower 8 bits of a previous arithmetic operation is even.

3.3.4.926 JP

PF = 0

See also JPE.

3.3.4.927 JPO

The jump is executed if the number of 1-bits in the lower 8 bits of a previous arithmetic operation is **odd**.

3.3.4.928 JNP

See JPO.

3.3.4.929 JNS

The jump is executed if the highest bit after an operation is 0.

This condition occurs when working with signed numbers and the result is 0.

3.3.4.930 JCXZ

The jump is executed if the contents of the **CX** registers are 0.

This instruction is employed in order to prevent entering a program loop in which **CS** serves as a counting register for the required loop cycles.

3.3.4.931 JECXZ

This jump is executed when the **ECX** register = **0**. This instruction is used prevent **ECS** from being used as a count register required for the "correct loop" at the start of a program loop.



```
JECXZ NOTHING  // If the Block length is 0,
REP            // the Block Shift is skipped,
MOVSB          // Otherwise a 64k Block of memory is
NOTHING -      // shifted.
```

3.3.4.932 LOOP

Function: **ECX** register is decremented. If it is then **<> 0**, a jump to the defined label is execute.

Application: Multiple processing of a program sequence depending on the value in **ECX**.

After the instruction: **ECX = 0**

Changes in the **FL** register: None



```
                    // The status of bits Variable1 - Variable32 should
L.ESI #Variable1    // ESI points to first bit
L.ECX 32            // 32 sequential bits
LABEL NEG.B(ESI)    // Negate bit addressed with SI
INC ESI             // Point to next
LOOP LABEL          // Repeat 32 times (value in CX)
```

3.3.4.933 LOOPE

Function

If **ZF** is set, the **ECX** register is decremented, if it is **<> 0**, a jump to the defined label is executed. Otherwise, the next instruction is processed.

Application

Repetitive execution of a program segment depending on **ZF**, however, the maximum number of executions is defined in **ECX**. After the instruction has been executed, **ECX** is reduced by the amount of loops completed.

Changes in the **FL** register: None



```
                        // The first bit whose status is 1 is searched in a
                        // of single bit memory locations Variable1 - Varia
L.ESI #Variable1        // ESI points to the first bit
                        // BEFORE the block of memory to be searched
L.ECX 00032             // 32 sequential bits
LABEL INC ESI           // Point SI to the next bit
TEST.B (ESI), 001       // Point to next LOOPE LABEL
                        // Search maximum 32 ESI now points to the first bi
                        // ECX contains the number of bits that are no long
```

3.3.4.934 LOOPNE

Function

If **ZF** is not set, then the **ECX** register is decremented. If **<> 0**, a jump is made to the defined label. Otherwise the next instruction is run.

Application

Repetitive execution of a program segment depending on the value of **ZF**, but only as often as defined in **ECX**.

After the instruction has been executed, **ECX** is reduced by the amount of completed loops.

Changes in the **FL** register: None



```
                        // The first bit with a status of "0" will be searc
                        // of single bit memory locations Variable1 - Varia
L.ESI #Variable1        // ESI points to the first bit
                        // BEFORE the block
L.ECX 00032             // 32 sequential bits
LABEL INC ESI           // Point ESI to the next bit
TEST.B (ESI), 001       // Point to next
                        // Search maximum 32 ESI now points to first bit no
                        // ECX contains the number of bits that are no long
```

3.3.4.935 JMP

Function

The program counter is set at the line number defined by the label from which the processor continues.

Application

For jumping over parts of a program not momentarily required (e.g. the section for manual operation when the machine is running in automatic operation mode).



```
JMP AUTO  // The processor continues processing at the line named Auto
```

3.3.4.936 CALL

CALL has the same function as **JMP** except that the processor stores the address of the program line following the subroutine in the stack. The **RET** instruction at the end of the subroutine lets the program continue at the line following the call for the subroutine.

Application

Calls program parts that are required more often in a program.



```
FUNCTION AWL MAIN
CMP.B MAN, 001    // Tests whether machine is in
JE MANUAL         // manual operation mode.
CALL AUTO         // No: Automatic subroutine call
JMP NEXT          // --> Look at what happens next
MANUAL CALL MAN   // Manual operation subroutine call
NEXT CALL SECURE  // Priority security function
END_FUNCTION      // ---- END MAIN PROGRAM ----
FUNCTION AWL AUTO // **** Subroutine: Automatic ***
....
END_FUNCTION
FUNCTION AWL MAN  // **** Subroutine: Manual *****
....
END_FUNCTION
```

3.3.4.937 RET

Triggers a return to the program line after from which the subroutine was called.

The processor assumes that the return address is in the stack.

For this reason this instruction is used to trigger a program segment, which is called using the call instruction. After the above instruction has been executed, the stack pointer is raised by 4.

The **END_FUNCTION** instruction calls the return function automatically.

3.3.4.938 FC

This instruction is obsolete. Use the CALL instruction instead.

3.3.4.939 CALLF

This instruction is obsolete. Use the CALL instruction instead.

3.3.4.940 MODEND

This instruction is obsolete. Use the RET instruction instead.

3.3.4.941 IRET

Triggers the return to each program line that was interrupted with an interrupt command.

The return is executed as follows:

1. **Instruction pointer EIP** is Loaded from stack, **SP=SP-4**
2. **CS** register is Load from stack, **SP=SP-4**
3. **Flag** register is Load from stack, **SP=SP-4**

This instruction is used to shut down an interrupt program that is running in an application program.



This function should only be used in conjunction with an application example from SIGMATEK!

3.3.4.10 Arithmetic instructions

INC	Increment a value by 1
DEC	Decrement a value by 1
ADD	Add
ADC	Add with carry
SUB	Subtract
SBC	Subtract with carry
NEG	Negate
MUL	Multiplication
IMUL	Signed Multiplication
DIV	Division
IDIV	Signed Division
MOD	Modulo-Division

3.3.4.101 INC

The operand is incremented by **1**.

This instruction can be used for **Byte** as well as for **Word** and **Dword** operands.

INC is mainly used for programming counters and for increasing address pointers in program loops.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**

Copy a table to the data space:



```
L.ESI #TAB1()     // SI points to table
L.ECX (ESI)       // The first dword holds the length!
L.EDI #field      // Data should start at D 0020
COPY1 L (ESI+4)   // +4 since 1st 4 bytes are only the length
STOSB             // Store to EDI with using Autoincrement
INC ESI           // Point to next byte in table
LOOP COPY1        // Repeat until CX=0
```

3.3.4.102 DEC

The operand is decremented by **1**.

This instruction can be used with **byte** and **word** and **dword** operands.

DEC is mainly used for programming counters and for increasing address pointers in program loops.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**

Each time the function is called, the value in the **counter** is reduced.

If the **counter** = 0, set flag = 1



```
CMP.W counter, 00000  // If 0: --> Output = 1
JE IS_0
DEC counter           // Counter - 1
JNZ NADA              // now = 0 ? Yes: Output = 1
IS_0 SET flag
NADA -
```

3.3.4.103 ADD

The left operand is replaced by the sum of the left and right operands.

A carry from a previous addition is not considered.

If a carry exists for the addition, **CF** is set.

This instruction is allowed for **byte** (8 bits) and **word** (16 bits) and **dword** (32 bits) operands, but both operands must have the same data length.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**



```
ADD.AX Var1      // Value in Var1 is added to AX
ADD.W Var1,AX    // AX is added to value in
ADD.ESI #LABEL() // Var1 Address of line "LABEL" is added to ESI
ADD.SI 00020     // Number 20 added to SI
ADD.B (EDI),003  // The value of the byte addressed with EDI is increa
ADD.D Var1,EAX
```

3.3.4.104 ADC

The left operand replaced by the sum of the left and write operands plus a carry if on exists.

In a carry exists in this addition, **CF** is set.

This instruction can be used with **Byte-** (8 Bits), **Word-** (16 Bits) and **DWord-** (32 bits) operands, however, both operands must have the **same data length**.

The ADC instruction is used to add **two or more bytes, Words** or **Dwords** since the carry from the previous position is also included.

The first addition must be established with the **ADD** instruction and began with the byte, Word or DWord that has the **lowest** value.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**

Add a 32-Bit value:



```
ADD.W Var1+0,AX  // The 32 Bit value in DX, AX is added to
ADC.W Var1+2,DX  // Var1+0,Var1+2. (D 1000 = lowest Word).
ADD.AX 00020     // 20 is added to the 32 Bit value in DX, AX
ADC.DX 00000     // If CF, then DX=DX+1
ADC.D Var1,EAX
```

3.3.4.105 SUB

The left operand is replaced with the difference between the left and right operand. A carry from a previous subtraction is not included in this case.

This instruction can be used with byte- (8-Bit), Word- (16-Bit) and DWord- (32-Bit) operands. Both operands must, however, have the same data width.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**



```
SUB.AX Var1        // The value in Var1 is subtracted from AX.
SUB.W Var1+2,AX    // AX is subtracted from the value in Var1.
SUB.SI DX          // DX is subtracted from SI.
SUB.SI 020         // SI is reduced by 20.
SUB.B (EDI), 003   // The value addressed with DI Data bytes is reduced by 3.
SUB.D (EBX), 0000000009
```

3.3.4.106 SBC

The left operand is replaced with the difference between the left and right operands plus a carry if one exists.

If a carry exists in this Subtraction exists, **CF** is set.

This instruction can be used with **Byte-** (8-Bit), **Word-** (16-Bit) and **DWord-** (32-Bit) operands. Both operands must, however, have the same **data width**. The SBC instruction is used with the subtraction of **two or more bytes, Words** or **Dwords** since the carry from the previous position is also included.

The first subtraction must be done with the **SUB** instruction and started with the byte, Word or DWord that has the **lowest** value.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**

Subtract 32-Bit value:



```
SUB.W Var1+0,AX  // 32 Bit values in DX, AX are subtracted from Var1+0
SBC.W Var1+2,DX  // Subtract Var1+2(Var1+0 = lowest word)
SUB.AX 00020     // Is subtracted from 32-Bit value DX,AX.
SBC.DX 00000     // If CF, then DX=DX-1
SBC.D var32, 0000000003
```

3.3.4.107 NEG

The operand is replaced with its two's complement.

Operand = **-operand**

E.g.: The operand is converted from a negative to a positive binary number (or reverse).

This instruction can be used with **Byte, Word** and **DWord** operands. The NEG instruction cannot be cascaded, which means that with multi byte or word numbers, the instruction must be used for the section with the lowest value and the remainder must be converted to 0 with a subtraction.

Application

Conversion of negative operands before **MUL / DIV** since this instruction can only calculate positive numbers.

Changes in the **FL** register: **CF, ZF, SF, PF, OF, AF**

The 32-bit number in **Var1+0** and **Var1+2** should be converted to a positive number if it is negative.



```
TEST.D Var1+2,16#80000000  // test, whether the highest bit=1 (-)
JE NIX                     // don't convert: is already +!
NEG.W Var1+0               // Low value 16 Bit with NEG
L.AX 00000                 // Higher value 16 converted to
SBC.AX Var1+2              // 0 through subtraction
S.AX Var1+2
NIX -
```

3.3.4.108 MUL

This instruction can be used to **multiply 2** unsigned **Byte, Word** or **Dword** operands. One operand is always the AL/AX/EAX register; the other must be a variable or register.

E.g.: "**MUL 10**" is **NOT Allowed**!

1. Byte Multiplication

The addressed byte is multiplied with the contents of the **AL** register.

The result in stored in the **AX** register.

2. Word Multiplication

The addressed word is multiplied with the contents of the **AX** register.

The result is stored in the register pair **DX:AX**, whereas **DX** contains the higher value section of the result.

3. Dword Multiplication

The addressed **Dword** is multiplied with the contents of the **EAX** register.

The result is stored in the register pair **EDX:EAX,** whereas **EDX** contains the higher value section of the result.

Changes in the **FL** register: **CF, OF, ZF, SF, PF, AF**

If the data width of the result is larger than the operands, the CF and OF flags are set.

Click on Arithmetic instructions, for an overview.

In a 16-bit platform, multiplication with operands less than 16 bits must be done in segments and the result added (the position for add is important).

GH*IK = H*K + H*I + G*K + G*I

Follow the three Steps below:

```
1.
Double Word Result = Value1 * Value2
L.AX Value1
MUL.W Value2
S.AX % Result+0  // Result Bit 0-15
S.DX % Result+1  // Result Bit 16-31

2.
Word Result = Value1 * 17
L 017
MUL.B Value1
S.AX Result      // Result is 16 Bits wide.

3.
ResultHigh  ResultLow = Value1 * Value2
```

The example above generates a 64-bit result. It should only be used as an example to show how a multiplication operation is cascaded. Make sure that word with the smallest value is stored in lowest memory location as with the result.

```
L.D ResHi, 00000     // Set highest 32 bit in result to 0.
L.AX Val1+0          // Multiply the lower 16 bits of the operands.
MUL.W Val2+0
S.AX ResHi+0         // And put the result in the
S.DX ResLo+2         // lower 32 bits
L.AX Val1+0          // Multiply the lower 16 bits of one operand with
MUL.W Val2+2         // of the other operand
ADD.W ResLo+2,AX     // and add the 16 bits moved to the first
ADC.C ResHi+0,DX     // partial result.
L.AX Val1+2          // The Multiplication is repeated but in the dire
MUL.W Val2+0
ADD.W ResLo+2,AX
ADC.W ResLo+0,DX
ADC.W ResHi+2,00000  // Here, a carry is already possible!
L.AX Val1+2          // Both upper 16 bits of the operands are now
MUL.W Val2+2         // multiplied and added to
ADD.W ResHi+0,AX     // the previous partial results in the proper
```

3.3.4.109 IMUL

This instruction performs a signed multiplication.

The multiplicand can be **8**, **16** or **32** bits long. The carry flag is set if the result is greater than **32 bits**. Both multiplications can be specified as well as the destination for the result.



```
IMUL.BX AX , 00400        // AX * 400 result in BX
IMUL.AX Variable1 , 00005 // Variable1 * 5, result in AX
IMUL.BX BX , 00005        // BX * 5 result in BX
IMUL.CX (ESI) , 00003     // (ESI) * 3 result in CX
```

Multiplication assignment; the same as **MUL**.



```
IMUL BX    // AX * BX result in AX:DX 32 Bit
IMUL CL    // AL * BL result in AX 16 Bit
IMUL EBX   // EAX * EBX result in EDX:EAX 64 Bit
```

3.3.4.1010 DIV

This instruction performs an unsigned division.

The dividend can be either **16** or **32** bits long.

1. 16-Bit Division

The **16-Bit dividend** in the **AX** register is divided with an 8-bit divisor in a register or in an addressed memory location.

The quotient is stored in the **AL** register and the reminder in **AH**.



Caution:

if the quotient is **less than 8**, the function reacts like a **32 bit division**

2. 32-bit division

The 32-bit dividend in **DX:AX** is divided by the **16-bit divisor** in a register or in the addressed memory location.

The quotient is stored in **AX** and the remainder in **DX**.



Caution:

If the quotient is **> 16** bits then a **system interrupt** is generated, which means, either the user has to write a special interrupt program in order to properly evaluate the situation or make sure that the quotient is not bigger than 16 bits before the division is performed.

3. 64-bit division

The **64-bit dividend** in **EDX:EAX** is divided by the **32-bit divisor** in a register or in the addressed memory location. The quotient is put in **EAX** and the remainder in **EDX**.

After the division, the contents of the **FL** register are undefined.



An undesired interrupt can occur only if 16-Bit /16-Bit divisions are performed and DX is set to 0 before the division.
Follow the instructions below:



1. Word uiVar1 / uiVar2 Res

```
L.AX uiVar1   // AX=Dividend
L.DX 00000    // DX=0, then Quotient guaranteed < 16 Bit
DIV.W uiVar2  // AX / uiVar2
S.AX Res      // Save result in Res, DX contains remainder, (e.g. furt
              // for decimal places)
```

2. Double Word im udVar1, uiVar2 / D 1002 udRes

Here, the division is carried out in 2 steps to avoid problems with the interrupt. The process is exactly the same as for division on paper:

You divide the highest part of the dividends by the divisor and then have the highest part of the result and a remainder. Then "hang" the lower part of the dividend on the remainder and divide by the divisor again. The result is the lower part of the quotient.

```
L.DX 00000
L.AX udVar1+2   // Higher 16 bits of the dividend
DIV.W uiVar1    // by Divisor
S.AX udRes+2    // = higher 16 bit quotient remainder in DX
L.AX udVar1+0   // "Hand" lower 16 bits of dividend on remainder in DX
DIV.W uiVar2    // by Divisor
S.AX udRes+0    // = Lower 16 bit quotient
```

3.3.4.1011 IDIV

Signed division, otherwise the same as DIV.

3.3.4.1012 MOD

This instruction is used to determine the whole number remainder of a division. The data length depends on which operands were used.

3.3.4.11 Status register manipulation

STC	Set carry
CLC	Clear carry
CMC	Invert carry
CLI	Clear interrupts
STI	Set interrupts
STD	Set direction upward
CLAD	Clear direction flag

3.3.4.111 STC

Sets the **CF**-Flag in the **FL**-Register.

Application

Returns **Yes/No** information of a subroutine.



```
STC
RCL.AX 001  // AX is displaced to the left one time and bit 0 = 1
```

3.3.4.112 CLC

Clears the **CF** flag in the **FL** register.

Application

Returns **Yes/No** information of a subroutine.



```
CLC
RCL.AX 001  // AX is displaced to the left one time and bit 0 = 0
```

3.3.4.113 CMC

CMC inverts the carry flag setting. Other flags are not affected.

3.3.4.114 CLI

This instruction hinders interrupts from stopping the program.

This instruction is used during block transfers in the PLC if **DF** is set (downward copy; see **MOVSB, MOVSW**). Or when parameter transfer to an interrupt controlled subroutine should occur (to avoid read errors with simultaneous access of multi byte variables).



Attention!

As a rule, this instruction should be used with caution and cleared after **1ms** (approx. 500-1000 instructions) with the most current STI. Otherwise operating systems could be blocked!

3.3.4.115 STI

This instruction clears the CLI instructions and releases interrupts.

If user-created interrupt controlled program is written in the application program, the STI instruction should be set at the beginning of such a program since the processor blocks other interrupts that can influence operating system functions when running an interrupt program.

```
STI();
```

3.3.4.116 STD

This instruction makes all instructions that automatically change registers **ESI** and **EDI** (**LODS, STOS, CMPS, SCAS, MOVS**) decrease instead of increase.

DF=1

This instruction must be cleared immediately after the desired operation (with **CLAD**).

In addition, all interrupts must also be blocked using CLI while **DF=1**.

3.3.4.117 CLD

This instruction sets **DF to 0** and clears the STD instruction.

3.3.4.12 Special instructions

-	No Function
ENTER	Reserved variables
LEAVE	Free, reserved variables
CBW	Signed expansion of AL register
CWD	Signed expansion of AX register
XLAT	Converted value in AL register
AAA	ASCII comparison for addition
AAS	ASCII comparison for subtraction
AAM	ASCII comparison for multiplication
AAD	ASCII comparison for division
DAA	Decimal comparison for Addition
DAS	Decimal comparison for subtraction
BOUND	Data area Index Test – upper and lower limit
BSF	Search bit forward
BSR	Reserved bit search
BT	Test bit
BC	Test bit and create opposite
BR	Test bit and reset

BS	Test bit and set
CWDE	Convert AX with sign
CDQ	Convert EAX with sign to EDX:EAX

3.3.4.121 - (NOP)

NOP is a single byte instruction that performs no operation. This instruction has no influence on the function of a program of computer. It serves only to change the **EIP** register.

3.3.4.122 ENTER

This instruction stores local variable to the stack and is used mainly by C-routines but is also used in assembler applications when, in a subroutine, the register is to small for buffering. ENTER saves the **BP** to the stack and then produces a number of local variables which can be addressed starting with (BP-002).

The instruction is programmed as follows:

```
PSH EBP
L.EBP ESP
SUB.ESP 00020 // 20 bytes from local Variables
```

It is the same as:

```
ENTER 20
```

The subroutine now contains the available variables (**EBP-001**) - (**EBP-020**).



Tip:

A **local variable** should be created with **VAR/END_VAR** instead of **ENTER**. (Uses symbolic names).

3.3.4.123 LEAVE

Repairs the stack pointer after an **ENTER** instruction.

The instruction **LEAVE** is actually the short form of syntax:

```
ADD.ESP 20 // after ENTER 20
POP EBP
```

The same as:

```
LEAVE
```



(ENTER,LEAVE)

```
//
// -- UPR TEST --
//
TEST ENTER 002   // Reserves 2 LOCAL VARIABLES
L.AX 00003
MUL BX
S.AX (EBP-002)   // Remember result
MUL CX
DIV.W (EBP-002)
LEAVE
RET              // Repair
```

Tip:



Use **VAR / END_VAR** for creating local variables!

3.3.4.124 CBW

This instruction extends the **AL** register with sign from **AL** to **AH**. For example, if the highest valued bit in **AL** is **0**, **AH** is set to **0** and if it's **1**, then **AH** is set to FFh.



	AH		AL	
	xxxx	xxxx	0111	1111
CBW	0000	0000	0111	1111
	xxxx	xxxx	1000	0000
CBW	1111	1111	1000	0000

3.3.4.125 CWD

This instruction extends the **AX** register sign from **AX** to **DX:AX**. That means, if the highest bit in **AX** is **0**, then **DX** is set to **0** and if it equals **1** then **DX** is set to FFFFh.



```
DX                              AX
X X X X X X X X X X X X X X X X  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
X X X X X X X X X X X X X X X X  0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

3.3.4.126 XLAT

The register **EBX** points to the translation list in the data segment. The value in **AL** is replaced by the corresponding value from the translation list.

```
AL <-- (EBX+AL)
```

Changes in the **FL**-Register: None



```
FUNCTION TAB Values
 10, 15, 20, 25
END_FUNCTION
FUNCTION AWL Transform
L     0
// Index
L.EBX #Values()
// Address length
ADD.EBX sizeof(UDINT)
XLAT
END_FUNCTION
```

Translates the value in **AL** as follows:

Before	After
00	10
01	15
02	20
03	25

3.3.4.127 AAA

AAA should only be used after an **ADD** operation, whose result was already stored as a byte in the **AL** register. The lower part of the **ADD** instruction operands should range from **0** to **9** (**BCD format**). In this case, the **AAA** adjusts the format of the **AL** register so that the result is in decimal format. If a decimal carry is generated, the **AH** register is

incremented and the Auxiliary flag is set to **1**. If no decimal carry has been generated, the carry and auxiliary flag are set to **0** and **AH** remains unchanged. In either case, the upper part of **AL** is set to **0**. To convert **AL** to an **ASCII** result, use the **OR 0** instruction after the **AAA** instruction.

```
IF((AL AND 16#0F) > 9) | (AF = 1)) THEN
AL := AL + 6;
AH := AH + 1;
AF := 1;
END_IF;
AL := AL AND 16#0F;
CF := AF;
```

Changes in the **FL** register: **AF, CF - OF, PF, SF, ZF** undefined

Examples

Before		After	
AH	AL	AH	AL
00	00	00	00
00	06	00	06
01	0E	02	04
00	10	00	00
02	16	02	06
01	1E	02	04
00	2E	01	04

The **AAS** instruction is only to be used after a **SUB** instruction, whose result is stored as a byte in the AL register.

The lower part of the operands for the **SUB** instruction must range from **0** to **9** (**BCD numbers**). If so, **AAS** adjusts the format of the **AL** register so that it contains a decimal number format. If a decimal carry is generated through the subtraction, the **AH** register is incremented and the carry and auxiliary flags are set to **0**. In either case, the upper part of **AL** is set to 0. In order to convert **AL** to an **ASCII** result, use the **OR 0** instruction after the **AAS**.

```
IF(((AL AND 16#0F) > 9) | (AF = 1)) THEN
AL := AL - 6;
AH := AH - 1;
AF := 1;
END_IF;
AL := AL AND 16#0F;
CF := AF
```

Changes in the **FL** register: **AF, CF – OF, PF, SF, ZF** undefined

Before		After	
AH	AL	AH	AL
00	00	00	00
00	06	00	06
00	0E	FF	08
00	10	00	00
00	16	02	06
02	1E	01	08
01	2E	01	00

3.3.4.128 AAM

The **AAM** instruction is used for processing a **MUL** instruction between two unpacked BCD digits, whose result is stored in the **AX** register. Because the result is less than 100, it is stored exclusively in the **AL** register. This instruction unpacks the **AL** result with a division by 10 and saves the quotient (the bit with the highest place value) in the AH register and the remainder (bit with the lowest place value) in the **AH** register.

Changes in the **FL** register: **AF, CF, OF** undefined; **SF, ZF** and **PF** are set depending on the result.

3.3.4.129 AAS

The **AAS** instruction is only to be used after a **SUB** instruction, whose result is stored as a byte in the **AL** register. The lower part of the operands for the **SUB** instruction must range from 0 to 9 (BCD numbers). If so, **AAS** adjusts the format of the **AL** register so that it contains a decimal number format. If a decimal carry is generated through the subtraction, the **AH** register is reduced and the carry and auxiliary flags are set to 0. If no decimal carry is generated **CF** and **AF** are cleared and the upper part of **AL** remains unchanged. In order to convert **AL** to an **ASCII** result, use the **OR 0** instruction after the **AAS** or **AL**, 30H.

Changes in the **FL** register: **OF, SF, ZF, PF** undefined; **AF, CF** are set when a decimal carry exists and cleared if not.

3.3.4.1210 AAD

AAD is used to prepare two decompressed **BCD** numbers (with the low value number in **AL** and the high value number in **AH**) for division. The result is also decompressed.

This results in **AL** containing the value **AL+(10*AH)** and **AH** being set to **0**.

AH contains the corresponding two digit binary number from the original decompressed number.

```
AL := AH * 16#0A + AL;
AH := 0;
```

Changes in the **FL** register: **PF, SF, ZF - AF, CF, OF** undefined.

Before		After	
AH	AL	AH	AL
00	00	00	00
00	06	00	06
01	00	00	0A
01	06	01	00
02	00	01	04
02	06	01	0A
03	00	01	0E

3.3.4.1211 DAA

The **DAA** instruction used only after an **ADD** that leaves a two digit **BCD** number as a byte result in the **AL** register. Two compressed **BCD** numbers should be used as the **ADD** operands. The **DAA** instruction adjusts **AL** so that the proper result appears in the form of a compressed two digit decimal number.

```
IF (AL & 0Fh) > 9 or AF = 1 then
AL = AL + 6
AF = 1
END_IF
IF AL > 9Fh of CF = 1 then
AL = AL + 60h
CF = 1
END_IF
```

Changes in the **FL** register: **AF, CF, PF, SF, ZF - OF** Undefined

Before	After

AH	AL	AH	AL
00	00	00	00
00	06	00	06
01	0E	00	14
00	10	00	10
02	16	00	16
01	1E	00	24
00	20	00	20

3.3.4.1212 DAS

DAS instruction is used only after a subtraction, which leaves a **BCD** number in the **AL** register. Two compressed **BCD** numbers should be used as operands. The **DAS** instruction adjusts **AL** so that the result appears in the form of a two digit decimal number.

```
if (AL & 0Fh) > 9 or AF = 1 then
AL = AL - 6
AF = 1
end_if;
if (AL > 9Fh) or CF = 1 then
AL = AL - 60h
CF = 1
end_if;
```

Changes in the **FL** register: **AF, CF, PF, SF, ZF - OF** Undefined

Before		After	
AH	AL	AH	AL
00	00	00	00
00	06	00	06
01	0E	00	08
00	10	00	10
02	16	00	16
01	1E	00	18
00	20	00	20

3.3.4.1213 BOUND

BOUND ensures that a signed array index exists between the defined upper and lower limits in an area of memory. Each limit is defined with a **WORD** for 16 bit operands or with a **DOUBLE WORD word** for **32 bit operands**. The first operand (a register) must be greater than or equal to the lower limit for the memory area and less that or equal to the upper limit. If the register is not within this range, an interrupt [**Bound Exceed**] occurs.

Only word or double word operands are allowed in this operation.

Changes in the **FL** register: None



```
VAR
Limits : STRUCT
Low    : HINT;
Hi     : HINT;
END_STRUCT
END_VAR
BOUND.BX Limits.Low
// raises the lower limit
BOUND.BX Limits.Hi
// raises the upper limit
```

3.3.4.1214 BSF

BSF searches the bits in the second word or double word operands starting at bit **0**. The **ZF** flag is set if all bits are **0**. If all bits are not 0, the ZF flag is cleared and the destination register is Load.

Only word or double words are allowed:

Changes in the **FL** register: ZF



```
L.W Bits , 16#1080
BSF.BX Bits  // ZF-Flag is cleared and the BX register contains value 7
L.W Bits, 16#0000
BSF.BX Bits  // ZF-Flag is set and the contents of BX is undefined
```

3.3.4.1215 BSR

BSR searches the bits in the second word or double word operands starting with the highest the lowest value bit. The **ZF** flag is set if all bits are **0**. If all bits are not 0, the **ZF** flag is cleared and the destination register is Load.

Only word or double word operands are allowed.

Changes in the **FL** register: **ZF**



```
L.W Bits, 16#1080
BSR.BX Bits // ZF-Flag is cleared and BX register contains value 12
L.W Bits, 16#0000
BSR.BX Bits // ZF-Flag is set and contents of BX are undefined
```

3.3.4.1216 BT

BT stores the value of the base bit (first operand), which is defined by the bit offset (the second operand), in the carry flag.

Only word or double word operands are allowed.

Changes in the **FL** register: **CF**



```
L.W Bits , 16#1080
BT.W Bits, 007  // CF-Flag is set
BT.W Bits, 006  // CF-Flag is cleared
L.BX 00007
BT.W Bits, BX   // CF-Flag is set
```

3.3.4.1217 BC

BC stores the value of the base bit (first operand), which is defined by the bit offset (second operand), in the carry flag and then creates the bits complement.

Only word or double word operands are allowed.

Changes in **FL** register: **CF**



```
L.W Bits , 16#1080
BC.W Bits, 007  // CF-Flag is set
BC.W Bits, 006  // CF-Flag is cleared
L.W 00007
BC.W Bits, BX   // CF-Flag is set
```

3.3.4.1218 BTR

BTR stores the value of the base bit (first operand) that is defined by the bit offset (second operand), in the carry flag and then sets the bit back to **0**.

Only word and double words are allowed.

Changes in the **FL** register: **CF**



```
L.W Bits , 16#1080
BTR.W Bits, 007   // CF-Flag is set
BTR.W Bits, 006   // CF-Flag is cleared
L.BX 00007
BTR.W Bits, BX    // CF-Flag is set
```

3.3.4.1219 BS

BS stores the value of the base bit (first operand) that is defined by the bit offset (second operand), in the carry flag then sets the bit to **1**.

Only words and double words are allowed.

Changes in **FL** register: **CF**



```
L.W Bits , 16#1080
BS.W Bits, 007
// CF-Flag is set
BS.W Bits, 006
// CF-Flag is cleared
L.BX 00007
BS.W Bits, BX
// CF-Flag is set
```

3.3.4.1220 CWDE

Converts a signed **AX** to signed Dword in **EAX**. This is done by transferring the highest valued bit (sign bit) into both high value bytes of **EAX**.



Be careful:

CWDE is not the same as **CWD**. **CWD** uses **DX:AX** as its target and not **EAX**.



```
Highest Values Bytes
EAX                               AX
X X X X X X X X X X X X X X X X   1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
X X X X X X X X X X X X X X X X   0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

3.3.4.1221 CDQ

CDQ converts **EAX** to a signed quad-word in **EDX:EAX**. This is done by transferring the highest value bit from **EAX** (sign bit) to the bits in **EDX**.



	EDX	EAX
	16#xxxxxxxx	16#7FFFFFFF
CBW	16#00000000	16#7FFFFFFF
	16#xxxxxxxx	16#80000000
CBW	16#FFFFFFFF	16#80000000

3.4 Contact Plan (COP) in LASAL CLASS 2

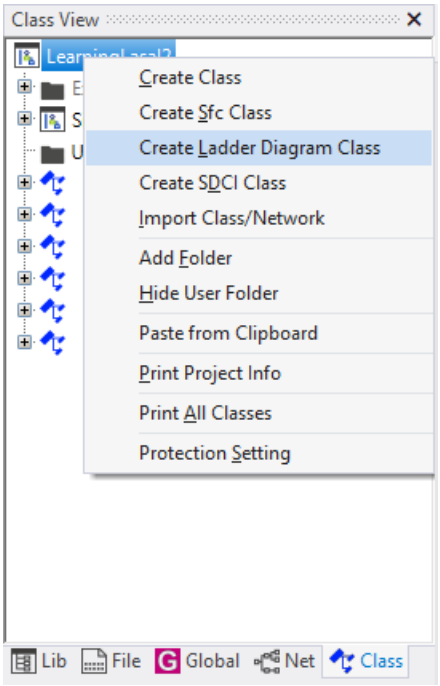
A contact plan (COP), also called a Ladder diagram (LD), is a method for programming for Programmable Logic Controls. It is a standardized graphic programming type similar to a Circuit diagram. The creation of the program with COP is used especially for creating logic controls.

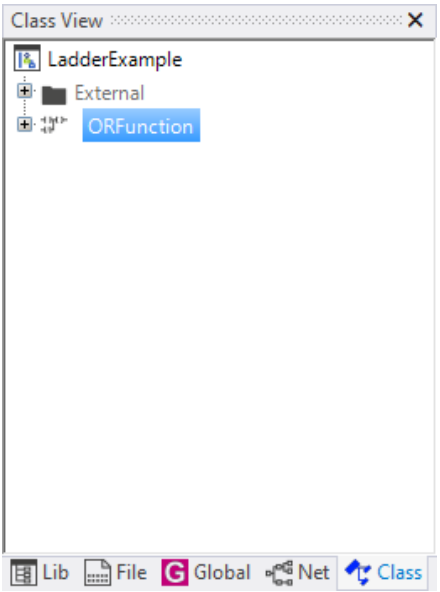
3.4.1 Fundamentals for Creating a Contact Plan



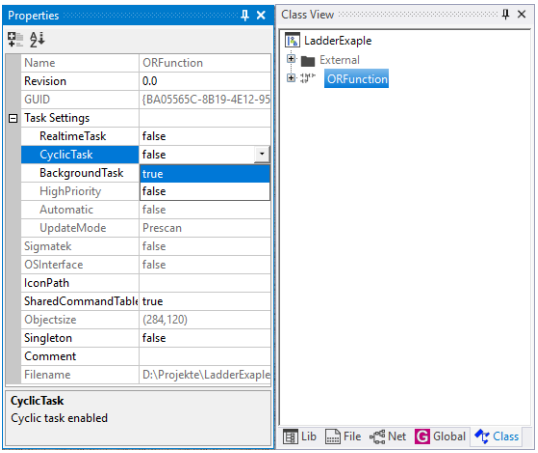
In the next pages, the Steps necessary for creating a contact plan are explained using an example. In the contact plan, 2 inputs and 1 output are place. An OR gate is then placed between the two inputs so that either input 1 OR input 2 must be set in order to set the output.

To begin create a new project (here: **LadderExample**) and in the context menu, select **Create Ladder Diagram Class** to create a ladder diagram class (here: **ORFunction**).


Class View
Create Class
Create Sfc Class
Create Ladder Diagram Class
Create SDCI Class
Import Class/Network
Add Folder
Hide User Folder
Paste from Clipboard
Print Project Info
Print All Classes
Protection Setting
Lib
File
Global
Net
Class

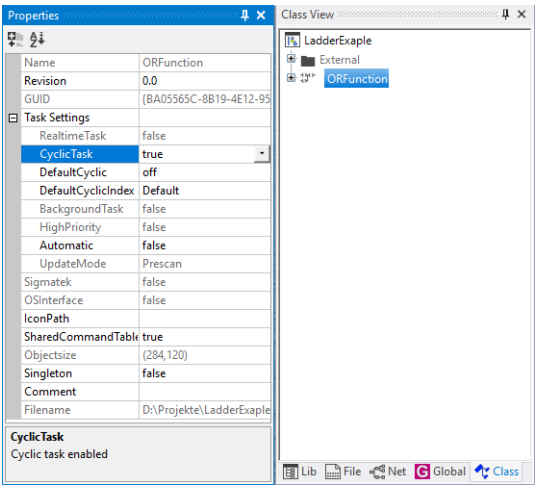

Class View
LadderExample
External
ORFunction
Lib
File
Global
Net
Class

Next select the class and in the **Properties window**, set **CyclicTask** to **True**.

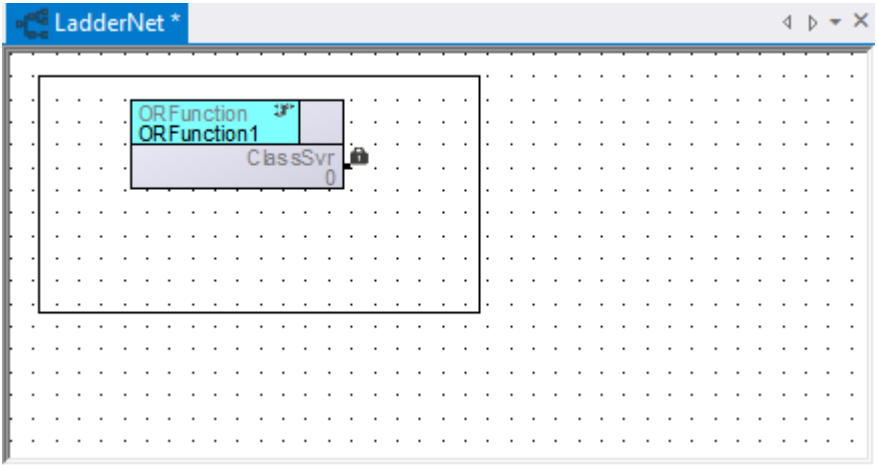




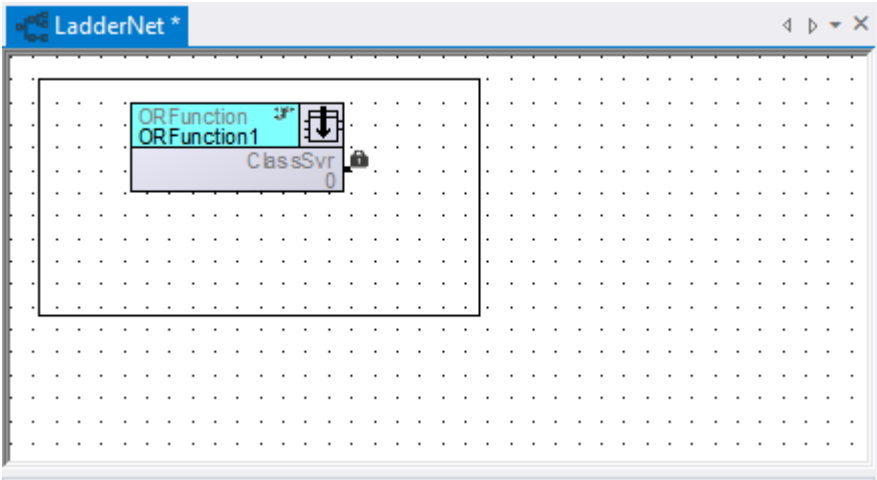
By activating the CyclicTasks, the other two tasks (Realtime and Background) are deactivated automatically and can no longer be selected. Only one task can be selected for a ladder diagram class.



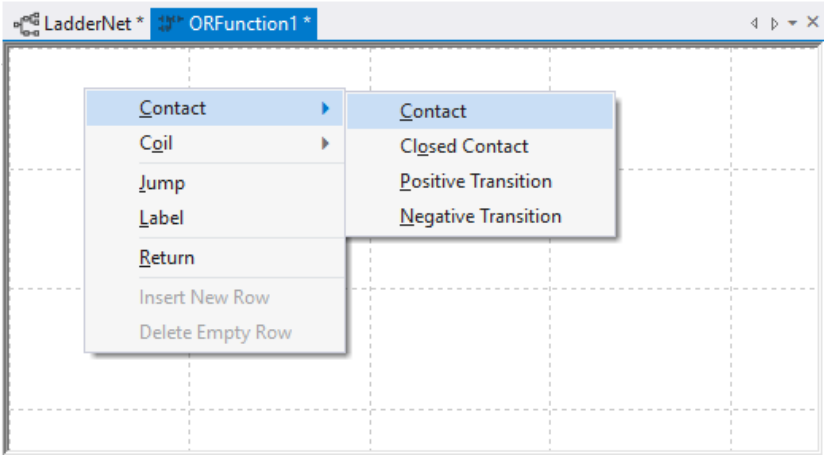
Create a new network (here: **LadderNet**) and place the **ORFunction** class within. Remember to set the cyclic time.



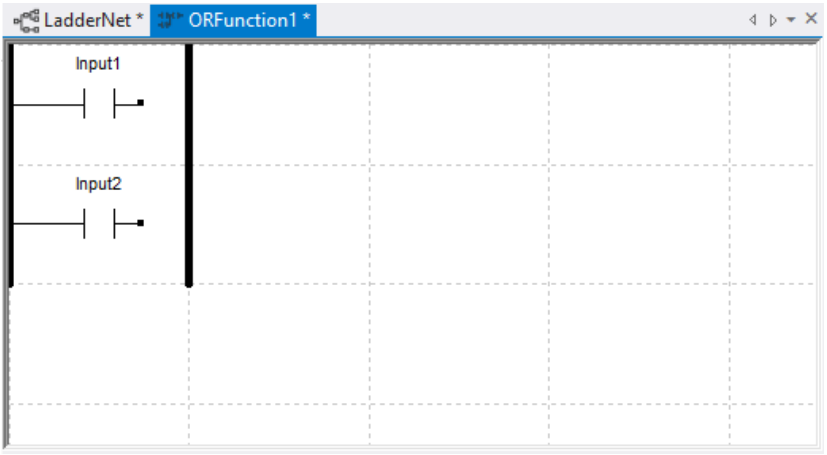
When creating a class, a network (here: **ORFunction1**) is created automatically. Place the mouse on the object to open the network as shown below.



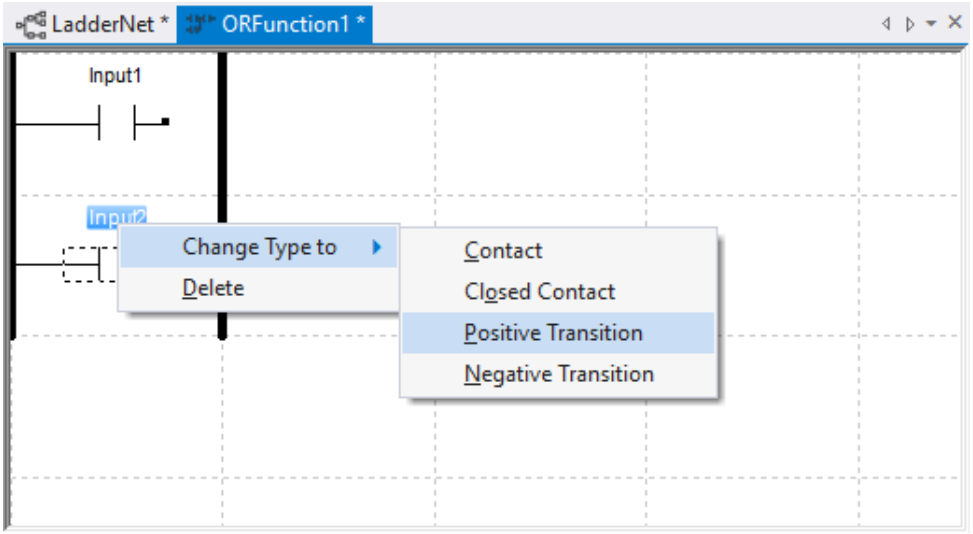
Open the network with a double click. Open the context menu with a right click in the network. Next, select **Contact, Contact** to create a new connection (input).



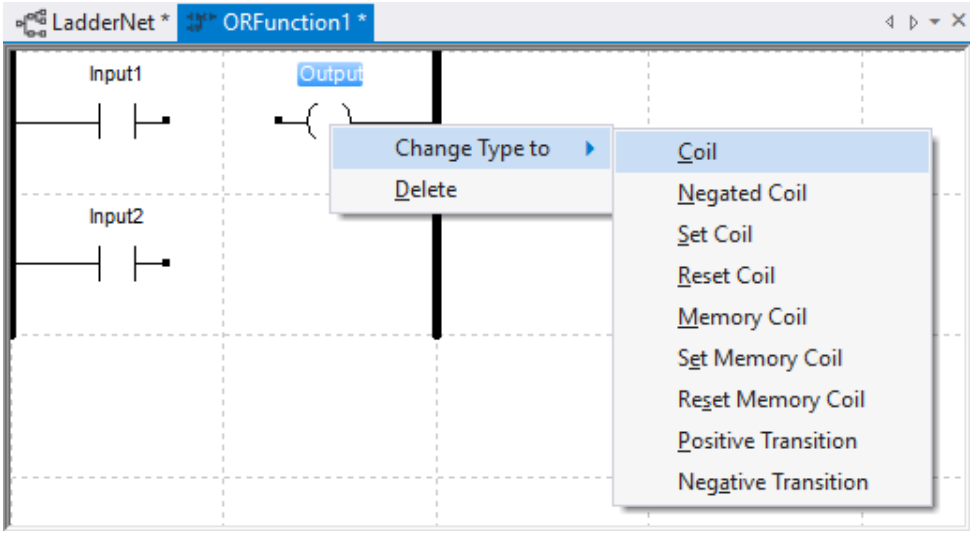
Place a second input in the lower field of the network.



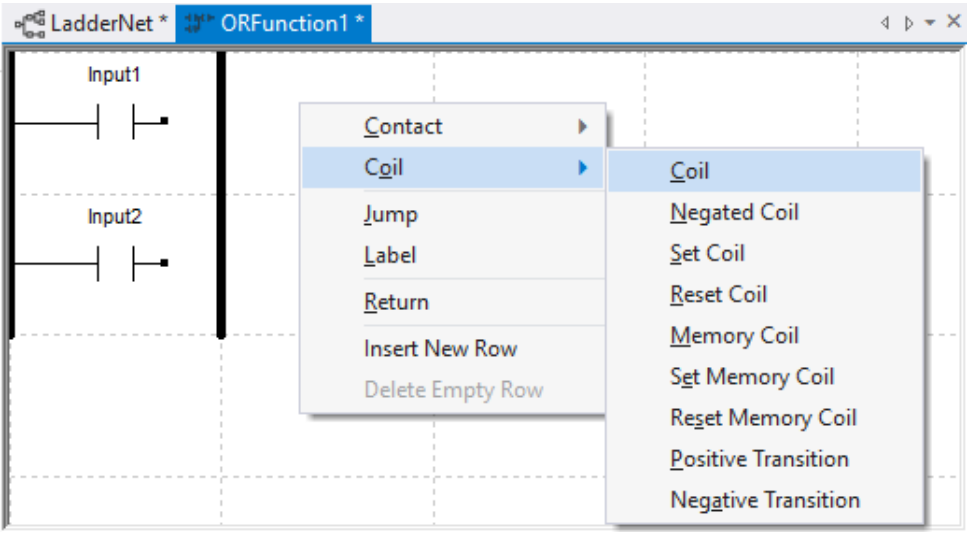
If you now rightclick one of the inputs, a context menu opens, where it can be deleted resp. its input type can **be changed later**.



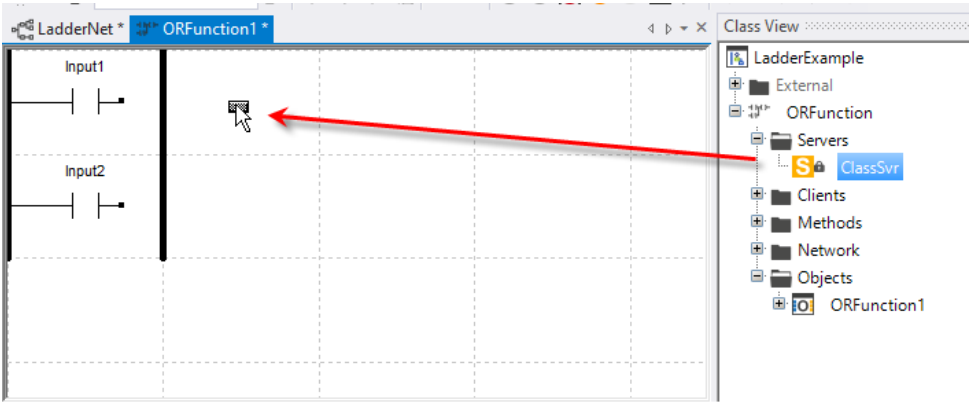
Select **Coil, Coil** from the context menu to place an output.



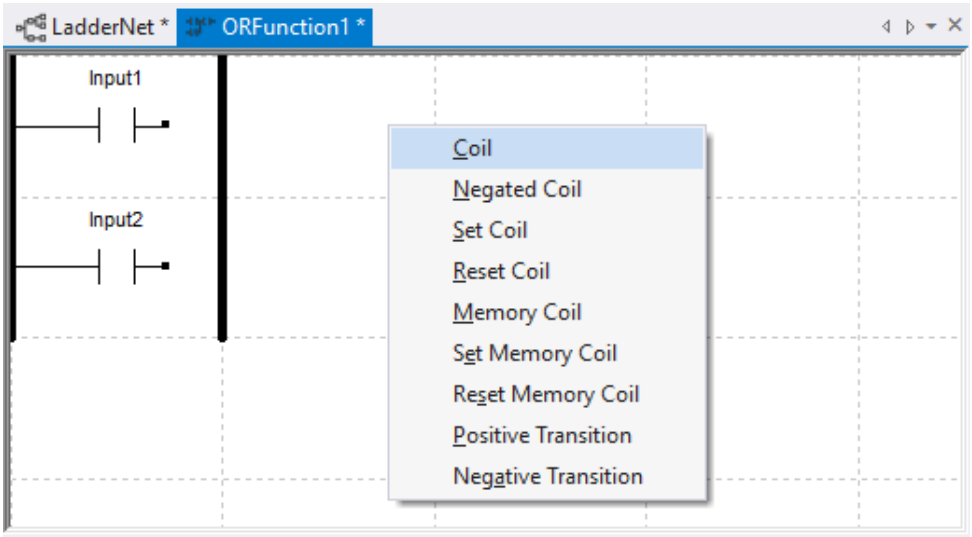
With a right click also the outputs can be **deleted** in the context menu resp. their types can **be changed later**.



Clients / servers can also be placed in the contact plan as in- resp. outputs from the class tree via Drag & Drop ...

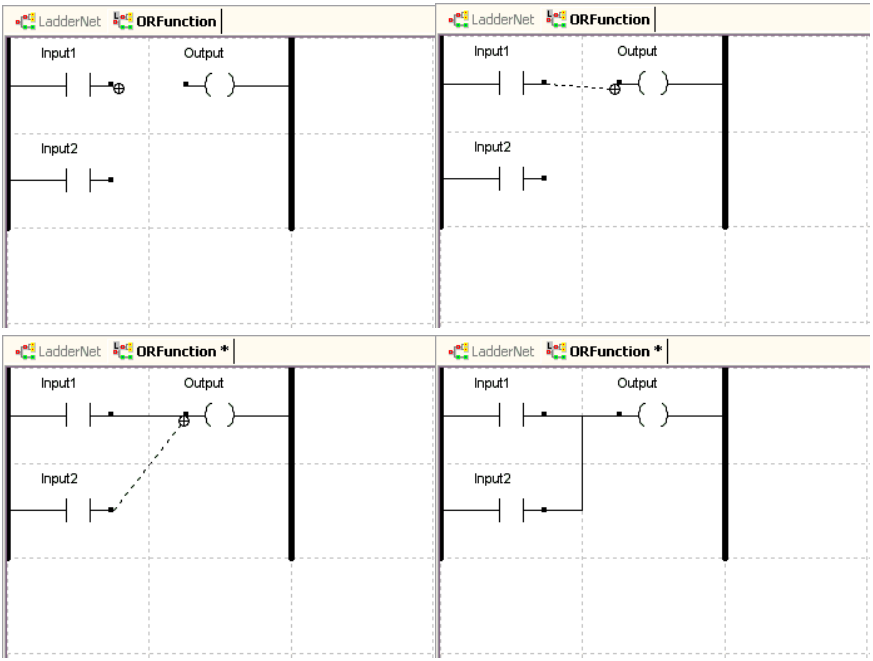


... and their type can be defined, before the procedure is finished.



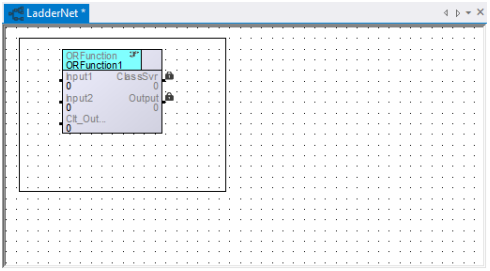
Each in- / output has a property **"Data elements"**. If it is set **true**, the in- / output "goes to" the folder **Variables**. Also from here it can be dragged to the contact plan as a data element. If the property is set to **false** again in the **Properties window**, the in- / output is moved back to the original folder (**Clients/Servers**).

Now connect the elements as shown in the following 4 illustrations.

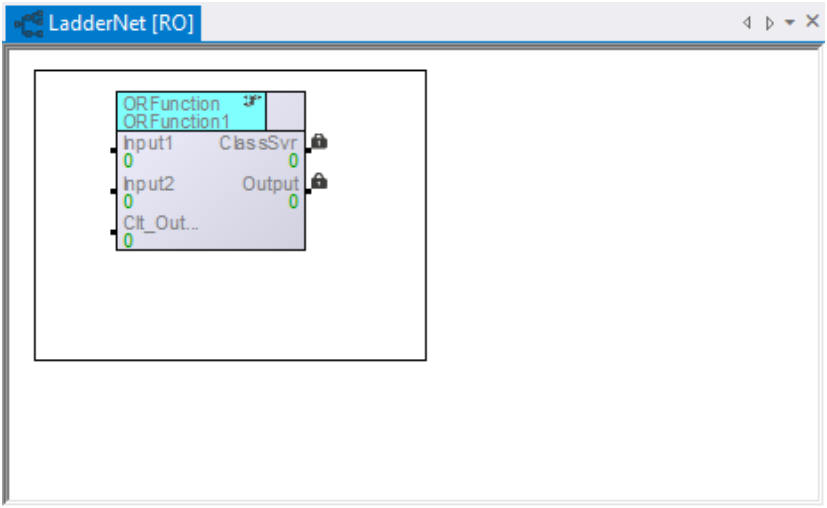


Close the **ORFunction** network. In the **LadderNet** network, the inputs (Clients **Input1** and **Input2**) and output (**Output** Server) are automatically assigned to the **objORFunction** object.

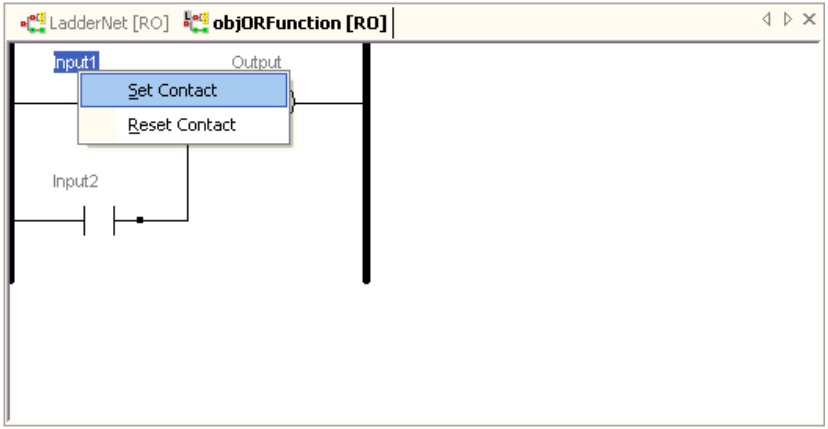
For the output, a client with the called **Clt_Output** is also created. This client serves to connect the output to the hardware.



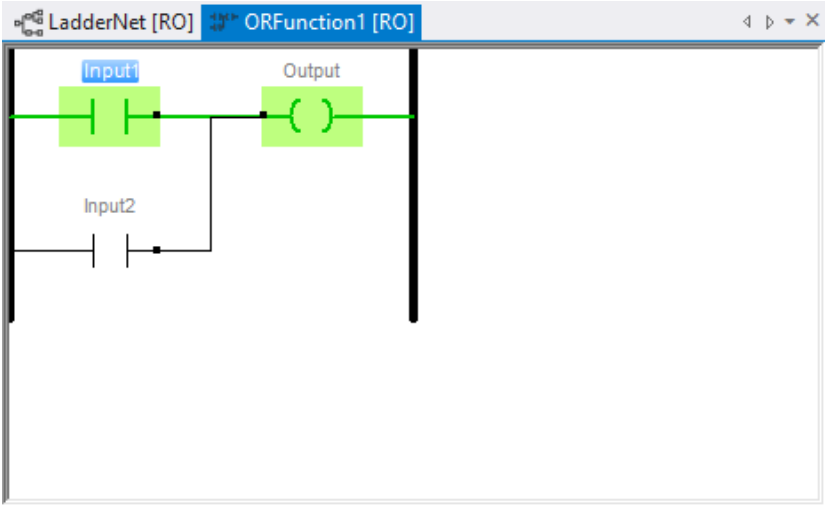
Next, create an online connection and download the project to the target system. Then open the **objORFunction** network with a double click on the arrow in the object (see below).



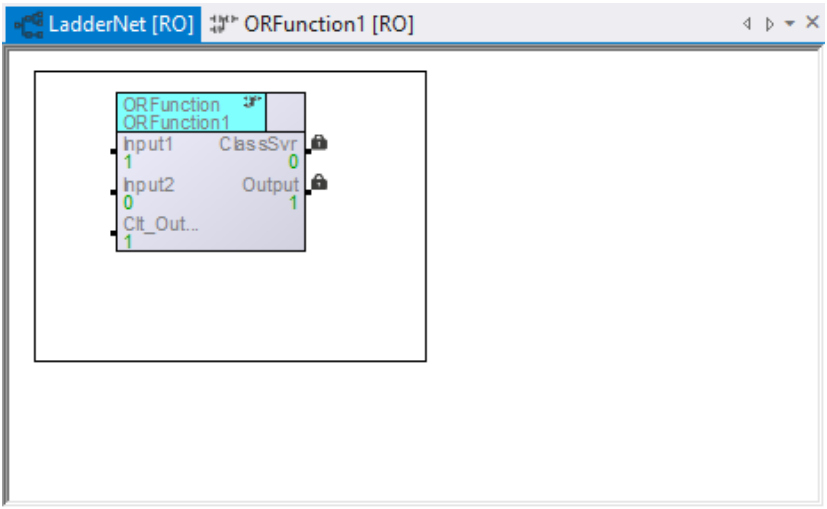
Because of the OR gate, a decision must be made as to which input (**Input1** or **Input2**) to set in order to activate the output. Open the context menu with a right click on one of the inputs (here: **Input1**) and select **Set Contact** to create the contact with the output.



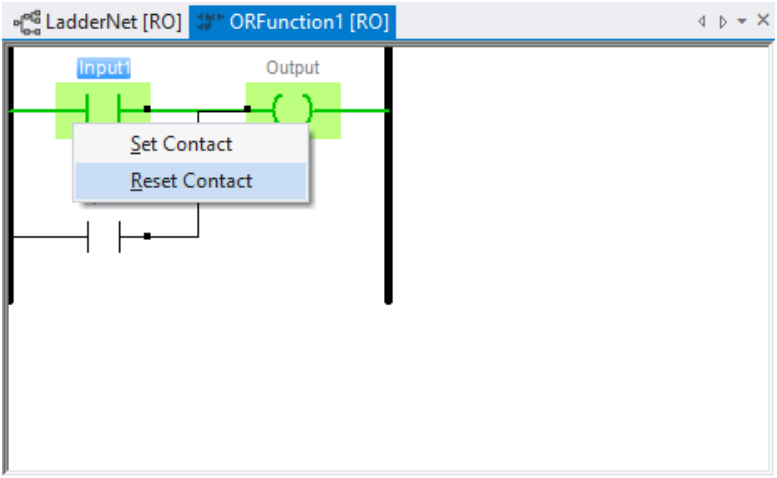
Once contact is made, the color of both the input and output changes to green.



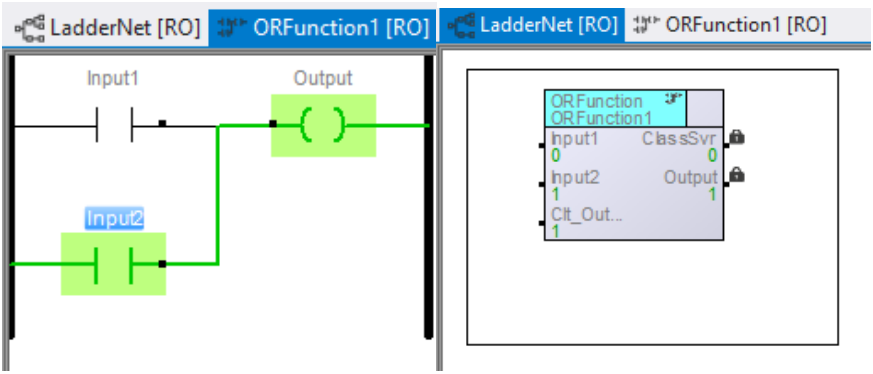
In the **LadderNet** network, the clients **Input1**, **Clt_Output** and the **output** server of the **objORFunction** object are set to **1**.



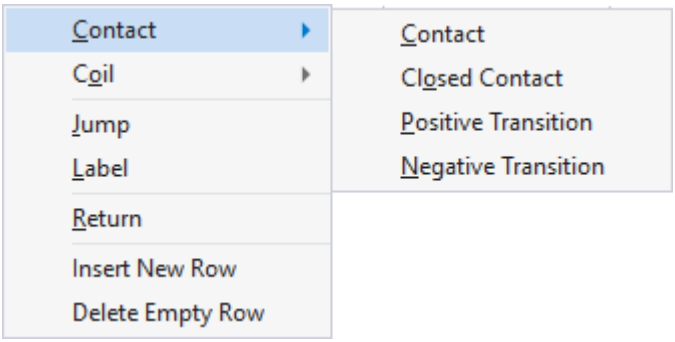
To reopen the contact, select **Reset Contact** in the context menu of **Input1**.



To connect **Input2** with the **Output**, the corresponding input is set to **1** in the **objORFunction** object.

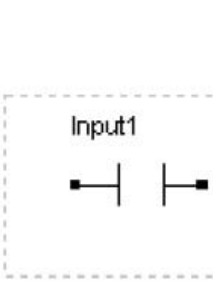


3.4.2 Contact Types



3.4.2.1 Normally Open (Open Contact)

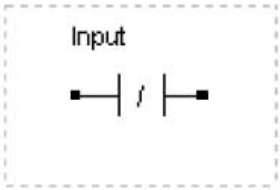
The normally open contact switches the left current path to the right current path as long as it's activated (TRUE, On or 1). When the contact is deactivated, the status of the right current path is 0.



Current path left	Contact	Current path right
1	1	1
1	0	0
0	1	0
0	0	0

3.4.2.2 Normally Closed (Closed Contact)

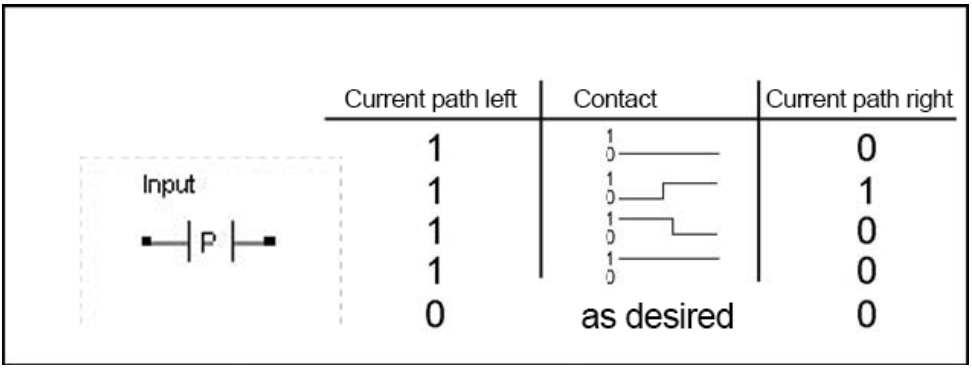
The closed contact switches the left current path to the right current path as long as it is deactivated (FALSE, off or 0). When the contact is activated the no current flows and the right current path is 0.



Current path left	Contact	Current path right
1	1	0
1	0	1
0	1	0
0	0	0

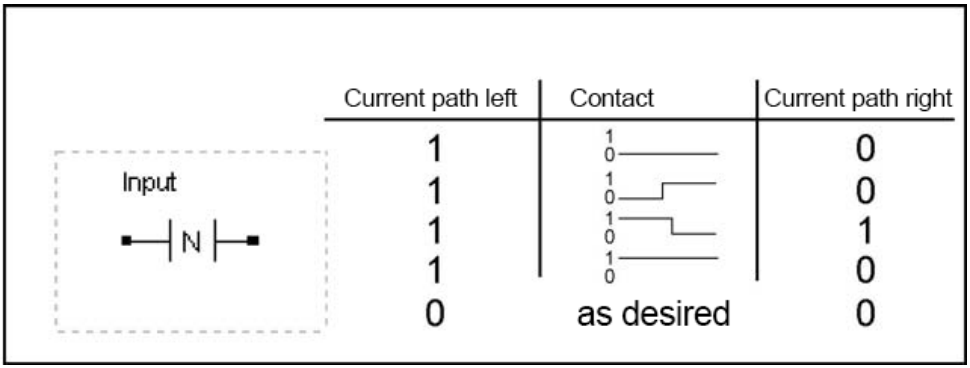
3.4.2.3 Positive Edge Contact

The contact used to recognize a positive flank is 1 if, with the last evaluation of the contact, a signal change from 0 to 1 has occurred. The status of this contact remains the same as long as the object is set for which the contact plan network is used.



3.4.2.4 Negative Edge Contact

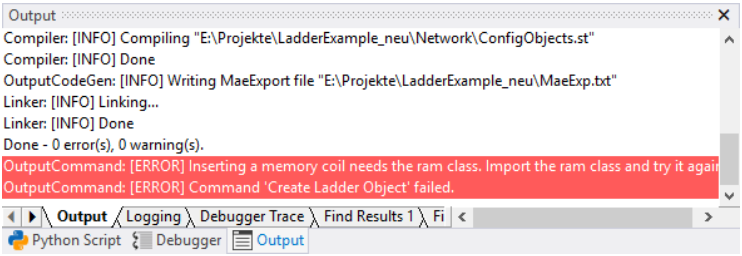
The contact used to recognize a negative flank is 1 if, with the last evaluation of the contact, a signal change from 1 to 0 has occurred. The status of this contact remains the same as long as the object is set for which the contact plan network is used.

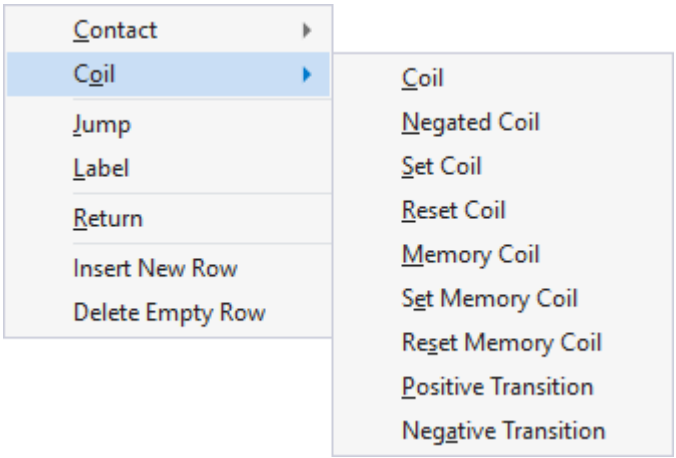


3.4.3 Coil Types

Preliminary Information:

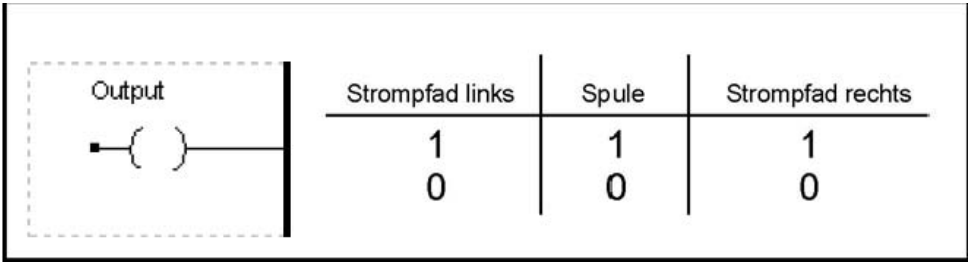
When using the contact plan tools **Memory Coil**, **Set Memory Coil** and **Reset Memory Coil**, the RAM class must be imported from the library. Otherwise the following error message appears in the output window when one of these coils is placed.





3.4.3.1 Coil

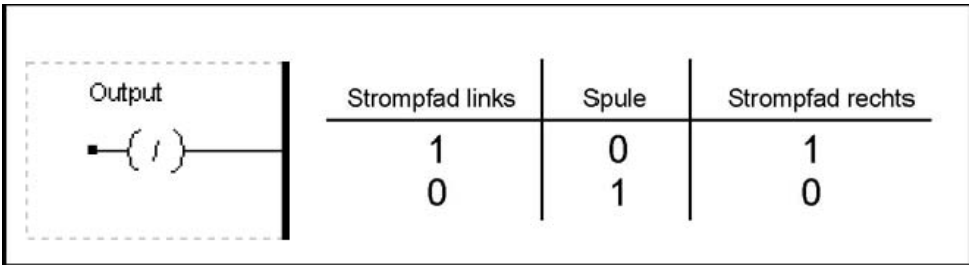
The coil shows a normal output.



Strompfad links	Spule	Strompfad rechts
1	1	1
0	0	0

3.4.3.2 Negated Coil

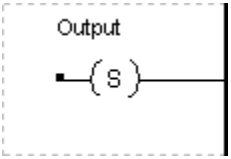
The Negate Coil represents an inverted output.



Strompfad links	Spule	Strompfad rechts
1	0	1
0	1	0

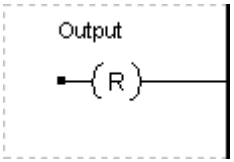
3.4.3.3 Set Coil

This coil is set to 1 when the status of the left current path is 1. When the left current path changes to 0, the value of this coil remains 1.



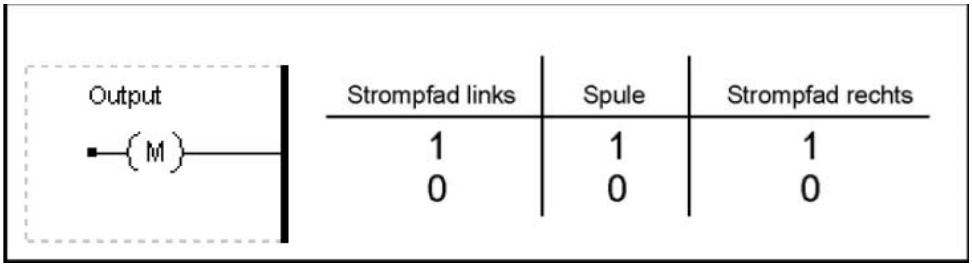
3.4.3.4 Reset Coil

A coil that has been set is reset to 0 following a reset instruction. With the Reset coil, a coil set using the Set coil can be reset to 0.



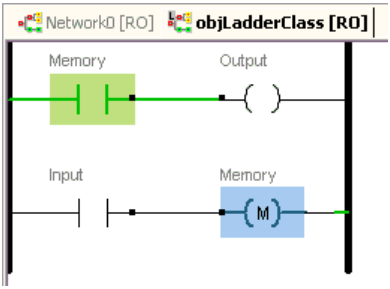
3.4.3.5 Memory Coil

The Memory coil saves the status of the left current path. After the CPU is restarted, the value is retained in the memory coil. This value is stored, however, only for the duration of the cycle time for the object on the memory module.

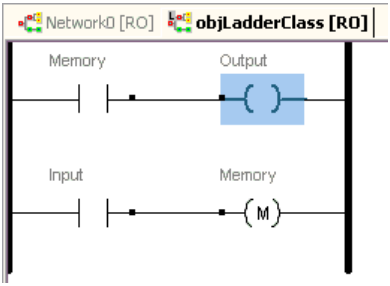


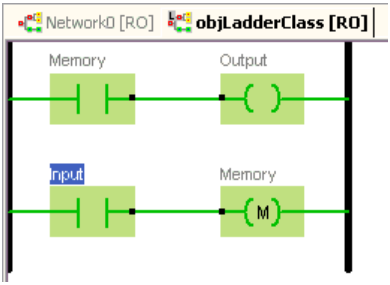
Example application:

In the following screen shots, the signal flow can be seen when the control is restarted after the input have been set. The signals are delayed according to the cycle time of the object.



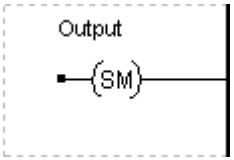
… program restart …





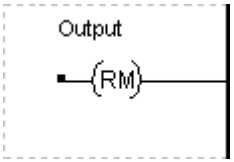
3.4.3.6 Set Memory Coil

This coil is the same as the Set Coil, except this coil retains the status even after the CPU is restarted.



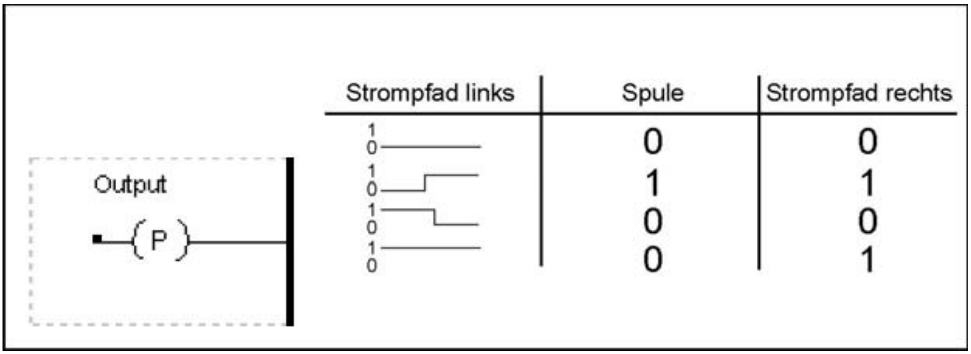
3.4.3.7 Reset Memory Coil

This coil is the same as the Reset Coil, however it also retains the status after a restart.



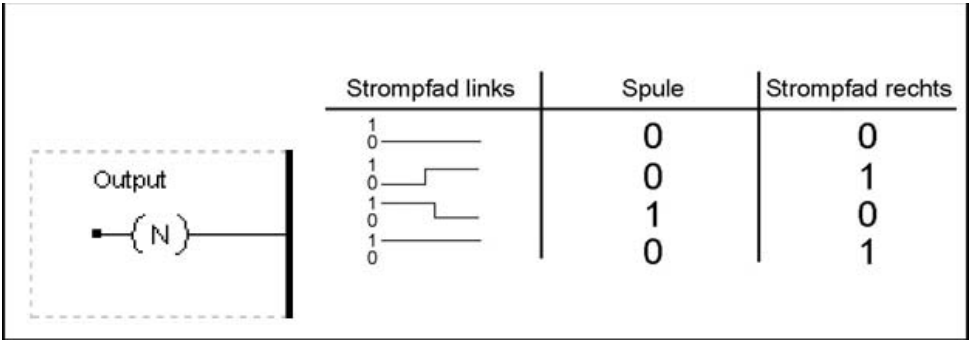
3.4.3.8 Positive Transition

This coil recognizes a positive flank on the left current path and sets the right path to one for a cycle (**Note:** cycle time of object).

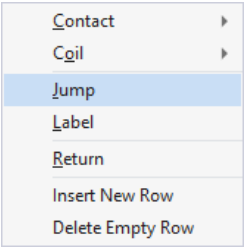


3.4.3.9 Negative Transition

This coil recognizes a negative flank on the left current path and d sets the right path to one for a cycle (**Note:** cycle time of object).



3.4.4 Additional Contact Plan Elements



3.4.4.1 Jump

With the Jump instruction, a conditional jump to a section of the circuit identified by a label. The jump instruction is always drawn toward the right frame.



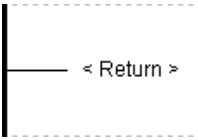
3.4.4.2 Label

If a part of a circuit called from different program location with a jump instruction, it must be identified with a label. The label is always placed in the left frame.



3.4.4.3 Return

Here, the processing of the contact plan is interrupted.



3.4.5 Additional Contact Plan Functions

3.4.5.1 Insert New Row

With this option, a new line can be added to the contact plan.

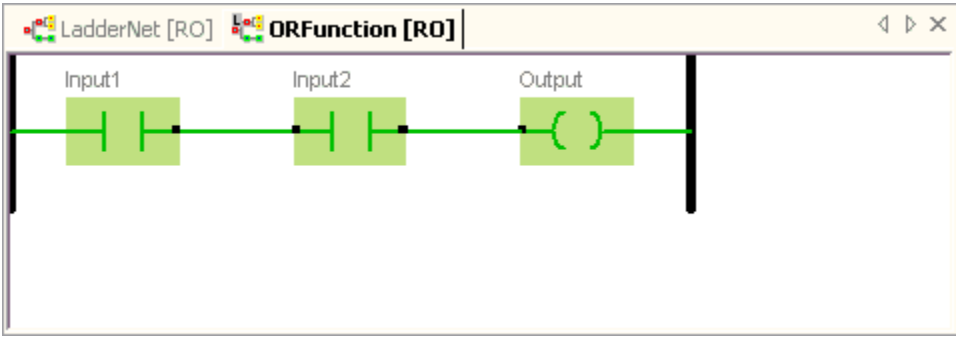
3.4.5.2 Delete Empty Row

This option is used to delete an empty line in the contact plan.

3.4.6 Important Display Modes and/or Settings

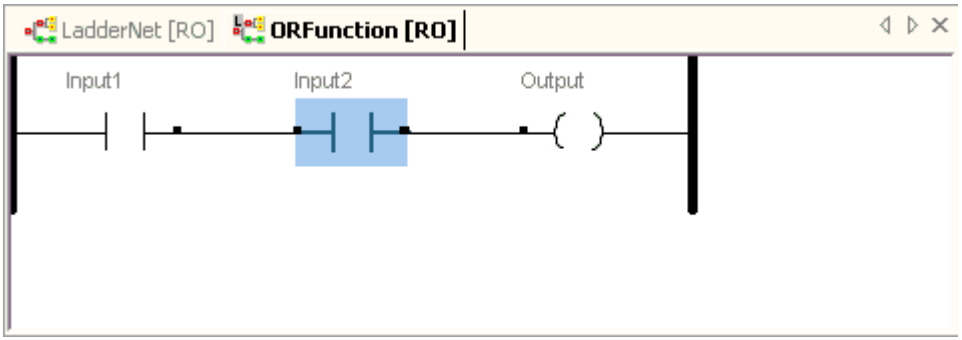
3.4.6.1 Switch Status Display in the Contact Plan

In both inputs (Input1 and Input2) are set as shown in the following screen, the output is also displayed with a green field.



Input and/or output with green background!

The following screen shows input 2 when input 1 is reset to 0. Input2 is shown with a blue background. This means that the input (input2) is still set but, according to the circuit diagram, is no longer supplied with power.

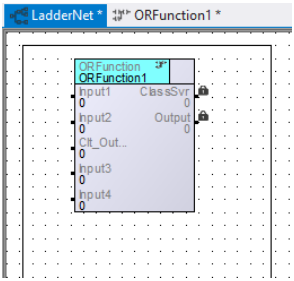


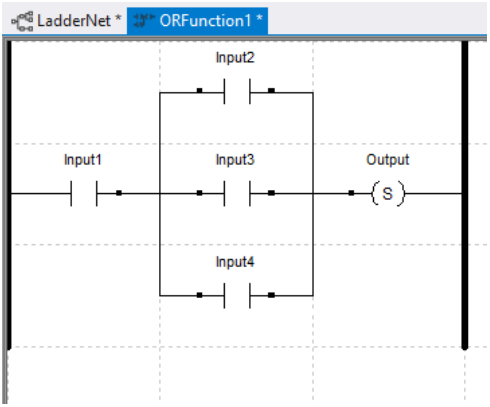
Input with blue background!

This display helps to identify the status of the switches in the contact plan.

3.4.6.2 Class Properties with a Contact Plan

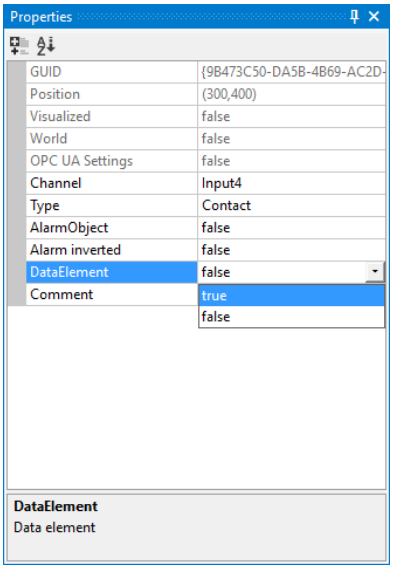
If a class is created with the following contact plan, it will have 5 clients and 2 servers extending outward. To make classes as small as possible (meaning as few clients as possible), also private variables instead of clients and servers can be used for ladder elements.

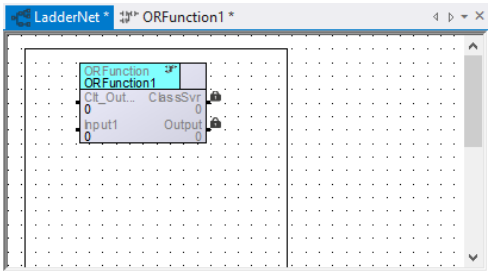




Open the contact plan network and click on the desired input (here: Input2, Input3 and Input4).

In the properties window set DataElement to **True**. The class with the hidden clients is shown below to the right.



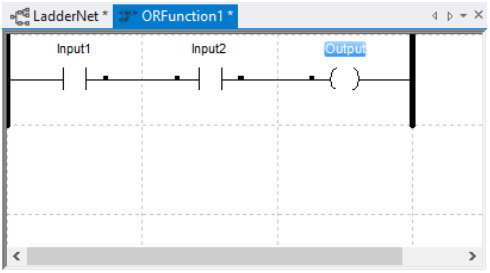


Attention!

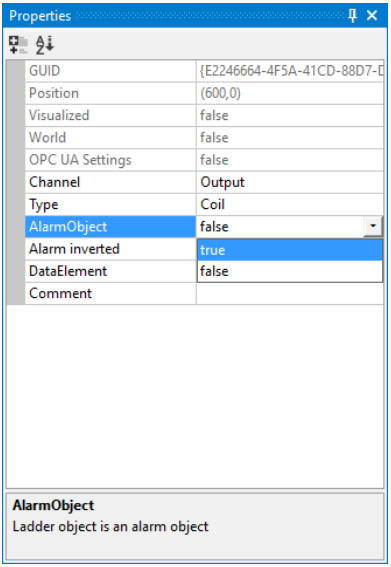
Hiding clients and servers have no effect on the function of the class.

3.4.6.3 Inputs and/or Outputs as Alarm displays

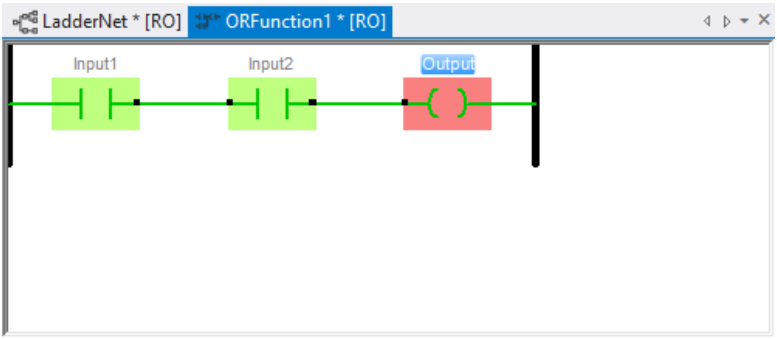
If an input or output should be prominent in the contact plan when active, the **AlarmObject** setting can be changed. The in- or output is then displayed with a red background (in online operation). These alarm objects only serve as support to display and have absolutely no influence on the machine operation.



Click on an output and in the properties window set the **AlarmObject** to **True**. If the output should also be inverted, **Alarm inverted** can be set to **True** in the properties window.



In online operation, once both input1 and input2 are set, the output should now appear with a red background.



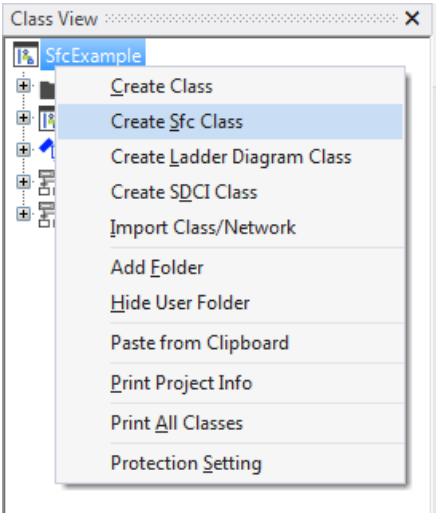
Output with red background!

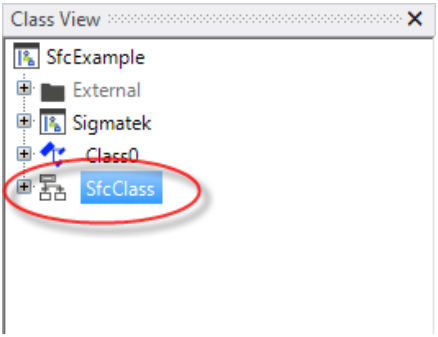
3.5 SFC

Sequential function chart (SFC), is a method for programming Programmable Logic Controls. It is a standardized graphic programming language, which is ideal for creating and maintaining process controls.

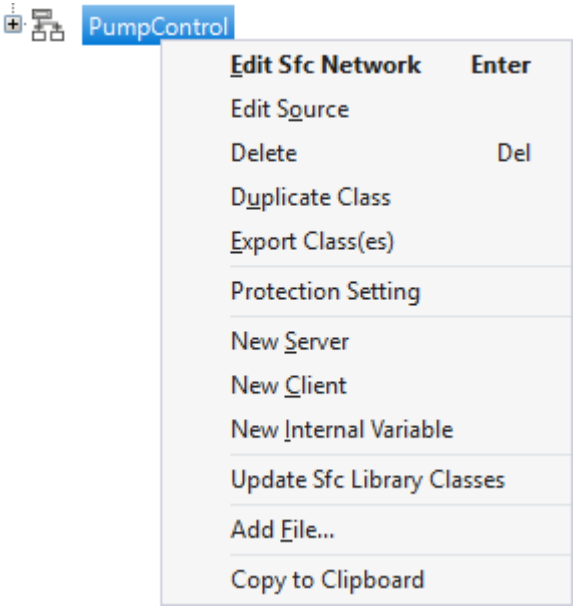
Fundamentals for creating an SFC Diagram

To begin create a new project (here: **SfcExample**) and in the context menu, select **Create Sfc Class** to create an SFC class (here: **SfcClass**).





An SFC class has the following context menu:



Edit SFC Network

Opens (if not already open) or displays the SFC network (if not already open).

Edit Source

Opens the corresponding *.st file for editing.

Delete

Deletes the SFC class.

Duplicate Class

Creates a copy of the SFC class.

Export Class(es)

Exports the SFC class.

Protection Setting

Opens the Protection Configuration dialog. Here, the SFC class can be encrypted, or protected against changes.

New Server

Adds a new server to the class.

New Client

Adds a new server to the class.

New Internal Variable

Adds a new internal variable to the class.

Update Sfc Library Classes

Retrieves the newest version for all SFC classes.

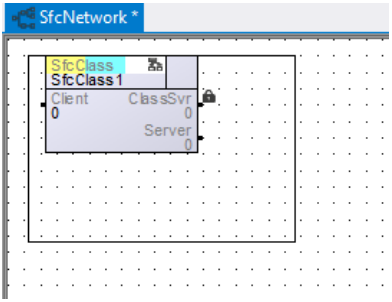
Add File...

Adds a file to the SFC class.

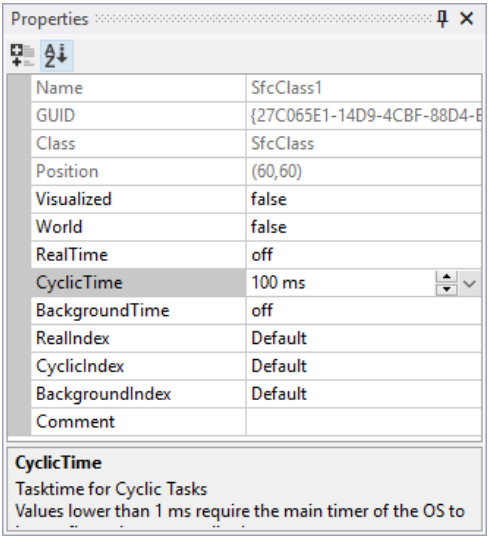
Copy to Clipboard

Copies the SFC class to the clipboard.

Create a network (here: **SfcNetwork**) and place the **SfcClass** in it.



Next, select which task the process control should perform and set the desired cycle time in the placed object.

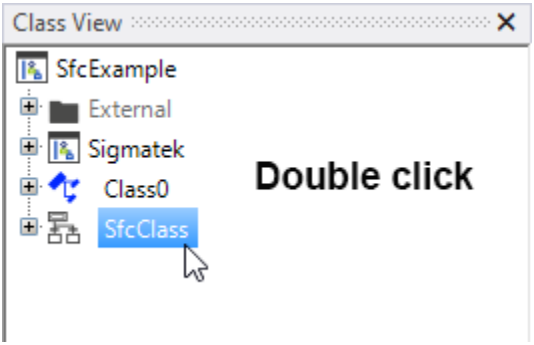




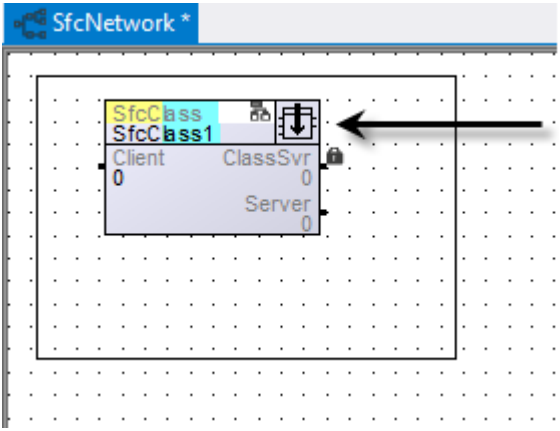
Caution!

When activating a task by entering the cycle time, the other two tasks are automatically disabled (**"off"**). This ensures that process control is performed in only one task.

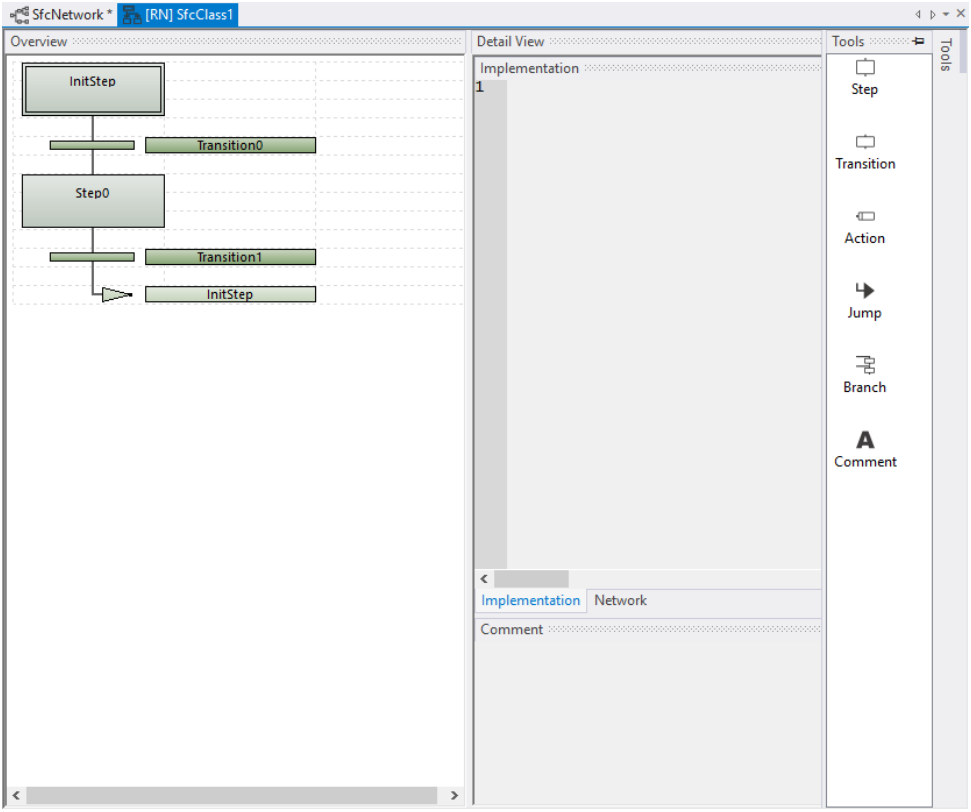
To edit the newly created SFC class, double click on either the SCF class in the tree



... or on the upper right corner of the object placed in the network.



The over**view of the SFC class** then opens. A small, automatically generated process can now be seen in the editor that the user can expand. This process consists of an initialization step ("InitStep"), a **step** ("Step0"), two **transitions** for moving to the next step (Transition0 and Transition1) and a jump that points to the InitStep.



To **move SFC components**, they must be clicked with the left mouse button and be moved without releasing the mouse button. While moving, spaces of the workspace are **colored green** – these are the **possible targets**. If you stay over a possible target with the mouse, the component is displayed dashed like already inserted, so that the future position of the component resp. a possible wrong position can be previewed.

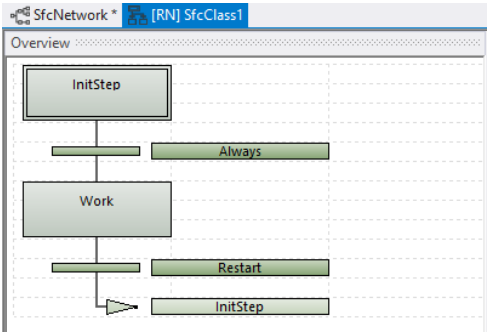


In the SFC Editor, components can only be moved to places marked green (= valid)!

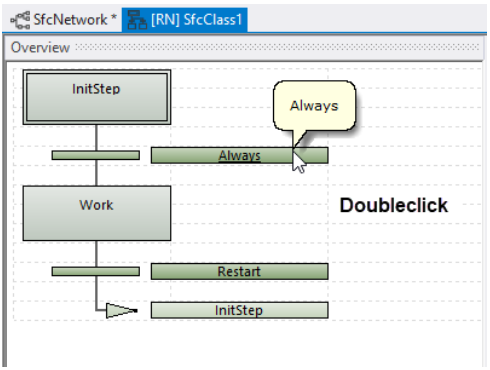
To **copy SFC components**, do the same as for moving, but keep the **CTRL key** pressed while moving.

To **delete SFC components**, they are selected and then deleted by pressing the **Del** key.

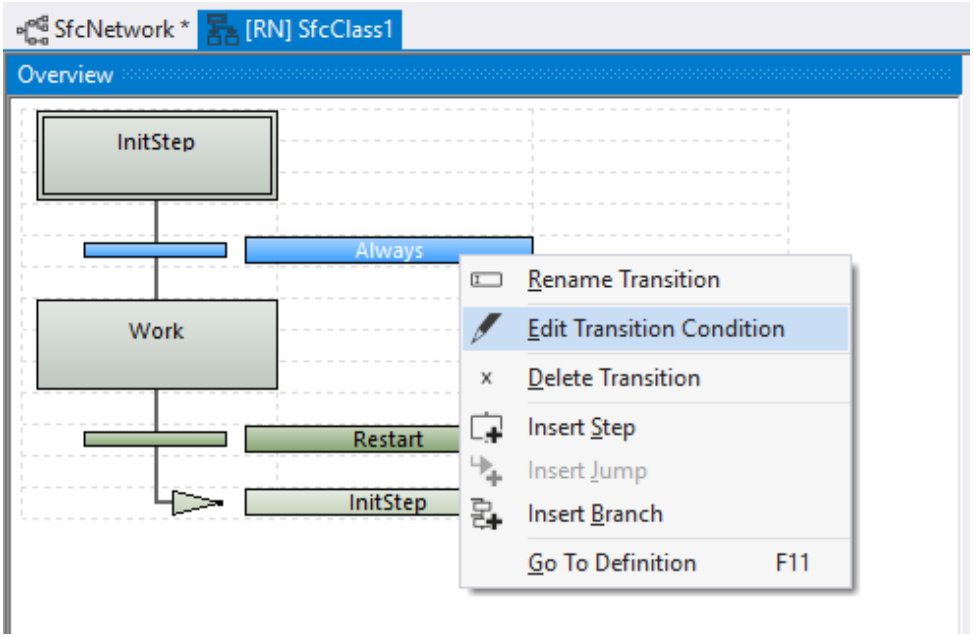
To **rename the Sfc components**, select either the component and press **F2,** click delayed on the name or select **"Rename Transition"** from the context menu. Rename "Transition0" to "Always", "Step0" to "Work" and "Transition1" to "Restart".



New transitions are assigned the default value "**False**". In the generated process, the InitStep is therefore never exited. To edit a transition, double-click on it,...



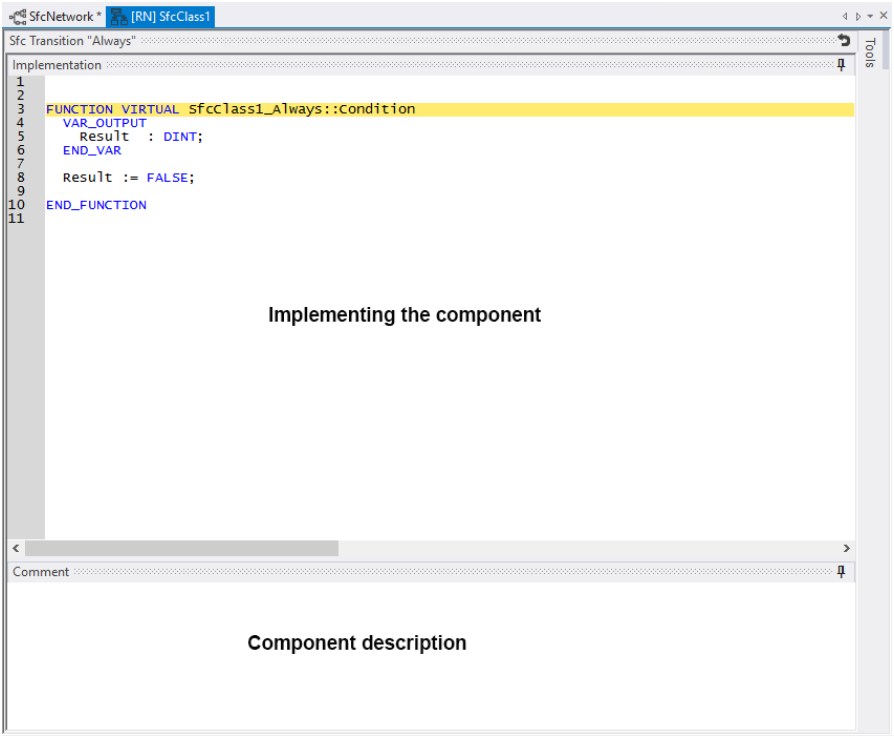
...or select **"Edit Transition Condition"** from the context menu.



Set in **Extras / Options / SFC Options „Always Show SFC Detail View“** to **true**. One click is sufficient to run a transition.

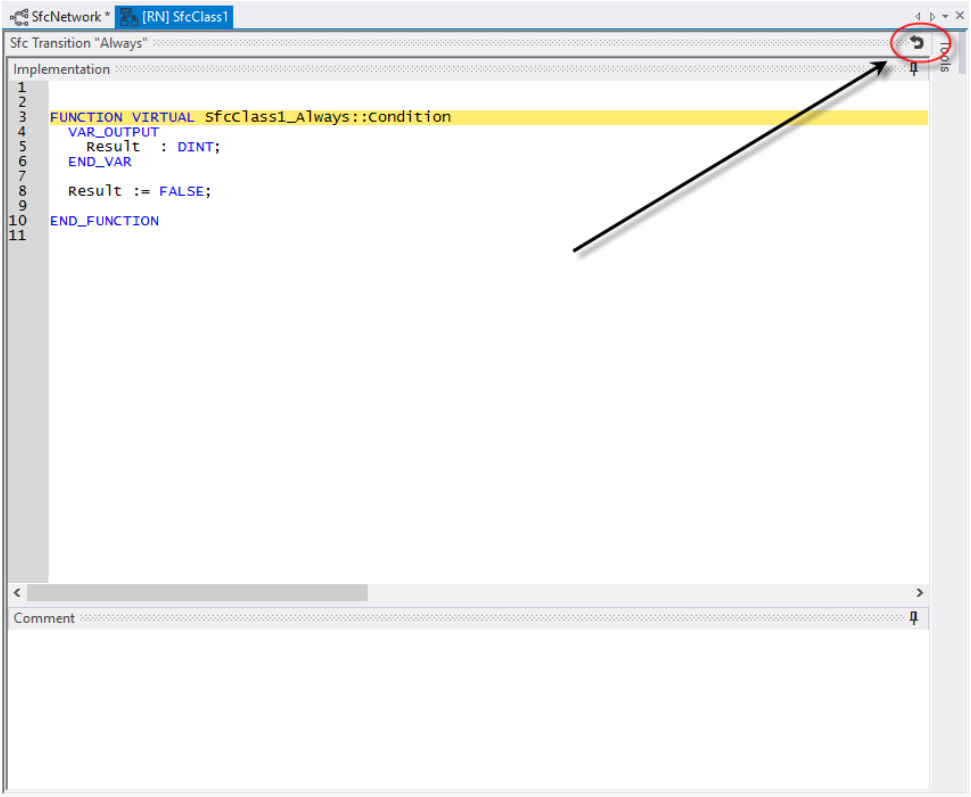
Now edit the transition "Always".

The **Detail view** of the SFC component now opens, in which the **component description**, **implementation for transitions** and **actions** used can be seen.



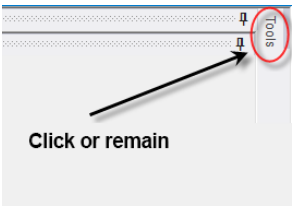
Change the return value "Result" in the Condition method from **"FALSE"** to **"TRUE"**. When the application is started, the InitStep is then active for one cycle and the transition **"Always"** immediately switches to the step **"Work"**.

The Detail view can be exited with click on the arrow in the upper right corner.

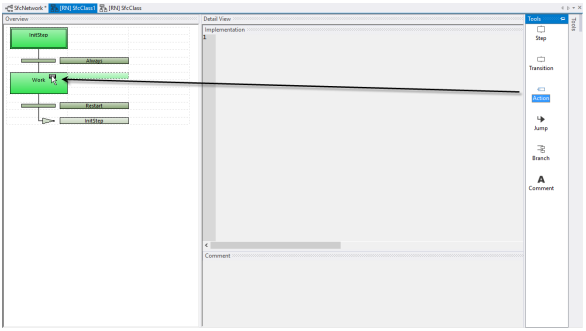


If the "Work" should perform a task, the step must be assigned an Action.

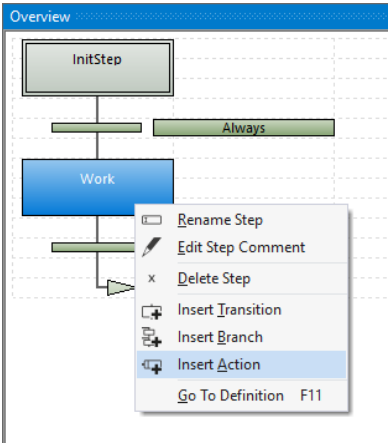
For this either from the SFC Toolbox (click on Tools resp. stay there with the mouse, so that it opens)



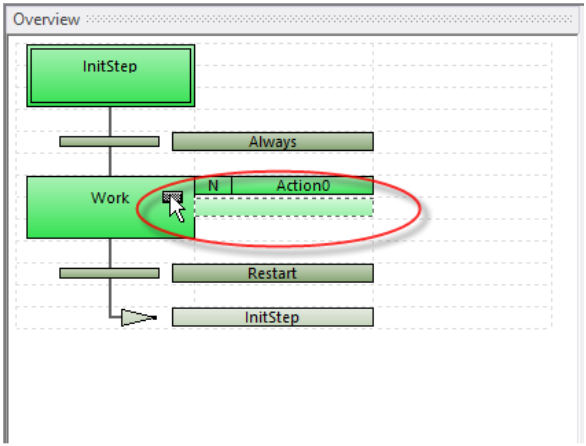
is dragged to the step with Drag & Drop, ...



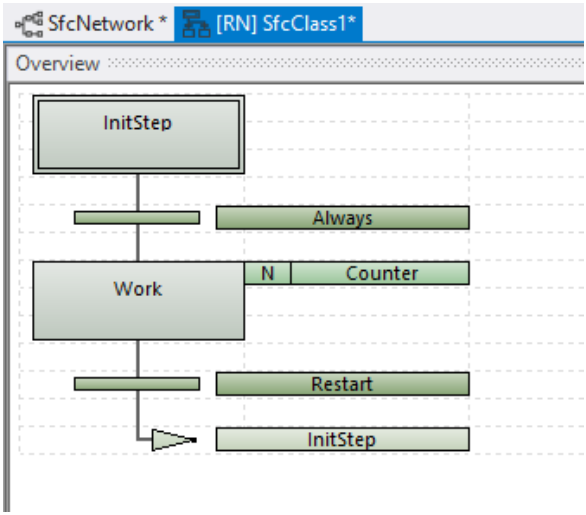
... or select "Insert Action" from the context menu of the step.



If the mouse is held over a step, the future position of the inserted element is shown with dashed contours as a preview. This applies to all inserted elements.

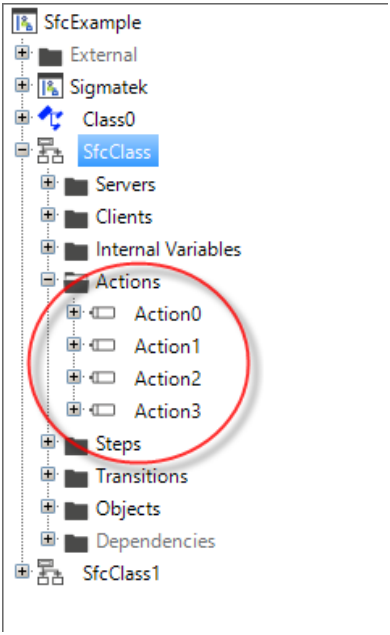


Next, enter the name “Counter” for the new action:

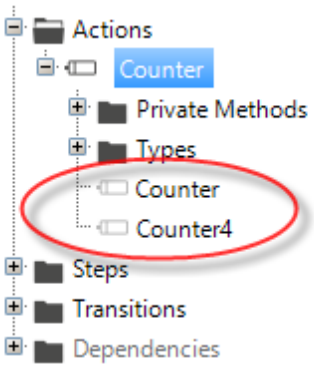


Next to the inserted action on the right, in what is now the third column, the in and outputs can be seen. The corresponding step is automatically added by default as an input.

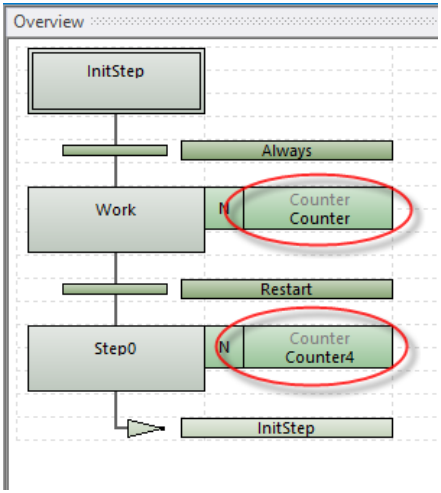
Actions and transitions are creates as “Template” automatically and are therewith reusable. Templates are marked with their own "action icon" in the tree:



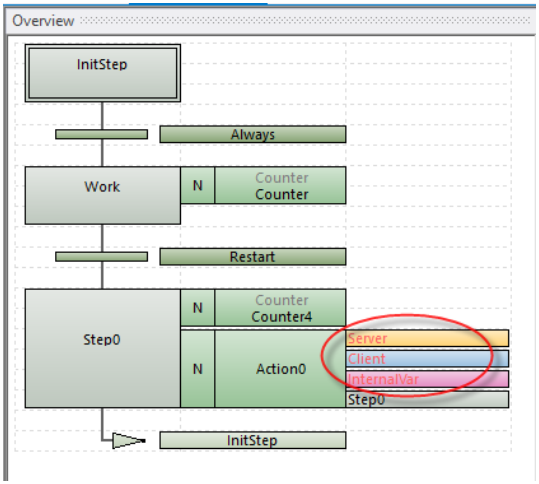
These templates can be placed in the SFC network multiple times. All objects created in this way from the same template are now grouped in the tree under the template, identified by an "action icon" in lighter gray:



In the SFC network, an object whose template was used more than once is shown on two lines. In the upper line is the template name; in the lower line, the object name is shown:



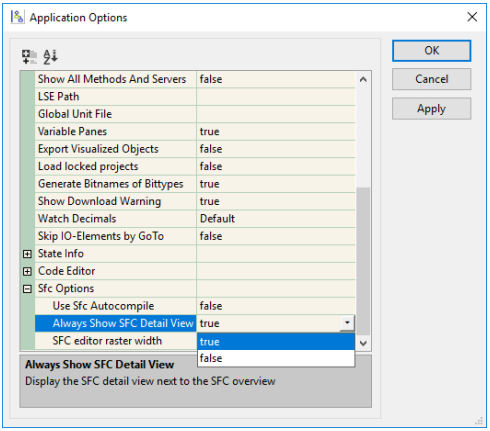
If the multiple use of the same template is not available in different class IO elements, they are marked in red:



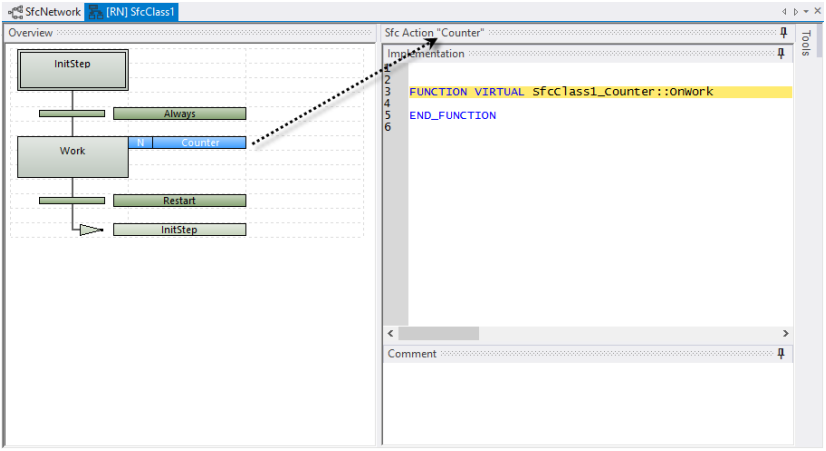
...and the corresponding error message is triggered during compilation:

```
Output
Compiler: [INFO] Compiling "E:\Projekte\SfcExample\Class\SfcClass1_Action0\SfcClass1_Action0.st"
Compiler: [ERROR] E 0029 "E:\Projekte\SfcExample\Class\SfcClass1_Action0\SfcClass1_Action0.st"(170) Class 'SfcClass1' has no element 'Client'
Compiler: [ERROR] E 0029 "E:\Projekte\SfcExample\Class\SfcClass1_Action0\SfcClass1_Action0.st"(171) Class 'SfcClass1' has no element 'Client'
Compiler: [WARN] W 0150 "E:\Projekte\SfcExample\Class\SfcClass1_Action0\SfcClass1_Action0.st"(171) Missing ';' instead of 'Client'
Compiler: [ERROR] E 0029 "E:\Projekte\SfcExample\Class\SfcClass1_Action0\SfcClass1_Action0.st"(176) Class 'SfcClass1' has no element 'Client'
Compiler: [INFO] Done
OutputCodeGen: [INFO] Writing Network Files
OutputCodeGen: [INFO] Done
OutputCodeGen: [INFO] Writing ConfigObjects.st
OutputCodeGen: [INFO] Done
Output I/O Manager: [INFO] Checking IO Connection Manager Entries
Output I/O Manager: [INFO] Done
OutputHelper: [ERROR] Sfc class 'SfcClass1' has no client "Client"!
OutputHelper: [ERROR] Sfc class 'SfcClass1' has no internal variable "InternalVar"!
OutputHelper: [ERROR] Sfc class 'SfcClass1' has no server "Server"!
OutputHelper: [ERROR] Wrong type of Sfc element 'SfcClass1.SfcClass1_Jump0'. The destination of Sfc step 'SfcClass1.Step0' must be a Sfc transition.
OutputCodeGen: [INFO] Writing MaeExport file "E:\Projekte\SfcExample\MaeExp.txt"
Done - 11 error(s), 1 warning(s).
OutputCommand: [ERROR] Command 'Build project' failed.
```

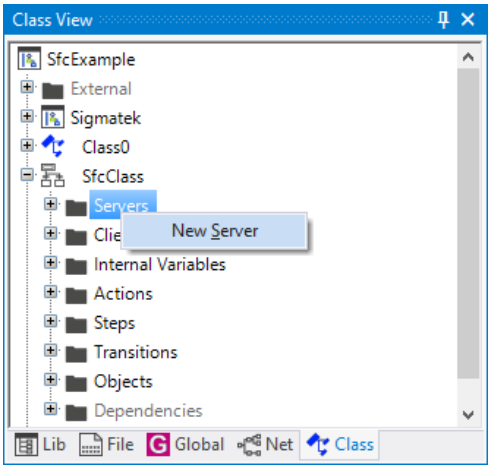
For comfort, here the SFC editor is configured so that the overview and detail view are always displayed simultaneously. For this setting, go to **Extras / Options / SFC-Options** and set **Always Show SFC Detail View** to **true**.



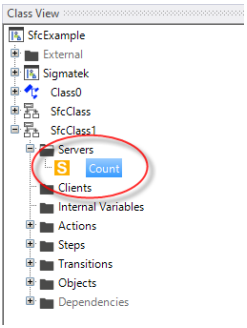
Next open the detail view of the action with a click on the action component in the SFC editor.



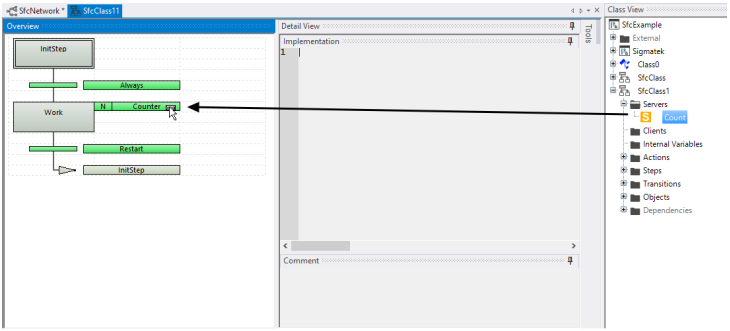
New **actions** have the action qualifier **N** by default, which indicates that the action is called cyclically as along as the step is assigned. To increment a server in the action, a new server must first be added to the SFC class. This must be added to the SFC class in the Class Tree.



Add a new server **"Count"** to the SFC class and remove the write protection.

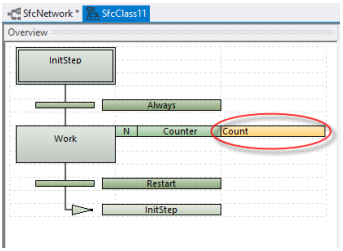


To access the new server in the **"Counter"** action, it must first be added to the action's in/outputs. To add the new server, drag it from the Sfc Class window and drop it in the SFC editor onto the action component.

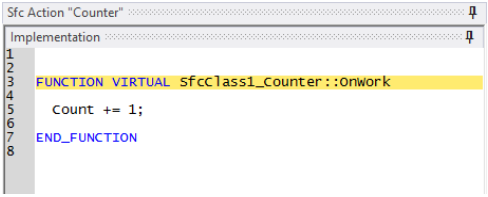


This also works with several servers / clients (multiselection).

The **"Count"** server in the Sfc class can now be fully accessed in the **"Counter"** action.



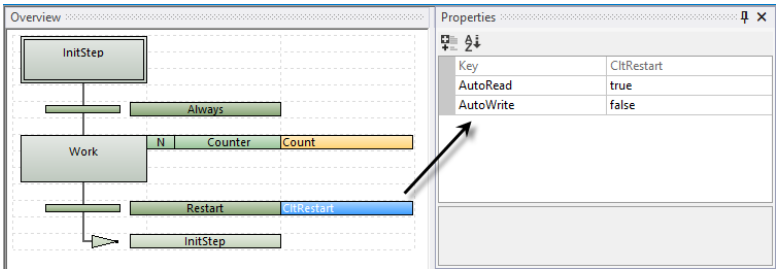
In the **"OnWork" method** in the **"Counter"** action implementation, add the line "`Count += 1;`" to cyclically increase the server value.



Alternatively to the last steps, there is a shorter way: **a client / server can be placed in the implementation directly from the ClassTree**. If the transition or action of the client / server was not yet added as an IO, this is the performed **automatically**.

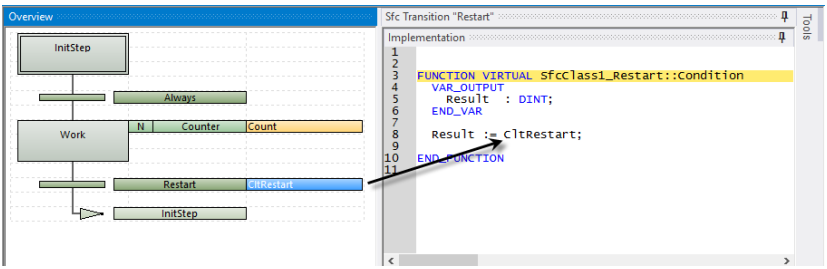
To restart the counter, a condition must be defined for the **"Restart"** transition. First create a new client **"CltRestart"** in the Sfc class, click on the **"Restart"** component in the SFC Editor and assign the new client by Drag & Drop as previously with the server. Next, change the value of the return variable **"Result"** in the "Condition" method of **"Restart"** from FALSE to `CltRestart;`.

With **SFC clients** to which an SFC component is assigned, it is not necessary to call the **Read()** and **Write()** methods. These clients therefore have **changeable Auto Read and Auto Write properties**.

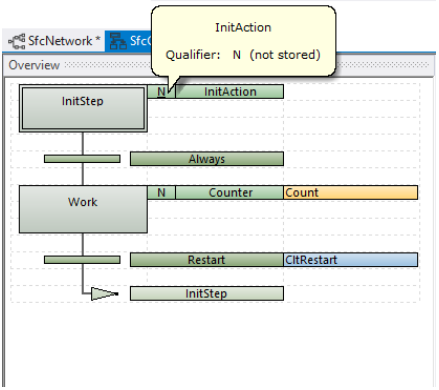


If **Auto Read** is set to **true**, the client is read automatically; if **Auto Write** is set to **true**, it is written automatically. In servers and internal variables, both these properties are permanently set to **true** – these are always read (Read() method call) and written (Write() method call) automatically.

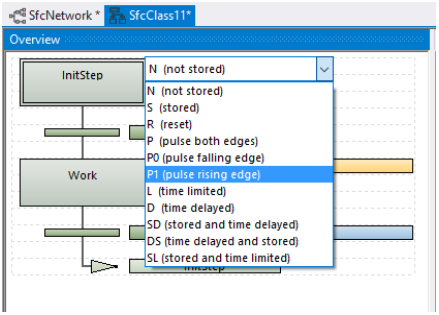
When the client reads the value 1, the transitions moves to the next step and activates the InitStep through the jump.



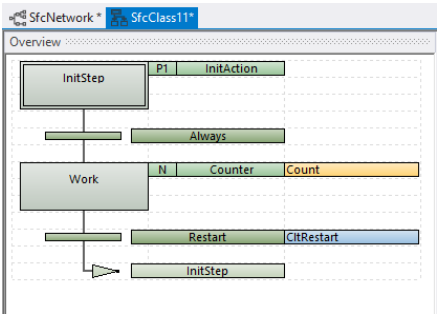
To reset the **Count** server after restarting, a new action **"InitAction"** is added to the InitStep and its action qualifier set to **P1**.



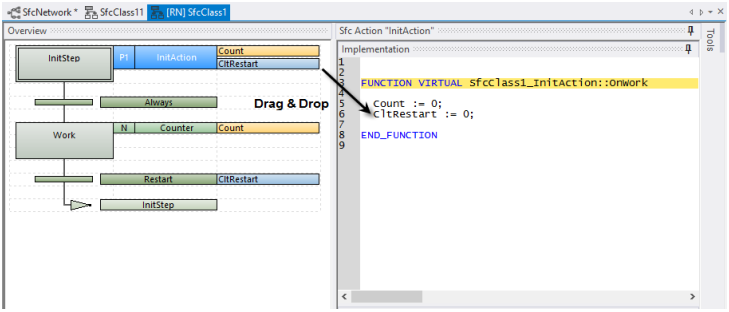
With a double click on the qualifier (here: **N** for **Not stored**) it can be edited:



The qualifier P1 (pulse rising edge), indicates that the **action is performed only once during activation** of the Init step (positive edge).

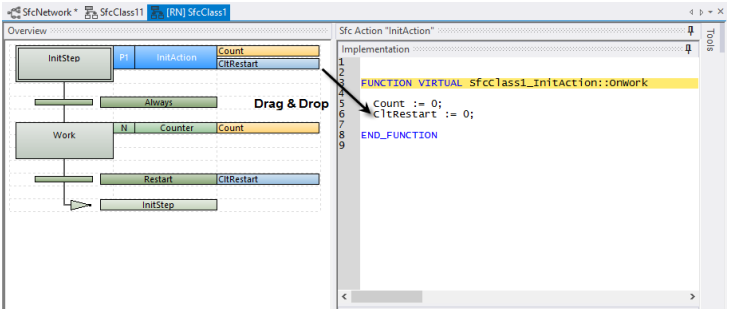


Next add the “Count” server of the new action “InitAction” and enter the line “`Count := 0;`“ in its OnWork method.



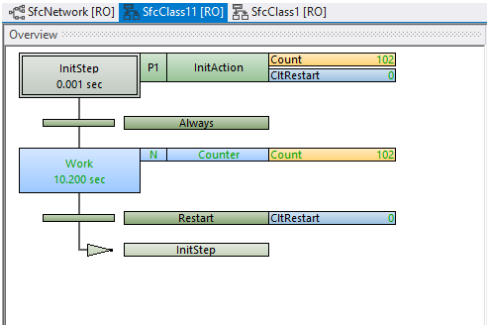
The **“CltRestart“** client must now only be added to **„InitAction“** and ensured that a reset is performed.

A continuous restart is thereby avoided.



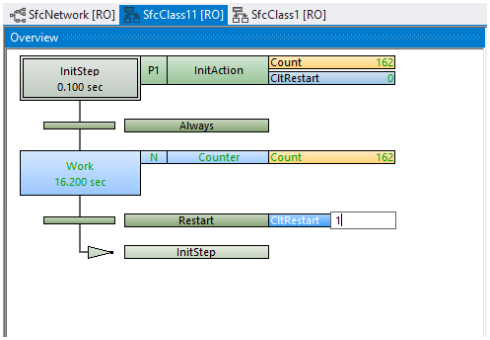
If the client were connected to a server, the **AutoWrite** property would have had to be set to **true** so that the value could be written to the connected serve.

The project can now be compiled and loaded into the control. To check the correct function of an SFC object (placed SFC class), the SFC object debugger can be opened in online mode with a double click on the upper right corner of the SFC object.



Current active components are shown with a blue background and with light green digits. How long the component has been active or when it was active last is displayed. The values in the in and outputs are – also light green – seen on the right side of the respective object.

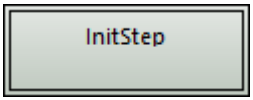
With a double click, the values can then be edited. CltRestart for example, can then be set directly in this view and the project restarted:



3.5.1 Description of SFC Components

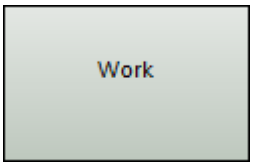
3.5.1.1 SFC Initstep

Each SFC diagram contains an **InitStep**, which is activated when the application is started. The InitStep cannot be deleted and is identified by a double frame. Other than this minor difference, it functions like a normal step.



3.5.1.2 SFC Step

SFC steps represent the possible steps/conditions the step sequence simulated by the SFC class. Through activation/deactivation, the step ensures the corresponding actions are enabled/disabled in accordance with the action qualifier. In active status, the step ensures that the following transitions are cyclically evaluated.



Steps have an **"X" server** that indicates whether the step is active and a **"T"** server that shows how long the step has been active in milliseconds or when it was last activated. If these servers required in a transition of action, they can be added to the in/outputs of respective transition or action by Drag & Drop from the SFC class and accessed with StepName.x or StepName.T.

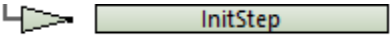
3.5.1.3 SFC Transition

SFC transitions define the conditions for moving between the possible steps/status of the sequence in the SFC class. Transitions are cyclically evaluated by the steps directly preceding them as soon all preceding steps are active. If the condition of a transition is met when evaluated, all steps directly preceding it are deactivated and the following step is activated. In one or more steps follow several alternative transitions, they are evaluated from left to right until the condition of a transition is met.



3.5.1.4 SFC Jump

SFC jumps are used to move to a step that cannot be reached over a normal connection. The step ensures that preceding transitions activate the jump destination when the condition is successfully evaluated.



Caution!

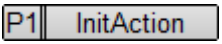
It should not be attempted to activate the step of a parallel branch with a jump, since this could lead to undefined conditions or conditions that cannot be ended.

3.5.1.5 SFC Action

SFC actions perform the actual work that should be completed in the steps. Actions react to the activation/deactivation of the steps to which they belong and respond differently, depending on the specified action qualifier.

Caution!

It should be taken into consideration that the **active actions** are called in an **undefined sequence**!



Actions have an **"A" server**, which indicates whether the action is active, a **"Q" server** that indicates whether the action is performed one last time before deactivation and a **"T" server** that shows how long the step has been active in milliseconds or when it was last activated.

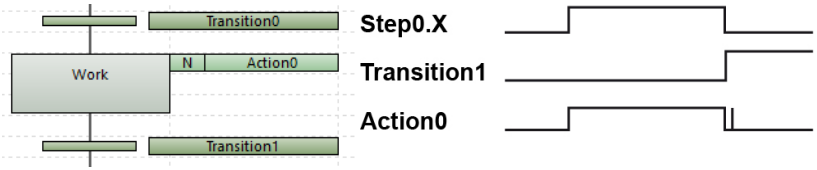
Caution!

All actions, which do not react to flanks (P, P0, P1), are called once more in the cycle following deactivation. This can be used to reset outputs through the X server of corresponding steps or the A and Q servers of the action.

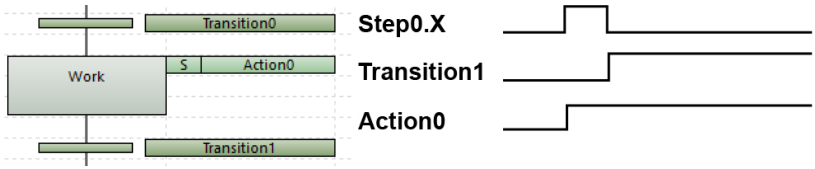
3.5.1.6 SFC Action Qualifier

The action qualifier indicates when and how an action is performed.

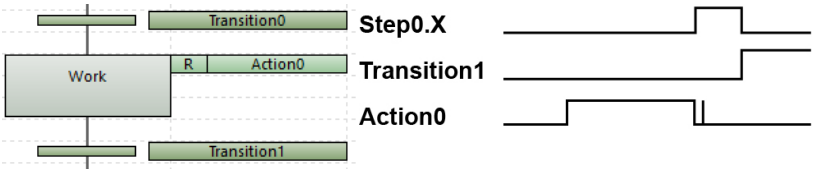
3.5.1.61 N (not saved)



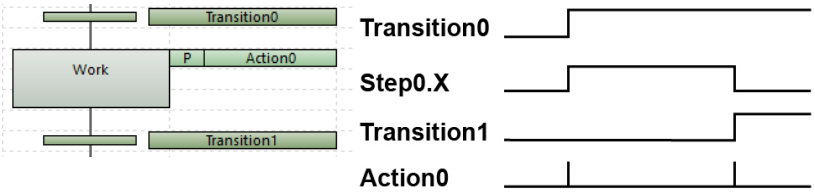
3.5.1.62 S (saved)



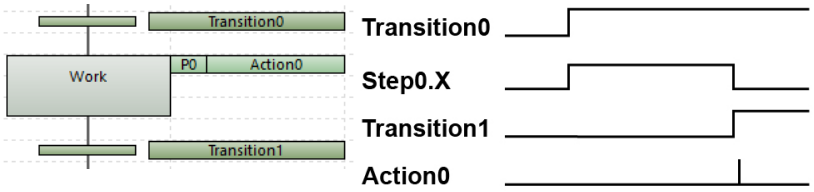
3.5.1.63 R (reset)



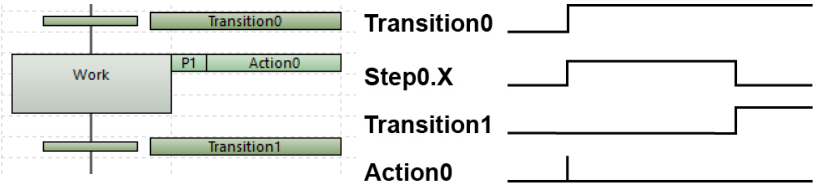
3.5.1.64 P (with rising and falling flank)



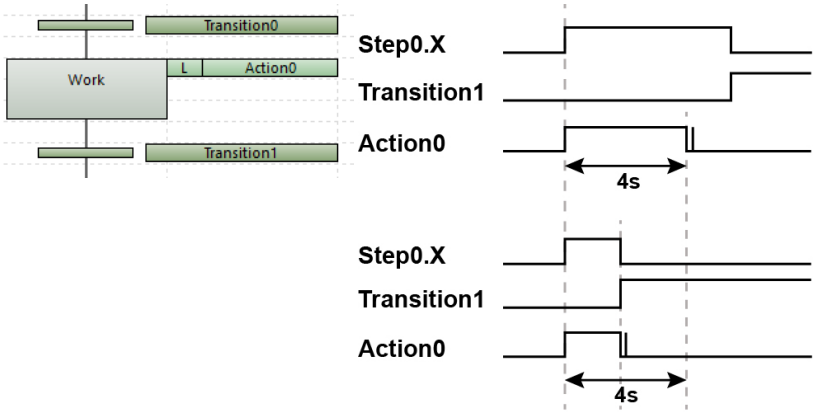
3.5.1.65 P0 (with falling flank)



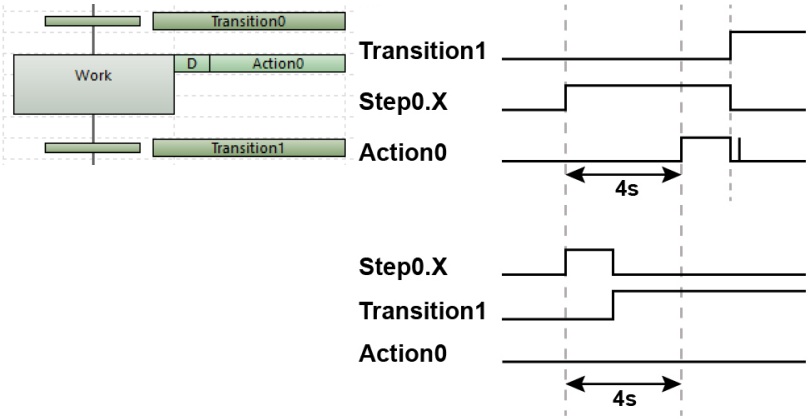
3.5.1.66 P1 (with rising flank)



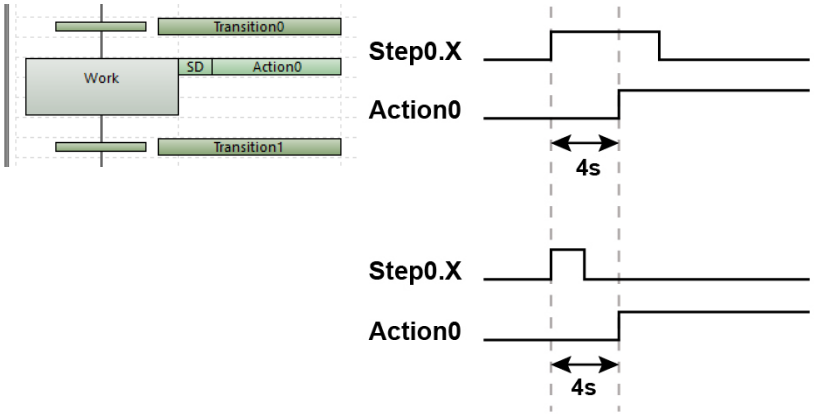
3.5.1.67 L (time-limited)



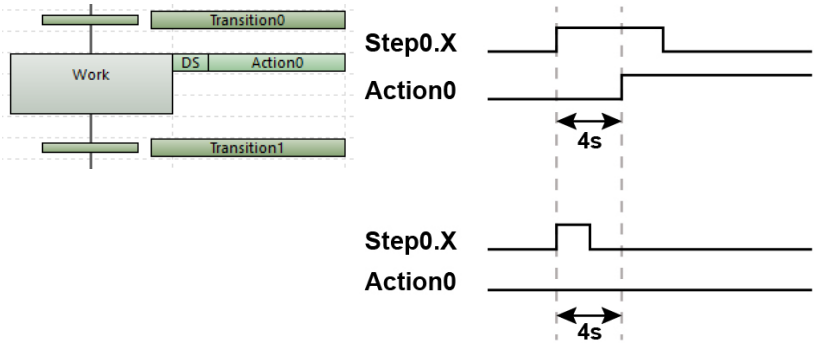
3.5.1.68 D (time-delayed)



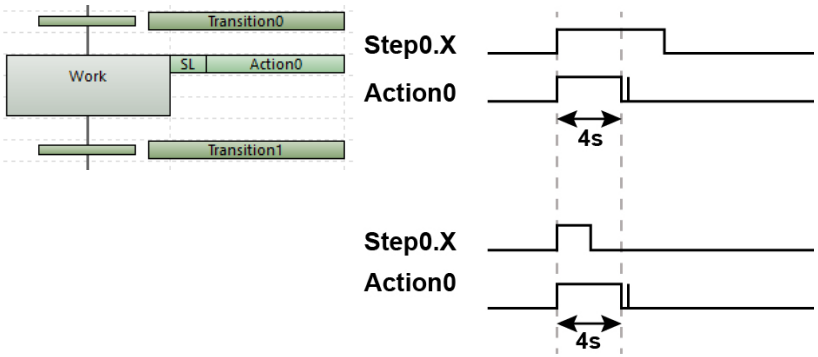
3.5.1.69 SD (saved and time-delayed)



3.5.1.610 DS (time-delayed and saved)



3.5.1.611 SL (saved and time-limited)



Caution!

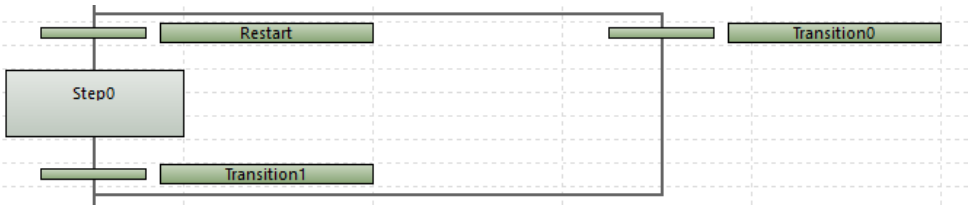
Saved actions also remain active if the corresponding step is deactivated. To reset a saved action, it can be added to a different step with the "R" qualifier. To achieve this, a new action is added to the affected step, given the same name as the action to reset and assigned the action qualifier "R".

3.5.1.7 SFC Alternative Branch

An SFC alternative branch is generated when several alternative transitions follow a step or several parallel steps. Only one of the alternative branches can be processed.



Alternative branches can also be reassembled.



Caution!

When evaluating transitions of an alternative branch, they are analyzed from left to right until a condition is met or all transitions have been evaluated.

3.5.1.8 SFC Parallel Branch

An SFC parallel branch is generated when several parallel steps follow a transition or several alternative transitions. Parallel branches are processed individually until they are reassembled.



Parallel branches can also be reassembled.

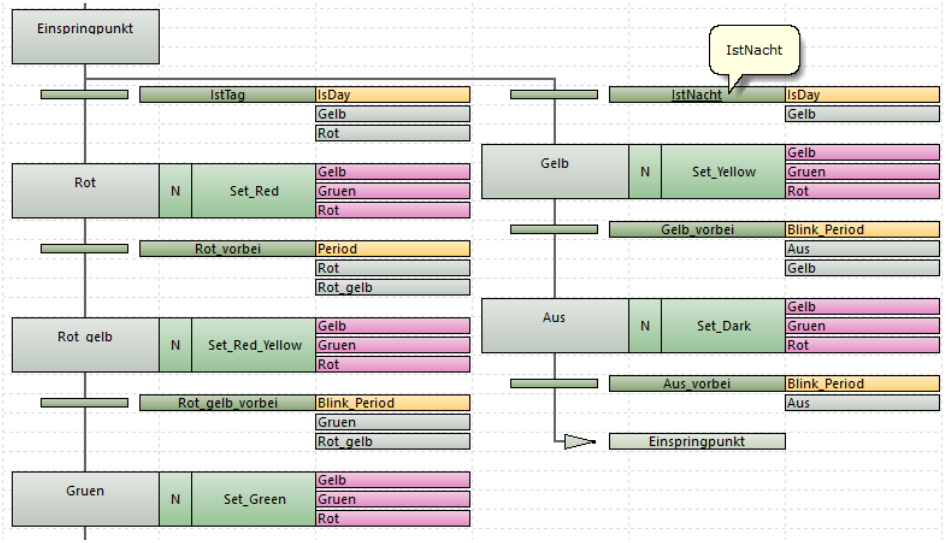


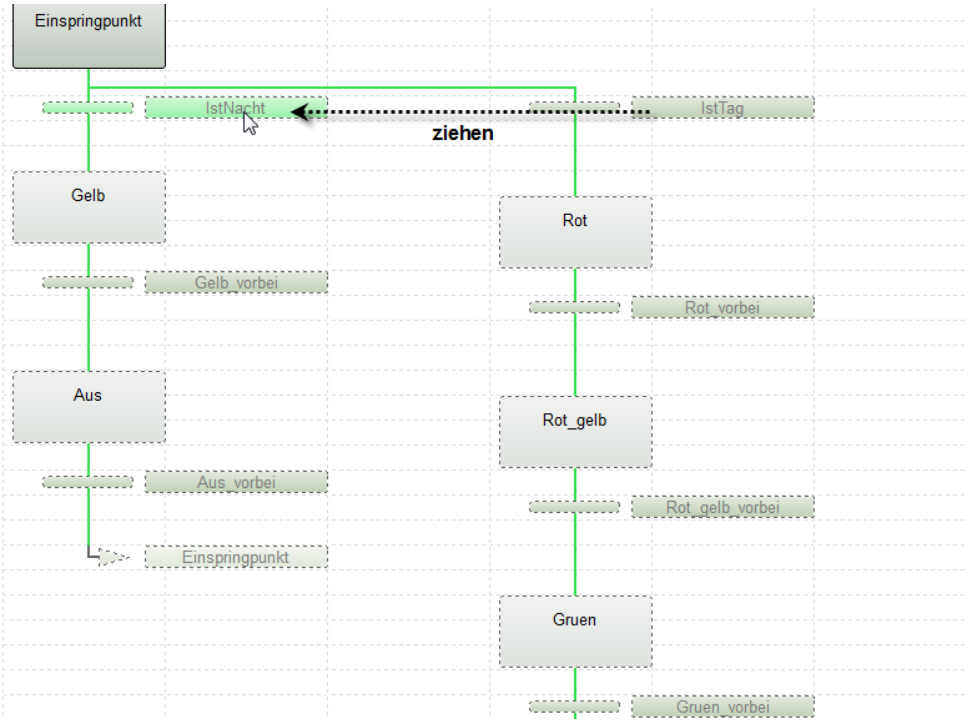
Caution!

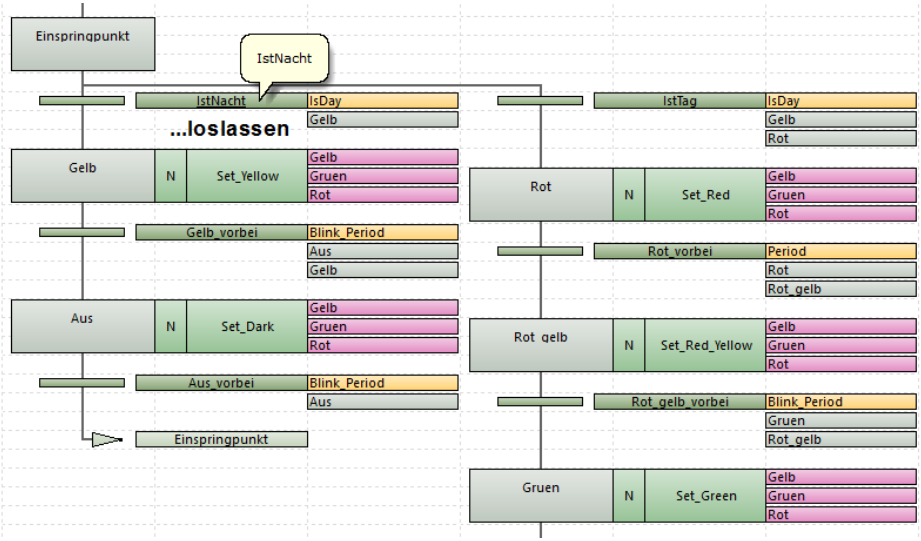
If several parallel branches are reassembled, the following transitions are only evaluated if each parallel branch of the last step is active.

Tip

Each branch in SFC can now be switched with an element on the same branch plane as long as it does not cause a collision with the feedback connections.

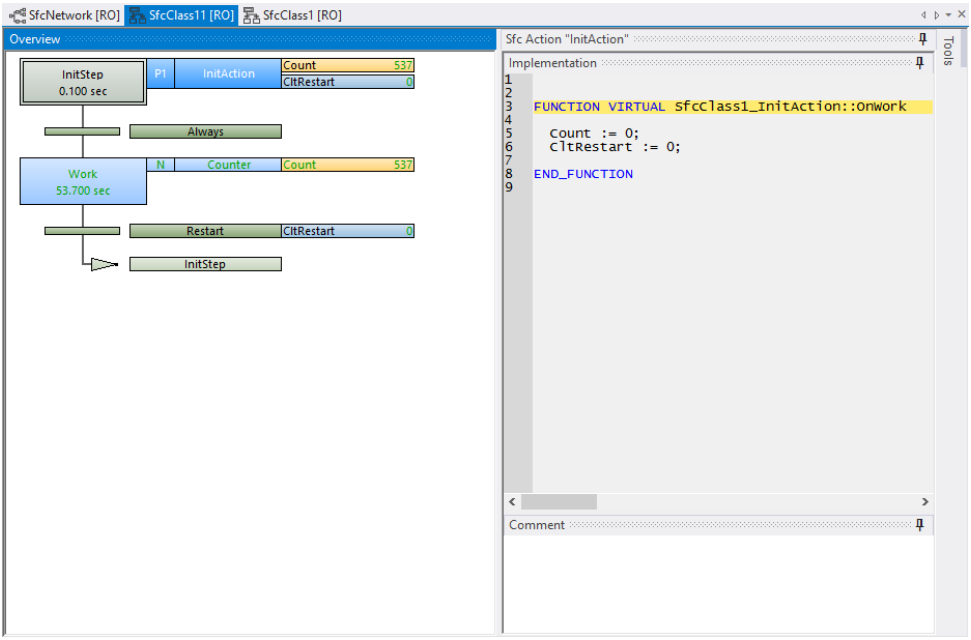



Einspringpunkt
IstNacht
IstTag
ziehen
Gelb
Gelb_vorbei
Rot
Rot_vorbei
Aus
Aus_vorbei
Rot_gelb
Rot_gelb_vorbei
Einspringpunkt
Gruen
Gruen_vorbei


Einspringpunkt
IstNacht
IstNacht
IsDay
Gelb
...loslassen
Gelb
N
Set_Yellow
Gelb
Gruen
Rot
Gelb_vorbei
Blink_Period
Aus
Gelb
Aus
N
Set_Dark
Gelb
Gruen
Rot
Aus_vorbei
Blink_Period
Aus
Einspringpunkt
IstTag
IsDay
Gelb
Rot
Rot
N
Set_Red
Gelb
Gruen
Rot
Rot_vorbei
Period
Rot
Rot_gelb
Rot_gelb
N
Set_Red_Yellow
Gelb
Gruen
Rot
Rot_gelb_vorbei
Blink_Period
Gruen
Rot_gelb
Gruen
N
Set_Green
Gelb
Gruen
Rot

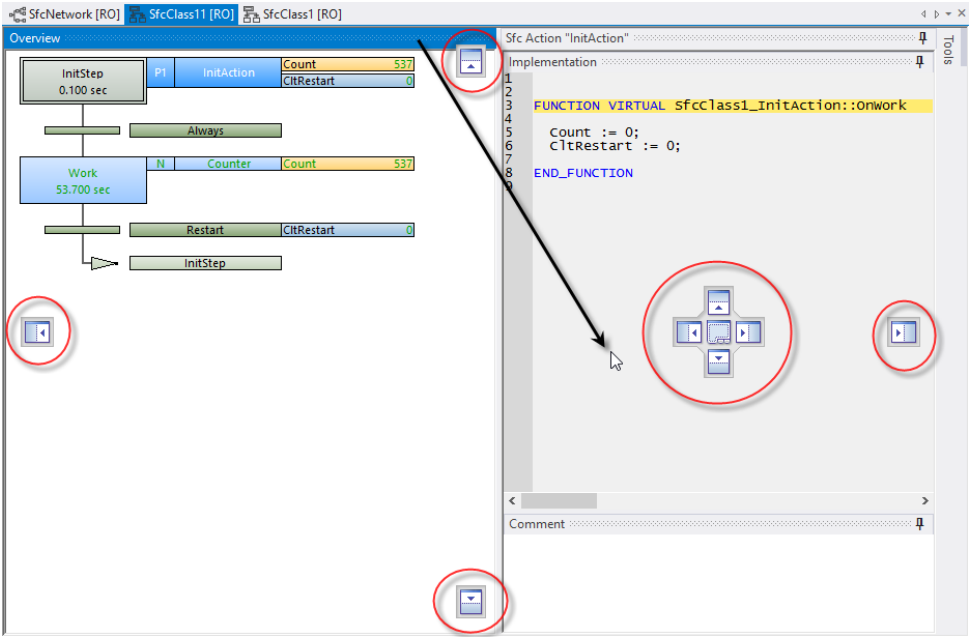
3.5.2 SFC User Interface Options

Through the LASAL menu item **Extras => Options**, the **Always Show SFC Detail View** parameter can used to set whether the overview Detail view should be displayed side by side. If **true** is selected, the SFC appears as follows:



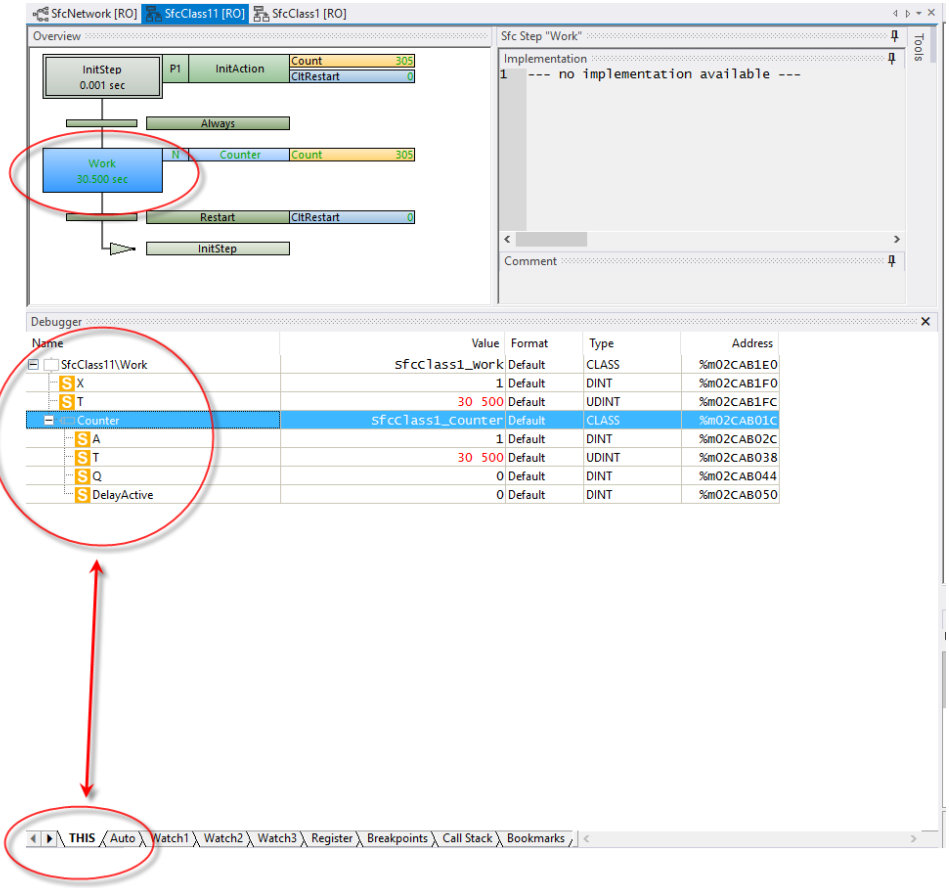
If enough space is available on the monitor, the user does not have to choose between the views.

All display sections (panels) of the SFC (these are: overview, detail view, Sfc toolbox and the commentary field) are so-called docking panes, which can be moved using the mouse, and configured as desired.

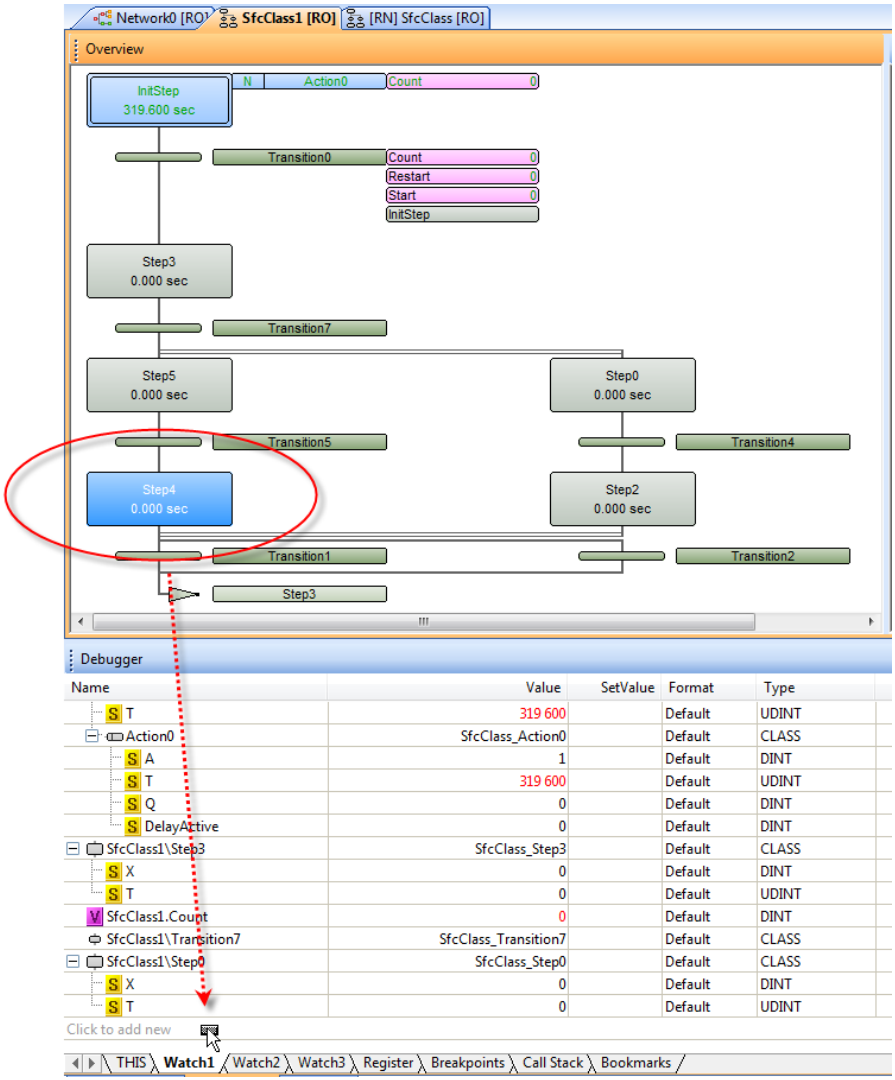


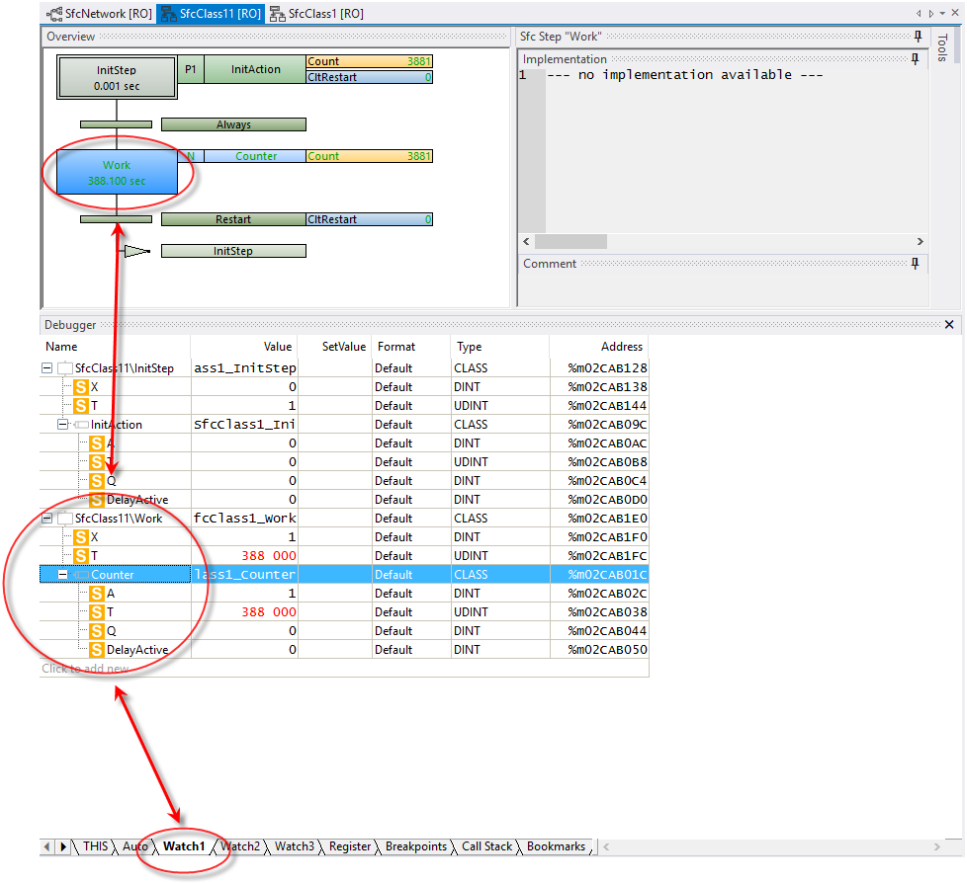
3.5.3 THIS and Watch Window in SFC

On debugging SFC classes, the **THIS** and **Watch** window can be used. All IO servers, IO clients, internal variables of the SFC class, as well as all steps and actions including their active and time servers are displayed. In order to display an element in the THIS window, click it in the SFC overview:



Dragging elements into the Watch window is also possible, just like elements of a normal class.


Network0 [RO]
SfcClass1 [RO]
[RN] SfcClass [RO]
Overview
InitStep
319.600 sec
N
Action0
Count
0
Transition0
Count
0
Restart
0
Start
0
InitStep
Step3
0.000 sec
Transition7
Step5
0.000 sec
Step0
0.000 sec
Transition5
Transition4
Step4
0.000 sec
Step2
0.000 sec
Transition1
Transition2
Step3
Debugger
Name
Value
SetValue
Format
Type
T
319 600
Default
UDINT
Action0
SfcClass_Action0
Default
CLASS
A
1
Default
DINT
T
319 600
Default
UDINT
Q
0
Default
DINT
DelayActive
0
Default
DINT
SfcClass1\Step3
SfcClass_Step3
Default
CLASS
X
0
Default
DINT
T
0
Default
UDINT
SfcClass1.Count
0
Default
DINT
SfcClass1\Transition7
SfcClass_Transition7
Default
CLASS
SfcClass1\Step0
SfcClass_Step0
Default
CLASS
X
0
Default
DINT
T
0
Default
UDINT
Click to add new
THIS
Watch1
Watch2
Watch3
Register
Breakpoints
Call Stack
Bookmarks


SfcNetwork [RO]
SfcClass11 [RO]
SfcClass1 [RO]
Overview
InitStep
0.001 sec
P1
InitAction
Count 3881
CltRestart 0
Always
Work
388.100 sec
N
Counter
Count 3881
Restart
CltRestart 0
InitStep
Sfc Step "Work"
Implementation
1 --- no implementation available ---
Comment
Tools
Debugger
Name
Value
SetValue
Format
Type
Address
SfcClass11\InitStep
ass1_InitStep
Default
CLASS
%m02CAB128
X
0
Default
DINT
%m02CAB138
T
1
Default
UDINT
%m02CAB144
InitAction
sfcclass1_Ini
Default
CLASS
%m02CAB09C
A
0
Default
DINT
%m02CAB0AC
0
Default
UDINT
%m02CAB0B8
Q
0
Default
DINT
%m02CAB0C4
DelayActive
0
Default
DINT
%m02CAB0D0
SfcClass11\Work
fcclass1_work
Default
CLASS
%m02CAB1E0
X
1
Default
DINT
%m02CAB1F0
T
388 000
Default
UDINT
%m02CAB1FC
Counter
lass1_Counter
Default
CLASS
%m02CAB01C
A
1
Default
DINT
%m02CAB02C
T
388 000
Default
UDINT
%m02CAB038
Q
0
Default
DINT
%m02CAB044
DelayActive
0
Default
DINT
%m02CAB050
Click to add new
THIS
Auto
Watch1
Watch2
Watch3
Register
Breakpoints
Call Stack
Bookmarks

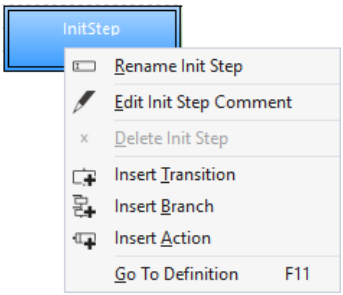
3.5.4 Call Squence of SFC Components

1. In the first task cycle after starting the application, the InitStep is first activated. With the activation of the InitStep, all actions assigned to it are initialized with the defined action qualifier. All active actions, which are triggered by a rising flank (P, P1) are called first. All other active actions are then called. After calling all active actions, the InitStep evalutates all following transitions from left to right. when the conditions of a transition are met, the InitStep is deactivated and all steps following the transition are activated. Active actions with the same qualifier are processed in the sequence they are displayed in the SFC network.

   After calling all active actions, the InitStep evaluates all following transitions from left to right. When the conditions of a transition are met, the InitStep is deactivated and all steps following the transition are activated.
2. Starting with the second task cycle after starting the application, the actions that were deactivated in the previous task cycle or are triggered with a falling edge (P, P0) of a step that was deactivated in the last task cycle are called first. Afterwards, all active actions that are triggered with a rising edge (P, P1) are called first, followed by all other active actions. After all active actions are called, all transitions directly following all active steps are evaluated from left to right. When a condition for a transition is met, the preceding step is deactivated and all the steps directly following the transition are activated.
3. With actions, the actions that were deactivated in the previous task cycle or those triggered with a falling flank (P, P0) of a step that was deactivated in the last task cycle are called first. Active actions, which are triggered by a rising flank (P, P1) are always called before all other active actions. Otherwise, the call sequence for the actions is undefined.
4. Transitions are always evaluated after all active actions have been called. The sequence for which step the transitions evaluate first is undefined.
5. Alternative transitions are always evaluated from left to right until the condition of a transition is met or all transitions have been evaluated.
6. If one or more transitions follow more than one step (combined parallel branches), the transitions are only evaluated when the last step in each parallel branch is active.

3.5.5 Context Menu of the SFC Editor Components

3.5.5.1 Init Step



Rename InitStep

Rename the InitStep – is identical to the function of the delayed click or with selecting and pressing **F2**.

Edit Init Step Comment

Opens the commentary window for entries or sets the focus if already visible. Functions the same as with a double click on the component.

Delete Init Step

Always deactivated, since the InitStep cannot be deleted.

Insert Transition

Inserts a new transition below the InitStep.

Insert Branch

Inserts a new branch below the InitStep.

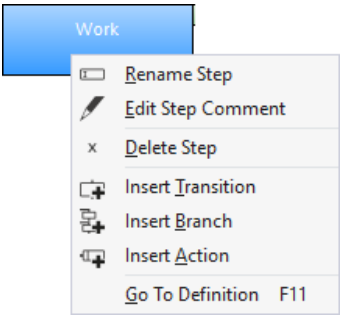
Insert Action

Inserts an action in the InitStep.

Go To Definition

Jumps to the definition of the InitStep in the **ClassTree**. Same function as **F11**.

3.5.5.2 Step



Rename Step

Rename the step – is identical to the function of the delayed click or with selecting and pressing **F2**.

Edit Step Comment

Opens the commentary window for entries or sets the focus if already visible. Functions the same as with a double click on the component.

Delete Step

Deletes the **step**.

Insert Transition

Inserts a new transition below the step.

Insert Branch

Inserts a new branch below the step.

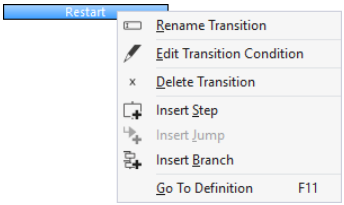
Insert Action

Inserts an action in the Step.

Go To Definition

Jumps to the definition of the step in the **ClassTree**. Same function as **F11**.

3.5.5.3 Transition



Rename Transition

Rename the transition – is identical to the function of the delayed click or with selecting and pressing **F2**.

Edit Transition Condition

Opens the implementation window or sets the focus, if already visible. Functions the same as with a double click on the component.

Delete Transition

Deletes the transition.

Insert Step

Inserts a new step below the transition.

Insert Jump

Inserts a new jump below the transition. This menus item is **only active when** the transition is **the last element in the branch**.

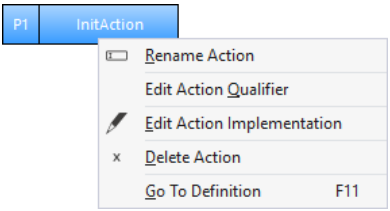
Insert Branch

Inserts a new branch below the transition. This menu point is **only active when Insert Jump is inactive**.

Go To Definition

Jumps to the definition of the transition in the **ClassTree**. Same function as **F11**.

3.5.5.4 Action

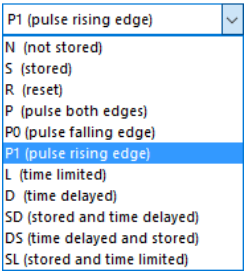


Rename Action

Rename the Action – is identical to the function of the delayed click or with selecting and pressing **F2**.

Edit Action Qualifier

Changes the action qualifier. Here, the following menu opens:



The new qualifier can then be seen here.

Edit Action Implementation

Opens the implementation window or sets the focus, if already visible. Functions the same as with a double click on the component.

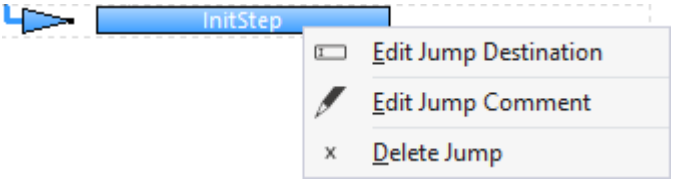
Delete Action

Deletes the action.

Go To Definition

Jumps to the definition of the action in the **ClassTree**. Same function as **F11**.

3.5.5.5 Jump



Edit Jump Destination

Rename the destination of the jump – is identical to the function of the delayed click or with selecting and pressing **F2**.

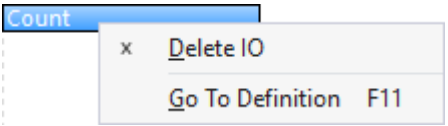
Edit Jump Comment

Opens the commentary window for entries or sets the focus if already visible. Functions the same as with a double click on the component.

Delete Jump

Deletes the jump.

3.5.5.6 IO



Delete IO

Deletes the input / output (server / client / internal variable).

Go To Definition

Jumps to the definition of the IO in the **ClassTree**. Same function as **F11**.

3.6 Pre-Processor Instructions

Pre-processor instructions are used to define constants or mark specific program lines which are treated as commentary (conditional compilation). With conditional compiling, the program lines are processed exactly the same as lines behind the \\ characters or in parenthesis with stars (*parentheses with stars*). With #pragma instructions, compiler warnings can be deactivated and the data type setting can be changed.

3.6.1 Predefined Constants

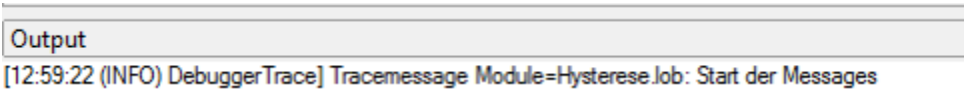
3.6.1.1 Trace Macro

The Trace Macro can be used to output messages during the processing of program code. This message is defined with help from constants in the program code and displayed in the **output window** under **Debugger Trace**.

The message is shown in the window, **Output**, **Debugger Trace**.



```
if Switch = 1 then
  TRACE("Start der Messages");
  Switch := 0;
end_if;
```



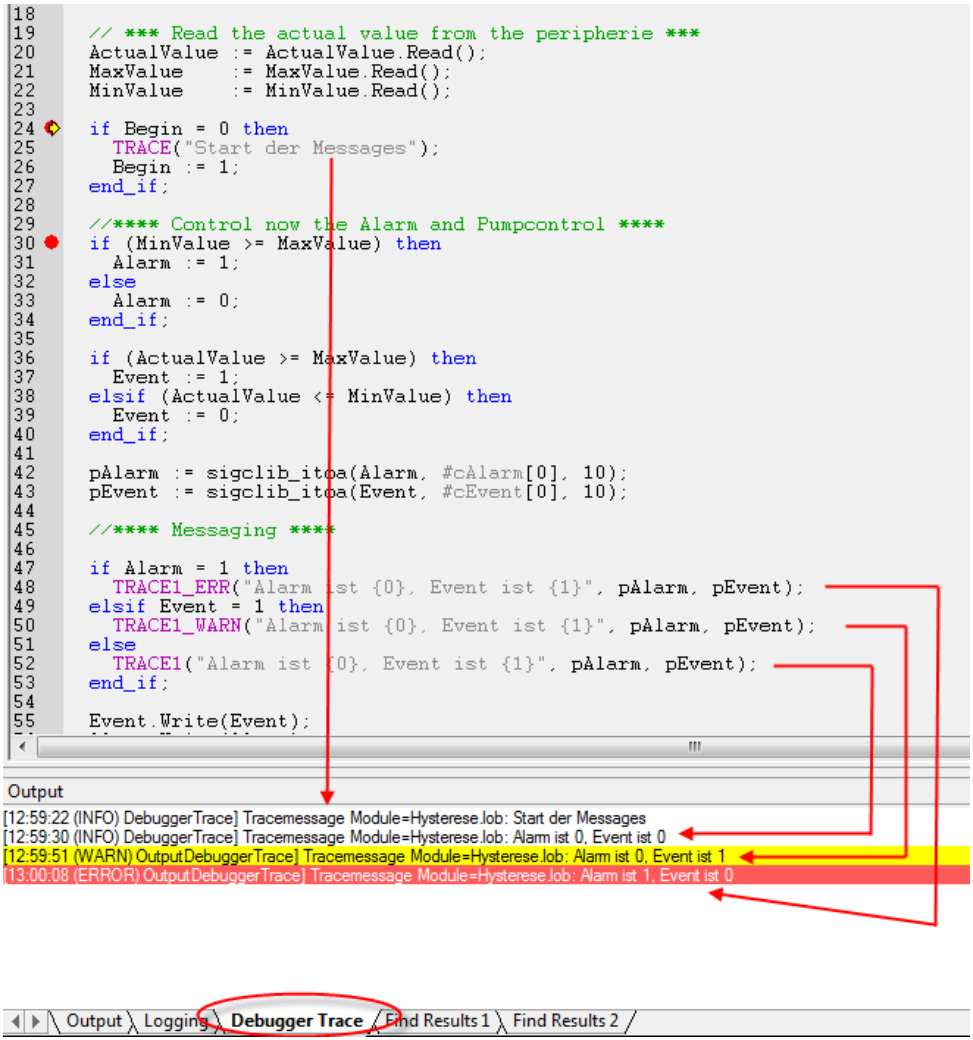
3.6.1.2 TRACE0 - TRACE2

These function the same as **Trace**, except that for message text, an **additional** parameter can be added with **TRACE0** that is output in the debugger console. With **TRACE1**, 2 additional parameters can be included and with **TRACE2**, 3 **additional** parameters are possible. With the additions **_MSG**, **_WARN** and **_ERR**, the type of the output can be defined. A few examples:

TRACE1_MSG => as with **TRACE**, the message with the addition (**INFO**) is only displayed in the Output, Debugger Trace window. Two additional parameters are included.

TRACE2_WARN => The message is output with the addition (**WARN**) with a yellow background in the Debugger Trace and Output windows. Three additional parameters are included.

TRACE0_ERR => The message is output with the addition (**WARN**) with a red background in the Debugger Trace and Output windows. One additional parameter is included.



The Trace Macro also counts as a predefined constant.

3.6.1.3 Constant _PI

The _PI constant returns the result 3.14... as a **REAL** number.

```
PiValue := _PI;
```



The **_PI constant** is also a predefined value.

3.6.1.4 _LSL_COMPILEREVERSION

This constant returns the current compiler version. In the form

```
#if _LSL_COMPILERVERSION >= 42
        ...
        ...
        ...
#endif
```

a compiler version dependent, conditional compilation is possible.

3.6.1.5 LSL_entriesof

The macro LSL_entriesof is used to determine the size of an array **in number of elements**. The macro is defined in the header file RTOS_STMacros.h (for ST files) or RTOS_CMacros.h (for C files).

Code example:

```
iNumber_of_elements := LSL_entriesof(m_array);
```

3.6.2 Available Pre Processor Instructions

3.6.2.1 #define

With **#define;** symbolic constants can be generated in a program or a header file. Constants defined with **#define** can be removed with the #undef instruction.

```
16
17  #define ENTER_  13
18  #define ESC_    27
19
20    case Key of
21
22      ENTER_: // Statement
23
24      ESC_:   // Statement
25
26    end_case;
27
28  #undef ENTER_
29  #undef ESC_
30
```

In this example, the constant with the name **ENTER_** is defined and represents the value 13.

The variable **Key** cannot be assigned as an ENUM type (rather as USINT, for example).

If the Key variable now contains the value 13, the step control will jump to the **ENTER_** step.

Through the instruction, **#undef ENTER_**, the constant is no longer valid. If a constant, which has been undefined, is used in the program, the message **ERROR: Undefined Symbol 'ENTER_'** appears in the output window.

3.6.2.2 #undef

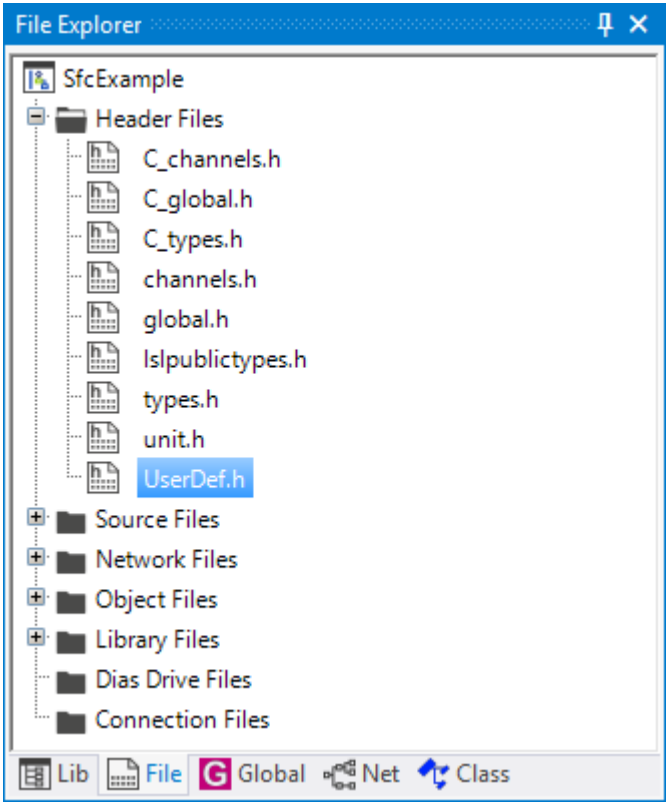
Removes the constants defined with the #define instruction.



```
#undef ENTER_
```

3.6.2.3 #ifdef

Constants defined in a Header file can be observed in a program as follows:
In the graphic is a section from a Header file in which a constants are defined.
The Header file, **UserDef.h**, is provided for the user to define global constants.



Tip

The header file, **global.h**, is not intended for the user to make program changes. This file is overwritten with the next **Rebuild All** of the project. The line **#define Touch_IPC_121** in the **UserDef.h** header file ensures that the variable resolution is set to **800*600**.

```
30
31  #ifdef TOUCH_IPC_151     // 15.1" Terminal
32
33    Resolution := 1024*768;
34
35  #elif TOUCH_IPC_121      // 12.1" Terminal
36
37    Resolution := 800*600;
38
39  #elif TOUCH_TERM_57      // 5,7" 1/4 VGA Terminal
40
41    Resolution := 324*240;
42
43  #else
44
45    Resolution := 0;
46
47  #endif
48
```

Note

Constants can be assigned a value when defined but its not necessary to do so.

```
#define MAX_COLOR        // only the constant is available.
#define MAX_COLOR 255    // the constant is available and has a value 255.
```

3.6.2.4 #ifndef

The program code shown below means, if the constant **CONST** is not defined then it contains the value 44.

```
49  #ifndef CONST
50    Value := 44;
51  #else
52    Value := 0;
53  #endif
```

3.6.2.5 #elif, #else, #endif

These Pre-Processor instructions are under the description #ifdef.

3.6.2.6 #if

When the #if-instruction is used, the constant must be assigned a value.

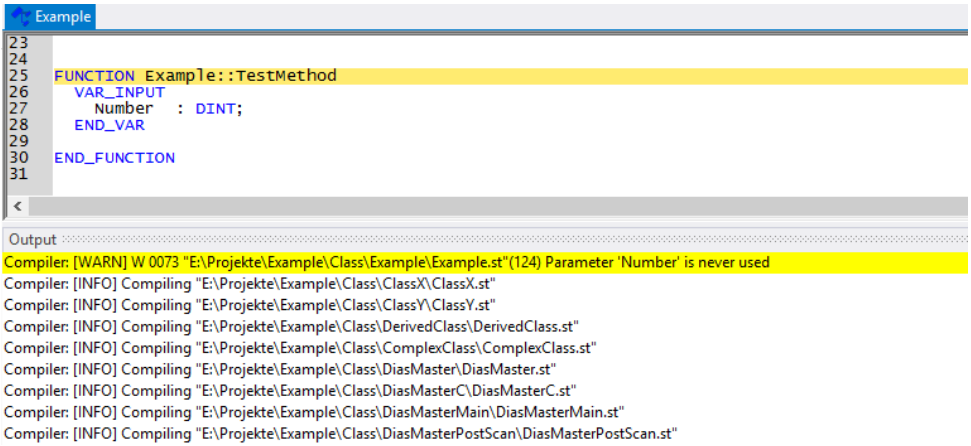
```
56  #define MY_CONST  11
57  //#define MY_CONST  22
58
59
60  #if MY_CONST = 11
61
62    Value := 44;
63
64  #elsif MYCONST = 22
65
66    Value := 55;
67
68  #else
69
70    Value := 0;
71
72  #endif
73
```

3.6.3 Pragma Instruction

The pragma instruction uses C-programming to provide access to several instructions. The function of the pre processor is expanded through the use of pragma instructions. The usable instructions are compiler and operating system specific. The instructions not recognized by the pre processor trigger a compiler message.

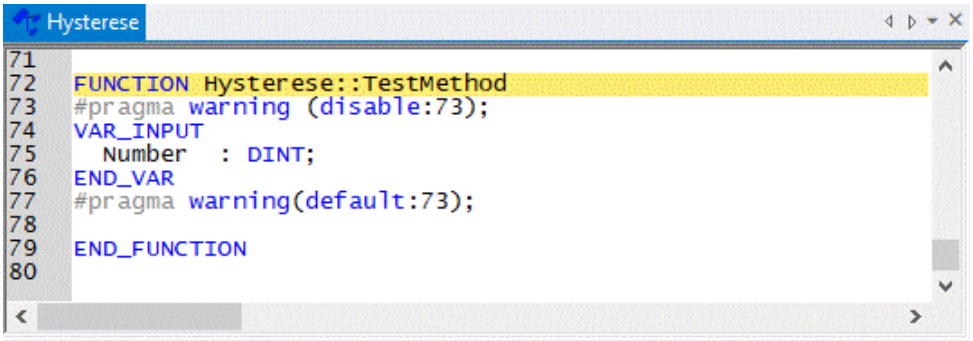
3.6.3.1 #pragma warning

The warning, **Number' declared but never used**, shown in the illustration below is displayed when a local variable is available but is never used in the program.



The Pragma instruction to deactivate this message is given through the output (**W 0073**).

Using this information from the compiler, the Pragma instruction **disable** and **default** can be use to deactivate this warning.



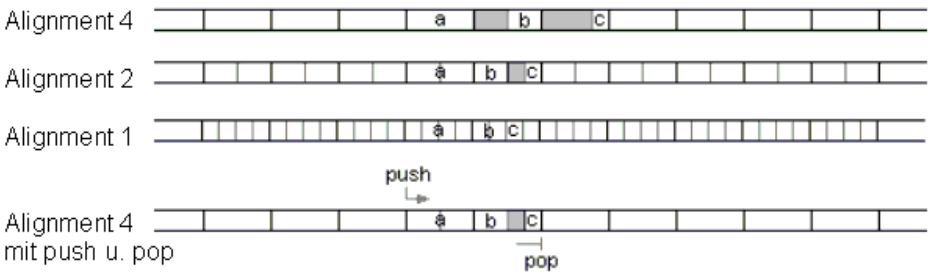
3.6.3.2 #pragma pack

With **#pragma pack** instructions, individual types, variables or function definitions can be changed in a complete project after the alignments (data alignment) have been made. The **#pragma pack (push)** instruction saves settings and **#pragma pack (pop)** recalls them. In the graphic below, the data alignment for various alignment settings is shown.

a: DINT

b: INT

c: SINT



In this illustration **push** and **pop instructions** can be seen:

```
TYPE
#pragma pack (push,1)
  RESULT       : STRUCT
    value      : USINT;
    hi_limit   : USINT;
    lo_limit   : USINT;
  END_STRUCT;
#pragma pack(pop)
END_TYPE
```

After offsetting both instructions, the data alignment is only valid for area between them.

3.6.3.3 #pragma pack (push, n)

Saves the actual alignment.

3.6.3.4 #pragma pack (pop)

pop reinstates the old alignment, which was made with **push**.



The number of **#pragma pack push** and **pop instructions** must be equal.

3.6.3.5 #pragma pack (n)

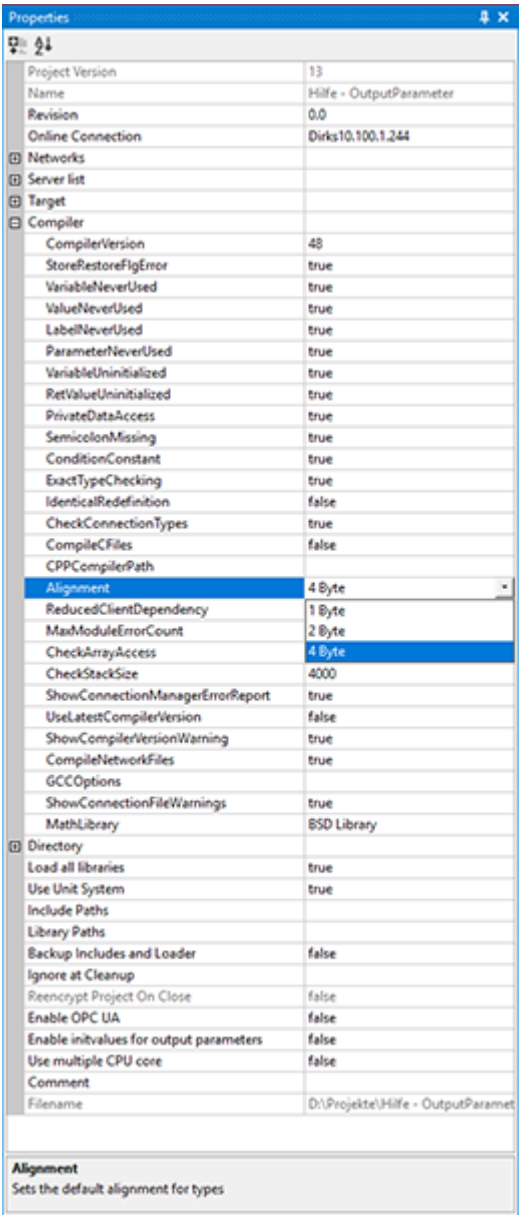
With this instruction, the alignment settings remain until the end of the file. The following alignments can be used: **n = 1, n = 2, n = 4**.



#pragma pack (2)

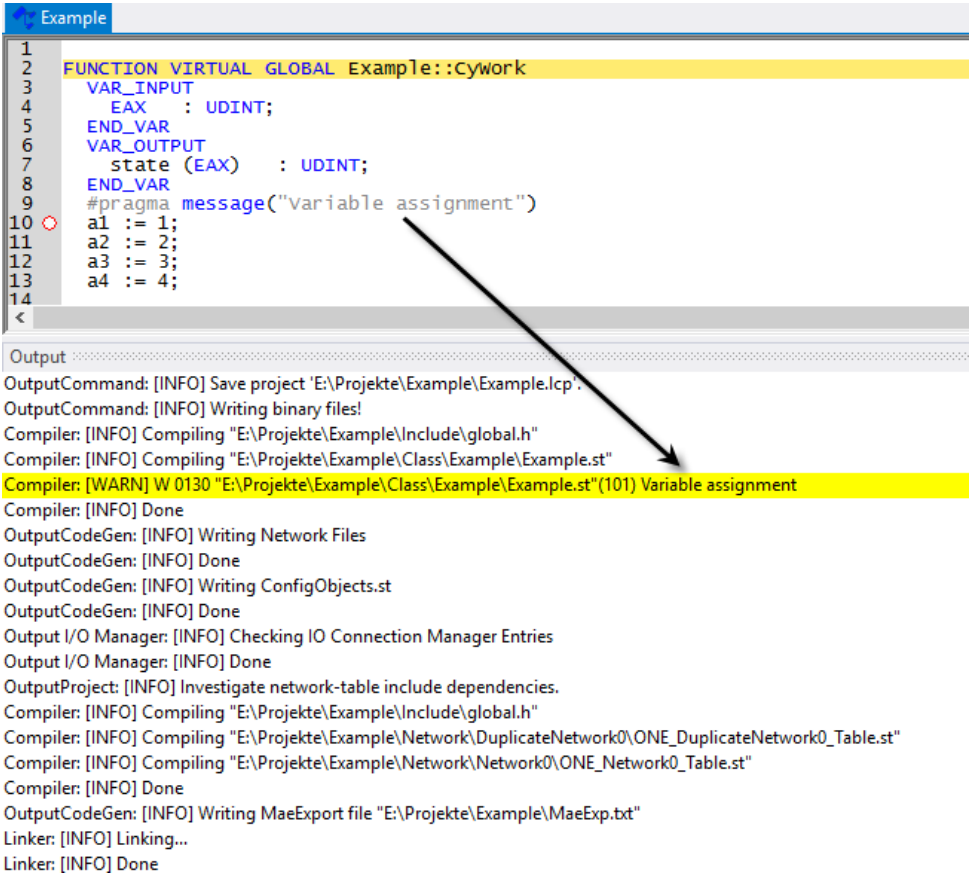
In this case the alignment is set to **2 Bytes**.

The data alignment can be set in the project properties in the **Properties window**. **Alignment 4** means the data is aligned to **4 bytes**.


Properties
Project Version 13
Name Hilfe - OutputParameter
Revision 0.0
Online Connection Dirks10.100.1.244
Networks
Server list
Target
Compiler
CompilerVersion 48
StoreRestoreFlgError true
VariableNeverUsed true
ValueNeverUsed true
LabelNeverUsed true
ParameterNeverUsed true
VariableUninitialized true
RetValueUninitialized true
PrivateDataAccess true
SemicolonMissing true
ConditionConstant true
ExactTypeChecking true
IdenticalRedefinition false
CheckConnectionTypes true
CompileCFiles false
CPPCompilerPath
Alignment 4 Byte
ReducedClientDependency
MaxModuleErrorCount
CheckArrayAccess
CheckStackSize 4000
ShowConnectionManagerErrorReport true
UseLatestCompilerVersion false
ShowCompilerVersionWarning true
CompileNetworkFiles true
GCCOptions
ShowConnectionFileWarnings true
MathLibrary BSD Library
Directory
Load all libraries true
Use Unit System true
Include Paths
Library Paths
Backup Includes and Loader false
Ignore at Cleanup
Reencrypt Project On Close false
Enable OPC UA false
Enable initvalues for output parameters false
Use multiple CPU core false
Comment
Filename D:\Projekte\Hilfe - OutputParamet
1 Byte
2 Byte
4 Byte
Alignment
Sets the default alignment for types

3.6.3.6 #pragma message

With the transmission of a **#pragma message**, a message is displayed in the output window under **Output, Compiler** after the compilation of this line.



3.6.3.7 #pragma once

This instruction tells the pre-processor that a Header file should only be included once. The can also be done using the #ifdef, #define and #endif. Both instructions are the same but **#pragma once** is faster used instead.

3.6.3.8 #pragma define (store)

This instruction sends the value of all currently buffered #defines to the preprocessor.

3.6.3.9 #pragma define (restore)

This instruction resets all #defines previously saved with #pragma define (store) to the original stored value.

3.6.3.10 #pragma checkarray (enable | disable | default)

This instruction switches on (enable) or off (disable) the option CheckArrayAccess or sets it to the value of the project settings. Thus it is possible to use CheckArrayAcces only for e.g. a class or a library.

3.6.4 #include

3.6.4.1 #include "File.h"

If the name of the header file is written between two apostrophes, the compiler searches for the **include file** in the actual project directory.

3.6.4.2 #include <File.h>

If the name of the included file is written between <> characters, the compiler searches for the Include file starting from the path in the **Options** menu under **Extras** where the **Include Paths** parameter is assigned.

3.6.4.3 #includeC "file.h"

This instruction is used to include C header files in the ST file. With the instruction #includeC “file.h”, Defines, Marcos, Declarations of types and functions from the C header file “file.h” are prepared for use in ST.

The following rules must be observerd:

All declarations must be made in the C programming language. This means that C programming syntax must be used. However, in an exception such as macros for example, implementation could be necessary in both programming languages (ST and C). Here, the predefined constant constant “_LSL_INCLUDEC „is provided.

Defines

Since the definitions are first triggered by the compiler for the respective application, their syntax mus comform to the respective program language. This has no effect on the simple definition of a constant. For complex definitions however, it can lead to problems.

See the following exampld for details:

```
#define BIT_0  0x0001
#define BIT_1  0x0002
#define BIT_2  0x0004
#define BIT_3  0x0008
#define BIT_ALL  (BIT_0 | BIT_1 | BIT_2 | BIT_3)
```

BIT_0 to BIT3 are still no problem. The definition “BIT_ALL” however, will not function in ST. The syntax does not conform to ST.

Solution

```
#ifdef _LSL_INCLUDEC
  #define BIT_ALL (BIT_0 or BIT_1 or BIT_2 or BIT_3)
#else
  #define BIT_ALL (BIT_0 | BIT_1 | BIT_2 | BIT_3)
#endif
```

Macros

The same applies with Defines for Macros. The respective syntax must also correspond to the programming language used.

Types

C and ST have different approaches to alignment and therefore all structures contained must have a 1-byte alignment. Otherwise, a warning is generated and the result of their use is not reliable.

Function Declarations

Caution, all function declarations must conform to C. Declarations that have been subjected to so-called name mangling (C++) cannot not be used in ST. The C++ compiler used is name mangled and ST is therefore not possible for the application. The keywork `extern „C“` solves this problem.

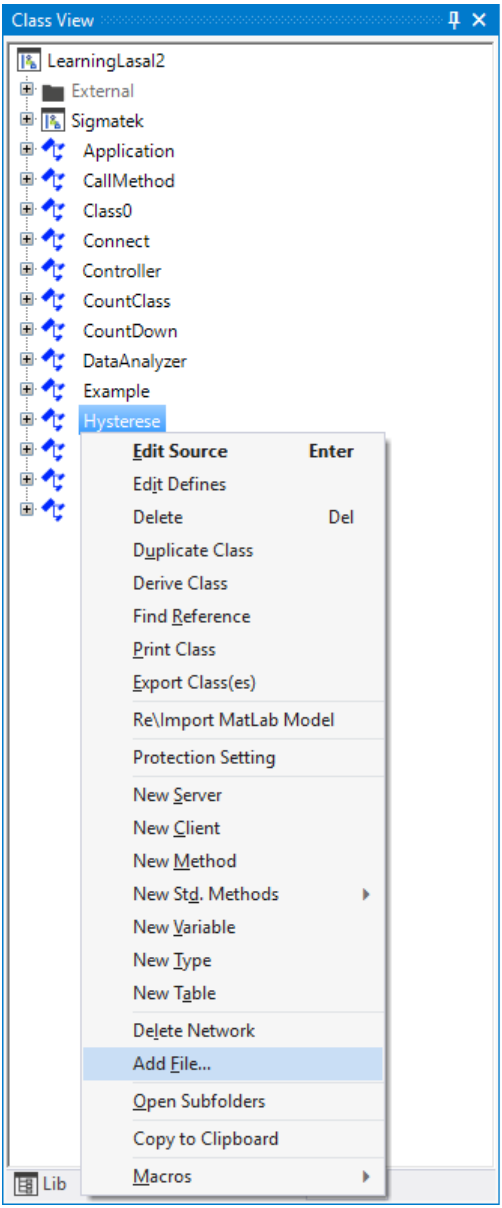
3.7 Linking Header Files

As soon as constants are used in a project, they can be defined in a **Header file**. This is not an absolute necessity but it can significantly improve clarity of projects with a large number of constants during development. The use of header files is completely at the discretion of the designer.

In LASAL CLASS 2 the define header files for the global project or linked to a specific class only.

3.7.1 Class Linked Header Files

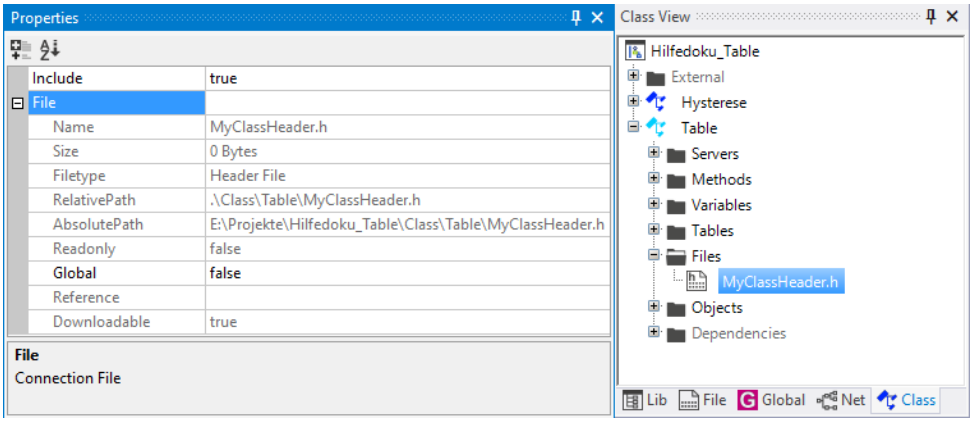
To link a header file to a class, the **Classes** tab in the project tree is selected. With a right click on the corresponding class (here: **Hysterese**), open the context menu and select the **Add File...** instruction.


Class View
LearningLasal2
External
Sigmatek
Application
CallMethod
Class0
Connect
Controller
CountClass
CountDown
DataAnalyzer
Example
Hysterese
Edit Source Enter
Edit Defines
Delete Del
Duplicate Class
Derive Class
Find Reference
Print Class
Export Class(es)
Re\Import MatLab Model
Protection Setting
New Server
New Client
New Method
New Std. Methods
New Variable
New Type
New Table
Delete Network
Add File...
Open Subfolders
Copy to Clipboard
Macros
Lib

When the dialog window **Open** appears, the desired header file can be selected or a new one (**MyClassHeader.h**) can be created.

In the properties window of the file, **Global** can be set to **true** or **false**. If the value is set to **True** then the header file can be used throughout the entire project. However, when this class is copied, the header file is transferred with it.

If the Global setting is left to **False**, the header file is only valid for the respective class.



If the Include value is set to **False**, the header file is not included in the *.st file. This is required, for example, in program sections written in C. When the class is copied, the header file is included however.

The default setting is **True**.

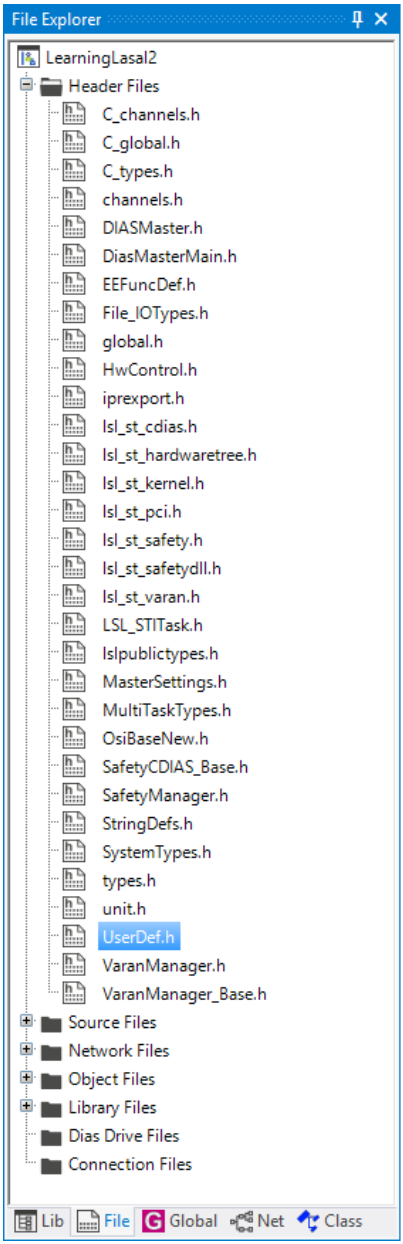


If a header file is included in several classes in a project, it is recommended to use the **#pragma once** instruction. With this instruction, the pre-processor is told that the header file is included only once. This has the advantage of accelerated compilation of the project.

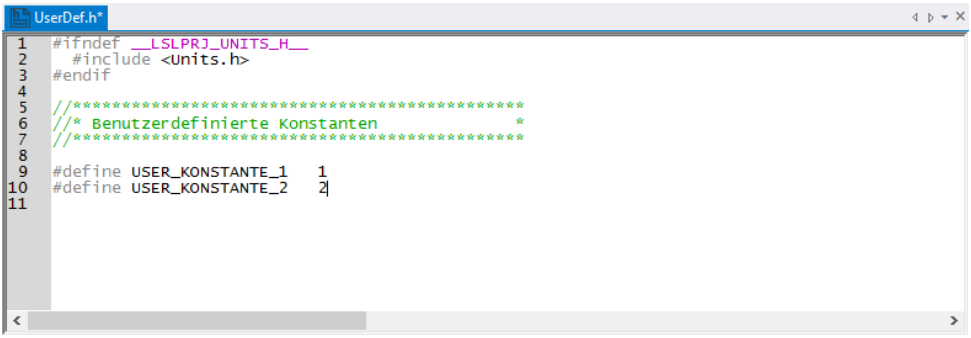
More information can found under Pragma Instructions in the chapter, #pragma once.

3.7.2 Global Header Files

The fastest and easiest way to define constants is to enter definitions in the pre-existing UserDef.h header file. To access the **UserDef.h** file, switch to the project tree and select the **Files** tab and open the **Header Files** folder.


File Explorer
LearningLasal2
Header Files
C_channels.h
C_global.h
C_types.h
channels.h
DIASMaster.h
DiasMasterMain.h
EEFuncDef.h
File_IOTypes.h
global.h
HwControl.h
iprexport.h
Isl_st_cdias.h
Isl_st_hardwaretree.h
Isl_st_kernel.h
Isl_st_pci.h
Isl_st_safety.h
Isl_st_safetydll.h
Isl_st_varan.h
LSL_STITask.h
Islpublictypes.h
MasterSettings.h
MultiTaskTypes.h
OsiBaseNew.h
SafetyCDIAS_Base.h
SafetyManager.h
StringDefs.h
SystemTypes.h
types.h
unit.h
UserDef.h
VaranManager.h
VaranManager_Base.h
Source Files
Network Files
Object Files
Library Files
Dias Drive Files
Connection Files
Lib
File
Global
Net
Class

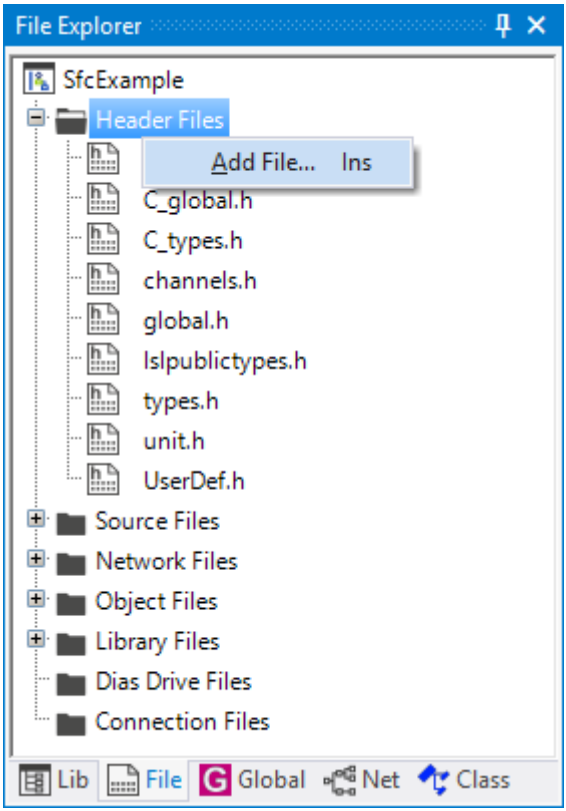
With a double click on the file, it is opened for editing. Once the file is opened, user specific definitions can be added.



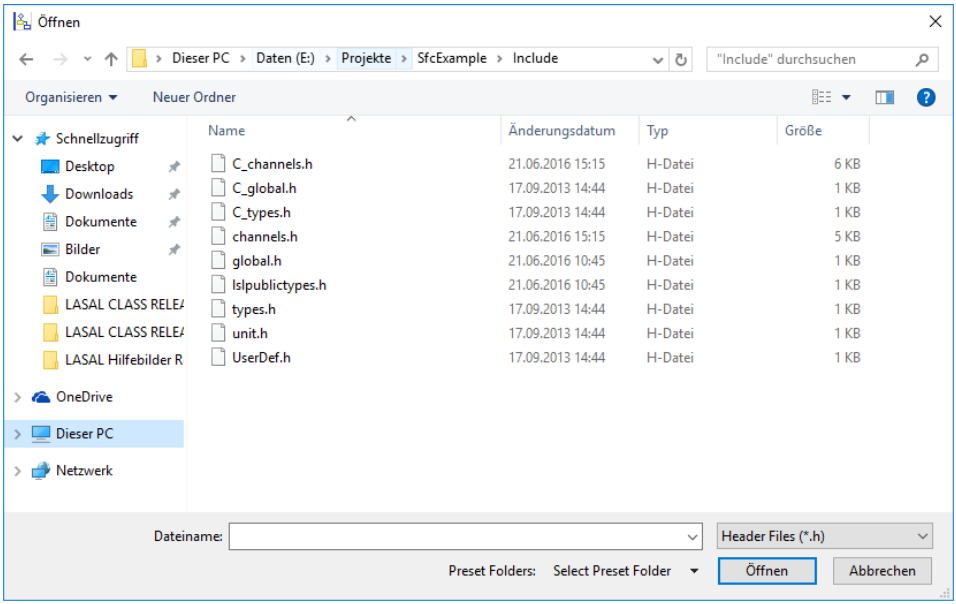
The **#include <Units.h>** definition shown above is available by default and imports units predefined by SIGMATEK, which can be used in the code (i.e.: mm, hour, min,...). Naturally, **Units.h** can be expanded or replaced by a user-defined header file.

3.7.3 Creating a Header File

A further possibility is the application of a new **header file**. In the project tree of the file explorer window, double click on **Header Files** to open the context menu and select **Add File...**.

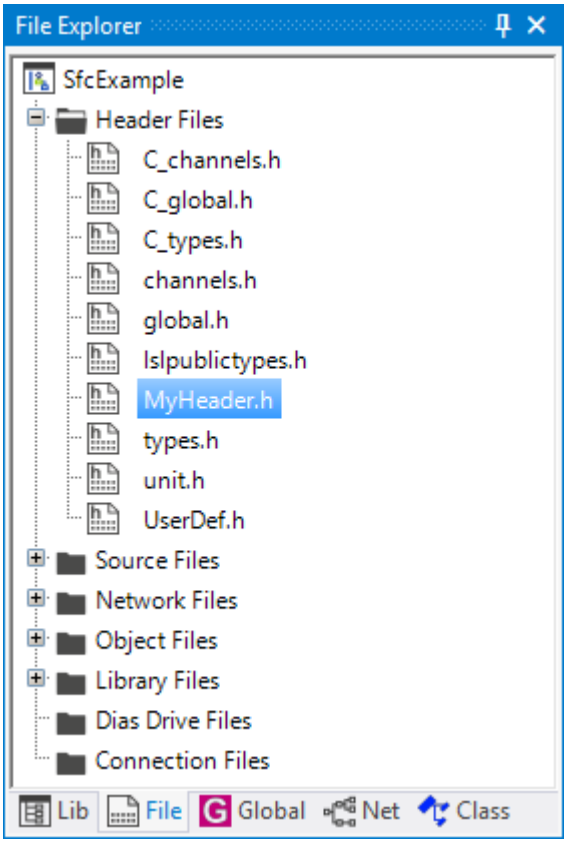


With this link the header file description is accessed, which can be directly linked to the options in the networks.

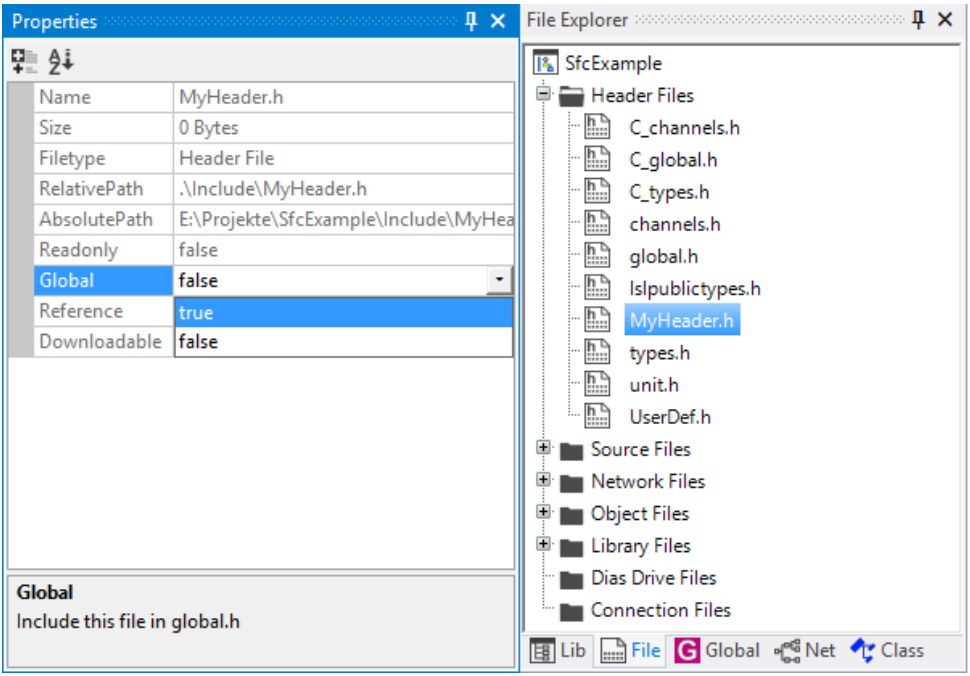


When selecting the target folder, the header file must be located in the **Source Folder** of the project or directly in the folder of the class, which uses the file. It is also important that the file name ends with a ***.h**.

Next, assign a name to the header file (**MyHeader.h**) and click on **Open**.



The newly created header file is now found in the Header Files Folder of the File Explorer. Open the new file with a double click to create a user specific definition. To use the header file globally, in the project tree, select the Files tab then select the header file to display the properties in the Properties window. Set Global to **True**. The header file is now included in the global.h header file.



3.7.3.1 A Note on Global Header Files

When using global header files, the constants can be used in every class in a project. The main disadvantage of this method is that the definitions are not transferred when a class is imported from one project to another. For this reason, linking global header files should be avoided.

The **Include** property in a global header file:

The status (**True**/**False**) of the property has no function in a global header file.

3.8 Interpreter in LASAL CLASS 2

An **interpreter** is a program that the program code (unlike assemblers or compilers) does **not** convert to a file, which can be executed in the system directly. An interpreter reads the source code, analyses or interprets it and executes the instructions immediately. The source code is analyzed during the program run time.

The advantage of an interpreter is that they function in any computer architecture, since the interpreter source code in translated there. The interpreter code can also be changed and run during the runtime.

3.8.1 Interpreter Applications

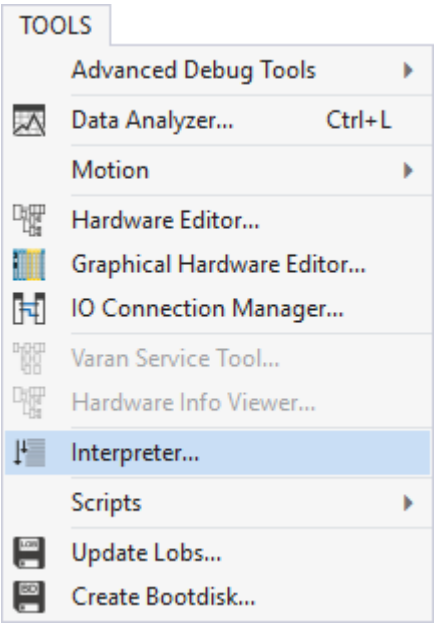
- Move processes for axle positioning
- Recipe management (i.e.: mixing materials and fluids)
- ...



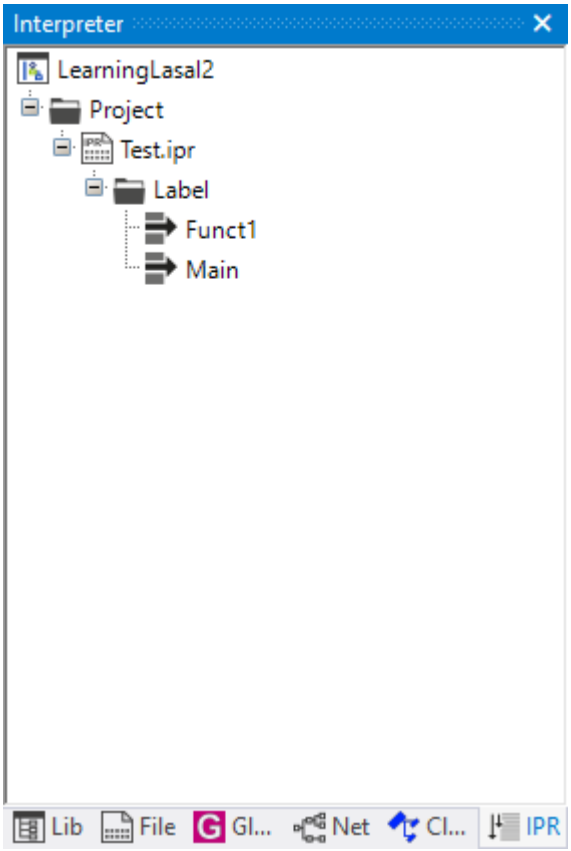
The basic requirement for creating an interpreter file is the importation of the RamEx class (found in the System library), which ensures that the IPR file can be sent and/or stored.

3.8.2 The Interpreter Tree

To activate the interpreter tree in LASAL CLASS 2, select **Interpreter** under the menu item **Tools** or click on the symbol in the symbol bar.



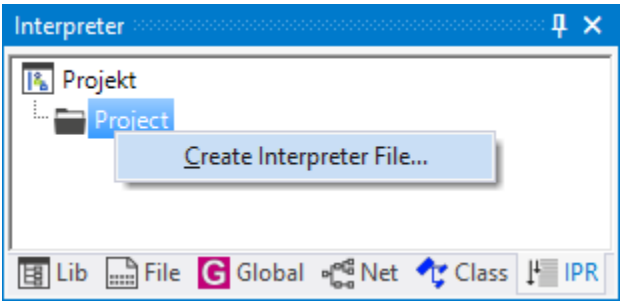
The Tree contains the overview of all the interpreters in the project as well as their contents. In online operation, the PLC interpreter is also shown. The following image shows, how an interpreter tree can look:



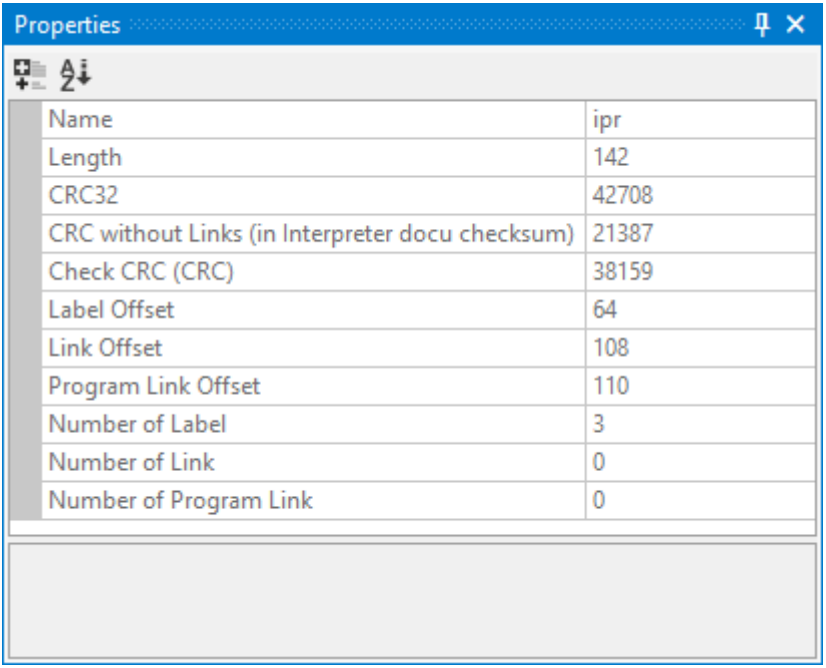
3.8.2.1 Layout of the Tree

3.8.2.11 Project

In this folder, all interpreters contained in the project are shown. New interpreters can be assigned using the context menu.

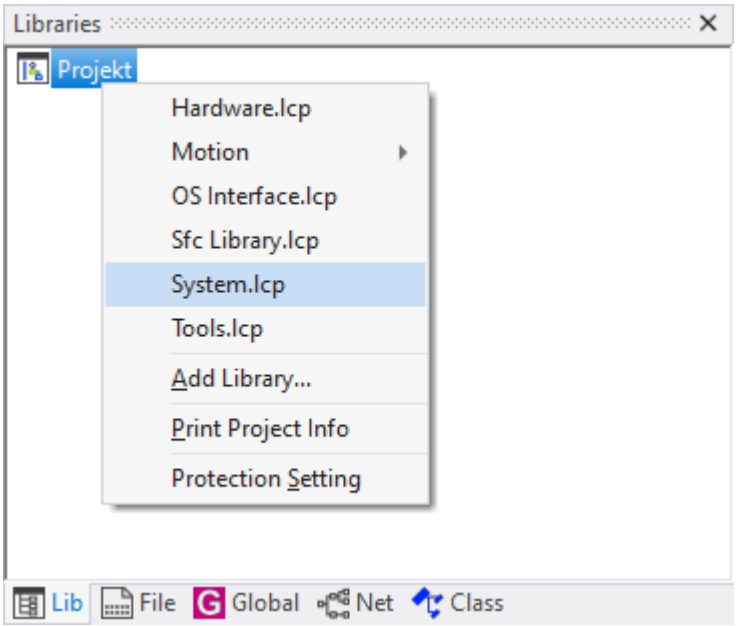


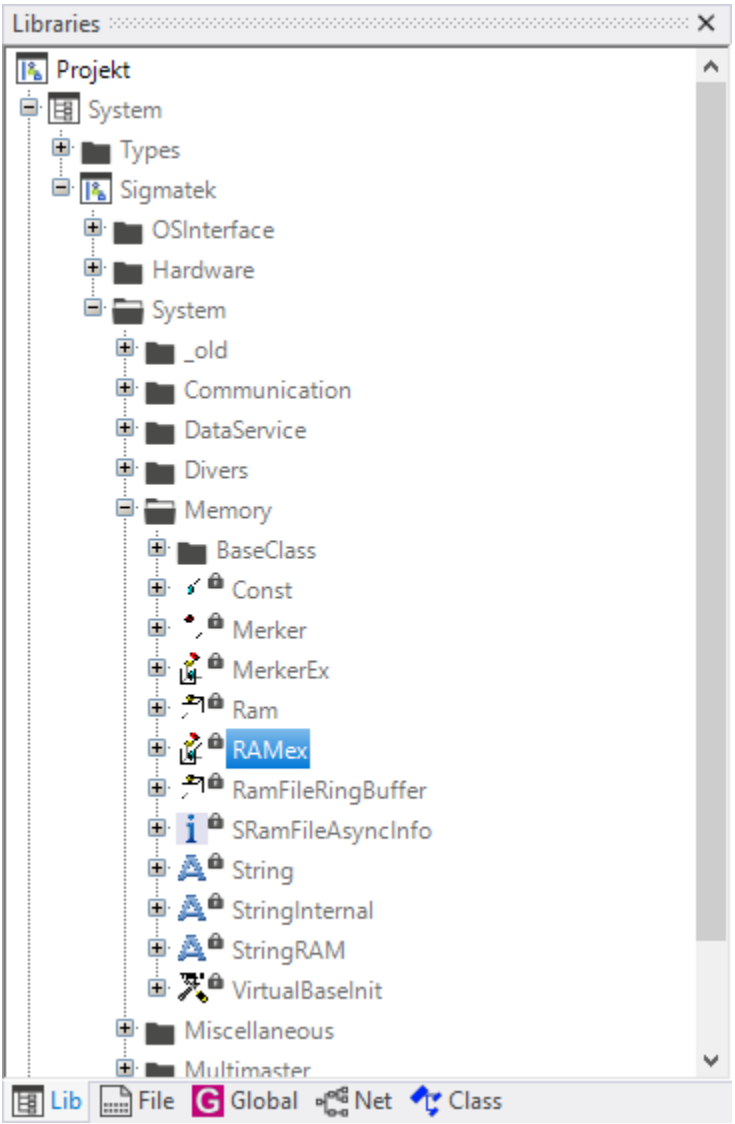
After an interpreter is selected, the information is shown in property browser.



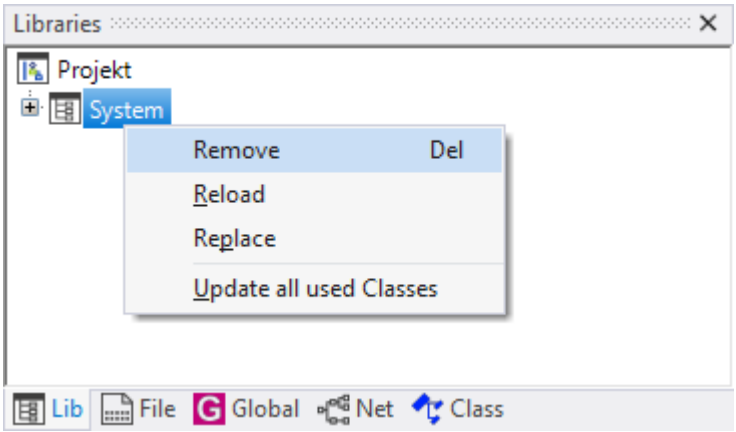
Name	ipr
Length	142
CRC32	42708
CRC without Links (in Interpreter docu checksum)	21387
Check CRC (CRC)	38159
Label Offset	64
Link Offset	108
Program Link Offset	110
Number of Label	3
Number of Link	0
Number of Program Link	0

Next, the **RamEx** class must be loaded from the **System** library.

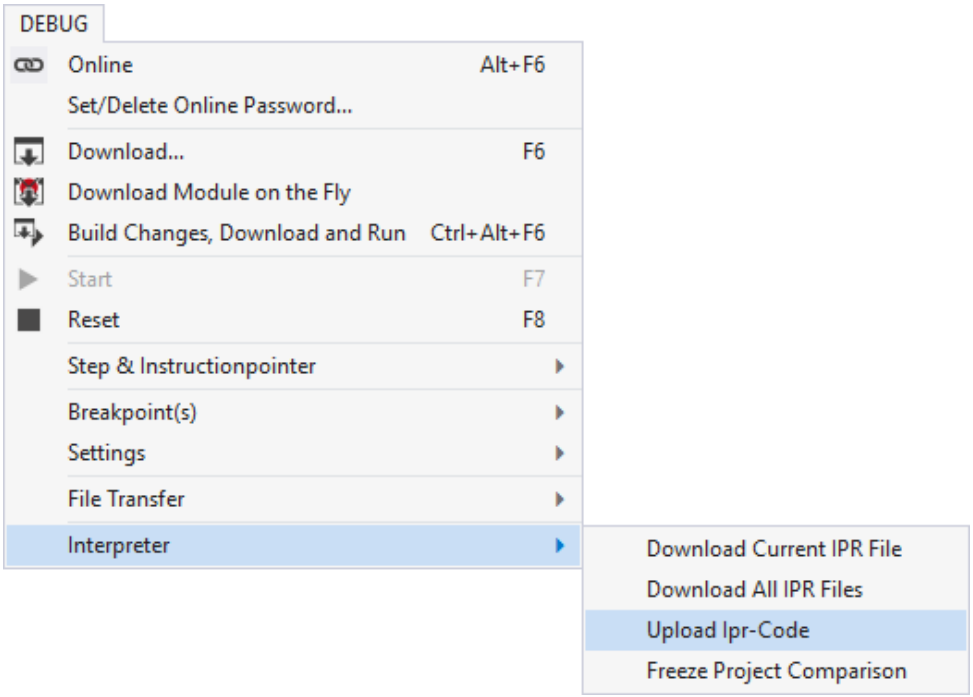

Libraries
Projekt
Hardware.lcp
Motion
OS Interface.lcp
Sfc Library.lcp
System.lcp
Tools.lcp
Add Library...
Print Project Info
Protection Setting
Lib
File
Global
Net
Class



If this is not done, the corresponding error message is triggered in the control during download. If nothing is nothing else is needed from the System library, it can be removed.



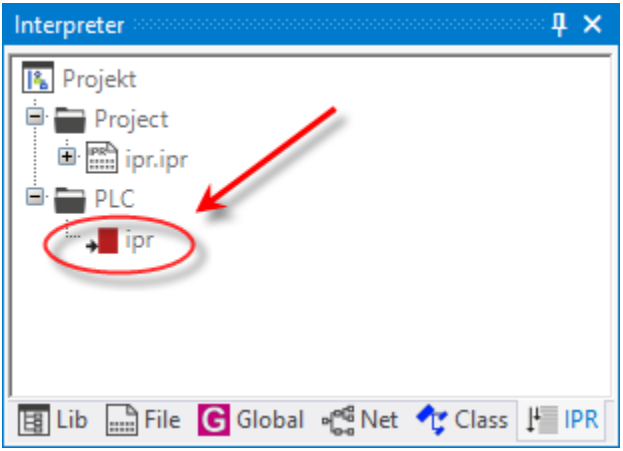
The interpreter must now be downloaded to the control. Here **Interpreter** is selected form the **Debug** menu and then **Upload Ipr-Code**.



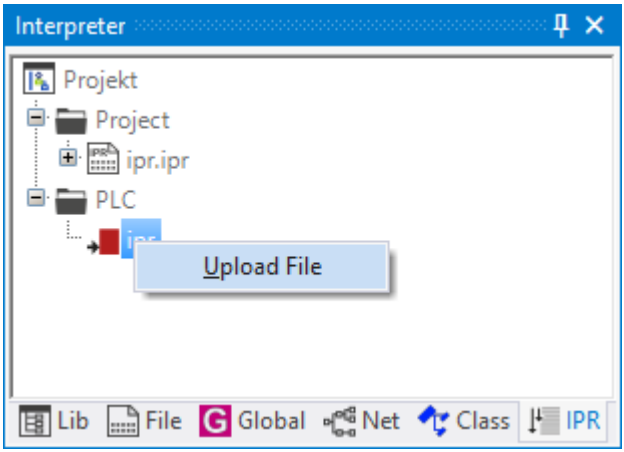
3.8.2.12 PLC

This folder is visible when there is online connection with the PLC. In this folder, all interpreters in the PLC are shown. In addition, whether the interpreter file is the PLC is identical to the one in the project can be determined.

If the Interpreter code was changed and compiled but not loaded into the PLC, the following symbol appears, which means that the code in the control does not match the code in the project.

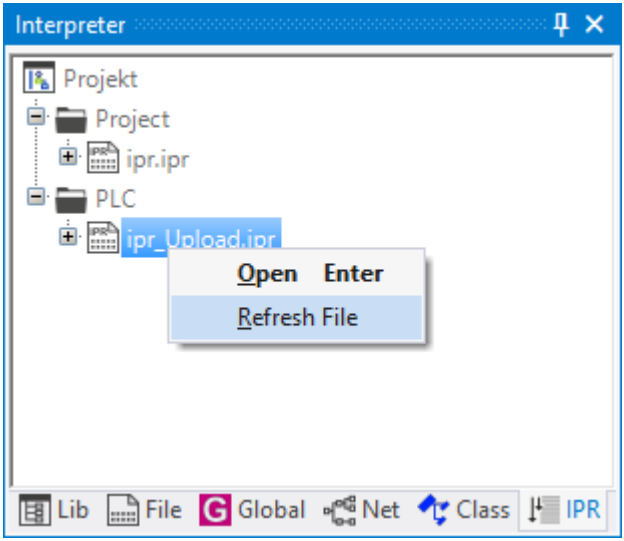


The interpreter file in the PLC can now be viewed by selecting the **Upload File** function from the corresponding context menu of the file (here: **test**):

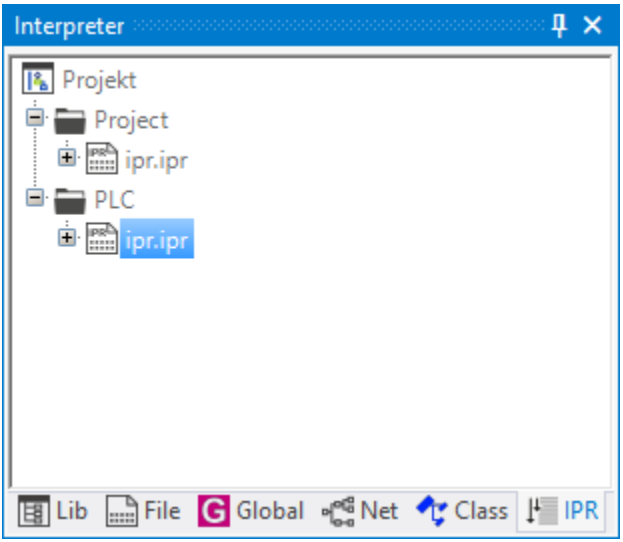


Details can be found under the Uploading Interpreters.

Through the context menu the existing interpreter(s) can be refreshed by selecting an interpreter file or the entire PLC folder.



The name of the interpreter file loses the addition _**Upload**, since the file in the project is now identical with that in the control:



Subdirectory Label

In this folder, all functions and jump points are shown. With a double click on the label, the program jumps to the position in the file.

Subdirectory Link

This folder contains all interpreter links as well as their line number in the code. With a double click on the link the position of the server in the network is located. Double clicking on the line number locates the position in the file.

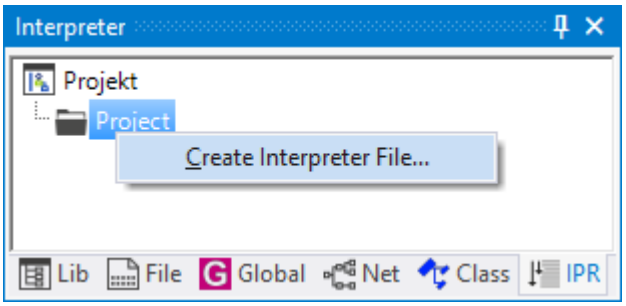
Subdirectory Program Link

In this file all function calls in their own or other interpreter and the line numbers in the code are shown. With a double click on the line number the position in the file is located.

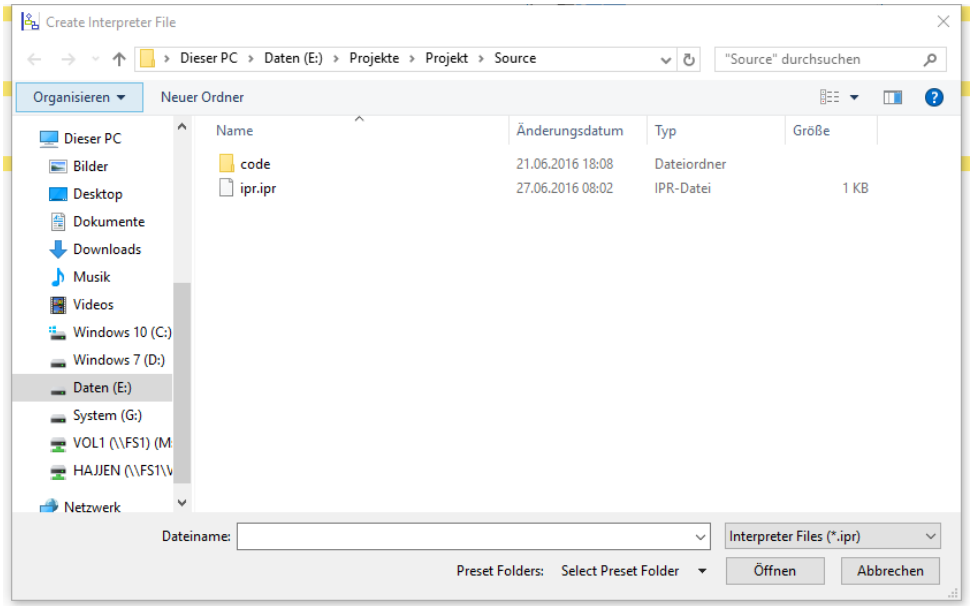
3.8.3 Applying an Interpreter in LASAL CLASS 2

If an interpreter is used in a project, the **RAMex** class must be linked to the project from the library if not already available. However, it is not necessary to place an object of this class in an object network.

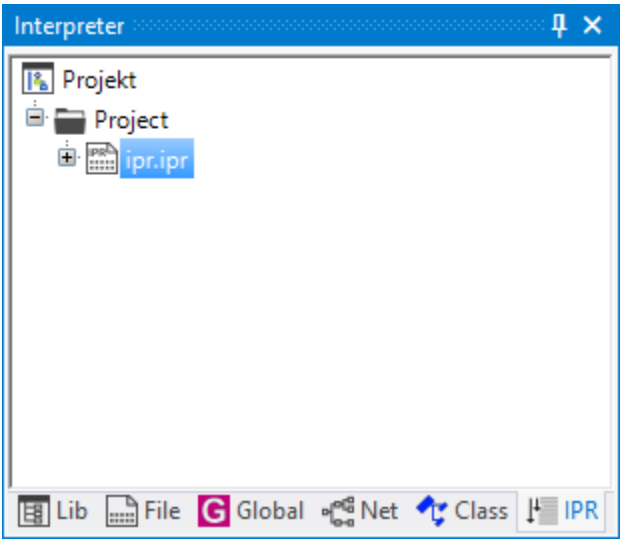
A new interpreter is generated when, in the context menu of the **Project** folder, **Create Interpreter File...** is selected from the interpreter tree.



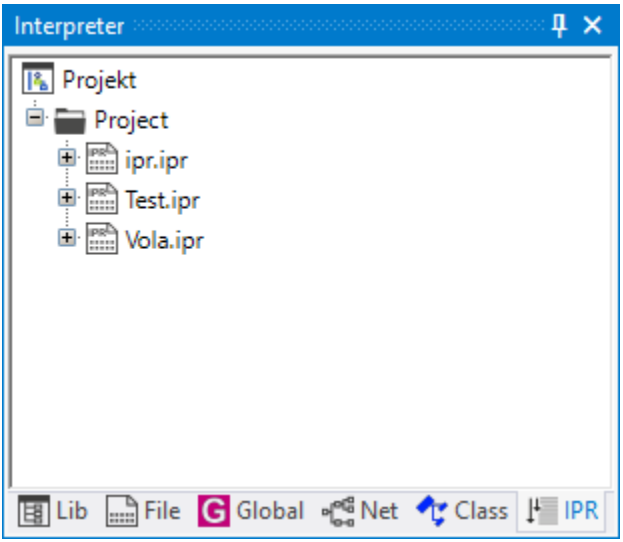
In the window the opens, assign a **file name** for the interpreter. The name assigned here is also the interpreter name.



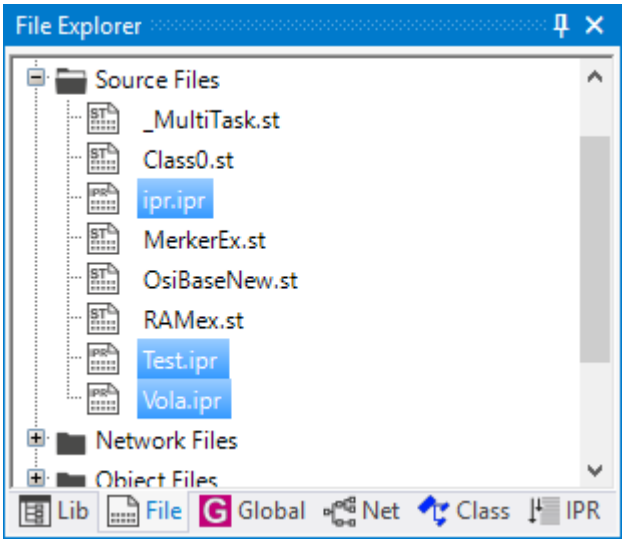
The new interpreter is then shown in the interpreter tree.



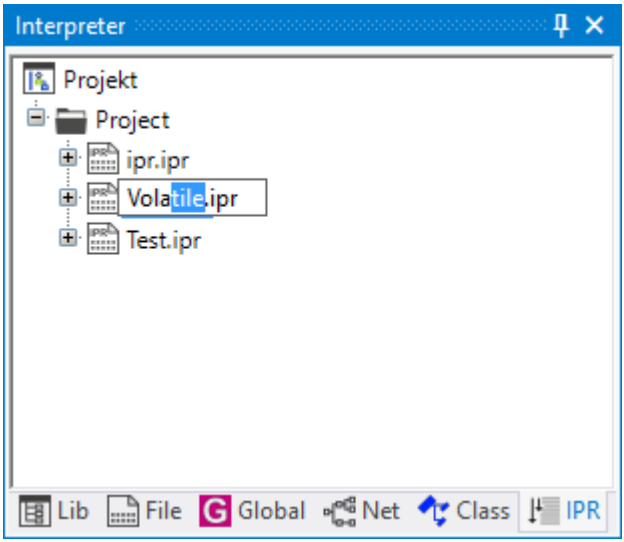
It is also possible to create several interpreters:



The stored interpreters are now located in the **Source Files** folder in the **File Explorer**:



While the interpreter files are displayed in alphabetical order and cannot be moved or renamed in the file explorer, it is possible to do so in the interpreter tree:



The interpreter files can be opened with a double-click on the name in the editor. An empty editoris shown if the interpreter is reassigned.



3.8.4 General Information on the Interpreter

3.8.4.1 Starting the Interpreter Automatically

The interpreter must have a function with the name **Main** in order to start automatically when the program is started. In the functions declaration of the interpreter, the **IPR** identification must be written between **FUNCTION** and the function name **Main** for an interpreter function.

```
FUNCTION IPR Main
  // Program of Main function …
END_FUNCTION
```



The main function is called by the loader once. Therefore, it is often necessary to program a loop (GOTO…)

3.8.4.2 Initializing the Interpreter

To initialize the interpreter when starting the control, an Init-Function is available to program in the interpreter code. Always after the interpreter code is changed and the code has been loaded into the control, it is called in the interpreter. Naturally, the Init function is run once during the initial start of the interpreter.

```
FUNCTION IPR Init
  // Initialize here …
END_FUNCTION
```

3.8.5 Interpreter Syntax

3.8.5.1 Label Definition

```
Labelname:
```

3.8.5.2 Subdirectory Definition

```
FUNCTION IPR Program name
END_FUNCTION
```

3.8.5.3 End Interpreter

An interpreter program is ended using this instruction:

```
END_PROG;
```

3.8.5.4 Exit Program

When the **RETURN** instruction is called, a function is exited.

```
RETURN;
```

3.8.5.5 Allocation

A value can be allocated to a server by the following method:

```
ObjectName.ServerName := (34 + 45)/23;
```



In the interpreter, the point before the division line is not read! Therefore, parentheses must be used.

When allocating a value to a server, the **Write()** method of the respective server is called.

```
ObjektName.ServerName2 := ObjektName.ServerName1;
```

In this case, the **READ()** method of the server (**ServerName1**) is called from the object (ObjectName) and for the allocation, the **WRITE()** from **ServerName2** is called from ObjectName.

3.8.5.6 Incrementation

Increments a server.

```
ObjektName.Counter ++;
```

3.8.5.7 Decrementation

Decrements a server.

```
ObjektName.Counter --;
```

3.8.5.8 IF - THEN Condition

If the condition is met, the instruction is executed.

```
IF condition THEN
// expression
END_IF
```

3.8.5.9 Instructions for Stopping an Interpreter

WAITFOR

The program is paused until the condition is met.

```
WAITFOR expression;
```

DELAY

The program is paused for specified length of time in ms.

```
DELAY (1000);
```

or is also written as:

```
DELAY 400;
```

Jumps, calls of functions, commands and other interpreters

GOTO

The program continues to run starting from the specified label.

```
GOTO Label;
```

Command calls

With command calls, NewInst commands can be called from objects (see NewInst).

Move is a command from **ObjectName**:

```
ObjektName.ServerName.Move(10, 20);
```

Here, the Move command from ObjecName is called with the parameters 10 and 20 and the ServerName.

To address the **first server** in the project, the object name alone is sufficient.

```
ObjektName.Move(12, 23);
```

If the command has a return parameter, it will appear as follows:

```
ret := Objekt.Command(para0, para1);
```

or for several return parameters in the command:

```
(ret1, ret2) := Objekt.Command(para0, para1);
```

Calling Subroutines

A subprogram is called in the interpreter simply by inserting the function name in parenthesis.

```
FunctionName();
```

Commentary

With two slashes before the text a single line is commented out of the program.

```
// This is a comment
```

Several lines can be commented out with the following method.

```
(*
   This is also a comment
```

```
    ...
    ...
*)
```

Starting an Additional Interpreter

An interpreter can be called using two different methods. To call an interpreter parallel to the current interpreter program:

```
*Interpretername::Sub-Programm();
```

With this method, the program waits until the current program is processed (**sequential call**).

```
Interpretername::Sub-Programm();
```

Loading the program in Memory

CHECK_FOR_LOAD

This function checks whether a new or active interpreter program is available and must be stored in memory.

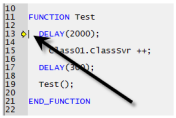
```
CHECK_FOR_LOAD();
```

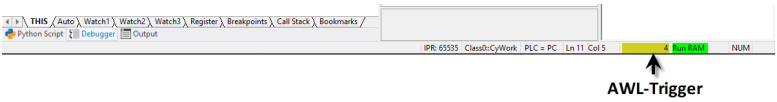
3.8.6 Interpreter Online Mode

If an interpreter is in the control and the program is running, the actual position is indicated with the yellow arrow  (Instruction Pointer).

AWL Trigger

If the Instruction list trigger  in the toolbar is activated, the cursor can be set on the desired line in the editor to see whether and how often a particular program line is processed. The number of times the specified line is run, if at all, can then be seen the status line.



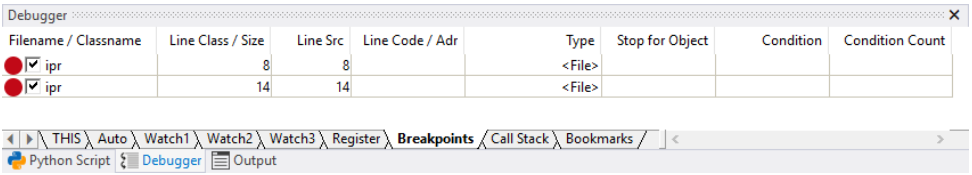


3.8.7 Debugging in the Interpreter

3.8.7.1 Setting Breakpoints in the Interpreter

A breakpoint is set with the **F4** key or the button  . When the interpreter encounters a breakpoint, it stops at the specified line before the command is executed.

To view the breakpoints in the interpreter, open the **Debug** folder and click the **breakpoints** tab.





It is possible to set several breakpoints in the interpreter.

3.8.7.2 Debugging in Single Step Mode

To run the interpreter in **single step mode**, a breakpoint must be set. If the interpreter encounters the breakpoint, the single step debugging mode is enabled.

3.8.7.3 Commands for Debugging in the Interpreter

Step Into

 from the toolbar or use the **F5** key

Jumps to the next instruction. With subroutines, the steps through the function.

Step Over

 from the toolbar or use the key combination **Alt + F5**

Jumps over the next instruction or function.

Step Out

 from the toolbar or use the key combination **Shift + F5**

Jumps out of the function in which the program is currently located.

Set Instruction Pointer

 from the toolbar

The instruction pointer is set on the line where the cursor is located.

Goto Instruction Pointer

 from the toolbar

The Cursor is set to the location of the instruction pointer.

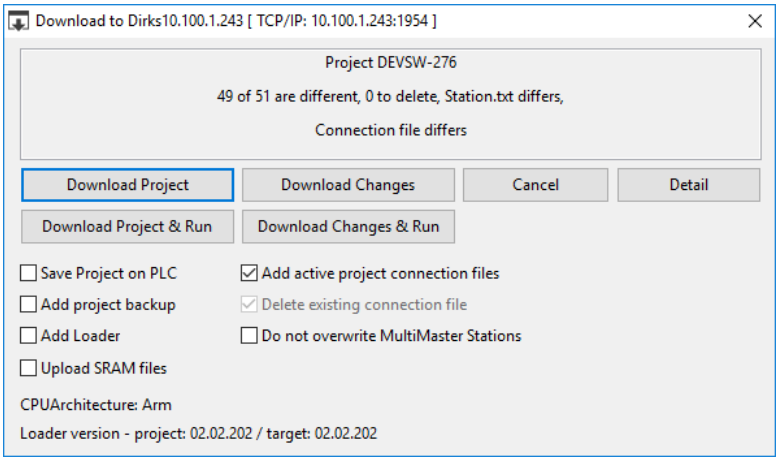


With the start button  or the **F7** key, the interpreter with a breakpoint runs in normal operation.

3.8.8 Downloading an Interpreter

3.8.8.1 Download in Reset Mode

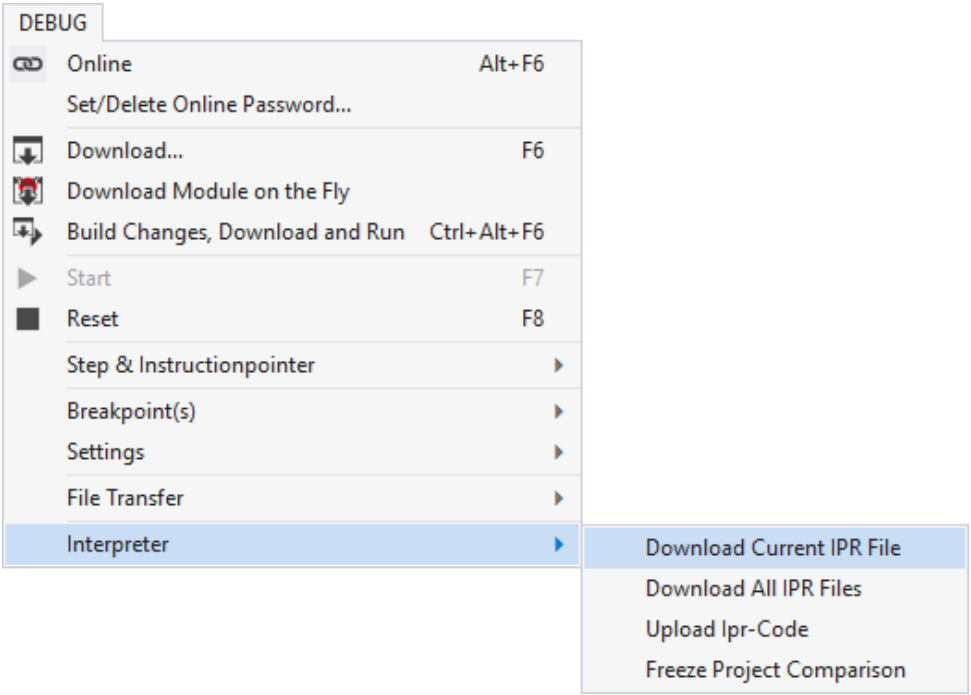
With usual download window, the interpreter can be loaded to the control (**F6** key or  button). Before the interpreter download starts, the control is stopped.



With this type of download, the **entire module** (the complete interpreter file) is loaded into the control. The interpreter code is exchanged with the next project start if they are not identical.

3.8.8.2 Downloading during Project Runtime

The second method to download an interpreter to a control is over **Debug, Interpreter, Download Current IPR file** or **Download All IPR Files** (if several interpreter files are available).



With this type of download, only the interpreter code and not the entire module is loaded. This code is then loaded automatically in the active memory of the interpreter through LASAL CLASS.

The project continues to run while downloading the interpreter!

Here, the following is attempted:

- With downloading a **time slot** of **3 seconds** is opened. During this time, a "**CHECK_FOR_LOAD()**" is called. The code in all non-updated interpreters is exchanged.

- If "**CHECK_FOR_LOAD()**" is not run in the first 3 seconds, an addition 3-second time slot is opened. When a label is encountered during this time, it is compared to the labels in the new IPR. If an identical label is found, the IPR file is exchanged and restarted with the new IPR label.

If these two variants generate errors, the interpreter must be stopped and the code exchanged. Before this occurs, the following message is shown, asking whether stopping the interpreter is actually desired:



3.8.9 Uploading an Interpreter

The interpreter code in the PLC to be processed can be loaded during runtime.

To upload the interpreter code, call **Upload All** from the context menu in **PLC** folder of the interpreter tree or select the **Upload File** instruction from the Files context menu. With this instruction, Interpreters can be uploaded together or individually. If the interpreter is already available, an update can be performed using the **Refresh instruction**.

A second method of uploading the interpreter code is through the **Debug** menu, selecting **Interpreter, Upload IPR-Code**. All interpreters are loaded to the PLC with this instruction.

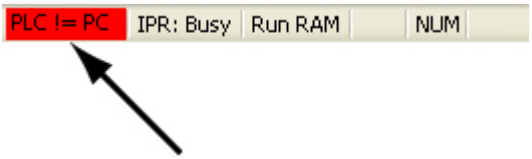


The uploaded interpreter code shifts the formatting (with blank lines, for example). The interpreter function, however, is not affected.

The file name indicates that an interpreter is uploaded (i.e.: **Interpreter_Upload.ipr**).

3.8.10 Freeze Project Comparison

Though the **Debug, Interpreter, Freeze Project Comparison** menus, the comparison between an uploaded project and the project already in the PLC can be disabled, In doing so, however, only the interpreter files comparison is deactivated. In the status line, the project will not be shown to differ if the interpreters are not the same. This can be necessary with interpreters, since they can change during runtime and differ from the project.

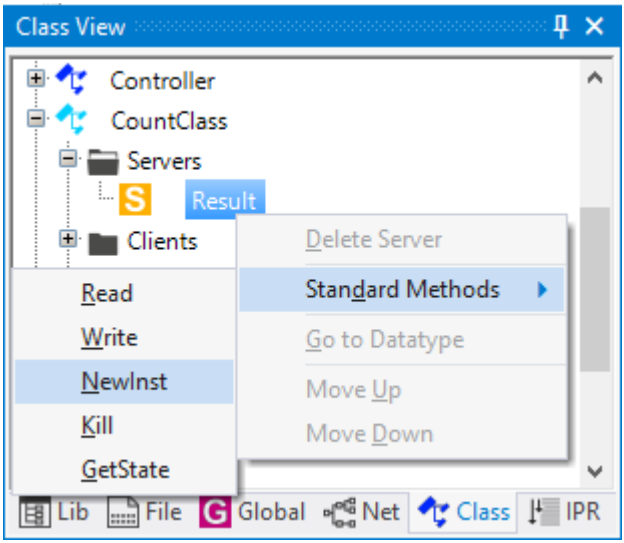


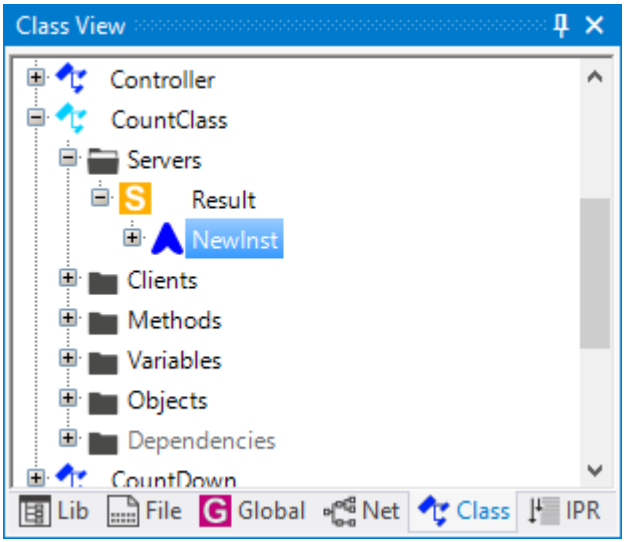
3.8.11 The Standard Methods NewInst(), GetState() and Kill()

3.8.11.1 General Information over the Standard Methods

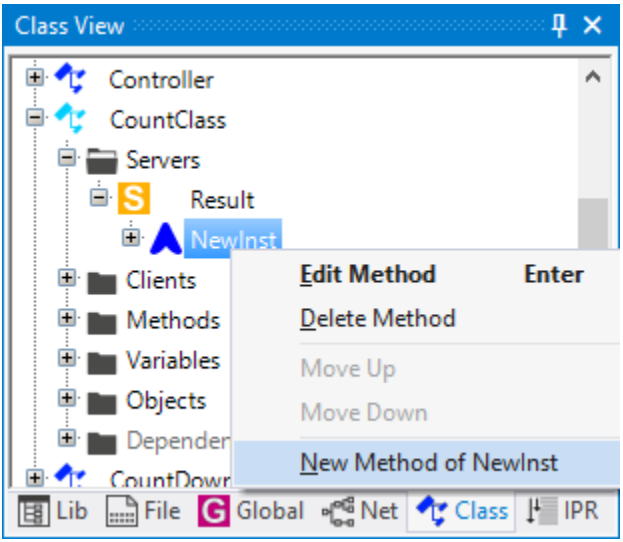
Using the standard methods, **NewInst** and **GetState**, Commands (functions) can be called from individual objects (except the **Kill** method).

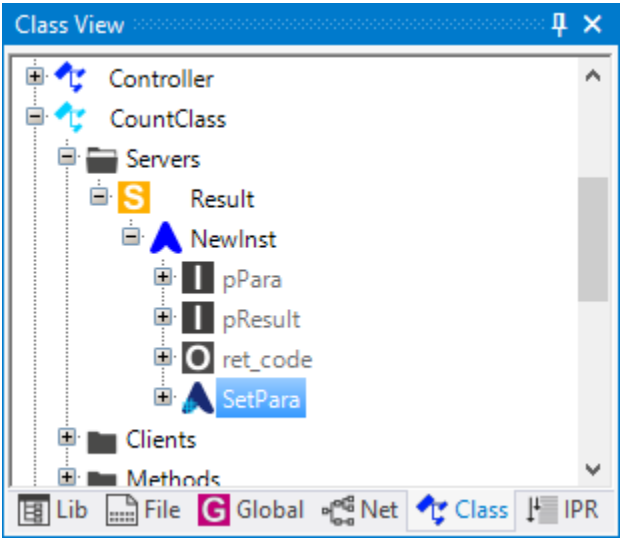
To assign the standard methods from the class server (first sever of a class “ClassSvr”), open the **Standard Methods** with a right click on the server and select the desired method.



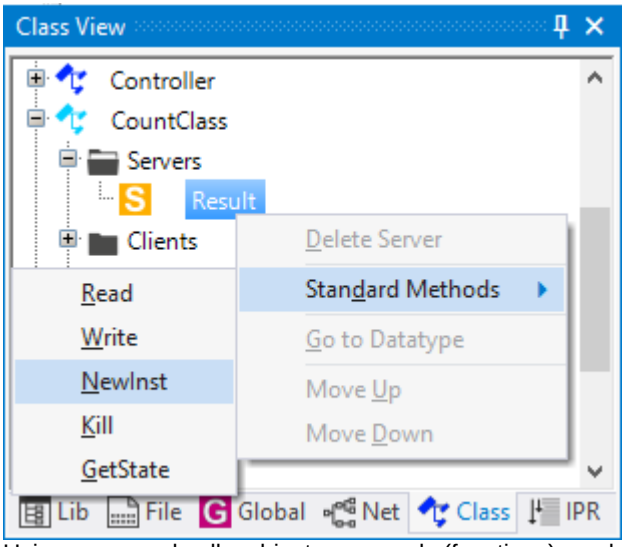


With a right click, a function can be defined for the standard methods selected, which can then be called by an interpreter.


Class View
Controller
CountClass
Servers
Result
NewInst
Clients
Methods
Variables
Objects
Depender
CountDowr
Edit Method
Enter
Delete Method
Move Up
Move Down
New Method of NewInst
Lib
File
Global
Net
Class
IPR


Class View
Controller
CountClass
Servers
Result
NewInst
pPara
pResult
ret_code
SetPara
Clients
Methods
Lib
File
Global
Net
Class
IPR

3.8.11.2 Applications for the NewInst() and GetState() Methods



Using command calls, object commands (functions) can be called.

To call a NewInstruction command from an interpreter, the object name written with the command as a function is sufficient.

```
e.g.:   Objekt0.MoveRel (1, 1000);     // Axle X will move with a 1 : 1000 ratio
                                       // Unit of 1000 is defined under Units:Length
```

If a return value is required, it should be written as follows:

```
ret  :=  Objekt.Command(para0, para1);
```

Or for several return parameters:

```
(ret1, ret2) := Objekt.Command(para0, para1);
```

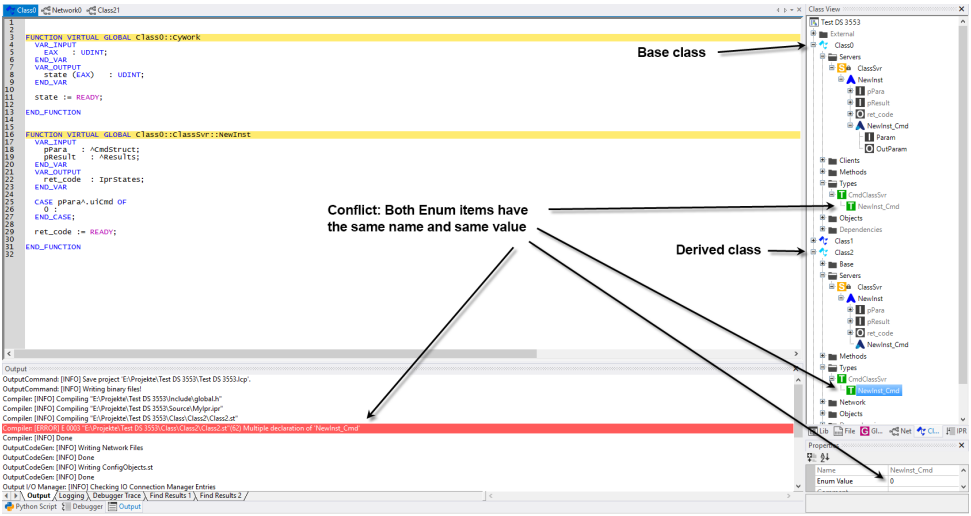
3.8.11.21 NewInst()

Whether the interpreter can continue (**ret_code = READY**) or not (**ret_code = BUSY**) is decided using the interpreter status. This can occur if an axle is bumped while moving and the interpreter can only continue when it's in position.

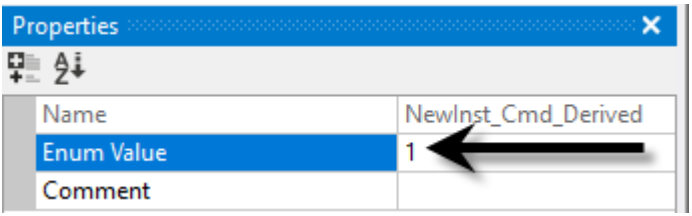
If an interpreter status is set to **BUSY**, the **GetState()** instruction is called instead of the "**NewInst()**" each time the interpreter is called. The **GetState()** instruction is called until **READY** or **ERROR** is returned. When an error is returned, the interpreter freezes and

must be restarted. The structures in the "**GetState()**" command are the same as those in "**NewInst()**".

If a derivation of a class has a NewInst and assigned functions, which was overwritten to the ClassSvr and the same commands assigned, a conflict occurs during compliation:



To solve this problem, simply rename the derived NewInst command. The compiler no longer sends an error message, but as long as the ENUM item values are the same in the derived and base classes, the program will not function. The ENUM item value can – and in this case must! – be changed by the user to avoid the conflict permanently:



3.8.11.22 GetState()

GetState() is constructed the same as **NewInst()**, however it is only called if the interpreter status is set to **BUSY** in the NewInst method. It is then called repeatedly until **READY** or **ERROR** is returned.

3.8.11.23 Kill()

If an interpreter status is "**BUSY**" or "**ERROR_BUSY**" and the **IprMgr_KillProgs()** function is called (the header file **IprMgr_KillProgs()** must be linked), the "**Kill**" function is called instead of "**NewInst()**" until the value **READY** or **ERROR** is returned.

3.8.11.3 Using the NewInst() and GetState() Methods via Multimaster

The **NewInst method on the host must return the status READY** for a call via multimaster. To ensure that the command is recognized as successfully executed, **an asynchronous implementation** of the NewInst command in connection with cyclic calls of the GetState method is **not possible**. The **GetState** method **of the host cannot be called from the consumer side**.

The **call of the GetState method** on the consumer side happens **locally** and serves **to wait for the response of the NewInst command** which is executed on the host side.

- The **GetState** method on the consumer side **returns BUSY until the execution of the NewInst command** on the host side **is completed**.
- If the **NewInst command was executed on the host and has returned READY, READY is also returned on the consumer side**.
- **ERROR_BUSY** is returned if the **NewInst command was executed on the host, but with a different result than READY**.
- **ERROR** is finally returned **if an error has occurred on the consumer side** (e.g. not online).

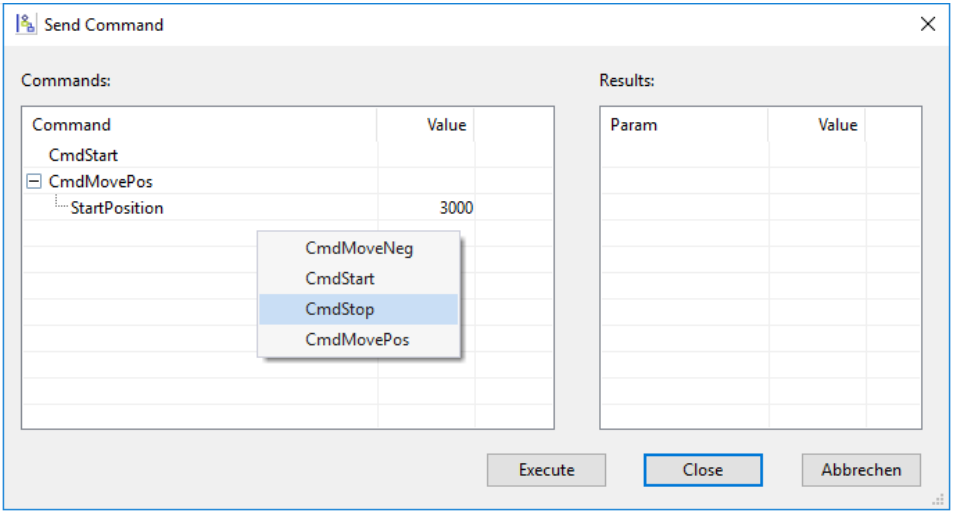
To ensure the correct assignment between a NewInst command by multimaster and the corresponding response, a **unique ID** is generated from the NewInst command ID and the addresses of the transfer parameters. For this reason it is **not possible to call several NewInst commands with identical ID** and identical transfer parameters **in parallel**. In this case, **after the first NewInst command the status ERROR is returned for all further NewInst commands** until the first command has been completely processed.

- **It is** therefore **returned** for a NewInst call on the consumer side **ERROR if no unique ID could be found** for the call - as described above - **or if the consumer side is not online**.
- **BUSY is returned** if the **NewInst command was accepted** and **the request is transmitted to the host**. The status must then be checked with GetState calls (see above)!

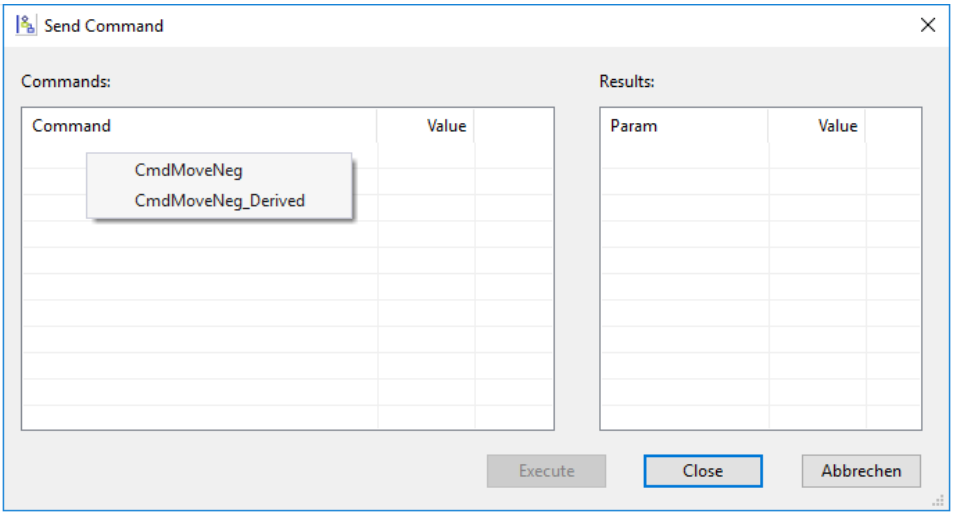
- **ERROR_BUSY is returned** if **a temporary error has occurred on the cosumer side** (e.g. too many NewInst requests).

3.8.12 Debugging with NewInst Commands

If the NewInst method of a class has commands, these commands can be called over a debug interface. The "**Send Command**" must be executed with a right mouse click in the context menu of the corresponding object. With a right mouse click in the dialog list, the predefined commands are entered with the parameters. The commands are then processed sequentially with a click on "**Execute**". The return value for the last command is shown on the right side.



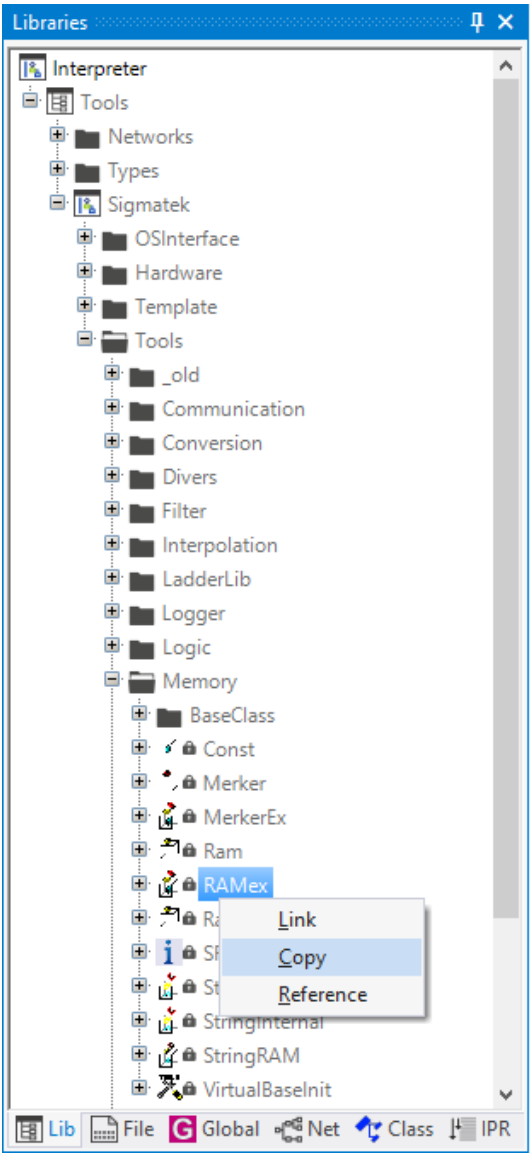
If a class is derived from another class with NewInst commands, the NewInst commands are inherited and can be selected from SendCommand.



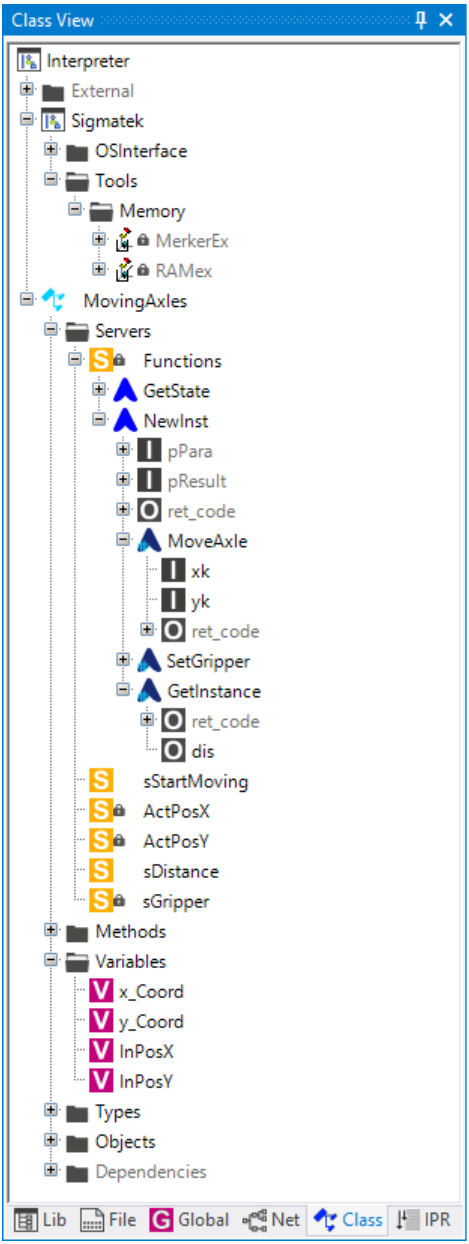
The following project shows the procedure for an axle and setting a gripper, whereby a gripper setting must be maintained until the axis is in position (**GetState()** method!).

Create a new project (File/New Project) with the name **Interpreter**.

Next, link a **RamEx** object to the project. It is not required to place an object in the network.


Libraries
Interpreter
Tools
Networks
Types
Sigmatek
OSInterface
Hardware
Template
Tools
_old
Communication
Conversion
Divers
Filter
Interpolation
LadderLib
Logger
Logic
Memory
BaseClass
Const
Merker
MerkerEx
Ram
RAMex
Link
Copy
Reference
StringRAM
VirtualBaseInit
Lib
File
Global
Net
Class
IPR

Create a class with the following configuration and set the **WriteProtect** property to **FALSE** for the servers that should be written to from outside the class (by an interpreter program).


Class View
Interpreter
External
Sigmatek
OSInterface
Tools
Memory
MerkerEx
RAMex
MovingAxles
Servers
Functions
GetState
NewInst
pPara
pResult
ret_code
MoveAxle
xk
yk
ret_code
SetGripper
GetInstance
ret_code
dis
sStartMoving
ActPosX
ActPosY
sDistance
sGripper
Methods
Variables
x_Coord
y_Coord
InPosX
InPosY
Types
Objects
Dependencies
Lib
File
Global
Net
Class
IPR

All servers used are of data type **DINT**. The variables **Xk** and **Yk** are of type **DINT** and **InPosX** and **InPosY** are of type **BOOL**.

Create a network (here: **IPRNet**) and an object of the class (here: **Axle**).

The following is true for the case, that in the NewIsnt properties the property “Use hidden NewInst” is set to **false**:

Double click on the NewInst method and insert the following program code:

```
FUNCTION VIRTUAL GLOBAL MovingAxles::Functions::NewInst
  VAR_INPUT
    pPara    : ^CmdStruct;
    pResult    : ^results;
  END_VAR
  VAR_OUTPUT
    ret_code  : iprStates;
  END_VAR

  ret_code := READY;

(*
  ***************************************************************************
  ************************                           ************************
  ***********************    step-by-step switch     ************************
  ************************                           ************************
  ***************************************************************************
*)
  CASE pPara^.uiCmd$CMDFunctions OF


    MoveAxle:   // fetch coordinates and start axle moving
                InPosX := false;
                InPosY := false;
                sGripper := 0;
                x_Coord := pPara^.aPara[0];
                y_Coord := pPara^.aPara[1];

                ret_code := BUSY; //getstate() is called


    SetGripper: //set the gripper
                if sGripper = 1 then
                   sGripper := 0;
                else
                   sGripper := 1;
                end_if;


    GetDistance: //Calculate distance to starting point

                pResult^.uiLng := 6;  // total length of used buffer incl. 2 bytes for uiLng
                pResult^.aData[0]$DINT :=  to_dint((sqrt(ActPosX**2 + ActPosY**2)));

  END_CASE;

END_FUNCTION

FUNCTION VIRTUAL GLOBAL MovingAxles::Functions::GetState
  VAR_INPUT
    pPara    : ^CmdStruct;
    pResult    : ^results;
  END_VAR
  VAR_OUTPUT
    ret_code  : iprStates;
  END_VAR

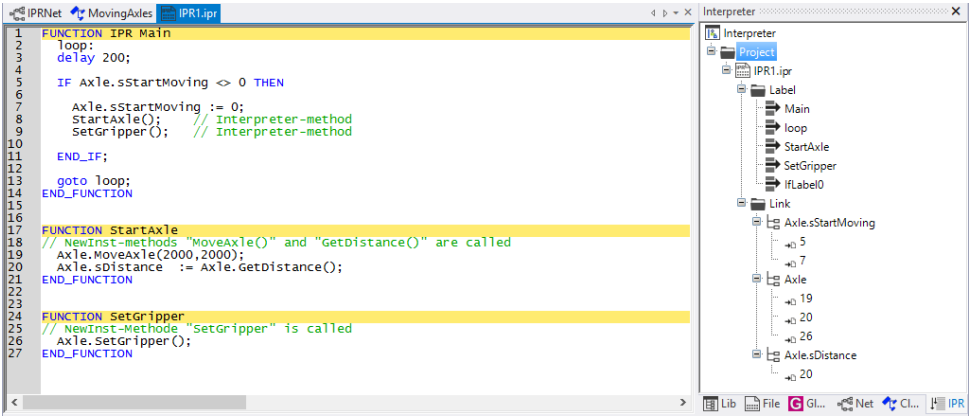
  //Move X-axis
  if ActPosX < x_Coord then
     ActPosX += 1;
  elsif ActPosX > x_Coord then
     ActPosX -= 1;
  else
     InPosX := true;
  end_if;

  //move Y-axis
  if ActPosY < y_Coord then
     ActPosY += 1;
  elsif ActPosY > y_Coord then
     ActPosY -= 1;
  ELSE
     InPosY := true;
  end_if;

  //check, if both axis are in position
  if InPosX & InPosY then
     ret_code := READY;
  ELSE
     ret_code := BUSY;
  end_if;

END_FUNCTION
```

In addition, create an interpreter file (here: IPR1) and expand the following program code as shown on the left side of the screenshot:



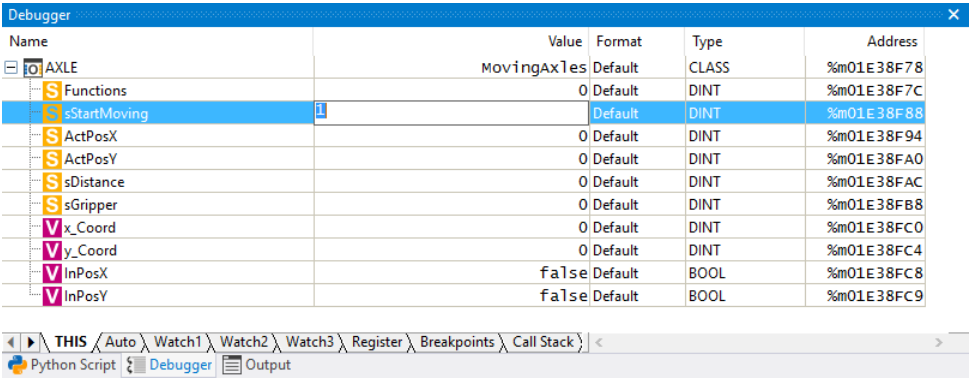
The entries in Label and Link folders occur automatically .The numbers under link indicate the line numbers in which the links are found.



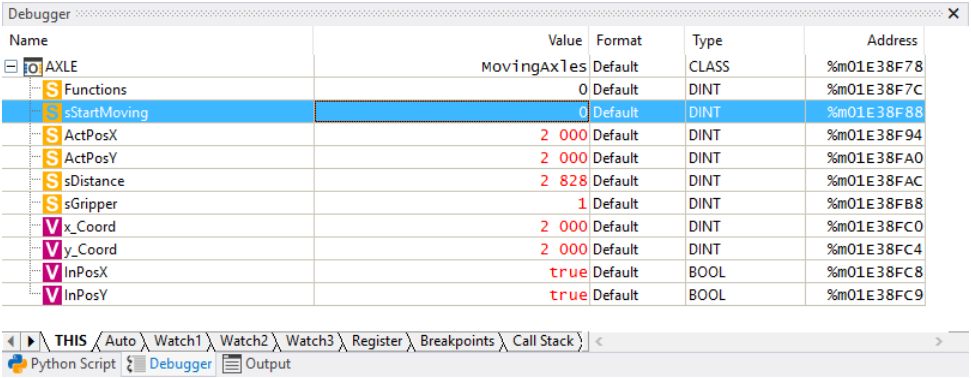
When assigning the functions, typing can be reduced by selecting "**Add Function...**" from the context menu of the **Label** folder.

Next download the entire project into the PLC and start it.

Now enter a value unlike 0 in the **sStartMoving** server to start the program:



After the program is run, the following results are shown:



After the Axle is in position (**InPosX** and **InPosY** = **TRUE**), the gripper is set (**sGripper** = 1).

It is also possible to work with the additional functionality “Use hidden NewInst”. In this case the functions NewInst() and GetState() are automatically generated “hidden” in a hidden area. In any case it has to be cared, that

- the status of the “Use hidden NewInst” has to be set before creating the first NewInst method, as this is no longer possible with existing NewInst methods; and
- the function has to be visible, if GetState() should be programmed. This is done in the according NewInst method with the property Implement Getstate, which has to be set to true.

The automatically generated and hidden implementations of the NewInst() and GetState() functions can be found in the *.st file of the class at the end of the program code in a separate hidden area.

Explanation for the interpreter program code

```
FUNCTION IPR Main
  loop:
  delay 200;

  IF Axle.sStartMoving <> 0 THEN

      Axle.sStartMoving := 0;
      StartAxle();  // Interpreter-method
      SetGripper();  // Interpreter-method

  END_IF;

  goto loop;
END_FUNCTION
```

This function is started by the loader only once (**IPR Main**). To keep the function "alive", the **loop** is required.

```
delay 200;
```

Time delay of 200 ms

```
IF Axle.sStartMoving <> 0 THEN
```

The IF-loop waits until the start for the axle movement is given (**Axle.StartMoving <> 0**).

```
StartAxle();
```

Here, the interpreter function StartAxle() is called, in which the NweInst methods MoveAxle() and GetDistance() are called.

```
SetGripper();
```

Here, the interpreter function SetGripper() is called, in which the NewInst method of the same name is called.

As soon as of one of the NewInst methods is called from the interpreter, the program jumps to the **MovingAxles:: Functions:: NewInst**.

```
FUNCTION VIRTUAL GLOBAL MovingAxles::Functions::NewInst
        VAR_INPUT
                pPara    : ^CmdStruct;
                pResult : ^results;
        END_VAR
        VAR_OUTPUT
                ret_code : iprStates;
        END_VAR

  ret_code := READY;

        CASE pPara^.uiCmd$CMDFunctions OF
```

Explanation for the LASAL CLASS program code:

```
ret_code := READY;
```

After a Newinst function has be executed, this interpreter status indicates that it was processed correctly and a further function can be run. Additional interpreter statuses are defined in the **RTOS_INTERPRETER.h** header file

```
CASE pPara^.uiCmd$CMDFunctions OF
```

In the CASE instruction, the input parameter **pPara^.uiCmd** with which the NewInst methods is meant, is detected. Casting with the **CMDFunctions** type has the advantage that the CASE steps do not have to be labeled with 0, 1, 2, rather the function names that were assigned in the class tree under the NewInst can be used. The Type name made up of **CMD** and the **server name**.

```
MoveAxle:   // fetch coordinates and start axle moving
                InPosX := false;
                InPosY := false;
                sGripper := 0;
                x_Coord := pPara^.aPara[0];
                y_Coord := pPara^.aPara[1];
```

```
                ret_code := BUSY;
```

Here, among other things, the input parameters (**xk, yk**) transferred by calling the methods in the interpreter in the **x_Coord** and **y_Coord** member variables are read. Since **aPara** is an array consisting of 20 entries of type DINT, a maximum of 20 parameters can be defined. aPara is defined in the **RTOS_INTERPRETER.h** header file. Because **ret_code := BUSY** was set, the program automatically jumps to the **GetState()** method and runs it until ret_code is set to READY. The interpreter will then continue and the next instructions and/or methods are processed

```
SetGripper: //set the gripper
                if sGripper = 1 then
                   sGripper := 0;
                else
                   sGripper := 1;
                end_if;
```

Here, the status of the **sGripper** is toggled.

```
GetDistance: //Calculate distance to starting point
      pResult^.uiLng := 6; //total length of used buffer incl. 2 bytes for uiLng
      pResult^.aData[0]$DINT := to_dint((sqrt(ActPosX**2 + ActPosY**2)));
```

The variable **pResult** of the NewInst method is defined in the **RTOS_INTERPRETER.h** header file.

In **pResult^.uiLng**, the number of return parameters for the corresponding methods (here: **GetDistance** has 1 return parameter **dis**) are determined; an additional 2 bytes are for value of uiLen itself. Because the return values are stored in **pResult^.aData** and **aData** is an array of type **USINT**, the value **6** must be entered in pResult^.uiLen since it has a 4-byte value (DINT) as a return value.

```
FUNCTION VIRTUAL GLOBAL MovingAxles::Functions::GetState
        VAR_INPUT
                pPara : ^CmdStruct;
                pResult : ^results;
        END_VAR
        VAR_OUTPUT
                ret_code : iprStates;
        END_VAR

  …
  //check, if both axis are in position
  if InPosX & InPosY then
     ret_code := READY;
  ELSE
     ret_code := BUSY;
  end_if;

END_FUNCTION
```

In this method, the axle is moved into the target position. As long as it has not reached the end position, **ret_code = BUSY** is returned. If the Boolean variables **InPosX** and **InPosY** are both TRUE, **ret_code = READY** is returned and the program exits **getstate()**.

3.8.13 IPR Editing during Runtime

LASAL provides functions that allow editing of an interpreter during runtime (start, stop, exchange). To use the functions in a user-defined class, the **LoaderItf.h** header file must be linked from the **Include folder**.

For some functions first a reference to the class ProgIp has to be established (part of the loader). The class **ProgIp** is declared in the file "**RTOS_IprIntern.h**".



Incorrect usage of the functions listed below can lead to a **fatal error** in the control!

3.8.13.1 Starting an Interpreter

If an interpreter is not declared in **MAIN**, it must be started using **PROGRAM**. If **IPR** is started externally, they always run parallel. If IPR are started from another IPR, it can be differed between **parallel** and **sequential start**.

Starting of parallel running programs

```
FUNCTION GLOBAL ProgIp_StartProgram
VAR_INPUT
    pp : ^ProgIp;        // Return of IprMgr_GetInterpreter(nProgNo)
    uiPrgNo : UINT;      // Number of the IPR, starting with 0
    uiLabelNo : UINT;    // Label No.: 0 = first line, IPR name is also the Label
END_VAR
VAR_OUTPUT
    state : IprStates;  // see type definition
END_VAR;
```

Starting sequentially running programs

```
FUNCTION GLOBAL ProgIp_RunProgram
VAR_INPUT
    pp : ^ProgIp;        // Return of IprMgr_GetInterpreter(nProgNo)
    uiProgNo : UINT;     // Number of the IPR, starting with 0
    uiLabelNo : UINT;    // Label No.: 0 = first line, IPR name is also the Label
END_VAR
VAR_OUTPUT
    state : IprStates;  // see type definition
END_VAR
```

The StartProgram and RunProgram functions differ only by when one interpreter is started from another.

e.g.:

```
VAR
```

```
     pIprT : ^ProgIp;
     state : IprStates;
END_VAR
     pIprT := IprMgr_GetInterpreter(0);        // Determine address of the first IPR
     state := ProgIp_StartProgram(pIprT, 0, 0) // Starting the IPR from the beginning
```

3.8.13.2 Stopping an Interpreter

```
FUNCTION GLOBAL ProgIp_StopProgram
VAR_INPUT
    pp : ^ProgIp;       // Return of IprMgr_GetInterpreter(nProgNo)
    uiProgNo : UINT;    // Interpreter number
END_VAR
VAR_OUTPUT
    state : IprStates;  // see type definition
END_VAR
```

e.g.:

=> Before an IPR is exchanged it must be stopped.

```
FUNCTION GLOBAL User_CreateIPR::StopIpr
VAR_INPUT
    uiIprNo : UINT;                         // IPR number
END_VAR
VAR_OUTPUT
    state : IprStates;                      // Interpreter status
END_VAR
VAR
    pIprT : ^ProgIp;
END_VAR
    pIprT := IprMgr_GetInterpreter(uiIprNo); // Determine Interpreter address
    state := ProgIp_StopProgram(pIprT, uiIpr
END_FUNCTION
```

3.8.13.21 Emergency Stop of an Interpreter

```
FUNCTION GLOBAL IprMgr_KillProgs
```

After this function is called, the object in which a "**NewInst**" has been programmed and its status set to "**BUSY**" or "**ERROR_BUSSY**" calls the virtual "**Kill()**" function.

3.8.14 Setting the Instruction Pointer

LASAL has no predefined function for setting the instruction pointer. This must be programmed by the user as in the following example:

```
VAR_INPUT
    uiIprNo : UINT;                         // IPR number
    udLNr : UDINT;                          // Line number
END_VAR
VAR
    pIprT : ^ProgIp;
```

```
    pIprHeadT : ^IprHead;
    pSubPrg : ^SubPrgHead;
    pInst : ^UINT;                          // Pointer to command line
END_VAR
    pIprT := IprMgr_GetInterpreter(uiIprNo);// Determine Interpreter address
    pIprHeadT := PropIp_GetHead(pIprT);     // Pointer to Interpreter head
    pSubPrg := ProgMgr_GetProgram(uiIprNo); // Pointer to IPR code, not OPCODE
    pInst := (pSubPrg + (pSubPrg^.udOffCode)// Set pInst to "FUNCTION"
    pIprHeadT^.pCode := pInst$^instruct;    // Set the actual line to be processed
END_FUNCTION
```

3.8.15 Determining the Interpreter Address

The return value is a pointer to the Interpreter object with the assigned number.

```
FUNCTION GLOBAL IprMgr_GetInterpreter
VAR_INPUT
    nProgNo : UDINT;   // Interpreter number
END_VAR
VAR_OUTPUT
    pp : pVoid;
END_VAR
```

The return value is a pointer to the interpreter file (see type definition "**SubPrgHead**").

```
FUNCTION GLOBAL ProgMgr_GetProgram
VAR_INPUT
    nProgNo : UDINT;   // Interpreter number
END_VAR
VAR_OUTPUT
    pp : ^SubPrgHead;
END_VAR
```

The return value is a pointer to the expanded interpreter file (see type definition "**NewSubPrgHead**").

```
FUNCTION GLOBAL ProgMgr_GetProgramPtr
VAR_INPUT
    nProgNo : UDINT;
END_VAR
VAR_OUTPUT
    pp : ^pNewSubPrgHea
END_VAR
```

The status of the interpreter can be determined using the structure IprHead (see type definition "**iprStates**"). Provides a pointer to the status marker of the interpreter.

```
FUNCTION GLOBAL ProgIp_GetHead
VAR_INPUT
    pp : ^ProgIp;      // Return of IprMgr_GetInterpreter(nProgNo)
END_VAR
VAR_OUTPUT
    pHead : ^IprHead;
END_VAR
```

e.g.:

```
// Determine Interpreter address
pIprT := IprMgr_GetInterpreter(uiIprNo);
pIprHead := ProgIp_GetHead(pIprT);
```

3.8.16 Breakpoints

3.8.16.1 Setting Breakpoints

Insert a breakpoint to the specified offset, given in bytes and based on the code (Code beginning = „Function“).

```
FUNCTION GLOBAL ProgIp_SetBreakpoint
VAR_INPUT
    pp : ^ProgIp;       // Return of IprMgr_GetInterpreter(nProgNo)
    dOffset : UDINT;    // Byte offset based on code
END_VAR
VAR_OUTPUT
    bBpValid : BOOL;    // TRUE = OK, FALSE = ERROR
END_VAR
```

e.g.:

```
// Determine Interpreter address
pIprT := IprMgr_GetInterpreter(nProgNo);
// Sets the breakpoint
bBpValid := PropIp_SetBreakpoint(pIprT, dOffset);
```

3.8.16.2 Clearing Breakpoints

Clearing a Breakpoint.

```
FUNCTION GLOBAL ProgIp_DeleteBreakpoint
VAR_INPUT
    pp : ^ProgIp;       // Return of IprMgr_GetInterpreter(nProgNo)
    dOffset : UDINT;    // Byte offset based on code
END_VAR
VAR_OUTPUT
    bBpValid : BOOL;    // TRUE = OK, FALSE = ERROR
END_VAR
```

e.g.:

```
// Determine Interpreter address
pIprT := IprMgr_GetInterpreter(nProgNo);
bBpValid := ProgIp_DeleteBreakpoint(pIprT, BreakOffsets[dOffset]);
```

3.8.16.3 Single Stepping

This function sets the code pointer to the next instruction but does not yet function for program calls and Return.

```
FUNCTION GLOBAL ProgIp_MakeSingleStep
VAR_INPUT
    pp : ^ProgIp;   // Return of IprMgr_GetInterpreter(nProgNo)
END_VAR
```

e.g.:

```
// Determine Interpreter address
pIprT := IprMgr_GetInterpreter(nProgNo);
ProgIp_MakeSingleStep(pIprT);
```

3.8.16.4 Continuing Normal Operation after Single Step

If the interpreter is in single step, it is started (status set to ready).

```
FUNCTION GLOBAL ProgIp_RunInterpreter
VAR_INPUT
    pp : ^ProgIp;   // Return of IprMgr_GetInterpreter(nProgNo)
END_VAR
```

e.g.:

```
// Determine Interpreter address
pIprT := IprMgr_GetInterpreter(nProgNo);
ProgIp_RunInterpreter(pIprT);
```

3.8.17 Determining the Current Line Number for Monitoring

The current line to be processed, to monitor for example, must be programmed by the user.

i.e.: => Calculates the actual line position of the running IPR

```
FUNCTION GLOBAL User_CreateIPR::GetActLineNr
VAR_INPUT
    udIprNr : UDINT;                        // IPR number: starting with 0
END_VAR
VAR_OUTPUT
    lineNr : DINT;                          // current line number: starting with 1
END_VAR
VAR
    pSubProg : ^SubPrgHead;
    pIprHeadT : ^IprHead;
    pIprT : ^ProgIp;
    udOff : UDINT;
    anz : UDINT;
```

```
    pLine : ^_CMD_HEADER;
END_VAR
    pSubProg := ProgMgr_GetProgram(udIprNr);
    // Determine Interpreter address
    pIprT := IprMgr_GetInterpreter(udIprNr);
    pIprHeadT := ProgIp_GetHead(pIprT);
    udOff := pIprHeadT^.pCode$UDINT - (pSubProg$UDINT + SubProg^.udOffCode);
    pLine := (pSubProg + pSubProg^.udOffCode)$^_CMD_HEADER; // first line
    anz := 0;
    LineNr := 1;
WHILE anz < udOff DO
    pLine += pLine^.uiLen;
    anz += pLine^.uiLen;
    LineNr += 1;
END_WHILE;
END_FUNCTION
```

3.8.18 Exchanging the Interpreter during Runtime

3.8.18.1 Required Functions

```
FUNCTION GLOBAL ProgMgr_AllocProg
VAR_INPUT
    udPrgSize : UDINT;              // New size
    nPrgNo : UDINT;                 // := OPS.pProg^.uiProgs + OPS.uiProgNoLoad;
    szProgName : ^CHAR := NIL;
END_VAR
VAR_OUTPUT
    bOk : BOOL;                     // TRUE = OK, FALSE = ERROR
END_VAR
```

Copies an interpreter program from temporary memory to the RAM and marks it as active.

```
FUNCTION GLOBAL ProgIp_LoadProgFromTemp
VAR_INPUT
    pp : ^ProgIp;                   // Return of IprMgr_GetInterpreter(nProgNo)
    uiIprNo : UINT;                 // Interpreter number
    bCheck : BOOL := TRUE;
    bDoNotLink : BOOL := FALSE;
    bBTnoRun : BOOL := FALSE;
END_VAR
VAR_OUTPUT
    ret_code : ConfStates;
END_VAR
```

3.8.18.2 Example: Exchange during Runtime

1.) The Interpreter to be changed should be stopped:

```
pIprT := IprMgr_GetInterpreter(uiIprNo);   // Interpreter address
ProgIp_StopProgram(pIprT, uiIprNo);
```

2.) Sufficient memory must be available for a new IPR allocation:

```
OPS.udProgLength := pNewIpr^.udLength;
// IPR to be exchanged
OPS.uiProgNoLoad := uiIprNo;
// temporarily number for exchange
uiPrgNoNew := OPS.pProg^.uiProgs + OPS.uiProgNoLoad;


IF (ProgMgr_AllocProg (
      OPS.udProgLength + NewSubPrgHead.subProgHead$UDINT,
      uiPrgNoNew) = FALSE) THEN
      // Did not function!
END_IF;
```

3.) Copy code:

```
OPS.pDestProg := ProgMgr_GetProgram(uiPrgNoNew);

_MEMCPY(Dest :OPS.pDestProg,
              Source := pNewIpr,
              udLength := pNewIpr^.udLength);
```

4.) Exchange:

```
pIprHead := ProgIp_GetHead(pIpr);
pIprHead^.funkSt := ERROR;
pIpr := IprMgr_GetInterpreter(OPS.uiProgNoLoad);

IF (ProgIp_LoadProgFromTemp (pIpr, Ops.uiProgNoLoad ) <> C_OK) THEN
        // Did not function!
```

```
        TRACE(" ! LINKER_ERROR ! ");
END_IF;
```

3.8.19 Required Types

```
iprStates(READY,                    // IPR-command completed
          ERROR,                    // e.g.: wrong parameter
          ERROR_BUSY,               // e.g.: movement aborted
          BUSY,                     // IPR-command is processing
          QUIT,
          STOP_BP,                  // IPR stopped at breakpoint
          SINGLESTEP);              // IPR is in Single Step mode

NewSubPrgHead : STRUCT
          udFileCheck : UDINT;      // = udCRCHead (see SubPrgHead)
          subProgHead : SubPrgHead; // IPR heading structure
END_STRUCT;

pSubPrgHead:^SUBPRGHEAD;            // Pointer to IPR heading
```

Structure of the IPR Head

```
SubPrgHead :    STRUCT                  // Head of sub prog
udLng :         UDINT;                  // Program length
udCRC :         UDINT;                  // crc32 from program
szName :        ARRAY [0..31] OF CHAR;  // Program name
udOffLbl :      UDINT;                  // Label table offset
udOffLinks :    UDINT;                  // Link table offset
udOffCode :     UDINT;                  // Code offset
udOffProgLinks  UDINT;                  // Program link table offset
udCRCorig :     UDINT;                  // CRC from a non-linked program
udCRCHead :     UDINT;                  // CRC from IPR heading
END_STRUCT;
```

Transfer Structure for „NewInst()“ and „GetState()“

```
#define MAXCMDPARA 20
CmdStruct :STRUCT
```

```
uiCmd :    UINT;                          // Number of the command
aPara :    ARRAY [0..MAXCMDPARA-1] OF DINT// Input value
END_STRUCT;
```

Return Structure for „NewInst()“ and „GetState()“

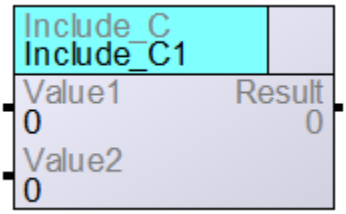
```
results : STRUCT
   uiLng : UINT;                                 // Length of buffer used inclusive
                                                 // 2 Byte for uiLng
   aData : ARRAY[0..16#FF-sizeof(UINT)] OF USIN// Return values
END_STRUCT;
```

3.9 Using C-Files in LASAL CLASS 2

In LASAL CLASS 2, the possibility exists to link existing and **tested** C-Files. The advantage of linking C-Files is that available program code doesn't have to be written again in structured text, which significantly shortens the project development time.

3.9.1 How are C-Files Linked in LASAL Projects?

To implement the following example, the **IncludeC** class with a CyWork method is required in which the values of two clients (**Value1** and **Value2**) are read using the Read method. These values will be transferred to a C function, which will return the result of a multiplication. The result will be displayed in the **Result** server of the **IncludeC** class.



In the CyWork method of this class, the following code is written:

Include_C*

```
1
2   FUNCTION VIRTUAL GLOBAL Include_C::CyWork
3     VAR_INPUT
4       EAX    : UDINT;
5     END_VAR
6     VAR_OUTPUT
7       state (EAX)    : UDINT;
8     END_VAR
9
10    Value1  := Value1.Read();
11    Value2  := Value2.Read();
12
13    Result := MyCFunct(to_int(Value1), to_int(Value2));
14
15    state := READY;
16
17  END_FUNCTION
18
```



In the next step, the C-file is linked. If a C-file already exists, the Create a C-File item can be skipped.

3.9.1.1 Creating a C-File

To create a C-file, select **File, New File** from the menu list or create it in the class directly by selecting **Add File...** in the context menu of the class. The desired code can now be written in the opened program code editor.

For this example the code appears as follows:

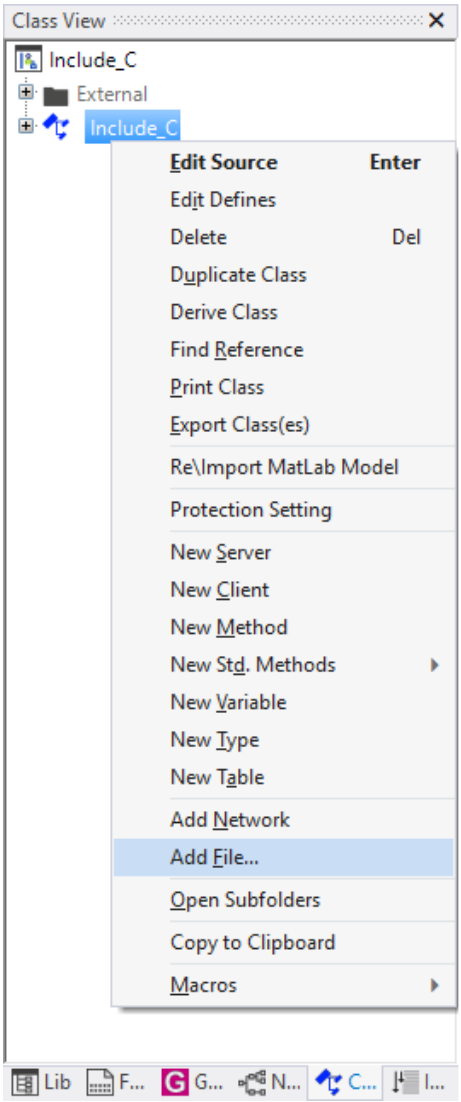
```
1
2  extern "C" short int MyCFunct (short int value1, short int value2)
3  {
4    short int result;
5
6    result = value1 * value2;
7    return result;
8  }
```

All C-Code files must be declared externally with "C".

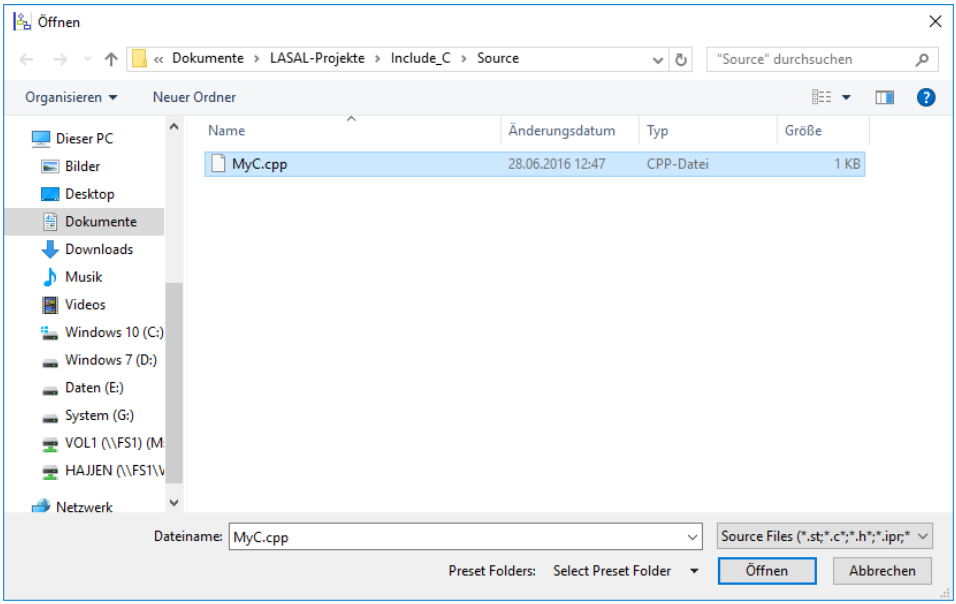
The file must now be saved; in the menu bar, select **File, Save File As**. Caution must be taken that the **Source folder** of the project is saved.

3.9.1.2 Linking a C-File

To link the file to the class (**Include_C**), open the context menu with a right click on the class name in the project tree and select **Add File...**.



In the **Open window**, select the desired C-file and press the **Open** button.



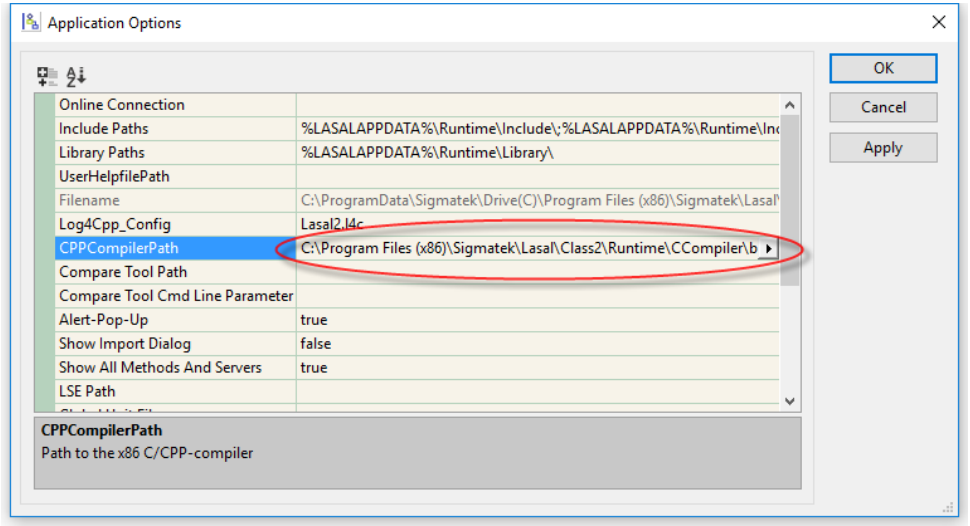
To use the function from the C-file, a private Header file must be created in the class using the following code:

```
1
2  FUNCTION GLOBAL __cdecl MyCFunct
3  VAR_INPUT
4    value1     : INT;
5    value2     : INT;
6  END_VAR
7  VAR_OUTPUT
8    retcode    : INT;
9  END_VAR;
10
```

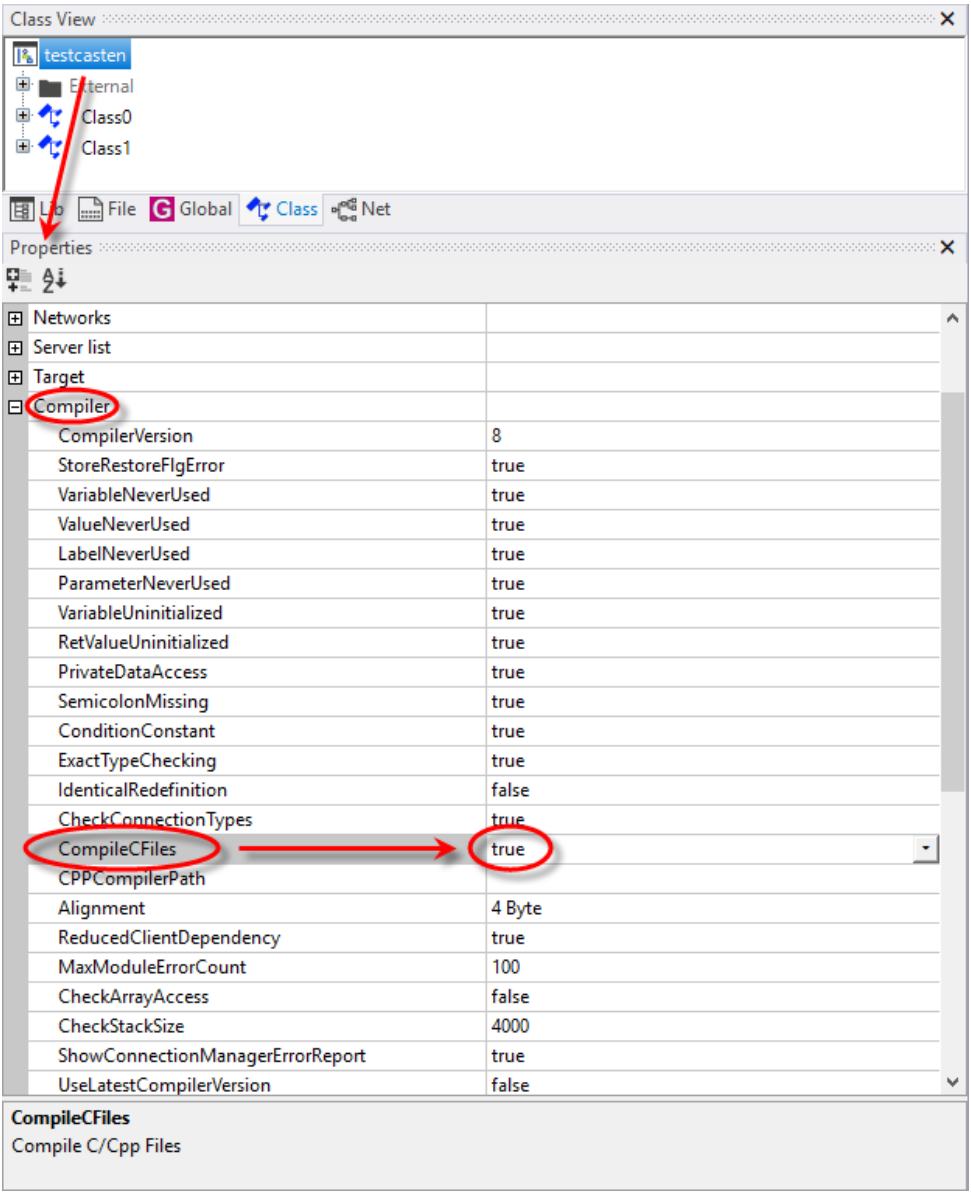
Alternatively, all function headers of the C functions can be written in the *.st file.

3.9.2 Important Details when Using C-Files

To load a project in the control, the project must be compiled with the C-files. In the menu bar under **Extras, Options** the **Applications Options window** is opened and under **CPPKompilerPath**, the path of the C-compiler is entered:



Attention! In order to use C files in the program, the flag "**CompileCFiles**" has to be set **true** in the **project properties** under **Compiler**!


Class View
testcasten
External
Class0
Class1
Lib
File
Global
Class
Net
Properties
Networks
Server list
Target
Compiler
CompilerVersion 8
StoreRestoreFlgError true
VariableNeverUsed true
ValueNeverUsed true
LabelNeverUsed true
ParameterNeverUsed true
VariableUninitialized true
RetValueUninitialized true
PrivateDataAccess true
SemicolonMissing true
ConditionConstant true
ExactTypeChecking true
IdenticalRedefinition false
CheckConnectionTypes true
CompileCFiles true
CPPCompilerPath
Alignment 4 Byte
ReducedClientDependency true
MaxModuleErrorCount 100
CheckArrayAccess false
CheckStackSize 4000
ShowConnectionManagerErrorReport true
UseLatestCompilerVersion false
CompileCFiles
Compile C/Cpp Files

3.9.3 C-Functions Supported by the Runtime Library



To use these functions, RTOS_C_interfaces.h must be included in the C-file where the calls are made as follows:

```
#include <RTOS_C_interfaces.h>
```

FTOL(dbl)	Float-to-Integer
ABS(intval)	Absolute value for integers
ACOS(dbl)	Arc Cosine
ASIN(dbl)	Arc Sine
ATAN(dbl)	Arc Tangent
ATAN2(dbl1, dbl2)	Arc tangent with a greater value range than ATAN, in regard to the angular direction (indicates a point on the Gaussian plane)
COS(dbl)	Cosine
COSH(dbl)	Hyperbolic cosine
EXP(dbl)	Exponential function (e to the power of x)
FABS(dbl)	Absolute value for floating point numbers
FMOD(dbl1, dbl2)	Remaining value for floating point numbers
LABS(lngval)	Absolute value for long integers
LOG(dbl)	Logarithm function of base e (=“natural Logarithm“)
LOG10(dbl)	Logarithm function of base 10 (=“base 10 Logarithm“)
POW(dbl1, dbl2)	Exponential function
SIN(dbl)	Sine
SINH(dbl)	Hyperbolic sine
TAN(dbl)	Tangent
TANH(dbl)	Hyperbolic tangent
SQRT(dbl)	Square root
ATOF(str)	String-to-floating point conversion
CEIL(dbl)	Returns the next higher whole number

FLOOR(dbl)	Returns the next lowest whole number
FREXP(dbl, pint)	Disassembles value (a) so that a = f * b^2 applies. b is provided via address
LDEXP(dbl, intval)	Inverse function of FREXP
MODF(dbl, pdbl)	The number a is disassembled into a whole number and decimal component. The whole number component is stored in b (it is via address), the decimal component is the return value of the function
_J0(dbl)	Returns the Bessel function of the first type, order 0
_J1(dbl)	Returns the Bessel function of the first type, order 1
_JN(intval, dbl)	Returns the Bessel function of the first type, order N
_Y0(dbl)	Returns the Bessel function of the second type, order 0
_Y1(dbl)	Returns the Bessel function of the second type, order 1
_YN(intval, dbl)	Returns the Bessel function of the second type, order N
_HYPOT(dbl1, dbl2)	Calculates the hypotenuse in a right triangle



To use these functions, the must be written **consistently upper case** (as shown in the list above) when called.

The **GNU compiler** (GNU GCC 4.6.6) is provided by SIGMATEK. The MS-VC 6.0 can also be used, however, it is not provided by SIGMATEK.



Every LASAL project linked to C-files must be compiled with the C-files at least once.

With the "CompileCFiles" compiler settings, whether the C files should also be compiled is specified.

3.10 Loader

3.10.1 Memory, String

3.10.1.1 _memmove

The _memmove function copies a specified number of bytes from a source to a destination. When the source and destination memory areas overlap, the original data from the source is copied to the overlapping area before it is overwritten.



```
FUNCTION GLOBAL _memmove
VAR_INPUT
  dest : ^void;
  src : ^void;
  anz : UDINT;
END_VAR;
```

Parameter

dest Pointer to the destination.

src Pointer to the source.

anz Number of bytes to be copied.

3.10.1.2 _memicmp

The _memicmp function compares to memory blocks, whereby no difference is made between the values corresponding to the ASCII characters A-Z and a-z. If both blocks are identical, 0 is returned. If not, a non-zero value is given.



```
FUNCTION GLOBAL STR _memicmp
VAR_INPUT
  str1  : ^void;
  str2  : ^void;
  len   : UDINT;
END_VAR
VAR_OUTPUT
  Result : UDINT;
END_VAR;
```

Parameter

str1 Pointer to the first memory block.

str2 Pointer to the second memory block.

len Number of bytes for comparison.

Return value

Result 0 when both blocks are identical (without upper/lower case comparison); otherwise a value unlike 0.

3.10.1.3 _memcmp

The _memcmp function compares to memory blocks. If both blocks are identical, 0 is returned. If not, a non-zero value is given.



```
FUNCTION _memcmp
VAR_INPUT
  ptr1  : ^void;
  ptr2  : ^void;
  cntr  : UDINT;
END_VAR
VAR_OUTPUT
  notequal  : UDINT;
END_VAR;
```

Parameter

ptr1 Pointer to first memory block.

ptr2 Pointer to second memory block.

cntr Number of bytes for comparison.

Return value

notequal 0 when both blocks are identical, otherwise a value unlike 0.

3.10.1.4 _memcpy

The _memcpy function copies one memory block to another. If both memory areas overlap, the block with the lowest address must serve as destination (defined in the ptr1 parameter).



```
FUNCTION _memcpy
VAR_INPUT
  ptr1  : ^void;
  ptr2  : ^void;
  cntr  : UDINT;
END_VAR;
```

Parameter

ptr1 Pointer to destination.

ptr2 pointer to source.

cntr Number of bytes to be copied.

3.10.1.5 _memcpyrev

The _memcpyrev function copies one memory block to another. If both memory areas overlap, the block with the lowest address must serve as source (defined in the ptr2 parameter).



```
FUNCTION _memcpyrev
VAR_INPUT
  ptr1 : ^void;
  ptr2 : ^void;
  cntr : UDINT;
END_VAR;
```

Parameter

ptr1 Pointer to destination.

ptr2 Pointer to source.

cntr Number of bytes to be copied.

3.10.1.6 _memset

The _memset function sets a number of bytes in a memory area to a specified character.



```
FUNCTION _memset
VAR_INPUT
  dest   : ^void;
  usByte : USINT;
  cntr   : UDINT;
END_VAR;
```

Parameter

dest Pointer to the target memory area.

usByte Character to set.

cntr Number of characters.

3.10.1.7 _strcat

Die _strcat function hangs a character string on the end of another character string.



```
FUNCTION GLOBAL _strcat
VAR_INPUT
  Dest : ^CHAR;
  src : ^CHAR;
END_VAR;
```

Parameter

dest 0-terminated destination string.

src 0-terminiertes source string.

3.10.1.8 _stricmp

The _stricmp function compares 2 character strings with one another without regard to upper or lower case letters. When both strings are equal, 0 is returned; otherwise 1.



```
FUNCTION GLOBAL _stricmp
VAR_INPUT
 str1 : ^char;
 str2 : ^char;
END_VAR
VAR_OUTPUT
 retcode : UDINT;
END_VAR;
```

Parameter

str1, str2 The compared 0-terminated character strings.

Return value

retcode 0 when equal, 1 when strings are not equal.

3.10.1.9 _strupr

The _strupr function converts a character string to upper case letters.



```
FUNCTION _strupr
VAR_INPUT
  str   : ^CHAR;
END_VAR;
```

Parameter

str 0-terminated character string.

3.10.1.10 _strchr

The _strchr function searches for a character within a string. If the character is found, a pointer is to the first character in the string is returned. When the character is not found then NIL is returned.



```
FUNCTION _strchr
VAR_INPUT
  src : ^char;
  chr : CHAR;
END_VAR
VAR_OUTPUT
  dest : ^CHAR;
END_VAR;
```

Parameter

src 0-terminated character string.

chr The search character.

Return value

dest Pointer to the first character in the string or NIL when the character is not available.

3.10.1.11 _strcmp

The _strcmp function compares two character strings with regard to upper and lower case letters. When equal, 0 is returned; otherwise a 1 is given.



```
FUNCTION _strcmp
VAR_INPUT
  str1  : ^char;
  str2  : ^char;
END_VAR
VAR_OUTPUT
  Retcode : UINT;
END_VAR;
```

Parameter

str1 0-terminated string.

str2 0-terminated string.

Return value

Retcode 0 when strings are equal, otherwise 1.

3.10.1.12 _strcpy

The _strcpy function copies one character string to another.



```
FUNCTION STR GLOBAL _strcpy
VAR_INPUT
  dest  : ^CHAR;
  src : ^CHAR;
END_VAR;
```

Parameter

dest 0-terminated destination string.

src 0-terminated source string.

3.10.1.13 _strlen

The _strlen function supplies the length of a 0-terminated string. The terminating 0 is not counted in the length of the string.



```
FUNCTION _strlen
VAR_INPUT
  src : ^CHAR;
END_VAR
VAR_OUTPUT
  retcode : UDINT;
END_VAR;
```

Parameter

src 0-terminated string.

Return value

retcode Length of the character string.

3.10.1.14 _strncpy

The _strncpy function copies the *src* string to the *dest* string, whereby a maximum number of *max* characters is copied. The destination string *dest* is always terminated with 0.



```
FUNCTION STR GLOBAL _strncpy
VAR_INPUT
  dest : ^CHAR;
  src : ^CHAR;
  max : UDINT;
END_VAR;
```

Parameter

dest 0-terminated destination string.

src 0-terminated source string.

max Maximum number of characters to copy.

3.10.1.15 _itoa

The _itoa function converts a UDINT value in 0-terminated string to hexadecimal format.



```
FUNCTION GLOBAL _itoa
VAR_INPUT
  udNumber : UDINT;
  pString  : ^CHAR;
END_VAR;
```

Parameter

udNumber The value to be converted.

pString 0-terminated String to which the result is written.

3.10.2 Classes and Objects

3.10.2.1 _LookUpEmbed_C

The _LookUpEmbed_C function analyzes an ASCII string and returns the address of the object or channel written by the character string *pText*.



```
FUNCTION GLOBAL __CDECL _LookUpEmbed_C
VAR_INPUT
  pText : PCHAR;
  pObj_ch : ^PVOID;
  pObj : ^pVirtualBase;
END_VAR
VAR_OUTPUT
  chMode : CHMODE;
END_VAR;
```

Parameter

pText 0-terminated string.

pObj_ch Pointer to an object pointer. Here, the address or channel described by pText written.

pObj Pointer to an object pointer. If this value does not equal NIL and pText used to describe a channel, the address of the connected object is written to this pointer.

Return value

chMode The channel type of the object or channel. If no type is found then _CH_NONE is returned.

3.10.3 Methods, Clients, Server

3.10.3.1 RD

The RD function calls the read method of a server channel.



```
FUNCTION RD
VAR_INPUT
  pt : ^void;
END_VAR
VAR_OUTPUT
  dData : DINT;
END_VAR;
```

Parameter

pt Pointer to the server channel.

Return value

dData The return value of read method from the server channel.

3.10.3.2 RD_CHNL

The RD_CHNL function calls the read method of the server channel to which the specified client is connected.



```
FUNCTION RD_CHNL
VAR_INPUT
  pt     : ^void;
  chOffs : UINT;
END_VAR
VAR_OUTPUT
  dData  : DINT;
END_VAR;
```

Parameter

pt Pointer to the object or client.

chOffset Offset added to the value of pt. If pt points to a client channel, 0 must be entered in the offset.

Return value

dData The return value of the read method from the connected server channel.

3.10.3.3 WR

The WR function calls the write method of the specified server channel.



```
FUNCTION WR
VAR_INPUT
  pt    : ^void;
  dData : DINT;
END_VAR
VAR_OUTPUT
  EAX   : UDINT;
END_VAR;
```

Parameter

pt Pointer to the server channel.

dData The value to be written.

Return value

result The return value from the write method.

3.10.3.4 WR_CHNL

The WR_CHNL function writes a value to the client channel and calls the write method from the server channel to which the specified client is connected.



```
FUNCTION WR_CHNL
VAR_INPUT
  pt : ^void;
  chOffs : UINT;
  dData : DINT;
END_VAR
VAR_OUTPUT
  EAX : UDINT;
END_VAR;
```

Parameter

pt Pointer to the server channel.

chOffset Offset added to pt. If pt points to a client channel, an offset of 0 must be entered.

dData The written value.

Return value

result The return value from the write method.

3.10.4 Interpreter Programs

3.10.4.1 Calling Interpreter Programs

The interpreter programs are processed in the cyclic part of the loader. This process is called “ProgSequences-call”. The cyclic part of the loader is called from the cyclic thread before the CyWork method of the Lasal object.

The execution time or the interpreter program has the following effects on the total function of the system:

- The longer the “ProgSequences-call” takes, the later the Cywork method is called from the Lasal object.

- The CPU output for the “ProgSequences-call” is not available for the lowest priority tasks (e.g. background, visualization).

There are functions with which the “ProgSequences-call” can be influenced and the process time for the interpreter programs can be limited:

ProgIp_SetMaxProgSeqDuration_ms

IprMgr_SetMaxProgSeqDurationAll_ms

IprMgr_SetProgSeqCycleTime_ms

3.10.4.2 ProgIp_SetMaxProgSeqDuration_ms

Used to set the processing time for an interpreter program. The standard value is 3 ms.



```
FUNCTION GLOBAL ProgIp_SetMaxProgSeqDuration_ms
VAR_INPUT
  pp : ^ProgIp;
  maxProgSeqDuration_ms : UDINT;
END_VAR;
```

Parameter

pp Pointer to the interpreter object.

maxProgSeqDuration_ms Time in milliseconds.

3.10.4.3 IprMgr_SetMaxProgSeqDurationAll_ms

Used to set the processing time for all interpreter programs. No limit is set as a standard value. If the process is interrupted due to the time limit, it is not reset to program number 0. Instead, it continues from the interrupted program.



```
FUNCTION GLOBAL IprMgr_SetMaxProgSeqDurationAll_ms
VAR_INPUT
  maxProgSeqDuration_ms : UDINT;
END_VAR;
```

Parameter

maxProgSeqDuration_ms The time in milliseconds.

3.10.4.4 IprMgr_SetProgSeqCycleTime_ms

Sets the cycle time of the “ProgSequences-call”. The standard value is system clock. This means, for the IPC, 2 ms must be set for 386 CPUs.



```
FUNCTION GLOBAL IprMgr_SetProgSeqCycleTime_ms
VAR_INPUT
  progSeqCycleTime_ms : UDINT;
END_VAR;
```

Parameter

progSeqCycleTime_ms The time in milliseconds.

3.10.4.5 ProgIp_InstallIprCallback

Registers a callback function that is called on various events in the interpreter processing.



```
FUNCTION GLOBAL ProgIp_InstallIprCallback
VAR_INPUT
  pThis : pVoid;
  pFunc : pVoid;
END_VAR;
```

Parameter

pThis When the callback function is a method of an object instead of a global function, then this parameter is a pointer to the object that contains the method. The callback method is called in the context of this object.

pFunc A pointer to a callback function or method. The prototype of this function is as follows:



```
FUNCTION IprCallback
VAR_INPUT
  eventType : UDINT;
  data : pVoid;
END_VAR;
```

eventType Specifies the type of event. data points to a structure that corresponds to the eventType.

```
#define IPRCB_IPCHANGED 0  // eventType
#define IPRCB_ERROR 1 // since 02.02.039

TYPE
  // data structure for eventType IPRCB_IPCHANGED
  LDR_IpChanged_Data : STRUCT
    sizeOfStruct : UDINT;
    ofs : UDINT;
    prgNbr : UINT;
    pp : ^ProgIp;
  END_STRUCT;
  // data structure for eventType IPRCB_ERROR
  LDR_Error_Data : STRUCT
    sizeOfStruct : UDINT;
    errorCode : DINT;
    prgNbr : UINT;
    pp : ^ProgIp;
  END_STRUCT;
END_TYPE;
```

When the instruction pointer of an interpreter program changes, the callback function is called with eventType IPRCB_IPCHANGED.

When an interpreter program enters error state, the callback function is called with eventType IPRCB_ERROR. The errorCode can be one of the following values:

```
#define PROGIP_ERR_GENERAL 1
#define PROGIP_ERR_ENDOFCODE 2
#define PROGIP_ERR_STACKOVERFLOW 3
#define PROGIP_ERR_LABELINVALID 4
#define PROGIP_ERR_SYNTAX 5
#define PROGIP_ERR_ICMDFAILED 6
#define PROGIP_ERR_RUNPROGFAILED 7
```

Requirements

Available in LASAL CLASS 2 only.

3.10.4.6 Creating Interpreter Programs

In addition to the LASAL CLASS interpreter programs, an interpreter program can be created by the user for the runtime.

3.10.4.7 ProgMgr_CreateIprProg

Allocates an interpreter object.



```
FUNCTION GLOBAL ProgMgr_CreateIprProg
VAR_INPUT
        flags : UDINT;
        preAllocSize : UDINT;
END_VAR
VAR_OUTPUT
        prgNbr : UINT;
END_VAR;
```

Parameter

flags Define the flags for the interpreter object. The following values are available:

Value	Meaning
PROGMGR_FLAGS_ALLOCFIX	Normally, the size of the allocated memory for the interpreter program is fit to the size of the user program. If, however, this flag is set then the program is pre-allocated with the parameter “preAllocSize” and is not changed at a later time. The memory for the regular as well as the temporary program is also pre-allocated.

preAllocSize If a value unlike 0 is entered, the interpreter program memory is pre-allocated with this value. The size of the SubPrgHead + code and tables are entered. NewSubPrgHead is not counted.

Return

prgNbr If the Interpreterobjekt is applied then the number of the interpreter object is returned. When an error occurs, the return value is 16#FFFF.

Requirements

Available in Lasal2 only.

3.10.4.8 ProgMgr_LoadIprProg

Loads the interpreter program. If the program is running during this function call, it is stored in the temporary program memory. As soon as it's stopped, the check_for_load() function is called that loads the program in the regular program memory. If the program is not running during this function call, it is loaded to the regular program memory immediately.



```
FUNCTION GLOBAL ProgMgr_LoadIprProg
VAR_INPUT
  prgNbr : UINT;
  pProg : ^SubPrgHead;
  prgSize : UDINT;
  bCheck : BOOL := TRUE;
END_VAR
VAR_OUTPUT
  retVal : DINT;
END_VAR;
```

Parameter

prgNgr Number of the interpreter object..

pProg Pointer to the loaded program (SubPrgHead).

prgSize Size of the loaded program.

bCheck Indicates whether the program is tested for linker instructions.

Return

retVal If no error occurs then 0 is returned; otherwise a negative error code is given.

Requirements

Available in LASAL CLASS 2 only.

3.10.5 Comlink

The following loader functions are found in the "LASALOS Comlink APIs in the Loader" documentation:

- **Login**
- **TxCommand**
- **TxUpd**
- **StartStopRefresh**

- **InstallCallBack**

3.10.6 Loader Helpfunctions for the Data Type REAL

3.10.6.1 Real_IsSigned



```
FUNCTION GLOBAL Real_IsSigned
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  bResult : BOOL;
END_VAR;
```

This function returns TRUE when the input value has a negative sign.

3.10.6.2 Real_IsNormal



```
FUNCTION GLOBAL Real_IsNormal
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  bResult : BOOL;
END_VAR;
```

This function returns TRUE when the input valus is “normal” (not null, subnormal, infinite and not NaN – not a number, invalid or undefined number).

3.10.6.3 Real_IsFinite



```
FUNCTION GLOBAL Real_IsFinite
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  bResult : BOOL;
END_VAR;
```

This function returns TRUE when the input values is null, normal or subnormal (not infinite and not NaN – not a number, invalid or undefined number).

3.10.6.4 Real_IsZero



```
FUNCTION GLOBAL Real_IsZero
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  bResult : DINT;
END_VAR;
```

This function returns TRUE when the input value is positive or negative null.

3.10.6.5 Real_IsSubnormal



```
FUNCTION GLOBAL Real_IsSubnormal
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  bResult : DINT;
END_VAR;
```

This function returns TRUE when the input value is subnormal.

3.10.6.6 Real_IsInfinite



```
FUNCTION GLOBAL Real_IsInfinite
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  bResult : DINT;
END_VAR;
```

This function returns TRUE when the input value is infinite.

3.10.6.7 Real_IsNaN



```
FUNCTION GLOBAL Real_IsNaN
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  bResult : DINT;
END_VAR;
```

This function returns TRUE when the input value is a silent or signaling NaN (not a number, an invalid, undefined number).

3.10.6.8 Real_IsSignaling



```
FUNCTION GLOBAL Real_IsSignaling
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  bResult : DINT;
END_VAR;
```

This function returns TRUE when the input value is a silent or signaling NaN (not a number, an invalid, undefined number).

3.10.6.9 Real_GetClass



```
FUNCTION GLOBAL Real_GetClass
VAR_INPUT
  Value : REAL;
END_VAR
VAR_OUTPUT
  Result : REAL_CLASS;
END_VAR;
```

This function returns the category of the given number. The return variable is of the type ENUM and can have the following values:

REAL_SignalingNaN,
REAL_QuietNaN,
REAL_NegativeInfinity,
REAL_NegativeNormal,
REAL_NegativeSubnormal,
REAL_NegativeZero,
REAL_PositiveZero,
REAL_PositiveSubnormal,
REAL_PositiveNormal,
REAL_PositiveInfinity

3.11 Debugging a Program

To illustrate how a program is debugged, a simple Step control is used as show below.

To start, a breakpoint is set on the line where the cursor is located using the  button or by pressing **F4**.

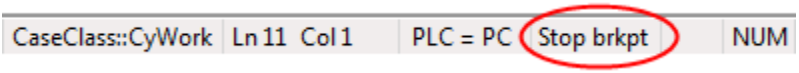
CaseClass [RO]

Debugging Context:

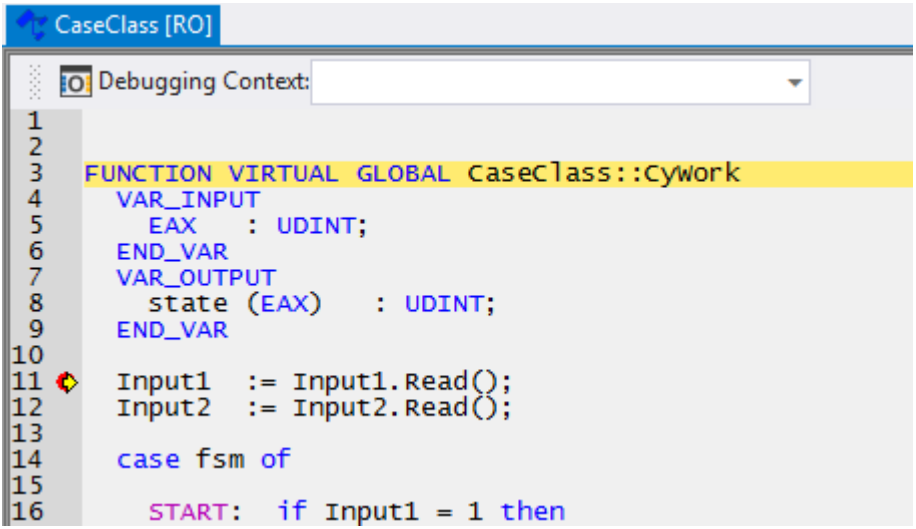
```
1
2
3   FUNCTION VIRTUAL GLOBAL CaseClass::CyWork
4     VAR_INPUT
5       EAX    : UDINT;
6     END_VAR
7     VAR_OUTPUT
8       state (EAX)    : UDINT;
9     END_VAR
10
11    Input1  := Input1.Read();
12    Input2  := Input2.Read();
13
14    case fsm of
15
16      START:  if Input1 = 1 then
17                fsm := WAIT;
18              end_if;
19
20      WAIT:   if (Input1 = 1) & (Input2 = 1) then
21                fsm := FINISH;
22              end_if;
23
24      FINISH: if (Input1 = 0) & (Input2 = 0) then
25                Counter += 1;
26                fsm := START;
27              end_if;
28
29    end_case;
30
31    state := READY;
```

If the program is currently running or the CPU is restarted, the program will stop at the location of the breakpoint (red dot).

This is seen when a yellow arrow appears over the red dot or **Stop Brkpt** is shown in the status line at the lower right of the screen.



The yellow arrow identifies the actual program line to be processed.



From this point on, the actual debugging of the program begins. The following tools are provided for debugging:

Jump to a function

Using either the button above or the **F5** key, the following line or method is processed.

Skip a function

Using this button or the combination **Alt + F5**, the following line or method is skipped.

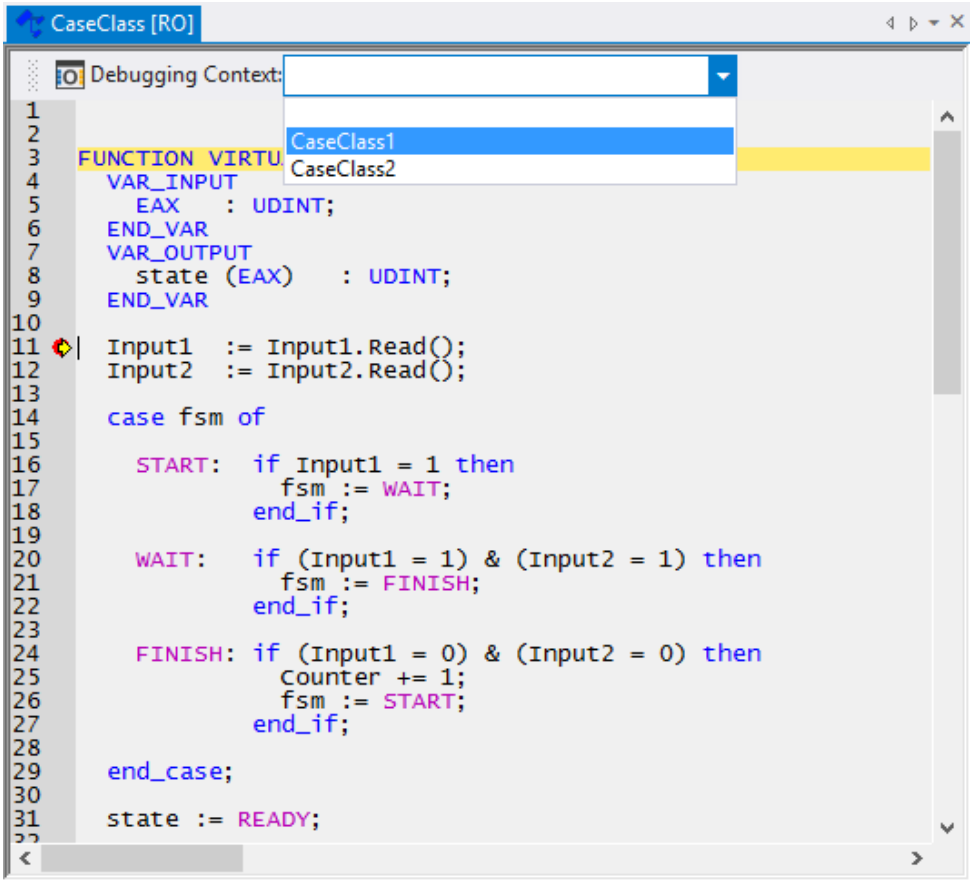
Exit a function

With this button or the combination **Shift + F5**, the user can exit a method, which was entered using the F5 key. The method will however, still be processed.

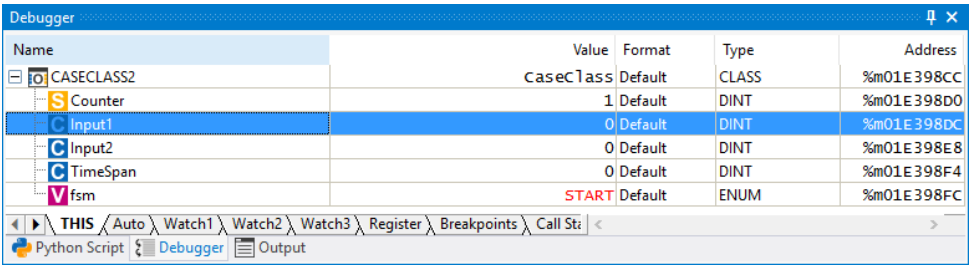


Select debugging context.

If several objects of a class are available and the class method code is open, debugging can be limited to the desired object by pressing the arrow button.



To see the values of the clients, open the output window (**Debugger, THIS**).



The first two lines in the program code should be skipped using the **Alt + F5** key combination so that the **Read() method** is not executed. Next, press the **F5** key only.

```
10
11    Input1  := Input1.Read();
12    Input2  := Input2.Read();
13
14    case fsm of
15
16      START:  if Input1 = 1 then
17                fsm := WAIT;
18              end_if;
```

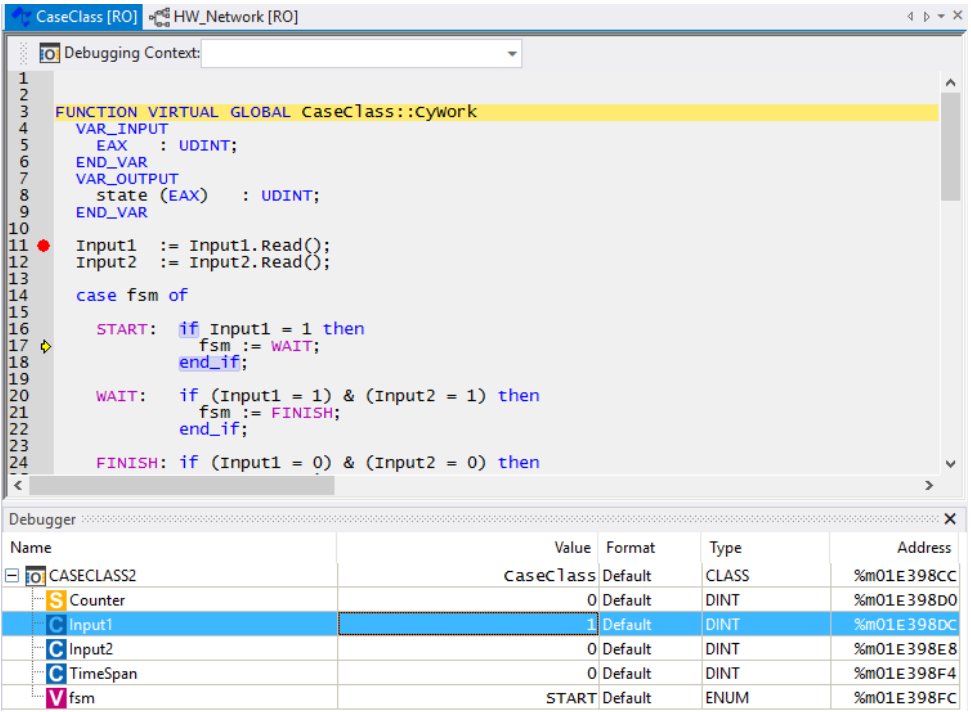
Because Input 1was not set (with a switch for example), the next step **"if Input1 = 1 then"** is skipped and the step control remains at START.

Now Input1 (switch) is set and the  **button** or **F7** key is pushed to start the program.

The program then stops with the yellow arrow on the breakpoint again. Press **Alt + F5** and then just the F5 key to continue from point.

If the F5 button is pressed, the **fsm** step control will switch from **START** to **WAIT** (see yellow arrow).

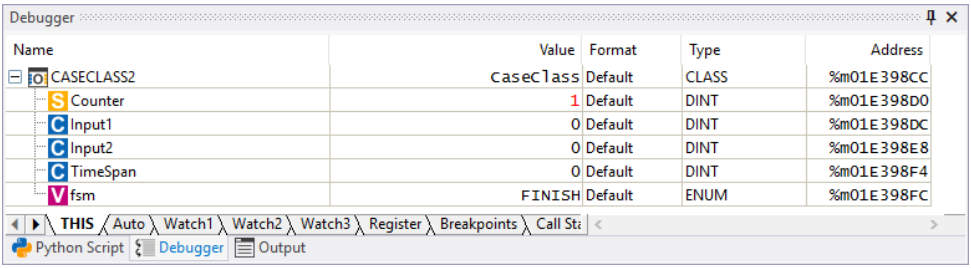
The values should always be observed in the output window when debugging.





In this example, the Counter should be incremented by one at least once up to the debugging point.

If a value has changed in the debugger since the last step / breakpoint, it is shown in red. (Here, the Counter value has changed from 0 to 1)

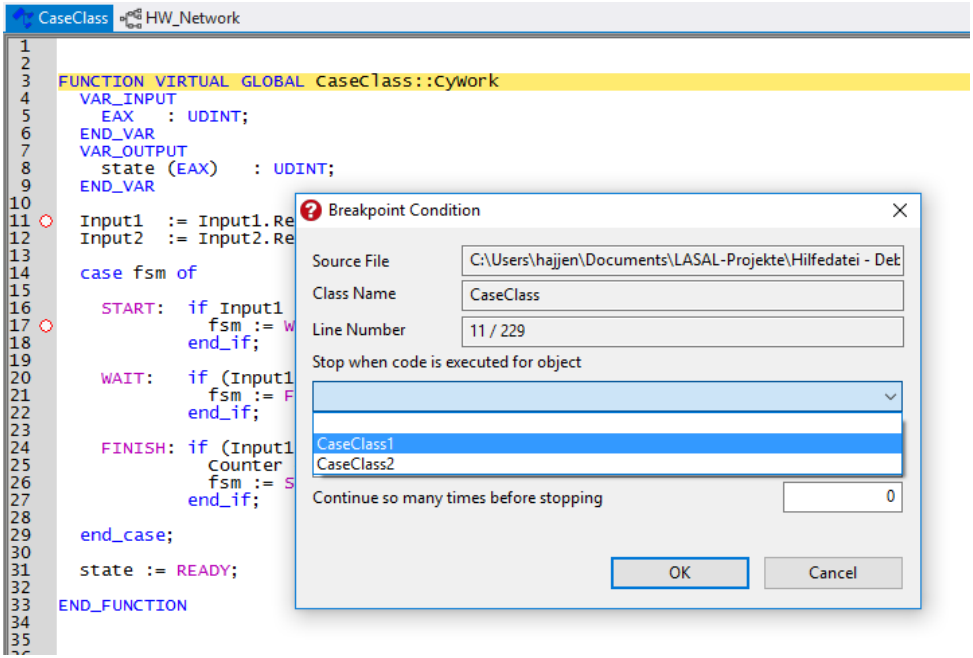


3.11.1 Debugging with Conditional Breakpoints

Conditional break points are used in debugging when several objects of a class are available which are queried for specific conditions or processed for a pre-defined number of executions. With debugging of this type, the data of specific object can be processed.

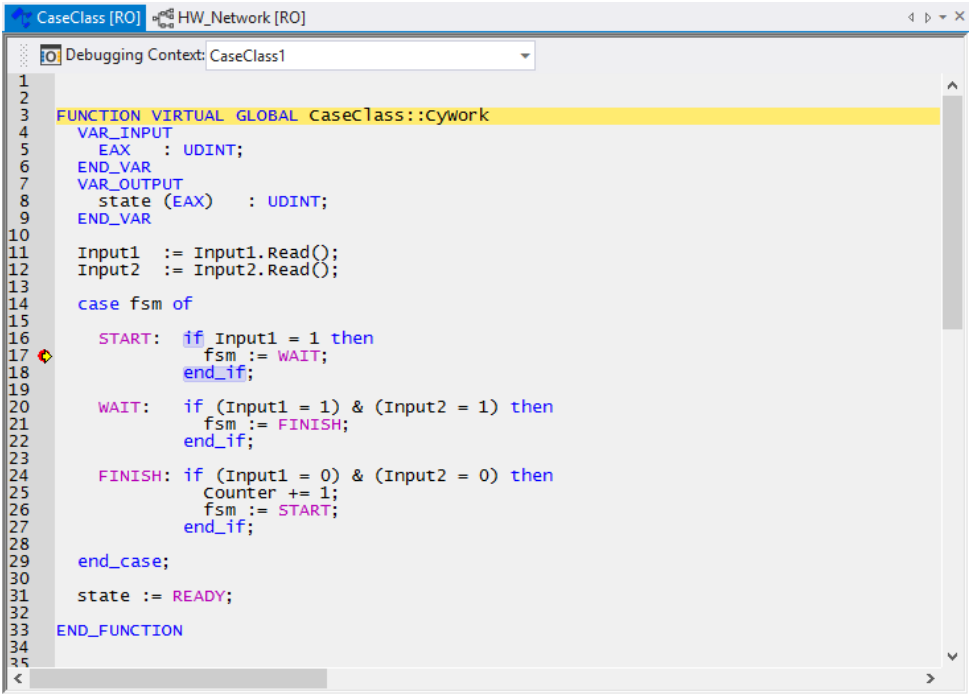
Place an additional object of the class, **CaseClass** in the object network and open the program code editor.

Set the cursor on the location shown in the following illustration and click on the  symbol to set a conditional breakpoint.



This opens the **Breakpoint Condition window** in which, under **Stop when code is executed for object**, the **CaseClass1** object is selected (here it is necessary that no debugging context was selected, otherwise this field is preset and grayed out in the context object!). Click **OK** to confirm the setting.

If in the destination system, **Input1** of **CaseClass2** is set to **1**, the breakpoint is ignored since it was set for **CaseClass1** only. If Input1 of **CaseClass1** is set to **1**, the program will stop at the breakpoint.





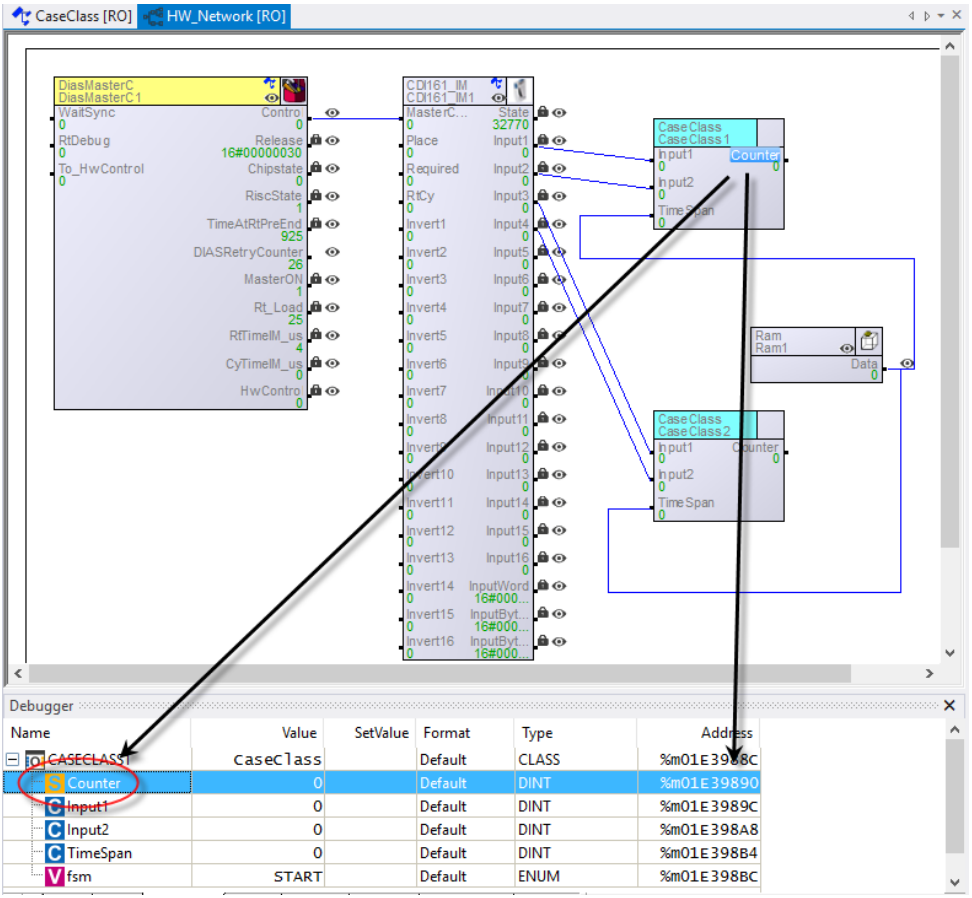
Under **Condition**, an additional condition can be set. In the field, **continue so many times before stopping**, the number of executions before the program stops at the breakpoint can be defined.

Now the program can be processed line-by-line using the debug symbol bar.

3.11.2 Debugging with Data Breakpoints

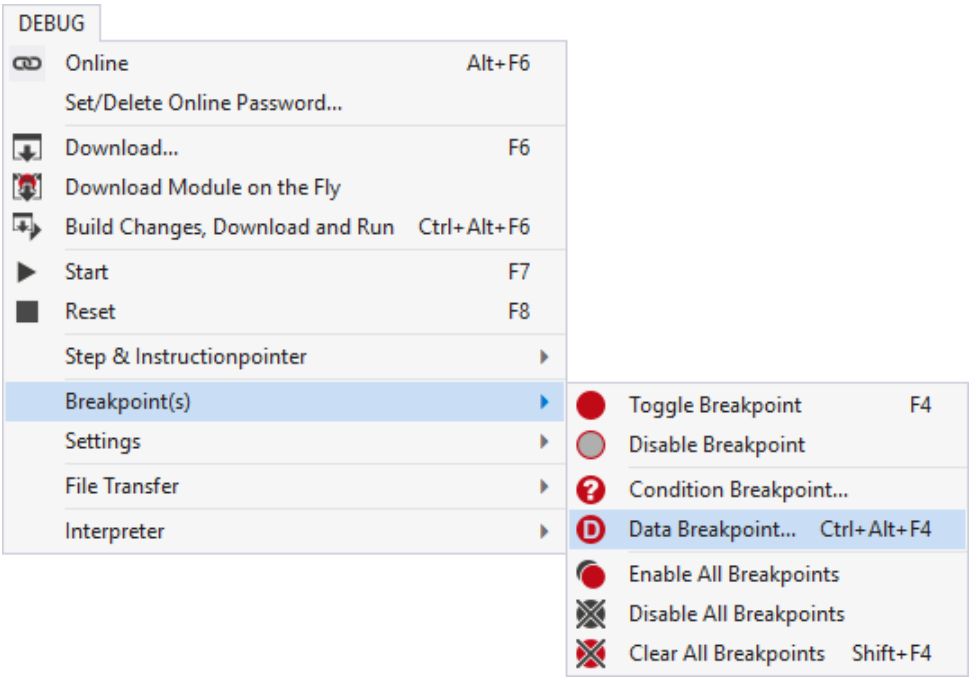
Clients, servers and data elements from objects of a class have definite addresses. To limit a breakpoint so that it stops the program only when the memory address of a specific client, server or data element is accessed, a data breakpoint is used. With a data breakpoint, the address of a specific client, server or data element can be assigned.

Before a data breakpoint can be set, the address of the Counter server from **objCaseClass1** must first be determined. This is necessary since the program only stops when the server with the defined address is accessed. In the **debug window**, locate the address of the **Counter** server and select it.

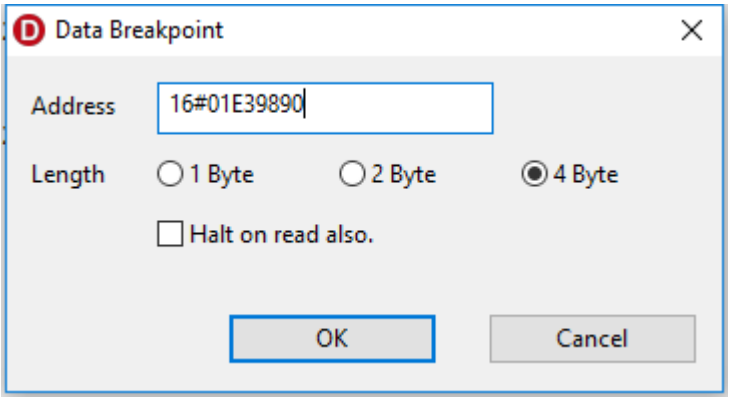


Because the breakpoint is now dependant on an address, the position of the cursor (breakpoint) no longer plays a role!

To set a data breakpoint, select **Debug, Breakpoint(s), Data Breakpoint** in the menu bar.

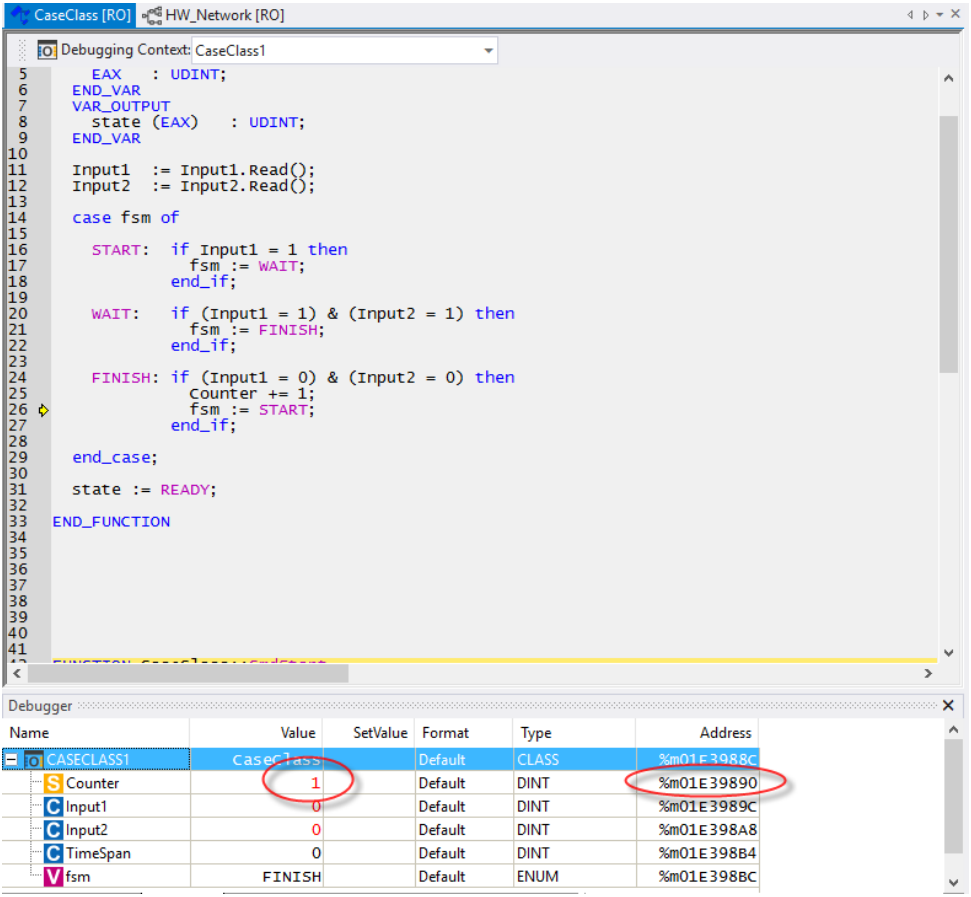


In the **Data Breakpoint window**, enter the address of the Counter server. Under length, the data length of is assigned depending on the address (here: 4 bytes). **Click OK**.



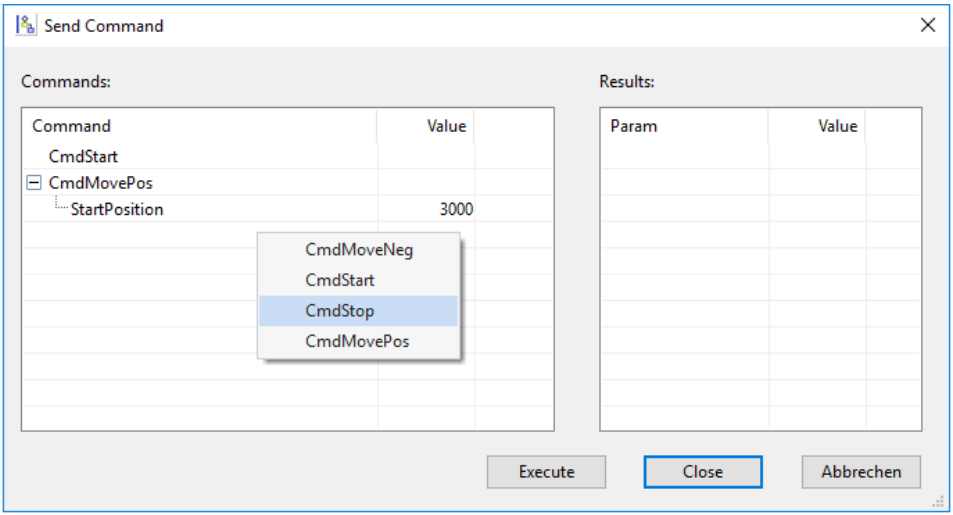
The assigned address must be divisible by the length. Every address is divisible by 1 (1byte) so 1 byte can certainly be used.

If the field next to **Halt on read** also is checked, then the program will stop also stop with a read access of this server. Set **Input1** and **Input2** of the **CaseClass1** to **1** and then back to **0**. This causes the program to jump **FINISH** and stop because of the data breakpoint at the line where the **Counter** server is incremented by 1.


CaseClass [RO]
HW_Network [RO]
Debugging Context: CaseClass1
EAX : UDINT;
END_VAR
VAR_OUTPUT
state (EAX) : UDINT;
END_VAR
Input1 := Input1.Read();
Input2 := Input2.Read();
case fsm of
START: if Input1 = 1 then
fsm := WAIT;
end_if;
WAIT: if (Input1 = 1) & (Input2 = 1) then
fsm := FINISH;
end_if;
FINISH: if (Input1 = 0) & (Input2 = 0) then
Counter += 1;
fsm := START;
end_if;
end_case;
state := READY;
END_FUNCTION
Debugger
Name
Value
SetValue
Format
Type
Address
CASECLASS1
CaseClass1
Default
CLASS
%m01E3988C
Counter
1
Default
DINT
%m01E39890
Input1
0
Default
DINT
%m01E3989C
Input2
0
Default
DINT
%m01E398A8
TimeSpan
0
Default
DINT
%m01E398B4
fsm
FINISH
Default
ENUM
%m01E398BC

3.11.3 Debugging with NewInst Commands

If the NewInst method is applied usint commands, these commands can be called over a debug interface. The **"Send Command"** must be executed with a right click in the context menu for the respective object.



With a right mouse click, the predefined commands with their input parameters are entered in the dialog list. The commands are then processed sequentially with a click on "Execute". The return value for the last command is shown on the right side.

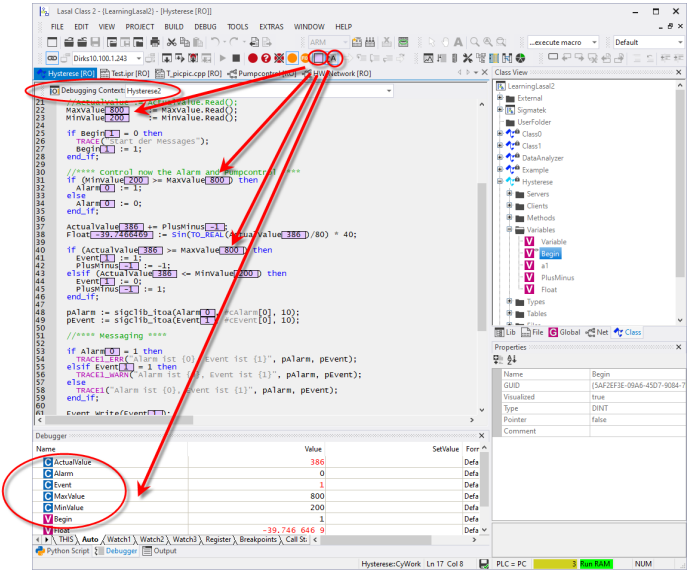
3.11.4 The Views Auto and LiveView

Live View

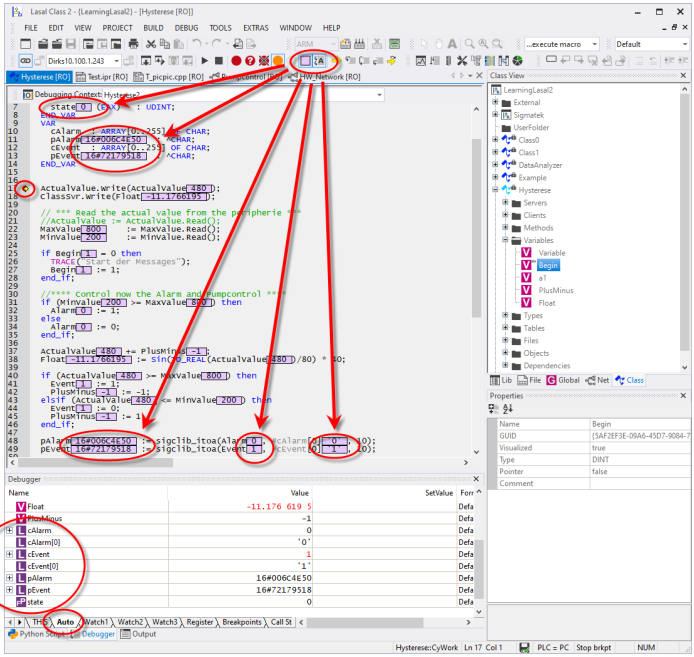
When debugging, it often is an advantage to see the valid values of all variables currently visible in the Code Editor directly in the Editor. For this the Live View button in the toolbar „Debug“ has to be pressed ( ).

Auto

The variables of the function, where the cursor in the Editor is currently placed, are all displayed in the Debugger pane in the register "**Auto**", if the button **Auto** ( ) is pressed.



If the program flow has been interrupted (e.g. because of a breakpoint), also the values of the transfer parameters and the local variables are displayed.



3.12 Scripting

3.12.1 What is Scripting?

Scripting is the creation of processes. With the instructions available in the script language, existing projects can be expanded or new projects generated. LASAL CLASS 2 uses Python for this purpose.

3.12.2 Introduction in Python

Python is a relatively easy to learn yet impressively strong script language and is also suitable for beginner programmers. For Python, no explicit interpreter process is required.

More information regarding programming with Python can be found at the following address:

http://www.python.org/

The most important commands for creating batch files are discussed next.



With the use of variables, no special mathematic characters are allowed!

`MyVar-3xy=...` => is wrong!

Attention should also be paid to upper and lower case characters!

`MyVar <> myvar`

3.12.3 LASAL CLASS 2 Python Commands

To use the Python interpreter language with LASAL CLASS 2, an additional SIGMATEK module was created which provide Python instructions especially for LASAL CLASS 2. The following instruction must be implemented it the script file:

import sigmatek.lasal.batch as batch

import sigmatek.lasal.classes as classes

The addition of "as batch" is assigned so that the complete module name doesn't have to be used.

The module "classes" expands the function of "batch" to communicate with the LASAL CLASS 2 program.

Using the **help (<Command>)** entry called in the **Python Script** command interpreter window, a description of the script instructions can be viewed. (i.e.: >>> help (batch.LoadProject))

By entering **dir(<Modul>)**, all commands contained in the module that can be searched with the **help(<Command>)** are displayed.

In the case of errors, whether commands return an error message or a Boolean return value or not can be set over the **batch.SetExceptionOnError(bool) method (default = false)**. With **batch.IsExceptionOnError()** setting can be queried.



All commands except for LoadProject and CreateProject supply a Boolean return value as to whether they were successfully processed. LoadProject and CreateProject return **None** if they cannot be executed.

The following instructions are in the **"batch"** module and can be used with the above import instruction (example: myPrj = batch.CreateProject (r "C:\temp\test"), respectively, batch.Compile(myPrj)).

The possibility exists, to deactivate the Escape-Sequence function in which **"r"** is written before the text (e.g. **r "C:\Lasal2\..."**). Otherwise the character string "\t" at the previous paragraph would be interpreted as Escape-Sequence.

3.12.3.1 AddFile

AddFile(prjDst, strFilename, bCopyToAddFiles=False, strAddFilePath="")

Adds a file to a project. If only the first two parameters are used, only a reference to the file is created. If bCopyToAddFiles = **True**, the file is physically copied to the project folder "**Additional Files**".

Transfer parameters	Type	Description
prjDst	Project	Name of the target project in which the file is to be added
strFilename	String	Name of the file that will be added to the project by reference
bCopyToAddFiles	Boolean	Physically copies the file to the "**Additional Files**" project folder. Optional parameter, default = **False**
strAddFilePath	String	Path in the "**Additional Files**" folder where to copy to. Optional parameter, default = ""
Return parameters	**Type**	**Description**

	Boolean	**True**: successful **False,** or **Exception**: failed

Example

```
import os.path as path
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sigmatek.lasal.classes as sig

def TestAddFile():
        # Initialize paths.
        strFile = "\\Source\\TestST.st"

        # Get test project.
        prj = classes.TheApplication().Project
        if prj == None :
                print "Project \"%s\" not loaded." % prj.Filename
                raise batch.BatchError

        strFilePath = path.dirname(prj.Filename) + strFile

        # Reference to the file
        if batch.AddFile(prj, strFilePath) == False :
                print "Adding file \"%s\" to project failed." % strFilePath
                raise batch.BatchError

        # The file will be physically added to the Additional Files folder
        if batch.AddFile(prj, strFilePath, True, "MySubdirectory") == False :
                print "Adding file \"%s\" to project failed." % strFilePath
                raise batch.BatchError

        strFilePath = path.dirname(prj.Filename) + strFile

        if batch.AddFile(prj, strFilePath) == False :
                print "Adding file \"%s\" to project failed." % strFilePath
                raise batch.BatchError

        strFilePath = path.dirname(prj.Filename) + strFile

        # Reference to the file
        if batch.AddFile(prj, strFilePath) == False :
                print "Adding file \"%s\" to project failed." % strFilePath
                raise batch.BatchError

        # The file will be physically added to the Additional Files folder
        if batch.AddFile(prj, strFilePath, True, "MySubdirectory") == False :
                print "Adding file \"%s\" to project failed." % strFilePath
                raise batch.BatchError

if __name__=="__main__" :
  TestAddFile()
```

3.12.3.2 AddLibrary

AddLibrary(prjDst, prjLibrary)

Adds a project to the library.

Transfer parameters	Type	Description
prjDst	Project	Target project, which should be added to the library
prjLibrary	Project / String	The library to which the project should be added (Project/String)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False,** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestAddLibrary():

        print "TestAddLibrary:"

        # Init paths.
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestAddLibrary"
        strLib = "\\TestLibrary"
        strPrjPath = (strDir + strPrj)
        strLibPath = (strDir + strLib)

        # Load the test project.
        prj = batch.LoadProject (strPrjPath)

        if prj == None :
                print "Loading project \"%s\" failed." % (strPrjPath)
                raise batch.BatchError

        # Load the test library
        lib = batch.LoadProject (strLibPath)

        if lib == None :
                print "Loading library \"%s\" failed." % (strLibPath)
                raise batch.BatchError

        #Add library to project:

        if batch.AddLibrary(prj, lib) == False :
                raise batch.BatchError
```

```
batch.CloseProject(lib);
batch.CloseProject(prj);

batch.CloseAllProjects();

print "TestAddLibrary successfull"
```

3.12.3.3 AddPrjConnectionFile

AddPrjConnectionFile(prjDst, strFilename)

Add a new or existing connection file to the project.

Transfer parameters	Type	Description
prjDst	Project	Target project to which the connection file should be added
strFilename	String	Name of the connection file, which should be added to the project
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def GetTemp() :

  return os.environ["TEMP"]

def TestConnectionFiles() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\ConnectionFiles"
  strPrjPath = (strDir + strPrj)

  bSuccess = True
```

```
print "strPrjPath: \"%s\"" % strPrjPath

# Load the project
prjDst = batch.LoadProject(strPrjPath)
if prjDst==None :
  bSuccess = False#raise batch.BatchError
  print "Loaded destination project %s" % prjDst.Filename

# Set old connection file inactive
strFilenameOld = "TestConFile1.lce"
if batch.SetPrjConnectionFileActive(prjDst, strFilenameOld, False)==False :
  print "Change active flag or connection file %s failed." % strFilenameOld
  bSuccess = False

# Add new connection file
strFilenameNew = "TestConFile2.lce"
strNetworkSubPath = "\\Network\\"
strFilepathNew = strPrjPath + strNetworkSubPath + strFilenameNew
if batch.AddPrjConnectionFile(prjDst, strFilepathNew)==False :
  print "Adding connection file %s failed." % strFilenameNew
  bSuccess = False#raise batch.BatchError

# Delete old connection file
if batch.DeletePrjConnectionFile(prjDst, strFilenameOld)==False :
  print "Delete connection file %s failed." % strFilenameOld
  raise batch.BatchError


batch.CloseAllProjects();

if bSuccess == False :
  raise batch.BatchError

print "Test successfull"
```

3.12.3.4 AddPrjDriveFile

AddPrjDriveFile(prjDst, strFilename, strObjectname, uiIndex=0)

Copies the file to the Drive folder and adds it to the project.

Transfer parameters	Type	Description
prjDst	Project	The target project
strFilename	String	Name and path of the drive parameter file that should be added to the project
strObjectname	String	Drive object name (axis)

uiIndex	Number	Index of the file in the series of files attached to the same drive object (Default = 0)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path

import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestDriveFileKonfiguration() :
  # Initalization
  batch.SetExceptionOnError(False)
  batch.CloseAllProjects()

  strAppPath = cls.TheApplication().Path

  # Current project
  #myPrj = cls.TheApplication().Project
  strPrj = os.path.join(strAppPath , r"..\Data\Scripting\DAandDriveTest")
  try :
    os.remove(os.path.join(strPrj, r"Drive\AKM11B-230V.lda"))
  except :
    print

  try :
    os.remove(os.path.join(strPrj, r"Drive\AKM11B-230V_0.lda"))
  except :
    print

  try :
    os.remove(os.path.join(strPrj, r"Drive\AKM21C-230V.lda"))
  except :
    print

  myPrj = batch.LoadProject(strPrj)
  if myPrj == None :
    raise batch.BatchError

  bSuccess = True

  batch.SetExceptionOnError(True)
  # Add a drive file
  batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM1\AKM11B
  batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM1\AKM11B
  batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM2\AKM21C
```

```
batch.SetExceptionOnError(False)
# Assign an object to a drive file
if batch.LinkObjectToPrjDriveFile(myPrj, "AKM11B-230V.lda", "Seppl") == True :
    bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "NoFile.lda", "NoObj") == True :
  bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "AKM11B-230V_0.lda", "_DriveAxis_21") == False :
  bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "AKM21C-230V.lda", "_DriveMng1") == True :
  bSuccess = False

# Delete the drive file
if batch.RemovePrjDriveFile(myPrj, r".\Drive\AKM11B-230V.lda") == False :
  bSuccess = False

if batch.RemovePrjDriveFile(myPrj, r".\Drive\NoFile-230V.lda") == True :
  bSuccess = False

if batch.RemovePrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM2\AK
  bSuccess = False

# End
print "Finished"
batch.CloseProject(myPrj)
if bSuccess == False :
      raise batch.BatchError
```

3.12.3.5 AddVirtualCommissioning

AddVirtualCommissioning(prjDst, strClassName, strObjectName)

Creates a new Virtual Commissioning object.

Transfer parameters	Type	Description
prjDst	Project	The target project to which the new element is to be added
strClassName	String	Name of the virtual commissioning class to be used
strObjectName	String	Object name to use
Return parameters	**Type**	**Description**
	Boolean	**True**: successfull **False** or **Exception**: failed

Example:

```
import sigmatek.lasal.classes as cls
import sigmatek.lasal.batch as batch

def AddVCObject() :
  prj = cls.TheApplication().Project
  prj.UseAdvancedIO = True
  batch.AddVirtualCommissioning(prj, "Class0", "DummySimPy")

#Main
if __name__ == "__main__" :
AddVCObject()
```

3.12.3.6 CalculateResourcesAndExport

CalculateResourcesAndExport(prjDst, strExportPath = "")

Calculates the necessary hardware resources and exports the result in several files (one for each bus system).

Transfer parameters	Type	Description
prjDst	Project	The target project
strExportPath	String	The export path. Default = "". Caution: **do not specify file names**, only the path, since the names are assigned by the command itself. A separate file in *.csv format is exported for each bus system.
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

# Loading a project
prj = batch.LoadProject("D:\\MyProject.lcp")

# Calculate and export hardware resources
batch.CalculateResourcesAndExport(prj, "D:\\Temp\\")
```

3.12.3.7 ChangeClass

ChangeClass(prjDst, keyNetwork, keyObject, keyClass)

Replaces the object of the original class with an object of the new class keeping all properties (object name, initialization values, ...).

Transfer parameters	Type	Description
prjDst	Project	The project in which the object is placed
keyNetwork	String	The network in which the object is placed
keyObject	String	The name of the object to be changed
keyClass	String	The name of the new class
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False,** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestChangeClass() :

        print "TestChangeClass:"

        # Initialize paths
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestChangeClass"
        strPrjPath = (strDir + strPrj)

        # Load the test project
        prj = batch.LoadProject (strPrjPath)

        if prj == None :
                print "Loading project \"%s\" failed." % (strPrjPath)
                raise batch.BatchError

        strClass1  = "Class0"
        strClass2  = "Class1"
        strObject = "NewObject"
        strNetwork = "NewNetwork"

        if batch.CreateNetwork(prj, strNetwork)==False :
                print "Create network %s failed." % strNetwork
                raise batch.BatchError
```

```
        # Create object
        iPosX   = 200
        iPosY   = 300
        if batch.CreateObject(prj, strNetwork, strClass1, strObject, iPosX, iPosY)==False :
                print "Create object %s failed." % strObject
                raise batch.BatchError

        # Switching the class
        if batch.ChangeClass(prj, strNetwork, strObject, strClass2)==False :
                print "Change class from %s to %s failed." % strClass1 % strClass2
                raise batch.BatchError

        # Compile
        if batch.Compile(prj)==False :
                print "Build %s failed." % prj.Filename
                raise batch.BatchError

        # Delete network
        if batch.DeleteNetwork(prj, strNetwork)==False :
                print "Delete network %s failed." % strNetwork
                raise batch.BatchError

        # Compile
        if batch.Compile(prj)==False :
                print "Build %s failed." % prj.Filename
                raise batch.BatchError

        batch.CloseAllProjects()

        print "TestChangeClass successful"


if __name__=="__main__" :
  TestChangeClass()
```

.

3.12.3.8 ChangeIOServer

ChangeIOServer (prjDst, strIOName, strSvrObject = "", strSvrChannel = "")

Changes the server connection of all IO objects with the given name.

Transfer parameters	Type	Description
prjDst	Project	Target project, which contains the IO objects to change
strIOName	String	Name of the IO object, for which the server connection should be changed

strSvrObject	String	Object name of the target server. (Default = "")
strSvrChannel	String	Channel name of the target server. (Default = "")
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sys
import _internal
from os import path
from sigmatek.lasal import batch
from sigmatek.lasal import classes

strDir = classes.TheApplication().Path + "..\\Data\\Scripting\\HWTree\\"
prj = batch.LoadProject (strDir)
if(prj == None) :
  print "no project"

if( batch.CreateIOObject(prj, "HW_Network", 200, 100, "eee", "class0X", "client") == False or
  _internal.msgbox ("Test failed!")

if(batch.ChangeIOServer(prj, "eee", "CSCP011_IM1", "Safe_Input2") == False) :
  _internal.msgbox ("Test failed!")

batch.CloseAllProjects()
```

3.12.3.9 ChangeLoadLibraryFlag

ChangeLoadLibraryFlag(prjDst, bLoadLibCompletely=True)

Sets the corresponding flag in the project properties, which determines whether all or only the required classes are loaded from the libraries.



Attention!

After sending this Python command, **a reload of all libraries** is also performed!

Transfer parameters	Type	Description
prjDst	Project	Target project, in which the library/ies should be inserted

bLoadLibCompletely	Boolean	**True**: all classes of the library are loaded **False**: only the classes needed by the project are loaded. Default = **True**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

# Ein Projekt laden
prj = batch.LoadProject("D:\\MyProject.lcp")

# Flag (Projekteigenschaft) auf True setzen
batch.ChangeLoadLibraryFlag(prj, True)
```

3.12.3.10 ChangeVirtualCommissioningClass

ChangeVirtualCommissioningClass(prjDst, strObjectName, strClassName)

Changes the class of a Virtual Commissioning object.

Transfer parameters	Type	Description
prjDst	Project	The project containing the Virtual Commissioning object
strObjectName	String	Name of the Virtual Commissioning object
strClassName	String	Name of the new class
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import lasal.classes as cls
import sigmatek.lasal.batch as batch

def ChangeVCObjectClass() :
  prj = cls.TheApplication().Project
  prj.UseAdvancedIO = True
  batch.AddVirtualCommissioning(prj, "Class0", "SimulationObjectName")
```

```
  batch.ChangeVirtualCommissioningClass(prj, "SimulationObjectName", "NewClass")

#Main
if __name__ == "__main__" :
ChangeVCObjectClass()
```

3.12.3.11 CleanupClassesAndLibraries

CleanupClassesAndLibraries(prjDst)

Deletes not used linked classes and not used libraries.

Transfer parameters	Type	Description
prjDst	Project	Name of the target project
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example:

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch         as batch

# Set the test directory
strDir = sig.TheApplication().Path + "..\\Data\\Scripting\\CleanupProjectDirectory\\"

# Load a "normal" project
prj = batch.LoadProject (strDir + "Normal")

# compile
batch.Compile (prj)

# Delete all unused classes and not used libraries from the project
batch.CleanupClassesAndLibraries(prj)

# Close project
batch.CloseProject (prj)
```

3.12.3.12 CleanupClasses

CleanupClasses(prjDst, listKeepClasses = [])

Delete unused classes from a project.

Transfer parameters	Type	Description
prjDst	Project	Is the target project
listKeepClasses	Objekt list	List of classes which should not be deleted. Default = [] (empty list)
Return parameters	**Type**	**Description**
	Boolean	**True**: successfull **False,** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch         as batch

# Set the test directory
strDir = sig.TheApplication().Path + "..\\Data\\Scripting\\CleanupProjectDirectory\\"

# Load a "normal" project
prj = batch.LoadProject (strDir + "Normal")

# compile
batch.Compile (prj)

# Delete all unused classes from the project, except those in the specified list
batch.CleanupClasses(prj, ["MySpecialClass"])

# Close project
batch.CloseProject (prj)
```

3.12.3.13 CleanupProjectDirectory

CleanupProjectDirectory(prjDst, bDeleteRestorableFiles=True, listKeepFiles)

Deletes non-referenced and restorable files – when bDeleteRestorableFiles = **true** – and closes the project. When using this instruction, take into consideration that non-referenced files are physically deleted and cannot be restored.

Transfer parameters	Type	Description

prjDst	Project	Name of the project whose directory should be cleaned
bDeleteRestorableFiles	Boolean	**True**: all unnecessary classes are deleted **False**: all unnecessary files except the files in the list **listKeepFiles** are deleted. Default = **True**
listKeepFiles	List	If **bDeleteRestorableFiles** = **False**, a list of the files that should not be deleted can be entered here Default= [] (empty list)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch         as batch

# Specify test directory
strDir = sig.TheApplication().Path + "..\\Data\\Scripting\\CleanupProjectDirectory\\"

# "normal" Projekt
prj = batch.LoadProject (strDir + "Normal")
batch.Compile (prj)
batch.CleanupProjectDirectory (prj, False, [strDir+"anyFile.txt"])
batch.CloseProject (prj)
```

3.12.3.14 ClearRamImage

ClearRamImage(strConnection)

Removes the RAM image information at the SPS.

Transfer parameters	Type	Description
strConnection	String	Name of the connection from the telephone book
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
```

```
#Use loaded GUI-Project
prjTest = classes.Project()

strIp = "10.10.116.184"

batch.ClearRamImage(strIp)

batch.Download(prjTest, strIp, True)
batch.Download(prjTest, strIp, False)

batch.Start(prjTest, strIp)

batch.Stop(prjTest, strIp)

batch.SaveProjectOnPLC(prjTest, strIp)

batch.CloseModemConnection()

batch.DownloadOS(strIp, "DummyOS")

del prjTest
```

3.12.3.15 CloseAllProjects

CloseAllProjects()

Closes all projects loaded via scripts.

Return parameters	Type	Description
none		

Example

```
import sigmatek.lasal.classes as sig
from sigmatek.lasal import batch

import shutil;

# Specify the test directory
strDir = sig.TheApplication().Path + "..\\Data\\Scripting\\CloseAllProjects\\"
shutil.rmtree(strDir, True)

# Create projects
prjDummy1 = batch.CreateProject (strDir + "Dummy1")
prjDummy2 = batch.CreateProject (strDir + "Dummy2")

# Close
batch.CloseAllProjects()
shutil.rmtree(strDir, True)
```

3.12.3.16 CloseLogfile

CloseLogfile()

Closes the Log file.

Return parameters	Type	Description
none		

Example

```
import sys
import os.path
import filecmp

import _internal
from sigmatek.lasal import batch

_internal.CloseAllProjects()

# make sure that logfile is in the same directory as script
strLogfile = os.path.join(os.path.dirname(sys.argv[0]),"LogTest.log")

_internal.OpenLogfile(strLogfile, "[(%p) %c] %m%n", False)
appPath = _internal.GetApplicationPath()

_internal.write("TestInternal")

_internal.logError("TestLogError")
_internal.logWarn("TestWarnung")
_internal.logInfo(appPath)

batch.CloseLogfile()

f = open(strLogfile, "r")

expectL1 = "[(ERROR) Scripting] TestLogError\n"
expectL2 = "[(WARN) Scripting] TestWarnung\n"

line = f.readline()                #1. line
if (line <> expectL1) :
        raise Exception
line = f.readline()                #2. line
if(line <> expectL2) :
        raise Exception
line = f.readline()                #3. line
if(line <> "[(INFO) Scripting] " + appPath + "\n") :
        raise Exception
```

```
f.close()
```

3.12.3.17 CloseModemConnection

CloseModemConnection()

Closes the connection via modem.

Return parameters	Type	Description
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

#Use loaded GUI-Project
prjTest = classes.Project()

strIp = "10.10.116.184"

batch.ClearRamImage(strIp)

batch.Download(prjTest, strIp, True)
batch.Download(prjTest, strIp, False)

batch.Start(prjTest, strIp)

batch.Stop(prjTest, strIp)

batch.SaveProjectOnPLC(prjTest, strIp)

batch.CloseModemConnection()

batch.DownloadOS(strIp, "DummyOS")

del prjTest
```

3.12.3.18 CloseProject

CloseProject(prjDst)

Closes a project.

Transfer parameters	Type	Description
prjDst	Project	Project which will be closed
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes     as sig
import sigmatek.lasal.batch               as batch
import shutil
import os.path

# Determine the test directory
strDir = sig.TheApplication().Path + "..\\Data\\Scripting\\ProjectManagement\\"
batch.SetExceptionOnError(True)

# 1. only CreateProject, without Project-Object it will be released when closing the
# application (by the ProjectManager)
# remove existing Project
if os.path.exists(strDir + "CreateProject1") == True :
    shutil.rmtree(strDir + "CreateProject1", False)
batch.CreateProject (strDir + "CreateProject1")

# 2. With del the project is also released only on exit
if os.path.exists(strDir + "CreateProject2") == True :
    shutil.rmtree(strDir + "CreateProject2", False)
prj2 = batch.CreateProject (strDir + "CreateProject2")
del prj2

# 3. CloseProject releases the project
if os.path.exists(strDir + "CreateProject3") == True :
    shutil.rmtree(strDir + "CreateProject3", False)
prj3 = batch.CreateProject (strDir + "CreateProject3")
batch.CloseProject (prj3)

# 4. via "constructor"
# In this case the responsibility lies with the script writer
# (projects created in this way are not managed by the ProjectManager)
# i.e. prj4 is destroyed by the GC.
prj4 = sig.Project()
```

3.12.3.19 Compile

Compile(prjDst, bRebuild = True, eOptions=batch.CompileOptions.Default)

Compiles a project.

Transfer parameters	Type	Description
prjDst	Project	The target project
bRebuild	Boolean	**True**: the entire project is compiles (default) **False**: only the changes are compiled
eOptions	ENUM	Various, freely combinable compiler options. Is only effective if the parameter **bRebuild** is set to **False**. Currently the following options are possible at this point: **'Default'**, **'RebuildAll'**, **'SuppressRebuild'**, **'UserClassesOnly'**. For more information see the chapter Build menu
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

The following compiler options are available at the *eOptions* parameter:

Compile option	Description
Default	Only the changes are compiled (currently "no option", the value of *bRebuild* is valid)
RebuildAll	Everything is compiled (built) completely new
SuppressRebuild	Suppresses a complete rebuild (see *RebuildAll*), does **not compile classes**, but **compiles all networks**! See the command CompileSuppressRebuild
UserClassesOnly	Only the user classes are compiled (no SIGMATEK classes)

Examples

```
import sigmatek.lasal.batch as batch

# compile
if batch.Compile(prj, True, batch.CompileOptions.Default)==False :
        print "Build %s failed." % prj.Filename
        raise batch.BatchError

print "TestCompile successful"
```

```
import sigmatek.lasal.batch as batch

# compile
```

```
if batch.Compile(prj, True, batch.CompileOptions.SuppressRebuild | batch.CompileOption
        print "Build %s failed." % prj.Filename
        raise batch.BatchError

print "TestCompile successful"
```

3.12.3.20 CompileSuppressRebuild

CompileSuppressRebuild(prjDst, eOptions=batch.CompileOptions.Default)

Does not compile any project classes, but all networks. This behavior also takes place when global header files have changed.



Tip

This command exists only for compatibility reasons, with the command Compile and *eOptions*=**batch.CompileOptions.SuppressRebuild** the complete functionality can be covered

Transfer parameters	Type	Description
prjDst	Project	The target project
eOptions	ENUM	Various, freely combinable compiler options. Currently the following options are possible at this point: **'Default'**, **'RebuildAll'**, **'SuppressRebuild'**, **'UserClassesOnly'**. For more information see the description in command Compile and the chapter Build menu
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import _internal
import filecmp
import shutil
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch as batch

# Specify the test directory
strDir = sig.TheApplication().Path + "..\\Data\\Scripting\\CmdBuild\\"
#DEBUG _internal.msgbox ("Test directory: " + strDir)

# Load source project and update binary files = RebuildAll
prjSrc = batch.LoadProject (strDir + "PrjSrc")
batch.Compile (prjSrc, True, batch.CompileOptions.Default)

# Create target project (delete before)
```

```
shutil.rmtree (strDir + "PrjDst", True)
prjDst = batch.CreateProject (strDir + "PrjDst")
#_internal.msgbox ("Hello!")

# Copy + change network
batch.CopyNetwork  (prjDst, prjSrc, "Nw1")
batch.SetInitValue (prjDst, "Nw1", "O1", "ClassSvr", "101")
# The class C1 must not be compiled (although channels.h (e.g.) has changed)
batch.CompileSuppressRebuild (prjDst, batch.CompileOptions.Default)
#DEBUG _internal.msgbox ("Hello!")
# not even later
batch.Compile (prjDst, False, batch.CompileOptions.Default)

# Determine whether class LOB has changed (= has been compiled)
#DEBUG _internal.msgbox (prjDst.Path)
strClassPath = "Class\\C1\\C1.lob"
bOK = filecmp.cmp (prjDst.Path + strClassPath, prjSrc.Path + strClassPath, 0)
if bOK == False:
        _internal.msgbox ("Test failed!")

batch.CloseProject (prjDst);
batch.CloseProject (prjSrc);
```

3.12.3.21 CopyAll

CopyAll(prjDst, prjScr, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False)

Copies all classes and networks from the pjrScr project to the prjDst-prj project.

Transfer parameters	Type	Description
prjDst	Project	The target project to which everything should be copied to
prjSrc	Project	The source project from which everything should be copied
bOverwrite	Boolean	**True** = overwrites already existing classes, networks and global types in the target project with the same name. Default = **True**
bCheckRevision	Boolean	**True** = Copy instruction tests the version number of the classes and copies only when they are different (version in the library is larger than that in the project). bCheckRevision is only important when the bOverwrite argument is **True**. Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activates the option of the same name from the import dialog. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful

		False or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\Li
bat.CopyAll (sig.TheApplication().Project, prjSrc)
bat.CloseProject (prjSrc)
```

3.12.3.22 CopyClass

CopyClass(prjDst, prjSrc, keyClass, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False, bKeepDependendClassesLinked=False)

Copies a class into a project.

Transfer parameters	Type	Description
prjDst	Project	The target project to which the class should be added
prjSrc	Project	The source project from which the class comes
keyClass	String	The name of the class that should be copied
bOverwrite	Boolean	**True** = overwrites an existing class with the same name to the target project. Default=**True**
bCheckRevision	Boolean	**True** = copy instruction checks the revision number of the class and copies the class only if the revision number is different (version in the library is greater than in the project). bCheckRevision is only significant when bOverwrite is F**alse**. Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activate the like named option with from the Import dialog (Default = **False**)
bKeepDependendClassesLinked	Boolean	**True** = Does NOT copy the linked classes (Default = **False**)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch

def GetTemp() :

  return os.environ["TEMP"]


def TestCopyClass() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  strDst = os.path.join(strCurDir,"CopyClassTest")
  # delete if already there
  shutil.rmtree(strDst, True)
  # create
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    raise batch.BatchError
  print "Created empty destination project %s" % prjDst.Filename

  # open the import project
  strSrc = os.path.join(os.path.dirname(sys.argv[0]), r"..\..\Runtime\Library\OS Interface")
  prjSrc = batch.LoadProject(strSrc)
  if prjSrc==None :
    raise batch.BatchError
  print "Loaded source project %s" % prjSrc.Filename

  bSuccess = True

  # copy class
  strClass = "_Grafix"
  if batch.CopyClass(prjDst, prjSrc, strClass)==False :
    print "Copy class %s failed." % strClass
    bSuccess = False

  # Close the source project again
  del prjSrc

  if bSuccess == False :
        raise batch.BatchError

  # compile
  if batch.Compile(prjDst)==False :
    print "Build %s failed." % prjDst.Filename
    raise batch.BatchError
```

```
# comparison
strRef = os.path.join(strCurDir, "CopyClassRef")
dc = filecmp.dircmp(strDst, strRef)
diff = dc.diff_files
if ( len(diff)>0 ) :
  print "Filecompare failed for files %s" % str(diff)
  raise batch.BatchError

print "Test successfull"
```

3.12.3.23 CopyClassList

CopyClassList(prjDst, prjSrc, keyClassList, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False)

Copies a list of classes to a project.

Transfer parameters	Type	Description
prjDst	Project	The project to which the classes should be added
prjSrc	Project	The source project from which the classes come
keyClassList	List	The list of names of the classes to be copied
bOverwrite	Boolean	**True** = overwrite an existing class in the target project with the same name. Default = **True**
bCheckRevision	Boolean	**True** = Copy instruction tests the version number of the classes and copies only when they are different (version in the library is larger than that in the project). bCheckRevision is only important when the bOverwrite argument is **True.** Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activates the option of the same name from the import dialog. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\Li
bat.CopyClassList (sig.TheApplication().Project,
                   prjSrc,
                   ["C1", "C2"])
```

```
bat.CloseProject (prjSrc
```

3.12.3.24 CopyNetwork

CopyNetwork(prjDst, prjSrc, keyNetwork, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False)

Copies a network with all the required classes to a project.

Transfer parameters	Type	Description
prjDst	Project	The project to which the network should be added
prjSrc	Project	The project from which the network comes
keyNetwork	String	The name of network to be copied
bOverwrite	Boolean	**True** = overwrites an existing network, respectively, existing classes in the target project with the same name (Default = **True**)
bCheckRevision	Boolean	**True** = Copy instruction tests the version number of the classes and copies only when they are different (version in the library is larger than that in the project). bCheckRevision is only important when the bOverwrite argument is **True.** Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activates the option of the same name from the import dialog (Default = **False**)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\
bat.CopyNetworkList (sig.TheApplication().Project,
                                          prjSrc,
                                          ["NW1", "NW2"])
bat.CloseProject (prjSrc)
```

3.12.3.25 CopyNetworkAndLinkClasses

CopyNetworkAndLinkClasses(prjDst, prjSrc, keyNetwork, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False)

Copies a network with all classes required to a project. The classes are then linked during this process.

Transfer parameters	Type	Description
prjDst	Project	The target project to which the network should be added
prjSrc	Project	The source project from which the network originates
keyNetwork	String	The name of the network that should be copied
bOverwrite	Boolean	**True** = Overwrites an existing network with the same name in the target project. Classes already existing as copy are replaced by links at name sameness. Default = **True**
bCheckRevision	Boolean	**True** = Copy instruction tests the version number of the classes and copies only when they are different (version in the library is larger than that in the project). bCheckRevision is only important when the bOverwrite argument is **True.** Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activates the option of the same name from the import dialog. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\Li
bat.CopyNetworkAndLinkClasses (sig.TheApplication().Project,
                                                              prjSrc,
                                                              "NW1")
bat.CloseProject (prjSrc)
```

3.12.3.26 CopyNetworkList

CopyNetworkList(prjDst, prjSrc, keyNetworkList, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False)

Copies a list of networks with the required classes to the project.

Transfer parameters	Type	Description
prjDst	Project	The target project to which the network should be added
prjSrc	Project	The source project from which the network originates
keyNetworkList	List	The list of network names that should be copied
bOverwrite	Boolean	**True** = Overwrites already existing networks, respectively, already existing classes with the same name in the target project. Default = **True**
BCheckRevision	Boolean	**True** = Copy instruction tests the version number of the classes and copies only when they are different (version in the library is larger than that in the project). bCheckRevision is only important when the bOverwrite argument is **True.** Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activates the option of the same name from the import dialog. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\
bat.CopyNetworkList (sig.TheApplication().Project,
                                          prjSrc,
                                          ["NW1", "NW2"])
bat.CloseProject (prjSrc)
```

3.12.3.27 CopyNetworkListAndLinkClasses

CopyNetworkListAndLinkClasses(prjDst, prjSrc, keyNetworkList, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False)

Copies a list of networks with the required classes to the project. Required classes are linked if necessary.

Transfer parameters	Type	Description
prjDst	Project	The target project to which the network should be added
prjSrc	Project	The source project from which the network originates
keyNetworkList	List	The list of network names that should be copied
bOverwrite	Boolean	**True** = Overwrites an existing network with the same name in the target project. Classes already existing as copy are replaced by links at name sameness. Default = **True**
bCheckRevision	Boolean	**True** = Copy instruction tests the version number of the classes and copies only when they are different (version in the library is larger than that in the project). bCheckRevision is only important when the bOverwrite argument is **True.** Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activates the option of the same name from the import dialog. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\Li
bat.CopyNetworkListAndLinkClasses (sig.TheApplication().Project,
                                                                    prjSrc,
                                                                    ["NW1", "NW2"])
bat.CloseProject (prjSrc)
```

3.12.3.28 CreateBootdisk

CreateBootdisk(prjDst, strPath, bOverwrite=False, bAllLobsInOne=False, iPlatformType=-1)

Creates a bootdisk in the defined directory.

Transfer parameters	Type	Description
prjDst	Project	The target project
strPath	String	The path to which the boot disk information is to be written
bOverwrite	Boolean	Should eventually existent information be overwritten? Default = **False**
bAllLobsInOne	Boolean	With this option one single LOB file is created instead of several LOB files of the program modules. Thus booting via USB stick is faster. Default = **False**
iPlatformType	Number	Architecture type. Defines the platform/s, for which the bootdisk should be created. **-1** **AUTO**, uses the defined platform **0** **ALL**, for the ARM as well as for the X86 platform the according files are created in an own folder for each **1** **ARM**, the bootdisk is only created for the ARM platform **2** **X86**, the bootdisk is only created for the X86 platform Default = **-1**, **AUTO**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestCreateBootdisk_WithoutPrj():

        print "TestCreateBootdisk_WithoutPrj:"

        # Initialize paths
```

```
        strPathToDelete = classes.TheApplication().Path + "..\\Data\\Scripting\\TestCreateBoot
        strPathToDelete_BDFolder = strPathToDelete + "\\TestPython1"

        # The command CmdCreateBootdisk is only executed if a project is specified. Therefore
        if batch.CreateBootdisk(None, strPathToDelete_BDFolder, False, False, 0) == True :
                print "TestCreateBootdisk_WithoutPrj Nr 1 failed"
                raise batch.BatchError
        if batch.CreateBootdisk(None, strPathToDelete_BDFolder, False, True, 1) == True :
                print "TestCreateBootdisk_WithoutPrj Nr 2 failed"
                raise batch.BatchError

        print "TestCreateBootdisk_WithoutPrj successful"

        return True

def TestCreateBootdisk_InklPrj():

        print "TestCreateBootdisk_InklPrj:"

        # Initialize paths
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestCreateBootdisk"
        strPrjPath = (strDir + strPrj)
        strPathToDelete = strDir + strPrj + "\\_ToDelete"

        # Load the test project
        prj = batch.LoadProject (strPrjPath)

        if prj == None :
                print "Loading project \"%s\" failed." % (strPrjPath)
                raise batch.BatchError

    # compile
        if batch.Compile(prj) == False :
          print "Build %s failed." % prj.Filename
          raise batch.BatchError

        if batch.CreateBootdisk(prj, strPathToDelete + "\\TestPython2", True, True, 2) == Fals
                print "TestCreateBootdisk_InklPrj Nr 1 failed"
                raise batch.BatchError

        if batch.CreateBootdisk(prj, strPathToDelete + "\\TestPython2", True, False, 0) == Fal
                print "TestCreateBootdisk_InklPrj Nr 2 failed"
                raise batch.BatchError

        batch.CloseProject(prj);

        print "TestCreateBootdisk_InklPrj successful"
```

3.12.3.29 CreateBootdiskMngr

CreateBootdiskMngr(prjDst, strPath, bOverwrite, bAllLobsInOne, iType, bAddAutoexec, strPathVisu, strFileOS, strFileAutoexec="", bUseActiveConnectionFiles=True, iPlatformType=-1, bAddMultiMasterStationsTXT=True, rstrFolderVisu_LVD="", bLVDRuntimeFlag=True)

Creates a bootdisk with the following optional parts:

- LASAL-Project
- LSE-Project (Visualization)
- Operating system Image (for operating system update)
- Boot menu

Transfer parameters	Type	Description
prjDst	Project	A LASAL project. If the LASAL project is not necessary for the bootdisk, "None" can be transferred
strPath	String	Path to a directory in which the bootdisk files are stored
bOverwrite	Boolean	Should eventually existing information be overwritten?
bAllLobsInOne	Boolean	With this option one single LOB file is created instead of several LOB files of the program modules. Thus booting via USB stick is faster. Possible values: **True** or **False**
iType	Number	Type of the update script (Rexx script) to be added: 0 No script will be added. 1 Script will be added, through which all existent is updated 2 Script will be added, with which a boot menu on the target system is available. Here the update can be selected then
bAddAutoexec	Boolean	Specifies if a Standard autoexec.lsl, which starts the Update script (Rexx script), should be copied to the output directory. Possible values: **True** or **False**
strPathVisu	String	Path to the Runtime directory of a LSE project. If not required a Null string can be transferred here ("")
strFileOS	String	Path and file name of an operating system file. If not required a Null string can be transferred here ("")

strFileAutoexec	String	Path and file name of an autoexec.lsl file replacing the file on the control. If this file is not required nothing, respectively, a Null string ("") can be transferred here
bUseActiveConnectionFiles	String	If = **True**, the connection files active in the project are used. (Default = **True**)
iPlatformType	Number	Defines the platform/s, for which the bootdisk should be created. **-1** **AUTO**, uses the defined platform **0** **ALL**, for the ARM as well as for the X86 platform the according files are created in an own folder for each **1** **ARM**, the bootdisk is only created for the ARM platform **2** **X86**, the bootdisk is only created for the X86 platform (Default = **-1**, **AUTO**)
bAddMultiMasterStationsTXT	Boolean	If **True** the MultiMasterStations.txt will be copied into the destination directory (lsldata folder). Default: **True**
rstrFolderVisu_LVD	String	Path to the directory 'Publish' of LVD project. If not needed, this parameter can be empty. Default = ""
bLVDRuntimeFlag	Boolean	If **True** and rstrFolderVisu_LVD not empty, the 'Runtime' directory of LVD project is used. Default=**True**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestCreateBootdiskViaMngr_WithoutPrj():

  print "TestCreateBootdiskViaMngr:"
  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\TestCreateBootdisk"

  strPathSourcen = strDir + strPrj + "\\_SOURCEN"
  strPathToDelete = strDir + strPrj + "\\_ToDelete"
  strVisu = strPathSourcen + "\\Visu_Runtime"
  strOS = strPathSourcen + "\\OS_CIPC_01.03.051\\CIPCqvga.lbi"
```

```
  strAutoexec = strPathSourcen + "\\autoexec.lsl"

  if batch.CreateBootdiskMngr(None, strPathToDelete + "\\TestPython3", False, False, 2, Fals
    print "TestCreateBootdiskViaMngr1_WithoutPrj Nr 1 failed"
    raise batch.BatchError

  if batch.CreateBootdiskMngr(None, strPathToDelete + "\\TestPython3", False, False, 0, Fals
                print "TestCreateBootdiskViaMngr1_WithoutPrj Nr 2 failed"
                raise batch.BatchError

  print "TestCreateBootdiskViaMngr_WithoutPrj successful"

def TestCreateBootdiskMngr_InklPrj():

        print "TestCreateBootdiskMngr_InklPrj:"

        # Initialize paths
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestCreateBootdisk"

        strPathSourcen = strDir + strPrj + "\\_SOURCEN"
        strPathToDelete = strDir + strPrj + "\\_ToDelete"
        strVisu = strPathSourcen + "\\Visu_Runtime"
        strOS = strPathSourcen + "\\OS_CIPC_01.03.051\\CIPCqvga.lbi"
        strAutoexec = strPathSourcen + "\\autoexec.lsl"

        # Load the test project
        prj = batch.LoadProject (strDir + strPrj)
        if prj == None :
                print "Loading project \"%s\" failed." % (strDir + strPrj)
                raise batch.BatchError

        # compile
        if batch.Compile(prj) == False :
                print "Build %s failed." % prj.Filename
                raise batch.BatchError

        #Definition:
        #CreateBootdiskMngr(prjDst, strPath, bOverwrite, bAllLobsInOne, iType, bAddAutoexec,

        if batch.CreateBootdiskMngr(prj, strPathToDelete + "\\TestPython4", False, False, 1,
                print "TestCreateBootdiskMngr_InklPrj Nr 1 failed"
                raise batch.BatchError

        if batch.CreateBootdiskMngr(prj, strPathToDelete + "\\TestPython4", False, True, 2,
                print "TestCreateBootdiskMngr_InklPrj Nr 2 failed"
                raise batch.BatchError

        batch.CloseProject(prj);


        print "TestCreateBootdiskMngr_InklPrj successful"


def TestCreateBootdiskMngr1_InklPrj():
```

```
        print "TestCreateBootdiskMngr1_InklPrj:"

        # Initialize paths
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestCreateBootdisk"

        strPathSourcen = strDir + strPrj + "\\_SOURCEN"
        strPathToDelete = strDir + strPrj + "\\_ToDelete"
        strVisu = strPathSourcen + "\\Visu_Runtime"
        strOS = strPathSourcen + "\\OS_CIPC_01.03.051\\CIPCqvga.lbi"
        strAutoexec = strPathSourcen + "\\autoexec.lsl"

        # Load the test project
        prj = batch.LoadProject (strDir + strPrj)
        if prj == None :
                print "Loading project \"%s\" failed." % (strDir + strPrj)
                raise batch.BatchError

        # compile
        if batch.Compile(prj) == False :
                print "Build %s failed." % prj.Filename
                raise batch.BatchError

        if batch.CreateBootdiskMngr(None, strPathToDelete + "\\TestPython4", False, False, 1,
                print "TestCreateBootdiskMngr1_InklPrj Nr 1 failed"
                raise batch.BatchError

        if batch.CreateBootdiskMngr(None, strPathToDelete + "\\TestPython4", False, True, 2, T
                print "TestCreateBootdiskMngr1_InklPrj Nr 2 failed"
                raise batch.BatchError

        batch.CloseProject(prj);

        print "TestCreateBootdiskMngr1_InklPrj successful"


def TestCreateBootdiskMngr1_InklLVDPrj():

        print "TestCreateBootdiskMngr1_InklLVDPrj:"

        # Initilaize paths
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestCreateBootdisk"

        strPathSourcen = strDir + strPrj + "\\_SOURCEN"
        strPathToDelete = strDir + strPrj + "\\_ToDelete"
        strVisuLVD = strPathSourcen + "\\Visu_LVD"

        # Test 1 : with LVD-Project, with LVD-Runtime.
        if batch.CreateBootdiskMngr(None, strPathToDelete + "\\TestPython5", False, False, 1,
                print "TestCreateBootdiskMngr1_InklLVDPrj Nr 1 failed"
                raise batch.BatchError

        # Test 2 : with LVD-Project, without LVD-Runtime.
        if batch.CreateBootdiskMngr(None, strPathToDelete + "\\TestPython5", False, False, 2,
                print "TestCreateBootdiskMngr1_InklLVDPrj Nr 2 failed"
```

```
                raise batch.BatchError
        print "TestCreateBootdiskMngr1_InklLVDPrj successful"
```

3.12.3.30 CreateBootdiskViaContainer

CreateBootdiskViaContainer(prjDst, strPath, dictParameters)

Creates a bootdisk in the defined directory.

Transfer parameters	Type	Description
prjDst	Project	The target project
strPath	String	The path to which the boot disk information is to be written
dictParameters	Dictionary (Map)	Dictionary with all parameters to be set
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def GetParameters():

        dict = batch.GetCreateBootdiskContainer()
      print dict
      dict["Platform"] = "X86"
      print dict

def CallCreateBootDisk():

        dict = batch.GetCreateBootdiskContainer()
      dict["Platform"] = "X86"
      dict["AdditionalFiles"] = [(r"D:\py\test.xml",r"C:\temp"), (r"D:\temp\lalala.txt",r"C:
      dict["UseAdditionalFiles"] = "true"
      print dict
      batch.CreateBootdiskViaContainer(None, r"D:/todelete/BDPC/", dict)
#Main
        if __name__ == "__main__" :
      reload(batch)
      GetParameters()
      CallCreateBootDisk()
```

3.12.3.31 CreateConnection

CreateConnection(prjDst, strSourceNetworkName, strSourceObjectName, strSourceClientName, strTargetNetworkName, strTargetObjectName, strTargetServerName)

Creates a new connection between a client and server channel.

Transfer parameters	Type	Description
prjDst	Project	The target project
strSourceNetworkName	String	The name of the network from which the connection starts
strSourceObjectName	String	The name of the object containing the client
strSourceClientName	String	The name of the client channel from which the connection starts
strTargetNetworkName	String	The name of the network in which the connection ends
strTargetObjectName	String	The name of the object containing the server channel, or name of the multimaster object
strTargetServerName	String	The name of the server at which the connection ends, or empty when using multimaster object
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestCreateConnection() :
        print "Test CreateConnection."

        # Initalizing
        batch.SetExceptionOnError(False)
        batch.CloseAllProjects()

        # Current projekt
        strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestConnections"
        myPrj = batch.LoadProject(strPrj)
        if myPrj == None :
                print "Loading test project failed."
                raise batch.BatchError
```

```
# Test 1: Create valid connections within a network
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_DINT", "NW2", "Class01", "S_DI
        print "CreateConnection between NW2.Class11.C_DINT and NW2.Class01.S_DINT fa
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Object", "NW2", "Class01", "Cl
        print "CreateConnection between NW2.Class11.C_Object and NW2.Class01.ClassSv
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Command", "NW2", "Class01", "S
        print "CreateConnection between NW2.Class11.C_Command and NW2.Class01.S_Comm
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_BDINT", "NW2", "Class01", "S_B
        print "CreateConnection between NW2.Class11.C_BDINT and NW2.Class01.S_BDINT
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Real", "NW2", "Class01", "S_Re
        print "CreateConnection between NW2.Class11.C_Real and NW2.Class01.S_Real fa
        raise batch.BatchError
# Connection may not be created because no client is specified
if batch.CreateConnection2(myPrj, "NW2", "Class11", "", "Class01", "ClassSvr") == Tr
        print "CreateConnection2 between Class11 and Class01.ClassSvr failed, client
        raise batch.BatchError

# Test2: Create valid connections between two networks
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_DINT", "NW1", "Class02", "S_DI
        print "CreateConnection between NW2.Class11.C_DINT and NW1.Class02.S_DINT fa
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Object", "NW1", "Class02", "Cl
        print "CreateConnection between NW2.Class11.C_Object and NW1.Class02.ClassSv
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Command", "NW1", "Class02", "S
        print "CreateConnection between NW2.Class11.C_Command and NW1.Class02.S_Comm
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_BDINT", "NW1", "Class02", "S_B
        print "CreateConnection between NW2.Class11.C_BDINT and NW1.Class02.S_BDINT
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Real", "NW1", "Class02", "S_Re
        print "CreateConnection between NW2.Class11.C_Real and NW1.Class02.S_Real fa
        raise batch.BatchError

# End
print "Finished"
batch.CloseProject(myPrj)
```

3.12.3.32 CreateConnection2

CreateConnection2(prjDst, strSourceObjectName, strSourceClientName, strTargetObjectName, strTargetServerName)

Creates a new connection between a client and server channel.

Transfer parameters	Type	Description

prjDst	Project	The target project
strSourceObjectName	String	The name of the object containing the client
strSourceClientName	String	The name of the client channel where the connections starts
strTargetObjectName	String	The name of the object containing the server channel, or name of the multimaster object
strTargetServerName	String	The name of the server where the connection ends, or empty when using multimaster object
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestCreateConnection() :
        print "Test CreateConnection."

        # Initalizing
        batch.SetExceptionOnError(False)
        batch.CloseAllProjects()

        # Current project
        strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestConnections"
        myPrj = batch.LoadProject(strPrj)
        if myPrj == None :
                print "Loading test project failed."
                raise batch.BatchError

        # Test1: Create valid connections within a network
        if batch.CreateConnection2(myPrj, "Class11", "C_DINT", "Class01", "S_DINT") == False :
                print "CreateConnection2 between Class11.C_DINT and Class01.S_DINT failed."
                raise batch.BatchError
        if batch.CreateConnection2(myPrj, "Class11", "C_Object", "Class01", "ClassSvr") == Fal
                print "CreateConnection2 between Class11.C_Object and Class01.ClassSvr failed
                raise batch.BatchError
        if batch.CreateConnection2(myPrj, "Class11", "C_Command", "Class01", "S_Command") == F
                print "CreateConnection2 between Class11.C_Command and Class01.S_Command faile
                raise batch.BatchError
        if batch.CreateConnection2(myPrj, "Class11", "C_BDINT", "Class01", "S_BDINT") == False
                print "CreateConnection between Class11.C_BDINT and Class01.S_BDINT failed."
                raise batch.BatchError
        if batch.CreateConnection2(myPrj, "Class11", "C_Real", "Class01", "S_Real") == False :
                print "CreateConnection2 between Class11.C_Real and Class01.S_Real failed."
                raise batch.BatchError
        # Connection may not be created because no client is specified
```

```
if batch.CreateConnection2(myPrj, "Class11", "", "Class01", "ClassSvr") == True :
        print "CreateConnection2 between Class11 and Class01.ClassSvr failed, client
        raise batch.BatchError

# Test2: Create valid connections between two networks
if batch.CreateConnection2(myPrj, "Class11", "C_DINT", "Class02", "S_DINT") == False
        print "CreateConnection between Class11.C_DINT and Class02.S_DINT failed."
        raise batch.BatchError
if batch.CreateConnection2(myPrj, "Class11", "C_Object", "Class02", "ClassSvr") == F
        print "CreateConnection2 between Class11.C_Object and Class02.ClassSvr faile
        raise batch.BatchError
if batch.CreateConnection2(myPrj, "Class11", "C_Command", "Class02", "S_Command") ==
        print "CreateConnection2 between Class11.C_Command and Class02.S_Command fai
        raise batch.BatchError
if batch.CreateConnection2(myPrj, "Class11", "C_BDINT", "Class02", "S_BDINT") == Fal
        print "CreateConnection2 between Class11.C_BDINT and Class02.S_BDINT failed.
        raise batch.BatchError
if batch.CreateConnection2(myPrj, "Class11", "C_Real", "Class02", "S_Real") == False
        print "CreateConnection2 between Class11.C_Real and Class02.S_Real failed."
        raise batch.BatchError

# End
print "Finished"
batch.CloseProject(myPrj)
```

3.12.3.33 CreateIOObject

CreateIOObject(prjDst, keyNetwork, xPos, yPos, strIOName, strCltObject, strCltChannel)

Creates a new IO object with the specified coordinates.

Transfer parameters	Type	Description
prjDst	Project	The target project
keyNetwork	String	The name of the network, in which the object should be created
xPos	Number	The x-position of the IO object in the graphic network view
yPos	Number	The y-position of the IO object in the graphic network view
strIOName	String	The IO name of the new object
strCltObject	String	Object name of the client (Default is „“)
strCltChannel	String	Channel name of the client (Default is „“)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful

		False or **Exception**: failed

Example

```
import sys
import _internal
from os import path
from sigmatek.lasal import batch
from sigmatek.lasal import classes

strDir = classes.TheApplication().Path + "..\\Data\\Scripting\\HWTree\\"
prj = batch.LoadProject (strDir)
if(prj == None) :
        print "no project"
        raise batch.BatchError

if(batch.CreateIOObject(prj, "HW_Network", 100, 100, "aaa", "class04", "client") == False) :
        print("Test failed!")
        raise batch.BatchError

batch.Save(prj)
batch.CloseAllProjects()
print("Test successfull")
```

3.12.3.34 CreateLabel

CreateLabel(prjDst, strIOName, strSvrObject, strSvrChannel, strCltObject, strCltChannel)

Create a new I/O name in the Connection Manager.

Transfer parameters	Type	Description
prjDst	Project	The target project
strIOName	String	The IO name to be created
strSvrObject	String	Object name of the destination / server (Default = „“)
strSvrChannel	String	Channel name of the destination / server (Default = „“)
strCltObject	String	Object name of the source / client (Default is „“)
strCltChannel	String	Channel name of the source / client (Default is „“)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sys
import _internal
from os import path
from sigmatek.lasal import batch
from sigmatek.lasal import classes

strDir = classes.TheApplication().Path + "..\\Data\\Scripting\\HWTree\\"
prj = batch.LoadProject (strDir)
if(prj == None) :
  print "no project"
  raise batch.BatchError

classes.TheApplication().Project = prj;

if(batch.CreateLabel(prj, "eee", "CSCP011_IM1", "Safe_Input1", "class04", "client") == False
  _internal.msgbox ("Test failed!")
  raise batch.BatchError

batch.Save(prj)
batch.CloseAllProjects()
print("Test successfull")
```

3.12.3.35 CreateNetwork

CreateNetwork(prjDst, keyNetwork)

Create a new network.

Transfer parameters	Type	Description
prjDst	Project	The project in which the network should be placed
keyNetwork	String	The name of the new network
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch
```

```
import sigmatek.lasal.classes as classes

def GetTemp() :

  return os.environ["TEMP"]

def TestCreateNetwork() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  # strDst = os.path.join(strCurDir,"CreateNetworkTest")
  # delete if already there
  # shutil.rmtree(strDst, True)

  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\CreateNetworkTest"
  strPrjPath = (strDir + strPrj)

  print "strPrjPath: \"%s\"" % strPrjPath

  # Create
  prjDst = batch.LoadProject(strPrjPath)
  if prjDst==None :
    raise batch.BatchError
    print "Created empty destination project %s" % prjDst.Filename


  # Create network
  strNetwork = "NewNetwork"
  if batch.CreateNetwork(prjDst, strNetwork)==False :
    print "Create network %s failed." % strNetwork
    raise batch.BatchError

  # Set comment
  strcomment = "TestComment"
  if batch.SetCommentNetwork(prjDst, strNetwork, strcomment)==False :
    print "Set comment for network %s failed." % strNetwork
    raise batch.BatchError

  # compile
  if batch.Compile(prjDst)==False :
    print "Build %s failed." % prjDst.Filename
    raise batch.BatchError

  # Delete network
  if batch.DeleteNetwork(prjDst, strNetwork)==False :
    print "Delete network %s failed." % strNetwork
    raise batch.BatchError

  # compile
  if batch.Compile(prjDst)==False :
    print "Build %s failed." % prjDst.Filename
    raise batch.BatchError
```

```
batch.CloseAllProjects();

print "Test successfull"
```

3.12.3.36 CreateObject

CreateObject(prjDst, keyNetwork, keyClass, keyObject, xPos, yPos, blsVisualized = False)

Create a new object with the assigned coordinates.

Transfer parameters	Type	Description
prjDst	Project	The project that contains the network
keyNetwork	String	The name of the network
keyClass	String	The name of the class on which the object is based
keyObject	String	The name of the object to be created
xPos	Number	The x-position of the object in the graphical view
yPos	Number	The y-position of the object in the graphical view
blsVisualized	Boolean	Indicates whether or not the object created should be visualized. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import _batch

def TestCreateObject() :

  print "TestCreateObject:"

  # Initialize paths
```

```
strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
strPrj = "\\CreateObjectTest"
strDst = (strDir + strPrj)

prjDst = batch.LoadProject(strDst)
if prjDst==None :
  print "Load project failed"
  raise batch.BatchError

# Create network for filling
strNetwork = "NewNetwork"
if batch.CreateNetwork(prjDst, strNetwork)==False :
  print "Create network \"%s\" failed." % strNetwork
  raise batch.BatchError

# Create object
strClass  = "Class0"
strObject = "NewObject"
iPosX   = 200
iPosY   = 300
if _batch.CreateObject0(prjDst, strNetwork, strClass, strObject, iPosX, iPosY, False)==False
  print "Create object %s failed." % strObject
  raise batch.BatchError

# Set comment
strcomment = "TestComment"
if batch.SetCommentObject(prjDst, strNetwork, strObject, strcomment)==False :
  print "Set comment for object %s failed." % strObject
  raise batch.BatchError

# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError

if batch.ExistsObject (prjDst, strNetwork, strObject) == False :
  print "Object \"%s\" doesn't exist." % (strObject)
  raise batch.BatchError

if batch.DeleteObject(prjDst, strNetwork, strObject) == False :
  print "Delete Object \"%s\" failed" % (strObject)
  raise batch.BatchError

if batch.DeleteNetwork(prjDst, strNetwork)==False :
  print "Delete network \"%s\" failed." % strNetwork
  raise batch.BatchError

# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError

batch.CloseAllProjects();

print "TestCreateObject successfull"
```

3.12.3.37 CreateProject

CreateProject(strFilename, bBackupIncludsAndLoader=False)

Create a new project.

Transfer parameters	Type	Description
strFilename	String	Name of the project file
bBackupIncludsAndLoader	Boolean	**True** = Includes and the loader are stored in the project directory. Default = **False**
Return parameters	**Type**	**Description**
	Project	**Project**: successful **None** or **Exception**: failed

Example

```
import sigmatek.lasal.classes     as sig
import sigmatek.lasal.batch               as batch
import shutil
import os.path

# Set the test directory
strDir = sig.TheApplication().Path + "..\\Data\\Scripting\\ProjectManagement\\"
batch.SetExceptionOnError(True)

# 1. Only CreateProject, without project object it will be released when closing
# the application is released (by the ProjectManager)
# remove existing Project
if os.path.exists(strDir + "CreateProject1") == True :
    shutil.rmtree(strDir + "CreateProject1", False)
batch.CreateProject (strDir + "CreateProject1")

# 2. With del the project is also released only on exit
if os.path.exists(strDir + "CreateProject2") == True :
    shutil.rmtree(strDir + "CreateProject2", False)
prj2 = batch.CreateProject (strDir + "CreateProject2")
del prj2

# 3. CloseProject releases the project
if os.path.exists(strDir + "CreateProject3") == True :
    shutil.rmtree(strDir + "CreateProject3", False)
prj3 = batch.CreateProject (strDir + "CreateProject3")
batch.CloseProject (prj3)

# 4. via "Konstruktor"
```

```
# In this case, the responsibility lies with the script writer (such
# created projects are not managed by the ProjectManager).
# I.e., prj4 is destroyed by the GC.
prj4 = sig.Project()
```

3.12.3.38 DecryptConnectionFile

DecryptConnectionFile(strConnFile, strPassword)

Decrypts an already encrypted connection file.

Transfer parameters	Type	Description
strConnFile	String	Path and name of the file to decrypt
strPassword	String	Password for decrypting
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestConnFileEncrypter() :

  # Init paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting\\TestConnFileEncrypter"
  strConnFileName = strDir + "\\TestConnFileEncrypter.lcf"

  bSuccess = True

  # do Decrypt
  if batch.DecryptConnectionFile(strConnFileName, "TestPassword") == False :
    bSuccess = False
    print "Decrypt ConnFile failed"

  if bSuccess == False :
    raise batch.BatchError
```

3.12.3.39 DeleteClass

DeleteClass(prjDst, keyClass, bDeleteForce = False)

Deletes a class from a project. If **bDeleteForce** is set to T**rue**, all objects of the class are also deleted

Transfer parameters	Type	Description
prjDest	Project	The project that contains the class
keyClass	String	The name of the class to be deleted
bDeleteForce	Boolean	**True** ... The class and all of its objects are deleted **False**... If objects exist, an error message is issued. Default = **False**
Return parameters	**Type**	**Description**
none		

Example

```
import os
import os.path
import shutil
import sys
import sigmatek.lasal.batch as batch

def TestDeleteClass() :

  # To delete a class, one is imported first

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  strDst = os.path.join(strCurDir,"CopyClassTest")
  # Delete if already present
  shutil.rmtree(strDst, True)
  # Create
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    raise batch.BatchError
  print "Created empty destination project %s" % prjDst.Filename

  # Open the import project
  strSrc = os.path.join(os.path.dirname(sys.argv[0]), r"..\..\Runtime\Library\OS Interface")
  prjSrc = batch.LoadProject(strSrc)
  if prjSrc==None :
    raise batch.BatchError
  print "Loaded source project %s" % prjSrc.Filename

  bSuccess = True
```

```
  # Copy class
  strClass = "_Grafix"
  if batch.CopyClass(prjDst, prjSrc, strClass)==False :
    print "Copy class %s failed." % strClass
    bSuccess = False

  # Close the import project again
  del prjSrc

  if bSuccess == False :
        raise batch.BatchError


  # Now we have the class, we test deleting it with all its objects (bDeleteForce =
True)
  if batch.DeleteClass(prjDst, strClass, True)==False :
        print "Delete class %s failed." % strClass
        bSuccess = False

  # Succeed
  print "Delete class %s succeed." % strClass

  if __name__=="__main__" :
      TestDeleteClass()
```

3.12.3.40 DeleteConnection

DeleteConnection(prjDst, strNetworkName, strObjectName, strClientName)

Deletes a connection between two channels.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	The name of the network containing the object with the start of the connections
strObjectName	String	The name of the object containing the client from which the connection starts
strClientName	String	The name of the client, identified as the start point of the connection
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestDeleteConnection() :
        print "Test DeleteConnection."

        # Intializing
        batch.SetExceptionOnError(False)
        batch.CloseAllProjects()

        # Current project
        strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestConnections"
        myPrj = batch.LoadProject(strPrj)
        if myPrj == None :
                print "Loading test project failed."
                raise batch.BatchError

        # Test1. Deleting existing connections within a network with the DeleteConnection fu
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_DINT") == False :
                print "DeleteConnection of NW2.Class11.C_DINTfailed."
                raise batch.BatchError
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_Object") == False :
                print "DeleteConnection of NW2.Class11.C_Object failed."
                raise batch.BatchError
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_Command") == False :
                print "DeleteConnection of NW2.Class11.C_Command failed."
                raise batch.BatchError
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_BDINT") == False :
                print "DeleteConnection of NW2.Class11.C_BDINT failed."
                raise batch.BatchError
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_Real") == False :
                print "DeleteConnection of NW2.Class11.C_Real failed."
                raise batch.BatchError

        # Create the connections between two networks for the next test
        if batch.CreateConnection(myPrj, "NW2", "Class11", "C_DINT", "NW1", "Class02", "S_DI
                print "CreateConnection between NW2.Class11.C_DINT and NW2.Class01.S_DINT fa
                raise batch.BatchError
        if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Object", "NW1", "Class02", "Cl
                print "CreateConnection between NW2.Class11.C_Object and NW2.Class01.ClassSv
                raise batch.BatchError
        if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Command", "NW1", "Class02", "S
                print "CreateConnection between NW2.Class11.C_Command and NW2.Class01.S_Comm
                raise batch.BatchError
        if batch.CreateConnection(myPrj, "NW2", "Class11", "C_BDINT", "NW1", "Class02", "S_B
                print "CreateConnection between NW2.Class11.C_BDINT and NW2.Class01.S_BDINT
                raise batch.BatchError
        if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Real", "NW1", "Class02", "S_Re
                print "CreateConnection between NW2.Class11.C_Real and NW2.Class01.S_Real fa
                raise batch.BatchError

        # Test 3. Deleting valid connections between two networks with the DeleteConnection
```

```
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_DINT") == False :
                print "DeleteConnection of NW2.Class11.C_DINT failed."
                raise batch.BatchError
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_Object") == False :
                print "DeleteConnection of NW2.Class11.C_Object failed."
                raise batch.BatchError
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_Command") == False :
                print "DeleteConnection of NW2.Class11.C_Command failed."
                raise batch.BatchError
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_BDINT") == False :
                print "DeleteConnection of NW2.Class11.C_BDINT failed."
                raise batch.BatchError
        if batch.DeleteConnection(myPrj, "NW2", "Class11", "C_Real") == False :
                print "DeleteConnection of NW2.Class11.C_Real failed."
                raise batch.BatchError

        # End
        print "Finished"
        batch.CloseProject(myPrj)
```

3.12.3.41 DeleteConnection2

DeleteConnection2(prjDst, strObjectName, strClientName)

Deletes a connection between two channels.

Transfer parameters	Type	Description
prjDst	Project	The target project
strObjectName	String	The name of the object containing the client from which the connection starts
strClientName	String	The name of client identified as the connection start
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestDeleteConnection2() :
        print "Test DeleteConnection."

        # Initalizing
        batch.SetExceptionOnError(False)
```

```
batch.CloseAllProjects()

# Current project
strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestConnections"
myPrj = batch.LoadProject(strPrj)
if myPrj == None :
        print "Loading test project failed."
        raise batch.BatchError

# Test1: Deleting existing connections within a network with the DeleteConnection2 f
if batch.DeleteConnection2(myPrj, "Class11", "C_DINT") == False :
        print "DeleteConnection2 of Class11.C_DINT failed."
        raise batch.BatchError
if batch.DeleteConnection2(myPrj, "Class11", "C_Object") == False :
        print "DeleteConnection2 of Class11.C_Object failed."
        raise batch.BatchError
if batch.DeleteConnection2(myPrj, "Class11", "C_Command") == False :
        print "DeleteConnection2 of Class11.C_Command failed."
        raise batch.BatchError
if batch.DeleteConnection2(myPrj, "Class11", "C_BDINT") == False :
        print "DeleteConnection of Class11.C_BDINT failed."
        raise batch.BatchError
if batch.DeleteConnection2(myPrj, "Class11", "C_Real") == False :
        print "DeleteConnection of Class11.C_Real failed."
        raise batch.BatchError

# Create the connections between two networks for the next test
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_DINT", "NW1", "Class02", "S_DI
        print "CreateConnection between NW2.Class11.C_DINT and NW2.Class01.S_DINT fa
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Object", "NW1", "Class02", "Cl
        print "CreateConnection between NW2.Class11.C_Object and NW2.Class01.ClassSv
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Command", "NW1", "Class02", "S
        print "CreateConnection between NW2.Class11.C_Command and NW2.Class01.S_Comm
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_BDINT", "NW1", "Class02", "S_B
        print "CreateConnection between NW2.Class11.C_BDINT and NW2.Class01.S_BDINT
        raise batch.BatchError
if batch.CreateConnection(myPrj, "NW2", "Class11", "C_Real", "NW1", "Class02", "S_Re
        print "CreateConnection between NW2.Class11.C_Real and NW2.Class01.S_Real fa
        raise batch.BatchError

# Test 2: Deleting valid connections between two networks with the DeleteConnection2
if batch.DeleteConnection2(myPrj, "Class11", "C_DINT") == False :
        print "DeleteConnection of Class11.C_DINT failed."
        raise batch.BatchError
if batch.DeleteConnection2(myPrj, "Class11", "C_Object") == False :
        print "DeleteConnection2 of Class11.C_Object failed."
        raise batch.BatchError
if batch.DeleteConnection2(myPrj, "Class11", "C_Command") == False :
        print "DeleteConnection2 of Class11.C_Command failed."
        raise batch.BatchError
if batch.DeleteConnection2(myPrj, "Class11", "C_BDINT") == False :
        print "DeleteConnection2 of Class11.C_BDINT failed."
```

```
            raise batch.BatchError
    if batch.DeleteConnection2(myPrj, "Class11", "C_Real") == False :
            print "DeleteConnection2 of Class11.C_Real failed."
            raise batch.BatchError
    # Ende
    print "Finished"
    batch.CloseProject(myPrj)
```

3.12.3.42 DeleteFileOnPLC

DeleteFileOnPLC(prjDst, strConnection, strFile)

Deletes the desired file from the the PLC

Transfer parameters	Type	Description
prjDst	Project	Project from whose project properties the online connection is read (the connection does not have to be entered). If nothing is entered here, the connection specified under strConnection is applied
strConnection	String	Name for the online connection from the telephone book. If nothing in entered here, the project settings are applied (s. prjDst). Alternatively, the IP address of the PLC can be entered here directly in the following format: `batch.Download(myPrj, „TCPIP:xxx.xxx.xxx.xxx“)`
strFile	String	The path and name of the file to delete from the target system
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sys
import os.path

def TestConnectionFilesPlc() :

  strCurDir = os.path.dirname(sys.argv[0])
  strConnection = sys.argv[1]

  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  #strPrj = "\\ConnectionFilesPlc"
  #strPrjPath = (strDir + strPrj)
```

```
bSuccess = True

# Load project
#prjDst = batch.LoadProject(strPrjPath)
prjDst = classes.TheApplication().Project
if prjDst==None :
  bSuccess = False#raise batch.BatchError
  print "Loaded destination project %s" % prjDst.Filename
# Reset Project before setting connection file
if batch.Stop(prjDst, strConnection) == False :
  bSuccess = False
  print "Stop project failed"
# Compile project
if batch.Compile(prjDst) == False :
  bSuccess = False
  print "Compile project failed"
# Delete old connection file
if batch.DeleteFileOnPLC(None, strConnection, r"C:\lsldata\prjcfg.lcf") == False :
  bSuccess = False
  print "Delete connection file failed"

# Download project
if batch.Download(prjDst, strConnection, True) == False :
  bSuccess = False
  print "Download project failed"
# Set the project to start
if batch.Start(prjDst, strConnection) == False :
  bSuccess = False
  print "Start project failed"

if bSuccess == False :
  raise batch.BatchError

print "Test successfull"
```

3.12.3.43 DeleteNetwork

DeleteNetwork(prjDst, keyNetwork, bDeleteConnection=False, bDeleteIOElem=False)

Deletes a network from a project.

Transfer parameters	Type	Description
prjDst	Project	The project from which the network should be deleted
keyNetwork	String	The name of the network to be deleted
bDeleteConnection	Boolean	**True**: all connections shown in this network are also deleted. Default = **False**

bDeleteIOElem	Boolean	**True**: also deletes all IO elements from this network. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch

def GetTemp() :

  return os.environ["TEMP"]

def TestDeleteNetwork() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  strDst = os.path.join(strCurDir,"DeleteNetworkTest")
  # delete if already there
  shutil.rmtree(strDst, True)
  # Create
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    raise batch.BatchError
  print "Created empty destination project %s" % prjDst.Filename


  # Create network
  strNetwork = "NewNetwork"
  if batch.CreateNetwork(prjDst, strNetwork)==False :
    print "Create network %s failed." % strNetwork
    raise batch.BatchError


  # compile
  if batch.Compile(prjDst)==False :
    print "Build %s failed." % prjDst.Filename
    raise batch.BatchError

  # Comparison (make sure all this was created)
  strRef = os.path.join(strCurDir, "CreateNetworkRef")
  dc = filecmp.dircmp(strDst, strRef)
  diff = dc.diff_files
  if ( len(diff)>0 ) :
```

```
    print "Filecompare failed for files %s" % str(diff)
    raise batch.BatchError


# Delete network
 if batch.DeleteNetwork(prjDst, strNetwork)==False :
    print "Delete network %s failed." % strNetwork
    raise batch.BatchError

 # compile
 if batch.Compile(prjDst)==False :
    print "Build %s failed." % prjDst.Filename
    raise batch.BatchError

 # Comparison
 strRef = os.path.join(strCurDir, "DeleteNetworkRef")
 dc = filecmp.dircmp(strDst, strRef)
 diff = dc.diff_files
 if ( len(diff)>0 ) :
    print "Filecompare failed for files %s" % str(diff)
    raise batch.BatchError

 print "Test successfull"
```

3.12.3.44 DeleteObject

DeleteObject(prjDst, keyNetwork, keyObject, bDeleteConnection=True)

Deletes an object from a network.

Transfer parameters	Type	Description
prjDst	Project	The project, which contains the network
KeyNetwork	String	The name of the network from which the object should be deleted
KeyObject	String	The name of the object to be deleted
bDeleteConnection	Boolean	**True** = All connections pointing to this object are also deleted. Default = **True**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
```

```
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch

def GetTemp() :

  return os.environ["TEMP"]


def TestCreateObject() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  strDst = os.path.join(strCurDir,"DeleteObjectTest")
  # delete if already there
  shutil.rmtree(strDst, True)
  # Create
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    raise batch.BatchError
  print "Created empty destination project %s" % prjDst.Filename

  # Open the import-project
  strSrc = os.path.join(os.path.dirname(sys.argv[0]), r"..\..\Runtime\Library\OS Interface")
  prjSrc = batch.LoadProject(strSrc)
  if prjSrc==None :
    raise batch.BatchError
  print "Loaded source project %s" % prjSrc.Filename

  # Copy class
  strClass = "_Grafix"
  if batch.CopyClass(prjDst, prjSrc, strClass)==False :
    print "Copy class %s failed." % strClass
    raise batch.BatchError

  # Create network for filling
  strNetwork = "NewNetwork"
  if batch.CreateNetwork(prjDst, strNetwork)==False :
    print "Create network %s failed." % strNetwork
    raise batch.BatchError

  # Create object
  strClass  = "_Grafix"
  strObject = "NewObject"
  iPosX   = 200
  iPosY   = 300
  if batch.CreateObject(prjDst, strNetwork, strClass, strObject, iPosX, iPosY)==False :
    print "Create object %s failed." % strObject
    raise batch.BatchError

  # Close the Src-project again
  del prjSrc
```

```
# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError

# Comparison (make sure all this was created)
strRef = os.path.join(strCurDir, "CreateObjectRef")
dc = filecmp.dircmp(strDst, strRef)
diff = dc.diff_files
if ( len(diff)>0 ) :
  print "Filecompare failed for files %s" % str(diff)
  raise batch.BatchError

# Delete object
if batch.DeleteObject(prjDst, strNetwork, strObject)==False :
  print "Delete object %s failed." % strObject
  raise batch.BatchError

# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError

# Comparison (make sure all this was created)
strRef = os.path.join(strCurDir, "CreateNetworkRef")
dc = filecmp.dircmp(strDst, strRef)
diff = dc.diff_files
if ( len(diff)>0 ) :
  print "Filecompare failed for files %s" % str(diff)
  raise batch.BatchError

print "Test successfull"
```

3.12.3.45 DeletePrjConnectionFile

DeletePrjConnectionFile(prjDst, strFilename)

Deletes a connection file from the project.

Transfer parameters	Type	Description
prjDst	Project	The project, which contains the connection file
strFilename	String	Name of the connection file, which should be deleted from the project
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import filecmp
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def GetTemp() :

  return os.environ["TEMP"]

def TestConnectionFiles() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\ConnectionFiles"
  strPrjPath = (strDir + strPrj)

  bSuccess = True

  print "strPrjPath: \"%s\"" % strPrjPath

  # Load project
  prjDst = batch.LoadProject(strPrjPath)
  if prjDst==None :
    bSuccess = False#raise batch.BatchError
    print "Loaded destination project %s" % prjDst.Filename

  # Set the old connection file inactive
  strFilenameOld = "TestConFile1.lce"
  if batch.SetPrjConnectionFileActive(prjDst, strFilenameOld, False)==False :
    print "Change active flag or connection file %s failed." % strFilenameOld
    bSuccess = False

  # Add new connection file
  strFilenameNew = "TestConFile2.lce"
  strNetworkSubPath = "\\Network\\"
  strFilepathNew = strPrjPath + strNetworkSubPath + strFilenameNew
  if batch.AddPrjConnectionFile(prjDst, strFilepathNew)==False :
    print "Adding connection file %s failed." % strFilenameNew
    bSuccess = False#raise batch.BatchError

  # Delete old connection file
  if batch.DeletePrjConnectionFile(prjDst, strFilenameOld)==False :
    print "Delete connection file %s failed." % strFilenameOld
    raise batch.BatchError

  batch.CloseAllProjects();

  if bSuccess == False :
    raise batch.BatchError

  print "Test successfull"
```

3.12.3.46 DeleteVirtualCommissioning

DeleteVirtualCommissioning(prjDst, strObjectName)

Removes a Virtual Commissioning entry from the list.

Transfer parameters	Type	Description
prjDst	Project	The project in which the Virtual Commissioning entry to be removed is located
strObjectName	String	Name of the Virtual Commissioning object to be removed from the list
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as cls
import sigmatek.lasal.batch as batch

def DeleteVCObject() :
  prj = cls.TheApplication().Project
  prj.UseAdvancedIO = True
  batch.AddVirtualCommissioning(prj, "Class0", "NewObjectName")
  batch.DeleteVirtualCommissioning(prj, "NewObjectName")

#Main
if __name__ == "__main__" :
DeleteVCObject()
```

3.12.3.47 Download

Download(prjDst, strConnection="", bAddLoaderAnyway=False, bNoOverwriteMultiMaster=False)

Loads a project to the PLC.

Transfer parameters	Type	Description
prjDst	Project	The target project

strConnection	String	Name for the online connection from the Telefone book. If nothing is transferred, the project setting is valid. Alternatively, the IP address of the PIX can be entered directly in the following form: `batch.Download(myPrj, „TCPIP:xxx.xxx.xxx.xxx“)` Default = ""
bAddLoaderAnyway	Boolean	**True** = Loader is included in the transfer to the target system (even when the existing operating system supports the loader decoupling and a loader is already available in the target system. Default = **False**
bNoOverwriteMultiMaster	Boolean	**True** = The Multimaster configuration is not transferred to the target system. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Beispiel

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sys
import os.path

def TestConnectionFilesPlc() :

  strCurDir = os.path.dirname(sys.argv[0])
  strConnection = sys.argv[1]

  # Initializing paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  #strPrj = "\\ConnectionFilesPlc"
  #strPrjPath = (strDir + strPrj)

  bSuccess = True

  # Load project
  #prjDst = batch.LoadProject(strPrjPath)
  prjDst = classes.TheApplication().Project
  if prjDst==None :
    bSuccess = False#raise batch.BatchError
    print "Loaded destination project %s" % prjDst.Filename
  # Reset Project before setting connection file
  if batch.Stop(prjDst, strConnection) == False :
    bSuccess = False
    print "Stop project failed"
  # Compile project
  if batch.Compile(prjDst) == False :
    bSuccess = False
    print "Compile project failed"
```

```
# Delete old connection file
if batch.DeleteFileOnPLC(None, strConnection, r"C:\lsldata\prjcfg.lcf") == False :
  bSuccess = False
  print "Delete connection file failed"
# Download project to the PLC
if batch.Download(prjDst, strConnection, True) == False :
  bSuccess = False
  print "Download project failed"
# Set the project to start
if batch.Start(prjDst, strConnection) == False :
  bSuccess = False
  print "Start project failed"

if bSuccess == False :
  raise batch.BatchError

print "Test successfull"
```

3.12.3.48 DownloadConnectionFile

DownloadConnectionFile(prjDst, strConnection="", strConnection-File="")

Loads a ConnectionFile to the PLC.

Transfer parameters	Type	Description
prjDst	Project	The target project
strConnection	String	Name for the online connection from the telephone book. If nothing in entered here, the project settings are applied. Alternatively, the IP address of the PLC can be entered here directly in the following format: `batch.Download(myPrj, „TCPIP:xxx.xxx.xxx.xxx“)` Default = ""
strConnectionFile	String	The name of the ConnectionFile to download into the target system. If nothing is entered here, the project ConnectionFile is downloaded. Default = ""
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sys
```

```
import os.path

def TestConnectionFilesPlc() :

  strCurDir = os.path.dirname(sys.argv[0])
  strConnection = sys.argv[1]

  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  #strPrj = "\\ConnectionFilesPlc"
  #strPrjPath = (strDir + strPrj)

  bSuccess = True

  # Load project
  prjDst = classes.TheApplication().Project
  if prjDst==None :
    bSuccess = False#raise batch.BatchError
    print "Loaded destination project %s" % prjDst.Filename
  # Reset Project before setting connection file
  if batch.Stop(prjDst, strConnection) == False :
    bSuccess = False
    print "Stop project failed"
  # Download Project connection file (active)
  if batch.DownloadConnectionFile(prjDst, strConnection) == False :
    bSuccess = False
    print "Send prj-connection file failed"

    strConnFile = (strPrjPath + r"\Network\TestConFile2.lcf")
  # Download Project connection file (active)
  if batch.DownloadConnectionFile(None, strConnection, strConnFile) == False :
    bSuccess = False
    print "Send other-connection file failed"

  # Set the project to start
  if batch.Start(prjDst, strConnection) == False :
    bSuccess = False
    print "Start project failed"

  if bSuccess == False :
    raise batch.BatchError

  print "Test successfull"
```

3.12.3.49 DownloadFile

DownloadFile(prjDst, strConnection, strFileSrc, strFileDest)

Downloads the desired file to the PLC.

Transfer parameters	Type	Description
prjDst	Project	The target project
strConnection	String	Name for the online connection from the telephone book. If nothing in entered here, the project settings are applied. Alternatively, the IP address of the PLC can be entered here directly in the following format: `batch.Download(myPrj, „TCPIP:xxx.xxx.xxx.xxx“)`
strFileSrc	String	The path and name of the file to download into the target system
strFileDest	String	The path and name of the file in the target system
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sys
import os.path
import filecmp

def TestConnectionFilesPlc() :

  strCurDir = os.path.dirname(sys.argv[0])
  strConnection = sys.argv[1]

  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strFileD = "\\DownloadFile.txt"
  strFilePathD = (strDir + strFileD)
  strFileU = "\\UploadFile.txt"
  strFilePathU = (strDir + strFileU)
  strFilePlc = "C:\\unittest.txt"

  bSuccess = True
  # Delete existing file
  if os.path.exists(strFilePathU) != False :
    os.remove(strFilePathU)

  # Download file
  if batch.DownloadFile(None, strConnection, strFilePathD, strFilePlc) == False :
    bSuccess = False
    print "Download file failed"

  # Upload file
  if batch.UploadFile(None, strConnection, strFilePlc, strFilePathU) == False :
    bSuccess = False
```

```
    print "Upload file failed"

  # Compare files
  if filecmp.cmp(strFilePathD, strFilePathU) == False :
    bSuccess = False
    print "file compare failed"

  if bSuccess == False :
    raise batch.BatchError

  print "Test successfull"
```

3.12.3.50 DownloadOS

DownloadOS(strConnection, strUpdateFile)

The command goes online with the control specified in **strConnection** and downloads the OS defined in **strUpdateFile**. Afterwards, it goes offline.

Transfer parameters	Type	Description
strConnection	String	Name of the control (entry in telephone book)
strUpdateFile	String	Name of the OS with path
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

# Use loaded GUI-Project
prjTest = classes.Project()

strIp = "10.10.116.184"

batch.ClearRamImage(strIp)

batch.Download(prjTest, strIp, True)
batch.Download(prjTest, strIp, False)

batch.Start(prjTest, strIp)

batch.Stop(prjTest, strIp)

batch.SaveProjectOnPLC(prjTest, strIp)
```

```
batch.CloseModemConnection()

batch.DownloadOS(strIp, "DummyOS")

del prjTest
```

3.12.3.51 DuplicateNetwork

DuplicateNetwork(prjDst, strNetwork, strNewName)

Duplicates an object network.

Transfer parameters	Type	Description
prjDst	Project	The target project, which contains the network
strNetwork	String	Name of the network to duplicate
strNewName	String	Name of the new network
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestDuplicateNetAndReplaceObj() :

        print "TestDuplicateNetAndReplaceObj:"

        # Init paths
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestChangeClass"
        strPrjPath = (strDir + strPrj)

        # Load test project
        prj = batch.LoadProject (strPrjPath)

        if prj == None :
                print "Loading project \"%s\" failed." % (strPrjPath)
                raise batch.BatchError

        strClass1  = "Class0"
        strClass2  = "Class1"
        strObject = "NewObject"
```

```
strRenamedObject = "RenamedObject"
strNetwork = "NewNetwork"
strRenamedNetwork = "RenamedNetwork"
strDuplicatedNetwork = "DuplicatedNetwork"

if batch.CreateNetwork(prj, strNetwork)==False :
        print "Create network %s failed." % strNetwork
        raise batch.BatchError

# Create object
iPosX   = 200
iPosY   = 300
if batch.CreateObject(prj, strNetwork, strClass1, strObject, iPosX, iPosY)==False :
        print "Create object %s failed." % strObject
        raise batch.BatchError

# Change the class
if batch.ChangeClass(prj, strNetwork, strObject, strClass2)==False :
        print "Change class from %s to %s failed." % strClass1 % strClass2
        raise batch.BatchError

# compile
if batch.Compile(prj)==False :
        print "Build %s failed." % prj.Filename
        raise batch.BatchError

# Rename network
if batch.RenameNetwork(prj, strNetwork, strRenamedNetwork)==False :
        print "Rename network %s to %s failed." % strNetwork % strRenamedNetwork
        raise batch.BatchError

# Duplicate Network
if batch.DuplicateNetwork(prj, strRenamedNetwork, strDuplicatedNetwork)==False :
        print "Duplicate network %s failed." % strNetwork
        raise batch.BatchError

# Rename object
if batch.RenameObject(prj, strRenamedNetwork, strObject, strRenamedObject)==False :
        print "Rename object %s to %s failed." % strObject % strRenamedObject
        raise batch.BatchError

# Delete network
if batch.DeleteNetwork(prj, strRenamedNetwork)==False :
        print "Delete network %s failed." % strRenamedNetwork
        raise batch.BatchError

# Delete network
if batch.DeleteNetwork(prj, strDuplicatedNetwork)==False :
        print "Delete network %s failed." % strDuplicatedNetwork
        raise batch.BatchError

# compile
if batch.Compile(prj)==False :
        print "Build %s failed." % prj.Filename
        raise batch.BatchError
```

```
batch.CloseAllProjects()

print "TestDuplicateNetAndReplaceObj successful"
```

3.12.3.52 EncryptConnectionFile

EncryptConnectionFile(strConnFile, strPassword)

Encrypts a connection file.

Transfer parameters	Type	Description
strConnFile	String	Path and name of the file to encrypt
strPassword	String	Password for encrypting
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestConnFileEncrypter() :

  # Init paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting\\TestConnFileEncrypter"
  strConnFileName = strDir + "\\TestConnFileEncrypter.lcf"

  bSuccess = True

  # do Encrypt
  if batch.EncryptConnectionFile(strConnFileName, "TestPassword") == False :
    bSuccess = False
    print "Encrypt ConnFile failed"

  if bSuccess == False :
    raise batch.BatchError
```

3.12.3.53 ExistsClass

ExistsClass(prjDst, keyClass)

Returns **True** at existence of the defined class.

Transfer parameters	Type	Description
prjDst	Project	The project in which it is searched
keyClass	String	The class name to be searched for
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: class doesn't exist or transferred invalid project

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\Li
if bat.ExistsClass(prjSrc,"C1") == False :
        raise Exception
if bat.ExistsClass(prjSrc,"NotExisting") == True :
        raise Exception
if bat.ExistsClass(prjSrc,"") == True :
        raise Exception
bat.CloseProject (prjSrc)
```

3.12.3.54 ExistsConnection

ExistsConnection(prjDst, strSourceNetworkName, strSourceObjectName, strSourceClientName, strTargetNetworkName, strTargetObjectName, strTargetServerName)

Returns **True** if the specified connection exists. This function works exactly the same as ExistConnection2(), with the difference that the network names are also specified here.

Transfer parameters	Type	Description
prjDst	Project	The target project, which includes the network

strSourceNetworkName	String	Name of the network where the connection starts
strSourceObjectName	String	Name of the object where the connection starts
strSourceClientName	String	Name of the client where the connection starts
strTargetNetworkName	String	Name of the network where the connection ends
strTargetObjectName	String	Name of the object where the connection ends
strTargetServerName	String	Name of the client where the connection ends
Return parameters	**Type**	**Description**
	Boolean	**True**: The connection with the specified parameters exists **False:** The connection with the specified parameters does not exist

Example

```
import sys
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestConn_NonePrj() :
        bSuccess = batch.ExistsConnection(None, "netS", "objS", "chnS", "netT", "objT", "chn
        exit(1)

def TestConn_NoParams() :
        if batch.ExistsConnection(prj, "", "", "", "", "", "") == True :
                exit(1)

def TestConn_Standard() :
        #Any Client Server connection -> Exists
        if batch.ExistsConnection(prj, "", "SourceObj1", "Client0", "", "TargetObj1", "Serve
                exit(1)
        #Unnamed ClassServer -> Exists
        if batch.ExistsConnection(prj, "", "SourceObj1", "Client", "", "TargetObj1", "") ==
                exit(2)
        #Network external -> Exists
        if batch.ExistsConnection(prj, "", "SourceObj1", "Client1", "", "ExtObj", "Server")
                exit(3)

def TestConn_NoServer() :
        #No server specified -> client has connection -> Exists
        if batch.ExistsConnection(prj, "", "SourceObj1", "Client0", "", "", "") == False :
                exit(1)
        if batch.ExistsConnection(prj, "", "SourceObj1", "Client3", "", "", "") == True :
                exit(2)

def TestConn_NoClient() :
        #Server specified -> No connection
        if batch.ExistsConnection(prj, "", "", "", "", "TargetObj2", "Server0") == True :
```

```
                exit(1)
        #No server specified -> No connection
        if batch.ExistsConnection(prj, "", "", "", "", "TargetObj2", "") == True :
                exit(2)
        #Server specified -> with connection
        if batch.ExistsConnection(prj, "", "", "", "", "TargetObj3", "Server0") == False :
                exit(3)
        #No server specified -> with connection
        if batch.ExistsConnection(prj, "", "", "", "", "TargetObj3", "") == False :
                exit(4)
        #Server specified -> With external connection
        if batch.ExistsConnection(prj, "", "", "", "", "TargetObj4", "Server0") == False :
                exit(5)
        #No server specified -> With external connection
        if batch.ExistsConnection(prj, "", "", "", "", "TargetObj4", "") == False :
                exit(6)

def TestConn_MMConn() :
        #Connection between server and MMObj (Obj and server disconnected) -> With connection
        if batch.ExistsConnection(prj, "", "SourceMMObj1", "Client", "", "ConnectionPrj.MMTarg
                exit(1)
        #Connection between server and MMObj -> With connection
        if batch.ExistsConnection(prj, "", "SourceMMObj1", "Client", "", "ConnectionPrj.MMTarg
                exit(2)
        #Connection to MMObj -> With connection
        if batch.ExistsConnection(prj, "", "", "", "", "ConnectionPrj.MMTargetObj1.ClassSvr",
                exit(3)
        #Connection to MMObj -> No connection
        if batch.ExistsConnection(prj, "", "", "", "", "ConnectionPrj.MMTargetObj1.Server0", "
                exit(4)
        #Connection no MMObj specified -> With connection
        if batch.ExistsConnection(prj, "", "SourceMMObj1", "Client", "", "", "") == False :
                exit(5)

def TestConn_MMConnExt() :
        #Like TestConn_MMConn only across networks:
        if batch.ExistsConnection(prj, "", "SourceObj4", "Client", "", "ConnectionPrj.MMTarget
                exit(1)
        if batch.ExistsConnection(prj, "", "SourceObj4", "Client", "", "ConnectionPrj.MMTarget
                exit(2)
        if batch.ExistsConnection(prj, "", "", "", "", "ConnectionPrj.MMTargetObj2.ClassSvr",
                exit(3)
        if batch.ExistsConnection(prj, "", "", "", "", "ConnectionPrj.MMTargetObj2.Server0", "
                exit(4)
        if batch.ExistsConnection(prj, "", "SourceObj4", "Client1", "", "", "") == False :
                exit(5)

def TestConn_XXX() :
        if batch.ExistsConnection(prj, "", "", "", "", "", "") == True :
                exit(1)

if __name__=="__main__" :
        prj = classes.TheApplication().Project
        if prj == None or len(sys.argv) <= 1 :
                print ("No Parameters or no project")
```

```
            raise Exception

print ("Do " + sys.argv[1])
if sys.argv[1] == 'None' :
            TestConn_NonePrj()
elif sys.argv[1] == 'NoParams' :
            TestConn_NoParams()
elif sys.argv[1] == 'Standard' :
            TestConn_Standard()
elif sys.argv[1] == 'NoServer' :
            TestConn_NoServer()
elif sys.argv[1] == 'NoClient' :
            TestConn_NoClient()
elif sys.argv[1] == 'MMConn' :
            TestConn_MMConn()
elif sys.argv[1] == 'MMConnExt' :
            TestConn_MMConnExt()

prj = None
print ("Finished " + sys.argv[1])
```

3.12.3.55 ExistsConnection2

ExistsConnection2(prjDst, strSourceObjectName, strSourceClientName, strTargetObjectName, strTargetServerName)

Returns **True** if the specified connection exists. This function works exactly the same as ExistsConnection(), with the difference that **no network names** are specified.

Transfer parameters	Type	Description
prjDst	Project	The target project, which includes the network
strSourceObjectName	String	Name of the object where the connection starts
strSourceClientName	String	Name of the client where the connection starts
strTargetObjectName	String	Name of the object where the connection ends
strTargetServerName	String	Name of the client where the connection ends
Return parameters	**Type**	**Description**
	Boolean	**True**: The connection with the specified parameters exists **False:** The connection with the specified parameters does not exist

Example

```
import sys
import sigmatek.lasal.batch as batch
```

```
import sigmatek.lasal.classes as classes

def TestConn_NonePrj() :
        bSuccess = batch.ExistsConnection2(None, "objS", "chnS", "objT", "chnT")
        exit(1)

def TestConn_NoParams() :
        if batch.ExistsConnection2(prj, "", "", "", "") == True :
                exit(1)

def TestConn_Standard() :
        #Any Client Server connection -> Exists
        if batch.ExistsConnection2(prj, "SourceObj1", "Client0", "TargetObj1", "Server0") == F
                exit(1)
        #Unnamed ClassServer -> Exists
        if batch.ExistsConnection2(prj, "SourceObj1", "Client", "TargetObj1", "") == False :
                exit(2)
        #Network external -> Exists
        if batch.ExistsConnection2(prj, "SourceObj1", "Client1", "ExtObj", "Server") == True :
                exit(3)

def TestConn_NoServer() :
        #No server specified -> client has connection -> Exists
        if batch.ExistsConnection2(prj, "SourceObj1", "Client0", "", "") == False :
                exit(1)
        if batch.ExistsConnection2(prj, "SourceObj1", "Client3", "", "") == True :
                exit(2)

def TestConn_NoClient() :
        #Server specified -> No connection
        if batch.ExistsConnection2(prj, "", "", "TargetObj2", "Server0") == True :
                exit(1)
        #No server specified -> No connection
        if batch.ExistsConnection2(prj, "", "", "TargetObj2", "") == True :
                exit(2)
        #Server specified -> with connection
        if batch.ExistsConnection2(prj, "", "", "TargetObj3", "Server0") == False :
                exit(3)
        #No server specified -> with connection
        if batch.ExistsConnection2(prj, "", "", "TargetObj3", "") == False :
                exit(4)
        #Server specified -> With external connection
        if batch.ExistsConnection2(prj, "", "", "TargetObj4", "Server0") == False :
                exit(5)
        #No server specified -> With external connection
        if batch.ExistsConnection2(prj, "", "", "TargetObj4", "") == False :
                exit(6)

def TestConn_MMConn() :
        #Connection between server and MMObj (Obj and server disconnected) -> With connection
        if batch.ExistsConnection2(prj, "SourceMMObj1", "Client", "ConnectionPrj.MMTargetObj1"
                exit(1)
        #Connection between server and MMObj -> With connection
        if batch.ExistsConnection2(prj, "SourceMMObj1", "Client", "ConnectionPrj.MMTargetObj1.
                exit(2)
```

```
        #Connection to MMObj -> With connection
        if batch.ExistsConnection2(prj, "", "", "ConnectionPrj.MMTargetObj1.ClassSvr", "") =
                exit(3)
        #Connection to MMObj -> No connection
        if batch.ExistsConnection2(prj, "", "", "ConnectionPrj.MMTargetObj1.Server0", "") ==
                exit(4)
        #Connection no MMObj specified -> With connection
        if batch.ExistsConnection2(prj, "SourceMMObj1", "Client", "", "") == False :
                exit(5)

def TestConn_MMConnExt() :
        #Like TestConn_MMConn only across networks:
        if batch.ExistsConnection2(prj, "SourceObj4", "Client", "ConnectionPrj.MMTargetObj2"
                exit(1)
        if batch.ExistsConnection2(prj, "SourceObj4", "Client", "ConnectionPrj.MMTargetObj2.
                exit(2)
        if batch.ExistsConnection2(prj, "", "", "ConnectionPrj.MMTargetObj2.ClassSvr", "") =
                exit(3)
        if batch.ExistsConnection2(prj, "", "", "ConnectionPrj.MMTargetObj2.Server0", "") ==
                exit(4)
        if batch.ExistsConnection2(prj, "SourceObj4", "Client1", "", "") == False :
                exit(5)

def TestConn_XXX() :
        if batch.ExistsConnection2(prj, "", "", "", "") == True :
                exit(1)

if __name__=="__main__" :
        prj = classes.TheApplication().Project
        if prj == None or len(sys.argv) <= 1 :
                print ("No Parameters or no project")
                raise Exception

        print ("Do " + sys.argv[1])
        if sys.argv[1] == 'None' :
                TestConn_NonePrj()
        elif sys.argv[1] == 'NoParams' :
                TestConn_NoParams()
        elif sys.argv[1] == 'Standard' :
                TestConn_Standard()
        elif sys.argv[1] == 'NoServer' :
                TestConn_NoServer()
        elif sys.argv[1] == 'NoClient' :
                TestConn_NoClient()
        elif sys.argv[1] == 'MMConn' :
                TestConn_MMConn()
        elif sys.argv[1] == 'MMConnExt' :
                TestConn_MMConnExt()

        prj = None
        print ("Finished " + sys.argv[1])
```

3.12.3.56 ExistsIOLabel

ExistsConnection(prjDst, strIOName)

Returns **True** if the IO object with the specified name exists.

Transfer parameters	Type	Description
prjDst	Project	The target project, which contains the IO object
strIOLabel	String	Name of the searched IO object
Return parameters	**Type**	**Description**
	Boolean	**True**: The IO object with the specified name exists **False:** The IO object with the specified name does not exist

Example

```
import sys
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestIOLabel_NonePrj() :
        bSuccess = batch.ExistsIOLabel(None, "bla")
        exit(1)

def TestIOLabel_NoName() :
        if batch.ExistsIOLabel(prj, "") == True :
                exit(1)

def TestIOLabel_Standard() :
        #Label not in ConnectionManager -> No entry
        if batch.ExistsIOLabel(prj, "NotExisting") == True :
                exit(1)
        #Label in ConnectionManager -> Entry available
        if batch.ExistsIOLabel(prj, "IOElem_07_09_1") == False :
                exit(2)
        if batch.ExistsIOLabel(prj, "IOElem_07_09_2") == True :
                exit(3)

if __name__=="__main__" :
        prj = classes.TheApplication().Project
        if prj == None or len(sys.argv) <= 1 :
                print ("No Parameters or no project")
                raise Exception

        print ("Do " + sys.argv[1])
        if sys.argv[1] == 'None' :
                TestIOLabel_NonePrj()
        elif sys.argv[1] == 'NoName' :
                TestIOLabel_NoName()
        elif sys.argv[1] == 'Standard' :
```

```
            TestIOLabel_Standard()
prj = None
print ("Finished " + sys.argv[1])
```

3.12.3.57 ExistsNetwork

ExistsNetwork(prjDst, keyNetwork)

Returns **True** at existence of the defined network.

Transfer parameters	Type	Description
prjDst	Project	The project in which it is searched
keyNetwork	String	The network name to be searched for
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: network doesn't exist or transferred invalid project

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\
if bat.ExistsNetwork (prjSrc, "NW1") == False :
        raise Exception
if bat.ExistsNetwork (prjSrc, "NotExisting") == True :
        raise Exception
if bat.ExistsNetwork (prjSrc, "") == True :
        raise Exception
bat.CloseProject (prjSrc)
```

3.12.3.58 ExistsObject

ExistsObject(prjDst, strNetworkName, strObjectName)

Returns **True** at existence of the defined object.

Transfer parameters	Type	Description
prjDst	Project	The project in which it is searched

strNetworkName	String	The network in which it is searched. Here also a null string can be transferred
strObjectName	String	The object name to be searched for
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: object doesn't exist or transferred invalid project

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\Li

if bat.ExistsObject (prjSrc, "", "O1") == False :
        raise Exception
if bat.ExistsObject (prjSrc, "NW1", "O1") == False :
        raise Exception

if bat.ExistsObject (prjSrc, "", "OComplex\C11") == False :
        raise Exception
if bat.ExistsObject (prjSrc, "NWComplex", "OComplex\C11") == False :
        raise Exception

if bat.ExistsObject (prjSrc, "NWComplex", "OComplex\C12") == True :
        raise Exception
if bat.ExistsObject (prjSrc, "", "OComplex\C12") == True :
        raise Exception
if bat.ExistsObject (prjSrc, "NW1", "OComplex\C12") == True :
        raise Exception

if bat.ExistsObject (prjSrc, "NW2", "O1") == True :
        raise Exception
if bat.ExistsObject (prjSrc, "NotExisting", "O1") == True :
        raise Exception
if bat.ExistsObject (prjSrc, "NW1", "NotExisting") == True :
        raise Exception
if bat.ExistsObject (prjSrc, "", "") == True :
        raise Exception

bat.CloseProject (prjSrc)
```

3.12.3.59 GetClassObjectNames

GetClassObjectNames(prjDst, keyClass, bIncludeBaseObjects=False)

Delivers a list of all object names that have been instanced. If the transferred class can not be found, the function returns none. If no instanced object exists, the function then returns an empty list.

Transfer parameters	Type	Description
PrjDst	Project	The project in which it is searched
keyClass	String	The class name for which it is searched
BIncludeBaseObjects	Boolean	**True** = delivers also the names of the base class objects. Default=**False**
Return parameters	**Type**	**Description**
	List	**None**: class could not be found **Empty string list [ ]**: object could not be found **String list with object names**: list of the searched object names [„asb“, „sdsf“]

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def GetTemp() :

  return os.environ["TEMP"]

def TestGetClassObjectNames() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\GetClassObjectNames"
  strPrjPath = (strDir + strPrj)

  bSuccess = True

  print "strPrjPath: \"%s\"" % strPrjPath
```

```
# Load the project
prjDst = batch.LoadProject(strPrjPath)
if prjDst==None :
  bSuccess = False#raise batch.BatchError
  print "Loaded destination project %s" % prjDst.Filename

# Determine objects for class without base objects
strCls = "TestCls"
arrObjNamesWOBase = batch.GetClassObjectNames(prjDst, strCls, False)
if len(arrObjNamesWOBase)!=4 :
  print "Wrong object count without base objects for class %s." % strCls
  bSuccess = False

# Determine objects for class with base objects
arrObjNamesWBase = batch.GetClassObjectNames(prjDst, strCls, True)
if len(arrObjNamesWBase)!=6 :
  print "Wrong object count including base objects for class %s." % strCls
  bSuccess = False

batch.CloseAllProjects();

if bSuccess == False :
  raise batch.BatchError

print "Test successfull"
```

3.12.3.60 GetCompilerVersion

GetCompilerVersion(prjDst)

Returns the compiler version of the specified project.

Transfer parameters	Type	Description
prjDst	Project	The target project
Return parameters	**Type**	**Description**
	Number	Compiler version of the specified project

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

batch.CloseAllProjects()

strPath = classes.TheApplication().Path + r"..\Data\Scripting\TestCompilerVersion"

prjTest = batch.LoadProject(strPath)
```

```
if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-----------------------------------------------------------------------------------------------
#GetCurrentCompilerVersion & GetCompilerVersion    & SetCompilerVersion

version = batch.GetCurrentCompilerVersion()
if(version <> 27) :
        raise Exception

if(prjTest.CompilerOptions.CompilerVersion <> batch.GetCompilerVersion(prjTest))     :
        raise Exception

versionBefore = prjTest.CompilerOptions.CompilerVersion
batch.SetCompilerVersion(prjTest, 10)

if(prjTest.CompilerOptions.CompilerVersion <> batch.GetCompilerVersion(prjTest))     :
        raise Exception

if(versionBefore == prjTest.CompilerOptions.CompilerVersion) :
        raise Exception

if(10 <> prjTest.CompilerOptions.CompilerVersion) :
        raise Exception

# current version must not have been changed
if(version <> batch.GetCurrentCompilerVersion()) :
        raise Exception

batch.SetCompilerVersion(prjTest, batch.GetCurrentCompilerVersion())
if(version <> batch.GetCompilerVersion(prjTest)) :
        raise Exception

# End:
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.61 GetCreateBootdiskContainer

GetCreateBootdiskContainer()

Returns a new container with all parameters and their default values for the CreateBootdiskMngr() function. The following possible parameters are available:

Parameters	Allowed values	Notes
BootDiskType	"None", "All", "Menu", "CreateArchive"	Equivalent to the update script group in the CreateBootdisk dialog

Platform	"AUTO", "ALL", "ARM", "X86"	Equivalent to the platform group in the CreateBootdisk dialog
UseAdditionalFiles	"true", "false"	If **true**, the parameter "AdditionalFiles" is taken into account, otherwise ignored
AdditionalFiles	A list of tuples (strings)	The first element is the source file, the second the destination folder. Represents the "Additional files" box in the CreateBootdisk dialog.
Return parameters	**Type**	**Notes**
	Container	The new container for the CreateBootdiskManager() function

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

def GetParameters():

        dict = batch.GetCreateBootdiskContainer()
      print dict
      dict["Platform"] = "X86"
      print dict

def CallCreateBootDisk():

        dict = batch.GetCreateBootdiskContainer()
      dict["Platform"] = "X86"
      dict["AdditionalFiles"] = [(r"D:\py\test.xml",r"C:\temp"), (r"D:\temp\lalala.txt",r"C:\t
      dict["UseAdditionalFiles"] = "true"
      print dict
      batch.CreateBootdiskViaContainer(None, r"D:/todelete/BDPC/", dict)
#Main
        if __name__ == "__main__" :
      reload(batch)
      GetParameters()
      CallCreateBootDisk()
```

3.12.3.62 GetCurrentCompilerVersion

GetCurrentCompilerVersion()

Returns the newest compiler version available.

Return parameters	Type	Description
	Number	Returns the current compiler version

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

batch.CloseAllProjects()

strPath = classes.TheApplication().Path + r"..\Data\Scripting\TestCompilerVersion"

prjTest = batch.LoadProject(strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------------
#GetCurrentCompilerVersion & GetCompilerVersion    & SetCompilerVersion

version = batch.GetCurrentCompilerVersion()
if(version <> 27) :
        raise Exception

if(prjTest.CompilerOptions.CompilerVersion <> batch.GetCompilerVersion(prjTest))     :
        raise Exception

versionBefore = prjTest.CompilerOptions.CompilerVersion
batch.SetCompilerVersion(prjTest, 10)

if(prjTest.CompilerOptions.CompilerVersion <> batch.GetCompilerVersion(prjTest))     :
        raise Exception

if(versionBefore == prjTest.CompilerOptions.CompilerVersion) :
        raise Exception

if(10 <> prjTest.CompilerOptions.CompilerVersion) :
        raise Exception

# current version must not have been changed
if(version <> batch.GetCurrentCompilerVersion()) :
        raise Exception

batch.SetCompilerVersion(prjTest, batch.GetCurrentCompilerVersion())
if(version <> batch.GetCompilerVersion(prjTest)) :
        raise Exception

# End:
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.63 GetVirtualCommissionProperties

GetVirtualCommissionProperties(prjDst, strObjectName)

Returns all available properties of a Virtual Commmissioning object, wrapped in a map.

Transfer parameters	Type	Description
prjDst	Project	The project in which the Virtual Commissioning object is located
strObjectName	String	Name of the Virtual Commissioning object from which the properties are to be supplied
Return parameters	**Type**	**Description**
	Map	**Map =** contains the available properties of the Virtual Commissioning object **None** = failed

Example

```
import sigmatek.lasal.classes as cls
import sigmatek.lasal.batch as batch

def Properties() :
  prj = cls.TheApplication().Project
  props = batch.GetVirtualCommissionProperties(prj, "VirtTwin");
  print props
  props["PortNumber"] = "125"
  props["IPAddress"] = "10.10.116.39"
  print props
  batch.SetVirtualCommissionProperties(prj, "VirtTwin", props);

#Main
if __name__ == "__main__" :
Properties()
```

3.12.3.64 ImportAmlHwConfig

ImportAmlHwConfig(prjDst, strFilename, strStation="", bForce=False, bCreateIO=True, strObjectName="", strBusType="", iPlace=0)

Imports an AML configuration file and replaces the hardware tree of the given project. If file contains more than one CPU use *strStation* to select one. If *strStation* is empty, the first CPU will be taken. *strStation* also take CPU GUIDs from the file. To import additional

hardware to an existing hardware tree the hardware object, the bus type and a correct place/port has to be set.

Übergabeparameter	Typ	Beschreibung
prjDst	Project	The target project
strFilename	String	File which should be loaded
strStation	String	Station which should be imported (Default="")
bForce	Boolean	Force import and ignore unknown device identifiers (Default=**False**)
bCreateIO	Boolean	Create IO elements if missing (Default=**True**)
strObjectName	String	(Needed for partial import) Name of the object to where partial import is inserted, if empty PLC object is taken (Default="")
strBusType	String	(Needed for partial import) Name of the bus type where hardware of the partial import gets placed, e.g. VARAN, SDIAS (Default="")
iPlace	Number	(Needed for partial import) Place or port identifier depending of the bus type (Default=0)
Rückgabeparameter	**Typ**	**Beschreibung**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sys
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

# Tests AML_ImportHWConfig
# Init paths
strDir = classes.TheApplication().Path + "..\\Data\\Scripting"

# Load test project
prj = classes.TheApplication().Project

# Arguments
# print sys.argv[1]
strFilePath = sys.argv[1]
strStation = sys.argv[2]
bIsForces = False
if sys.argv[3] != "False" :
        bIsForces = True

if batch.ImportAmlHwConfig(prj, strFilePath, strStation, bIsForces) == False :
        raise Exception

# Tests AML_PartImportHWConfig
```

```
# Init paths
strDir = classes.TheApplication().Path + "..\\Data\\Scripting"

# Load test project
prj = classes.TheApplication().Project

# Arguments
# print sys.argv[1]
strFilePath = sys.argv[1]
strStation = sys.argv[2]
bIsForces = False
bIsCreateIO = False
strObject = sys.argv[5]
strBus = sys.argv[6]
iPlace = int(sys.argv[7])

if sys.argv[3] != "False" :
        bIsForces = True
if sys.argv[4] != "False" :
        bIsCreateIO = True

if batch.ImportAmlHwConfig(prj, strFilePath, strStation, bIsForces, bIsCreateIO, strObject, st
        raise Exception
```

3.12.3.65 ImportFromFile

ImportFromFile(prjDst, strFilename, bOverwrite=True, bCheckRevision=True, bKeepExistingClassesLinked=False)

Imported classes and networks from an LCZ file in a project. Classes, which were **explicitly selected during export**, are **always** imported and classes with the same name overwritten! For all other classes, the rules defined by the three Boolean parameters apply.

Transfer parameters	Type	Description
prjDst	Project	Target project
strFilename	String	Name of the LCZ, whose content to import
bOverwrite	Boolean	**True** (overwrites an existing class with the same name in the target project) or **False** (skips the class). Default is **True**.
bCheckRevision	Boolean	**True** (checks the revision number of the class to import. An existing class with the same name is overwritten, only when the revision number of the class to import is higher) **False** (the revision number is not checked). Default is True. This flag is only considered when **bOverwrite = True**!

bKeepExistingClassesLinked	Boolean	**True** (allows linked classes with the name as the class to import to exist), **False** (also overwrites linked classes). Default is **False**. This flag is only considered when **bOverwrite = True!**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path as path
import sys
import shutil
import sigmatek.lasal.batch as batch

def TestImportNetFromFile() :
  # Init paths
  strFile = "\\Scripting\\ImportNetworksFromFile.lcz"
  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  strDst = os.path.join(strCurDir,"ImportNetTest")
  # Delete if already exists
  shutil.rmtree(strDst, True)
  # create
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    return False
  print "Created empty destination project %s" % prjDst.Filename

  if batch.ExistsNetwork(prjDst, "Network0") <> False :
    return False
  if batch.ExistsNetwork(prjDst, "Network1") <> False :
    return False
  if batch.ExistsNetwork(prjDst, "Network2") <> False :
    return False
  if batch.ExistsClass(prjDst, "Class4Obj_1") <> False :
    return False
  if batch.ExistsClass(prjDst, "Class4Obj_1_derived") <> False :
    return False
  if batch.ExistsClass(prjDst, "Class4Obj_2") <> False :
    return False

  strFilePath = path.dirname(strCurDir) + strFile
  if batch.ImportFromFile(prjDst, strFilePath) == False :
    return False

  if batch.ExistsNetwork(prjDst, "Network0") == False :
    return False
  if batch.ExistsNetwork(prjDst, "Network1") == False :
    return False
```

```
  if batch.ExistsNetwork(prjDst, "Network2") <> False :
    return False
  if batch.ExistsClass(prjDst, "Class4Obj_1") == False :
    return False
  if batch.ExistsClass(prjDst, "Class4Obj_1_derived") == False :
    return False
  if batch.ExistsClass(prjDst, "Class4Obj_2") <> False :
    return False

  # Close the src-project again
  batch.CloseProject(prjDst)
  del prjDst
  print "Test successfull"

def TestImportClsFromFile() :
  # Inits paths
  strFile = "\\Scripting\\ImportClassesFromFile.lcz"
  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  strDst = os.path.join(strCurDir,"ImportClsTest")
  # delete if already exists
  shutil.rmtree(strDst, True)
  # cerate
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    return False
  print "Created empty destination project %s" % prjDst.Filename

  if batch.ExistsNetwork(prjDst, "Network0") <> False :
    return False
  if batch.ExistsNetwork(prjDst, "Network1") <> False :
    return False
  if batch.ExistsNetwork(prjDst, "Network2") <> False :
    return False
  if batch.ExistsClass(prjDst, "ImportCls_1") <> False :
    return False
  if batch.ExistsClass(prjDst, "ImportCls_1_derived") <> False :
    return False
  if batch.ExistsClass(prjDst, "ImportCls_2") <> False :
    return False

  strFilePath = path.dirname(strCurDir) + strFile
  if batch.ImportFromFile(prjDst, strFilePath) == False :
    return False

  if batch.ExistsNetwork(prjDst, "Network0") <> False :
    return False
  if batch.ExistsNetwork(prjDst, "Network1") <> False :
    return False
  if batch.ExistsNetwork(prjDst, "Network2") <> False :
    return False
  if batch.ExistsClass(prjDst, "ImportCls_1") == False :
    return False
  if batch.ExistsClass(prjDst, "ImportCls_1_derived") == False :
```

```
    return False
  if batch.ExistsClass(prjDst, "ImportCls_2") == False :
    return False

  # Close the src-project again
  batch.CloseProject(prjDst)
  del prjDst
  print "Test successfull"

  return True
```

3.12.3.66 IsExceptionOnError

IsExceptionOnError()

Queries whether an exception is generated when an error occurs. Return is **True** or **False**. This value is set with SetExceptionOnError().

Return parameters	Type	Description
	Boolean	**True** (Exceptions are activated) **False** (Exceptions are deactivated)

Example

```
import sigmatek.lasal.batch as batch

# Setting the value to True = an exception is thrown every time an error occurs
batch.SetExceptionOnError(True)

# Query the previously set value
bErr = batch.IsExceptionOnError()
```

3.12.3.67 LinkClass

LinkClass(prjDst, prjSrc, keyClass, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False)

Links a class to a project.

Transfer parameters	Type	Description
prjDst	Project	The target project to which the class should be added
prjSrc	Project	The source project from which the class originates

keyClass	String	The name of the class that should be copied
bOverwrite	Boolean	**True** = a class already existing as copy of the same name is replaced by the defined link. Default = **True**
bCheckRevision	Boolean	**True** = Link instruction tests the version number of the classes and links only when they are different (version in the library is larger than that in the project). bCheckRevision is only important when the bOverwrite argument is **True.** Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activates the option of the same name from the import dialog. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch

#def GetTemp() :
#
#  return os.environ["TEMP"]


def TestCopyClass() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Pfad zum leeren Test-Zielprojekt
  strDst = os.path.join(strCurDir,"LinkClassTest_RCD")
  # loeschen falls schon da
  shutil.rmtree(strDst, True)
  #erzeugen
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    raise batch.BatchError
  #print "Created empty destination project %s" % prjDst.Filename

  # Import-Projekt oeffnen
  strSrc = os.path.join(os.path.dirname(sys.argv[0]), r"..\..\Runtime\Library\OS Interface")
  prjSrc = batch.LoadProject(strSrc)
  if prjSrc==None :
    raise batch.BatchError
  #print "Loaded source project %s" % prjSrc.Filename
```

```
# Klasse kopieren
strClass = "_Grafix"
if batch.LinkClass(prjDst, prjSrc, strClass, True, True, False)==False :
  print "Link class %s failed." % strClass
  raise batch.BatchError

# compilieren
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError

# Vergleich
strRef = os.path.join(strCurDir, "CopyClassRef")
dc = filecmp.dircmp(strDst, strRef)
diff = dc.diff_files
if ( len(diff)>0 ) :
  print "Filecompare failed for files %s" % str(diff)
  raise batch.BatchError

print "Test successfull"
```

3.12.3.68 LinkClassList

LinkClassList(prjDst, prjSrc, keyClassList, bOverwrite=True, bCheckRevision=True, bUseReducedClientDependency=False)

Links a list of classes in a project.

Transfer parameters	Type	Description
prjDst	Project	The target project to which the class should be added
prjSrc	Project	The source project from which the classes originate
keyClassList	List	The list of class names that should be copied
bOverwrite	Boolean	**True** = a class already existing as copy of the same name is replaced by the defined link. Default = **True**
bCheckRevision	Boolean	**True** = Link instruction tests the version number of the classes and links only when they are different (version in the library is larger than that in the project). bCheckRevision is only important when the bOverwrite argument is **True.** Default = **True**
bUseReducedClientDependency	Boolean	**True** = Activates the option of the same name from the import dialog. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful

		False or **Exception**: failed

Example

```
import sigmatek.lasal.classes as sig
import sigmatek.lasal.batch   as bat

prjSrc = bat.LoadProject (sig.TheApplication().Path + "..\\Data\\Scripting\\CopyAndLinkCmd\\Li
bat.LinkClassList (sig.TheApplication().Project,
                   prjSrc,
                   ["C1", "C2"])
bat.CloseProject (prjSrc)
```

3.12.3.69 LinkObjectToPrjDriveFile

LinkObjectToPrjDriveFile(prjDst, strFilename, strObjectname, uiObjectIndex=0)

Creates a connection between a file from a drive folder and an object of a drive class within the project.

Transfer parameters	Type	Description
prjDst	Project	The target project
strFilename	String	Name of the drive parameter file that should be linked to the object of the drive class
strObjectname	String	Object name (represents the drive axis). If this parameter is left empty, the connection is removed.
uiObjectIndex	Number	Index of the file in the sequence of the files added to the same drive object (Default = 0)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path

import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestDriveFileKonfiguration() :
```

```
# Initalization
batch.SetExceptionOnError(False)
batch.CloseAllProjects()

strAppPath = cls.TheApplication().Path

# Current project
#myPrj = cls.TheApplication().Project
strPrj = os.path.join(strAppPath , r"..\Data\Scripting\DAandDriveTest")
try :
  os.remove(os.path.join(strPrj, r"Drive\AKM11B-230V.lda"))
except :
  print

try :
  os.remove(os.path.join(strPrj, r"Drive\AKM11B-230V_0.lda"))
except :
  print

try :
  os.remove(os.path.join(strPrj, r"Drive\AKM21C-230V.lda"))
except :
  print

myPrj = batch.LoadProject(strPrj)
if myPrj == None :
  raise batch.BatchError

bSuccess = True

batch.SetExceptionOnError(True)
# Add a drive file
batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM1\AKM11B
batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM1\AKM11B
batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM2\AKM21C

batch.SetExceptionOnError(False)
# Assign an object to a drive file
if batch.LinkObjectToPrjDriveFile(myPrj, "AKM11B-230V.lda", "Seppl") == True :
    bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "NoFile.lda", "NoObj") == True :
  bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "AKM11B-230V_0.lda", "_DriveAxis_21") == False :
  bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "AKM21C-230V.lda", "_DriveMng1") == True :
  bSuccess = False

# Delete the drive file
if batch.RemovePrjDriveFile(myPrj, r".\Drive\AKM11B-230V.lda") == False :
  bSuccess = False

if batch.RemovePrjDriveFile(myPrj, r".\Drive\NoFile-230V.lda") == True :
```

```
    bSuccess = False

  if batch.RemovePrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM2\AKM
    bSuccess = False

  # End
  print "Finished"
  batch.CloseProject(myPrj)
  if bSuccess == False :
        raise batch.BatchError
```

3.12.3.70 LoadProject

LoadProject(strFilename)

Loads an existing project.

Transfer parameters	Type	Description
strFilename	String	Name of the project file
Return parameters	**Type**	**Description**
	Project	**Project**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes     as sig
import sigmatek.lasal.batch               as batch

# Set the test directory
strDir = sig.TheApplication().Path + "..\\Data\\Scripting\\ProjectManagement\\"

# 1. only LoadProject, without project object it will be released when the
# application is released (by the ProjectManager)
batch.LoadProject (strDir + "LoadProject1")

# 2. with del the project is also released only on exit
prj2 = batch.LoadProject (strDir + "LoadProject2")
del prj2

# 3. CloseProject releases the project
prj3 = batch.LoadProject (strDir + "LoadProject3")
batch.CloseProject (prj3)

# 4. Assign a project to a variable 2x (LoadProject4 remains)
prj4 = batch.LoadProject (strDir + "LoadProject4")
prj4 = batch.LoadProject (strDir + "LoadProject5")
# closes LoadProject5
```

```
batch.CloseProject (prj4)

# 5. Load the same project twice
prj5 = batch.LoadProject (strDir + "LoadProject6")
prj6 = batch.LoadProject (strDir + "LoadProject6")
batch.CloseProject (prj5)
# The project can still be accessed via prj6
batch.Compile       (prj6)
# now LoadProject6 is really released
batch.CloseProject (prj6)
```

3.12.3.71 MoveNetworkToFolder

MoveNetworkToFolder(prjDst, keyNetwork, strFolder)

Moves a network to a folder. If it does not yet exist when the network is moved, it is created.

Transfer parameters	Type	Description
prjDst	Project	The target project
keyNetwork	String	Name of the network to move
strFolder	String	Path and name of the folder into which the network should be moved. Nonexistent folders are newly created
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import shutil
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

# A project with only 2 networks Net0 and Net1 is required:
def TestMoveNetworkToFolder() :
  # Initalization
  batch.SetExceptionOnError(True)
  batch.CloseAllProjects()
  # Init paths
  strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestMoveNetworkToFolder"
  # Remove existing project directory
  shutil.rmtree(strPrj, True)
  # Create new project
  myPrj = batch.CreateProject (strPrj)
  if myPrj == None :
```

```
    raise batch.BatchError

# Create required networks
batch.CreateNetwork(myPrj, "Net0")
batch.CreateNetwork(myPrj, "Net1")

# Start of the actual test routine
iCnt = 0
# Try several times (new errors may result):
while iCnt < 5 :
  batch.MoveNetworkToFolder(myPrj, "Net1", r"TestFolder\TestNet\NewFolder")
  batch.MoveNetworkToFolder(myPrj, "Net0", r"TestFolder\TestNet\NewFolder")

  batch.MoveNetworkToFolder(myPrj, "Net1", r"TestFolder\NextFolder\NewFolder")
  batch.MoveNetworkToFolder(myPrj, "Net0", r"TestFolder\NextFolder\NewFolder")

  batch.MoveNetworkToFolder(myPrj, "Net0", "")
  iCnt += 1

# End
batch.CloseProject(myPrj);
print "Finish"
```

3.12.3.72 OpenLogfile

OpenLogfile(strFilename, strLayout="[%d{%H:%M:%S} (%p) %c] %m%n", bAppend=False)

Opens a Log file.

Transfer parameters	Type	Description
strFilename	String	The name of the file
strLayout	String	Optional parameter for formatting the layout of error messages **%d** Date **%H** Hours **%M** Minutes **%S** Seconds **%p** Priority **%c** Category

		%m Message **%n** The platform specific line separator
bAppend	Boolean	**True** = Reports are added to an existing Log file. **False** = a new Log file is created. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sys
import os.path
import filecmp

import _internal
from sigmatek.lasal import batch

_internal.CloseAllProjects()

# make sure that logfile is in the same directory as script
strLogfile = os.path.join(os.path.dirname(sys.argv[0]),"LogTest.log")

_internal.OpenLogfile(strLogfile, "[(%p) %c] %m%n", False)
appPath = _internal.GetApplicationPath()

_internal.write("TestInternal")

_internal.logError("TestLogError")
_internal.logWarn("TestWarnung")
_internal.logInfo(appPath)

batch.CloseLogfile()

f = open(strLogfile, "r")

expectL1 = "[(ERROR) Scripting] TestLogError\n"
expectL2 = "[(WARN) Scripting] TestWarnung\n"

line = f.readline()                #1. line
if (line <> expectL1) :
        raise Exception
line = f.readline()                #2. line
if(line <> expectL2) :
        raise Exception
line = f.readline()                #3. line
if(line <> "[(INFO) Scripting] " + appPath + "\n") :
        raise Exception
```

```
f.close()
```

3.12.3.73 RemoveFile

RemoveFile(prjDst, strFilename)

Deletes a file from a project.

Transfer parameters	Type	Description
prjDst	Project	Target project from which the file is deleted
strFilename	String	The name of the file to be deleted
Return parameters	**Type**	**Description**
	Project	**Project**: successful **None** or **Exception**: failed

Example

```
import os.path as path
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sigmatek.lasal.classes as sig

def TestRemoveFile():

        # Initialize paths
        strFile = "\\Include\\TestHeader.h"

        # Get the test project
        prj = classes.TheApplication().Project
        if prj == None :
                print "Project \"%s\" not loaded." % prj.Filename
                raise batch.BatchError

        strFilePath = path.dirname(prj.Filename) + strFile

        if batch.RemoveFile(prj, strFilePath) == False :
                print "Removing file \"%s\" from project failed." % strFilePath
                raise batch.BatchError
```

3.12.3.74 RemovePrjDriveFile

RemovePrjDriveFile(prjDst, strFilename)

Removes a file from the drive folder of the defined project.

Transfer parameters	Type	Description
prjDst	Project	The target project
strFilename	String	Name and path of the drive parameter file that should be removed by the project
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path

import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestDriveFileKonfiguration() :
  # Initalization
  batch.SetExceptionOnError(False)
  batch.CloseAllProjects()

  strAppPath = cls.TheApplication().Path

  # Current project
  #myPrj = cls.TheApplication().Project
  strPrj = os.path.join(strAppPath , r"..\Data\Scripting\DAandDriveTest")
  try :
    os.remove(os.path.join(strPrj, r"Drive\AKM11B-230V.lda"))
  except :
    print

  try :
    os.remove(os.path.join(strPrj, r"Drive\AKM11B-230V_0.lda"))
  except :
    print

  try :
    os.remove(os.path.join(strPrj, r"Drive\AKM21C-230V.lda"))
  except :
    print

  myPrj = batch.LoadProject(strPrj)
```

```
if myPrj == None :
  raise batch.BatchError

bSuccess = True

batch.SetExceptionOnError(True)
# Add a drive file
batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM1\AKM11B-23
batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM1\AKM11B-23
batch.AddPrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM2\AKM21C-23

batch.SetExceptionOnError(False)
# Assign an object to a drive file
if batch.LinkObjectToPrjDriveFile(myPrj, "AKM11B-230V.lda", "Seppl") == True :
    bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "NoFile.lda", "NoObj") == True :
  bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "AKM11B-230V_0.lda", "_DriveAxis_21") == False :
  bSuccess = False

if batch.LinkObjectToPrjDriveFile(myPrj, "AKM21C-230V.lda", "_DriveMng1") == True :
  bSuccess = False

# Delete the drive file
if batch.RemovePrjDriveFile(myPrj, r".\Drive\AKM11B-230V.lda") == False :
  bSuccess = False

if batch.RemovePrjDriveFile(myPrj, r".\Drive\NoFile-230V.lda") == True :
  bSuccess = False

if batch.RemovePrjDriveFile(myPrj, strAppPath + r"..\RUNTIME\Drive\StdParameters\AKM\AKM2\AKM
  bSuccess = False

# End
print "Finished"
batch.CloseProject(myPrj)
if bSuccess == False :
      raise batch.BatchError
```

3.12.3.75 RenameNetwork

RenameNetwork(prjDst, strOldName, strNewName)

Renames a network.

Transfer parameters	Type	Description
prjDst	Project	The project that contains the network

strOldName	String	The old name of he network, which should be renamed
strNewName	String	The new name of the network
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch

#def GetTemp() :
#
#  return os.environ["TEMP"]


def TestRenameNetwork() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  strDst = os.path.join(strCurDir,"RenameNetworkTest")
  # delete if already there
  shutil.rmtree(strDst, True)
  # Create
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    raise batch.BatchError
  print "Created empty destination project %s" % prjDst.Filename

  # Create network
  strNetwork = "NewNetwork"
  if batch.CreateNetwork(prjDst, strNetwork)==False :
    print "Create network %s failed." % strNetwork
    raise batch.BatchError

  # Rename network
  strNetworkRename = "RenamedNetwork"
  if batch.RenameNetwork(prjDst, strNetwork, strNetworkRename)==False :
    print "Rename network %s to %s failed." % strNetwork % strNetworkRename
    raise batch.BatchError

  # compile
  if batch.Compile(prjDst)==False :
    print "Build %s failed." % prjDst.Filename
    raise batch.BatchError
```

```
# Comparison
strRef = os.path.join(strCurDir, "RenameNetworkRef")
dc = filecmp.dircmp(strDst, strRef)
diff = dc.diff_files
if ( len(diff)>0 ) :
  print "Filecompare failed for files %s" % str(diff)
  raise batch.BatchError

print "Test successfull"
```

3.12.3.76 RenameObject

RenameObject(prjDst, strNetwork, strOldName, strNewName)

Renames an object.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetwork	String	The name of the network that contains the object
strOldName	String	The old name of the object to be renamed
strNewName	String	The new name of the object
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def TestChangeClassAndRenameNetAndObj() :

        print "TestChangeClassAndRenameNetAndObj:"

        # Initialize paths
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestChangeClass"
        strPrjPath = (strDir + strPrj)

        # Load test project
        prj = batch.LoadProject (strPrjPath)

        if prj == None :
                print "Loading project \"%s\" failed." % (strPrjPath)
```

```
        raise batch.BatchError

strClass1  = "Class0"
strClass2  = "Class1"
strObject = "NewObject"
strRenamedObject = "RenamedObject"
strNetwork = "NewNetwork"
strRenamedNetwork = "RenamedNetwork"

if batch.CreateNetwork(prj, strNetwork)==False :
        print "Create network %s failed." % strNetwork
        raise batch.BatchError

# Create object
iPosX   = 200
iPosY   = 300
if batch.CreateObject(prj, strNetwork, strClass1, strObject, iPosX, iPosY)==False :
        print "Create object %s failed." % strObject
        raise batch.BatchError

# Change the class
if batch.ChangeClass(prj, strNetwork, strObject, strClass2)==False :
        print "Change class from %s to %s failed." % strClass1 % strClass2
        raise batch.BatchError

# compile
if batch.Compile(prj)==False :
        print "Build %s failed." % prj.Filename
        raise batch.BatchError

# rename network
if batch.RenameNetwork(prj, strNetwork, strRenamedNetwork)==False :
        print "Rename network %s to %s failed." % strNetwork % strRenamedNetwork
        raise batch.BatchError

# Rename object
if batch.RenameObject(prj, strRenamedNetwork, strObject, strRenamedObject)==False :
        print "Rename object %s to %s failed." % strObject % strRenamedObject
        raise batch.BatchError

# Delete network
if batch.DeleteNetwork(prj, strRenamedNetwork)==False :
        print "Delete network %s failed." % strRenamedNetwork
        raise batch.BatchError

# compile
if batch.Compile(prj)==False :
        print "Build %s failed." % prj.Filename
        raise batch.BatchError

batch.CloseAllProjects()

print "TestChangeClassAndRenameNetAndObj successful"
```

3.12.3.77 RenameVirtualCommissioningObject

RenameVirtualCommissioningObject(prjDst, strOldObjectName, strNewObjectName)

Changes the name of a Virtual Commissioning object.

Transfer parameters	Type	Description
prjDst	Project	The project in which the Virtual Commissioning object whose name is to be changed is located
strOldObjectName	String	Old name of the Virtual Commissioning object
strNewObjectName	String	New name of the Virtual Commissioning object
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import lasal.classes as cls
import sigmatek.lasal.batch as batch

def RenameVCObject() :
  prj = cls.TheApplication().Project
  prj.UseAdvancedIO = True
  batch.AddVirtualCommissioning(prj, "Class0", "OldSimulationObjectName")
  batch.RenameVirtualCommissioningObject(prj, "OldSimulationObjectName", "MyNewObjectName")

#Main
if __name__ == "__main__" :
RenameVCObject()
```

3.12.3.78 ReplaceObject

ReplaceObject(prjDst, prjSrc, strNetworkName, strObjectName, strSourceNetwork, strSourceObject, bUseSrcConnections=False, bCheckObjectName=True)

Replaces an object with another object.

Transfer parameters	Type	Description

prjDst	Project	The target project
prjSrc	Project	The project the new object is coming from
strNetworkName	String	The name of the network containing the object to be replaced
strObjectName	String	The name of the object
strSourceNetwork	String	The name of the network from which the new object comes
strSourceObject	String	The name of the new object
bUseSrcConnections	Boolean	If this parameter is set then the existing client connections of the imported object are taken over. Clients without connection have the same behaviour as with a parameter set to **False**. If the parameter is not set, then the original client connection should be entered. Default = **False**
bCheckObjectName	Boolean	If this parameter is set, it checks if an object named strSourceObject exists in the target project and if it is named differently than strObjectName. Default = **True**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch

def GetTemp() :

  return os.environ["TEMP"]


def TestReplaceObject() :

  print "TestReplaceObject:"

  batch.SetExceptionOnError(True)

  batch.CloseAllProjects()

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty test target project
  strDst = os.path.join(strCurDir,"ReplaceObjectTest")
  # Delete if already there
```

```
shutil.rmtree(strDst, True)
# Create
prjDst = batch.CreateProject(strDst)
if prjDst==None :
  raise batch.BatchError

#print "Created empty destination project %s" % prjDst.Filename

dirSrc = os.path.join(strCurDir,"ReplaceObject")
prjSrc = batch.LoadProject(dirSrc)
if prjSrc==None :
  raise batch.BatchError


#1. Test: Simply exchange one obj for another with a different name
batch.CopyNetwork(prjDst, prjSrc, "Network0")
batch.ReplaceObject(prjDst, prjSrc,"Network0","Dummy1","InitNet","ReplaceCls1");
# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError
# Comparison
strRef = os.path.join(strCurDir, r"ReplaceObject\ONE_NETWORK0_A.ref")
strNet = os.path.join(strDst,    r"Network\Network0\ONE_Network0_Table.st")
if filecmp.cmp(strRef, strNet) == False :
  raise batch.BatchError

#2. Test: Take over connections from new object
batch.CopyNetwork(prjDst, prjSrc, "Network0")
batch.ReplaceObject(prjDst, prjSrc,"Network0","Dummy1","InitNet","ReplaceCls1", True);
# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError
# Comparison
strRef = os.path.join(strCurDir, r"ReplaceObject\ONE_NETWORK0_B.ref")
strNet = os.path.join(strDst,    r"Network\Network0\ONE_Network0_Table.st")
if filecmp.cmp(strRef, strNet) == False :
  raise batch.BatchError

#3. Test: Replace object with other names and external connections
batch.CopyNetwork(prjDst, prjSrc, "ExternalObjs")
batch.CopyNetwork(prjDst, prjSrc, "Network0")
batch.ReplaceObject(prjDst, prjSrc,"Network0","Dummy2","InitNet","ReplaceCls1");
# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError
# Comparison
strRef = os.path.join(strCurDir, r"ReplaceObject\ONE_NETWORK0_C.ref")
strNet = os.path.join(strDst,    r"Network\Network0\ONE_Network0_Table.st")
if filecmp.cmp(strRef, strNet) == False :
  raise batch.BatchError
      # Comparison of the network whether superfluous connections were removed
strRef = os.path.join(strCurDir, r"ReplaceObject\Network0_C.ref")
```

```
  strNet = os.path.join(strDst,    r"Network\Network0\Network0.lcn")
  if filecmp.cmp(strRef, strNet) == False :
    raise batch.BatchError
        # Comparison of the network from where an external connection to the object leaves
  strRef = os.path.join(strCurDir, r"ReplaceObject\ONE_EXTERNALOBJS_C.ref")
  strNet = os.path.join(strDst,    r"Network\ExternalObjs\ONE_ExternalObjs_Table.st")
  if filecmp.cmp(strRef, strNet) == False :
    raise batch.BatchError

  print "TestReplaceObject successful"

  batch.CloseAllProjects();

  return True
```

3.12.3.79 ResetNetworkOptions

ResetNetworkOptions(prjDst, strNetworkName, listOptionNames)

Resets the network options of a network (see Dialog "Add File" for networks in the LASAL 2 user interface).

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	The name of the network whose options should be reset
listOptionNames	List	A list of the option names that should be reset
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestSetNetworkOptions() :
  print "Test ResetNetworkOptions."

  # Initalization
  batch.SetExceptionOnError(False)
  batch.CloseAllProjects()

  # Current project
  strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestNetworkOptions"
  myPrj = batch.LoadProject(strPrj)
```

```
if myPrj == None :
      print "Loading test project failed."
      raise batch.BatchError

#Begin of test

# Try several times (new errors may result):
for i in range(0, 5):
        #Reset Options Opt0, Opt1
        if batch.ResetNetworkOptions(myPrj, "NW0", ["Opt0", "Opt1"]) == False :
              # after SaveProject see : Opt0 and its files, Op1 reset
              print "ResetNetworkOptions for Opt0, Opt1 failed."
              raise batch.BatchError
        #Reset Options Opt1, Opt2, Opt3
        if batch.ResetNetworkOptions(myPrj, "NW0", ["Opt1", "Opt2", "Opt3"]) == False :
              # after SaveProject see : Opt1, Opt2, Opt3 and its files reset
              print "ResetNetworkOptions for Opt1, Opt2, Opt3 failed."
              raise batch.BatchError
        #Call function with wrong parameter OptionKey
        # Option "Opt" does not exist in network "NW0". Function change only "own" options,
        # not existing Options do not disturb. ResetNetworkOptions returns true.
        if batch.ResetNetworkOptions(myPrj, "NW0", ["Opt"]) == False :
              print "ResetNetworkOptions for Opt failed."
              raise batch.BatchError

#End
print "Finished"
batch.Save(myPrj)                                  # save is only to look, if Options set corr
batch.CloseProject(myPrj)
```

3.12.3.80 Save

Save(prjDst, bSaveAll = false)

Saves a project.

Transfer parameters	Type	Description
prjDst	Project	The target project
bSaveAll	Boolean	**True** ... a SaveAll() will be executed **False** ... a SaveChanges() will be executed. Default is **False**
Return parameters	**Type**	**Description**
	Boolean	**True** ... successful **False** or **Exception** ... failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsL
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.81 SaveAs

SaveAs(prjDst, strNewName, bClear=False, bCopyClass=False)

Saves a project under a new name.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNewName	String	The new project name
bClear	Boolean	If **True**, all temporary files are removed. Default=**False**
bCopyClass	Boolean	If **True**, all linked classes are copied. Default=**False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
# creates a completely empty new project and tries to save it
def main() :
  from sigmatek.lasal import batch
  from sigmatek.lasal import classes

  prj = classes.Project()
  batch.SaveAs(prj, r"D:\ToDelete\VeryEmpty4")
```

```
  batch.SaveAs(prj, r"D:\ToDelete\VeryEmpty5")

if __name__=="__main__" :
  main()
```

3.12.3.82 SaveProjectOnPLC

SaveProjectOnPLC(prjDst, strConnection="")

Saves a project on the PLC. Does not download the project and does not check if the projects are equal. The project parameter is just needed to get the connection string.

Transfer parameters	Type	Description
prjDst	Project	The target project
strConnection	String	Name from the address book. If this parameter is not used the setting from the project will be taken. Default = ""
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

#Use loaded GUI-Project
prjTest = classes.Project()

strIp = "10.10.116.184"

batch.ClearRamImage(strIp)

batch.Download(prjTest, strIp, True)
batch.Download(prjTest, strIp, False)

batch.Start(prjTest, strIp)

batch.Stop(prjTest, strIp)

batch.SaveProjectOnPLC(prjTest, strIp)

batch.CloseModemConnection()

batch.DownloadOS(strIp, "DummyOS")

del prjTest
```

3.12.3.83 SetAdvancedObject

SetAdvancedObject(prjDst, strTempObjectName, eValue)

Activates or deactivates the AdvancedIO object of an IO element.

Transfer parameters	Type	Description
prjDst	Project	The project that contains the IO element
strTempObjectName	String	Name of the IO element
eValue	Enum | Bool | Integer	As Enum: **True** to create an Advanced IO object, **False** to create a "simple" IO object. The values are to be written case-sensitively in the same way! (Are converted internally to 1 and 0)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def SetAdvIO() :
  prj = cls.TheApplication().Project
  batch.SetAdvancedObject(prj, "Example", _batch.True)

if __name__ == "__main__" :
        SetAdvIO()
```

3.12.3.84 SetClassProtection

SetClassProtection(prjDst, strClassName, strPassword, strPasswordNew, SecurityLevel)

Sets the password protection and security level of a class.

Transfer parameters	Type	Description
prjDst	Project	The target project

strClassName	String	Name of the class, whose security level should be changed
strPassword	String	Current password (empty string when the class is not currently protected)
strPasswordNew	String	New password. Must be filled for password protection or encryption. If this is not the case, the parameter is ignored.
SecurityLevel	Number	The new security level. The following options are available: 0 no protection 1 write protected 2 encrypted
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as sig

def TestSetClassProtection():
        # Init paths
        strDir = sig.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestSetProtection"
        strCls = "Class0"

        bSuccess = True

        # Load test project
        prj = batch.LoadProject (strDir + strPrj)
        if prj == None :
                print "Loading project \"%s\" failed." % strdir + strPrj
                raise batch.BatchError
                bSuccess = False

        if batch.SetClassProtection(prj, strCls, "", "myPwd", 1) == False :
                print "Changing protection level of class \"%s\" to \"write protected\" failed
                bSuccess = False

        if batch.SetClassProtection(prj, strCls, "myPwd", "myNewPwd", 2) == False :
                print "Changing protection level of class \"%s\" to \"encrypted\" failed." % s
                bSuccess = False

        if batch.SetClassProtection(prj, strCls, "myNewPwd", "", 0) == False :
                print "Changing protection level of class \"%s\" to \"none\" failed." % strCls
                bSuccess = False

        batch.CloseProject(prj)
```

```
        if bSuccess == False :
                raise batch.BatchError
```

3.12.3.85 SetCommentNetwork

SetCommentNetwork(prjDst, strNetworkName, strComment)

Sets a new comment to a object network.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	Name of the network containing the object which channel should be changed
strComment	String	The new commentary for the object network
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def GetTemp() :

  return os.environ["TEMP"]

def TestCreateNetwork() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Init paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\CreateNetworkTest"
  strPrjPath = (strDir + strPrj)

  print "strPrjPath: \"%s\"" % strPrjPath

  #create
  prjDst = batch.LoadProject(strPrjPath)
```

```
if prjDst==None :
  raise batch.BatchError
  print "Created empty destination project %s" % prjDst.Filename

# Create network
strNetwork = "NewNetwork"
if batch.CreateNetwork(prjDst, strNetwork)==False :
  print "Create network %s failed." % strNetwork
  raise batch.BatchError

# Set comment
strcomment = "TestComment"
if batch.SetCommentNetwork(prjDst, strNetwork, strcomment)==False :
  print "Set comment for network %s failed." % strNetwork
  raise batch.BatchError

# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError

# delete network
if batch.DeleteNetwork(prjDst, strNetwork)==False :
  print "Delete network %s failed." % strNetwork
  raise batch.BatchError

# compile
if batch.Compile(prjDst)==False :
  print "Build %s failed." % prjDst.Filename
  raise batch.BatchError

batch.CloseAllProjects();

print "Test successfull"
```

3.12.3.86 SetCommentObject

SetCommentObject(prjDst, strNetworkName, strObjectName, strComment)

Sets a new comment to a specific object.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	Name of the network containing the object which channel should be changed (could be empty)
strObjectName	String	Name of the object
strComment	String	The new comment for the object

Return parameters	Type	Description
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import _batch

def TestCreateObject() :

  print "TestCreateObject:"

  # Init paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\CreateObjectTest"
  strDst = (strDir + strPrj)

  prjDst = batch.LoadProject(strDst)
  if prjDst==None :
    print "Load project failed"
    raise batch.BatchError

  # Create network to fill
  strNetwork = "NewNetwork"
  if batch.CreateNetwork(prjDst, strNetwork)==False :
    print "Create network \"%s\" failed." % strNetwork
    raise batch.BatchError

  # Create object
  strClass  = "Class0"
  strObject = "NewObject"
  iPosX   = 200
  iPosY   = 300
  if _batch.CreateObject0(prjDst, strNetwork, strClass, strObject, iPosX, iPosY, False)==Fal
    print "Create object %s failed." % strObject
    raise batch.BatchError

  # Set comment
  strcomment = "TestComment"
  if batch.SetCommentObject(prjDst, strNetwork, strObject, strcomment)==False :
    print "Set comment for object %s failed." % strObject
    raise batch.BatchError

  # compile
  if batch.Compile(prjDst)==False :
```

```
    print "Build %s failed." % prjDst.Filename
    raise batch.BatchError

  if batch.ExistsObject (prjDst, strNetwork, strObject) == False :
    print "Object \"%s\" doesn't exist." % (strObject)
    raise batch.BatchError

  if batch.DeleteObject(prjDst, strNetwork, strObject) == False :
    print "Delete Object \"%s\" failed" % (strObject)
    raise batch.BatchError

  if batch.DeleteNetwork(prjDst, strNetwork)==False :
    print "Delete network \"%s\" failed." % strNetwork
    raise batch.BatchError

  # compile
  if batch.Compile(prjDst)==False :
    print "Build %s failed." % prjDst.Filename
    raise batch.BatchError

  batch.CloseAllProjects();

  print "TestCreateObject successfull"
```

3.12.3.87 SetCompilerVersion

SetCompilerVersion(prjDst, CompilerVer)

Sets the specified compiler version in the target project.

Transfer parameters	Type	Description
prjDst	Project	The target project
CompilerVer	Number	The compiler version
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

batch.CloseAllProjects()

strPath = classes.TheApplication().Path + r"..\Data\Scripting\TestCompilerVersion"
```

```
prjTest = batch.LoadProject(strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------------
#GetCurrentCompilerVersion & GetCompilerVersion    & SetCompilerVersion

version = batch.GetCurrentCompilerVersion()
if(version <> 27) :
        raise Exception

if(prjTest.CompilerOptions.CompilerVersion <> batch.GetCompilerVersion(prjTest))     :
        raise Exception

versionBefore = prjTest.CompilerOptions.CompilerVersion
batch.SetCompilerVersion(prjTest, 10)

if(prjTest.CompilerOptions.CompilerVersion <> batch.GetCompilerVersion(prjTest))     :
        raise Exception

if(versionBefore == prjTest.CompilerOptions.CompilerVersion) :
        raise Exception

if(10 <> prjTest.CompilerOptions.CompilerVersion) :
        raise Exception

# current version must not have been changed
if(version <> batch.GetCurrentCompilerVersion()) :
        raise Exception

batch.SetCompilerVersion(prjTest, batch.GetCurrentCompilerVersion())
if(version <> batch.GetCompilerVersion(prjTest)) :
        raise Exception

# End:
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.88 SetExceptionOnError

SetExceptionOnError(bExceptionFlag)

Determines whether an exception should be generated when an error occurs. Transfer value is **True** or **False**. The current value can be queried with IsExceptionOnError().

Transfer parameters	Type	Description

bExceptionFlag	Boolean	Defines, whether in case of an error an exception should be generated (**true**) or not (**false**)
Return parameters	**Type**	**Description**
none		

Example

```
import sigmatek.lasal.batch as batch

# Setting the value to True = an exception is thrown every time an error occurs
batch.SetExceptionOnError(True)

# Query the previously set value
bErr = batch.IsExceptionOnError()
```

3.12.3.89 SetFileProtection

SetFileProtection(prjDst, strFilePath, strPassword, strPasswordNew, securityLevel)

Sets the password protection and security level of a file.

Transfer parameters	Type	Description
prjDst	Project	The target project
strFilePath	String	Path of the file, whose security level should be changed (absolute or relative)
strPassword	String	Current password (empty string when the file is not currently protected)
strPasswordNew	String	New password. Must be filled for password protection or encryption. If this is not the case, the parameter is ignored.
SecurityLevel	Number	The new security level. The following options are available: 0 no protection 1 write protected 2 encrypted 3 encrypted and complex classes locked
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as sig

def TestSetFileProtection():
        # Init paths
        strDir = sig.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = strDir + "\\TestSetProtection"
        strFileRel = ".\\Source\\TestCpp.cpp"
        strFileAbs = strPrj + "\\Source\\TestCpp.cpp"

        bSuccess = True

        # Load test project
        prj = batch.LoadProject (strPrj)
        if prj == None :
                print "Loading project \"%s\" failed." % strPrj
                raise batch.BatchError
                bSuccess = False

        # Test with relative path specification
        if batch.SetFileProtection(prj, strFileRel, "", "myPwd", 1) == False :
                print "Changing protection level of file \"%s\" to \"write protected\" faile
                bSuccess = False

        if batch.SetFileProtection(prj, strFileRel, "myPwd", "myNewPwd", 2) == False :
                print "Changing protection level of file \"%s\" to \"encrypted\" failed." %
                bSuccess = False

        if batch.SetFileProtection(prj, strFileRel, "myNewPwd", "", 0) == False :
                print "Changing protection level of file \"%s\" to \"none\" failed." % strFi
                bSuccess = False

        # Test with absolute path specification
        if batch.SetFileProtection(prj, strFileAbs, "", "myPwd", 1) == False :
                print "Changing protection level of file \"%s\" to \"write protected\" faile
                bSuccess = False

        if batch.SetFileProtection(prj, strFileAbs, "myPwd", "myNewPwd", 2) == False :
                print "Changing protection level of file \"%s\" to \"encrypted\" failed." %
                bSuccess = False

        if batch.SetFileProtection(prj, strFileAbs, "myNewPwd", "", 0) == False :
                print "Changing protection level of file \"%s\" to \"none\" failed." % strFi
                bSuccess = False

        batch.CloseProject(prj)

        if bSuccess == False :
                raise batch.BatchError
```

3.12.3.90 SetInitValue

SetInitValue(prjDst, strNetworkName, strObjectName, strChannelName, strValue)

Sets the start value of a channel.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	The name of the network containing the object whose channel should be changed
strObjectName	String	The name of the object
strChannelName	String	The name of the channel whose init value should be changed
strValue	String	The new init value of the channel
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestSetInitValues() :
        # test0 : set InitValue to write-protected server in NW1.
        # SetInitValue must return False
        # in Object Network
        if batch.SetInitValue(myPrj, "NW1", "Base_C12", "Ser0", "100") == True :
                print "SetInitValue to NW1::Base_C12::Ser0 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "Base_C12", "Ser0", "100") == True :
                print "SetInitValue2 to Base_C12::Ser0 failed."
                raise batch.BatchError
        if batch.SetInitValue(myPrj, "NW1", "C12", "Ser0", "100") == True :
                print "SetInitValue to NW1::C12::Ser0 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C12", "Ser0", "100") == True :
                print "SetInitValue2 to C12::Ser0 failed."
                raise batch.BatchError
        if batch.SetInitValue(myPrj, "NW1", "C22", "Ser0", "100") == True :
                print "SetInitValue to NW1::C22::Ser0 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C22", "Ser0", "100") == True :
                print "SetInitValue2 to C22::Ser0 failed."
                raise batch.BatchError
```

```
# in Complex Network of Class
if batch.SetInitValue(myPrj, "Complex_C3", "Base_C11", "Ser0", "100") == True :
        print "SetInitValue to Complex_C3::Base_C11::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "Base_C11", "Ser0", "100") == True :
        print "SetInitValue2 to Base_C11::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C11", "Ser0", "100") == True :
        print "SetInitValue to Complex_C3::C11::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C11", "Ser0", "100") == True :
        print "SetInitValue2 to C11::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C21", "Ser0", "100") == True :
        print "SetInitValue to Complex_C3::C21::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C21", "Ser0", "100") == True :
        print "SetInitValue2 to C21::Ser0 failed."
        raise batch.BatchError

# test1 : set InitValue to not-write-protected, not initialized server in NW1
# SetInitValue must return False
if batch.SetInitValue(myPrj, "NW1", "Base_C12", "Ser1", "100") == True :
        print "SetInitValue to NW1::Base_C12::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "Base_C12", "Ser1", "100") == True :
        print "SetInitValue2 to Base_C12::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "NW1", "C12", "Ser1", "100") == True :
        print "SetInitValue to NW1::C12::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C12", "Ser1", "100") == True :
        print "SetInitValue2 to C12::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "NW1", "C22", "Ser1", "100") == True :
        print "SetInitValue to NW1::C22::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C22", "Ser1", "100") == True :
        print "SetInitValue2 to C22::Ser1 failed."
        raise batch.BatchError
# in Complex Network of Class
if batch.SetInitValue(myPrj, "Complex_C3", "Base_C11", "Ser1", "100") == True :
        print "SetInitValue to Complex_C3::Base_C11::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "Base_C11", "Ser1", "100") == True :
        print "SetInitValue2 to Base_C11::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C11", "Ser1", "100") == True :
        print "SetInitValue to Complex_C3::C11::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C11", "Ser1", "100") == True :
        print "SetInitValue2 to C11::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C21", "Ser1", "100") == True :
        print "SetInitValue to Complex_C3::C21::Ser1 failed."
```

```
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C21", "Ser1", "100") == True :
                print "SetInitValue2 to C21::Ser1 failed."
                raise batch.BatchError

        #test2 : set InitValue to not-write-protected, initialized server in NW1
        # SetInitValue must return True for ObjectNetworks - always, for other - if NetworkKey
        if batch.SetInitValue(myPrj, "NW1", "Base_C12", "Ser5", "100") == False :
                print "SetInitValue to NW1::Base_C12::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "Base_C12", "Ser5", "100") == False :
                print "SetInitValue2 to Base_C12::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue(myPrj, "NW1", "C12", "Ser55", "100") == False :
                print "SetInitValue to NW1::C12::Ser55 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C12", "Ser5", "100") == False :
                print "SetInitValue2 to C12::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue(myPrj, "NW1", "C22", "Ser5", "100") == False :
                print "SetInitValue to NW1::C22::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C22", "Ser5", "100") == False :
                print "SetInitValue2 to C22::Ser5 failed."
                raise batch.BatchError
        # in Complex Network of Class
        if batch.SetInitValue(myPrj, "Complex_C3", "Base_C11", "Ser5", "100") == False :
                print "SetInitValue to Complex_C3::Base_C11::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "Base_C11", "Ser5", "100") == True :
                print "SetInitValue2 to Base_C11::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue(myPrj, "Complex_C3", "C11", "Ser5", "100") == False :
                print "SetInitValue to Complex_C3::C11::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C11", "Ser5", "100") == True :
                print "SetInitValue2 to C11::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue(myPrj, "Complex_C3", "C11", "Ser55", "100") == False :
                print "SetInitValue to Complex_C3::C11::Ser55 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C11", "Ser55", "100") == True :
                print "SetInitValue2 to C11::Ser55 failed."
                raise batch.BatchError
        if batch.SetInitValue(myPrj, "Complex_C3", "C21", "Ser5", "100") == False :
                print "SetInitValue to Complex_C3::C21::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C21", "Ser5", "100") == True :
                print "SetInitValue2 to C21::Ser5 failed."
                raise batch.BatchError

# ----------------------- main ---------------------------------
if __name__=="__main__":
        print "Test SetInitValues."
```

```
# Initalization
batch.SetExceptionOnError(False)
batch.CloseAllProjects()

#Aktuelles Projekt
strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestSetInitValue"
myPrj = batch.LoadProject(strPrj)
if myPrj == None :
        print "Loading test project failed."
        raise batch.BatchError

TestSetInitValues()
batch.Save(myPrj)         # save project to see results
batch.CloseProject(myPrj)

# End
print "Finished"
```

3.12.3.91 SetInitValue2

SetInitValue2(prjDst, strObjectName, strChannelName, strValue)

Sets the start value of a channel.

Transfer parameters	Type	Description
prjDst	Project	The target project
strObjectName	String	The name of the object
strChannelName	String	The name of the channel, whose init value should be changed
strValue	String	The new init value of a channel
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestSetInitValues() :
        # test0 : set InitValue to write-protected server in NW1.
        # SetInitValue must return False
        # in Object Network
        if batch.SetInitValue(myPrj, "NW1", "Base_C12", "Ser0", "100") == True :
                print "SetInitValue to NW1::Base_C12::Ser0 failed."
                raise batch.BatchError
```

```
if batch.SetInitValue2(myPrj, "Base_C12", "Ser0", "100") == True :
        print "SetInitValue2 to Base_C12::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "NW1", "C12", "Ser0", "100") == True :
        print "SetInitValue to NW1::C12::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C12", "Ser0", "100") == True :
        print "SetInitValue2 to C12::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "NW1", "C22", "Ser0", "100") == True :
        print "SetInitValue to NW1::C22::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C22", "Ser0", "100") == True :
        print "SetInitValue2 to C22::Ser0 failed."
        raise batch.BatchError
# in Complex Network of Class
if batch.SetInitValue(myPrj, "Complex_C3", "Base_C11", "Ser0", "100") == True :
        print "SetInitValue to Complex_C3::Base_C11::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "Base_C11", "Ser0", "100") == True :
        print "SetInitValue2 to Base_C11::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C11", "Ser0", "100") == True :
        print "SetInitValue to Complex_C3::C11::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C11", "Ser0", "100") == True :
        print "SetInitValue2 to C11::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C21", "Ser0", "100") == True :
        print "SetInitValue to Complex_C3::C21::Ser0 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C21", "Ser0", "100") == True :
        print "SetInitValue2 to C21::Ser0 failed."
        raise batch.BatchError

# test1 : set InitValue to not-write-protected, not initialized server in NW1
# SetInitValue must return False
if batch.SetInitValue(myPrj, "NW1", "Base_C12", "Ser1", "100") == True :
        print "SetInitValue to NW1::Base_C12::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "Base_C12", "Ser1", "100") == True :
        print "SetInitValue2 to Base_C12::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "NW1", "C12", "Ser1", "100") == True :
        print "SetInitValue to NW1::C12::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C12", "Ser1", "100") == True :
        print "SetInitValue2 to C12::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "NW1", "C22", "Ser1", "100") == True :
        print "SetInitValue to NW1::C22::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C22", "Ser1", "100") == True :
        print "SetInitValue2 to C22::Ser1 failed."
        raise batch.BatchError
```

```
# in Complex Network of Class
if batch.SetInitValue(myPrj, "Complex_C3", "Base_C11", "Ser1", "100") == True :
        print "SetInitValue to Complex_C3::Base_C11::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "Base_C11", "Ser1", "100") == True :
        print "SetInitValue2 to Base_C11::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C11", "Ser1", "100") == True :
        print "SetInitValue to Complex_C3::C11::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C11", "Ser1", "100") == True :
        print "SetInitValue2 to C11::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C21", "Ser1", "100") == True :
        print "SetInitValue to Complex_C3::C21::Ser1 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C21", "Ser1", "100") == True :
        print "SetInitValue2 to C21::Ser1 failed."
        raise batch.BatchError

#test2 : set InitValue to not-write-protected, initialized server in NW1
# SetInitValue must return True for ObjectNetworks - always, for other - if NetworkK
if batch.SetInitValue(myPrj, "NW1", "Base_C12", "Ser5", "100") == False :
        print "SetInitValue to NW1::Base_C12::Ser5 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "Base_C12", "Ser5", "100") == False :
        print "SetInitValue2 to Base_C12::Ser5 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "NW1", "C12", "Ser55", "100") == False :
        print "SetInitValue to NW1::C12::Ser55 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C12", "Ser5", "100") == False :
        print "SetInitValue2 to C12::Ser5 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "NW1", "C22", "Ser5", "100") == False :
        print "SetInitValue to NW1::C22::Ser5 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C22", "Ser5", "100") == False :
        print "SetInitValue2 to C22::Ser5 failed."
        raise batch.BatchError
# in Complex Network of Class
if batch.SetInitValue(myPrj, "Complex_C3", "Base_C11", "Ser5", "100") == False :
        print "SetInitValue to Complex_C3::Base_C11::Ser5 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "Base_C11", "Ser5", "100") == True :
        print "SetInitValue2 to Base_C11::Ser5 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C11", "Ser5", "100") == False :
        print "SetInitValue to Complex_C3::C11::Ser5 failed."
        raise batch.BatchError
if batch.SetInitValue2(myPrj, "C11", "Ser5", "100") == True :
        print "SetInitValue2 to C11::Ser5 failed."
        raise batch.BatchError
if batch.SetInitValue(myPrj, "Complex_C3", "C11", "Ser55", "100") == False :
        print "SetInitValue to Complex_C3::C11::Ser55 failed."
```

```
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C11", "Ser55", "100") == True :
                print "SetInitValue2 to C11::Ser55 failed."
                raise batch.BatchError
        if batch.SetInitValue(myPrj, "Complex_C3", "C21", "Ser5", "100") == False :
                print "SetInitValue to Complex_C3::C21::Ser5 failed."
                raise batch.BatchError
        if batch.SetInitValue2(myPrj, "C21", "Ser5", "100") == True :
                print "SetInitValue2 to C21::Ser5 failed."
                raise batch.BatchError

# ----------------------- main --------------------------------
if __name__=="__main__":
        print "Test SetInitValues."

        # Initalization
        batch.SetExceptionOnError(False)
        batch.CloseAllProjects()

        #Aktuelles Projekt
        strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestSetInitValue"
        myPrj = batch.LoadProject(strPrj)
        if myPrj == None :
                print "Loading test project failed."
                raise batch.BatchError

        TestSetInitValues()
        batch.Save(myPrj)         # save project to see results
        batch.CloseProject(myPrj)

        # End
        print "Finished"
```

3.12.3.92 SetMultiCPUCore

SetMultiCPUCore(prjDst, bMultiCore=True)

Set using multiple CPU core.

Transfer parameters	Type	Description
prjDst	Project	The target project
bMultiCore	Boolean	If true, multiple CPU cores are used. Default = True
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sys
import _internal
from os import path
from sigmatek.lasal import batch
from sigmatek.lasal import classes

def TestMultiCore():
  # Init paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\TestMultiCore"
  strPrjPath = (strDir + strPrj)

  print "strPrjPath: \"%s\"" % strPrjPath

  # Load project
  prjDst = batch.LoadProject(strPrjPath)
  if prjDst==None :
    raise batch.BatchError

  classes.TheApplication().Project = prjDst

  if batch.SetMultiCPUCore(prjDst, True)==False :
    print "Set multiple CPU core of project %s failed." % strPrjPath
    raise batch.BatchError

  if batch.SetTaskCPUCore(prjDst, "", "Class01", "realtime", 1)==False :
    print "Set real CPU core failed."
    raise batch.BatchError

  if batch.SetTaskCPUCore(prjDst, "", "Class11", "cyclicwork", 1)==False :
    print "Set cyclic CPU core failed."
    raise batch.BatchError

  if batch.SetTaskCPUCore(prjDst, "", "Class11", "cyclicwork", 5)==True :
    print "Set cyclic CPU core failed (1)."
    raise batch.BatchError

  if batch.SetTaskCPUCore(prjDst, "", "Class111", "cyclicwork", 1)==True :
    print "Set cyclic CPU core failed (2)."
    raise batch.BatchError

  if batch.SetTaskCPUCore(prjDst, "", "Class11", "cyclicwork1", 1)==True :
    print "Set cyclic CPU core failed (3)."
    raise batch.BatchError

  batch.CloseAllProjects();
```

3.12.3.93 SetNetworkOptions

SetNetworkOptions(prjDst, strNetworkName, listOptionNames, bResetAllOthers=False)

Changes the network options of a network (see Dialog "Add File" for the networks in the LASAL 2 – user interface).

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	The name of the network whose options should be set
listOptionNames	List	A list of the option names that should be set
bResetAllOthers	Boolean	If this flag is set, all options not in the specified list are reset. Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestSetNetworkOptions() :
  print "Test SetNetworkOptions."

  # Initalization
  batch.SetExceptionOnError(False)
  batch.CloseAllProjects()

  # Current project
  strPrj = cls.TheApplication().Path + r"..\Data\Scripting\TestNetworkOptions"
  myPrj = batch.LoadProject(strPrj)
  if myPrj == None :
        print "Loading test project failed."
        raise batch.BatchError

  # Begin of test

  # Try several times (new errors may result):
  for i in range(0, 5):
          #Set Options Opt0, Opt1, reset other
          if batch.SetNetworkOptions(myPrj, "NW0", ["Opt0", "Opt1"], True) == False :
                # after SaveProject see : Opt0 and its files set, Op1 set,
                # Opt2, Opt3 and all its files reset.
                print "SetNetworkOptions for Opt0, Opt1 failed."
```

```
            raise batch.BatchError
        #Set Options Opt1, Opt2, Opt3, reset other
        if batch.SetNetworkOptions(myPrj, "NW0", ["Opt1", "Opt2", "Opt3"], True) == False
            # after SaveProject see : Opt1, Opt2, Opt3 and its files set
            # Opt0 and all its files reset.
            print "SetNetworkOptions for Opt1, Opt2, Opt3 failed."
            raise batch.BatchError
        #Set Options Opt0, Opt1, don't reset other. Parameters is List not as in Network t
        if batch.SetNetworkOptions(myPrj, "NW0", ["Opt1", "Opt0"], False) == False :
            # after SaveProject see : all Options and its files set
            print "SetNetworkOptions for Opt1, Opt0 failed."
            raise batch.BatchError
        #Set Options Opt1, Opt2, Opt3, don't reset other. Parameters is List not as in Net
        if batch.SetNetworkOptions(myPrj, "NW0", ["Opt1", "Opt3", "Opt2"], False) == False
            # after SaveProject see : all Options and its files set
            print "SetNetworkOptions for Opt1, Opt3, Opt2 failed."
            raise batch.BatchError

        #Call function with wrong parameter OptionKey
        # Option "Opt" does not exist in network "NW0". Function change only "own" options
        # not existing Options do not disturb. SetNetworkOptions returns true.
        if batch.SetNetworkOptions(myPrj, "NW0", ["Opt"], False) == False :
            print "SetNetworkOptions for Opt failed."
            raise batch.BatchError

#End
print "Finished"
batch.Save(myPrj)                                   # save is only to look, if Options set co
batch.CloseProject(myPrj)
```

3.12.3.94 SetOPCUAFolderObjectServer

SetOPCUAFolderObjectServer(prjDst, strNetworkName, strObject-Name, strServerName, strFolder)

Sets the OPC UA Settings / Default folder property in the specified server of the object.

Transfer parameters	Type	Description
prjDst	Project	Target project
strNetworkName	String	Name of the network in which the object is located
strObjectName	String	Name of the object which contains the server
strServerName	String	Name of the server
strFolder	String	Value of the OPC UA / Folder property
Return parameters	**Type**	**Description**

	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsLib
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------
#
batch.SetPrjOPCUAEnable(prjTest,True);
batch.SetVisualizedFlagOPCUA(prjTest, "Network0", "Class01", True);
batch.SetOPCUAPhysicObjectServer(prjTest, "Network0", "Class01", "physics" , "OPCUAPhysics")
batch.SetOPCUAUnitObjectServer(prjTest, "Network0", "Class01", "Units", "OPCUAUnits")
batch.SetOPCUAFolderObjectServer(prjTest, "Network0", "Class01", "Folder" , "OPCUAFolders")
batch.SetOPCUAHostnameObjectServer(prjTest, "Network0", "Class01", "Hostname" , "OPCUAHostname
batch.SetOPCUAWriteProtectedObjectServer(prjTest, "Network0", "Class01", "physics" , True)
batch.SetOPCUASettingObjectServer(prjTest, "Network0", "Class01", "ALL", "ALLPHYSICS" ,"ALLUNI
#
# End
batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.95 SetOPCUAHostnameObjectServer

SetOPCUAHostnameObjectServer(prjDst, strNetworkName, strObjectName, strServerName, strHostname)

Sets the OPC UA Settings / Hostname property in the specified server of the object.

Transfer parameters	Type	Description
prjDst	Project	Target project
strNetworkName	String	Name of the network in which the object is located
strObjectName	String	Name of the object which contains the server

strServerName	String	Name of the server
strHostname	String	Value of the OPC UA / Hostname property. By default, it preassigned with *Objectname_Servername*
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsL
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------
#
batch.SetPrjOPCUAEnable(prjTest,True);
batch.SetVisualizedFlagOPCUA(prjTest, "Network0", "Class01", True);
batch.SetOPCUAPhysicObjectServer(prjTest, "Network0", "Class01", "physics" , "OPCUAPhysics")
batch.SetOPCUAUnitObjectServer(prjTest, "Network0", "Class01", "Units", "OPCUAUnits")
batch.SetOPCUAFolderObjectServer(prjTest, "Network0", "Class01", "Folder" , "OPCUAFolders")
batch.SetOPCUAHostnameObjectServer(prjTest, "Network0", "Class01", "Hostname" , "OPCUAHostna
batch.SetOPCUAWriteProtectedObjectServer(prjTest, "Network0", "Class01", "physics" , True)
batch.SetOPCUASettingObjectServer(prjTest, "Network0", "Class01", "ALL", "ALLPHYSICS" ,"ALLU
#
# End
batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.96 SetOPCUAPhysicObjectServer

SetOPCUAPhysicObjectServer(prjDst, strNetworkName, strObject-Name, strServerName, strPhysic)

Sets the property OPC UA Settings / Default Physic in the server of the specified object.

Transfer parameters	Type	Description
prjDst	Project	Target project
strNetworkName	String	Name of the network in which the object is located
strObjectName	String	Name of the object which contains the server
strServerName	String	Name of the server
strPhysics	String	Value of the OPC UA / Physic property
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsLib
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------------
#
batch.SetPrjOPCUAEnable(prjTest,True);
batch.SetVisualizedFlagOPCUA(prjTest, "Network0", "Class01", True);
batch.SetOPCUAPhysicObjectServer(prjTest, "Network0", "Class01", "physics" , "OPCUAPhysics")
batch.SetOPCUAUnitObjectServer(prjTest, "Network0", "Class01", "Units", "OPCUAUnits")
batch.SetOPCUAFolderObjectServer(prjTest, "Network0", "Class01", "Folder" , "OPCUAFolders")
batch.SetOPCUAHostnameObjectServer(prjTest, "Network0", "Class01", "Hostname" , "OPCUAHostname
batch.SetOPCUAWriteProtectedObjectServer(prjTest, "Network0", "Class01", "physics" , True)
batch.SetOPCUASettingObjectServer(prjTest, "Network0", "Class01", "ALL", "ALLPHYSICS" ,"ALLUNI
#
# End
batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.97 SetOPCUASettingObjectServer

SetOPCUASettingObjectServer(prjDst, strNetworkName, strObjectName, strServerName, strPhysic, strUnit, strFolder, strHostname, blsWriteProtected)

Sets all OPC UA Settings variables in the specified server of the object.

Transfer parameters	Type	Description
prjDst	Project	Target project
strNetworkName	String	Name of the network in which the object is located
strObjectName	String	Name of the object which contains the server
strServerName	String	Name of the server
strPhysics	String	Value of the OPC UA / Physic property
strUnit	String	Value of the OPC UA / Unit property
strFolder	String	Value of the OPC UA / Folder property
strHostname	String	Value of the OPC UA / Hostname property. By default, it preassigned with *Objectname_Servername*
blsWriteProtected	Boolean	**True** (write protection = on), **False** (write protection = off). Caution! If in either the class or the object WriteProtected already = **True**, the server of the object cannot be set to WriteProtected = **False**!
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsL
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------
```

```
#
batch.SetPrjOPCUAEnable(prjTest,True);
batch.SetVisualizedFlagOPCUA(prjTest, "Network0", "Class01", True);
batch.SetOPCUAPhysicObjectServer(prjTest, "Network0", "Class01", "physics" , "OPCUAPhysics")
batch.SetOPCUAUnitObjectServer(prjTest, "Network0", "Class01", "Units", "OPCUAUnits")
batch.SetOPCUAFolderObjectServer(prjTest, "Network0", "Class01", "Folder" , "OPCUAFolders")
batch.SetOPCUAHostnameObjectServer(prjTest, "Network0", "Class01", "Hostname" , "OPCUAHostname
batch.SetOPCUAWriteProtectedObjectServer(prjTest, "Network0", "Class01", "physics" , True)
batch.SetOPCUASettingObjectServer(prjTest, "Network0", "Class01", "ALL", "ALLPHYSICS" ,"ALLUNI
#
# End
batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.98 SetOPCUAUnitObjectServer

SetOPCUAUnitObjectServer(prjDst, strNetworkName, strObjectName, strServerName, strUnit)

Sets the OPC UA Settings / Default Unit property in the specified server of the object.

Transfer parameters	Type	Description
prjDst	Project	Target project
strNetworkName	String	Name of the network in which the object is located
strObjectName	String	Name of the object which contains the server
strServerName	String	Name of the server
strUnit	String	Value of the OPC UA / Unit property
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsLib
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)
```

```
if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------------
#
batch.SetPrjOPCUAEnable(prjTest,True);
batch.SetVisualizedFlagOPCUA(prjTest, "Network0", "Class01", True);
batch.SetOPCUAPhysicObjectServer(prjTest, "Network0", "Class01", "physics" , "OPCUAPhysics")
batch.SetOPCUAUnitObjectServer(prjTest, "Network0", "Class01", "Units", "OPCUAUnits")
batch.SetOPCUAFolderObjectServer(prjTest, "Network0", "Class01", "Folder" , "OPCUAFolders")
batch.SetOPCUAHostnameObjectServer(prjTest, "Network0", "Class01", "Hostname" , "OPCUAHostnar
batch.SetOPCUAWriteProtectedObjectServer(prjTest, "Network0", "Class01", "physics" , True)
batch.SetOPCUASettingObjectServer(prjTest, "Network0", "Class01", "ALL", "ALLPHYSICS" ,"ALLU
#
# End
batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.99 SetOPCUAWriteProtectedObjectServer

SetOPCUAWriteProtectedObjectServer(prjDst, strNetworkName, strObjectName, strServerName, bIsWriteProtected)

Sets the OPC UA Settings / Write Protected property in the specified server of the object.

Transfer parameters	Type	Description
prjDst	Project	Target project
strNetworkName	String	Name of the network in which the object is located
strObjectName	String	Name of the object which contains the server
strServerName	String	Name of the server
bIsWriteProtected	Boolean	**True** (write protection = on), **False** (write protection = off). Caution! If in either the class or the object WriteProtected already = **True**, the server of the object cannot be set to WriteProtected = **False**!
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch
```

```
strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsLib
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------------
#
batch.SetPrjOPCUAEnable(prjTest,True);
batch.SetVisualizedFlagOPCUA(prjTest, "Network0", "Class01", True);
batch.SetOPCUAPhysicObjectServer(prjTest, "Network0", "Class01", "physics" , "OPCUAPhysics")
batch.SetOPCUAUnitObjectServer(prjTest, "Network0", "Class01", "Units", "OPCUAUnits")
batch.SetOPCUAFolderObjectServer(prjTest, "Network0", "Class01", "Folder" , "OPCUAFolders")
batch.SetOPCUAHostnameObjectServer(prjTest, "Network0", "Class01", "Hostname" , "OPCUAHostname
batch.SetOPCUAWriteProtectedObjectServer(prjTest, "Network0", "Class01", "physics" , True)
batch.SetOPCUASettingObjectServer(prjTest, "Network0", "Class01", "ALL", "ALLPHYSICS" ,"ALLUNI
#
# End
batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.10 SetParameterValue

SetParameterValue(prjDst, strNetworkName, strObjectName, strParameterName, strValue)

Sets parameter values of any object.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	Name of the network in which the object is located
strObjectName	String	Name of the object whose parameter is to be changed
strParameterName	String	Name of the parameter to be changed
strValue	String	New value of the parameter
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sys
import _internal
from os import path
from sigmatek.lasal import batch
from sigmatek.lasal import classes

def TestSetParameter():
 # Init paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\TestMultiCore"
  strPrjPath = (strDir + strPrj)

  print "strPrjPath: \"%s\"" % strPrjPath

  # Load project
  prjDst = batch.LoadProject(strPrjPath)
  if prjDst==None :
    raise batch.BatchError
    print "Created empty destination project %s" % prjDst.Filename

  classes.TheApplication().Project = prjDst

  if batch.SetParameterValue(prjDst, "HW_Network", "SdiasPLC1", "CableLength", "5") == False
                print "SetParameterValue to HW_Network::SdiasPLC1::CableLength failed."
                raise batch.BatchError

  print "TestSetParameter successful"

  batch.CloseAllProjects();
```

3.12.3.10 SetPrjConnectionFileActive

SetPrjConnectionFileActive(prjDst, strFilename, bActive)

Set the activation flag of the specified connection file.

Transfer parameters	Type	Description
prjDst	Project	The target project
strFilename	String	The connection file to set active /inactive
bActive	Boolean	Flag which indicates whether the connection file should be activated (**true**) or deactivated (**false**)
Return parameters	**Type**	**Description**
	Boolean	**True**: successful

		False or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

def GetTemp() :

  return os.environ["TEMP"]

def TestConnectionFiles() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Init paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\ConnectionFiles"
  strPrjPath = (strDir + strPrj)

  bSuccess = True

  print "strPrjPath: \"%s\"" % strPrjPath

  # Load project
  prjDst = batch.LoadProject(strPrjPath)
  if prjDst==None :
    bSuccess = False#raise batch.BatchError
    print "Loaded destination project %s" % prjDst.Filename

  # Set the old connection file inactive
  strFilenameOld = "TestConFile1.lce"
  if batch.SetPrjConnectionFileActive(prjDst, strFilenameOld, False)==False :
    print "Change active flag or connection file %s failed." % strFilenameOld
    bSuccess = False

  # Add new connection file
  strFilenameNew = "TestConFile2.lce"
  strNetworkSubPath = "\\Network\\"
  strFilepathNew = strPrjPath + strNetworkSubPath + strFilenameNew
  if batch.AddPrjConnectionFile(prjDst, strFilepathNew)==False :
    print "Adding connection file %s failed." % strFilenameNew
    bSuccess = False #raise batch.BatchError

  # Delete old connection file
  if batch.DeletePrjConnectionFile(prjDst, strFilenameOld)==False :
    print "Delete connection file %s failed." % strFilenameOld
    raise batch.BatchError
```

```
batch.CloseAllProjects();

if bSuccess == False :
  raise batch.BatchError

print "Test successfull"
```

3.12.3.10 SetPrjOPCUAEnable

SetPrjOPCUAEnable(prjDst, bOPCUAEnable)

Switches the OPC UA in the project on or off.

Transfer parameters	Type	Description
prjDst	Project	Target project
bOPCUAEnable	Boolean	If = **True**, settings can be made in the OPC UA servers; if = **False**, then no settings can be made.
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsL
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------
#
batch.SetPrjOPCUAEnable(prjTest,True);
batch.SetVisualizedFlagOPCUA(prjTest, "Network0", "Class01", True);
batch.SetOPCUAPhysicObjectServer(prjTest, "Network0", "Class01", "physics" , "OPCUAPhysics")
batch.SetOPCUAUnitObjectServer(prjTest, "Network0", "Class01", "Units", "OPCUAUnits")
batch.SetOPCUAFolderObjectServer(prjTest, "Network0", "Class01", "Folder" , "OPCUAFolders")
batch.SetOPCUAHostnameObjectServer(prjTest, "Network0", "Class01", "Hostname" , "OPCUAHostna
batch.SetOPCUAWriteProtectedObjectServer(prjTest, "Network0", "Class01", "physics" , True)
```

```
batch.SetOPCUASettingObjectServer(prjTest, "Network0", "Class01", "ALL", "ALLPHYSICS" ,"ALLUNI
#
# End
batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.10 SetTaskCPUCore

SetTaskCPUCore(prjDst, strNetworkName, strObjectName, strTask, iCore)

Assigns a task to a designated CPU core.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	The name of the network that contains the object
strObjectName	String	The name of the object whose task sequence time should be changed
strTask	String	Task name [realtime | cyclicwork]
iCore	Number	The new CPU core to which the task is assigned
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sys
import _internal
from os import path
from sigmatek.lasal import batch
from sigmatek.lasal import classes

def TestMultiCore():
  # Init paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strPrj = "\\TestMultiCore"
  strPrjPath = (strDir + strPrj)

  print "strPrjPath: \"%s\"" % strPrjPath

  # Load project
  prjDst = batch.LoadProject(strPrjPath)
  if prjDst==None :
```

```
    raise batch.BatchError

classes.TheApplication().Project = prjDst

if batch.SetMultiCPUCore(prjDst, True)==False :
  print "Set multiple CPU core of project %s failed." % strPrjPath
  raise batch.BatchError

if batch.SetTaskCPUCore(prjDst, "", "Class01", "realtime", 1)==False :
  print "Set real CPU core failed."
  raise batch.BatchError

if batch.SetTaskCPUCore(prjDst, "", "Class11", "cyclicwork", 1)==False :
  print "Set cyclic CPU core failed."
  raise batch.BatchError

if batch.SetTaskCPUCore(prjDst, "", "Class11", "cyclicwork", 5)==True :
  print "Set cyclic CPU core failed (1)."
  raise batch.BatchError

if batch.SetTaskCPUCore(prjDst, "", "Class111", "cyclicwork", 1)==True :
  print "Set cyclic CPU core failed (2)."
  raise batch.BatchError

if batch.SetTaskCPUCore(prjDst, "", "Class11", "cyclicwork1", 1)==True :
  print "Set cyclic CPU core failed (3)."
  raise batch.BatchError

batch.CloseAllProjects();
```

3.12.3.10 SetTaskOrder

SetTaskOrder(prjDst, strNetworkName, strObjectName, strTask, strPos)

Changing the order of tasks.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	The name of the network containing the object
strObjectName	String	The name of the object in which the application sequence should be changed
strTask	String	The name of the task [realtime | cyclicwork | background]
strPos	String	The new task-position of the application
Return parameters	**Type**	**Description**

	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestObjectTaskConfiguration() :
  # Initalization
  batch.SetExceptionOnError(False)
  testPrj = cls.TheApplication().Project

  batch.SetTaskTime(testPrj, "", "Obj1", "realtime", "100")
  batch.SetTaskTime(testPrj, "", "Obj1", "cyclicwork", "200")
  batch.SetTaskTime(testPrj, "", "Obj1", "background", "300")

  batch.SetTaskOrder(testPrj, "", "Obj1", "realtime", "Sigmatek Outputs")
  batch.SetTaskOrder(testPrj, "", "Obj1", "cyclicwork", "Sigmatek Outputs")
  batch.SetTaskOrder(testPrj, "", "Obj1", "background", "Sigmatek Outputs")

  # Wait until OS has implemented function:
  #batch.SetTaskOffset(testPrj, "", "Obj1", "realtime", "10")
  #batch.SetTaskOffset(testPrj, "", "Obj1", "cyclicwork", "20")
  #batch.SetTaskOffset(testPrj, "", "Obj1", "background", "30")

  batch.SetTaskTime2(testPrj, "Obj2", "realtime", "100")
  batch.SetTaskTime2(testPrj, "Obj2", "cyclicwork", "200")
  batch.SetTaskTime2(testPrj, "Obj2", "background", "300")

  batch.SetTaskOrder2(testPrj, "Obj2", "realtime", "Post 1")
  batch.SetTaskOrder2(testPrj, "Obj2", "cyclicwork", "Post 2")
  batch.SetTaskOrder2(testPrj, "Obj2", "background", "Post 3")

  # Wait until OS has implemented function:
  #batch.SetTaskOffset2(testPrj, "Obj2", "realtime", "10")
  #batch.SetTaskOffset2(testPrj, "Obj2", "cyclicwork", "20")
  #batch.SetTaskOffset2(testPrj, "Obj2", "background", "30")

  batch.SetTaskTime(testPrj, "", "Obj3", "realtime", "100µs")
  batch.SetTaskTime(testPrj, "", "Obj3", "cyclicwork", "250µs")
  batch.SetTaskTime(testPrj, "", "Obj3", "background", "always")

  batch.SetTaskTime(testPrj, "", "Obj4", "realtime", "off")
  batch.SetTaskTime(testPrj, "", "Obj4", "cyclicwork", "off")
  batch.SetTaskTime(testPrj, "", "Obj4", "background", "off")
```

3.12.3.10 SetTaskOrder2

SetTaskOrder2(prjDst, strObjectPath, strTask, strPos)

Changing the order of tasks.

Transfer parameters	Type	Description
prjDst	Project	The target project
strObjectPath	String	The name of the object in which the sequence should be changed
strTask	String	The name of the task [realtime | cyclicwork | background]
strPos	String	The new task position of the application
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestObjectTaskConfiguration() :
  # Initalization
  batch.SetExceptionOnError(False)
  testPrj = cls.TheApplication().Project

  batch.SetTaskTime(testPrj, "", "Obj1", "realtime", "100")
  batch.SetTaskTime(testPrj, "", "Obj1", "cyclicwork", "200")
  batch.SetTaskTime(testPrj, "", "Obj1", "background", "300")

  batch.SetTaskOrder(testPrj, "", "Obj1", "realtime", "Sigmatek Outputs")
  batch.SetTaskOrder(testPrj, "", "Obj1", "cyclicwork", "Sigmatek Outputs")
  batch.SetTaskOrder(testPrj, "", "Obj1", "background", "Sigmatek Outputs")

  # Wait until OS has implemented function:
  #batch.SetTaskOffset(testPrj, "", "Obj1", "realtime", "10")
  #batch.SetTaskOffset(testPrj, "", "Obj1", "cyclicwork", "20")
  #batch.SetTaskOffset(testPrj, "", "Obj1", "background", "30")

  batch.SetTaskTime2(testPrj, "Obj2", "realtime", "100")
  batch.SetTaskTime2(testPrj, "Obj2", "cyclicwork", "200")
  batch.SetTaskTime2(testPrj, "Obj2", "background", "300")

  batch.SetTaskOrder2(testPrj, "Obj2", "realtime", "Post 1")
  batch.SetTaskOrder2(testPrj, "Obj2", "cyclicwork", "Post 2")
  batch.SetTaskOrder2(testPrj, "Obj2", "background", "Post 3")

  # Wait until OS has implemented function:
```

```
#batch.SetTaskOffset2(testPrj, "Obj2", "realtime", "10")
#batch.SetTaskOffset2(testPrj, "Obj2", "cyclicwork", "20")
#batch.SetTaskOffset2(testPrj, "Obj2", "background", "30")

batch.SetTaskTime(testPrj, "", "Obj3", "realtime", "100µs")
batch.SetTaskTime(testPrj, "", "Obj3", "cyclicwork", "250µs")
batch.SetTaskTime(testPrj, "", "Obj3", "background", "always")

batch.SetTaskTime(testPrj, "", "Obj4", "realtime", "off")
batch.SetTaskTime(testPrj, "", "Obj4", "cyclicwork", "off")
batch.SetTaskTime(testPrj, "", "Obj4", "background", "off")
```

3.12.3.10 SetTaskTime

SetTaskTime(prjDst, strNetworkName, strObjectName, strTask, strTime)

Set the task time of an object.

Transfer parameters	Type	Description
prjDst	Project	The target project
strNetworkName	String	The name of the network containing the object
strObjectName	String	The name of the object, whose task time should be changed
strTask	String	The name of the application [realtime | cyclicwork | background]
strTime	String	The new task time
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestObjectTaskConfiguration() :
  # Initalization
  batch.SetExceptionOnError(False)
  testPrj = cls.TheApplication().Project

  batch.SetTaskTime(testPrj, "", "Obj1", "realtime", "100")
  batch.SetTaskTime(testPrj, "", "Obj1", "cyclicwork", "200")
  batch.SetTaskTime(testPrj, "", "Obj1", "background", "300")

  batch.SetTaskOrder(testPrj, "", "Obj1", "realtime", "Sigmatek Outputs")
```

```
batch.SetTaskOrder(testPrj, "", "Obj1", "cyclicwork", "Sigmatek Outputs")
batch.SetTaskOrder(testPrj, "", "Obj1", "background", "Sigmatek Outputs")

# Wait until OS has implemented function:
#batch.SetTaskOffset(testPrj, "", "Obj1", "realtime", "10")
#batch.SetTaskOffset(testPrj, "", "Obj1", "cyclicwork", "20")
#batch.SetTaskOffset(testPrj, "", "Obj1", "background", "30")

batch.SetTaskTime2(testPrj, "Obj2", "realtime", "100")
batch.SetTaskTime2(testPrj, "Obj2", "cyclicwork", "200")
batch.SetTaskTime2(testPrj, "Obj2", "background", "300")

batch.SetTaskOrder2(testPrj, "Obj2", "realtime", "Post 1")
batch.SetTaskOrder2(testPrj, "Obj2", "cyclicwork", "Post 2")
batch.SetTaskOrder2(testPrj, "Obj2", "background", "Post 3")

# Wait until OS has implemented function:
#batch.SetTaskOffset2(testPrj, "Obj2", "realtime", "10")
#batch.SetTaskOffset2(testPrj, "Obj2", "cyclicwork", "20")
#batch.SetTaskOffset2(testPrj, "Obj2", "background", "30")

batch.SetTaskTime(testPrj, "", "Obj3", "realtime", "100µs")
batch.SetTaskTime(testPrj, "", "Obj3", "cyclicwork", "250µs")
batch.SetTaskTime(testPrj, "", "Obj3", "background", "always")

batch.SetTaskTime(testPrj, "", "Obj4", "realtime", "off")
batch.SetTaskTime(testPrj, "", "Obj4", "cyclicwork", "off")
batch.SetTaskTime(testPrj, "", "Obj4", "background", "off")
```

3.12.3.10 SetTaskTime2

SetTaskTime2(prjDst, strObjectPath, strTask, strTime)

Sets a new task time for an object.

Transfer parameters	Type	Description
prjDst	Project	The target project
strObjectPath	String	The full name of the object, whose task time should be changed
strTask	String	The name of the application [realtime | cyclicwork | background]
strTime	String	The new task time
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

def TestObjectTaskConfiguration() :
  # Initalization
  batch.SetExceptionOnError(False)
  testPrj = cls.TheApplication().Project

  batch.SetTaskTime(testPrj, "", "Obj1", "realtime", "100")
  batch.SetTaskTime(testPrj, "", "Obj1", "cyclicwork", "200")
  batch.SetTaskTime(testPrj, "", "Obj1", "background", "300")

  batch.SetTaskOrder(testPrj, "", "Obj1", "realtime", "Sigmatek Outputs")
  batch.SetTaskOrder(testPrj, "", "Obj1", "cyclicwork", "Sigmatek Outputs")
  batch.SetTaskOrder(testPrj, "", "Obj1", "background", "Sigmatek Outputs")

  # Wait until OS has implemented function:
  #batch.SetTaskOffset(testPrj, "", "Obj1", "realtime", "10")
  #batch.SetTaskOffset(testPrj, "", "Obj1", "cyclicwork", "20")
  #batch.SetTaskOffset(testPrj, "", "Obj1", "background", "30")

  batch.SetTaskTime2(testPrj, "Obj2", "realtime", "100")
  batch.SetTaskTime2(testPrj, "Obj2", "cyclicwork", "200")
  batch.SetTaskTime2(testPrj, "Obj2", "background", "300")

  batch.SetTaskOrder2(testPrj, "Obj2", "realtime", "Post 1")
  batch.SetTaskOrder2(testPrj, "Obj2", "cyclicwork", "Post 2")
  batch.SetTaskOrder2(testPrj, "Obj2", "background", "Post 3")

  # Wait until OS has implemented function:
  #batch.SetTaskOffset2(testPrj, "Obj2", "realtime", "10")
  #batch.SetTaskOffset2(testPrj, "Obj2", "cyclicwork", "20")
  #batch.SetTaskOffset2(testPrj, "Obj2", "background", "30")

  batch.SetTaskTime(testPrj, "", "Obj3", "realtime", "100µs")
  batch.SetTaskTime(testPrj, "", "Obj3", "cyclicwork", "250µs")
  batch.SetTaskTime(testPrj, "", "Obj3", "background", "always")

  batch.SetTaskTime(testPrj, "", "Obj4", "realtime", "off")
  batch.SetTaskTime(testPrj, "", "Obj4", "cyclicwork", "off")
  batch.SetTaskTime(testPrj, "", "Obj4", "background", "off")
```

3.12.3.10 SetVirtualCommissionProperties

SetVirtualCommissionProperties(prjDst, strObjectName, dictProperties)

Sets the properties of a Virtual Commissioning object.

Transfer parameters	Type	Description
prjDst	Project	Name of the target project that contains the Virtual Commissioning object
strObjectName	String	Name of the Virtual Commissioning object
dictProperties	String	A map (Key + Value) of the properties, composed of name and value (both as string!). Here it is to be noted that **also number must be indicated as string** as well as that in case of a **value "true"** this **must be written consistently lower case**!
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False**, or **Exception**: failed or no prjDst specified

Example

```
import sigmatek.lasal.classes as cls
import sigmatek.lasal.batch as batch

def Properties() :
  prj = cls.TheApplication().Project
  props = batch.GetVirtualCommissionProperties(prj, "VirtTwin");
  print props
  props["PortNumber"] = "125"
  props["IPAddress"] = "10.10.116.39"
  print props
  batch.SetVirtualCommissionProperties(prj, "VirtTwin", props);

#Main
if __name__ == "__main__" :
Properties()
```

3.12.3.10 SetVisualizedFlag

SetVisualizedFlag(prjDst, strNetworkName, strObjectName, bIsVisualized)

Sets the visualization flag of the object.

Transfer parameters	Type	Description

prjDst	Project	The target project
strNetworkName	String	The name of the network in which the object is located
strObjectName	String	Object name
blsVisualized	Boolean	Indicates whether the object should be visualized
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sigmatek.lasal.classes as sig

def TestSetVisualizedFlag():
        strObjName = "TestObj"
        strNetName = "TestNet"

        # Load test project
        prj = classes.TheApplication().Project
        if prj == None :
                print "Project \"%s\" not loaded." % prj.Filename
                raise batch.BatchError

        if batch.SetVisualizedFlag(prj, strNetName, strObjName, True) == False :
                print "Changing visualized flag of object \"%s\" to true failed." % strObjName
                raise batch.BatchError

        if batch.SetVisualizedFlag(prj, strNetName, strObjName, False) == False :
                print "Changing visualized flag of object \"%s\" to false failed." % strObjNam
                raise batch.BatchError
```

3.12.3.11 SetVisualizedFlag2

SetVisualizedFlag2(prjDst, strObjectPath, blsVisualized)

Sets the visualization flag of the object.

Transfer parameters	Type	Description
prjDst	Project	The target project
strObjectPath	String	Full name (incl. path) of the object

blsVisualized	Boolean	Indicates whether the object should be visualized
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sigmatek.lasal.classes as sig

def TestSetVisualizedFlag():
        strObjName = "TestObj"
        strNetName = "TestNet"

        # Load test project
        prj = classes.TheApplication().Project
        if prj == None :
                print "Project \"%s\" not loaded." % prj.Filename
                raise batch.BatchError

        if batch.SetVisualizedFlag2(prj, strNetName + "." + strObjName, False) == False :
                print "Changing visualized flag of object \"%s\" to false failed." % strObjN
                raise batch.BatchError

        if batch.SetVisualizedFlag2(prj, strNetName + "." + strObjName, True) == False :
                print "Changing visualized flag of object \"%s\" to true failed." % strObjNa
                raise batch.BatchError
```

3.12.3.11 SetVisualizedFlagOPCUA

SetVisualizedFlagOPCUA(prjDst, strNetworkName, strObjectName, bOPCUAVisible)

Sets the OPC UA visible flag of the specified object.

Transfer parameters	Type	Description
prjDst	Project	Target project
strNetworkName	String	The name of the network in which the object is located
strObjectName	String	Object name
bOPCUAVisible	Boolean	Value of the OPC UA visible flag for the specified object

Return parameters	Type	Description
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.classes as classes
import sigmatek.lasal.batch as batch

strPath = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettings\OPCSettings.lcp"
strPathLib = classes.TheApplication().Path + r"..\Data\Scripting\OPCSettingsLib\OPCSettingsLib
prjTest = batch.LoadProject(strPathLib)

batch.SaveAs(prjTest,strPath)

if (prjTest == None) :
        print "Loading test project failed."
        raise Exception

#-------------------------------------------------------------------------------------------------
#
batch.SetPrjOPCUAEnable(prjTest,True);
batch.SetVisualizedFlagOPCUA(prjTest, "Network0", "Class01", True);
batch.SetOPCUAPhysicObjectServer(prjTest, "Network0", "Class01", "physics" , "OPCUAPhysics")
batch.SetOPCUAUnitObjectServer(prjTest, "Network0", "Class01", "Units", "OPCUAUnits")
batch.SetOPCUAFolderObjectServer(prjTest, "Network0", "Class01", "Folder" , "OPCUAFolders")
batch.SetOPCUAHostnameObjectServer(prjTest, "Network0", "Class01", "Hostname" , "OPCUAHostname
batch.SetOPCUAWriteProtectedObjectServer(prjTest, "Network0", "Class01", "physics" , True)
batch.SetOPCUASettingObjectServer(prjTest, "Network0", "Class01", "ALL", "ALLPHYSICS" ,"ALLUNI
#
# End
batch.Save(prjTest)
batch.CloseProject(prjTest)
del prjTest
```

3.12.3.11 Start

Start(prjDst, strConnection="")

Starts a project loaded to the PLC.

Transfer parameters	Type	Description
prjDst	Project	The target project
strConnection	String	Either the connection string, or the name from the telephone book. If this parameter is empty, the project setting is used. Default = ""

Return parameters	Type	Description
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

#Use loaded GUI-Project
prjTest = classes.Project()

strIp = "10.10.116.184"

batch.ClearRamImage(strIp)

batch.Download(prjTest, strIp, True)
batch.Download(prjTest, strIp, False)

batch.Start(prjTest, strIp)

batch.Stop(prjTest, strIp)

batch.SaveProjectOnPLC(prjTest, strIp)

batch.CloseModemConnection()

batch.DownloadOS(strIp, "DummyOS")

del prjTest
```

3.12.3.11 StartMatlabSimulinkImport

StartMatlabSimulinkImport(prjDst, strXmlFilePath)

Starts the Matlab Simulink import.

Transfer parameters	Type	Description
prjDst	Project	The target project
strXmlFilePath	String	Path of the XML file that contains the import information
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as sig

def TestMatlabSimulinkImport():
        # Init paths
        strDir = sig.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestMatlabSimulinkImport"
        strXml = "\\Source\\Xml\\import.xml"

        # Load test project
        prj = batch.LoadProject (strDir + strPrj)
        if prj == None :
                print "Loading project \"%s\" failed." % (strDir + strPrj)
                raise batch.BatchError

        # Copy with CheckRevision = True and RCD = False
        if batch.StartMatlabSimulinkImport(prj, strDir + strPrj + strXml) == False :
                print "Matlab Simulink import failed."

        if batch.ExistsClass(prj, "GeneratedCls") == False:
                print "Class \"GeneratedCls\" was not created."
                raise batch.BatchError

        # Close the project and the library
        batch.CloseProject(prj)
```

3.12.3.11 StartRecordingImportFiles

StartRecordingImportFiles(prjDst)

Starts recording all imported files (except generated files such as LOB, BIN, ST from NW). Calls the member function of the project. If an import (e.g. CopyClass) is performed afterwards, the import sequence is logged. If the import is finished, the StopRecordingImportFiles() command will stop the recording and the result will be created as a file in XML format. Here: for each imported file there is one line with 3 values each:

```
<Files>
        <File Dest="D:\todelete\DEVSW-1760\Class\Ram\Ram.ICO" Src="C:\Standard_Projects\LASAL2
        <File Dest="D:\todelete\DEVSW-1760\Class\Ram\Ram.h" Src="C:\Standard_Projects\LASAL2\R
        <File Dest="D:\todelete\DEVSW-1760\Class\Ram\Ram.st" Src="C:\Standard_Projects\LASAL2\
</Files>
```

Dest indicates the location **where** the file was imported. Src indicates the location **where** the file comes **from**. And Cnt indicates **how many times** the same file was imported.

Transfer parameters	Type	Description
prjDst	Project	The target project
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch

def TestCopyClass() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty target test project
  strDst = os.path.join(strCurDir,"CopyClassTest")
  # Delete if already exist
  shutil.rmtree(strDst, True)
  # Create
  prjDst = batch.CreateProject(strDst)
  if prjDst==None :
    raise batch.BatchError
  print "Created empty destination project %s" % prjDst.Filename

  # Send the StartRecording-command
  StartRecordingImportFiles(prjDst)

  # Open the import project
  strSrc = os.path.join(os.path.dirname(sys.argv[0]), r"..\..\Runtime\Library\OS Interface")
  prjSrc = batch.LoadProject(strSrc)
  if prjSrc==None :
    raise batch.BatchError
  print "Loaded source project %s" % prjSrc.Filename

  bSuccess = True

  # Copy class
  strClass = "_Grafix"
  if batch.CopyClass(prjDst, prjSrc, strClass)==False :
    print "Copy class %s failed." % strClass
    bSuccess = False

  # Close the Src project again
  del prjSrc
```

```
# Send the StopRecording-command
StopRecordingImportFiles(prjDst, "D:\\Temp\\MyImportedFiles.tyt"
```

3.12.3.11 Stop

Stop(prjDst, strConnection="")

Stops a running PLC program.

Transfer parameters	Type	Description
prjDst	Project	The target project
strConnection	String	Either the connection string, or the name from the telephone book. If this parameter is empty, the project setting is used. Default = ""
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes

#Use loaded GUI-Project
prjTest = classes.Project()

strIp = "10.10.116.184"

batch.ClearRamImage(strIp)

batch.Download(prjTest, strIp, True)
batch.Download(prjTest, strIp, False)

batch.Start(prjTest, strIp)

batch.Stop(prjTest, strIp)

batch.SaveProjectOnPLC(prjTest, strIp)

batch.CloseModemConnection()

batch.DownloadOS(strIp, "DummyOS")

del prjTest
```

3.12.3.11\StopRecordingImportFiles

StopRecordingImportFiles(prjDst, Filename="")

Stops the recording that was started with StartRecordingImportFiles(), and writes the information to the file specified in the *strFilename* parameter. **Otherwise the recording will be stopped automatically when the script ends**, and the information **will be deleted**. In the output file there are always **filenames with absolute path** to the projects. The command calls the member function of the project.

The result is a file in a special xml format: for each imported file there is a separate line with 3 values each.

```
<Files>
        <File Dest="D:\todelete\DEVSW-1760\Class\Ram\Ram.ICO" Src="C:\Standard_Projects\LASA
        <File Dest="D:\todelete\DEVSW-1760\Class\Ram\Ram.h" Src="C:\Standard_Projects\LASAL2
        <File Dest="D:\todelete\DEVSW-1760\Class\Ram\Ram.st" Src="C:\Standard_Projects\LASAL
</Files>
```

Dest indicates the location **where** the file was imported. Src indicates the location **where** the file comes **from**. And Cnt indicates **how many times** the same file was imported.

Transfer parameters	Type	Description
prjDst	Project	The target project
Filename	String	The output file
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example:

```
import os
import os.path
import shutil
import sys
import filecmp
import sigmatek.lasal.batch as batch

def TestCopyClass() :

  strCurDir = os.path.dirname(sys.argv[0])

  # Path to the empty target test project
  strDst = os.path.join(strCurDir,"CopyClassTest")
```

```
# Delete if already exist
shutil.rmtree(strDst, True)
# Create
prjDst = batch.CreateProject(strDst)
if prjDst==None :
  raise batch.BatchError
print "Created empty destination project %s" % prjDst.Filename

# Send the StartRecording-command
StartRecordingImportFiles(prjDst)

# Open the import project
strSrc = os.path.join(os.path.dirname(sys.argv[0]), r"..\..\Runtime\Library\OS Interface")
prjSrc = batch.LoadProject(strSrc)
if prjSrc==None :
  raise batch.BatchError
print "Loaded source project %s" % prjSrc.Filename

bSuccess = True

# Copy class
strClass = "_Grafix"
if batch.CopyClass(prjDst, prjSrc, strClass)==False :
  print "Copy class %s failed." % strClass
  bSuccess = False

# Close the Src project again
del prjSrc

# Send the StopRecording-command
StopRecordingImportFiles(prjDst, "D:\\Temp\\MyImportedFiles.tyt"
```

As a result, in the above example, the export file is written to the "`D:\Temp`" directory under the name `MyImportedFiles.tyt` in xml format.

3.12.3.11 UpdateAllUsedClassesOfLib

UpdateAllUsedClassesOfLib(prjDst, prjSrc, importMode, bCheckRevision=True, bUseReducedClientDependency=False)

Updates all used classes in the project from the library.

Transfer parameters	Type	Description
prjDst	Project	The target project
prjSrc	Project	The source project
importMode	Number	Indicates how the classes should be imported 0 COPY

		1 LINK
bCheckRevision	Boolean	**True** = Check revision to overwrite classes. Default = **True**
bUseReducedClientDependency	Boolean	If **True**, uses Reduced Client Dependency.If this function is activated, only classes are imported that are directly linked to the selected class (no more classes than the 2nd dependency). Default = **False**
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import os
def TestPilzPatent():

        print "TestPilzPatent:"

        # Init paths
        strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
        strPrj = "\\TestUpdateAllUsedClasses_PilzPatent"

        strAppPath = classes.TheApplication().Path

        strHardwareLib = os.path.join(strAppPath , r"..\Runtime\Library\Hardware")

        # Check if the path of the hardware library exists; only then load the project
        if os.path.exists(strHardwareLib) == False :
                print "Path of library 'Hardware' doesn't exist: %s" % strHardwareLib
                raise batch.BatchError

        # Load test project
        prj = batch.LoadProject (strDir + strPrj)
        if prj == None :
                print "Loading project \"%s\" failed." % (strDir + strPrj)
                raise batch.BatchError

        # Load test library
        print "Load Library 'Hardware' (%s)" % (strHardwareLib)
        lib = batch.LoadProject (strHardwareLib)
        if lib == None :
                print "Loading library \"%s\" failed." % (strHardwareLib)
                raise batch.BatchError

        if batch.UpdateAllUsedClassesOfLib(prj, lib, 0, True, False) == False:
                raise batch.BatchError

        if batch.ExistsClass(prj, "HWControl") == False :
```

```
            print "Class 'HWControl' doesn't exist."
            raise batch.BatchError

    if batch.ExistsClass(prj, "SafetyManager") == False :
            print "Class 'SafetyManager' doesn't exist."
            raise batch.BatchError

    batch.Save(prj)

    batch.CloseProject(prj)
    batch.CloseProject(lib)

    print "TestPilzPatent successful"
```

3.12.3.11 UpdatePrjIncludesAndLoader

UpdatePrjIncludesAndLoader(prjDst)

Copies the Includes and the loader files of the open application to the project directory when this option is enabled.

Transfer parameters	Type	Description
prjDst	Project	The target project
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as cls

# get the current project
myPrj = cls.TheApplication().Project
if (myPrj == None) :
        raise Exception

if(batch.UpdatePrjIncludesAndLoader(myPrj) == False) :
        raise Exception

batch.Save(myPrj)                                                    # save project to
```

3.12.3.11 UploadFile

UploadFile(prjDst, strConnection, strFileSrc, strFileDest)

Loads the desired file from the PLC to host.

Transfer parameters	Type	Description
prjDst		Project from whose project properties the online connection is read (the connection does not have to be entered). If nothing is entered here, the connection specified under strConnection is applied
strConnection		Name for the online connection from the telephone book. If nothing in entered here, the project settings are applied (s. prjDst). Alternatively, the IP address of the PLC can be entered here directly in the following format: `batch.Download(myPrj, „TCPIP:xxx.xxx.xxx.xxx“)`
strFileSrc		The path and name of the file to upload from the target system
strFileDest		The path and name of the file on the PC
Return parameters	**Type**	**Description**
	Boolean	**True**: successful **False** or **Exception**: failed

Example

```
import sigmatek.lasal.batch as batch
import sigmatek.lasal.classes as classes
import sys
import os.path
import filecmp

def TestConnectionFilesPlc() :

  strCurDir = os.path.dirname(sys.argv[0])
  strConnection = sys.argv[1]

  # Initialize paths
  strDir = classes.TheApplication().Path + "..\\Data\\Scripting"
  strFileD = "\\DownloadFile.txt"
  strFilePathD = (strDir + strFileD)
  strFileU = "\\UploadFile.txt"
  strFilePathU = (strDir + strFileU)
  strFilePlc = "C:\\unittest.txt"

  bSuccess = True
  # Delete existing file
  if os.path.exists(strFilePathU) != False :
    os.remove(strFilePathU)
```

```
# Download file
if batch.DownloadFile(None, strConnection, strFilePathD, strFilePlc) == False :
  bSuccess = False
  print "Download file failed"

# Upload file
if batch.UploadFile(None, strConnection, strFilePlc, strFilePathU) == False :
  bSuccess = False
  print "Upload file failed"

# Compare files
if filecmp.cmp(strFilePathD, strFilePathU) == False :
  bSuccess = False
  print "file compare failed"

if bSuccess == False :
  raise batch.BatchError

print "Test successfull"
```

3.12.4 Calling Scripts in LASAL CLASS 2

3.12.4.1 Calling from the Console

Command Line:

```
>lasal2 /Script:Build.py
```

At transfer of a parameter to the interpreter, the whole parameter has to be under quotation marks:

```
C:\>lasal2 /script:"C:\Scripts\MyTestScript.py 1 2 3"
```

3.12.4.2 Calling from LASAL Class 2

Under the menu item **Tools, Scripts** can be selected as to whether the actual script loaded in the editor (button combination: **Strg + Shift + F7**) or a script assigned using a file dialog should be run.

3.12.4.3 Calling a Python Script in LASAL Class 2 from another Python Script

Here, it should be noted that the first parameter is skipped. A call appears (in part) as follows:

```
import os
os.spawnve(os.P_WAIT, r"C:\Programme\Sigmatek\LASAL\Class2\Bin\lasal2.exe ", ["lasal2","/script
```

Or if parameters are entered additionally:

```
import os
os.spawnve(os.P_WAIT, r"C:\Programme\Sigmatek\LASAL\Class2\Bin\lasal2.exe ", ["lasal2","/scr
```

3.12.4.4 Python Script Window

In the Python Script window of LASAL CLASS 2, start the script file using:

```
>>>execfile(Build.py)
```

3.12.5 LASAL CLASS 2 - Python Classes

An additional property of Python is the object orientation. LASAL CLASS 2 supports python and provides some classes for it (called runtime objects). Depending on the type of class (from which the object comes), various parameters can be tested and/or set.

The "classes" module only has to be imported, when information about the application or an empty project is required.

3.12.5.1 Project

To make the following project settings available in the script, the following command has to be used:

```
#importiert Modul 'classes'
import sigmatek.lasal.classes as classes
```

Name

The name of the project.

Filename

The full assignment of the .lcp file.

Path

The path to the project directory.

Comment

Are written about the project (commentary).

CompilerOptions

Compiler settings (see project properties).

DirectoryOptions

The assignment of the project directory structure (see project properties).

ConnectionName

The name from the address book for the online connection.

LibraryPath

Directory to the project dependant library.

IncludePath

The path t the project dependant Include directories.

An object of this class is obtained using the commands:

CreateProject (...) or **LoadProject (...)**

in the **'sigmatek.lasal.batch'** module or through the 'project' parameter of the class application, which is supplied by the actual project from LASAL CLASS 2.

UseFPU (True/False)

True means that the mathematic coprocessor is used in the target.

SolutionType (Large/Small)

Defines the object network types of the project.

WriteServerListOnSave (True/False)

If this parameter is set to **true**, the server list is also saved automatically.

3.12.5.2 Application

To make the following application settings available in the script, the following commands have to be used:

```
#imports 'classes' module
import sigmatek.lasal.classes as classes

#holds objects to the LASAL CLASS 2 application
AppObj = classes.TheApplication()

#instructs LASAL CLASS 2 to remain opened after script ending
AppObj.ExitAppAfterEnd = False
```

ExitAppAfterEnd

Is a boolean flag (**True**, **False**) and instructs LASAL CLASS 2 to close after ending of the script process or not.

Path

The director to the LASAL CLASS 2 installation.

Filename

The complete name of the LASAL CLASS 2 program, including the path.

Name

The name of the LASAL CLASS 2 program.

Project

The actual project displayed (filled project tree).

LibraryPath

The globally set path to the library.

IncludePath

The globally set path to the Include directory.

Visible

Decides whether the application itself is visible.

EmbeddedStringServerFix

Enables the sorting fix for the CH_REDIRECT_EMB_TAB to be used in earlier compiler versions than C63.

An object of this class is obtained through the function call, TheApplication(), in the Module ‚sigmatek.lasal.classes'.

3.12.6 Additional Commands

Listed below are several additional commands, which are integrated into the LASAL CLASS 2 **"batch"** module. _**internal** is included so than there is no confusion or conflict with other instructions.

3.12.6.1 _internal.msgbox(strText)

Displays a modal message box on the screen (OK-Button only).

strText

The text displayed in the message box.

3.12.6.2 _internal.write(strText)

Writes a message in the Python output window.

strText

The text written in the Python output window.

3.12.6.3 _internal.logInfo(strText)

Writes an information message to the Log file.

strText

The text written to the Log file.

3.12.6.4 _internal.logWarn(strText)

Writes a warning to the Log file.

strText

The text written to the Log file.

3.12.6.5 _internal.logError(strText)

Writes an error message in the Log file.

strText

The text written in the Log file

3.12.6.6 _internal.GetApplicationPath()

Returns the path of the application.

3.12.6.7 _internal.inputBox(strText)

Shows an input box with the defined text as the title. Returns the value entered by the user as a string. Caution: Unlike **input(strText)**, numbers are not interpreted as such and must be converted when necessary!

strText

Text that should be shown in the box as the title.

Return

The entered value as a string.

3.12.7 Scripting - Example

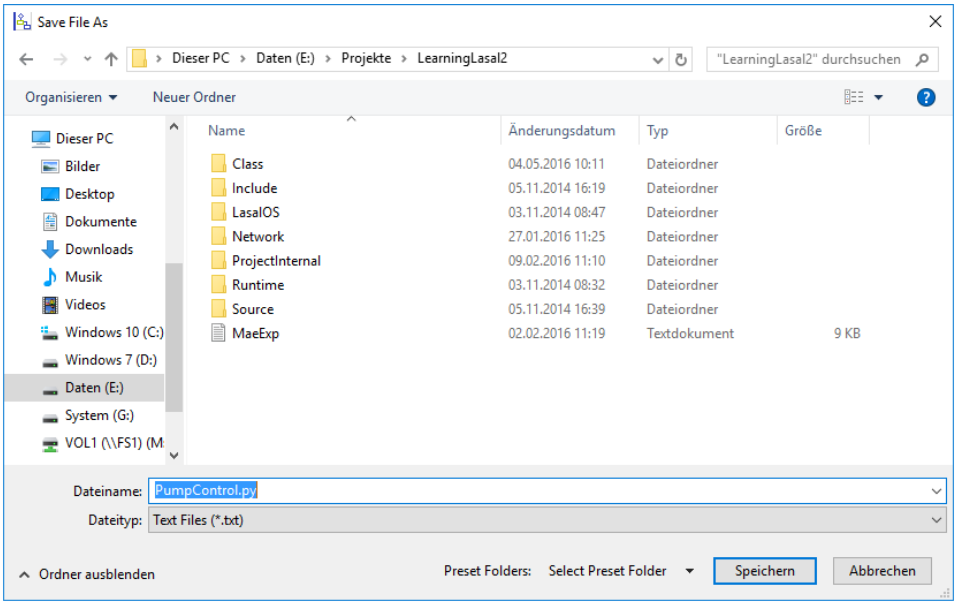


With help from a script file, the **pump work** example should be generated automatically.

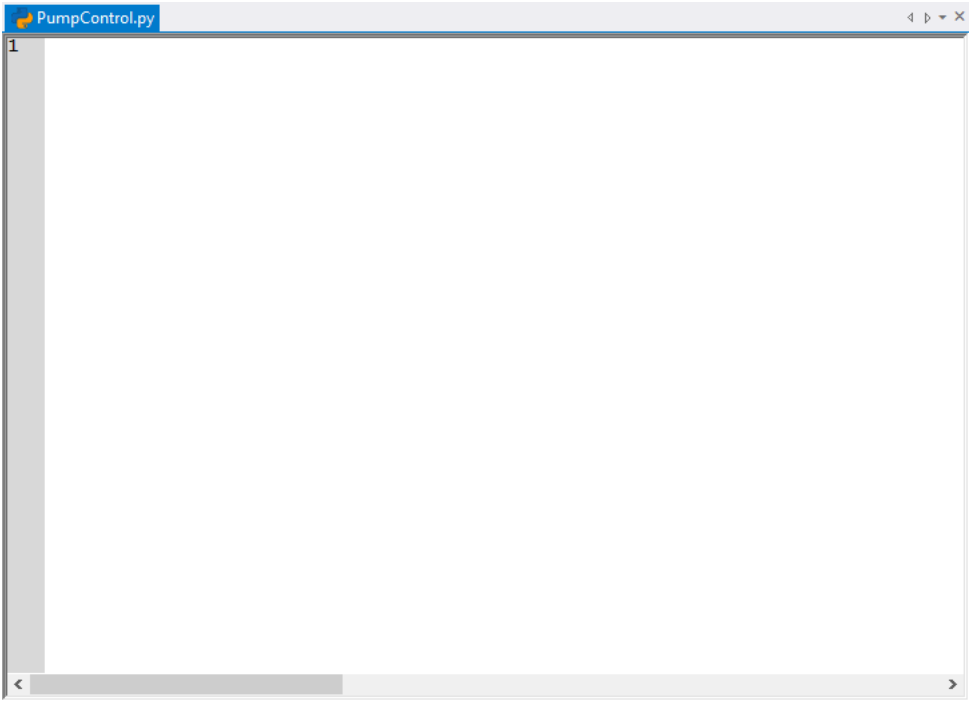
To create a script file, select **File, New File...** in the menu bar. Open the **Save File As window** with **File, Save File As** in the menu and assign a file name (**PumpControl.py**) and click on save.



Be sure the file name contains the ending **.py**! A running script process can be aborted with Ctrl+Break or the Cancel screen button.



The following window then appears in LASAL CLASS 2:



To use the SIGMATEK script instructions, the previously mentioned script module must be linked.

It is not possible to create new classes or change existing ones using script since additional source code cannot be inserted into the class.

To create the pump work, existing classed must be accessed. The classes required for this example are available in the **DiasMaster.lcp** library and the **PumpControl** project created earlier.

Both projects will now be linked in the script file under the **LoadProject** instruction:

PumpControl.py*

```
1
2  #Make the script commands of Lasal available in this script
3  from sigmatek.lasal import batch
4
5  #Import libraries which are needed
6  prjDiasMaster = batch.LoadProject("C:/Lasal2/Runtime/Library/DiasMaster")
7  prjHysterese  = batch.LoadProject("C:/Sample/PumpControl")
8
```

Because the **LoadProject** instruction comes from the SIGMATEK module, **batch** is added before the instruction.

Both projects are assigned to the variables **prjDiasMaster** and **prjHysterese**. From now on only these variables are used. The # symbol identifies the line of text following it as commentary.

To create a new project, the **CreateProject** project instruction must be used. All required classes are then copied with the **CopyClass** instruction.

PumpControl.py*

```
1
2  #Make the script commands of Lasal available in this script
3  from sigmatek.lasal import batch
4
5  #Import libraries which are needed
6  prjDiasMaster = batch.LoadProject("C:/Lasal2/Runtime/Library/DiasMaster")
7  prjHysterese  = batch.LoadProject("C:/Sample/PumpControl")
8
9  #Create a project at a new place
10 prjScriptPmp = batch.CreateProject("D:/Sample/ScriptPmp")
11
12 #Copy all needed classes to the target project
13 batch.CopyClass(prjScriptPmp, prjDiasMaster, "CAM123_IM")
14 batch.CopyClass(prjScriptPmp, prjDiasMaster, "CTO163_IM")
15 batch.CopyClass(prjScriptPmp, prjDiasMaster, "DiasMasterC")
16 batch.CopyClass(prjScriptPmp, prjHysterese, "Hysterese")
17
```

After the required classes have been successfully imported, the objects for the classes must be created with the **CreateObject** instruction. Since objects have to be placed in an object network, a new network must be generated using the **CreateNetwork** instruction.

PumpControl.py*

```
#Make the script commands of Lasal available in this script
from sigmatek.lasal import batch

#Import libraries which are needed
prjDiasMaster = batch.LoadProject("C:/Lasal2/Runtime/Library/DiasMaster")
prjHysterese  = batch.LoadProject("C:/Sample/PumpControl")

#Create a project at a new place
prjScriptPmp = batch.CreateProject("D:/Sample/ScriptPmp")

#Copy all needed classes to the target project
batch.CopyClass(prjScriptPmp, prjDiasMaster, "CAM123_IM")
batch.CopyClass(prjScriptPmp, prjDiasMaster, "CTO163_IM")
batch.CopyClass(prjScriptPmp, prjDiasMaster, "DiasMasterC")
batch.CopyClass(prjScriptPmp, prjHysterese, "Hysterese")

#Create the objects, after a network was created
batch.CreateNetwork(prjScriptPmp, "Pumpcontrol")
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "DiasMasterC", "objDiasMasterC", 150, 200)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "CAM123_IM", "objCAM123_IM", 550,200)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "CTO163_IM", "objCTO163_IM", 950, 400)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "Hysterese", "objHysterese", 1350, 250)
```

Because the **DiasMasterC** has a **Realtime task** and the **Hysterese** has a **Cyclic task**, it is necessary to set a time for both objects. This is done using the instruction **SetTaskTime**. The objects also need initial values, which are set with the **SetInitValue** instruction.

PumpControl.py*

```
#Create the objects, after a network was created
batch.CreateNetwork(prjScriptPmp, "Pumpcontrol")
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "DiasMasterC", "objDiasMasterC", 150, 200)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "CAM123_IM", "objCAM123_IM", 550,200)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "CTO163_IM", "objCTO163_IM", 950, 400)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "Hysterese", "objHysterese", 1350, 250)

# Set initialize values and task times
batch.SetTaskTime(prjScriptPmp, "Pumpcontrol", "objHysterese", "cyclicwork", "100")
batch.SetTaskTime(prjScriptPmp, "Pumpcontrol", "objDiasMasterC", "realtime", "2")

batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objHysterese", "MaxValue", "800")
batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objHysterese", "MinValue", "200")
batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objCTO163_IM", "Place", "1")
batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objCAM123_IM", "Place", "2")

#An example how to initialize several channels of a object with the same value
list Clients = ["AI1-Scale", "AI2_Scale", "AI3_Scale",
                "AI4_Scale", "AI5_Scale", "AI6_Scale",
                "AI7_Scale", "AI8_Scale", "AO1_Scale",
                "AO2_Scale", "AO3_Scale", "AO4_Scale"]

for strClient in listClients :
  batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objCAM123_IM", strClient, "1000")
```

Several clients of the **objCAM123_IM** object require the same initial value. A for-loop is used to accomplish this. In the next Step, the connection between the objects is made (**CreateConnection**). The project is then compiled (**Compile**) and saved (**Save**). The libraries are then closed, as they are no longer necessary (**CloseProject**).

PumpControl.py*

```
batch.CopyClass(prjScriptPmp, prjDiasMaster, "DiasMasterC")
batch.CopyClass(prjScriptPmp, prjHysterese, "Hysterese")

#Create the objects, after a network was created
batch.CreateNetwork(prjScriptPmp, "Pumpcontrol")
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "DiasMasterC", "objDiasMasterC", 150, 200)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "CAM123_IM", "objCAM123_IM", 550,200)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "CTO163_IM", "objCTO163_IM", 950, 400)
batch.CreateObject(prjScriptPmp, "Pumpcontrol", "Hysterese", "objHysterese", 1350, 250)

# Set initialize values and task times
batch.SetTaskTime(prjScriptPmp, "Pumpcontrol", "objHysterese", "cyclicwork", "100")
batch.SetTaskTime(prjScriptPmp, "Pumpcontrol", "objDiasMasterC", "realtime", "2")

batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objHysterese", "MaxValue", "800")
batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objHysterese", "MinValue", "200")
batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objCTO163_IM", "Place", "1")
batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objCAM123_IM", "Place", "2")

#An example how to initialize several channels of a object with the same value
list Clients = ["AI1-Scale", "AI2_Scale", "AI3_Scale",
                "AI4_Scale", "AI5_Scale", "AI6_Scale",
                "AI7_Scale", "AI8_Scale", "AO1_Scale",
                "AO2_Scale", "AO3_Scale", "AO4_Scale"]

for strClient in listClients :
  batch.SetInitValue(prjScriptPmp, "Pumpcontrol", "objCAM123_IM", strClient, "1000")

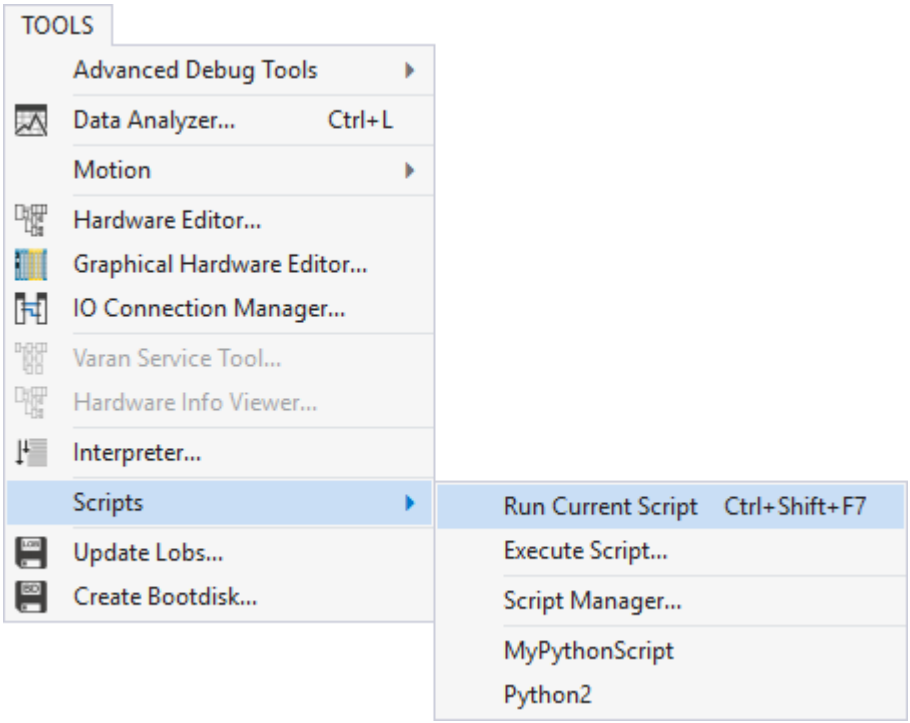
# Build connections
batch.CreateConnection(prjScriptPmp, "Pumpcontrol", "objCAM123_IM", "MasterConnect", "Pumpcontrol", "objDiasMasterC", "Control")
batch.CreateConnection(prjScriptPmp, "Pumpcontrol", "objCTO163_IM", "MasterConnect", "Pumpcontrol", "objDiasMasterC", "Control")
batch.CreateConnection(prjScriptPmp, "Pumpcontrol", "objHysterese", "ActualValue",   "Pumpcontrol", "objCAM123_IM",   "AI1")
batch.CreateConnection(prjScriptPmp, "Pumpcontrol", "objHysterese", "Event",    "Pumpcontrol", "objCTO163_IM",    "Output1")
batch.CreateConnection(prjScriptPmp, "Pumpcontrol", "objHysterese", "Alarm",    "Pumpcontrol", "objCTO163_IM",    "Output2")

# Save the project before continue
batch.Save(prjScriptPmp)

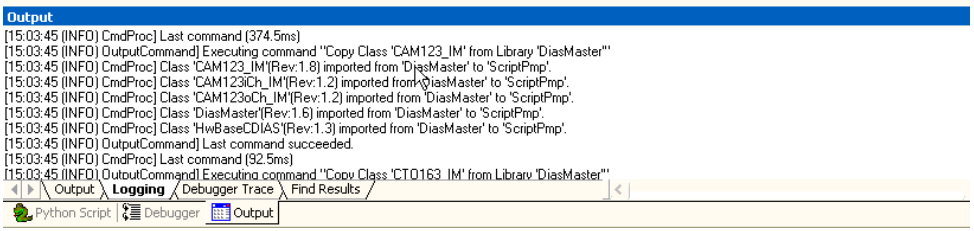
# Compile the whole project
batch.Compile(prjScriptPmp)

#Close libraries which are not longer needed
batch.CloseProject(prjDiasMaster)
batch.CloseProject(prjHysterese)
```

In the menu bar select **Tools, Scripts, Run Current Script** to execute the script file.



The processing of the script instructions can be seen in the Output window.





It is also possible to execute the script file in LASAL CLASS 2 externally. The following instruction must be processed in the command line of the LASAL CLASS 2 directory:

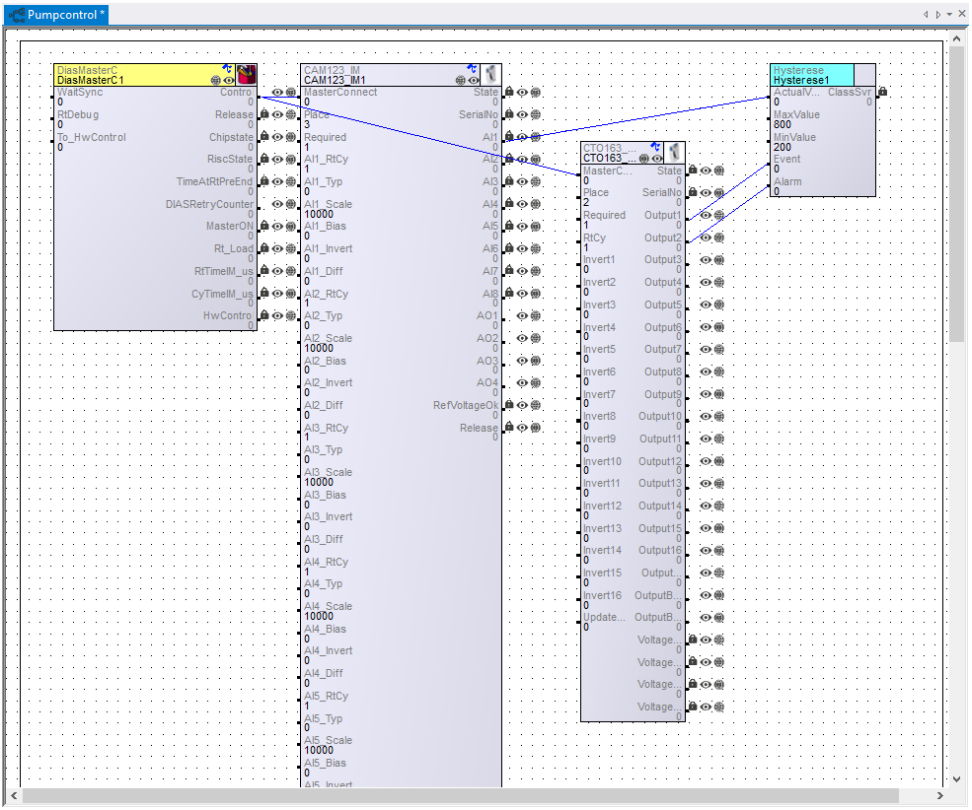
Lasal2 /script:C:\sample\PumpControl.py

Here also parameters can be transferred, which can be called from the script. If a parameter with space has to be transferred, it easily can be enclosed with apostrophe:

Lasal2 /script:"C:\ArgTest.py 'Sending of' 3 Parameter"

At process abort because of a not trapped Exception the error code 102 is returned in the command line.

The following screen should appear when the project is loaded and the object network, **PumpControl**, is opened:



3.12.8 Error Handling

Error Handling

ExampleErrorHandling.py

```
import sigmatek.lasal.batch as batch

#No exception if an error occurred.
batch.SetExceptionOnError(False)

#Load a project that does noct exist
PrjTest = batch.LoadProject(r"C:\Temp\PrjNotExist")

if PrjTest == None :
  print "ERROR: Cannot load project!"
  PrjTest = batch.CreateProject(r"C:\Temp\NewProject")

print "Create a new network: "
#create a new network
if batch.CreateNetwork(PrjTest, "NewNetwork") == False :
  print "ERROR: Cannot execute command"

print "Create a new object:"
#try to create an object of a not existing class
if batch.CreateObject(PrjTest, "NewNetwork", "NotExistingClass", "NewObject", 150, 150) == False :
  print "ERROR: Cannot execute command"

print "End of command sequence."

print "Script finish."
```

Exception Handling

ExampleExeptionHandling.py

```
import sigmatek.lasal.batch as batch

#Throw an exception if something went wrong
batch.SetExceptionOnError(True)

try:
  #Load a project that does not exist
  PrjTest = batch.LoadProject(r"C:\Temp\PrjNotExist")
except batch.BatchError :
  print "ERROR: Cannot load project!"
  PrjTest = batch.CreateProject(r"C:\Temp\NewProject")

try:
  print "Create a new network:"
  #create a new network
  batch.CreateNetwork(PrjTest, "NewNetwork")
  print "Create a new object:"
  #try to create an object of a not existing class
  batch.CreateObject(PrjTest, "NewNetwork", "NotExistingClass", "NewObject", 150, 150)
  print "End of command sequence."

except batch.BatchError :
  print "ERROR: Cannot execute command"

print "Script finish"
```

3.12.9 Setting of Project Settings

```
import sigmatek.lasal.classes as classes

#Loadalllibraries
prjTest.Loadalllibraries = True
prjTest.Loadalllibraries = False

#UseUnitSystem
```

```
prjTest.UseUnitSystem = True
prjTest.UseUnitSystem = False

#IncludePath
prjTest.IncludePath = ["Heute", "Morgen"]

#LibraryPath
prjTest.LibraryPath = ["Heute", "Morgen"]

#BackupIncludesAndLoader
prjTest.BackupIncludesAndLoader = True
prjTest.BackupIncludesAndLoader = False

#IgnoreatCleanup
str = "C:\\"
prjTest.IgnoreatCleanup = str

#ReencryptProjectOnClose
prjTest.ReencryptProjectOnClose = True
prjTest.ReencryptProjectOnClose = False

#EnableOPCUA
prjTest.EnableOPCUA = True
prjTest.EnableOPCUA = False

#EnableInitValueForOutputParams
prjTest.EnableInitValueForOutputParams = True
prjTest.EnableInitValueForOutputParams = False

#Comment
prjTest.Comment = "This is a Testproject."
```

3.12.9.1 Setting the Network Options

For more information, see the chapter Properties of a project.

```
import sigmatek.lasal.classes as classes

#CodeGenVersion
prjTest.CodeGenVersion = 2
prjTest.CodeGenVersion = -1 # Set to latest version

#SolutionType
prjTest.SolutionType = classes.SolutionType.Small
prjTest.SolutionType = classes.SolutionType.Large

#RestrictedDefineUsage
prjTest.RestrictedDefineUsage = True
prjTest.RestrictedDefineUsage = False

#WarnNotConnectedClient
prjTest.WarnNotConnectedClient = True
```

```
prjTest.WarnNotConnectedClient = False

#ErrorIfDestinationNotFound
prjTest.ErrorIfDestinationNotFound = True
prjTest.ErrorIfDestinationNotFound = False

#ErrorIfComponentNotFound
prjTest.ErrorIfComponentNotFound = True
prjTest.ErrorIfComponentNotFound = False
```

3.12.9.2 Setting the Server list Options

```
import sigmatek.lasal.classes as classes

#WriteServerListOnSave
prjTest.WriteServerListOnSave = True
prjTest.WriteServerListOnSave = False

#WriteGlobVarList
prjTest.WriteGlobVarList = True
prjTest.WriteGlobVarList = Fasle

#ExportSymTab
prjTest.ExportSymTab = True
prjTest.ExportSymTab = False

#WriteVisualMembers
prjTest.WriteVisualMembers = True
prjTest.WriteVisualMembers = False
```

3.12.9.3 Setting the Target Options

```
import sigmatek.lasal.classes as classes

#UseFPU
prjTest.UseFPU = True
prjTest.UseFPU = False

#Processor
prjTest.Processor = "ARM"
prjTest.Processor = "X86"

#UseSignedCharOnArm
prjTest.UseSignedCharOnArm = True
prjTest.UseSignedCharOnArm = False

#CompileAllArchs
prjTest.CompileAllArchs = True
prjTest.CompileAllArchs = False
```

3.12.9.4 Setting the Compiler Settings

```
import sigmatek.lasal.classes as classes

#CompilerVersion
prjTest.CompilerOptions.CompilerVersion = 25
prjTest.CompilerOptions.CompilerVersion = 27

#StoreRestoreFlgError
prjTest.CompilerOptions.StoreRestoreFlgError = True
prjTest.CompilerOptions.StoreRestoreFlgError = False

#VariableNeverUsed
prjTest.CompilerOptions.VariableNeverUsed = True
prjTest.CompilerOptions.VariableNeverUsed = False

#ValueNeverUsed
prjTest.CompilerOptions.ValueNeverUsed = True
prjTest.CompilerOptions.ValueNeverUsed = False

#LabelNeverUsed
prjTest.CompilerOptions.LabelNeverUsed = True
prjTest.CompilerOptions.LabelNeverUsed = False

#ParameterNeverUsed
prjTest.CompilerOptions.ParameterNeverUsed = True
prjTest.CompilerOptions.ParameterNeverUsed = False

#VariableUnitialized
prjTest.CompilerOptions.VariableUnitialized = True
prjTest.CompilerOptions.VariableUnitialized = False

#RetValueUnitialized
prjTest.CompilerOptions.RetValueUnitialized = True
prjTest.CompilerOptions.RetValueUnitialized = False

#PrivateDataAccess
prjTest.CompilerOptions.PrivateDataAccess = True
prjTest.CompilerOptions.PrivateDataAccess = False

#SemicolonMissing
prjTest.CompilerOptions.SemicolonMissing = True
prjTest.CompilerOptions.SemicolonMissing = False

#ConditionConstant
prjTest.CompilerOptions.ConditionConstant = True
prjTest.CompilerOptions.ConditionConstant = False

#ExactTypeChecking
prjTest.CompilerOptions.ExactTypeChecking = True
prjTest.CompilerOptions.ExactTypeChecking = False

#IdenticalRedefinition
prjTest.CompilerOptions.IdenticalRedefinition = True
prjTest.CompilerOptions.IdenticalRedefinition = False
```

```
#CheckConnectionTypes
prjTest.CompilerOptions.CheckConnectionTypes = True
prjTest.CompilerOptions.CheckConnectionTypes = False

#CompileCFiles
prjTest.CompilerOptions.CompileCFiles = True
prjTest.CompilerOptions.CompileCFiles = classes.CompileCFiles.True
prjTest.CompilerOptions.CompileCFiles = False
prjTest.CompilerOptions.CompileCFiles = classes.CompileCFiles.False
prjTest.CompilerOptions.CompileCFiles = classes.CompileCFiles.DetailDebugInfo

#CPPCompilerPath
strRef = "C:\\"
prjTest.CompilerOptions.CPPCompilerPath = strRef
prjTest.CompilerOptions.CPPCompilerPath = ""

#Alignment
prjTest.CompilerOptions.Alignment = classes.Alignment.Byte_1
prjTest.CompilerOptions.Alignment = classes.Alignment.Byte_2
prjTest.CompilerOptions.Alignment = classes.Alignment.Byte_4

#ReducedClientDependency
prjTest.CompilerOptions.ReducedClientDependency = True
prjTest.CompilerOptions.ReducedClientDependency = False

#MaxModuleErrorCount
prjTest.CompilerOptions.MaxModuleErrorCount = 50
prjTest.CompilerOptions.MaxModuleErrorCount = 100

#CheckArrayAccess
prjTest.CompilerOptions.CheckArrayAccess = True
prjTest.CompilerOptions.CheckArrayAccess = False

#CheckStackSize
prjTest.CompilerOptions.CheckStackSize = 5000
prjTest.CompilerOptions.CheckStackSize = 4000

#ShowConnectionManagerErrorReport
prjTest.CompilerOptions.ShowConnectionManagerErrorReport = True
prjTest.CompilerOptions.ShowConnectionManagerErrorReport = False

#UseLatestCompilerVersion
prjTest.CompilerOptions.UseLatestCompilerVersion = True
prjTest.CompilerOptions.UseLatestCompilerVersion = False

#ShowCompilerVersionWarning
prjTest.CompilerOptions.ShowCompilerVersionWarning = True
prjTest.CompilerOptions.ShowCompilerVersionWarning = False

#CompileNetworkFiles
prjTest.CompilerOptions.CompileNetworkFiles = True
prjTest.CompilerOptions.CompileNetworkFiles = False

#CompatibilityMode                                          // Only available in the Compiler
```

```
prjTest.CompilerOptions.CompatibilityMode = True            // Only available in the Compiler
prjTest.CompilerOptions.CompatibilityMode = False            // Only available in the Compil

#GCCOptions
prjTest.CompilerOptions.GCCOptions = ""

#ShowConnectionFileWarnings
prjTest.CompilerOptions.ShowConnectionFileWarnings = True
prjTest.CompilerOptions.ShowConnectionFileWarnings = False

#MathLibrary
prjTest.CompilerOptions.MathLibrary = 0     # Legacy
prjTest.CompilerOptions.MathLibrary = 1     # MathBSD

#ThisCallScope
prjTest.CompilerOptions.ThisCallScope = True
prjTest.CompilerOptions.ThisCallScope = False
```

3.12.9.5 Setting the Directory Settings

```
import sigmatek.lasal.classes as classes

#Class
prjTest.DirectoryOptions.Class = ".\Class\"

#Network
prjTest.DirectoryOptions.Network = ".\Network\"

#Source
prjTest.DirectoryOptions.Source = ".\Source\"

#Include
prjTest.DirectoryOptions.Include = ".\Include\"

#Loader
prjTest.DirectoryOptions.Loader = ".\Loader\"

#PrjInternal
prjTest.DirectoryOptions.PrjInternal = ".\ProjectInternal\"

#DiasDrive
prjTest.DirectoryOptions.DiasDrive = ".\Drive\"
```

3.12.9.6 Setting the Virtual Commissioning Settings

```
import sigmatek.lasal.classes as cls

prj = cls.TheApplication().Project
advOptions = prj.AdvancedIOOptions

print advOptions.PLCPath
print advOptions.Version
```

```
print advOptions.DefaultNetwork
print advOptions.AdvancedIO

advOptions.AdvancedIO = "DupClass"

print advOptions.AdvancedIO
print advOptions.Simulation

advOptions.Simulation = "B"

print advOptions.Simulation

advOptions.PLCPath = advOptions.PLCPath +"1"

print advOptions.PLCPath
print advOptions.Manager

advOptions.Manager = "DupMgr"

print advOptions.Manager
```

3.12.10 Brief Overview of the Python Modules

(see www.python.org)

Modul	Art	Description
Basic sevices		
sys	Installed	Interpreter parameters and functions
types	Standard	Constant for all installed types
traceback	Standard	Print or retrieve objects from the Traceback stack
pickle	Standard	Convert Python objects to/from byte streams
shelve	Standard	Persistent memory of Python objects (uses pickle)
Copy	Standard	(Width and depth) copy operations (uses pickle)
marshal	Installed	Convert Python objects to/from byte streams (limited)
imp	Installed	Access to import implementation
__builtin__	Installed	Access to predefined Python designator
__main__	Installed	Name space of the module in the highest level (top-level module): Interpreter or script execution
Strings		
string	Standard	Additional String operations and constants (uses installed C-strop modules)

regex	Installed	Regular expressions for string analysis, pattern definition in EMACS style.
regsub	Standard	Regular expression replacement and splitting functions
struct	Installed	Interprets strings as packed binary data
strop	Installed	The hidden C implementation of string module operations (uses string)
re	Installed	Regular expressions for string analysis, pattern definition in Perl style. Will replace regex in the future
Various		
math	Installed	Mathematic functions, defined in C standard: sin, floor, sqrt, pi etc.
rand	Standard	Pseudo random number generator for integers, such as C rand()
whrandom	Standard	Whichmann-Hill pseudo random number generator for floating point numbers
array	Installed	Type for the efficient disply of arrays with uniform numeric values
OS services		
OS	Standard	Portable interfaces for operating system-dependent functions.
time	Installed	Time based function (standard C library functions: time, pause, etc.)
getopt	Standard	Analyzes command line arguments in sys.argv (such as UNIX getopt)
tempfile	Standard	Generates a temporary file name
Expanded OS services		
Signal	Installed	Handles asynchronous events (IPC: process signals)
socket	Installed	Access to BSD sockets interface (IPC: networking on lower levels)
select	Installed	Access to >UNIX select multiplex file synchronization< (IPC)
thread	Installed	The basis function for working with lightweight processes
UNIX Services		
posix	Installed	Standardized POSIX-UNIX system calls
posixpath	Standard	POSIX path name functions
pwd	Installed	Access to the UNIX password data base
grp	Installed	Interface to UNIX group data base
dbm	Installed	Dictionary / file-like interface to UNIX ndbm data base library
gdbm	Installed	Dictionary / file-like interface for GNUs dbm-like data base library
anydbm	Standard	Generic interface for dbm extensions

bsddb	Installed	Interface for >BSD db keyed-access file library<
curses	Installed	Interface for the ncurses library, for character based interfaces
termios	Installed	tty control with POSIX style
fcntl	Installed	fcntl() and ioctl()system calls
posixfile	Standard	File-like object with support for locking operations
Additional services		
pdb	Standard	Text oriented (command lines) debugger
profile	Standard	Python source code profiler
pstats	Standard	Python code profiler help program for viewing statistic files
dis	Standard	Python byte code disassembler
glob	Standard	File name globbing (such as UNIX Shell file name expansion)
grep	Standard	File string search (such as UNIX grep Shell command)
util	Standard	Missing functions that cannot be assigned to other locations
shutil	Standard	Functions for copying a file
Internet services		
cgi	Standard	Interprets WWW forms in server scripts
urllib	Standard	Opens the corresponding object after the URL is entered (requires sockets)
httplib	Standard	HTTP protocol client (requires sockets)
ftplib	Standard	FTP protocol client (requires sockets)
gopherlib	Standard	Gopher protocol client (requires sockets)
nntlib	Standard	Gopher protocol client (requires sockets)
urlparse	Standard	Parst a URL string in a >tuple<
htmllib	Standard	A parser for HTML files
sgmllib	Standard	Section of an SGML parser required for parsing HTML
rfc8222	Standard	A parser for mail header in RFC style
mimetools	Standard	Tools for parsing MIME-like message headers
Multimedia services		
audioop	Installed	Useful operations with sound fragments
imageop	Installed	Useful operations with images

aifc	Standard	Accesses audio file in AIFF or AIFC format
jpeg	Installed	Accesses JPEG image compressor and decompressor
rgbimg	Installed	Accesses SGI RGB image files (not SGI-specific)
Cryptography		
md5	Installed	Interface for RSAs MID5 Message Digest algorithm
mpz	Installed	Interface for the integer section of the >GNU multiple precision library<
rotor	Standard	Implementation of a >rotor-based< encryption algorithm
GUI Interfaces		
Tkinter	Standard	Python class-based interface for Tk GUI API (uses tkinter module)
Tkinter	Installed	The hidden C extension module used by Tkinter (not for importing)
stdwin	Installed	STDWIN standard Windows system interface
stdwinevents	Standard	STDWIN event, command and select constants
rect	Standard	STDWIN >rectangle manipulation operations

4 ADDITIONAL VIEWS, TOOLS,...

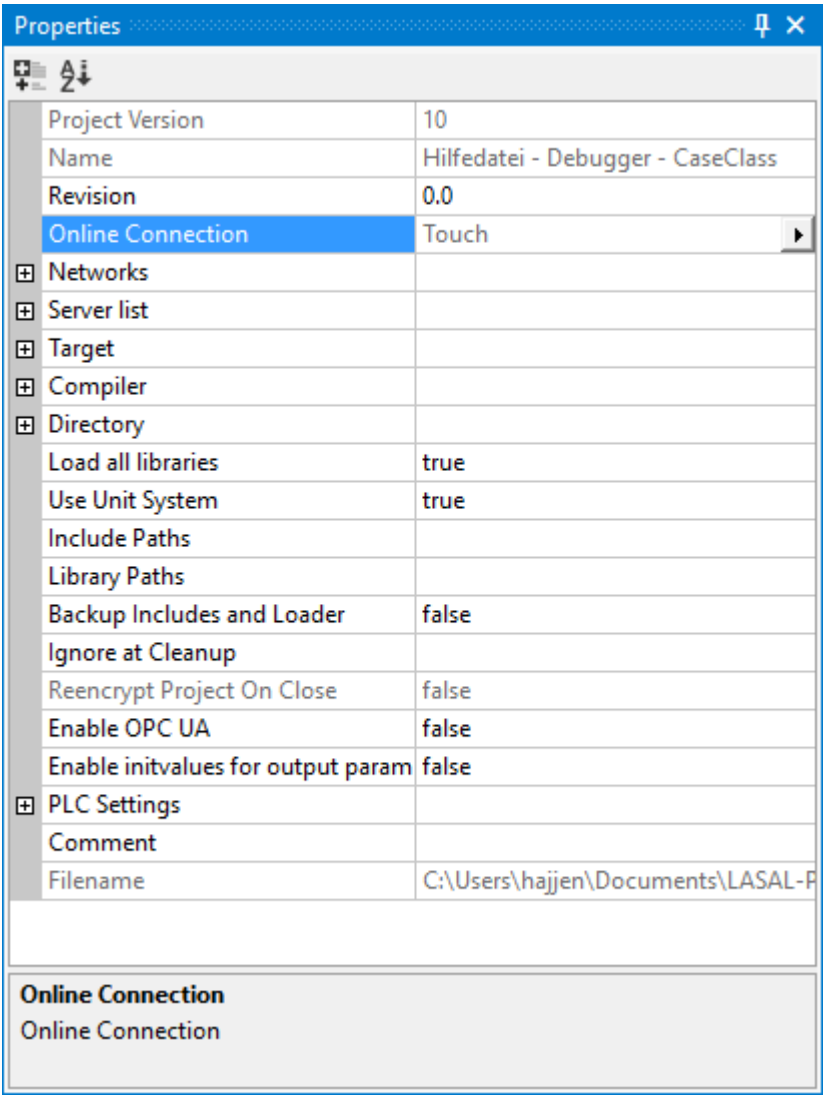
4.1 Online Settings

The following pages describe method for communicating with controls used by SIGMATEK. Using this document, it should be possible to establish a connection with all SIGMATEK products using **LASAL CLASS 2**.

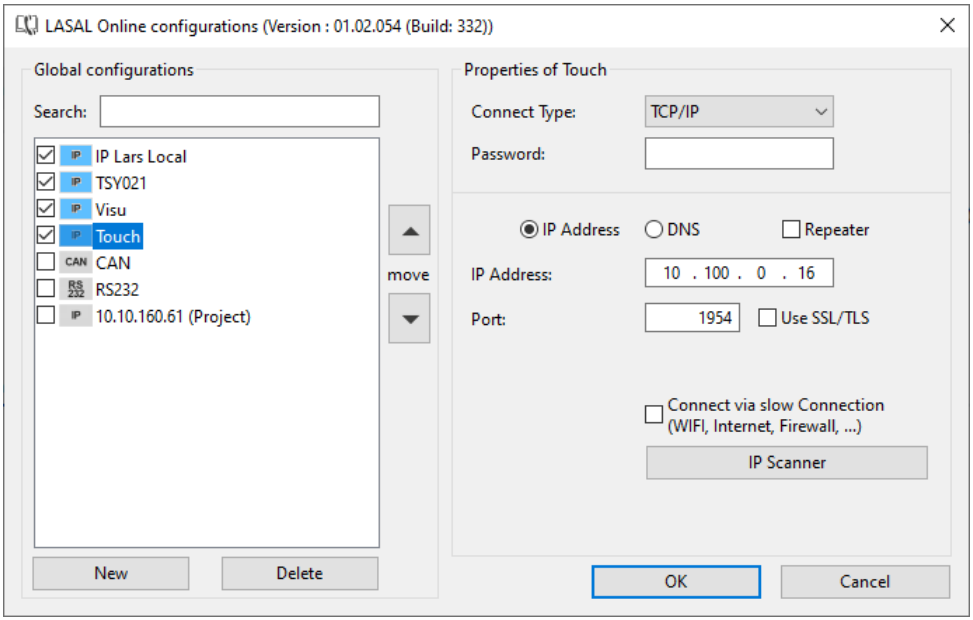
Here you find the various connections available over the **serial interface**, **CAN bus**, **TCP/IP** and **Modem**. How the connection type and their configurations are created, can be found in their respective product manuals. The settings required for communicating with the **LARS** simulation program are also explained.

4.1.1 Basics

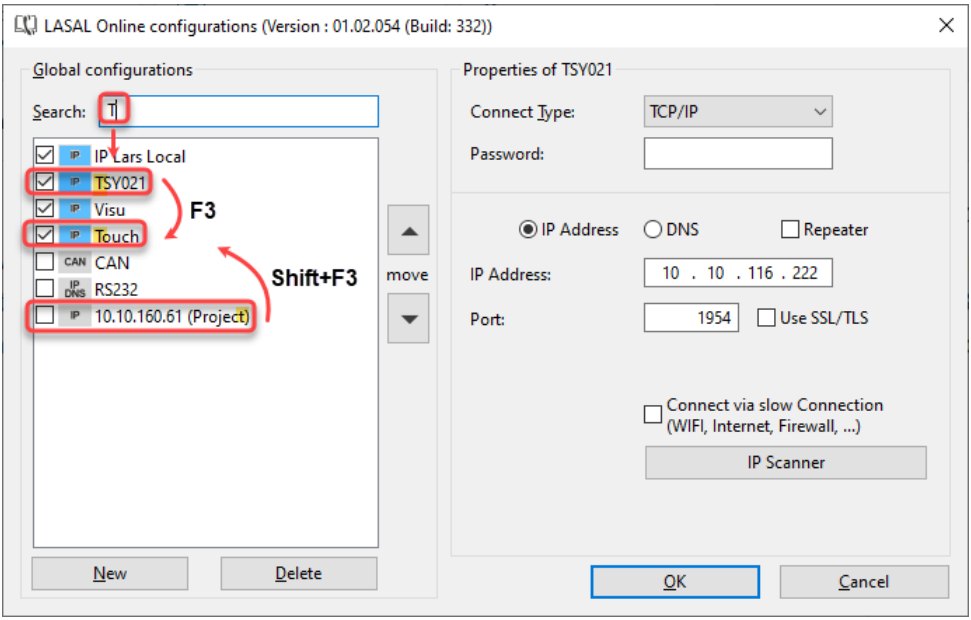
This chapter shows the steps required to create an **online connection** in **LASAL CLASS 2**. First click on **Online Connection** in the **Properties window**. The menu item is activated and an arrow appears on the right. Press the arrow.



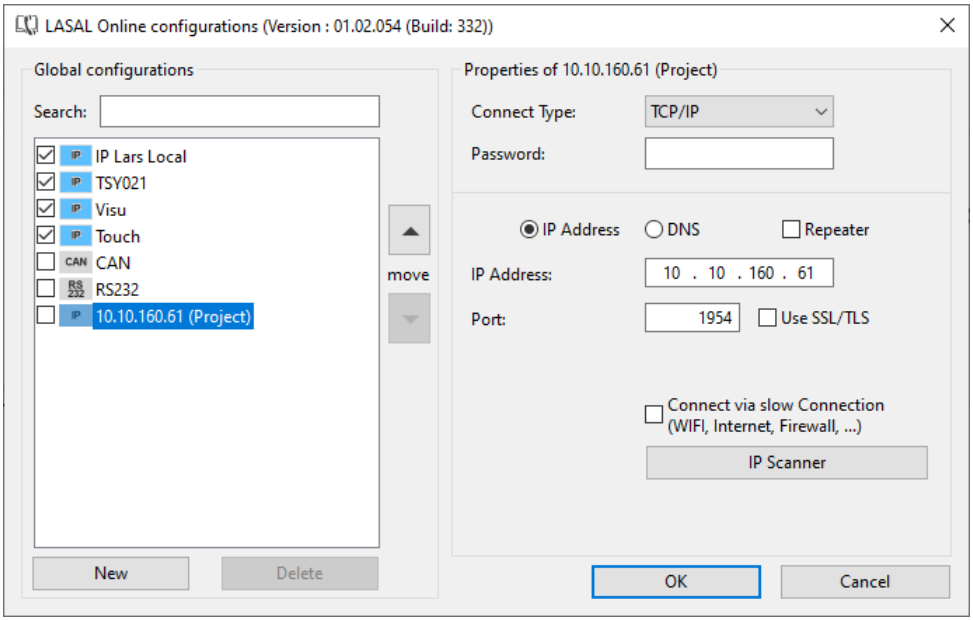
The **LASAL Online Configurations** window then opens.



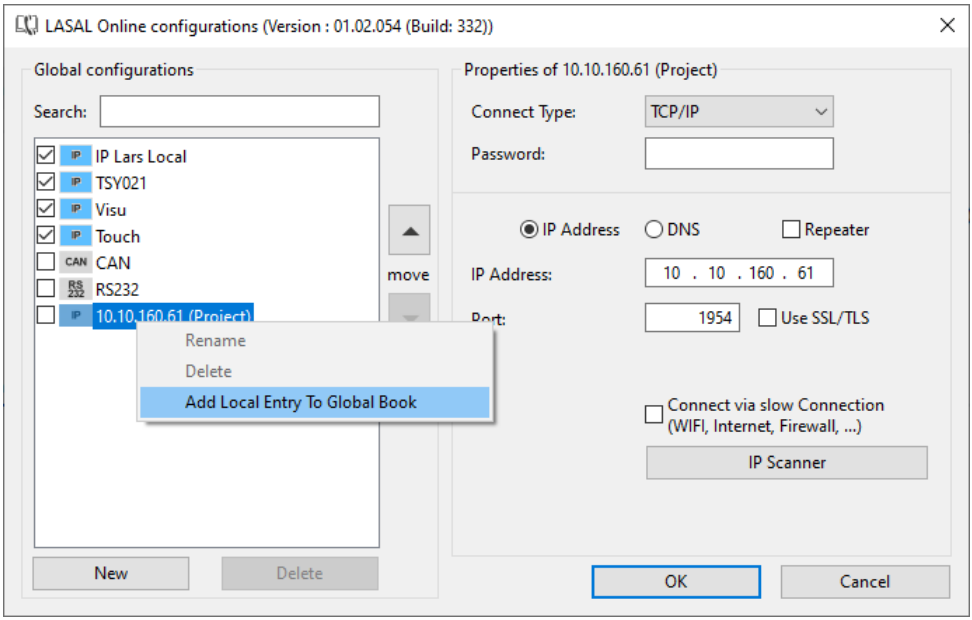
In this window, new connections can be created (**New button**), or existing ones deleted (**Delete button**). The **Move buttons** (**arrow buttons**) can be used to move connections up or down. It is also possible to search for connection names, IP addresses or DNS names in the **search field**. The first hit is highlighted, the **F3 key** can be used to jump to the next one, and **Shift+F3** to jump back to the previous one. You can also search for parts of the name / address. In the following image, three connections are found with the letter "T" entered in the search field:



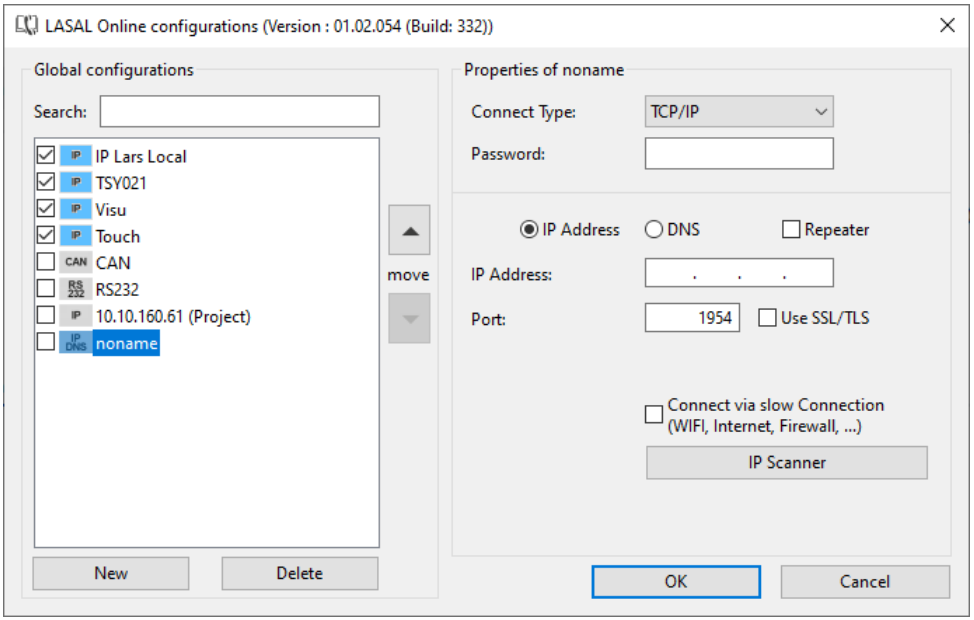
If the connection stored in the project is not available in the telephone book, “(Project)” is added to the name:



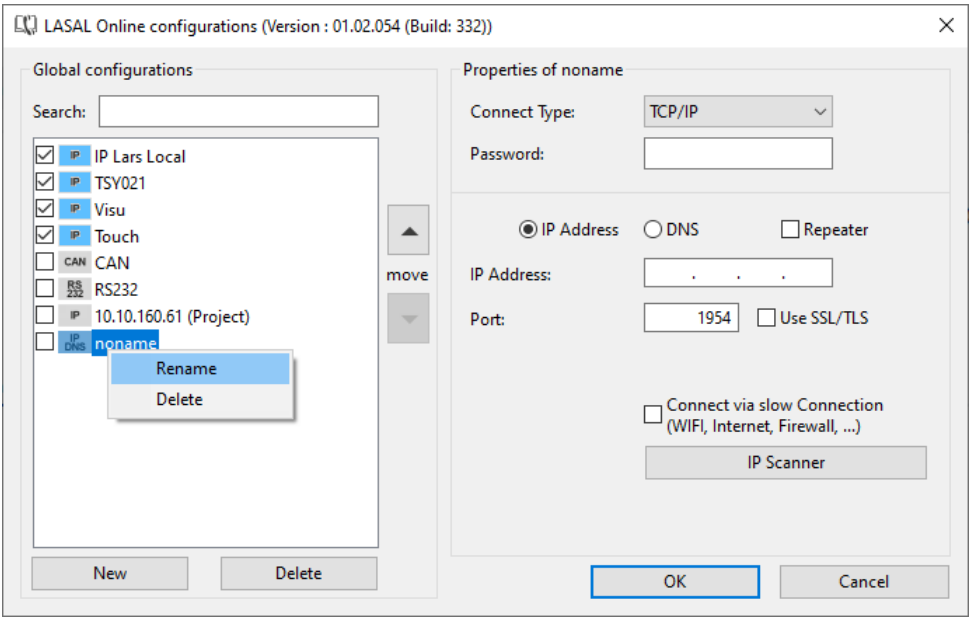
In the context menu of such a local connection, **“Add Local Entry To Global Book“** can be selected and the local connection entered in the global telephone book.



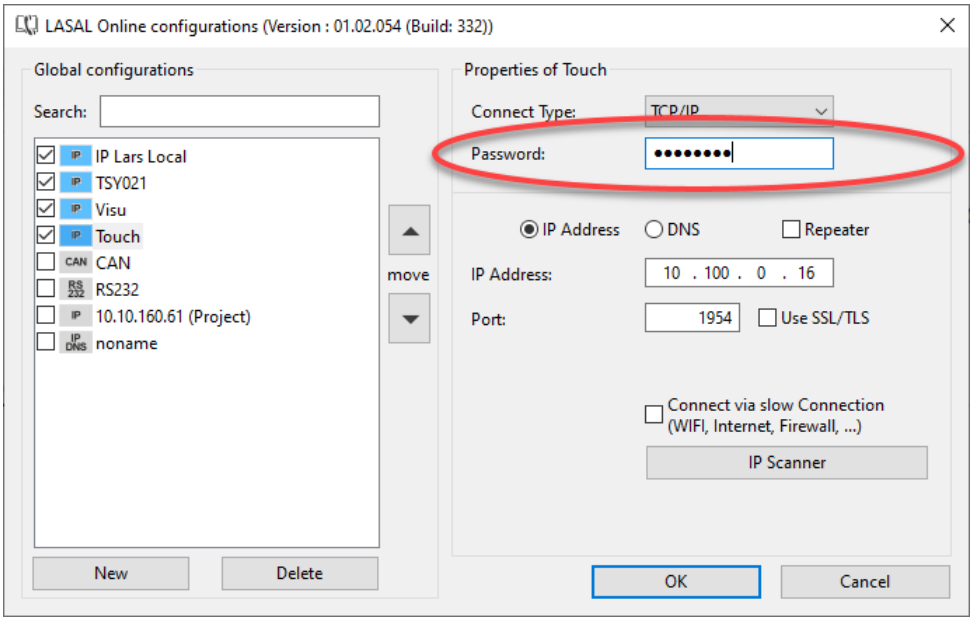
In the left section of the following window, a newly created connection can be seen. The default name of this connection is **noname**.



To change the name of a connection, click on it the name (**noname**) the right mouse button and select **Rename** from the context menu then enter the new name.

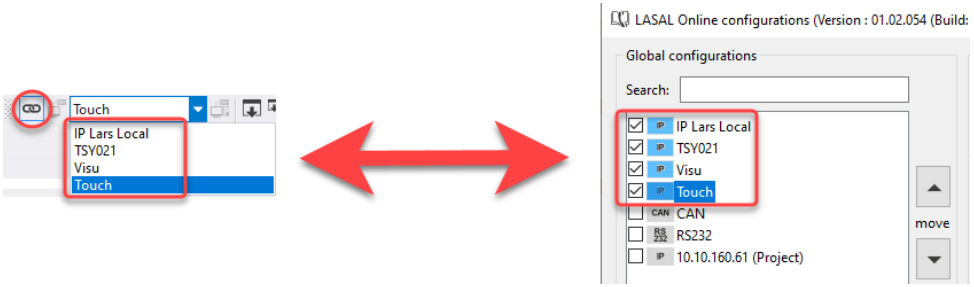


If going online with the control should be prevented for unauthorized persons, a password is entered in the appropriate field. To go online with this control a later time, the password is first requested – if the entry is incorrect, no connection is made. This password protection is only available at TCP / IP and ROUTE.



With `SET ONLINEPWD <Passwd>`, the password is set in the control; to change it, the `SET ONLINEPWD <altes Passwd> <neues Passwd>` command is used.

The **checkboxes** on the left side of the configuration name stand for “Favorite”. If a connection is marked as a favorite, the **connection type is marked in blue** and it is added to the **favorites list**. If you go online in LASALL CLass 2 and click the Down arrow in the toolbar, the DropDown list **shows** offline all, online **only the connections in the favorites list**.



4.1.2 Connection Types

In this chapter, the configurations that must be made for the connection types to use a specific connection type are covered. The following interfaces can be selected:

4.1.2.1 Connection over the Serial Interface

This communication interface is extremely important since a valid connection can always be made with a serial connection. If a serial connection is made, the autoexec.lsl with the required online settings can be downloaded from the destination system. These online settings can be used to establish a faster connection (e.g. TCP/IP).

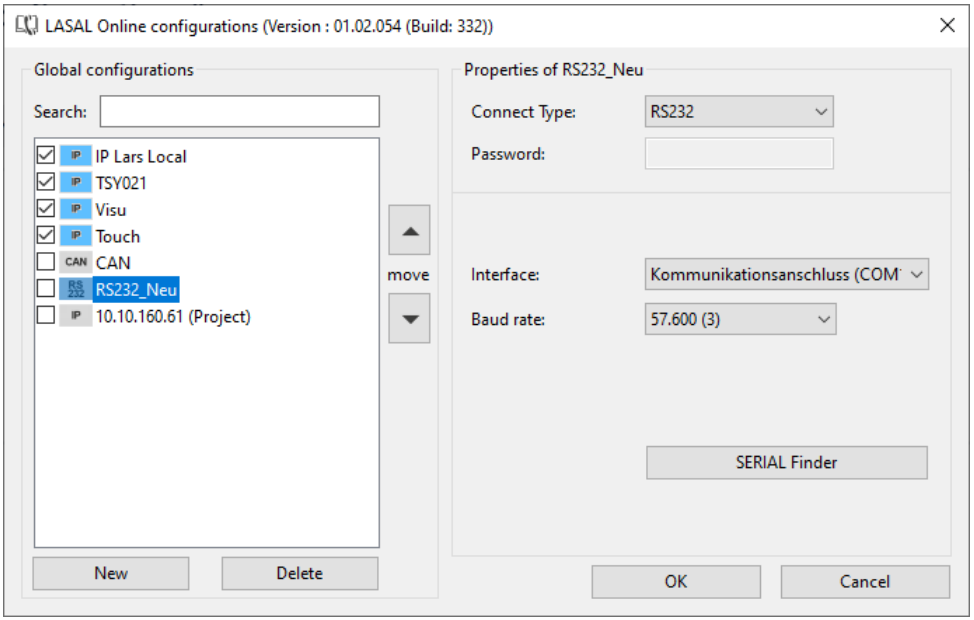
4.1.2.11 Settings in the LASAL Online Configurations Window

Here, a valid configuration must first be selected. An appropriate name is recommended for the connection type (e.g. **serial** or **RS232**, here **RS232_Neu** is assigned). Next, under Connection Type:, **RS232** is selected. The **interface** and **baud rate** must also be set.

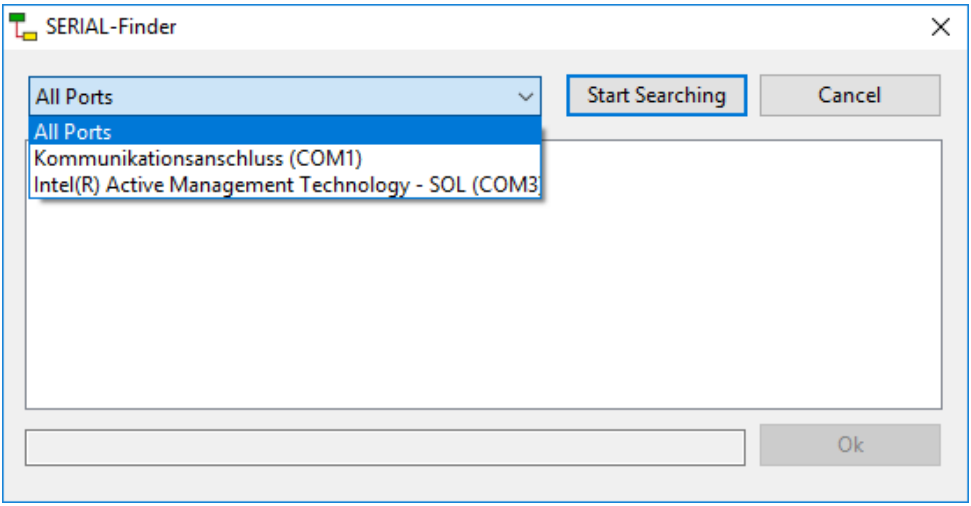
Select the corresponding **COM** number depending on the port to which the serial cable is connected. If only one serial connector is available, **COM1** must be selected as the interface.

Set the Baud rate for each interface defined In the CPU. If the baud rate is not known, simply try all the available baud rates until a connection is established.

Interface: **COM1** and baud rate: **57.600 (3)** are default settings.

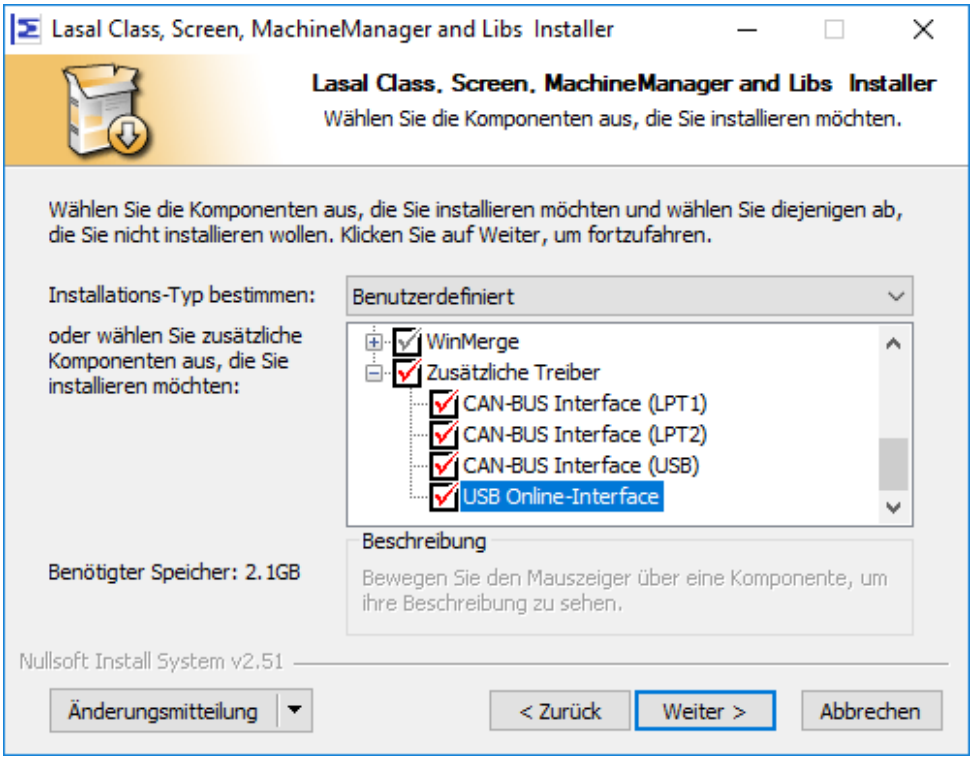


At click on the **SERIAL Finder** button a dialog appears wherein on all COM interfaces is searched for SIGMATEK controls. Thereby it is attempted to go online with all possible baud rates.

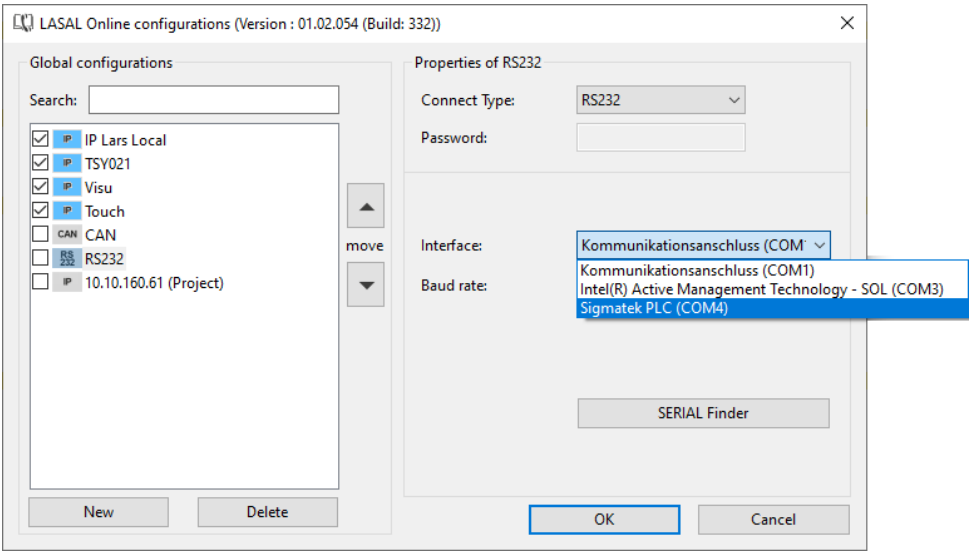

SERIAL-Finder
All Ports
Start Searching
Cancel
All Ports
Kommunikationsanschluss (COM1)
Intel(R) Active Management Technology - SOL (COM3)
Ok

4.1.2.2 Connection over USB Cable

For CPUs with a device interface, the option to go directly online over a USB cable is available. Here, the USB Online-Interface must be selected in the installation dialog box under "Additional Drivers".



If LASAL 2 is installed in this way and RS232 is selected as the connection type and a SIGMATEK CPU is connected to the PC via USB plug and booted up, the **SIGMATEK PLC** selection option is displayed under Interface.



.

4.1.2.3 Connecting over the CAN Bus

The CAN bus is faster than the serial connection and is especially suited for controlling DIAS modules.

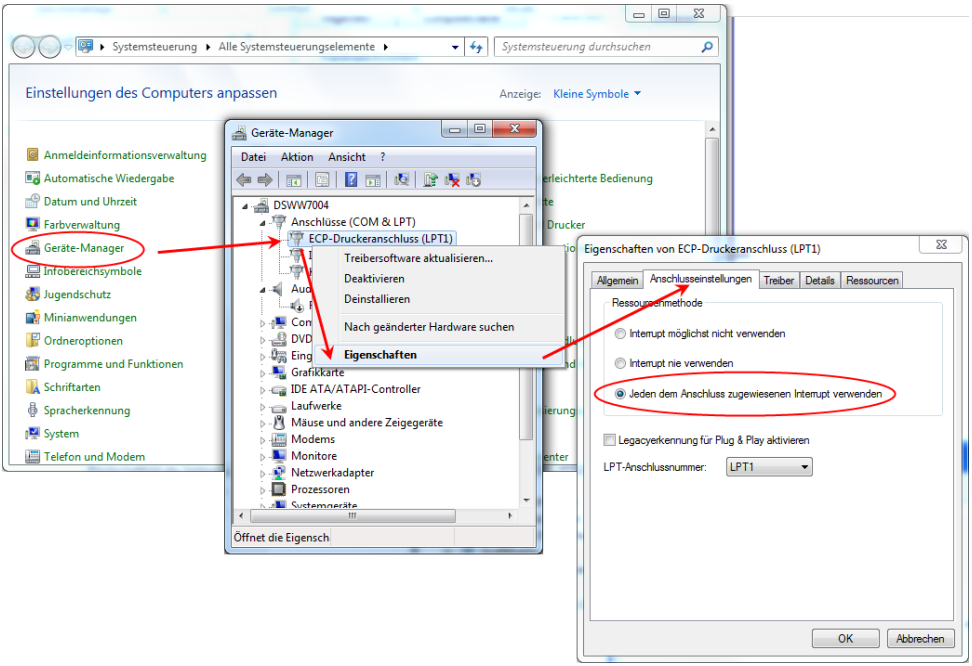
The DIAS adapter (see picture) is connected to the **LPT printer port** of the PC. Therefore, the operating system configuration must be changed.



4.1.2.31 Settings in the Operating System

Under **System Control** in the **System folder**, the **Hardware manager** is found under the **Hardware tab**. In the menu item **connections**, the desired printer port can be selected. In most cases a computer has only one printer port so only **LPT1** can be used.

Next, open the properties window with a double click on the printer port (**LPT1**) then open the connection settings tab and activate the **Use interrupts for every connection** option field as shown in the following illustration.



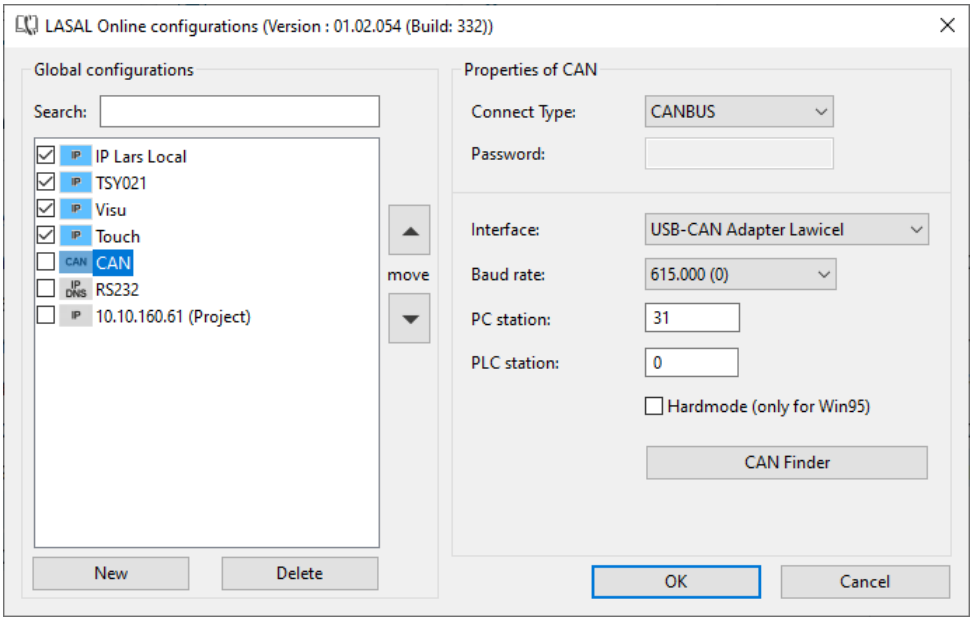
The PC must be restarted in order for the modifications to take effect.

4.1.2.32 Settings in the LASAL Online Configurations Window

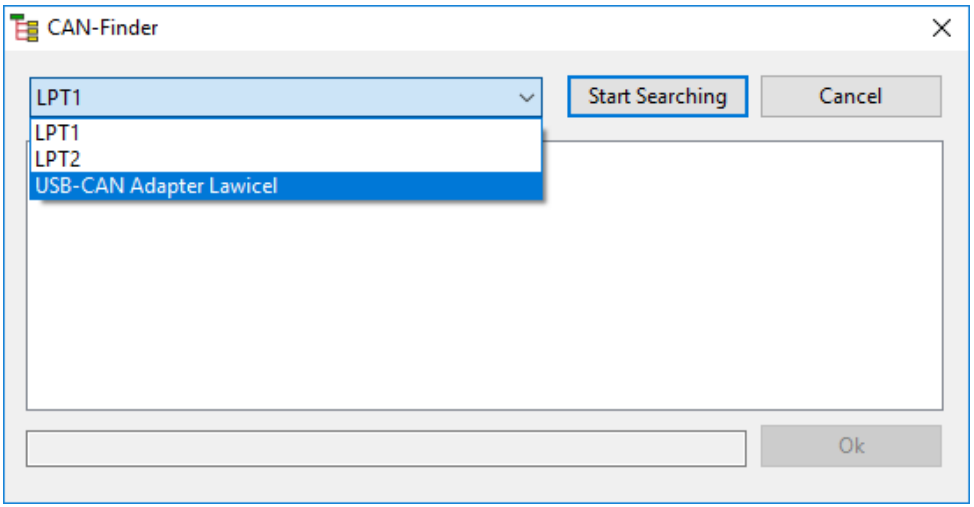
To begin, select a new configuration and under **Connect Type:** select **CAN BUS**. Afterward, select the printer port (**LPT1**) in the **Hardware manager** under **Interface:**. The **baud rate** and **PLC station** must have the same setting as the destination control. For the **PC station** setting, different numbers are set in **LASAL CLASS 2** and the LSE project.



It is recommended to activate the **Hardmode (only for Win95)** selection, as the connection is faster.



In the CAN-Finder window you can search for a CAN connection on the desired interface (**LPT1, LPT2** and **USB-CAN Adapter Lawicel**).



Thereby via the selected interface all station numbers with all baud rates are scanned.

4.1.2.4 Connecting Over TCP/IP

The **TCP/IP connection type** has the fastest data transfer rate. This connection is a must for acceptable download time for large projects. A further advantage is that with a TCP/IP connection, several controls in the same network can be handled simultaneously.

4.1.2.41 Settings in the Operating System



If in a company network, the network administrator should be consulted before making any changes in the operating system. In such a case, one or more IP addresses should be requested and not assigned arbitrarily.

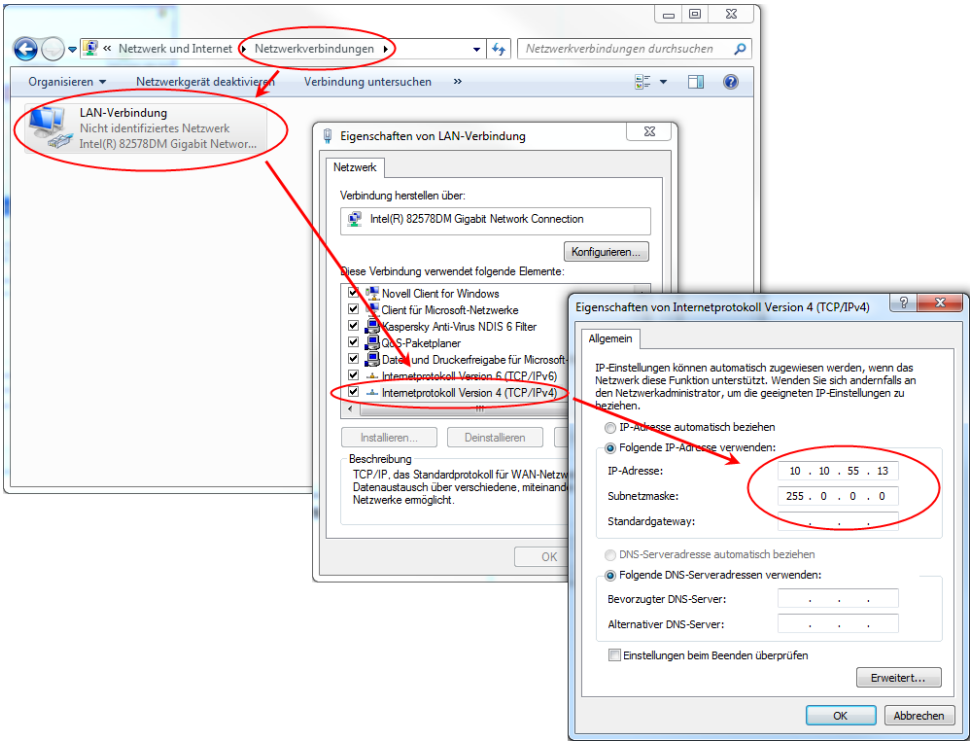
If only one connection between the PC and the control is to be made, the following instructions must be followed:

First, a functioning network card and a crossed network cable are needed.

Once these requirements are met, open the **Network Connection window** in the operating system. Click on **network environment** and select **properties** in the context menu. In the following window, **LAN connection** must be clicked with the right mouse button and **Properties** selected in the context menu. The **Properties of the LAN Connection window** opens as shown below. In this window, select the **Internet Protocol (TCP/IP)** and under This connection uses the following elements:, select **Properties**.

The **Properties of Internet protocol (TCP/IP) window** now appears on the screen. In this window, a valid **IP address** and **subnet mask** are assigned.

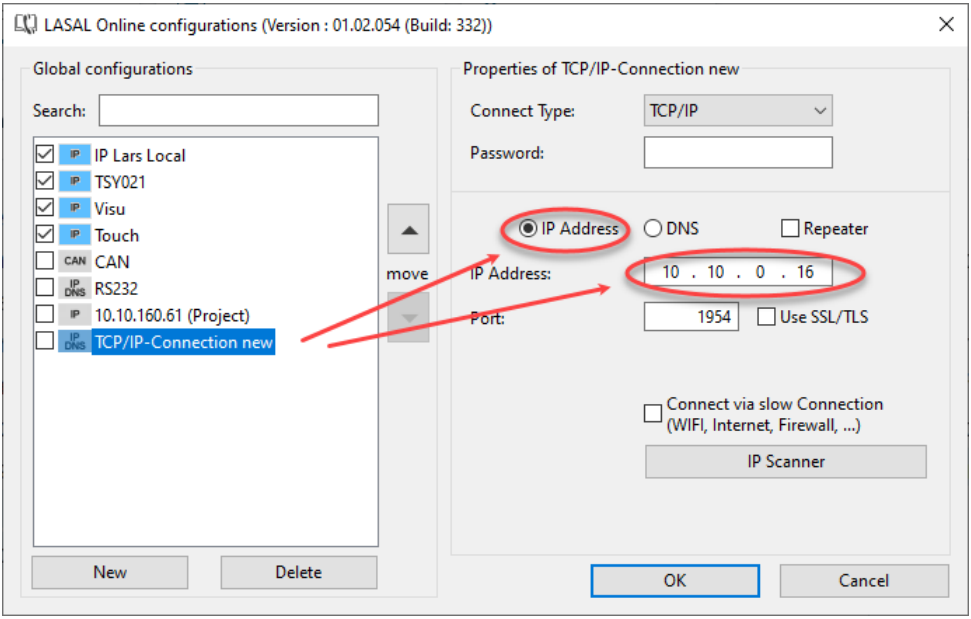
It is best that the same setting be used as those in the graphic below. The Subnet mask must also be set for the control as wall as a similar IP address. Under **IP Address:** assign an IP address that corresponds to the one assigned in the operating system (only the last value must be changed and can be between 1 and 254, here the last number is 14 instead of 13).



In a network, two machines can **never have the same IP address**!

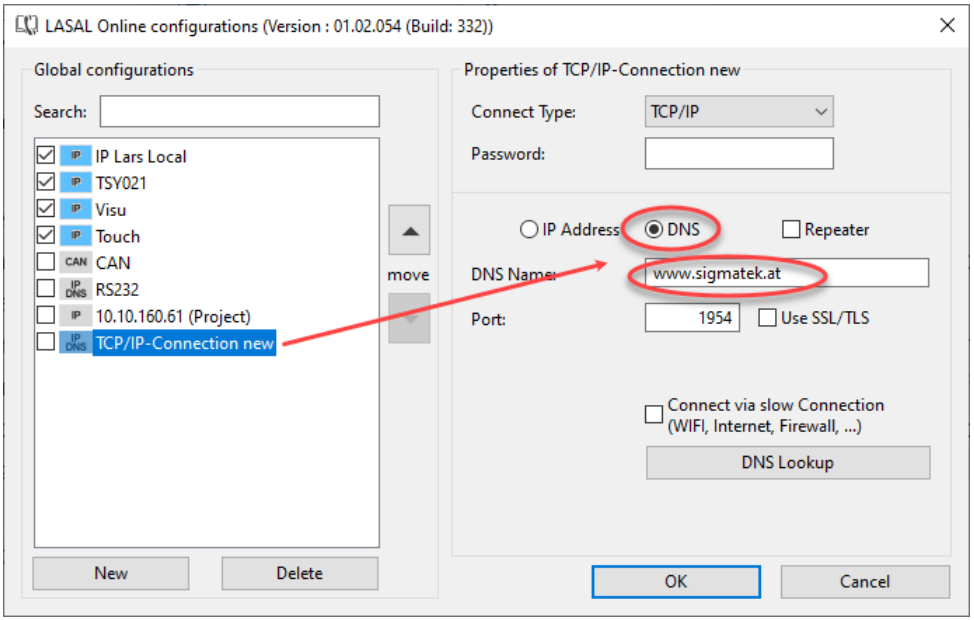
4.1.2.42 Settings in the LASAL Online Configurations Window

Select a valid configuration and choose TCP/IP in **Connect Type**. You can choose between IP address, DNS name or Repeater. Depending on the selection, the dialog changes slightly:

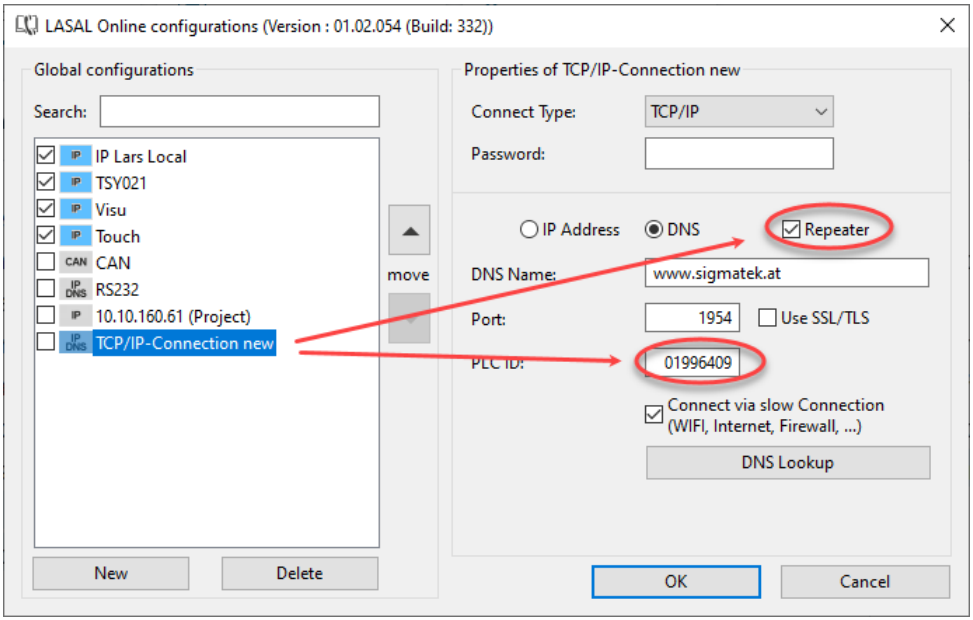


IP Address

By clicking on the **IP Scanner** button, a dialog containing a list of all SIGMATEK controls available in the network. The desired control can be selected with help from the IP address, control type, serial number and project name.


LASAL Online configurations (Version : 01.02.054 (Build: 332))
Global configurations
Search:
IP Lars Local
TSY021
Visu
Touch
CAN
RS232
10.10.160.61 (Project)
TCP/IP-Connection new
move
New
Delete
Properties of TCP/IP-Connection new
Connect Type:
TCP/IP
Password:
IP Address
DNS
Repeater
DNS Name:
www.sigmatek.at
Port:
1954
Use SSL/TLS
Connect via slow Connection
(WIFI, Internet, Firewall, ...)
DNS Lookup
OK
Cancel

DNS Name



DNS with Repeater

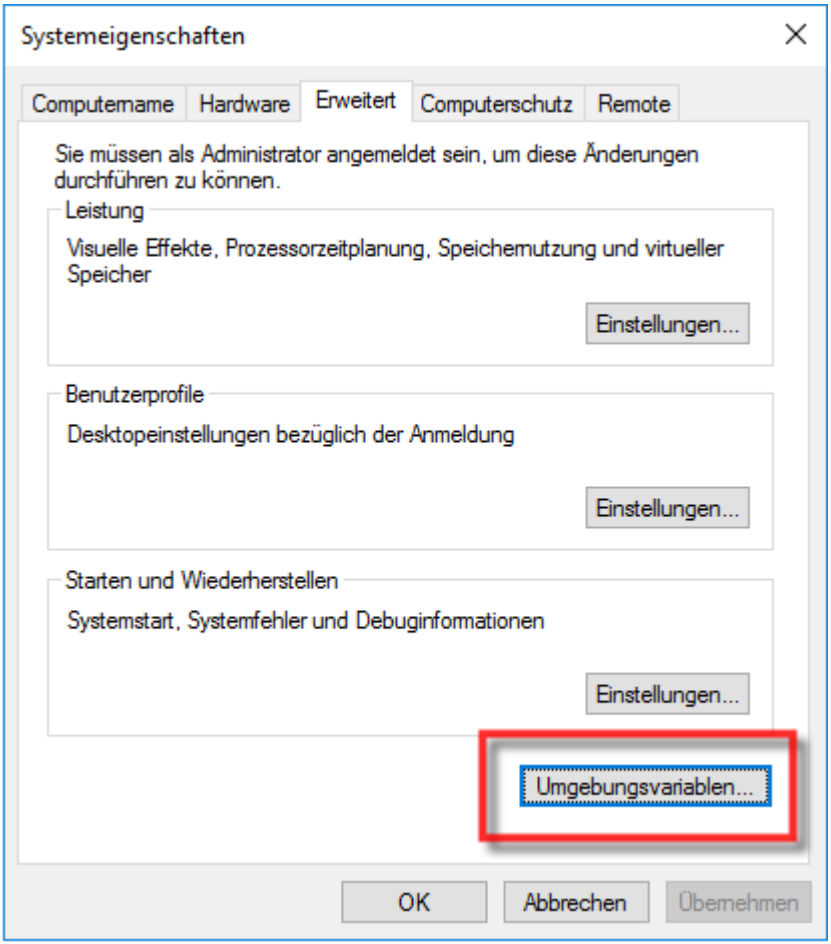
While creating a TCP connection, the availability of the remote station is checked. If no answer is received after a certain time, it is assumed that no remote station exists. The timeout is 2.5 seconds. If using a slower online connection however (WiFi, Firewall Internet etc.), it could take longer to establish the connection. This can be solved by enabling the check box **Connect via slow Connection**. If checked, the timeout is increased to 30 seconds.

If you communicate with a control via the internet, a slow LAN connection or a Windows platform with very short TCP timeout settings, transferring especially bigger files can lead to timeouts.

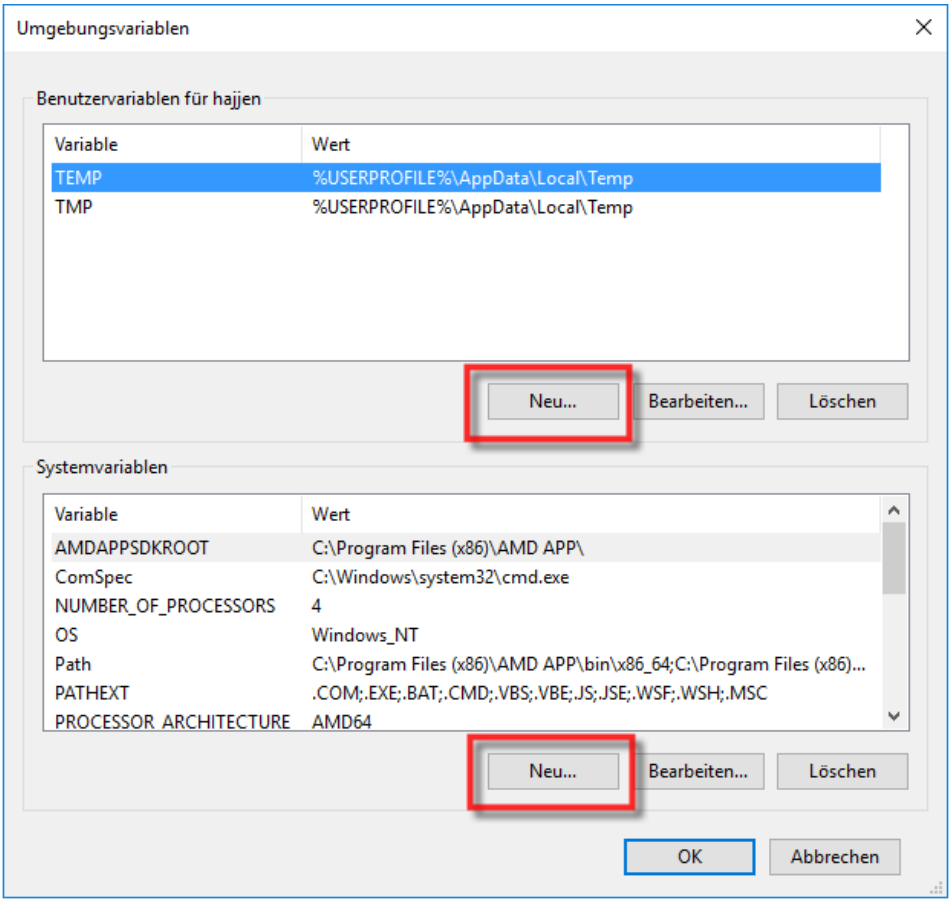
The Lasal32.dll uses the Windows TCP timeout settings as default values. It is possible to adjust these values to the own requirements with the help of environment variables. For this open the Control Panel and there the topic System.

Windows 10: Start -> Settings -> Control Panel -> System and Security -> System

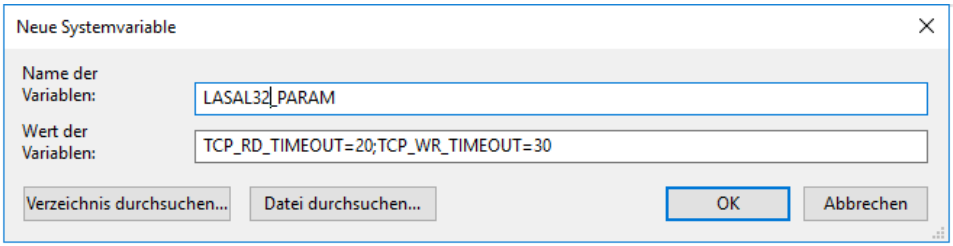
There in the tab **"Extended"** you will find the possibility to create **environment variables**.



In the window **"Environment variables"** opening now, you find the possibility to create variables for the logged-in user or for the entire system (computer). Click the button **New** in the desired area of application.



In the window opening now, the name as well as the value of the variable have to be entered.



Name of the Variable:

LASAL32_PARAM

Value of the variable:

TCP_RD_TIMEOUT=<tcp/ip read timeout in sek.>

TCP_WR_TIMEOUT=<tcp/ip write timeout in sek.>

Example:

LASAL32_PARAM = TCP_RD_TIMEOUT=20;TCP_WR_TIMEOUT=30;

To confirm the settings, the system has to be restarted.

Connecting via Repeater

The LASAL Repeater for SIGMATEK controls was developed to connect with controls – e.g. through a firewall – that do not allow access via the Internet.

The control establishes a connection to a repeater server. This connection is maintained as long as the repeater function is active in the control. After the control has logged in to the server, its standard poerts are provided in the repeater server.

So that the Lasal Repeater can be started, a valid ***lslrepeater.cfg*** file must be available in the **„C:\LSLSYS\“** directory of the control.

With the **lslrpt start** command via CLI or directly in the Autoexec.lsl, **the function is started in the control**.

The command **lslrpt stop** ends the function in the control.

Under certain conditions, a gateway must also be set (consult the network administrator for this information). The command for this is:

set ip gateway *192.168.1.1*

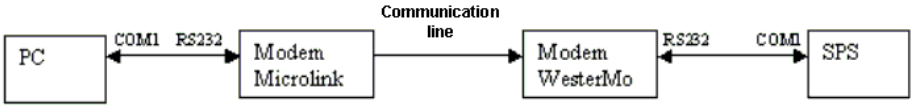
In the **LASAL Online Configuration** window – depending on whether **IP address** or DNS was selected – the IP address in enter under IP Address or under DNS Name, the name of the repeater server. Under **PLC ID**, the number of defined in the configuration file as

REPEATER_ID is entered. If the value **SERIALNUM** is entered for this parameter, the serial number of the control is used and must be entered in the field **PLC_ID**.

4.1.2.5 Connecting via Modem

Basically, both Modems must control the **DIREKT-Modus (AT\N1)** and the Modem on the destination system must be able to establish a connection without a hardware handshake. For this reason, it is recommended that the PLC side be equipped with a **WESTERMO TD-32 DC** and the LASAL CLASS 2 side, an **ELSA-Modem**.

4.1.2.51 Block Diagram of the Connection



4.1.2.52 Destination System Settings

- The Modem cannot send any messages to the PLC.
- The Modem itself must take an incoming call.
- A baud rate of 9600 and an 8N1 data format must be fixed
- The Modem must operate in direct mode. Neither buffer operation or error correction is allowed.
- No data flow controls can be activated (RTS-CTS or XON-XOFF).

4.1.2.53 Example of WESTERMO TD-32 DC

The Modem operating instructions should also be read.

Standard dial connection

SW 1	on				
		●	●	●	●
		1	2	3	4

DTR/DSR switched off

SW 2	on				●				
		●	●	●		●	●	●	●
		1	2	3	4	5	6	7	8

Two wire connection

SW 3	on	●							
			●	●	●	●	●	●	●
		1	2	3	4	5	6	7	8

9600 Baud, 8N1

SW 4	on		●	●		●	●		
		●			●			●	●
		1	2	3	4	5	6	7	8

Automatic recording of speed

SW 5	on								
		●	●	●	●	●	●	●	●
		1	2	3	4	5	6	7	8

4.1.2.54 Online Configuration

The setting must be made in a terminal program (e.g.: Windows HyperTerminal).

If setting the modem is unclear, the standard settings can be called as follows: **AT&F&W0 …… Reset to standard**

The configurations string for the Westermo Modem written as:

ATQ1E0&K0\N1S0=2&W0

ATQ1 Quiet mode, no return messages

ATE0	No echo from commands (not required)
AT&K0	No data flow control
AT\N1	„Direct mode"
ATS0=2	Automatic pick up after to call signals
AT&W0	Save as actual configuration

Init-String for **Westermo TD-32 DE** modem:

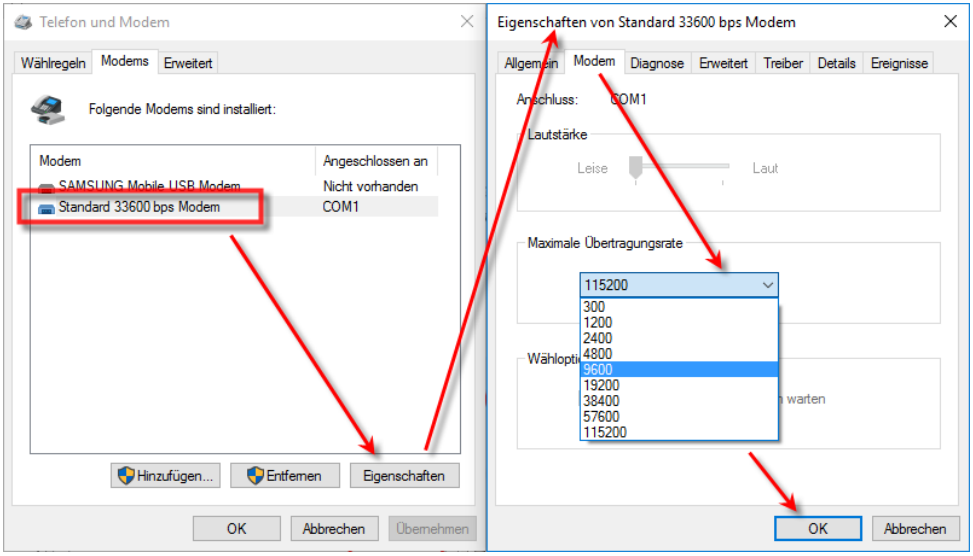
ATQ0V1E0S0=0&S1&D2X3L1\N1&K0\C2

With the **AT\S** command, the actual configuration can be tested and should correspond to the list shown below:

```
CMD DESCRIPTION / OPTION    CMD DESCRIPTION / OPTION    CMD DESCRIPTION / OPTION
--- --------------------    --- --------------------    --- --------------------
    COUNTRY..........GER    &A  CHR ABORT OPT.....NO    *H  NEG. SPEED......HIGH
    DTE BPS.........9600    &B  DTR DIAL OPTION...NO    S0  RINGS TO ANS.....002
    DTE PARITY.....8NONE    &C  DCD OPTION........ON    S1  RING COUNT.......000
    LINE SPEED.......NONE   &D  DTR OPTION.........0    S2  <ESC> CHAR.......043
B   BELL MODE........OFF    &G  GUARD TONE......NONE    S3  <CR> CHAR........013
E   CMD ECHO.........OFF    &K  FLOW CONTROL....NONE    S4  <LF> CHAR........010
F   LINE MODE.......AUTO    &L  NETWORK.........PSTN    S5  <BS> CHAR........008
L   SPKR VOLUME......LOW    &Q  ASYNC/SYNC.........0    S7  CONNECT TIME.....060
M   SPKR CONTROL....CALL    &R  RTS/CTS.........AUTO    S8  PAUSE TIME.......002
N   AUTO MODE.........ON    &S  DSR OPT............0    S12 ESC GUARD TIME...050
Q   QUIET............ON    &T  ENABLE RDL........NO    S30 CONNECT INACT....000
V   RESULT FORM.....LONG    &X  SYNC CLOCK.......INT    S32 XON CHAR.........017
W   EC MSG............0    &Y  PROFILE........NVM.0    S33 XOFF CHAR........019
X   EXT RESULTS........4    \A  MAX BLK SIZE.....192    S36 FALLBACK ACTION..007
Y   LONG SPACE DISC...NO    \G  REMOTE FLOW......OFF    S37 MODE SELECT......000
%C  COMPRESSION.....BOTH    \K  BRK OPT............5    S48 V42 NEG CTRL.....007
-K  EXT. SERVICES......0    \N  ECL MODE.........DIR    S95 RES. CODE........000
```

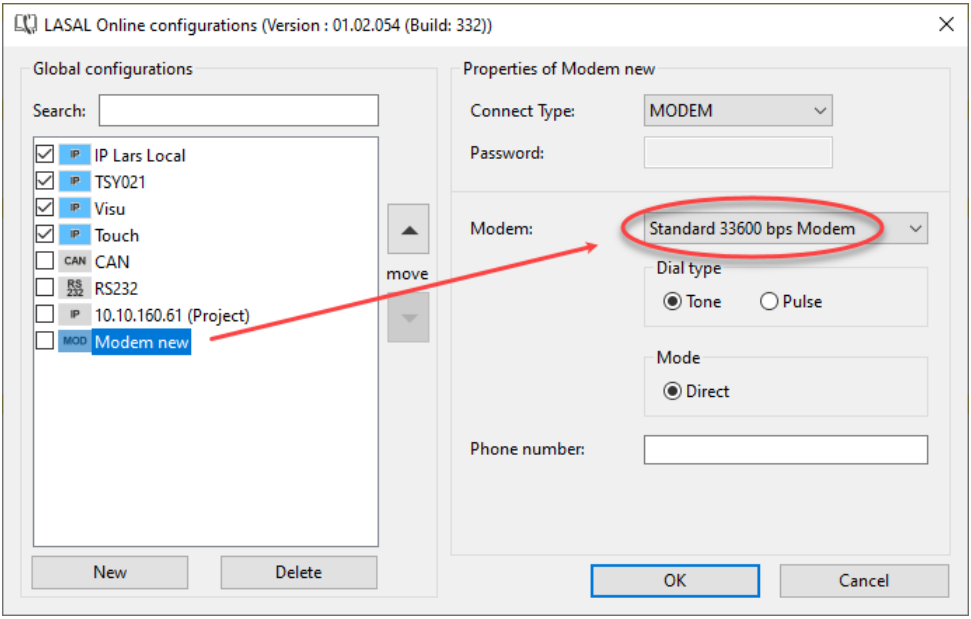
4.1.2.55 Operating System Settings

To start, the Modem must first be installed. In the modem settings this modem must be set to **9600 Baud** and **8N1 data format**.

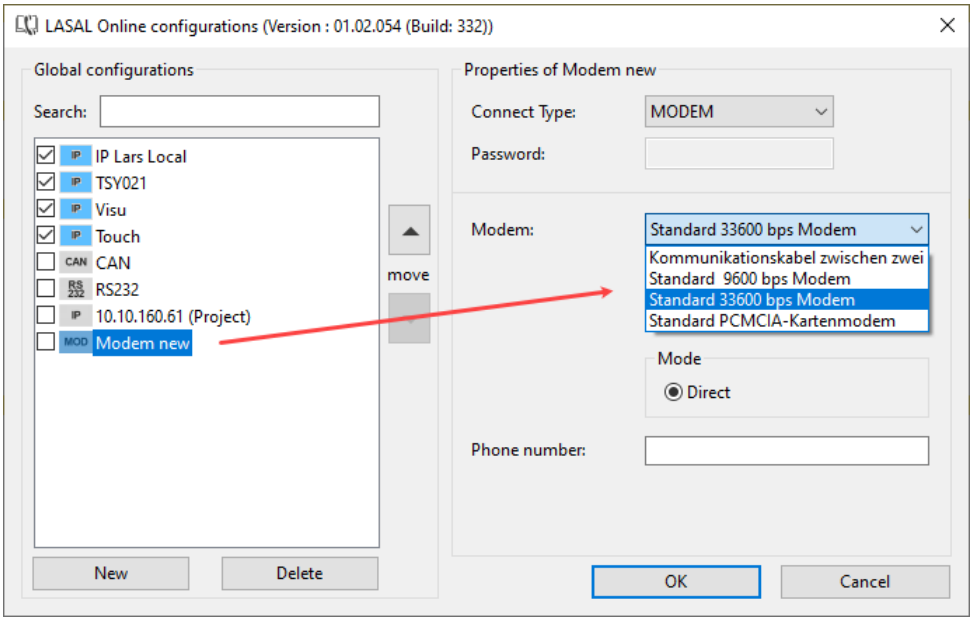


4.1.2.56 Settings in the LASAL Online Configurations Window

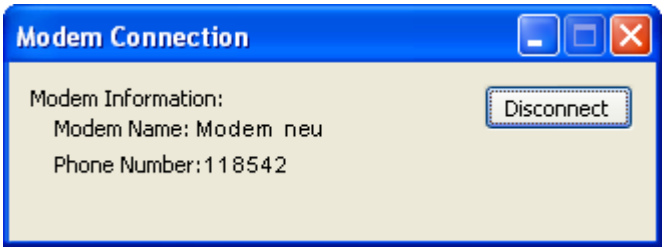
First the Modem in use must be selected. If it was installed, it should be in the list.



In the Modem field, the modem type can be selected.



After establishing the modem connection, a dialog box for disconnecting appears.



If Disconnect is clicked, the modem connection is terminated. The connection can also be terminated using the Python script instruction "CloseModemConnection()".

4.1.2.6 Connection via ROUTE

The connection type ROUTE serves to go online to a second control via an already known online connection to a control. If, for example, the IP address of control A is known and control B is connected to control A via CAN bus, so it is possible to go online with control B without any information about station number and baud rate. Likewise, it is possible to go online via the known IP address of station A to station C with an unknown IP address. Prerequisite for this is that a routing table is stored at all CPUs, with whose information the controller B res. controller C can be addressed. Likewise there must be corresponding entries in the respective `AUTOEXEC.LSL` on the CPUs.

Which files have to be created at the single CPUs within the network is explained by the following example:

The user wants to go online on a CPU (station 3), which in turn is connected via CAN cable to another CPU (station 2). This CPU in turn is connected via TCPIP to a CPU (station 1) whose IP address is known to the user.

So: PC --> via TCPIP --> first CPU (Station1) --> via TCPIP --> second CPU (Station2) --> via CAN interface --> third CPU (Station3)

- Examples for the routing tables (ROUTE.TXT)
- Examples for the AUTOEXEC.LSL entries
- The dialog

Routing tables

As already described, there must be a corresponding ROUTING.TXT file on each affected CPU.

Necessary ROUTING.TXT (Station1)

```
----------------------------------------
# STATION 1                          // This station is known to LASAL CLASS 2 (it is in the
 STATION1 1 LOCAL                    // LOCAL means: Station1 itself is the destination. If da
                                     // they are processed internally on this CPU and do not
 STATION1 2 IP 192.168.171.221:1954 // This is the IP address of Station2.
 STATION1 3 IP 192.168.171.221:1954 // This address is identical to that of Station2, since
                                     // via CAN bus, and therefore has no IP address of its ow
                                     // This means that if the destination is Station2 or Sta
                                     // the data is sent to Station2 for further processing in
----------------------------------------
```

Necessary ROUTING.TXT (Station2)

```
----------------------------------------
# STATION 2                          // This is the station to which the target station (Stat
                                     // The IP address of this station is not known to LASAL
 STATION2 1 IP 192.168.171.220:1954 // This is the IP address of Station1.
 STATION2 2 LOCAL                    // LOCAL means: Station1 itself is the destination. If da
```

```
                                    // they are processed internally on this CPU and do not hav
 STATION2 3 CAN1 2                  // Station3 is connected to the CAN1 interface of station2
----------------------------------------
```

Necessary ROUTING.TXT (Station3)

```
----------------------------------------
# STATION 3                         // This is the target station. It is connected to station2
                                    // Attention: Station1 is unknown to this station!
 STATION3 2 CAN1 1                  // Station2 is the station connected via CAN bus with the
 STATION3 3 LOCAL                   // LOCAL means: Station1 itself is the destination. If data
                                    // they are processed internally on this CPU and do not hav
----------------------------------------
```

For the sake of clarity, these three ROUTING.TXT contents can be merged into one file:

Clear content of all three ROUTING.TXT files (this time without comments):

```
----------------------------------------
# STATION 1
 STATION1 1 LOCAL
 STATION1 2 IP 192.168.171.221:1954
 STATION1 3 IP 192.168.171.221:1954

# STATION 2
 STATION2 1 IP 192.168.171.220:1954
 STATION2 2 LOCAL
 STATION2 3 CAN1 2

# STATION 3
 STATION3 2 CAN1 1
 STATION3 3 LOCAL
----------------------------------------
```

This file can then be uploaded to all 3 stations (C:\ROUTING.TXT).

If LASAL CLASS 2 knows the address of a CPU (i.e. if it is in the phone book, in the above example it is the address of station1), it can simply go online to the CPU with the route number 3 (to be entered in the route dialog, this is the station number). But first the entries must be inserted into the respective AUTOEXEC.LSL files:

AUTOEXEC.LSL

Entry into the AUTOEXEC.LSL file on the CPU Station1:

```
SET STATION 1 // This is my own station number
SET ROUTINGTABLE C:\ROUTING.TXT  // Location and name of the routing table file
```

In the first line you define your own station number within the route (here: 1). In the second line the path and the file that contains the routing table.

Entry into the AUTOEXEC.LSL file on the CPU Station2:

```
SET STATION 2 // This is my own station number
SET ROUTINGTABLE C:\ROUTING.TXT  // Location and name of the routing table file
```

If, as shown in the example, the remote CPU is connected via CAN bus, **the following line must also be added** to AUTOEXEC.LSL:

```
SET CAN 1 BAUD 1 STATION 1 // For connections via the first CAN interface with baud rate 1 I
```

Entry into the AUTOEXEC.LSL file on the CPU Station3:

```
SET STATION 3 // This is my own station number
SET ROUTINGTABLE C:\ROUTING.TXT  // Location and name of the routing table file
SET CAN 1 BAUD 1 STATION 2 // For connections via the first CAN interface with baud rate 1 I
```

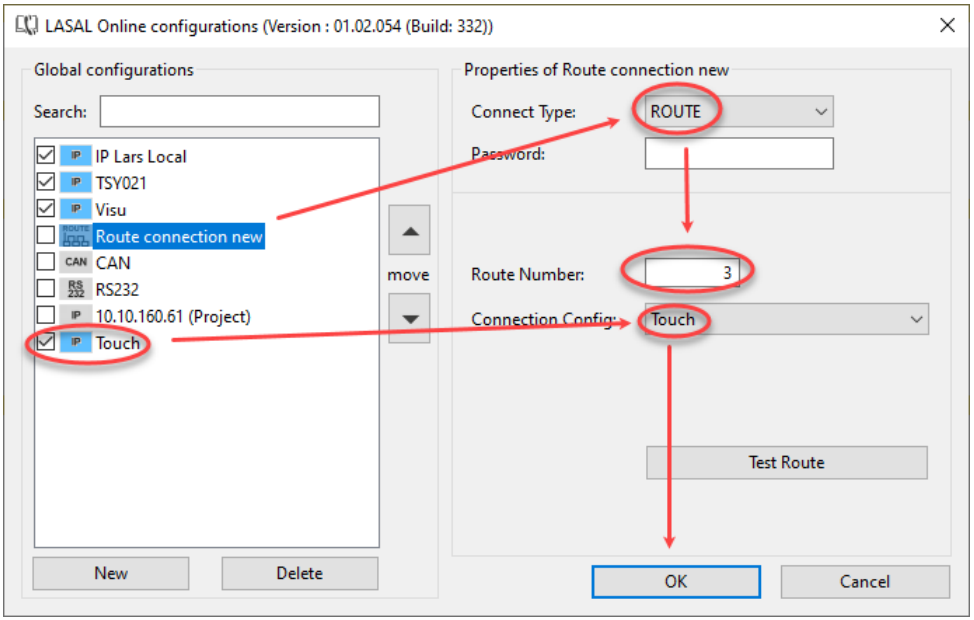


Attention!

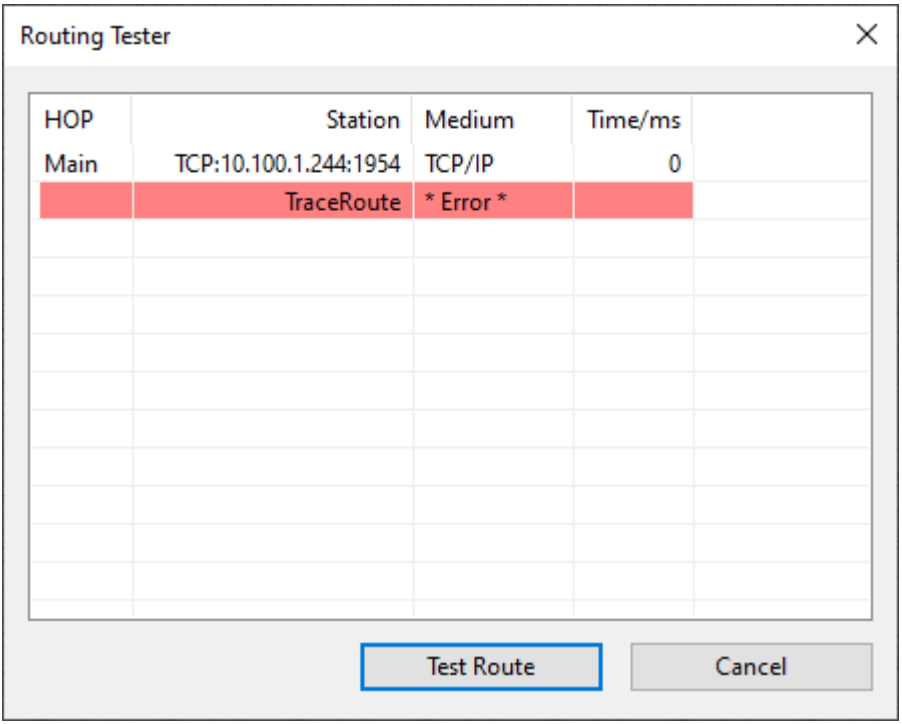
For CAN connections, **the baud rates in AUTOEXEC.LSL must be identical for both CPUs!**

The dialog

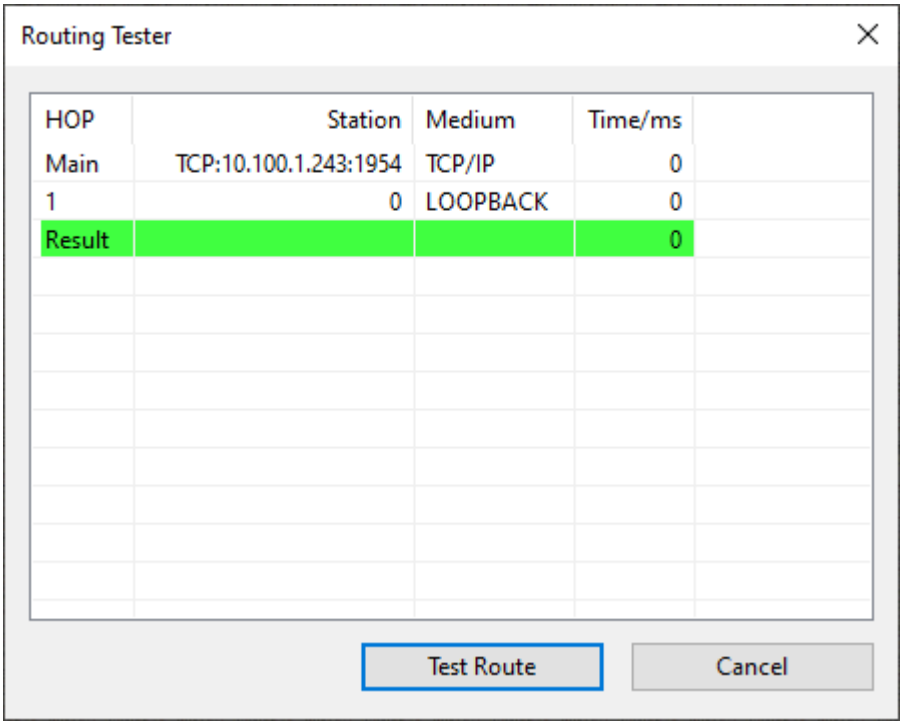
Now the connection type ROUTE can be selected in the configuration window and the destination station number (in our example this is number 3) of the routing table can be entered. In the **Connection Config** field, the address known for LASAL CLASS 2 (in our example, the address of Station1) must be entered. With the button **OK** the route - here with the name "ROUTE connection new" - is saved.



By clicking on the **Test Route** button, you can check whether the final target system can be accessed or up to where the connection works. A dialog opens in which a table without entries is displayed and the **Test Route** button should be pressed again. Thereupon the current route is tested. If there is an error somewhere, it will be **marked red** in the table:



If everything is OK, a result line with a **green mark** is output at the end:



4.1.3 LARS

Because the **LARS** program runs on its own PC, the communication over **TCP/IP** must be constructed manually. Because only limited realtime (windows dependant) and hardware access is available under LARS, it is recommended that only visualization or simulations be run.

A connection to the user computer can be made two ways:

Loopback Address

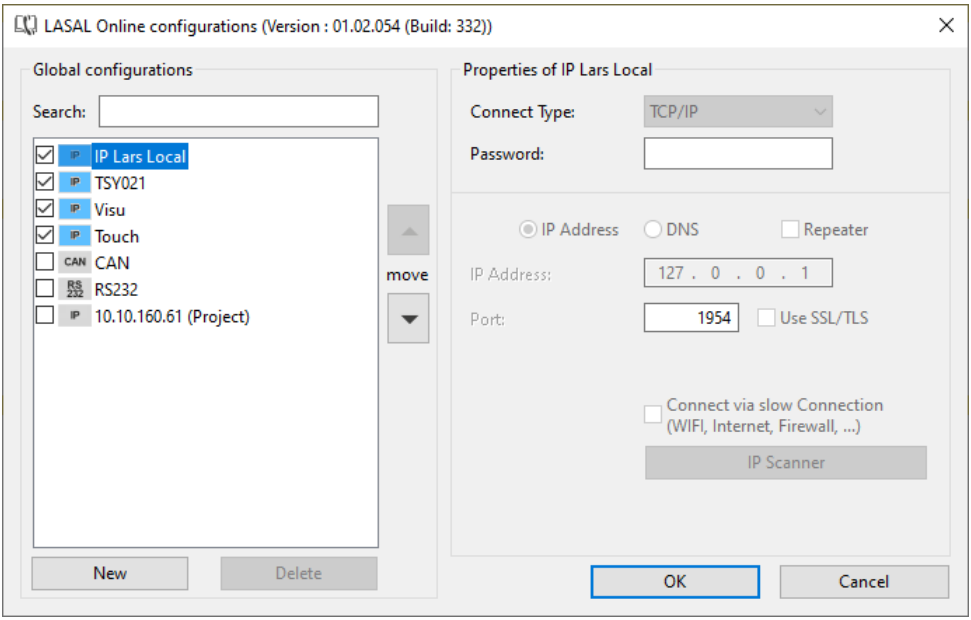
Own IP Address

4.1.3.1 Loopback Address

This is the IP address of the PC, which should always be provided when TCP/IP is used. As soon as an inquiry is made in the network for the Loopback address, the connection to its own interface is returned. The **Loopback address** is normally used for troubleshooting but in this case, it's the best solution for accessing **LARS**.

A Loopback address is an IP address, whose first number must be **127**.

In LASAL 2 the IP address **127.0.0.1** for access to LARS has been defined standardized. The **IP Lars Local** entry is fixed in the phone book and can neither be changed nor deleted:



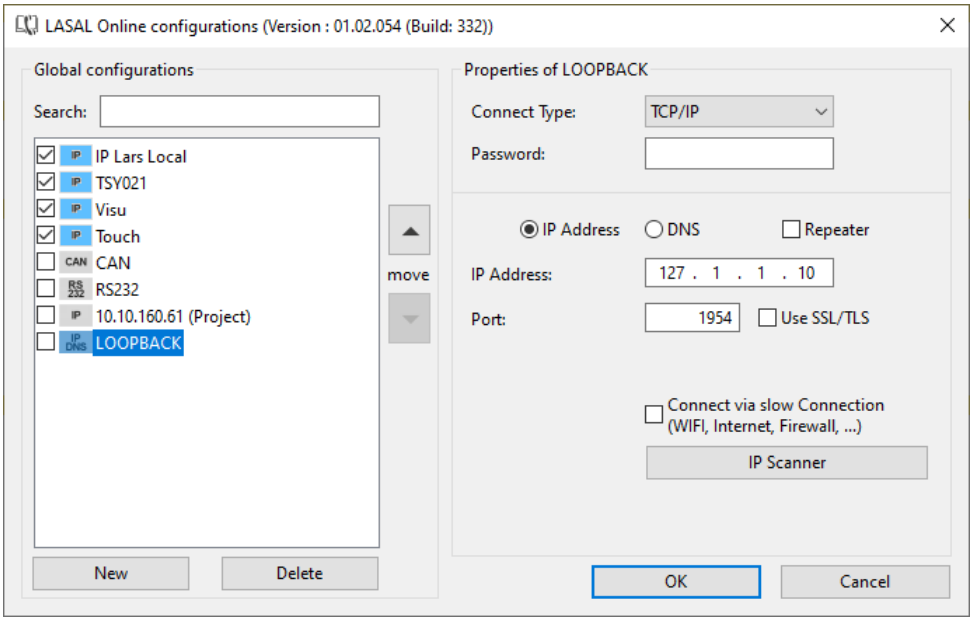
However, it can also be used another IP address:

Loopback Address:**127.x.x.x**

The x can be any value between **1** and **254**.

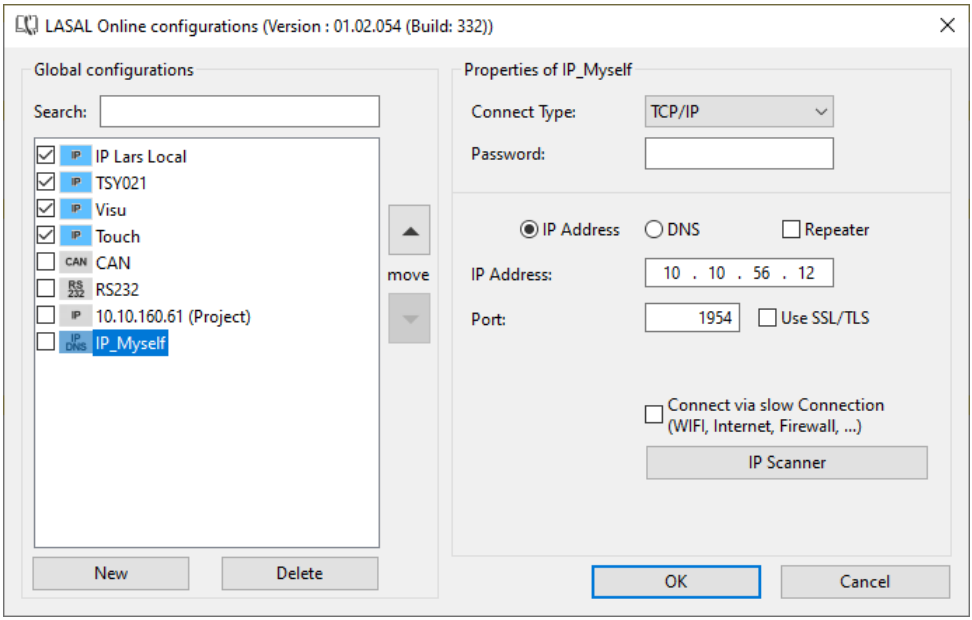
Note:

To use the Loopback address, you cannot be in a network. A network card need only be installed.



4.1.3.2 Own IP Address

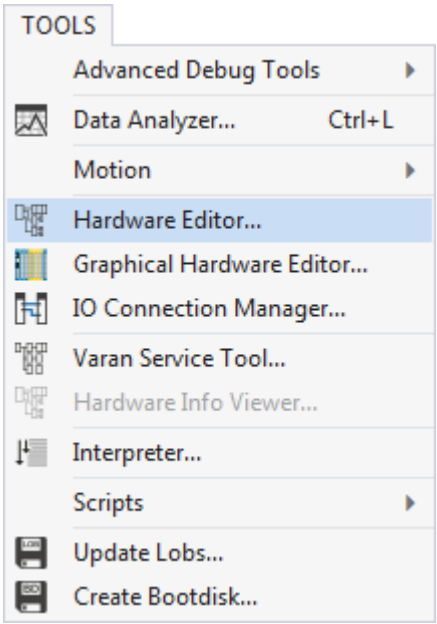
If connected to a network, this type can be used to establish a connection to **LARS**.
The users own **IP address** must be assigned.


LASAL Online configurations (Version : 01.02.054 (Build: 332))
Global configurations
Search:
IP Lars Local
TSY021
Visu
Touch
CAN
RS232
10.10.160.61 (Project)
IP_Myself
move
New
Delete
Properties of IP_Myself
Connect Type:
TCP/IP
Password:
IP Address
DNS
Repeater
IP Address:
10 . 10 . 56 . 12
Port:
1954
Use SSL/TLS
Connect via slow Connection
(WIFI, Internet, Firewall, ...)
IP Scanner
OK
Cancel

4.2 Hardware Editor

The **Hardware Editor** provides the possibility to display **available hardware** with defined hardware classes in **LASAL** using just a few mouse clicks. The **Connection Manager** is discussed in the same connection, as it belongs to the hardware editor and works very closely with it. In principle, the hardware editor and ConnectonManager can each be seen as an additional view of LASAL classes – limited to special hardware classes from the hardware library.

The hardware editor is started by pushing the  button in the toolbar or under the menu item **Tools / Hardware Editor**.



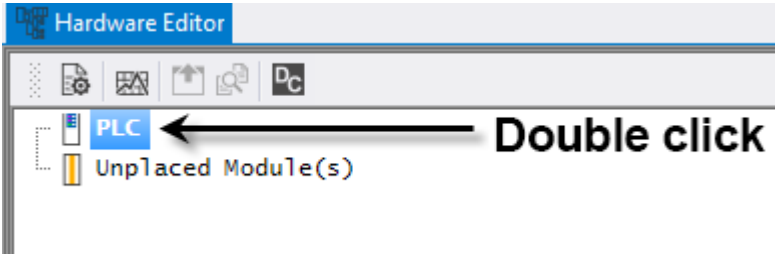
The following window then appears:



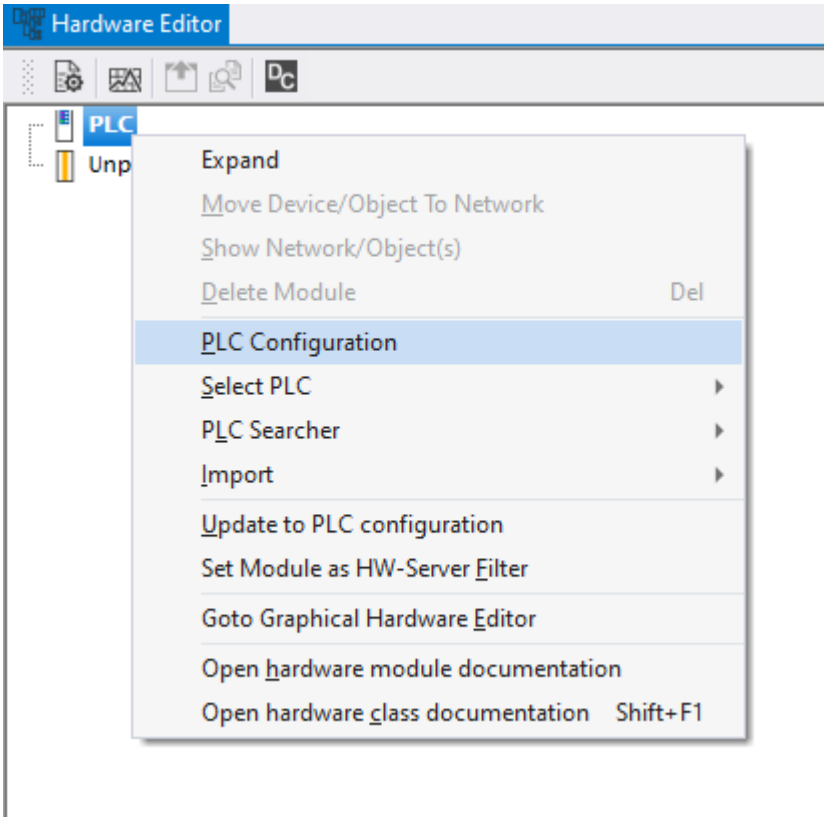
4.2.1 Configuring the PLC

In the left side of the window, the hardware tree can be seen. Currently, it consists of the control only (**PLC**). To now add to these hardware modules, it must first be **configured**.

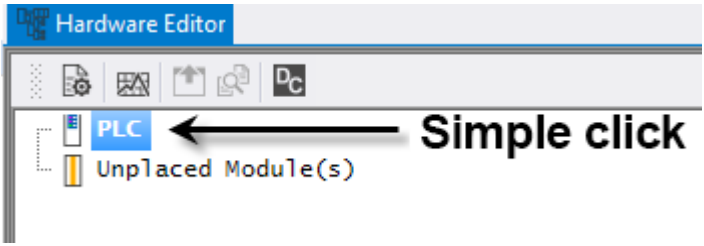
This can be achieved in three different ways: either with a double click on the PLC …



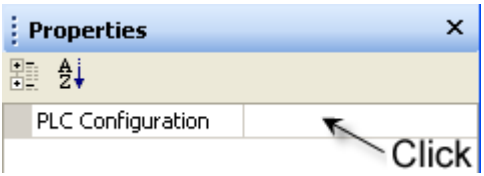
... or with a right click and selecting **PLC Configuration** from the context menu …



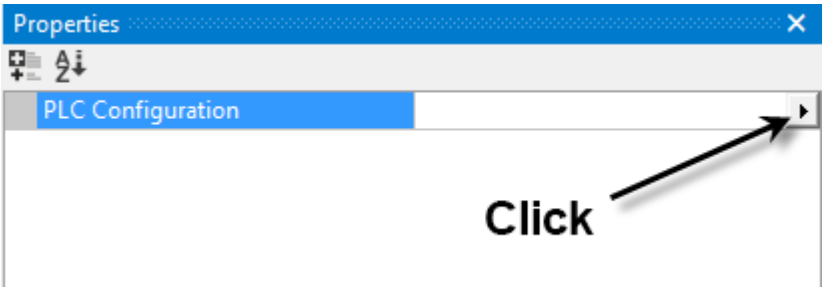
...or by clicking once on PLC …



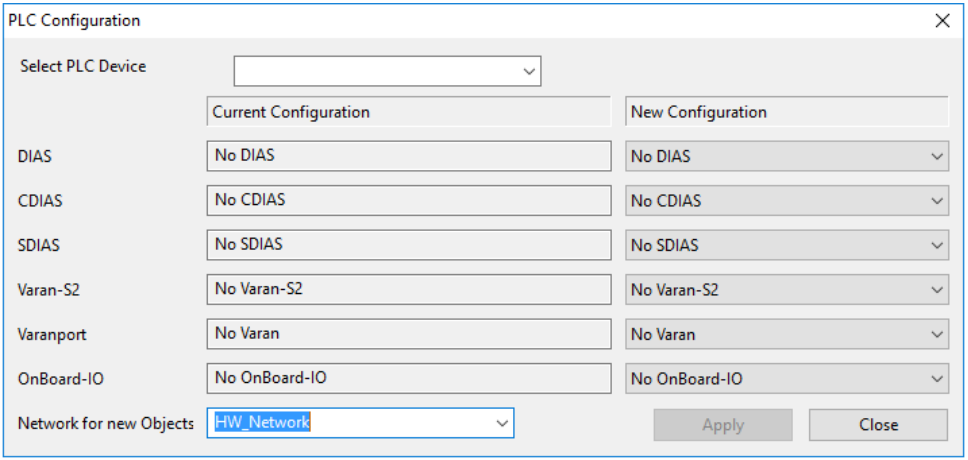
... then in the PLC field in the properties window …



... afterwards on the **More arrow** that appears.

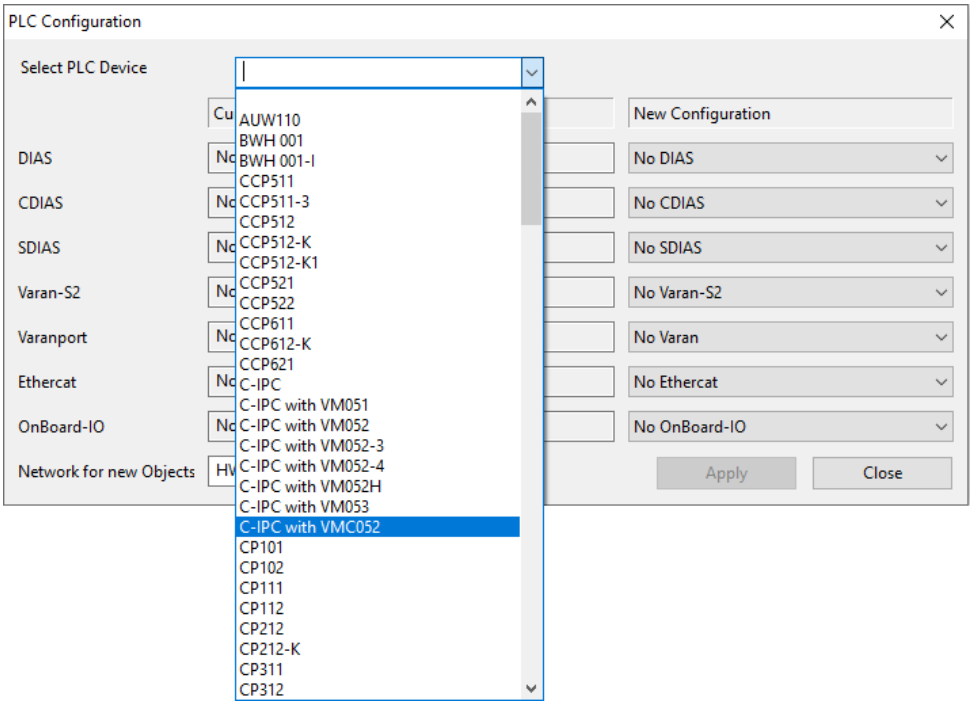


In all three cases, the same configuration dialog boy opens:

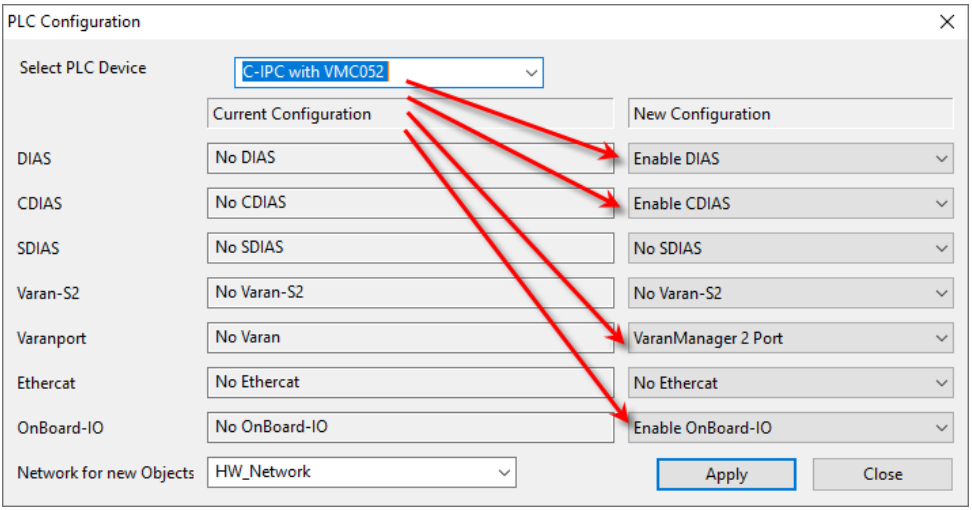


On the left, the actual status is shown (**Current Configuration**), on the right the desired status (**New Configuration**) can be seen. On the bottom left, the **Network for new Objects** field is located. From now on, all HW objects are placed in this network first (sections of the PLC such as the DIASMaster as well as all modules that can be connected). By default, **HW_Network** is entered; the name can be changed however. The buttions Apply und Close are located to the right. **Apply** can only be clicked when

something in the status has changed, with **Close** the window is exited without changes. At the top left, the Select PLC Device field is displayed. With the More button, a specific PLC type can be selected.

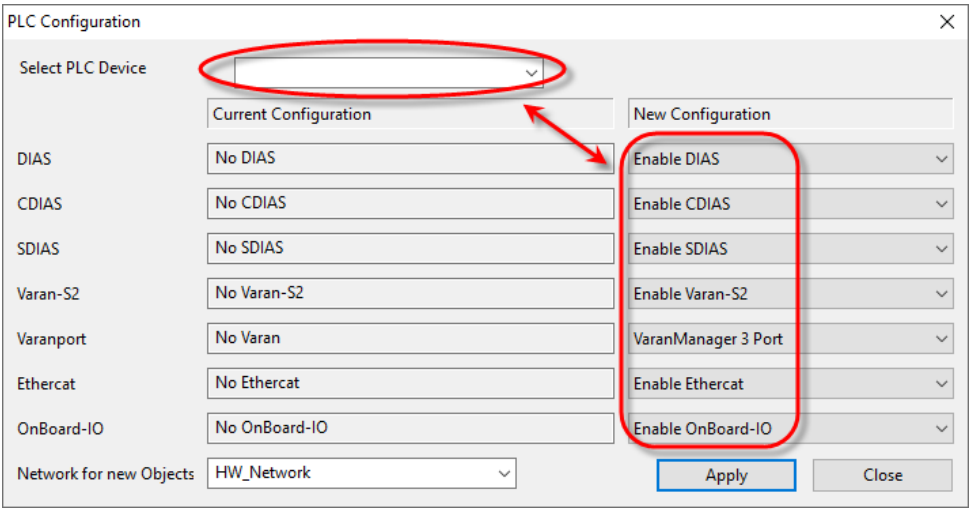


The configuration is automatically completed but can also be changed by the user.

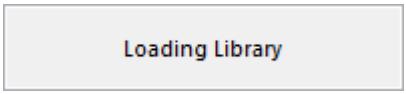


If this occurs the name in the selected PLC device field disappears since the configuration no longer matches the selected type. This process enables the use of an existing PLC type as an template.

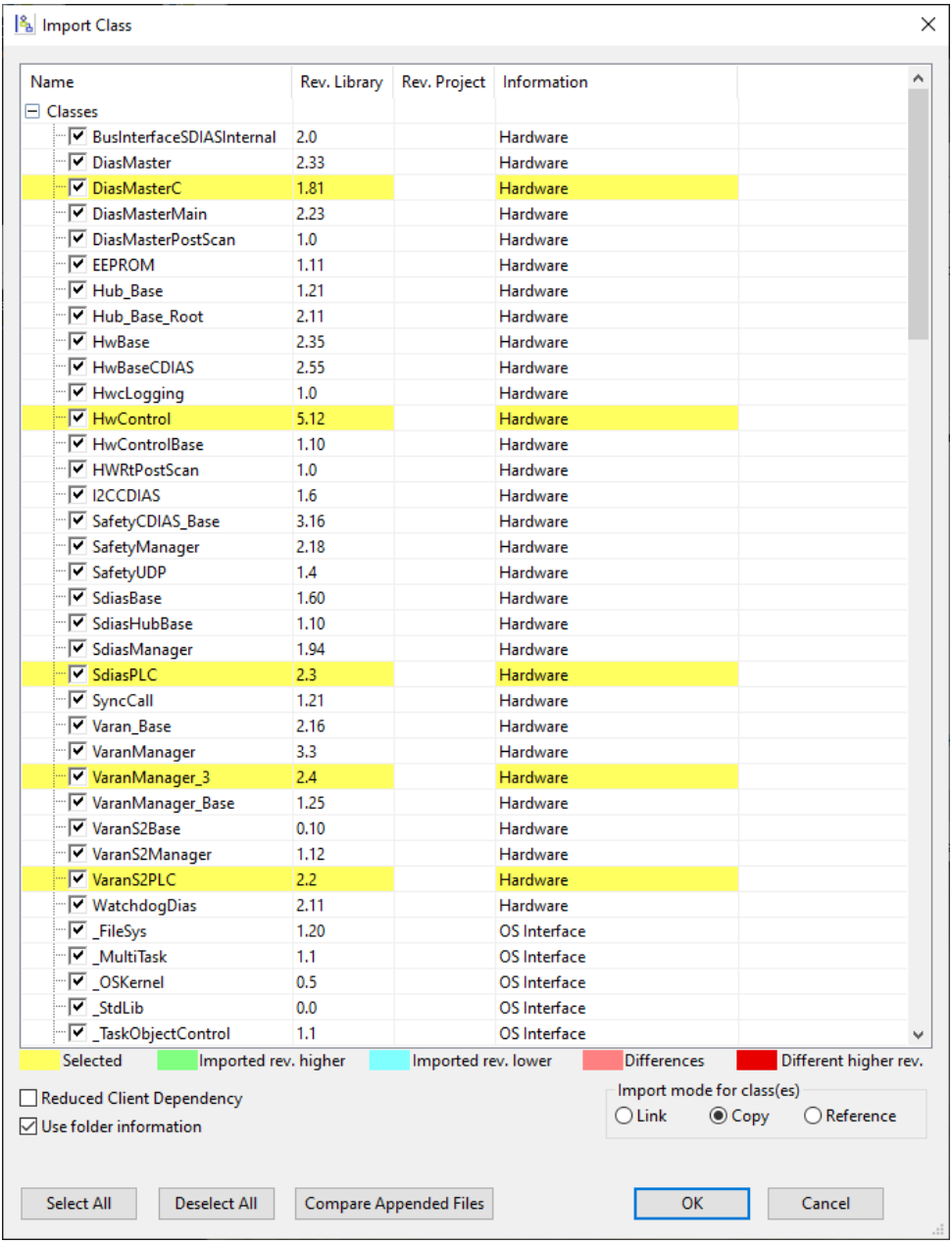
A theoretical "maximum configuration" then looks like this (whereby it is practically impossible to combine all bus types in one CPU!):



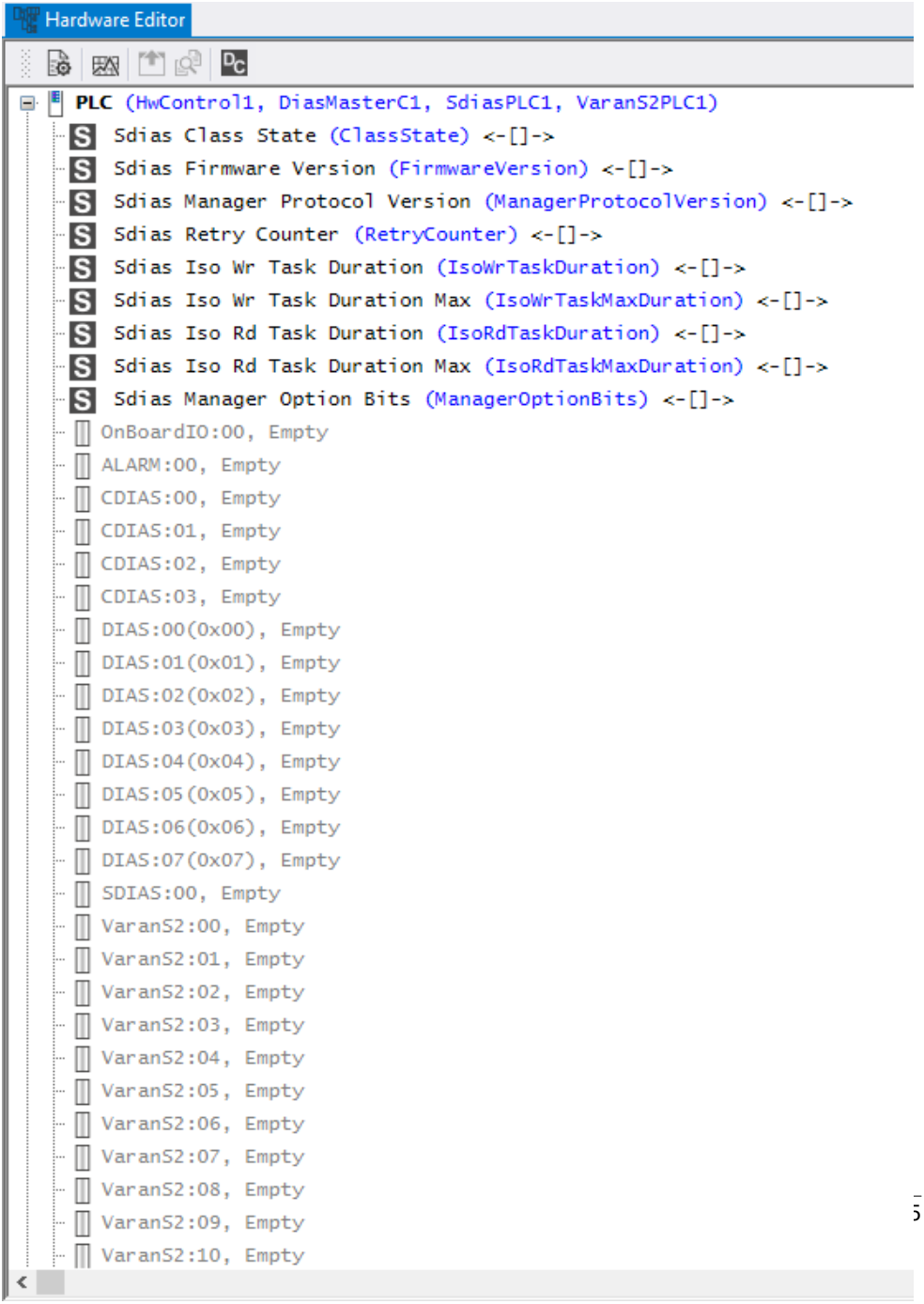
The Apply button can now be clicked. When clicked, the follow quickly appears



on the screen. Afterwards, either the module is added directly (when the flag under Extras-Optionen, **„Show Import Dialog“** is set to **false**), or the Import Class dialog (when **„Show Import Dialog“** is set to **true**).


Import Class
Name
Rev. Library
Rev. Project
Information
Classes
BusInterfaceSDIASInternal 2.0 Hardware
DiasMaster 2.33 Hardware
DiasMasterC 1.81 Hardware
DiasMasterMain 2.23 Hardware
DiasMasterPostScan 1.0 Hardware
EEPROM 1.11 Hardware
Hub_Base 1.21 Hardware
Hub_Base_Root 2.11 Hardware
HwBase 2.35 Hardware
HwBaseCDIAS 2.55 Hardware
HwcLogging 1.0 Hardware
HwControl 5.12 Hardware
HwControlBase 1.10 Hardware
HWRtPostScan 1.0 Hardware
I2CCDIAS 1.6 Hardware
SafetyCDIAS_Base 3.16 Hardware
SafetyManager 2.18 Hardware
SafetyUDP 1.4 Hardware
SdiasBase 1.60 Hardware
SdiasHubBase 1.10 Hardware
SdiasManager 1.94 Hardware
SdiasPLC 2.3 Hardware
SyncCall 1.21 Hardware
Varan_Base 2.16 Hardware
VaranManager 3.3 Hardware
VaranManager_3 2.4 Hardware
VaranManager_Base 1.25 Hardware
VaranS2Base 0.10 Hardware
VaranS2Manager 1.12 Hardware
VaranS2PLC 2.2 Hardware
WatchdogDias 2.11 Hardware
_FileSys 1.20 OS Interface
_MultiTask 1.1 OS Interface
_OSKernel 0.5 OS Interface
_StdLib 0.0 OS Interface
_TaskObjectControl 1.1 OS Interface
Selected
Imported rev. higher
Imported rev. lower
Differences
Different higher rev.
Reduced Client Dependency
Use folder information
Import mode for class(es)
Link
Copy
Reference
Select All
Deselect All
Compare Appended Files
OK
Cancel

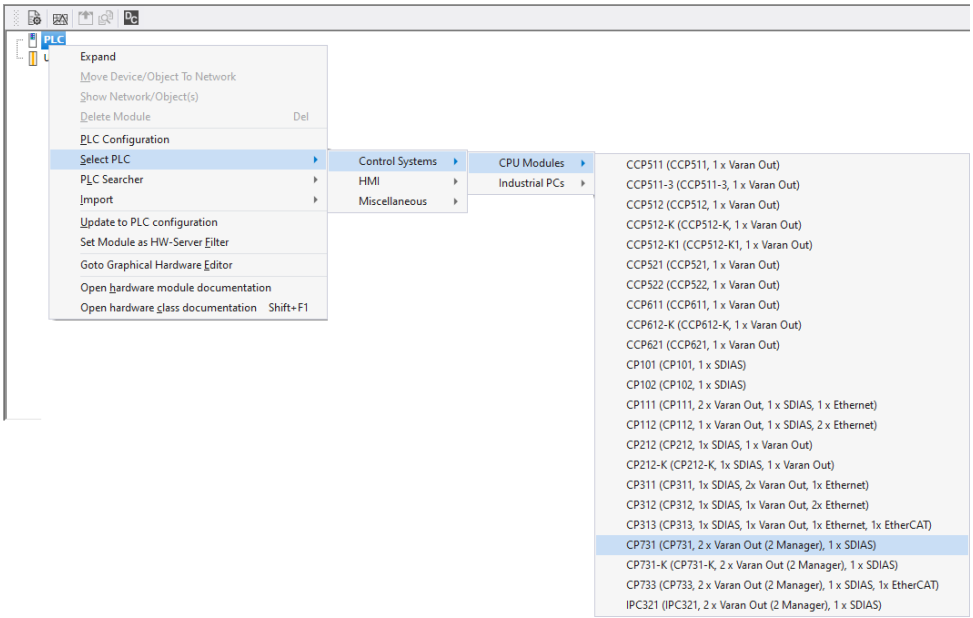
Here, the decision can be made to link, copy or reference classes and much more. For a complete description of the **Import Class dialog**, see the chapter **Libraries**. Once the Import dialog is confirmed with **OK -** or is not displayed, because **Show Import Dialog** is set to **false -** the hardware editor window appears as below (here in the configuration with **OnBoardIO**, **DIAS/CDIAS** and **3 Varan ports**):


Hardware Editor
PLC (HwControl1, DiasMasterC1, SdiasPLC1, VaranS2PLC1)
Sdias Class State (ClassState) <-[]->
Sdias Firmware Version (FirmwareVersion) <-[]->
Sdias Manager Protocol Version (ManagerProtocolVersion) <-[]->
Sdias Retry Counter (RetryCounter) <-[]->
Sdias Iso Wr Task Duration (IsoWrTaskDuration) <-[]->
Sdias Iso Wr Task Duration Max (IsoWrTaskMaxDuration) <-[]->
Sdias Iso Rd Task Duration (IsoRdTaskDuration) <-[]->
Sdias Iso Rd Task Duration Max (IsoRdTaskMaxDuration) <-[]->
Sdias Manager Option Bits (ManagerOptionBits) <-[]->
OnBoardIO:00, Empty
ALARM:00, Empty
CDIAS:00, Empty
CDIAS:01, Empty
CDIAS:02, Empty
CDIAS:03, Empty
DIAS:00(0x00), Empty
DIAS:01(0x01), Empty
DIAS:02(0x02), Empty
DIAS:03(0x03), Empty
DIAS:04(0x04), Empty
DIAS:05(0x05), Empty
DIAS:06(0x06), Empty
DIAS:07(0x07), Empty
SDIAS:00, Empty
VaranS2:00, Empty
VaranS2:01, Empty
VaranS2:02, Empty
VaranS2:03, Empty
VaranS2:04, Empty
VaranS2:05, Empty
VaranS2:06, Empty
VaranS2:07, Empty
VaranS2:08, Empty
VaranS2:09, Empty
VaranS2:10, Empty

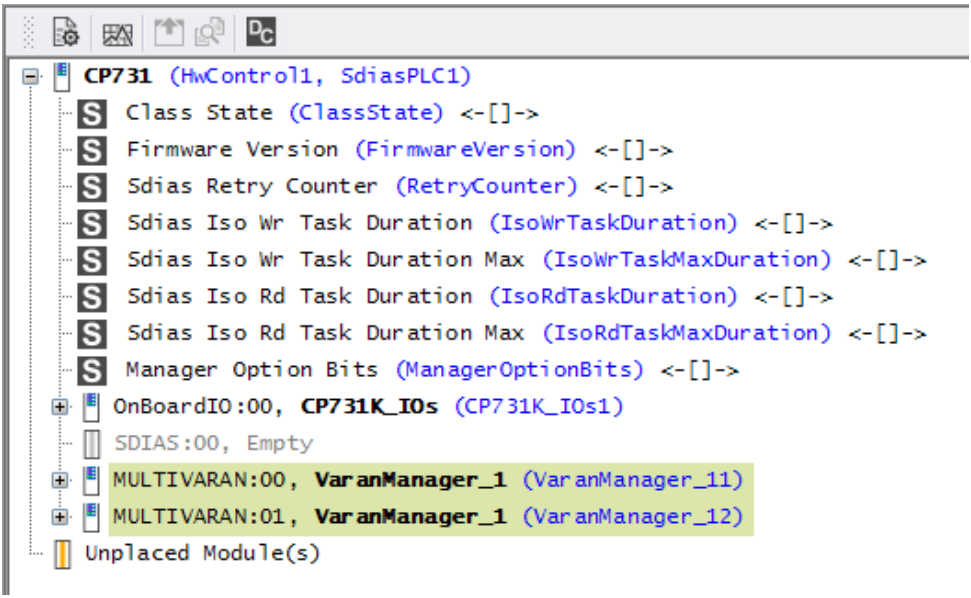
4.2.1.1 Using 2 VARAN Managers in LASAL Class 2

Creating 2 VaranManagers via the selection of a PLC with multiple VARANManagers

Example of a selected PLC with 2 Varan Managers:

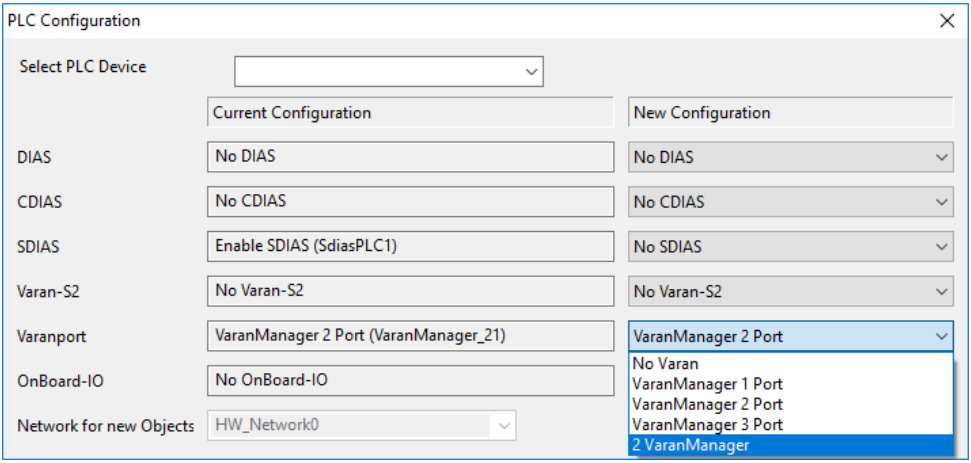


After the PLC is selected, it is displayed in the hardware editor accordingly.



Creating 2 VaranManagers via the Configuration Dialog

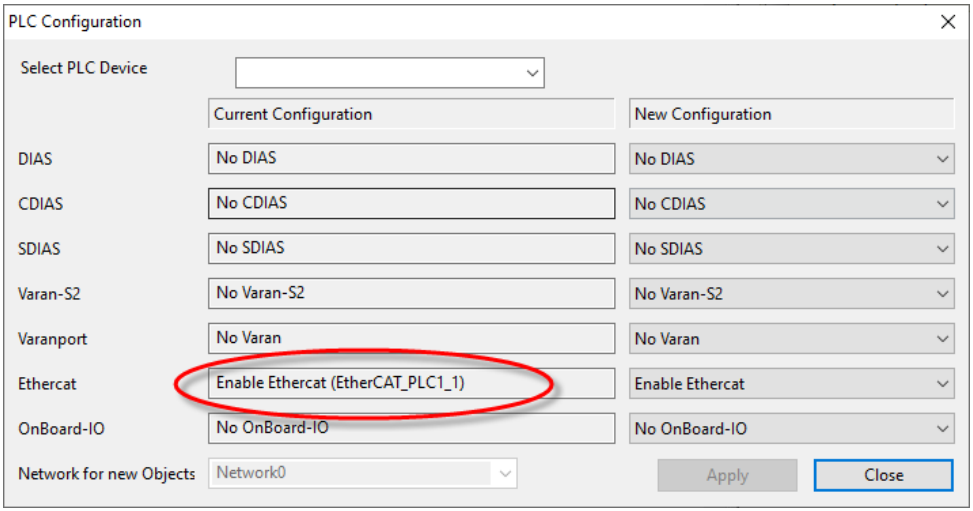
Example of the selection via the Configuration dialog:



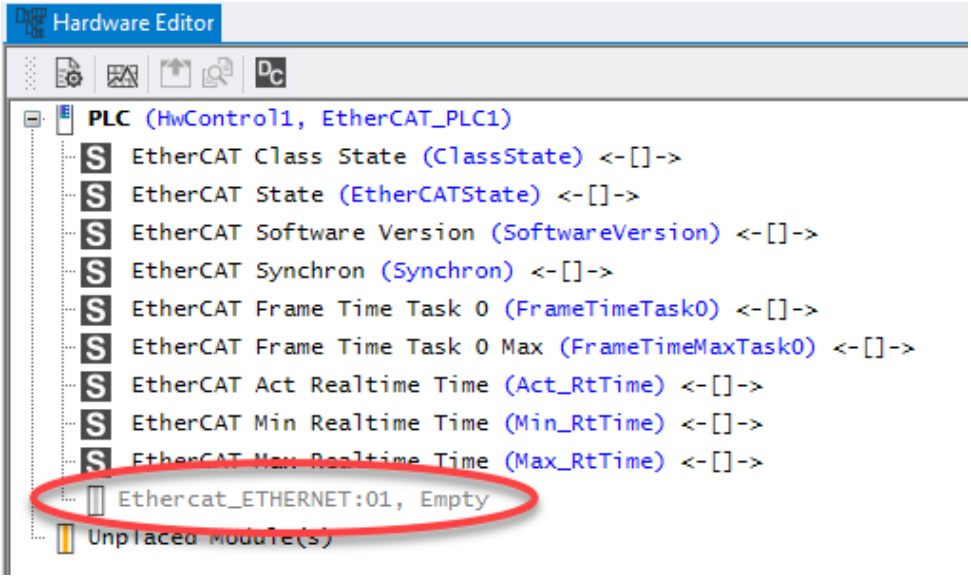
4.2.2 EtherCAT

Beside the Varan bus, S-DIAS, C-DIAS and DIAS there is another special case whose configuration differs a little from the other systems: **EtherCAT**.

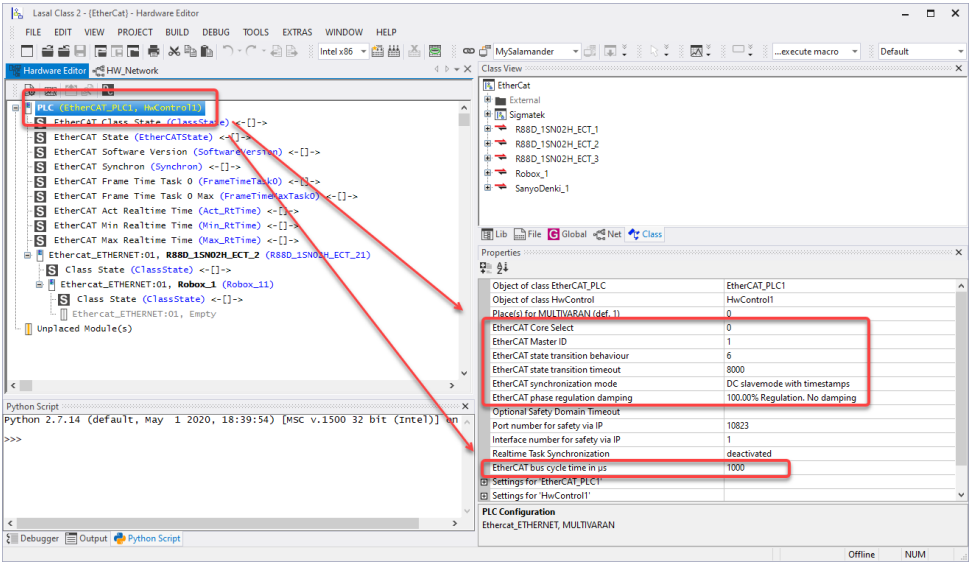
There are PLCs that have an EtherCAT interface. If the PLC is configured so that EtherCAT is allowed...



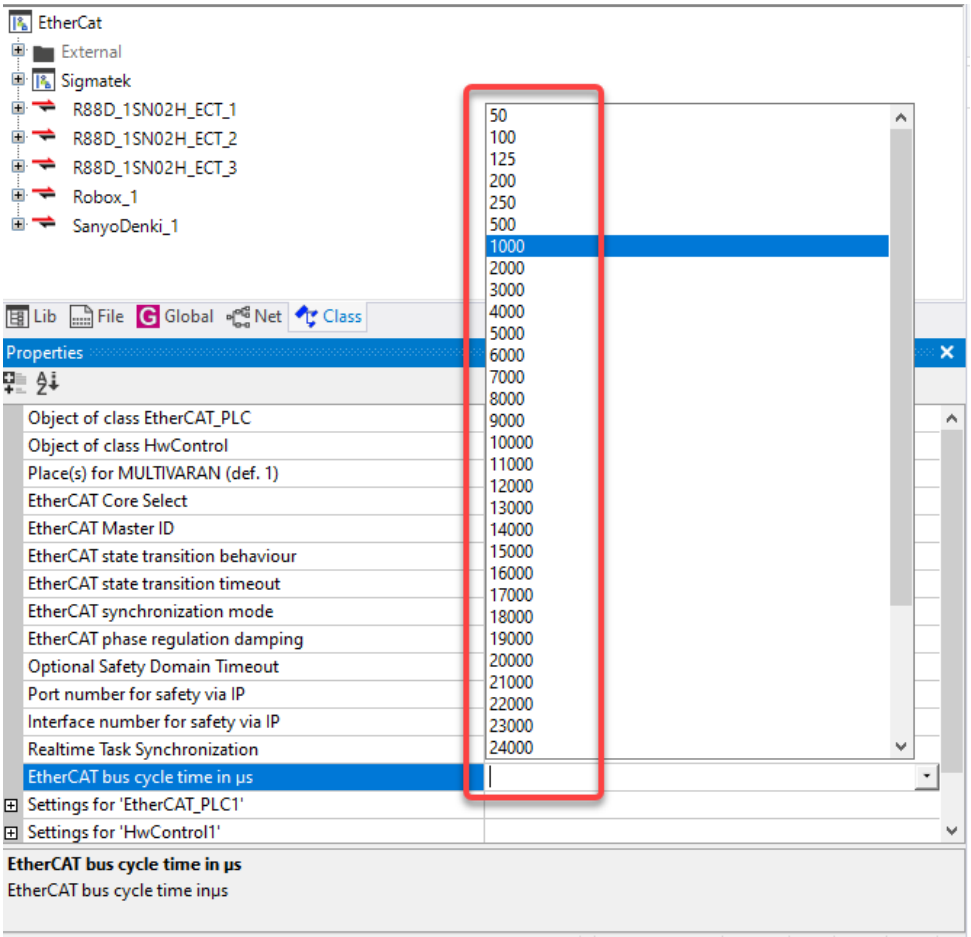
...you get a PLC with an **EtherCAT interface**:



In the properties of the PLC additional fields become visible:

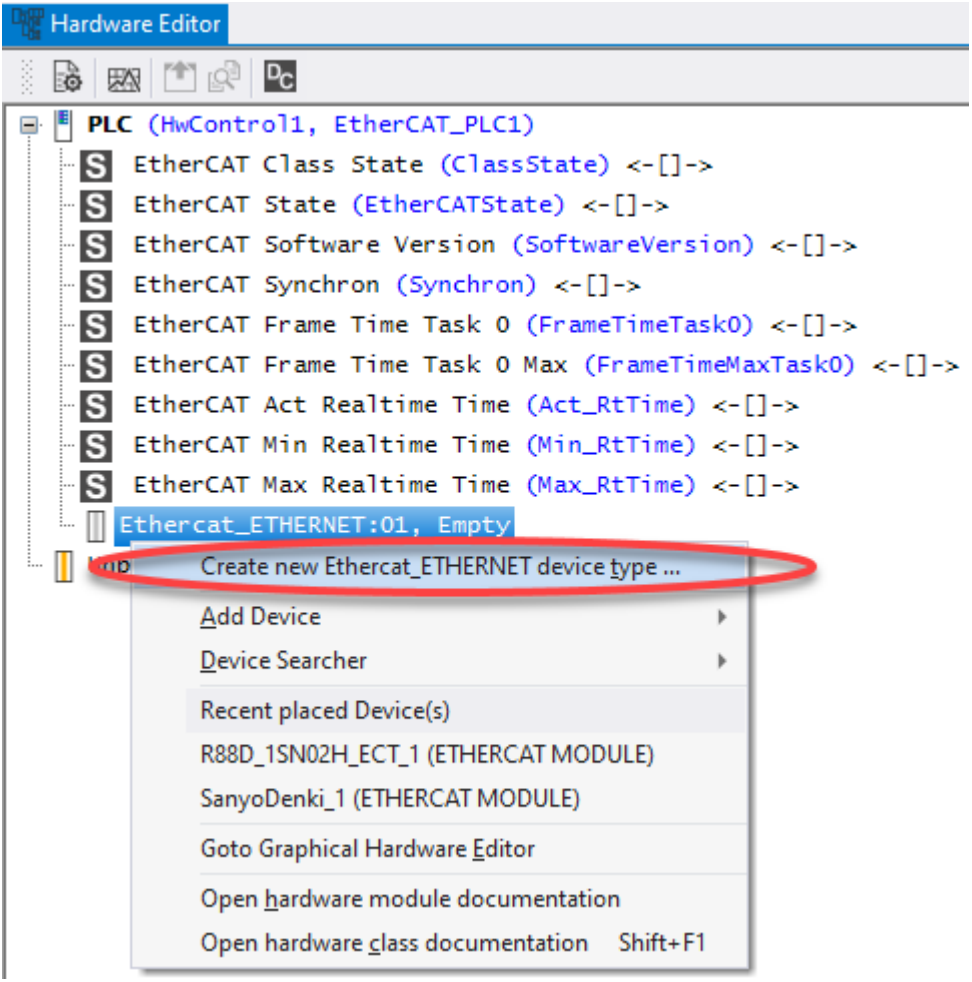


In the field **EtherCAT bus cycle time in µs** the bus cycle time for the PDO data transfer can be set. Only values from the list can be selected:

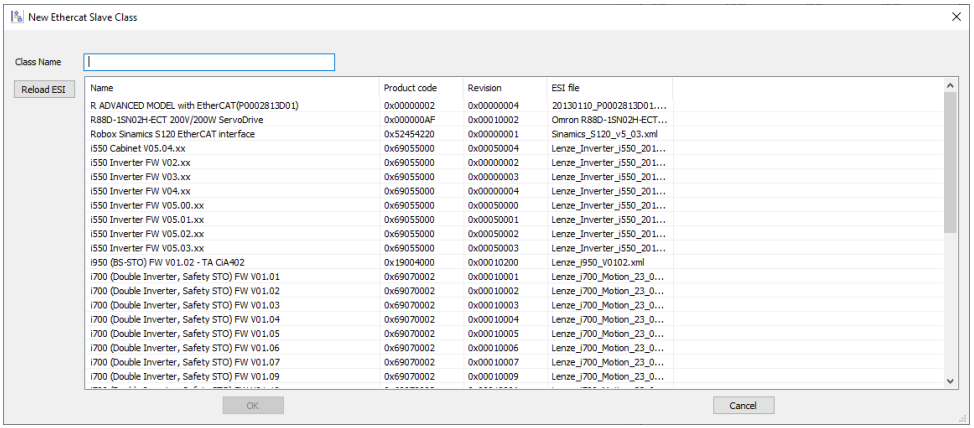


Further information regarding the other EtherCAT fields can be found in the HW class documentation.

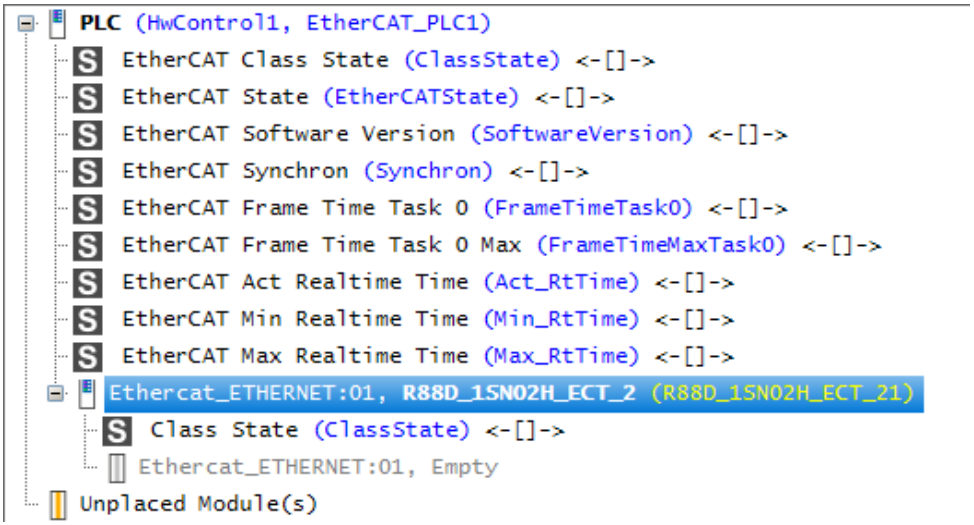
With a right click on the empty interface you will see a new menu item: **Create new Ethercat_ETHERNET device type**.



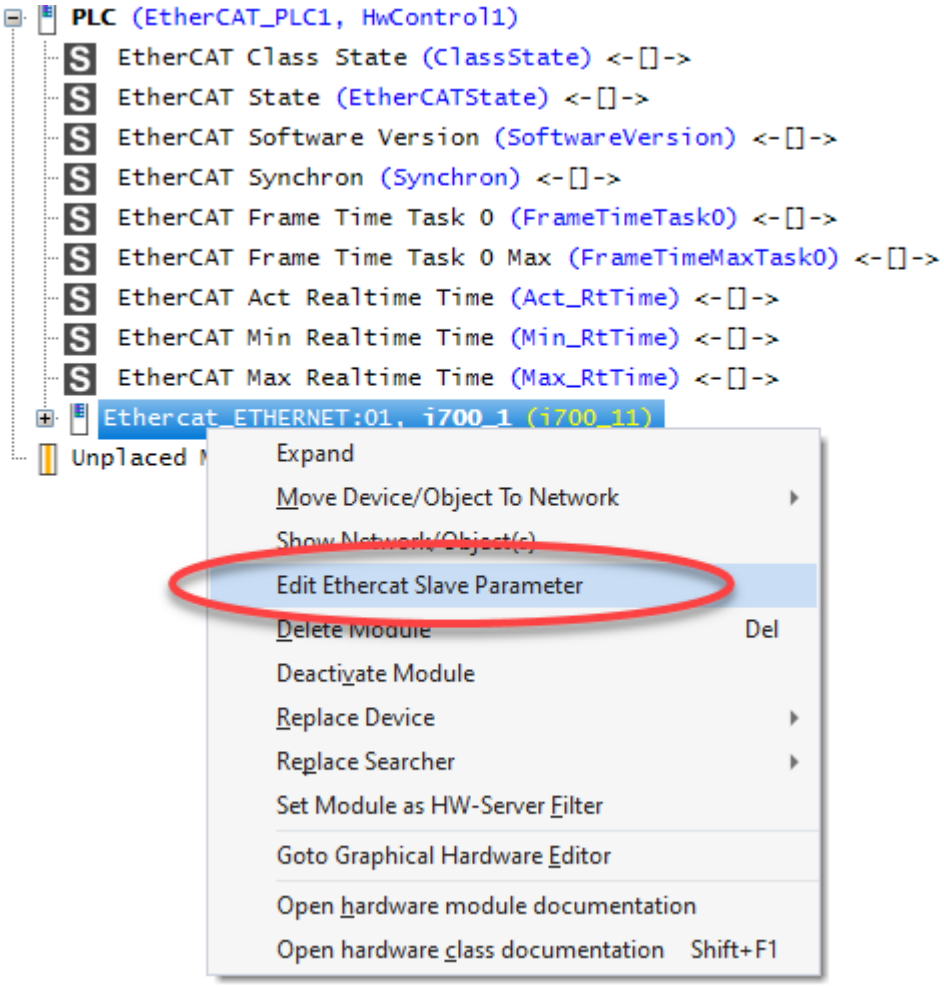
Selecting this menu item displays the **New Ethercat Slave Class** dialog:



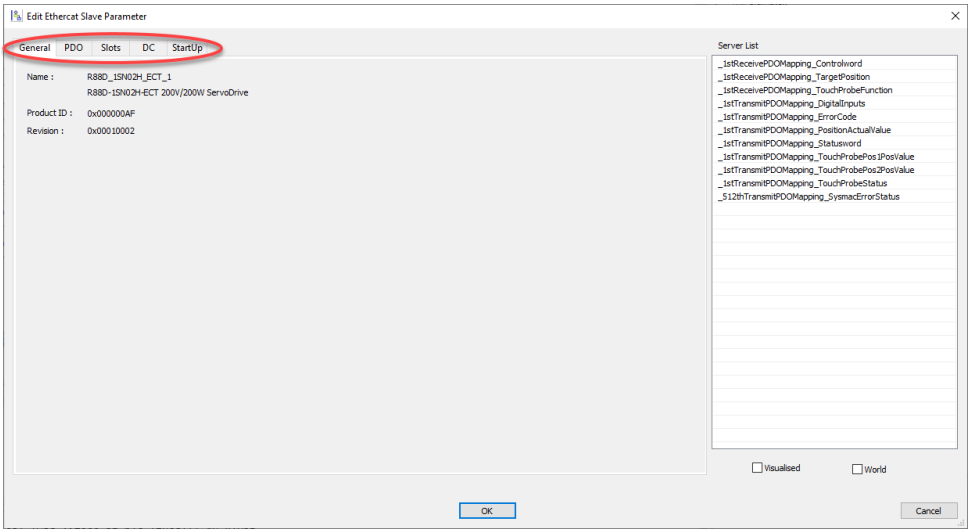
This means that a new slave class is **not searched via Device Searcher**, but a **new slave class** is added to the project from this list of preconfigured devices.



The **parameterization** of the individual EtherCAT modules is done by right-clicking on the EtherCAT module and selecting the menu item **Edit Ethercat parameter**:

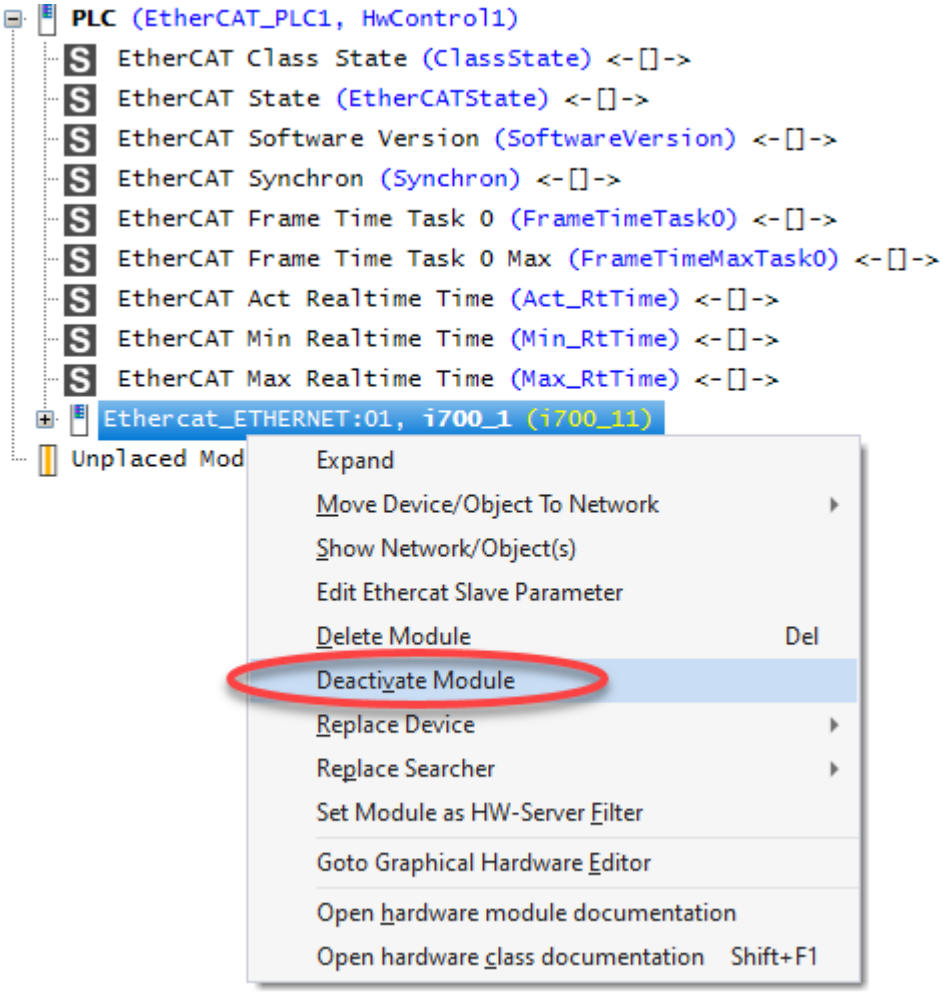


The resulting **Edit Ethercat Slave Parameter** dialog consists of up to five tabs, depending on the type of module and whether the corresponding information is available on the module:

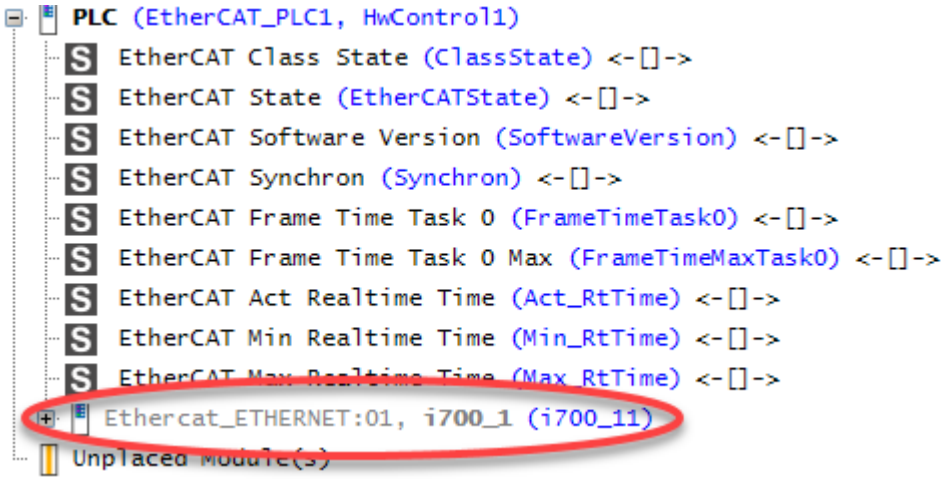


Once all desired changes have been made on all tabs, the new server list can be confirmed at any time in the **Edit Ethernet Slave Parameter** dialog by clicking on the **OK** button, or canceled with **Cancel** and the previous status is restored.

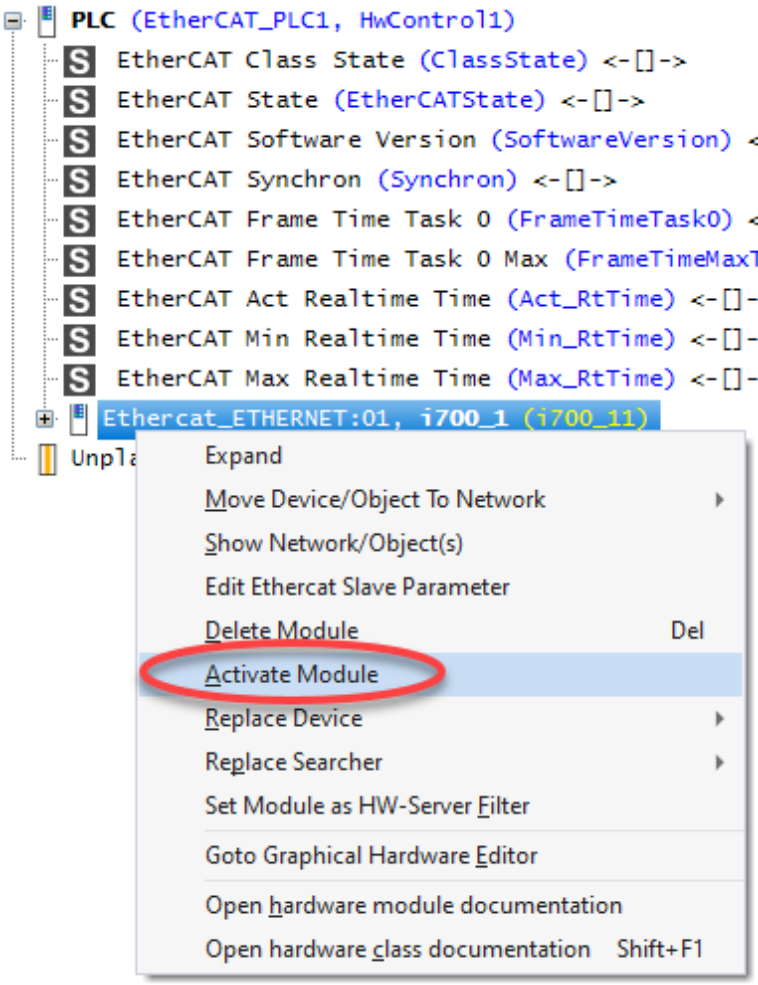
EtherCAT modules can be deactivated like other modules via the context menu with the menu item **Deactivate Module**:



An inactive module is displayed in gray:



To reactivate the module, select **Activate Module** from the context menu:


PLC (EtherCAT_PLC1, HwControl1)
EtherCAT Class State (ClassState) <-[]->
EtherCAT State (EtherCATState) <-[]->
EtherCAT Software Version (SoftwareVersion) <-[]->
EtherCAT Synchron (Synchron) <-[]->
EtherCAT Frame Time Task 0 (FrameTimeTask0) <-[]->
EtherCAT Frame Time Task 0 Max (FrameTimeMaxTask0) <-[]->
EtherCAT Act Realtime Time (Act_RtTime) <-[]->
EtherCAT Min Realtime Time (Min_RtTime) <-[]->
EtherCAT Max Realtime Time (Max_RtTime) <-[]->
Ethercat_ETHERNET:01, i700_1 (i700_11)
Unpl
Expand
Move Device/Object To Network
Show Network/Object(s)
Edit Ethercat Slave Parameter
Delete Module
Del
Activate Module
Replace Device
Replace Searcher
Set Module as HW-Server Filter
Goto Graphical Hardware Editor
Open hardware module documentation
Open hardware class documentation
Shift+F1

4.2.2.1 EtherCAT - Feature Übersicht

The EtherCAT Master meets all necessary requirements for a class A master. The following is an overview of the EtherCAT Master features defined according to ETG.1500.

Feature name	Short description	Feature ID	Supported
Basic Features			
Service Commands	Support of all commands	101	?
IRQ field in datagram	Use IRQ information from Slave in datagram header	102	-
Slaves with Device Emulation	Support Slaves with and without application controller	103	?
EtherCAT State Machine	Support of ESM special behavior	104	?
Error Handling	Checking of network or slave errors, e.g. Working Counter	105	?
VLAN	Support VLAN Tagging	106	-
EtherCAT Frame Types	Support EtherCAT Frames	107	?
UDP Frame Types	Support UDP Frames	108	-
Process Data Exchange			
Cyclic PDO	Cyclic process data exchange	201	?
Multiple Tasks	Different cycle tasks Multiple update rates for PDO	202	?
Frame repetition	Send cyclic frames multiple times to increase immunity	203	-
Network Configuration			
Online scanning	Network configuration functionality included in EtherCAT Master	301	?
Reading ENI	Network Configuration taken from ENI file		
Compare Network configuration	Compare configured and existing network configuration during boot-up	302	?
Explicit Device Identification	Identification used for Hot Connect and prevention against cable swapping	303	?
Station Alias Addressing	Support configured station alias in slave, i.e. enable 2nd Address and use it	304	?
Access to EEPROM	Support routines to access EEPROM via ESC register	305	?
Mailbox Support			

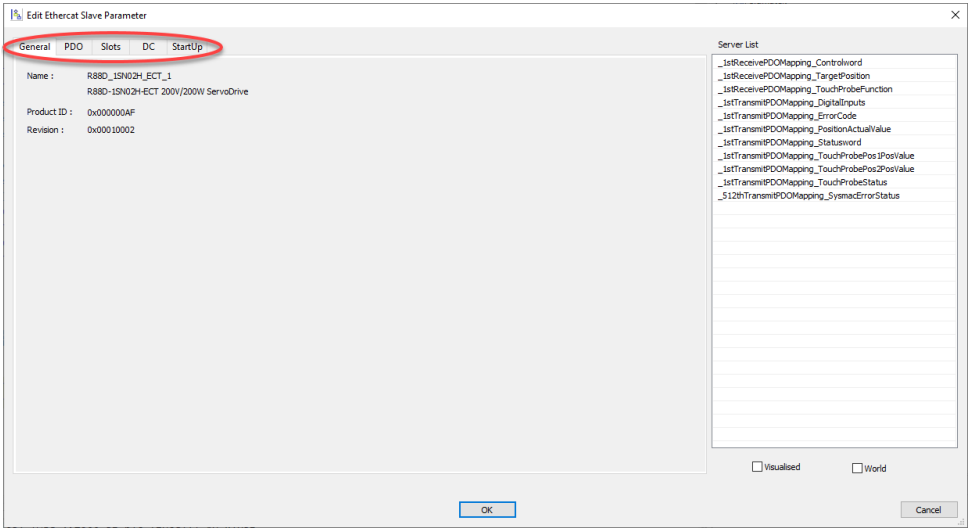
Support Mailbox	Main functionality for mailbox transfer	401	?
Mailbox resilient Layer	Support underlying resilient layer	402	?
Multiple Mailbox channels		403	?
Mailbox polling	Polling Mailbox state in slaves	404	?
CAN application layer over EtherCAT (CoE)			
SDO Up/Download	Normal and expedited transfer	501	?
Segmented Transfer	Segmented transfer	502	?
Complete Access	Transfer the entire object (with all sub-indices) at once	503	?
SDO Info service	Services to read object dictionary	504	?
Emergency Message	Receive Emergency messages	505	-
PDO in CoE	PDO services transmitted via CoE	506	?
Ethernet over EtherCAT (EoE)			
EoE protocol	Services for tunneling Ethernet frames. includes all specified EoE services	601	-
Virtual Switch	Virtual Switch functionality	602	-
EoE Endpoint to Operation Systems	Interface to the Operation System on top of the EoE layer	603	-
File access over EtherCAT (FoE)			
FoE Protocol	Support FoE Protocol	701	-
Firmware Up- /Download	Password, FileName should be given by the application	702	-
Boot State	Support Boot-State for Firmware Up/Download	703	-
Serverdrive-Profile over EtherCAT (SoE)			
SoE Services	Support SoE Services	801	-
ADS over EtherCAT (AoE)			
AoE Protocol	Support AoE Protocol	901	-
Vendor over EtherCAT (VoE)			
VoE Protocol	External Connectivity supported	1001	-
Synchronization with Distributed Clock (DC)			
DC support	Support of Distributed Clock	1101	?

Continuous Propagation Delay compensation	Continuous Calculation of the propagation delay	1102	?
Sync window monitoring	Continuous monitoring of the Synchronization difference in the slaves	1103	?
Slave-to-Slave Communication			
via Master	Information is given in ENI file or can be part of any other network configuration Copying of the data can be handled by master stack or master's application	1201	-
Master information			
Master Object Dictionary	Support of Master Object Dictionary (ETG.5001 MDP sub profile 1100)	1301	-

? is supported

– is not supported

4.2.2.2 Edit Ethercat Slave Parameter Dialog



On the **right side** - **independent of the tab** - **the current server list** is always displayed. The **Visualized** and **World** checkboxes cause **the corresponding flags to be set for all servers of this class**.

On the left side you can see the details of the respective tab. The possible tabs (depending on the module not all are available) are:

General

PDO (Process Data Objects)

Slots

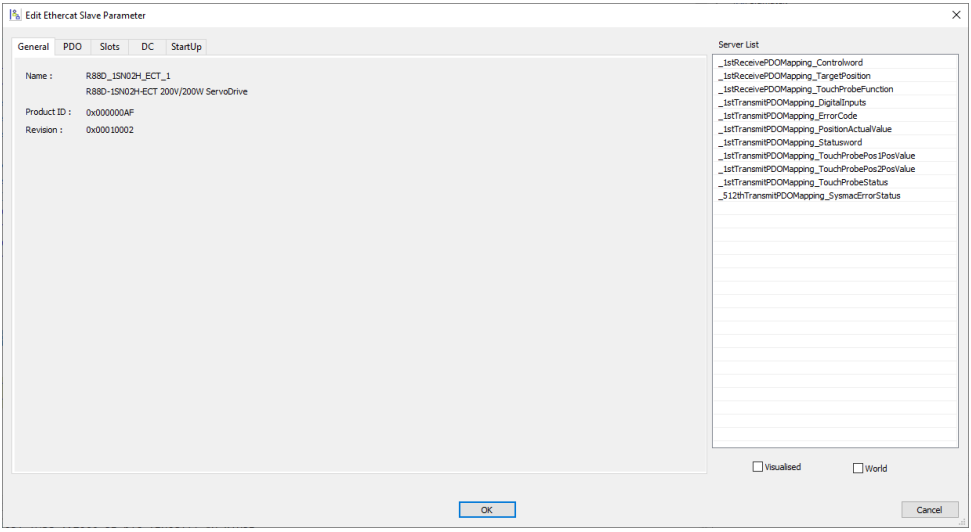
DC (Distributed Clock)

StartUp

Attention

The **General** tab is a display panel only. The **PDO** and **Slots** tabs **change the class**, the **DC** and **StartUp** tabs **change the object**!

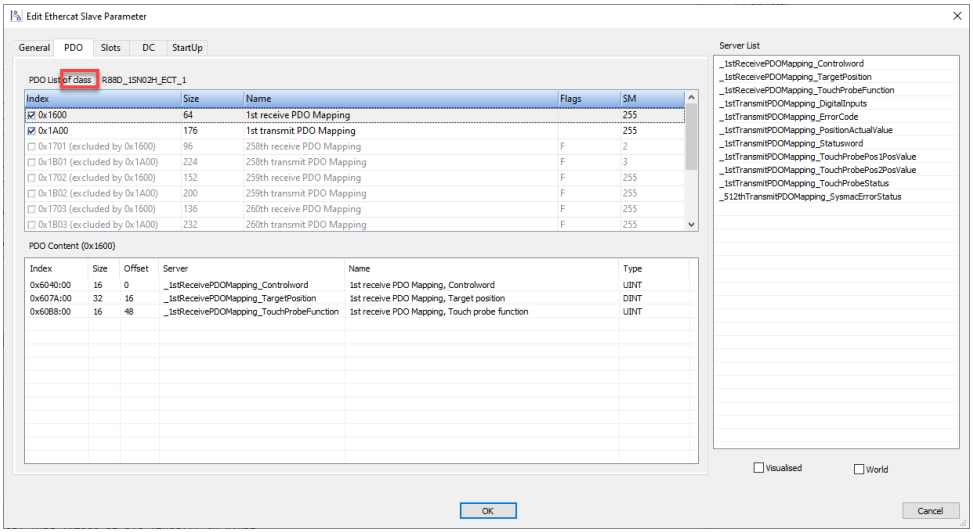
4.2.2.21 General



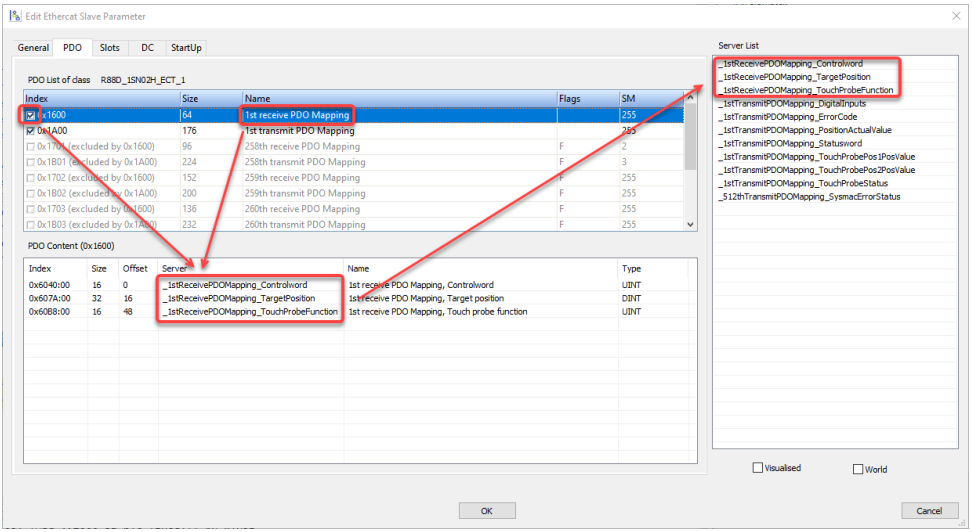
The first tab, **General**, is - as shown above - a **pure display tab**, here general information is displayed on the left side.

4.2.2.22 PDO

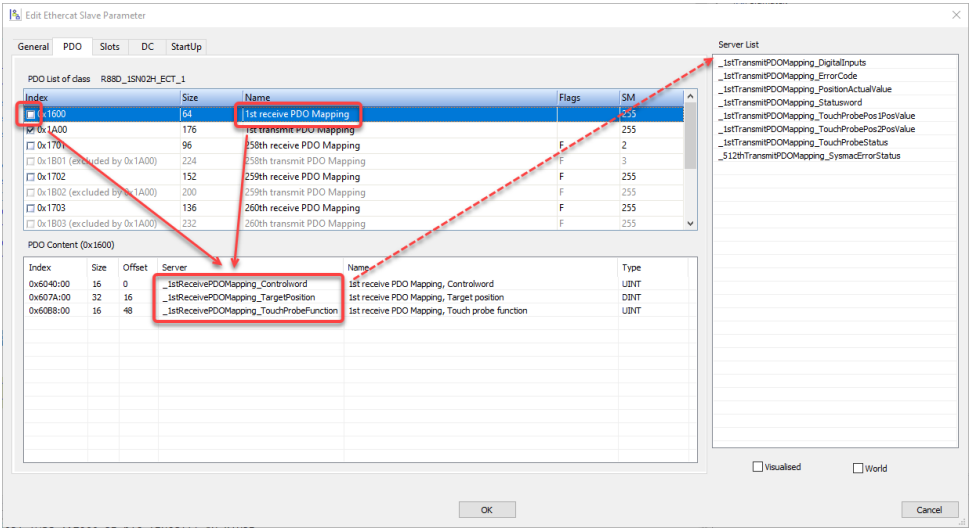
In the second tab, **PDO** (**P**rocess **D**ata **O**bjects), parts of the server list configuration can be enabled or disabled at once:



On the **PDO** tab there are two tables on the left side, one below the other. At the top the **PDO List**table, in which all **input/output sets that can be switched on and off** are listed; at the bottom in the **PDO Content** table the **content** of the PDO entry selected above is displayed. In the upper table there is a **checkbox** in front of each entry - with this checkbox the input-output set can be **switched on or off** . In the following example, the PDO named **1st receive PDO Mapping** contains three outputs, which can be activated...



...or deactivated by clicking the corresponding checkbox.

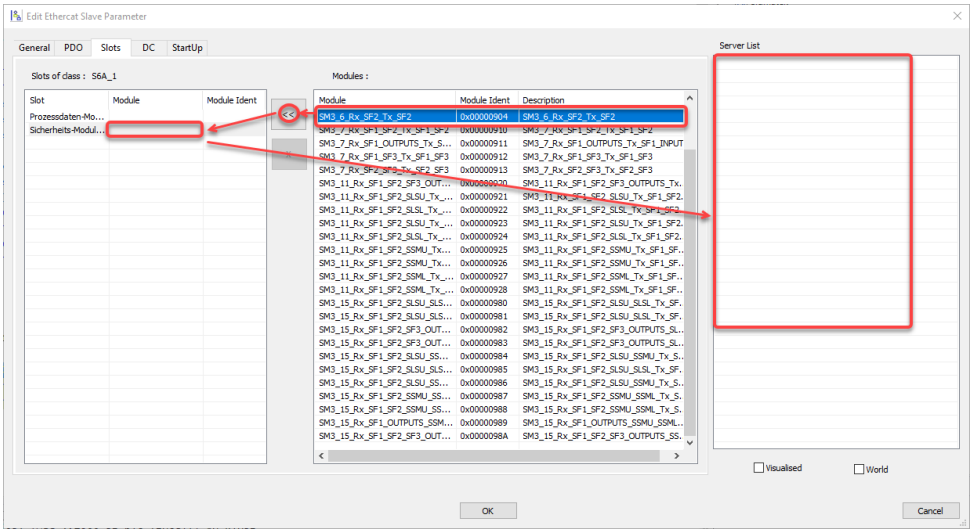


Attention

All changes made on this tab affect the class and all its objects!

4.2.2.23 Slots

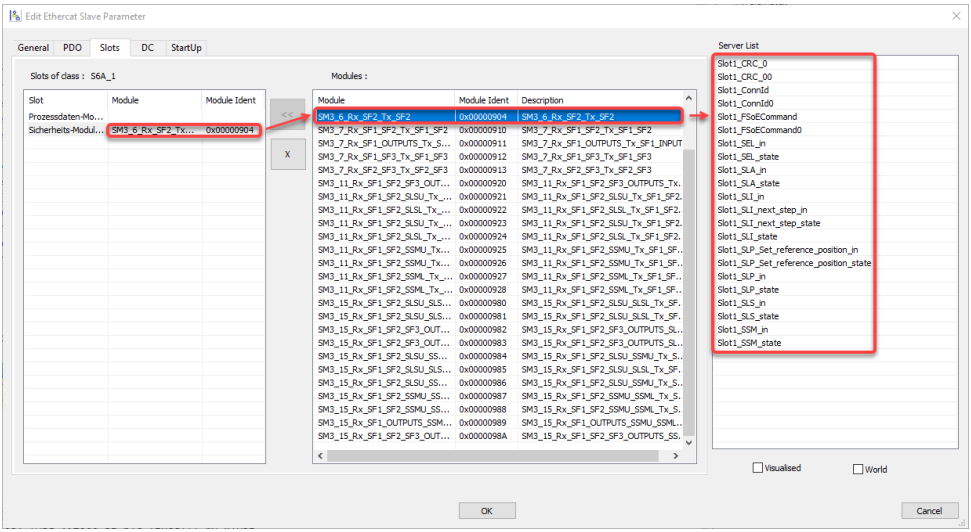
The **Slots** tab is only displayed for EtherCAT modules to which **external modules** can be connected. On this card an **external module** can be **connected to the selected slot** .



The figure above shows that **no external module is currently connected** in slot "Drive_1" - the fields **Module** (Module name) and **Module Ident** (Module ID) are **empty**. Also the server list is - for better illustration - empty in this example.

With the **Add button** 

, **the module**selected in the Modules table is **connected to the slot** marked in the Slots table. The module name and module ID are then entered in the slots table, and all **servers** that belong to the connected module are **added to the server list**:



With the **delete key** 

the selected **external module** is **removed** from the slot. Likewise, all servers that belong to PDO of this external module are also removed from the server list.



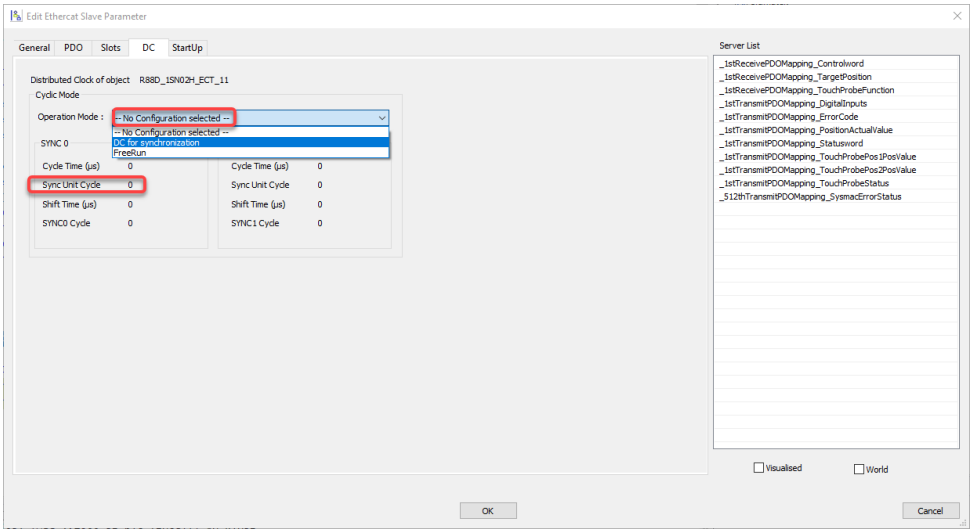
Attention

All changes made on this tab affect the class and all its objects!

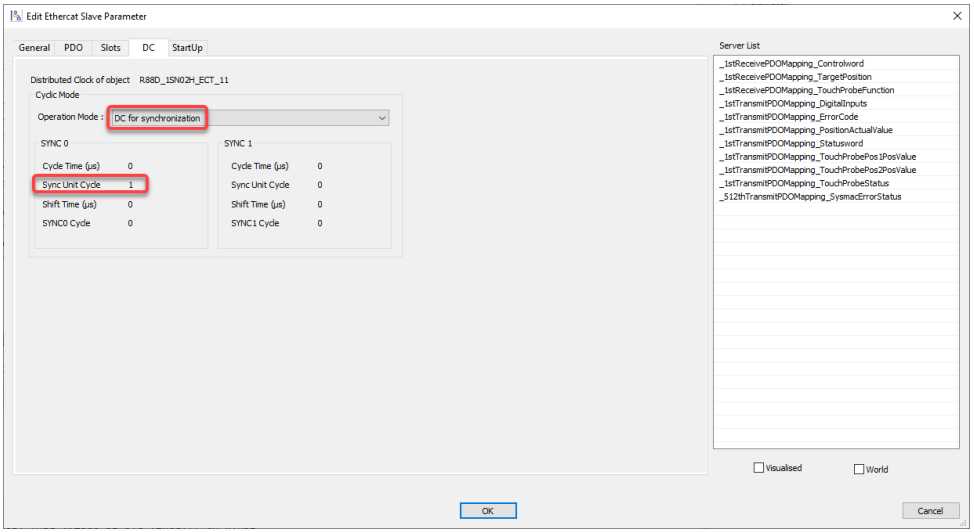
4.2.2.24 DC

On the **DC** (**Distributed Clock**) tab, you can switch between the possible DC modes in the **Operation Mode** field. Whether DC modes are available or which ones they are, is specified by the manufacturer of the EtherCat slave (refer to the EtherCat slave documentation of the according manufacturer).

The following image shows an example of the possible DC modes, below the corresponding parameters of the two Sync outputs are displayed.



After switching the **Operation Mode**, the parameters are adjusted accordingly:





Attention

All changes made on this tab affect the class and all its objects!

4.2.2.25 StartUp

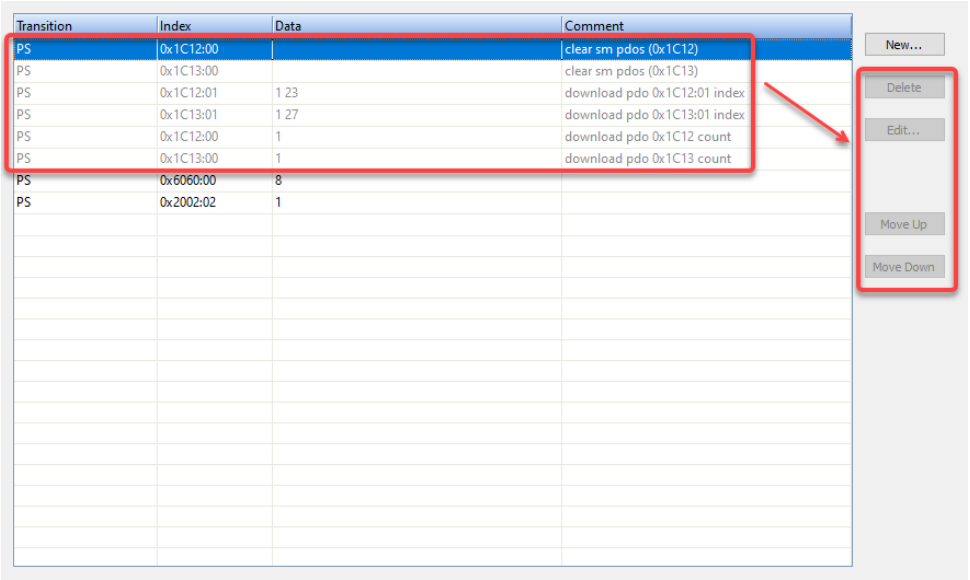
The last tab is **StartUp**.

EtherCat participants (slaves) can have the following states: **I**nitialization, **P**reOperational, **S**afeOperational, **O**perational. In the **transitions** parameters can be set/changed. Actually the EtherCat Master does this in the background, but by means of the StartUp commands the user can also transfer configuration parameters with the status change for EtherCat participants.

StartUp commands can be created with the "**New**" button. The commands are processed from top to bottom in the order they are listed in the table. With the buttons "**Move Up**" and "**Move Down**" they can be moved.

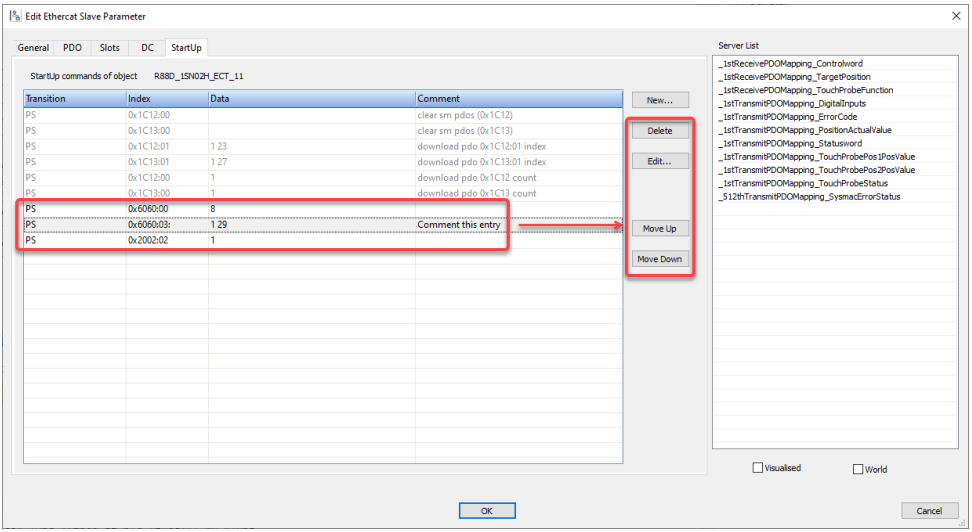
In the table, the StartUp commands are displayed in four columns. **Transition, Index, Data** and **Comment**.

There are StartUp commands, which are predefined by the manufacturer and cannot be edited - these are shown in grey.

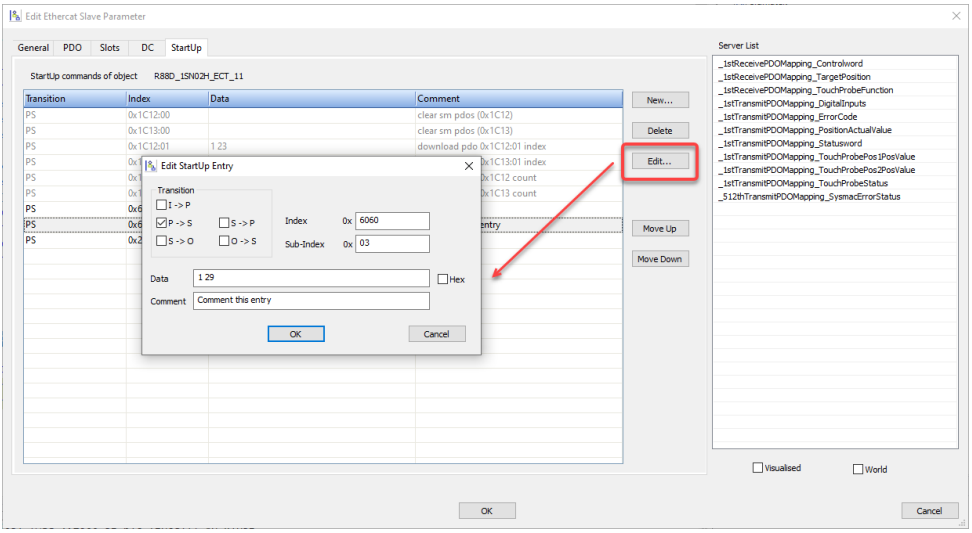


Entries created by the user are of course editable.

This also applies to those StartUp commands that were provided by the manufacturer and released for editing (marked as editable).



The editing itself is done using the "**Edit**" button:

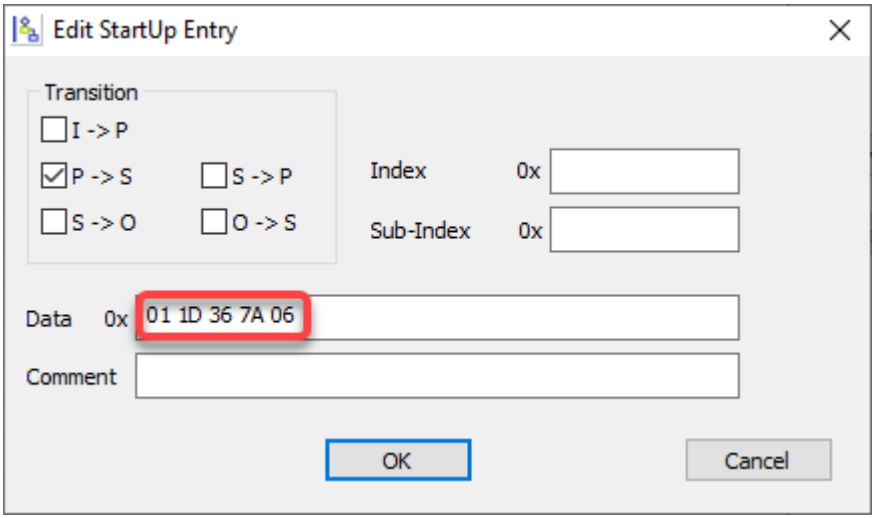


In the **Edit StartUp Entry** dialog that appears, the values of the four table columns can be entered. In the upper left corner we see the possible **transitions** which can be selected via checkbox:

- Initialization --> PreOperational
- PreOperational --> SafeOperational
- SafeOperational --> PreOperational
- SafeOperational --> Operational
- Operational --> SafeOperational

To the right of it, the **index** and the **sub-index** can each be specified in a separate field. This must be in hexadecimal format. These values **define the parameter to be changed**. For further information see the documentation of the respective module.

Below this, the **value** (Data) and a **comment** can be entered. It is also possible to specify 2 or more bytes, for this the bytes must be separated by a space (the length of the data is calculated automatically):



The comment field serves only for a better overview in the display and has no effect on the module.

With the "**OK**" button the StartUp entry is **entered** or the change is accepted, with "**Cancel**" it is **rejected**.

Attention

All changes made on this tab affect the class and all its objects!

4.2.3 The Properties Window of the PLC

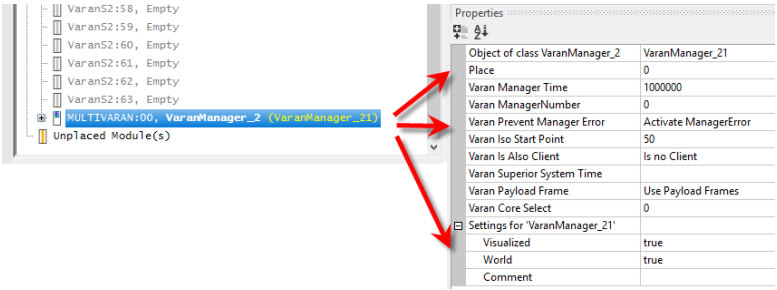
If the PLC is clicked once, the following view is displayed in the Properties window (here completely opened):

Properties

Object of class HwControl	HwControl1
Object of class DiasMasterC	DiasMasterC1
Object of class SdiasPLC	SdiasPLC1
Object of class VaranS2PLC	VaranS2PLC1
Place(s) for CDIAS (def. 4)	
Place(s) for DIAS (def. 8)	
Place(s) for MULTIVARAN (def. 1)	1
Port number for safety via IP	10823
Interface number for safety via IP	1
Sdias Manager Time in µs	1000
Sdias Iso Start Point	50
Sdias Core Select	
VaranS2 Manager Time in µs	1000
VaranS2 Iso Start Point	50
CableLength	1
⊟ Settings for 'HwControl1'	
Visualized	true
World	true
Comment	
⊟ Settings for 'DiasMasterC1'	
Visualized	true
World	true
RealTime	1 ms
RealIndex	Sigmatek Inputs
Comment	
⊟ Settings for 'SdiasPLC1'	
Visualized	true
World	true
Comment	
⊟ Settings for 'VaranS2PLC1'	
Visualized	true
World	true
Comment	
PLC Configuration	CDIAS, DIAS, MULTIVARAN, OnBoa

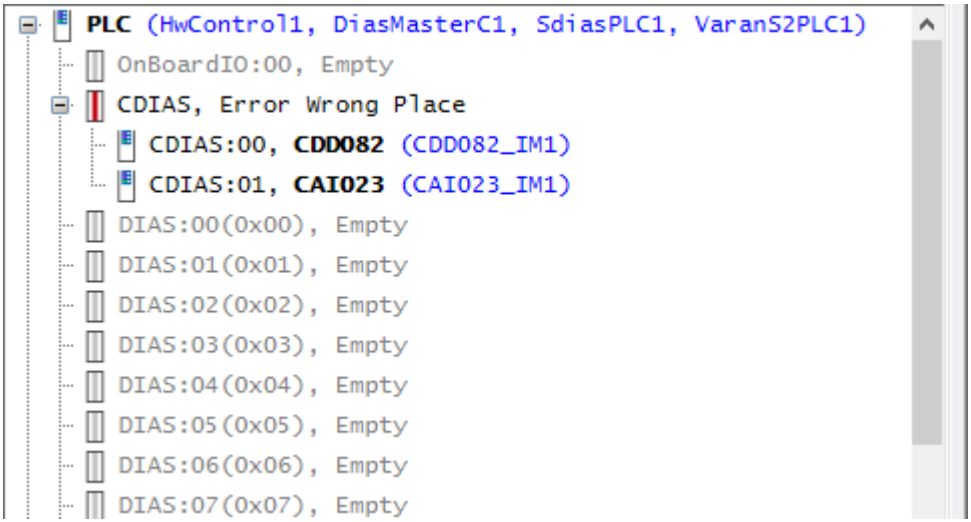
As can be seen, a PLC in the hardware editor with the maximum configuration consists of an **HWControl**, a **DiasMasterC**, a **SdiasPLC** and a **VaranS2PLC**. Therefore, a maximum of four **Object of Class...** fields are displayed, which show the respective object names that can be changed.

The **VaranManager** is not displayed in the hardware editor as part of the PLC, but - for the sake of clarity - as a separate "module" further down in the hardware editor tree at slot **MULTIVARAN:00**. All Varan-specific properties and a separate comment field are now located in the properties of the **VaranManager** located here.

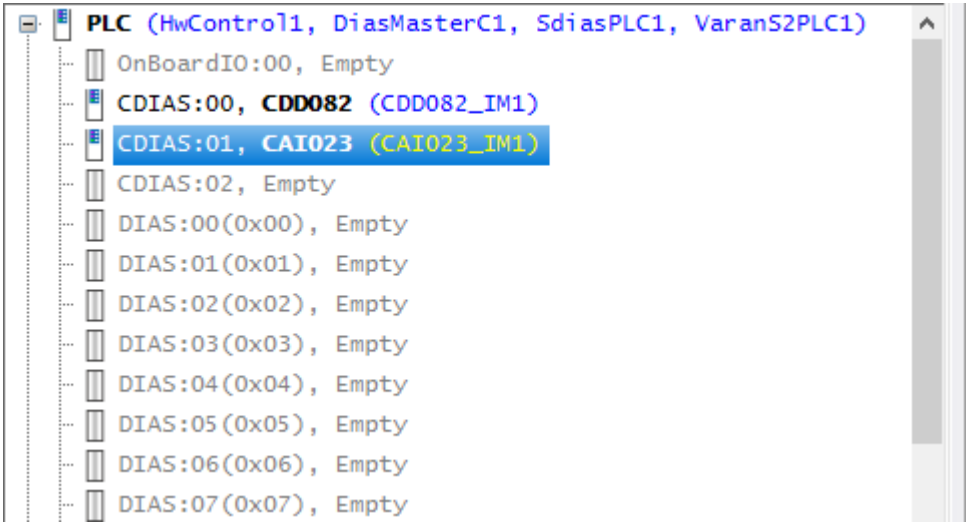


Place(s) for CDIAS (def.4)

Indicates how many CDIAS interfaces the PLC has. If no number is entered here, the default value is shown in the hardware tree (in brackets, here: 4). If the value is 0, no CDIAS interfaces are shown. Modules that have already been added to interfaces are now moved to the "CDIAS, Error WrongPlace" folder, which instead of the CDIAS interfaces, is now shown in the hardware tree.



If the 0 is deleted from the field in the properties, the module is removed from the Error Wrong Place and returned to the correct position. The Error Wrong Place folder then disappears.



Place(s) for DIAS (def.8)

Analog to the previous field, for DIAS interfaces only. The default value here is 8 interfaces (DIAS 0 to DIAS 7).

Place(s) for MULTIVARAN (def. 1)

Since there are now also modules with two independent VARAN interfaces, you can enter 1 (default setting) or 2 here. A VARAN manager with 1, 2 or 3 VARAN outputs can be inse-rted at each of the MULTIVARAN interfaces.

Port number for safety via IP

This value should only be changed if absolutely necessary, otherwise it can lead to conflicts in the system! The default value is 10823.

Interface number for safety via IP

Here the IP interface can be determined for safety use via IP. The default value is 1.

SDIAS Manager Time in µs

Setting for the SDIAS bus cycle in µs (maximum 32000).

SDIAS Iso Start Point)

Indicates the time point within the available SDIAS manager time in percent (1...99%), from which reading should start (writing always starts immediately after the hardware class has processed its tasks).

Sdias Core Select

Select the processor core on which the real-time and cyclic tasks of the SDIAS Manager are to run. Supported only for multi-core CPUs!

VARAN S2 Manager Time in µs

Setting for the VARAN S2 bus cycle in µs (maximum 32000).

VARAN S2 Iso Start Point

Indicates the time point within the available VARAN S2 manager time in percent (1...99%), from which reading should start (writing always starts immediately after the hardware class has processed its tasks).

CableLength

Length of the Varan cable connected to the VARAN In interface. Default is 1 (meter).

Settings

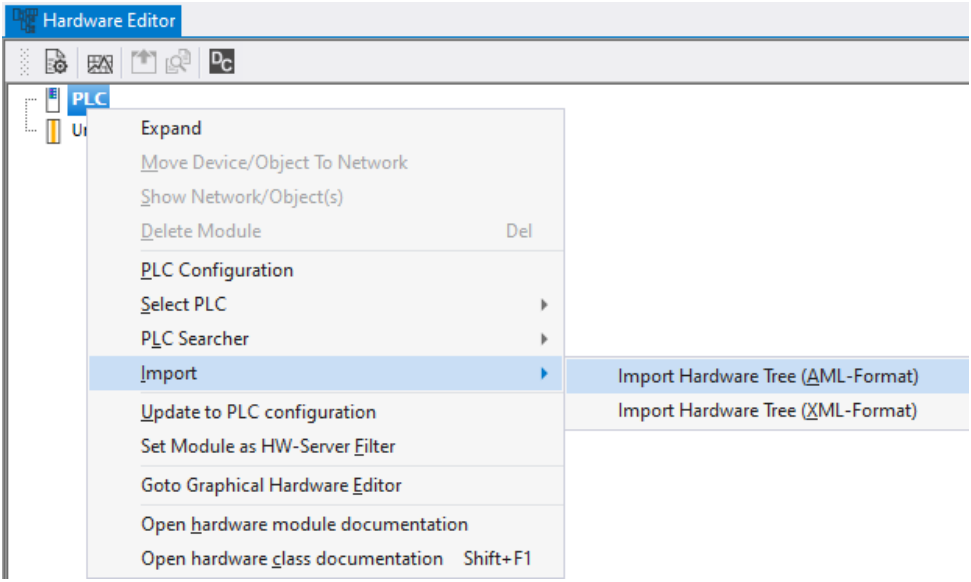
Here, the individual object properties can be found. The individual fields were already addressed with the object properties.

PLC Configuration

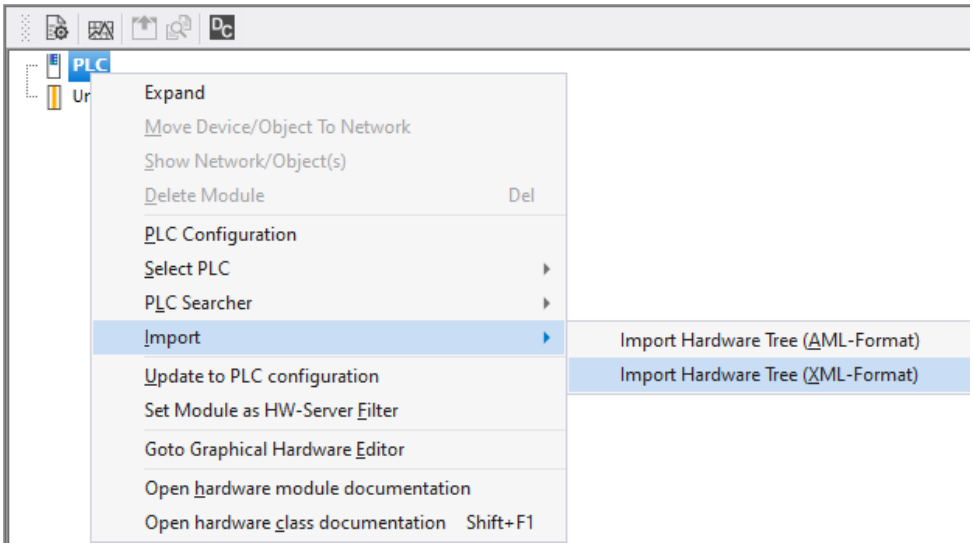
Shows all Types of modules, which can be added to the PLC in the current configuration (with a maximum configuration these are: **DIAS**-, **C-DIAS**- **S-DIAS, VARAN** and **VARANS2** modules, in addition, the PLC can have an **OnBoard-IO** module).

4.2.4 Adding Modules

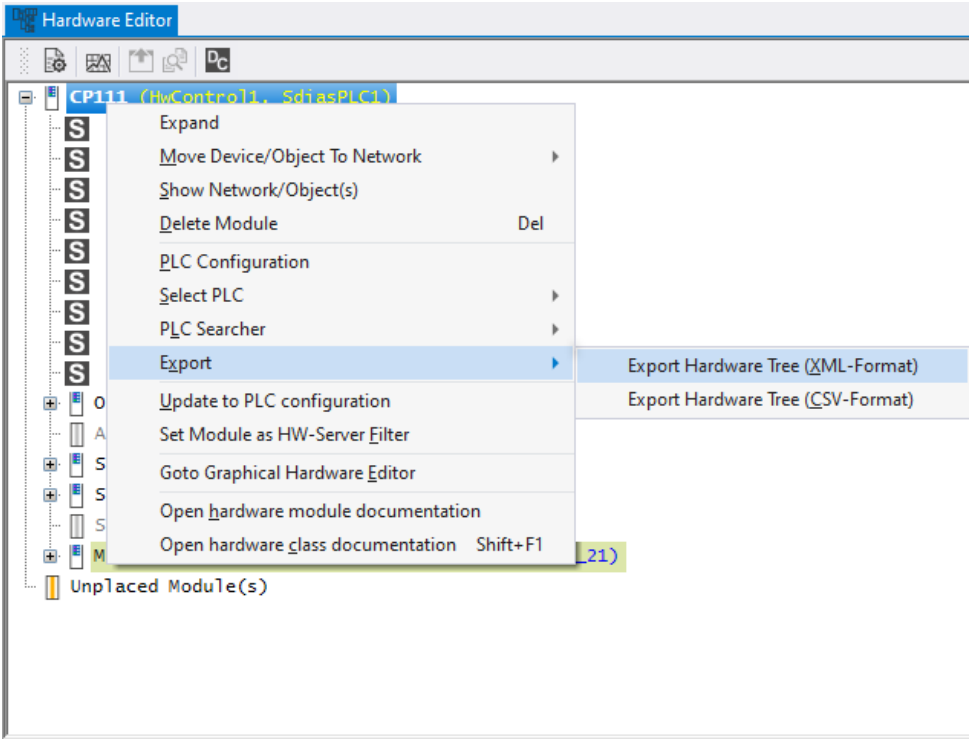
If the PLC is not yet configured, a previously saved, complete hardware tree can be imported. And either in AML-...



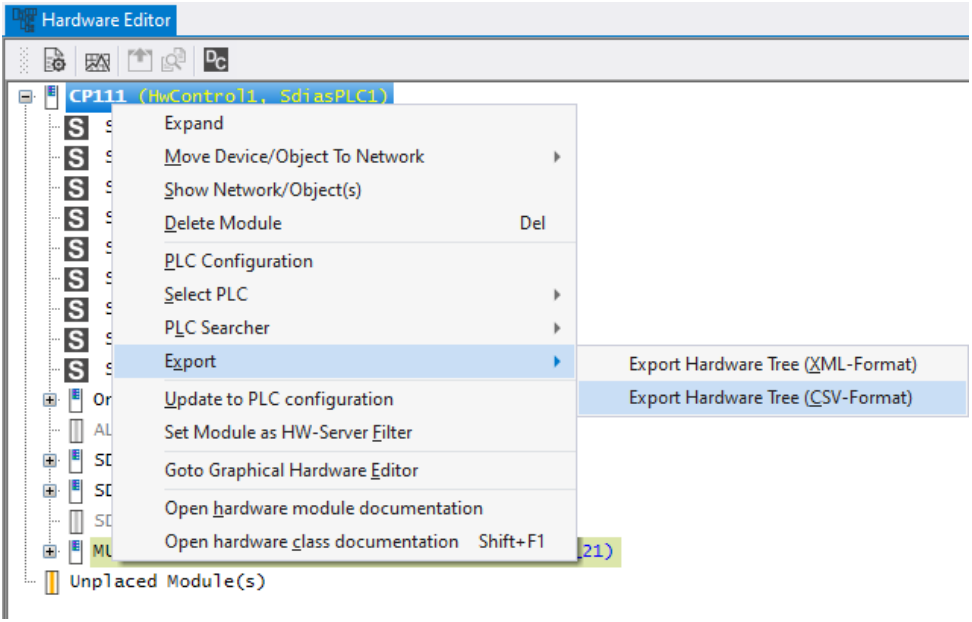
...or in XML format:



If the PLC is already configured, the hardware tree can be exported and re-imported later. This can be stored in XML ...

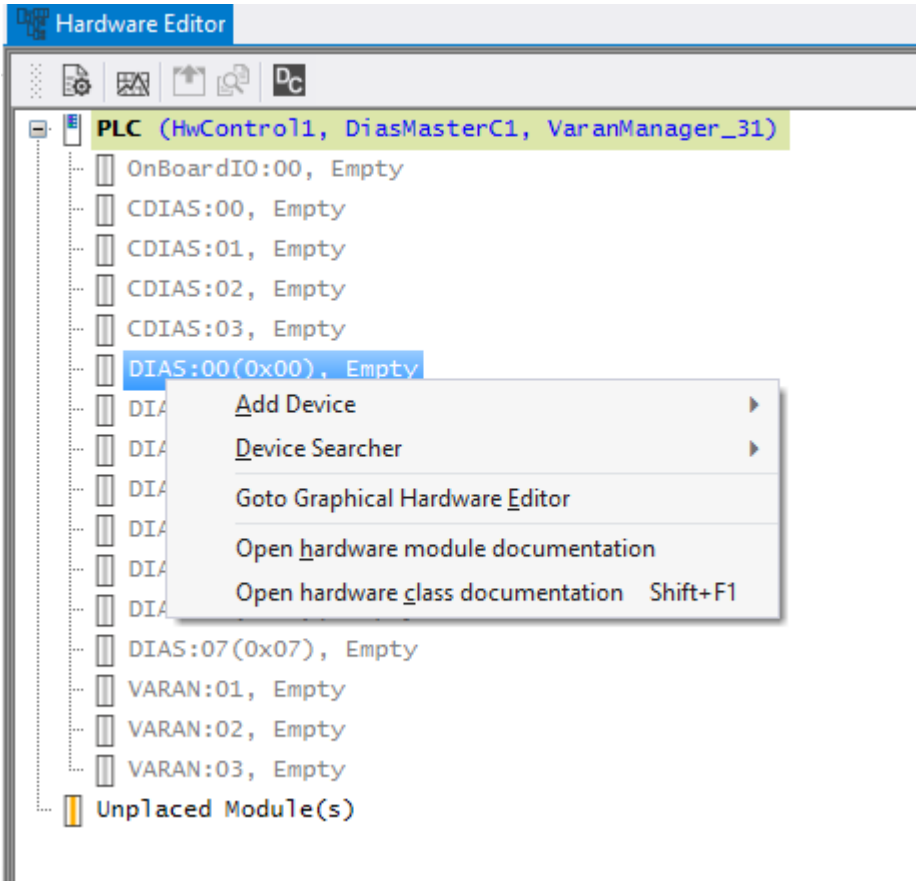


...or CSV format:

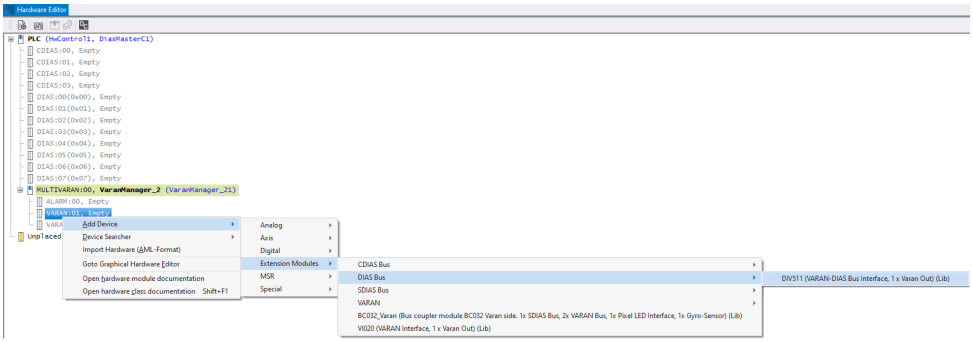


The normal addition of modules functions as follows:

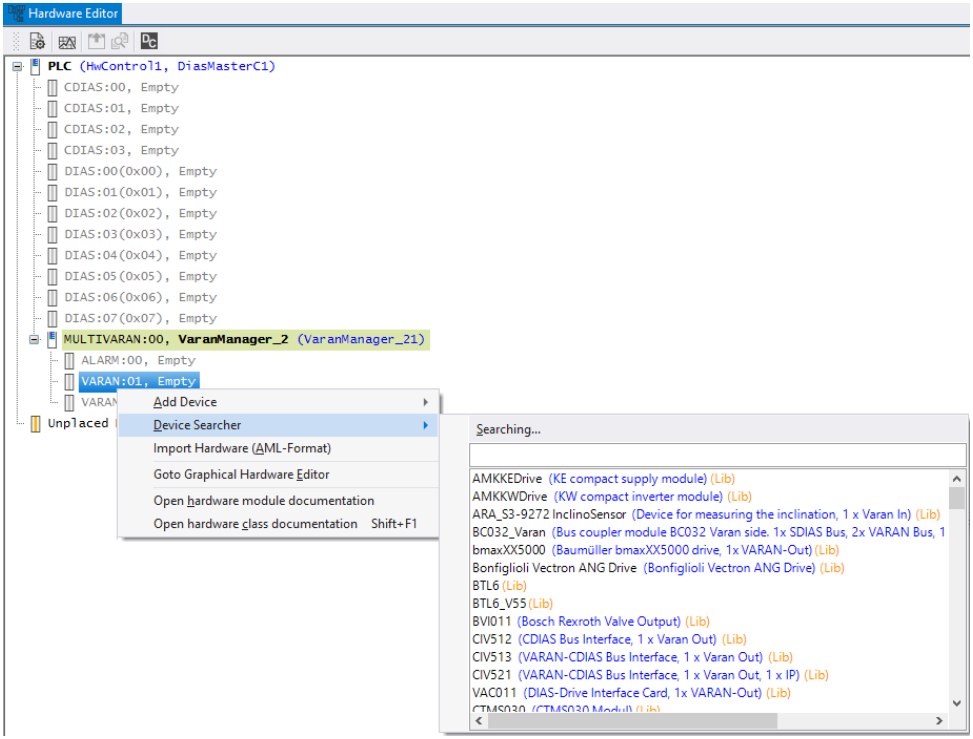
With a right click on one of the empty interfaces, the context menu appears, which contains the items **Add Device, Device Searcher** and **Set Module as HW-Server Filter and Goto Graphical Hardware Editor**, **Open hardware module documentation** and **Open hardware class documentation**:



When **Add Device** is selected, the module search can be limited by category:



The second option is to select **Device Searcher**. This menu point should only be selected, the module name is at least partially known.

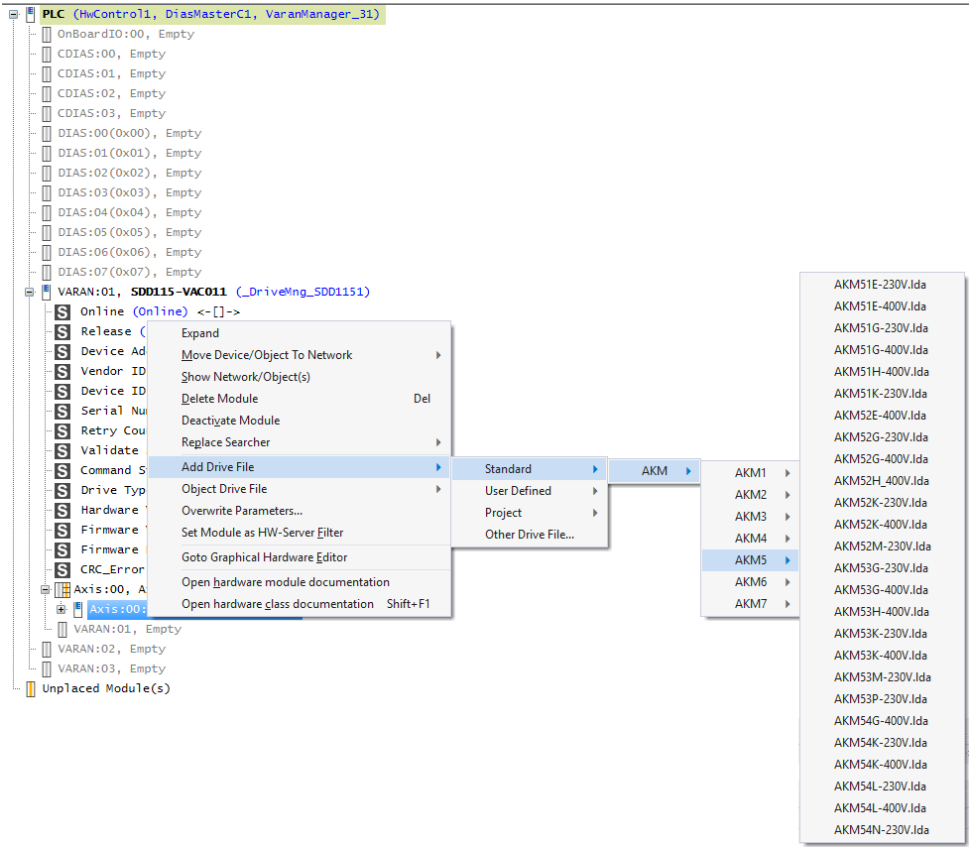


The third option is to **Set Module as HW-Server** filter. This applies to the filter in the Connection Manager. (For further information, see the chapter IO-Connection Manager).

With a click on the menu item **Goto Graphical Hardware Editor**, the graphic hardware editor (GHE) is started.

The two last menu items, **Open hardware module documentation** and **Open hardware class documentation**, open the respective documentation, if available.

Files can be added to a drive object (axes). Here, open the context menu of the axis objects and select **Add Drive File**:



Four types of files can be added:

Standard

Motor files (currently AKM) included in LASAL.

User Defined

Motor files found under Dias **Drive => User Defined**.

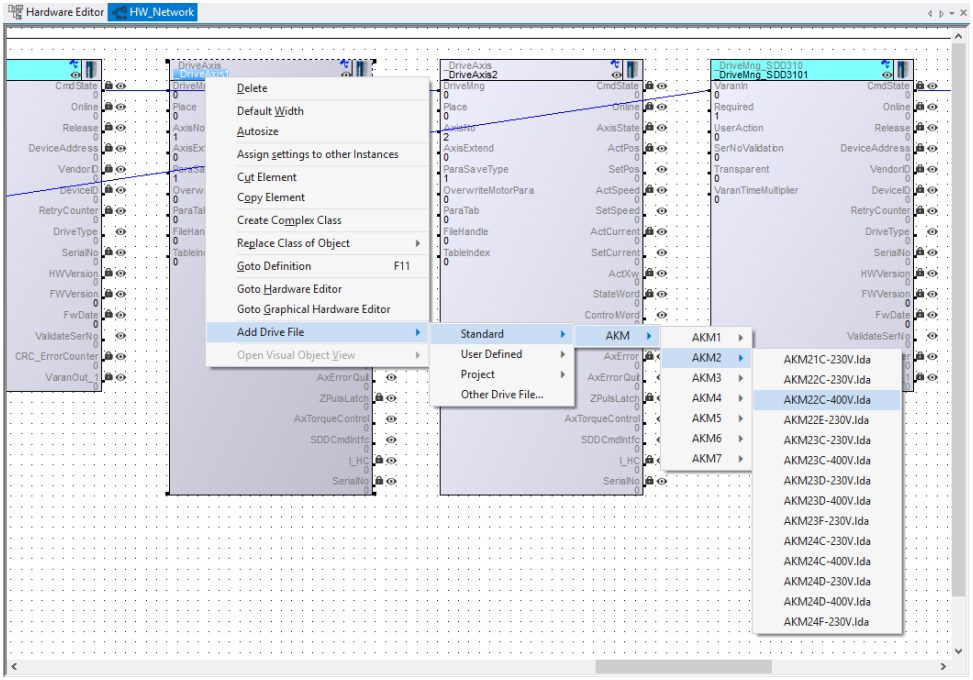
Project

Motor files, which were already added to an axis object (useful for example, when the same files must be added to several axes).

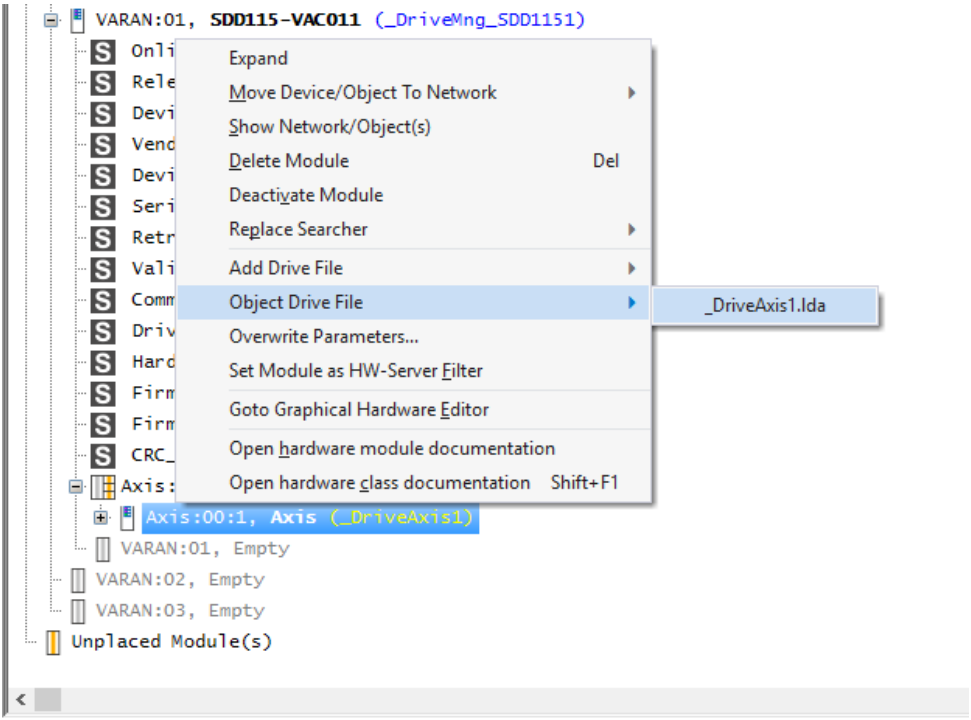
Other Drive File

Miscellaneous motor files that can be searched in the file system.

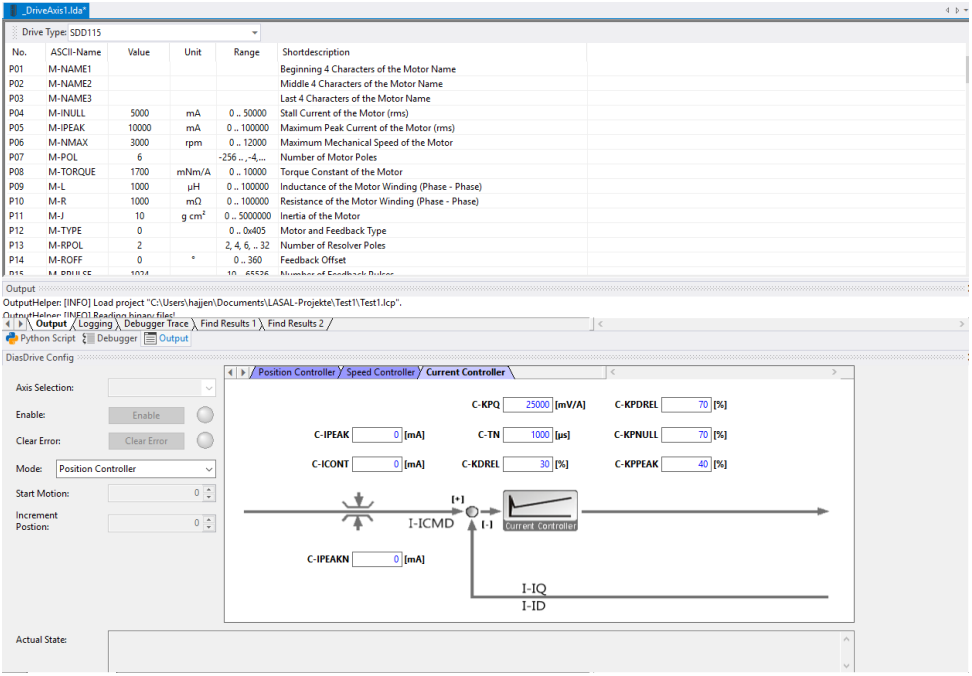
Motor files can also be added in the network view. An axis object need only be positioned and the context menu opened.



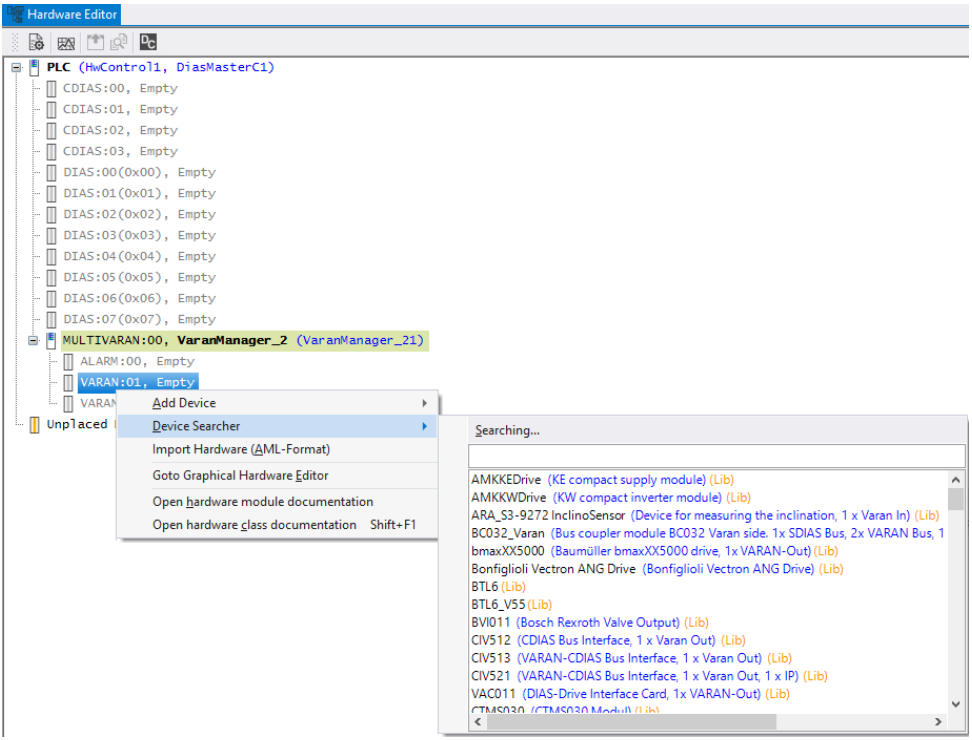
If a file has already been added to the axis, the menu item **Object Drive File** can then be selected.



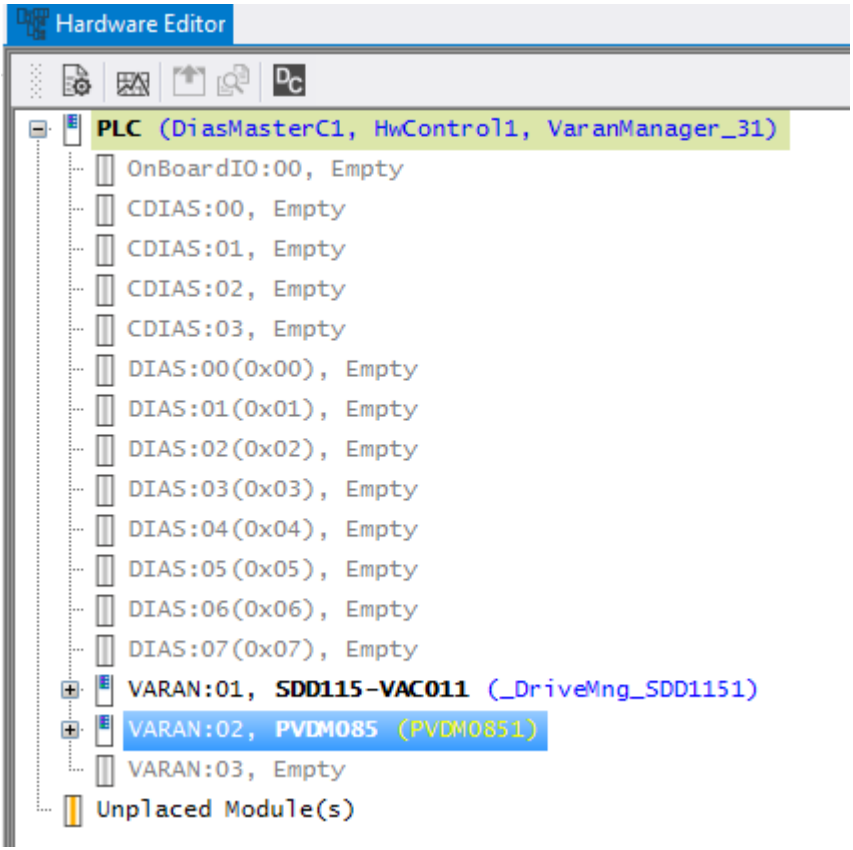
If this is selected, the drive object is shown:



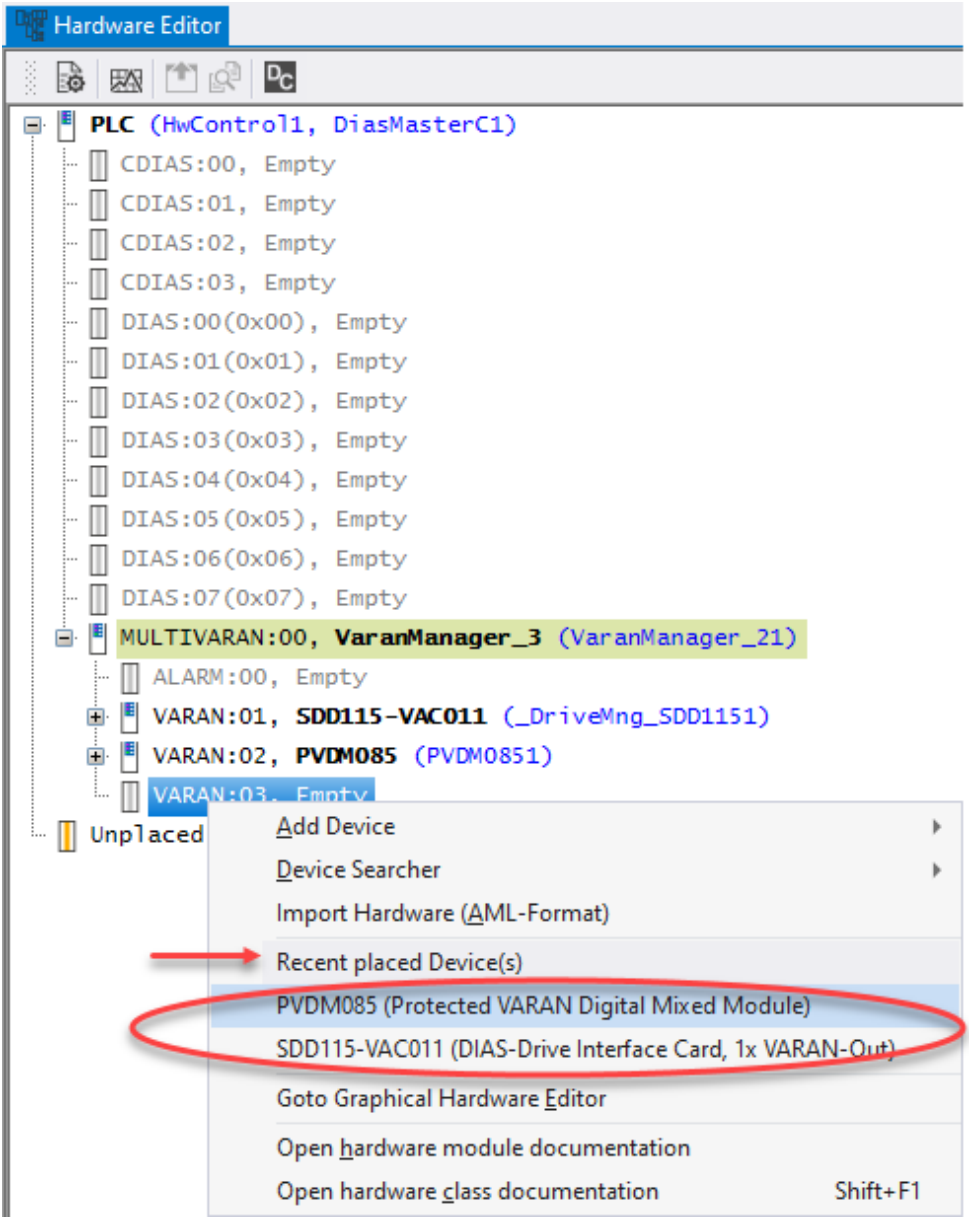
With the mouse, it is possible to scroll through the modules available at this interface alphabetically – or by pressing the **S button**, the cursor springs to the input field where the user can search by module name:



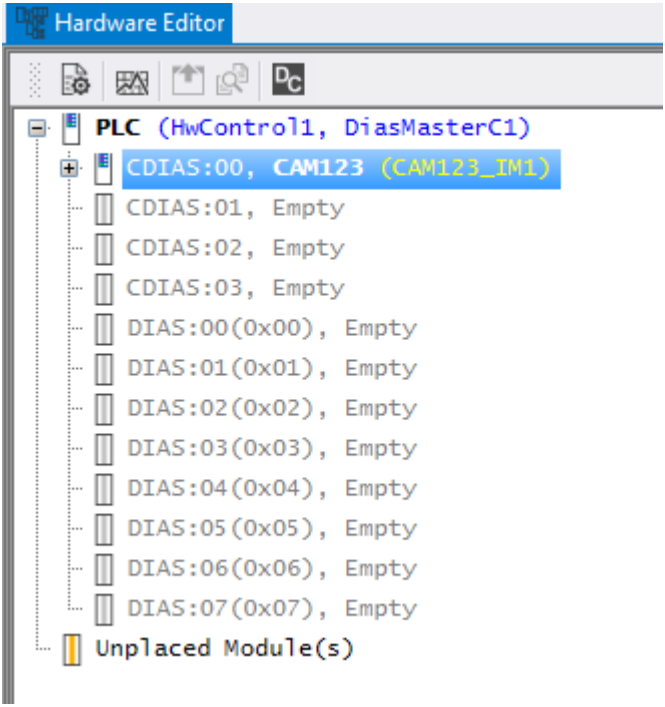
If the module is now selected, an **import Class dialog** is - if **Show Import Dialog** is set to **true** - opened for only those classes required for the module. When confirmed, the module is added to the interface.



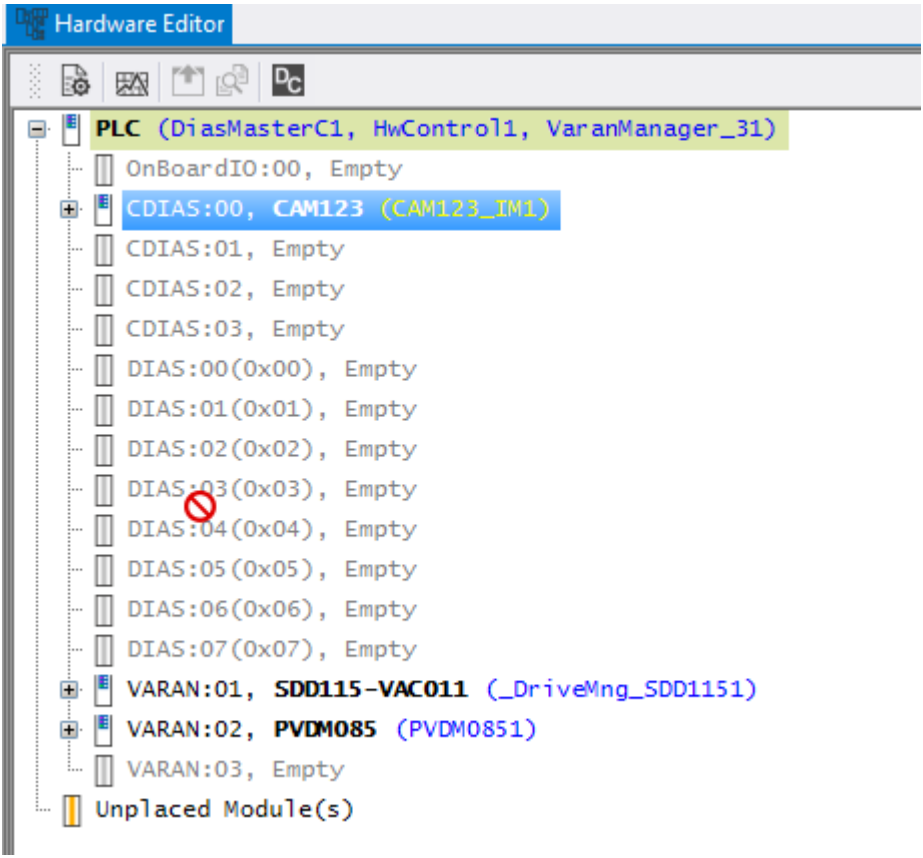
If another module must be added to VARAN interface, the context menu expands with the section "**Recent placed Device(s)**", in which all previously placed modules are found in the lower section of the menu (also those that were possibly deleted). These can be added directly with a mouse click, without having to search.


Hardware Editor
PLC (HwControl1, DiasMasterC1)
CDIAS:00, Empty
CDIAS:01, Empty
CDIAS:02, Empty
CDIAS:03, Empty
DIAS:00(0x00), Empty
DIAS:01(0x01), Empty
DIAS:02(0x02), Empty
DIAS:03(0x03), Empty
DIAS:04(0x04), Empty
DIAS:05(0x05), Empty
DIAS:06(0x06), Empty
DIAS:07(0x07), Empty
MULTIVARAN:00, VaranManager_3 (VaranManager_21)
ALARM:00, Empty
VARAN:01, SDD115-VAC011 (_DriveMng_SDD1151)
VARAN:02, PVDM085 (PVDM0851)
VARAN:03, Empty
Unplaced
Add Device
Device Searcher
Import Hardware (AML-Format)
Recent placed Device(s)
PVDM085 (Protected VARAN Digital Mixed Module)
SDD115-VAC011 (DIAS-Drive Interface Card, 1x VARAN-Out)
Goto Graphical Hardware Editor
Open hardware module documentation
Open hardware class documentation
Shift+F1

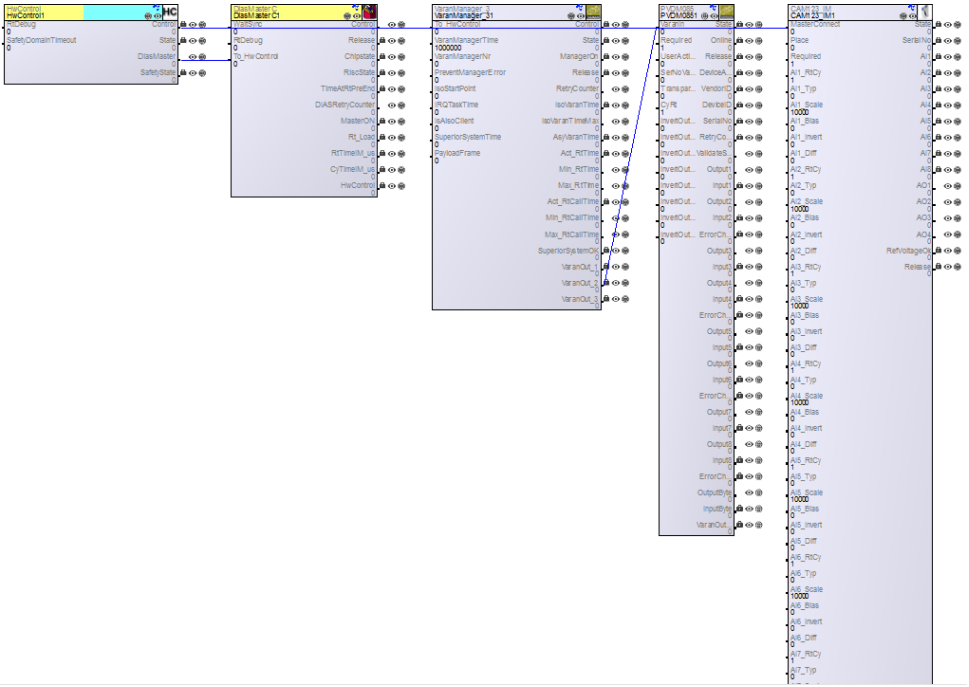
If a module is placed, it can then be moved with Drag&Drop over the same interface. A corresponding mouse pointer indicates whether or not the desired location can be added (**here**: a CDIAS module is moved to a CDIAS interface),...



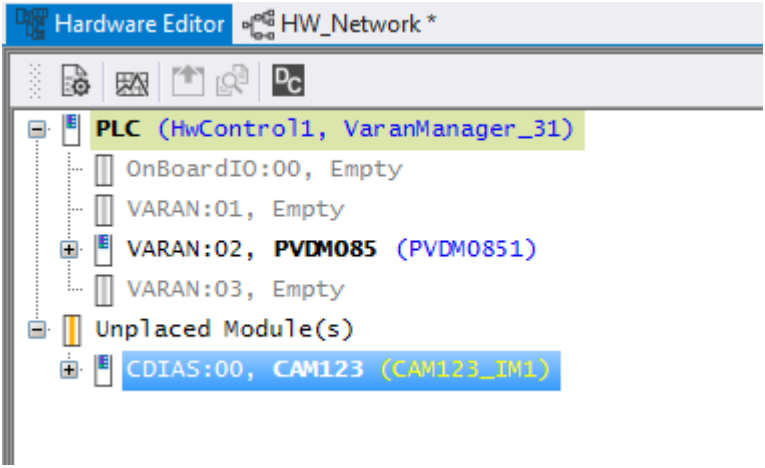
(**here**: an attempt to move a CDIAS module to a DIAS interface).



When the network view for **HW_Network** is opened, the effects of just a few mouse clicks can be seen.



If the PLC is later reconfigured, so that suddenly modules have no interface (here: **No DIAS/CDIAS**, the **CAM123** module no longer has an interface), the "floating" modules are placed in the "**Unplaced Module(s)**" folder.

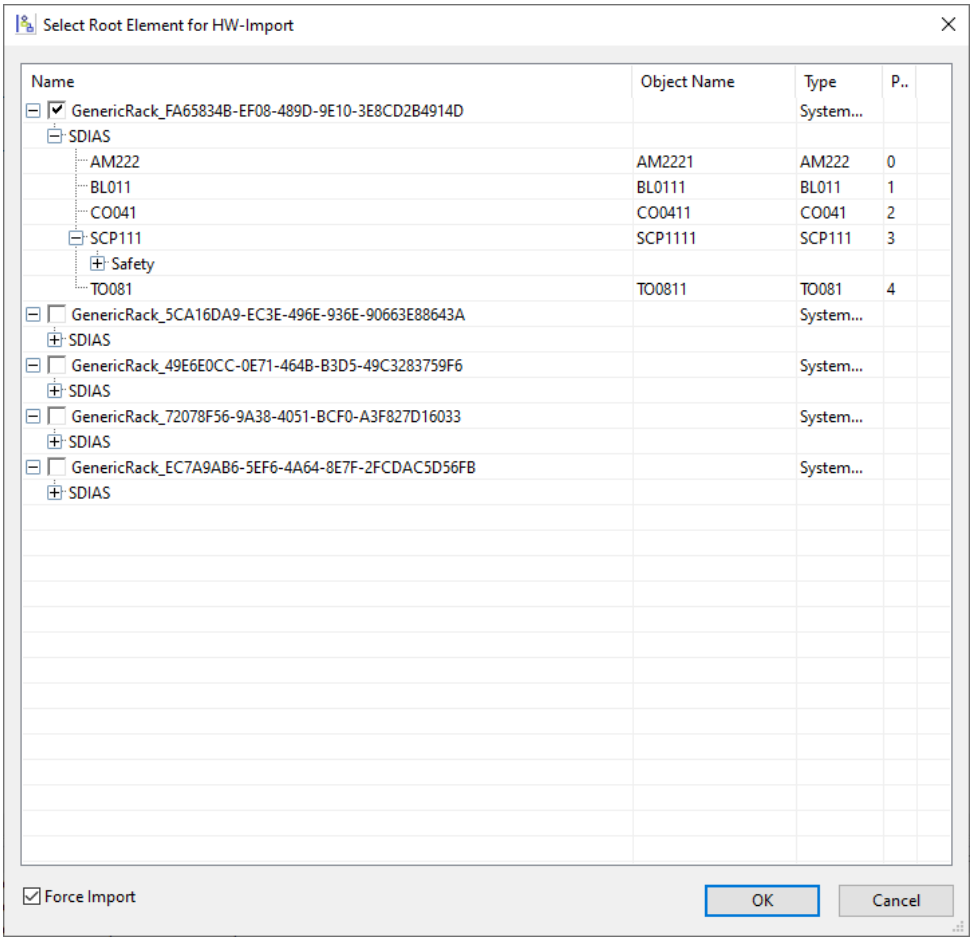


If the DIAS/CDIAS PLC configuration is allowed again, the module must be moved to the appropriate interface by hand.

Import Hardware (AML-Format)

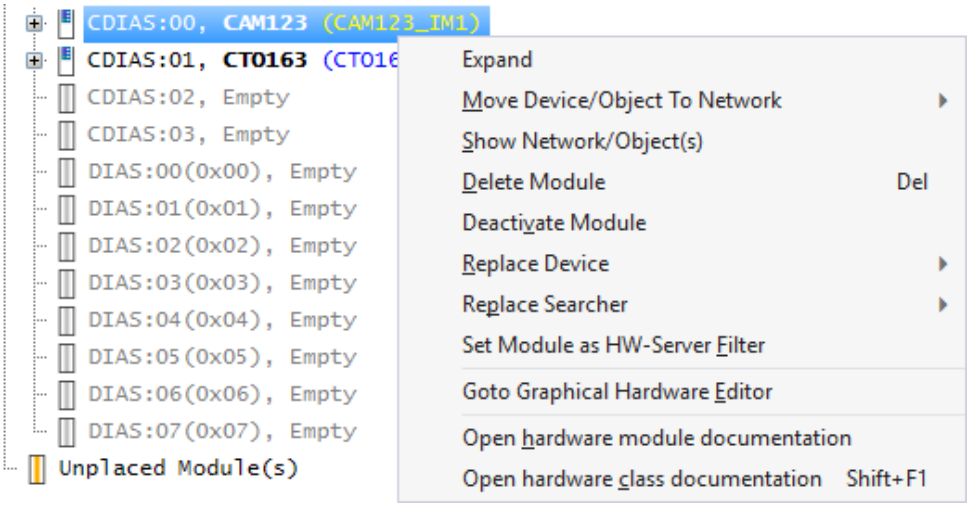
In the context menu of SDIAS, VARAN, CDIAS or DIAS slots there is a menu item "**Import Hardware (AML format)**". Here you can import an *.aml file exported from EPLAN (for more information see chapter AutomaticML).

If a whole EPLAN project is imported (at least all branches of a CPU) under the menu item Project / AutomationML / Import Hardware Tree or in the configuration context menu of the PLC, then a corresponding branch (i.e. an SDIAS, VARAN, CDIAS or DIAS branch) can be imported with this menu item. In the displayed import dialog only the corresponding branches are shown, from which one can be selected and imported:


Select Root Element for HW-Import
Name
Object Name
Type
P..
GenericRack_FA65834B-EF08-489D-9E10-3E8CD2B4914D
System...
SDIAS
AM222
AM2221
AM222
0
BL011
BL0111
BL011
1
CO041
CO0411
CO041
2
SCP111
SCP1111
SCP111
3
Safety
TO081
TO0811
TO081
4
GenericRack_5CA16DA9-EC3E-496E-936E-90663E88643A
System...
SDIAS
GenericRack_49E6E0CC-0E71-464B-B3D5-49C3283759F6
System...
SDIAS
GenericRack_72078F56-9A38-4051-BCF0-A3F827D16033
System...
SDIAS
GenericRack_EC7A9AB6-5EF6-4A64-8E7F-2FCDAC5D56FB
System...
SDIAS
Force Import
OK
Cancel

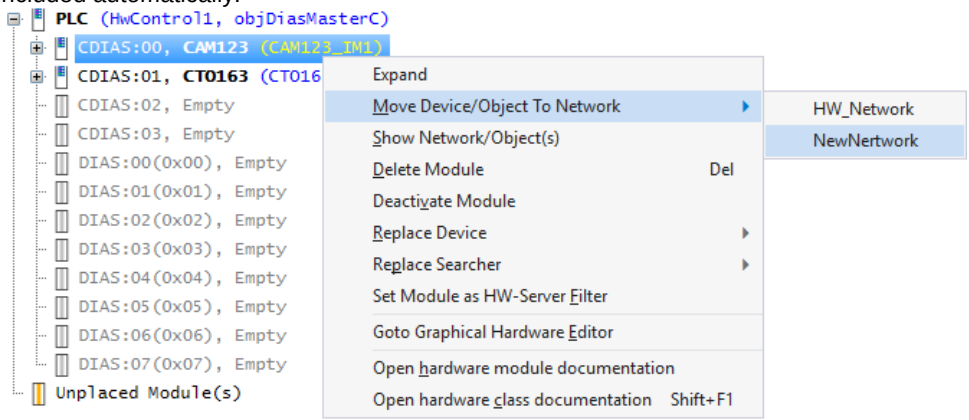
4.2.5 Context Menu of the Modules

A right click on the added module opens the following context menu:

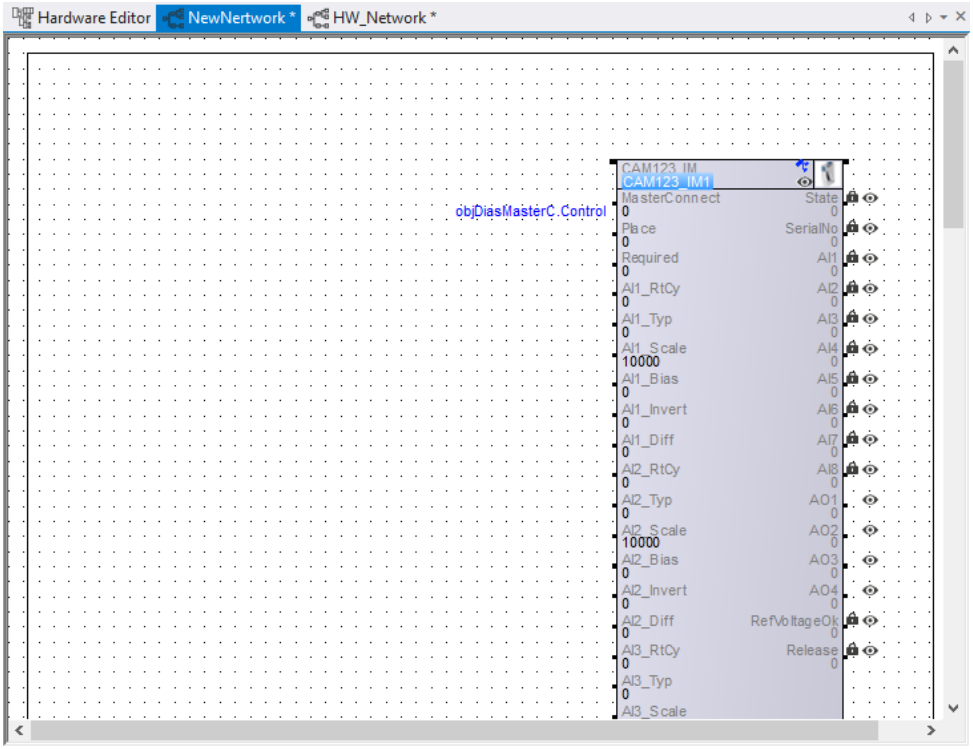


With Expand – similar to the PLC – the module is completely expanded.

With **Move Device/Object To Network**, a module can be moved to another network (here: NewNetwork) with only a mouse click. The required changes for existing connections are included automatically.

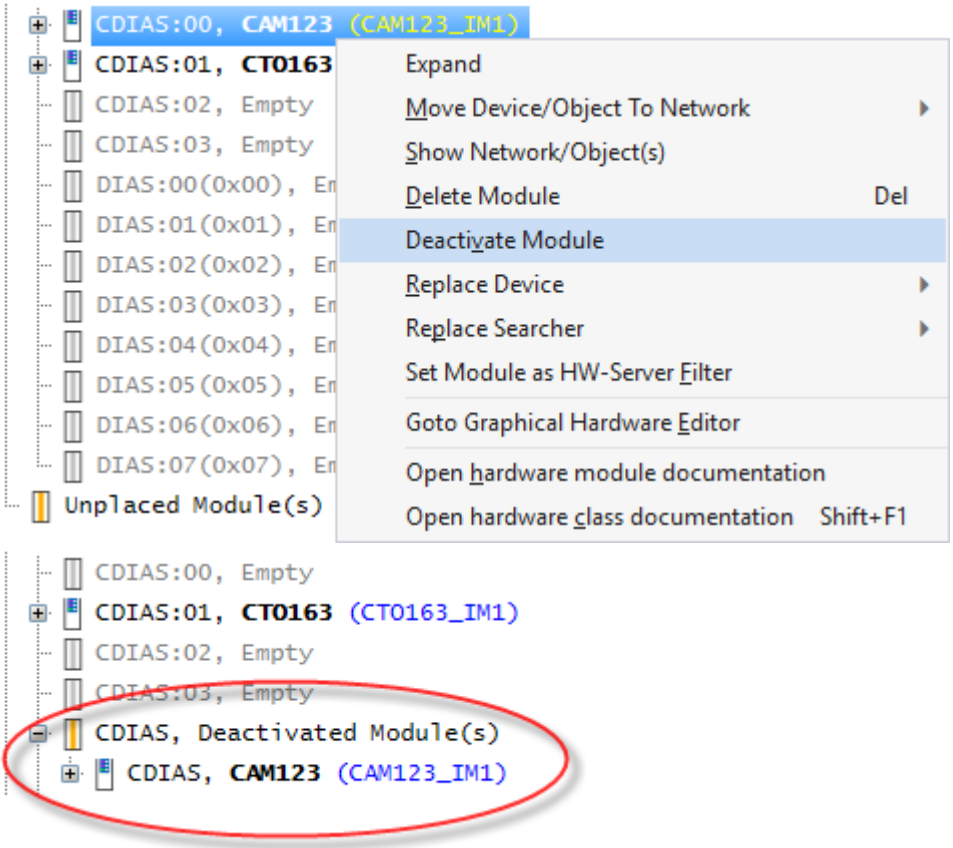


Show Network/Objects changes to the network view, in which the selected module is located. The module can now be seen in the new network:



As can be seen, the connection to DiasMasterC1 is included in the new network. With **Delete Module**, the selected module is then deleted.

Deactivate Module moves the module to the **Deactivated Modules** folder, which every section has (DIAS, CDIAS, VARAN...).



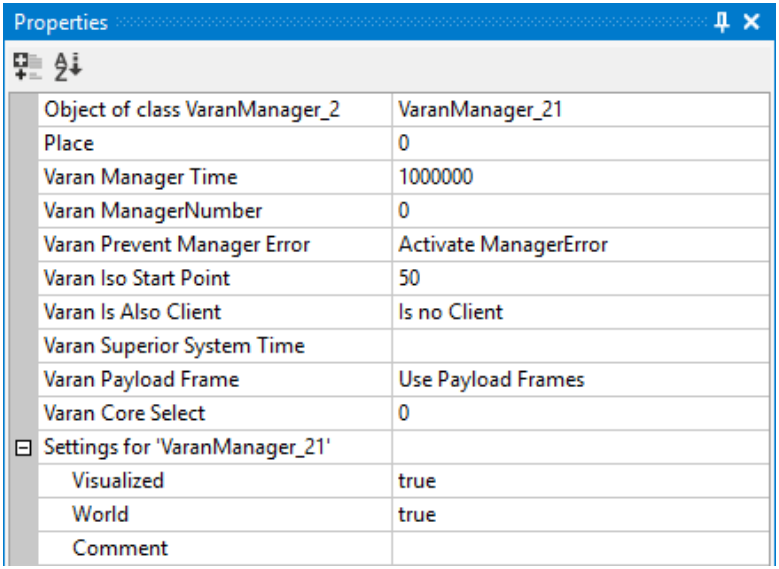
If this change was made unintentionally, the module can be returned to the original interface or moved to a different one at any time and therewith reactivated.

The next two items, **Replace Device** and **Replace Searcher** allows the exchange of the selected module with one click. The function of both menu items is the same as the two analog items **Add Device** and **Device Searcher** in the context menu of an empty interface (see: Adding Modules).

The menu item **Set Module as HW-Server Filter** applies to the Connection Manager and sets the module as a filter in the HW Object column. For further information, see the chapter IO-Connection Manager.

The last three menu items (**Goto Graphical Hardware, Open hardware module documentation and Open hardware class documentation**) are identical to that of the PLC context menu.

4.2.6 Properties of the VaranManager



Object of class VaranManager_XX

Name of the VaranManager object.

Place

Number of the MULTIVARAN slots, to which the VaranManager is connected. Can only be 0 or 1.

Varan Manager Time

The time entered here is the cycle time of the VaranManager hardware classes and the isochronous VARAN DataObjectList as init value in ns. Default is 1000000 ns (= 1 ms). For more information on VARAN task processing, see the chapter Task Sequence with RtPostScan (up to LASAL 02.02.054)

VARAN Manager Number

If multiple VARAN managers are available in the system, the number is entered here to distinguish between them. The default value is 0.

Varan Prevent Manager Error

Here, whether the system should stop with a Varan Manager error can be set (Activate ManagerError) or not (Deactivate ManagerError).

Varan Iso Start Point

Is the starting point of the processing of the VARAN task within the isochronous cycle time. It is specified as a percentage of the VARAN Manager Time (1 % to 99 %). Example: Varan Iso Start Point = 70 (%) and Varan Manager Time = 1000000 (ns) would mean that the hardware classes (controller real time) would get 700 µs, the isochronous Varan task 300 µs.

Varan Is Also Client

Indicates whether the manager is also a client (VMC, also a Varan Manager client)

Varan Superior System Time

Cycle time of the primary Varan mode. This must a multiple of the Varan Manager Time. This is assigned for a VMC only; when Is Also Client is **true**.

Varan Payload Frame

Setting whether the VARAN Manager should use payload frames and DMA accesses. There are three possibilities: use payload frames (Use Payload Frames), do not use (Never use Payload Frames), or only use, if the operating system fully supports this (No Pay-load Frames unless full support from OS)

Varan Core Select

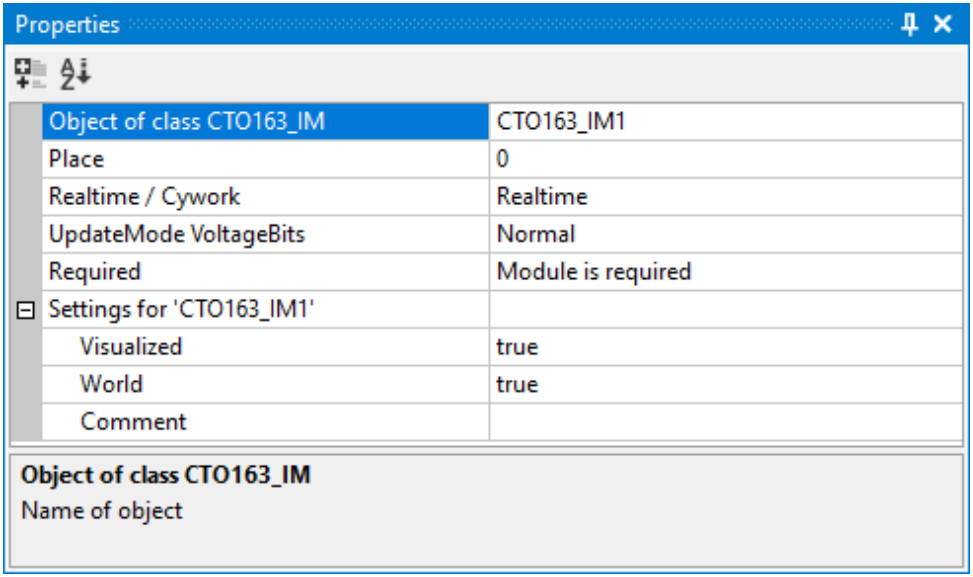
The desired processor core on which the real-time and cyclic tasks of the VARAN manager are to run can be specified here. Default is 0 (core 0). Core 1 is only supportrd by multi-core CPUs.

Settings

Visualized, World, Comment - same as the other objects.

4.2.7 The Properties Window of the Module

If a module is clicked (here: **CAM123**), the property window appears as follows:



Object of Class ...

Shows the object names that can be changed.

Place

Shows the PLC interface to which the module is connected. If the module is moved in the hardware tree, this number changes. In the number is changed here, the module is also relocated I the hardware tree.

Comment

Here, comments on the class can be entered.

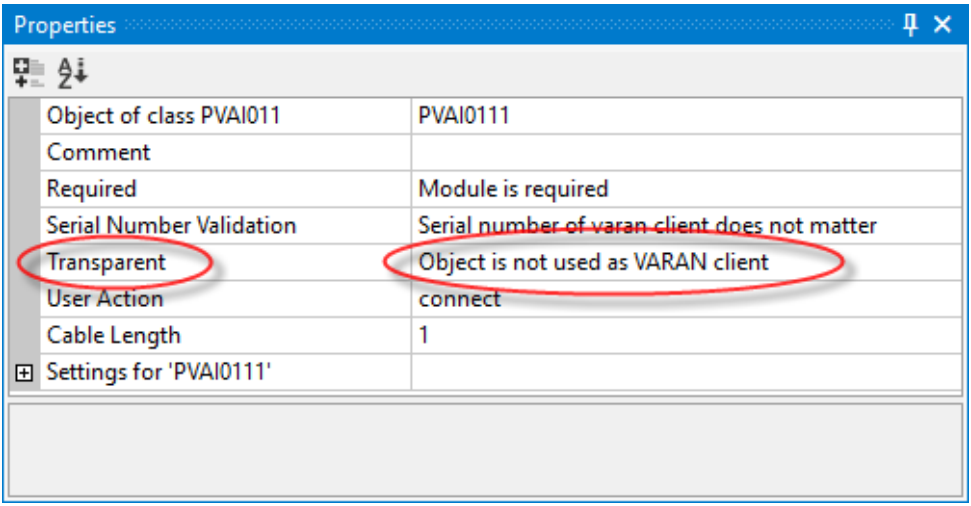
Required

Here, whether the module is required for starting the system can be set.

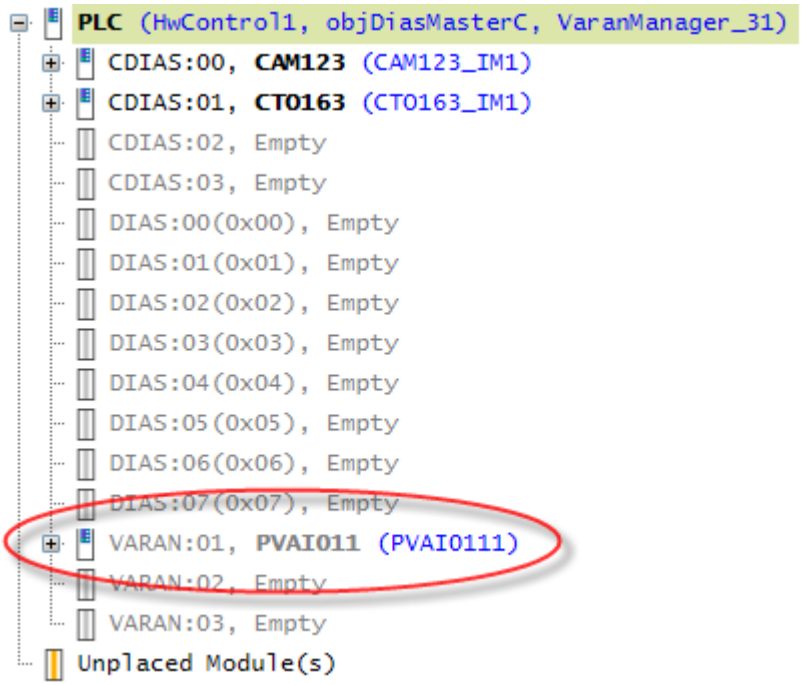
Settings

Here, the object properties can be found. In the hardware classes only Visualized (true or false the visualization of the object), World (defines the visibility of the server in the Multimaster - only if an object of the class with World = true flag exists) and Comment (object-specific commentary) are processed.

A **VARAN Module** also the “Transparent” property. If the Transparent flag is set to **“Object is not used as VARAN Client“**,

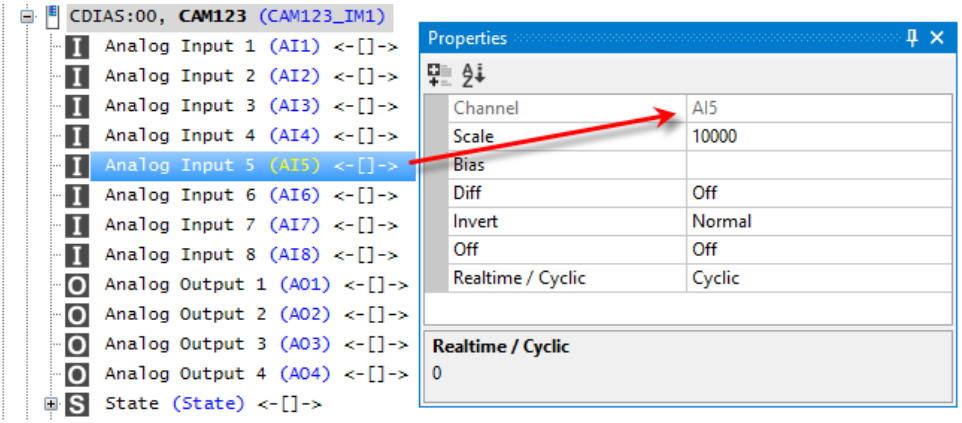


the module is deactivated. In the HW Editor, the module is grayed:



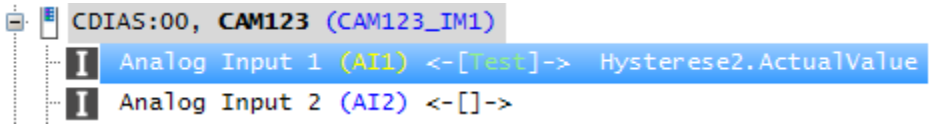
4.2.8 The Properties Window of the In- and Outputs

If the + sign is in front of the module in the hardware tree and an in- or output of the module is clicked, the Properties window shows the properties of the selected in- or output. The settings, which can now be made here, must have already been made in the client of the object in the network view.

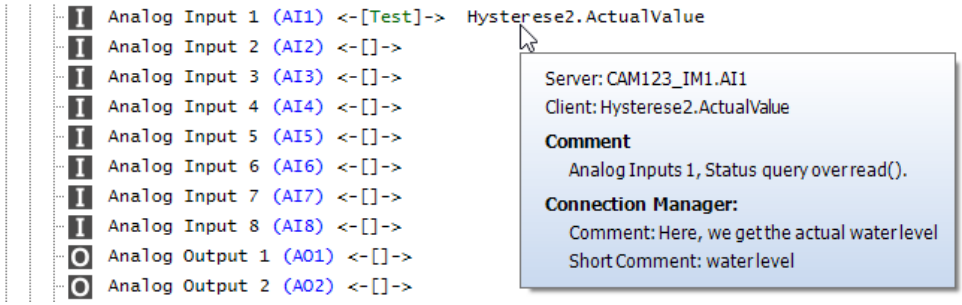


4.2.9 Connections in the Hardware Editor

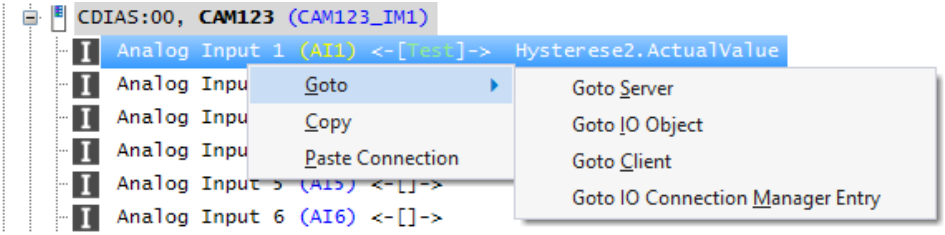
If there is connection between a hardware module server and a client that is also suggested by the **Connection Manager**, it is visible in the hardware editor.



If the mouse pointer stays on a connection somewhat longer, a Tooltip appears that contains useful information:



With a right click, the following items are given in the context menu:



Goto

Goto Server

The program jumps to the **server** connection in the network view.

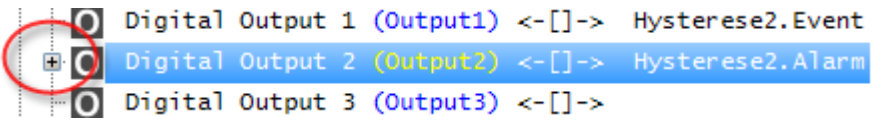
Goto IO Object

The program jumps to the **IO object** (blue) in the network. The menu item is only available, if such an object exists **Symb.-Name** was assigned and a connection in the network established).

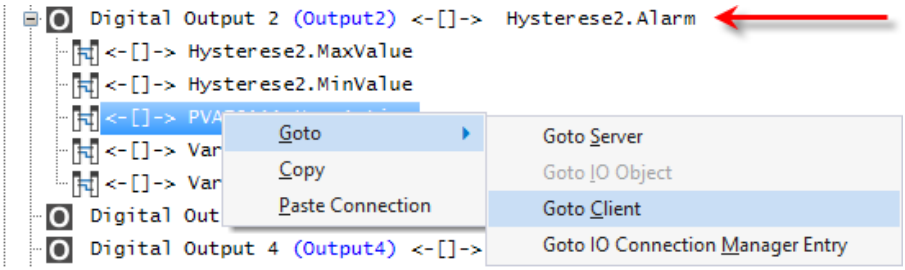
Goto Client

In the network view, a jump to the client connection is made.

If the server has several connections, Goto Client jumps to the client of the first connection. Multiple connections are indicated by a plus symbol in front of the server.

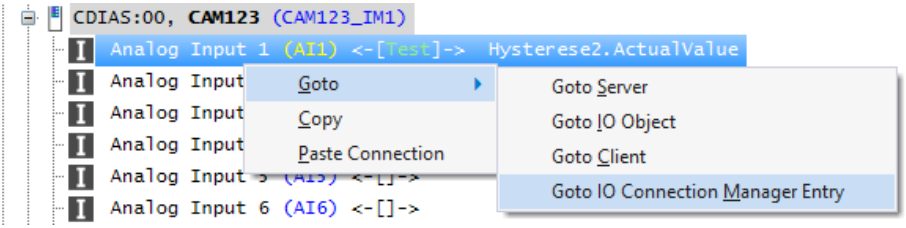


If the symbol is clicked, the connections open and in the context menu of the according connection you can directly jump to the according client.

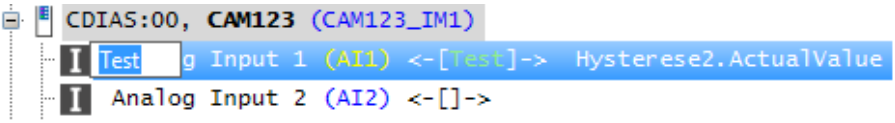


Goto IO Connection Manager Entry

The program jumps directly to the connection entered in the **Connection Manager**. If the **Connection Manager** is closed, it now opens automatically. (For further information, see the chapter Connection Manager).



Here, the **Symb.-Names** can be entered or changed. Simply click and hold the mouse button on the connection.



Copy

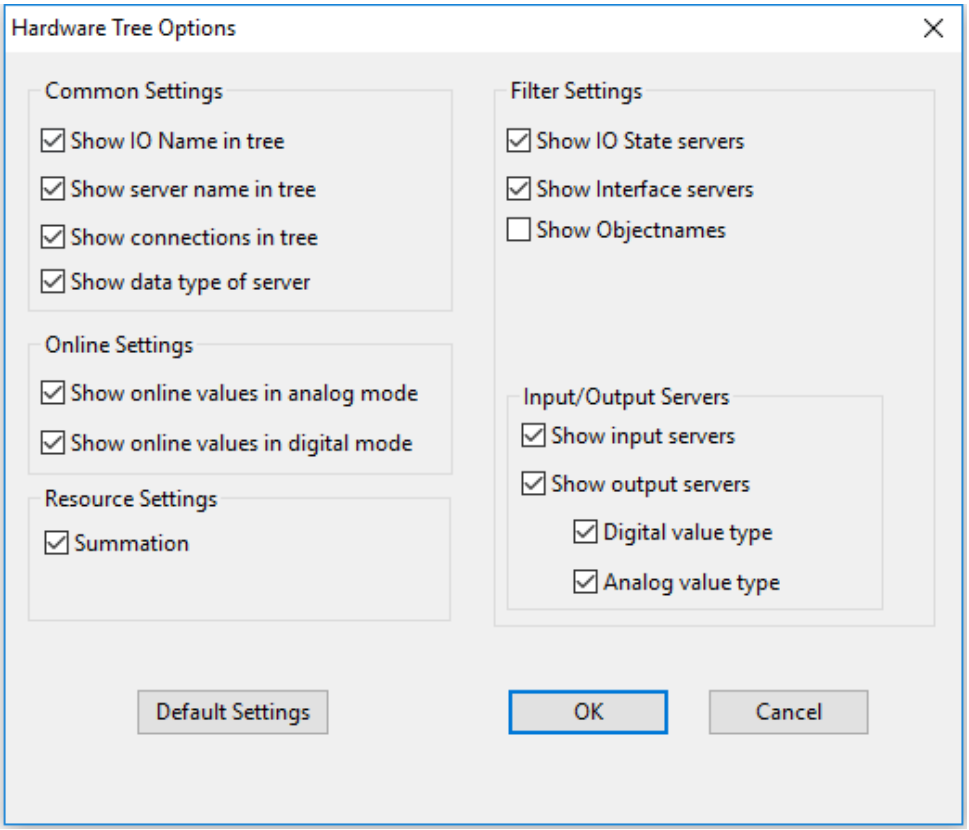
Copies the selected server for to create a connection with “Paste Connection”. The same function is available with a right click on the sever value in the object network.

Paste Connection

Adds a copied client to the selected server as a connection.

4.2.10 View Options in the HW Editor

If the first icon  in the toolbar of the hardware editor is clicked a dialog box is opened, in which the content displayed in the HW editor can be defined.



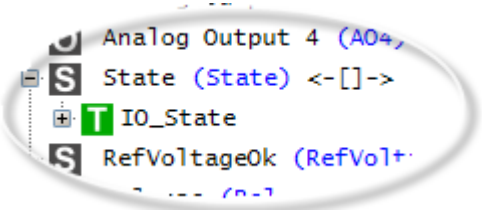
The dialog is divided into four areas: **Common Settings**, **Online Settings, Resource Settings, Filter Settings** and **Server Settings** at the last low area **Input/Output Servers**. In the first section – Common Settings – for adjustable display parameters are available:

If **Show IO Name in tree** is selected, the IO name of the connection is shown in the hardware tree in brackets. If the item is deactivated, the display of the IO name is suppressed.

If **Show server name in tree** is deactivated, the display of the server is suppressed.

If **Show connections in tree** is selected, connections to the servers are shown (or the connected clients respectively). Otherwise nothing.

If **Show data type of server** is selected, the data type of a server is shown if available.



In the **Online Settings** area, two items can be selected:

Show online values in analog mode and **Show online values in digital mode** control whether the status (set, not set, no value) for analog /digital in- an outputs should be identified with a colored background when online.

The third section is **Resource Settings**; this has only one item: **Summation**.

The fourth area is **Filter Settings**, this has a further subsection: **Input/Output Servers**. In total, seven items are available for selection.

Show IO_State servers and **Show Interface servers** control, as indicated in the label, whether state and interface servers should be displayed or not. **Show Objectnames** also controls, as expected, whether the object names should be displayed.

In the **Input/Output Servers** area, whether these are displayed is selected with **Show input servers / Show output servers**. Two additional items allow the respective server types to also be limited by selecting **Digital value type** and/or **Analog value type**.

These options prove to be powerful tools for customizing the display, especially with large projects that have a high number of modules, connections, etc.

With the **Default Settings** button, the display parameters are reset to the previous status (all selected).

The  icon starts the Resource Calculation.

The icons  (upload) and  (compare) are described in the chapter **Online with the Hardware Editor**.

The last icon  switches between the **Detail** and **Configuration** modes. In the Detail mode, all the serves of a module are visible. In the configuration mode only the module and - depending on the hardware - the most important in- and outputs can be seen. This can be an advantage when, for example, during the beginning phase of a large project: Configuration mode helps to maintain an overview.

4.2.11 Online with the Hardware-Editor

If **online** and the **PLC is not yet configured**, the existing hardware configuration in the control can be loaded. This occurs when the **Upload icon**  in the toolbar is clicked. If the **Import Class dialog** is confirmed, the **complete hardware tree** in the control is **imported** with all networks.

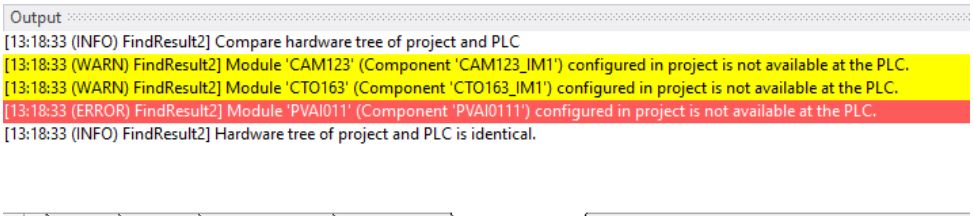


Attention

It is **not possible** to load several identical EtherCAT modules in the hardware tree **with different revision numbers**. In this case the module **with the highest revision number** is always loaded during the upload. If this is undesirable, the user must replace the module manually!

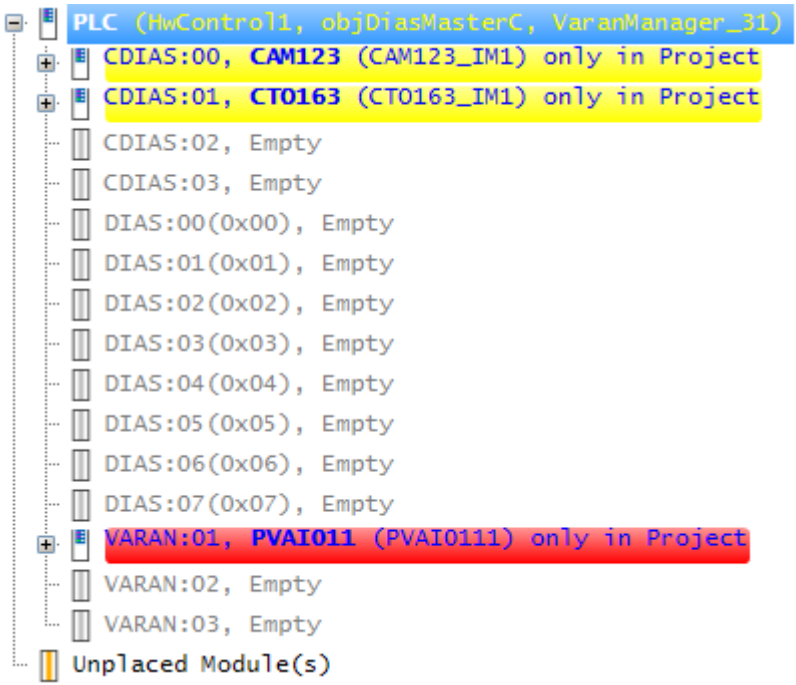
If the **PLC is already configured** and goes **online**, a **comparison** of the hardware configuration in the control and the configuration in hardware editor can be made. If **differenced** are detected, they are output to the **Find Results 2 window**.

The **hardware tree comparison** is started with the toolbar button  . In the following screen shot, an example of the differences is shown.

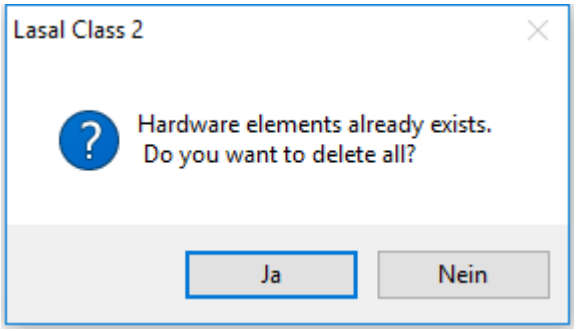


If the "Required" client is set in a module, error messages with a red background are shown. If the "Required" client is not set, only warnings with a yellow background are given.

In the hardware tree modules with set Required flag, which are missing in the hardware tree or in the configuration of the control, are marked red; modules without Required flag are marked yellow.



If the PC is already configured, and the user goes online to upload the hardware, the user is then asked whether the existing configuration should be overwritten. To replace the existing hardware with the changes, confirm with **Yes**.



The option to **change online values in directly in the hardware tree** is also available. First **go online** and to the servers that should be changed – they should not be write-protected! – Next click **select** and then press **F2**. The value is shown in an input field and can be changed directly.

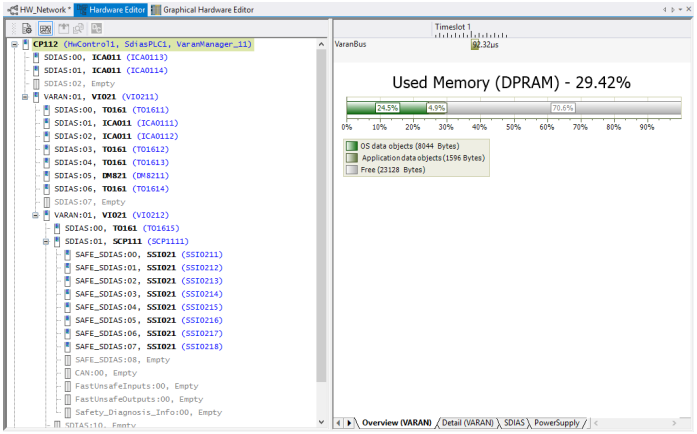


If a server is not of a standard type, the type is blended directly into the underlying plane.


State Word (StateWord) <-[]-> (2#00000000000000000000000000001000)
DS402_State (2#00000000 00000000 00000000 00001000)
ReadyToSwitchOn (0)
SwitchedOn (0)
OperationEnabled (0)
Fault (1)
VoltageEnabled (0)
QuickStop (0)
SwitchOnDisabled (0)
Warning (0)
ManufactureSpec1 (0)
Remote (0)
TargetReached (0)
InternalLimitActive (0)
OperationModeSpec1 (0)
OperationModeSpec2 (0)
ManufactureSpec2 (0)
ManufactureSpec3 (0)
Bit Nr.17 (0)
Bit Nr.18 (0)
Bit Nr.19 (0)
Bit Nr.20 (0)
Bit Nr.21 (0)
Bit Nr.22 (0)
Bit Nr.23 (0)
Bit Nr.24 (0)
Bit Nr.25 (0)
Bit Nr.26 (0)
Bit Nr.27 (0)
Bit Nr.28 (0)
Bit Nr.29 (0)
Bit Nr.30 (0)
Bit Nr.31 (0)
Bit Nr.32 (0)
Control Word (ControlWord) <-[]-> (2#00000000000000000000000000000000)
Axis Enable (AxEnable) <-[]-> (0)

4.2.12 Resources Calculation for VARAN and S-DIAS

With the icon  you can start the **resources calculation** for the VARAN communication. Here only **VARAN and S-DIAS modules** are displayed. It is also possible to start this tool from the **Graphical Hardware Editor** with the same icon. From what point of view whatever the resources calculation is started, on the **left side** the **Hardware Editor Tree** is always displayed.

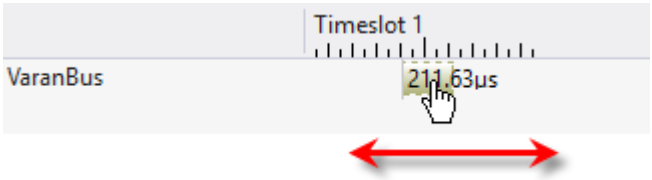


There are four views in the right area - realized with its own tab each - **Overview (VARAN)**, **Detail (VARAN)**, **SDIAS** and **PowerSupply**.

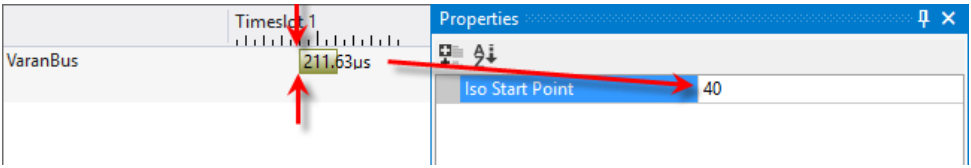
4.2.12.1 Overview (VARAN)

In the **Overview (VARAN)** tab, the **Iso-starting point of the VARAN communication**, depending on the VARAN Manager time (top) set, as well as the **consumption of the individual data objects** and the remaining available memory (**below**) are displayed.

The **Iso-starting point** can be **moved freely** within the time window (which can be adjusted under the **VaranManagerTime** property at PLC in the **HW editor**):



The value set here can be directly read in the Properties window:

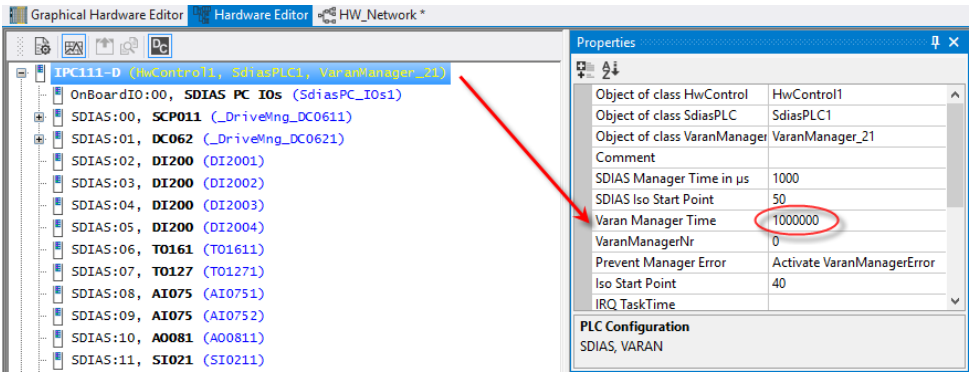


If it is moved so that the VaranBus-bar is no longer completely viewable, the bar turns red:

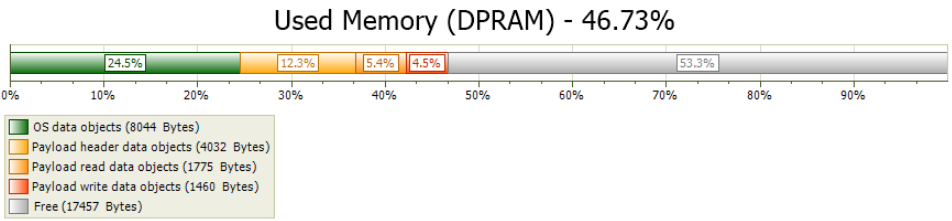


Attention

In this condition, **the project does not work**! The Iso-starting point whether has to be moved further to the left or a higher value has to be set for the **VARAN Manager Time** in the Properties window!

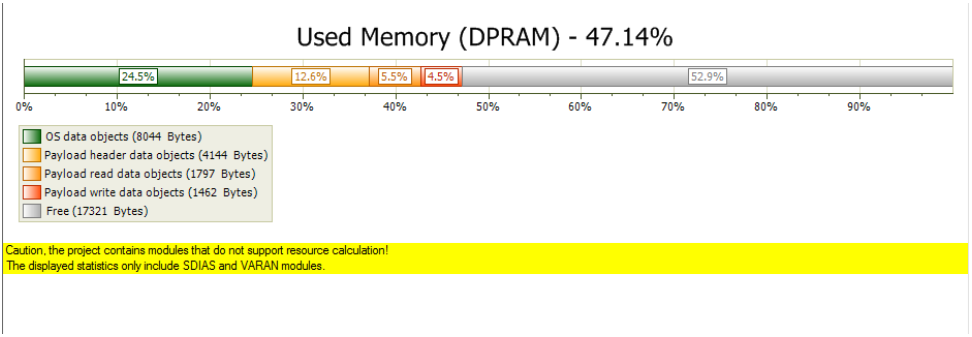


Below the total consumption of the VARAN Data Objects memory (DPRAM) is displayed, together with the detailed layout.



The individual consumers (data objects of the operating system and the application) and the free space are listed.

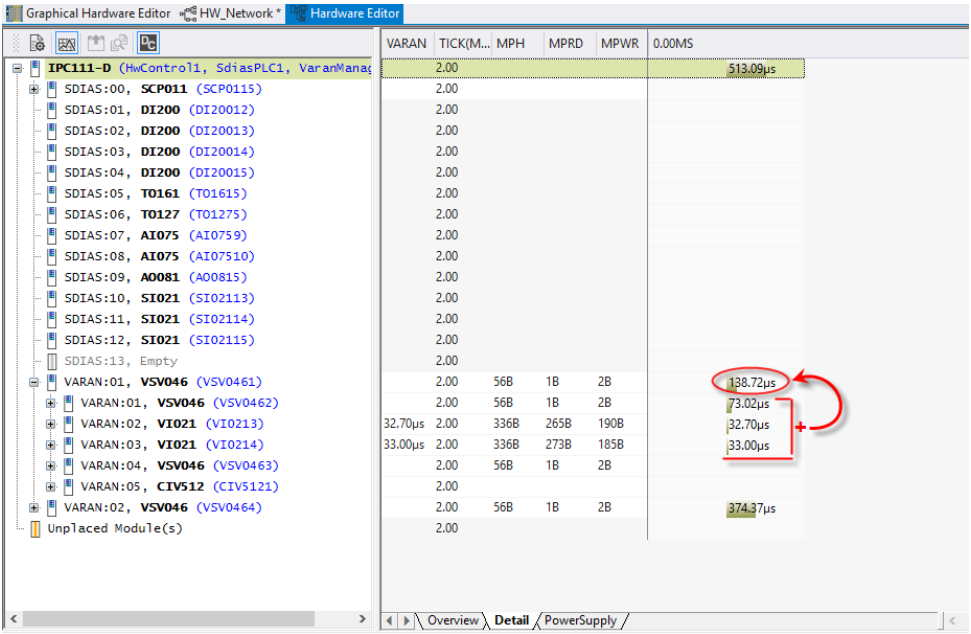
If there are unsupported modules (e.g. DIAS) in the project, an error message appears:



For modules that are not supported nothing is calculated!

4.2.12.2 Detail (VARAN)

On the **Detail (VARAN)** tab, the consumption data of the respective modules are displayed.



The meaning of each column:

- **Varan**

shows the **time actually required** by the module for the VARAN communication.

- **Tick(ms)**

shows the time interval in which the VARAN communication for the respective module takes place.

- **Memory**

(Only active when the PLC property **Payload frame** is set to **No Payload Frames**): shows the **memory consumption of the VARAN data objects**

- **MPH (Memory Payload Header object data)**

(Only active when the PLC property **Payload Frame** is set to **Use Payload Frames**): shows the **memory consumption of the Header of the Payload VARAN data objects**

- **MPRd (Memory Payload Read object data)**

(Only active when the PLC property **Payload Frame** is set to **Use Payload Frames**): shows the **memory consumption of the Read Payload VARAN data objects**

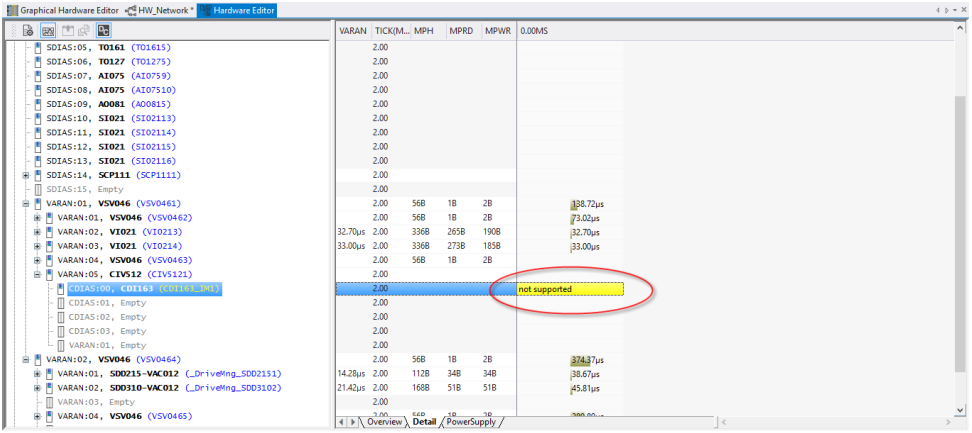
- **MPWr (Memory Payload Write object data)**

(Only active when the PLC property **Payload Frame** is set to **Use Payload Frames**): shows the **memory consumption of the Write Payload VARAN data objects**

- **All following columns (0.00 ms...)**

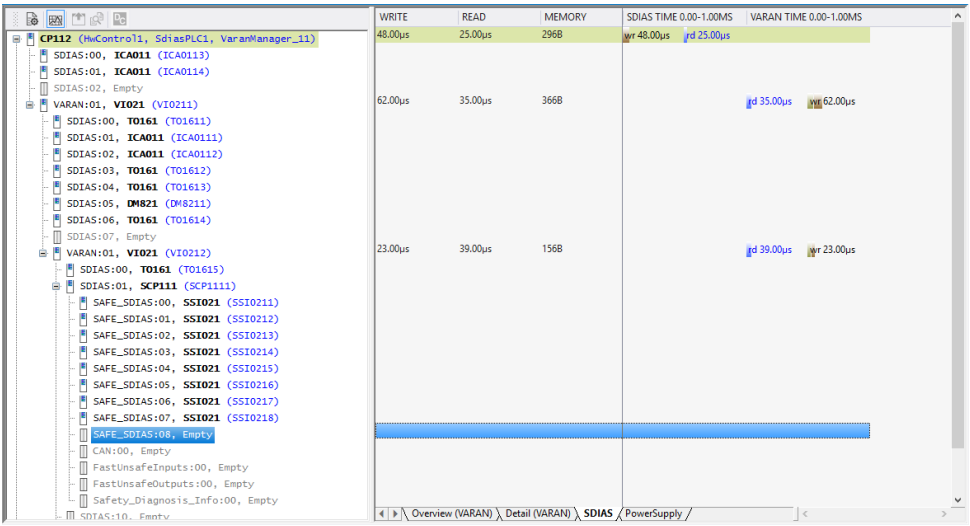
shows the required **VARAN communication time** of the module and all connected child modules at the **respective moment**

Also on the **Detail tab** it is referred to the supported modules:



4.2.12.3 SDIAS

On the **SDIAS** tab the consumption data are displayed interpreted in terms of the corresponding SDIAS modules.



The calculation of the SDIAS resources **only works offline**. **Real online values cannot be displayed**.

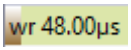
The claculated values for the SDIAS bus are displayed **either on the PLC or on the extension modules** (e.g. VI 021).

The meaning of the single columns:

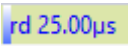
- **WRITE**
  needed time for the writing SDIAS accesses in µs
- **READ**
  needed time for the reading SDIAS accesses in µs
- **MEMORY**
  needed memory for the SDIAS modules in bytes

Additionally the reading and writing start for the SDIAS modules is displayed. Here, a distinction mus be made between PLC and extension module. For the **PLC** the accesses depend on the **SDIAS cycle time** and the **SDIAS-IsoStartTime**. For the **extension modul** the accesses depend on the **VaranManagerTime** and the **Varan-IsoStartTime**.

- **SDIAS Time (Display for PLC)**

  The **writing** accesses **start at the end of the realtime but before the IsoStartTime** on the SDIAS (**Realtime** is **not known** by the value). The display is a **brown bar** and the text **wr <Wert> µs**: 

  The bar gets red, if the access time is too long; the writing **accesses must be finished until the SDIAS IsoStartTime**.

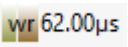
  The **reading** accesses **start with the IsoStartTime** on the SDIAS. The display is a **blue bar** and the text **rd <Wert> µs**: 

  The bar gets red, if the access time is too long; the reading **accesses must be finished until the end of the SDIAS cycle time**.

- **VARAN Time (Display for Extension Module)**

  The **reading** accesses **start with the realtime tick (here 0)**. The display is a **blue bar** and the text **rd <Wert> µs**: 

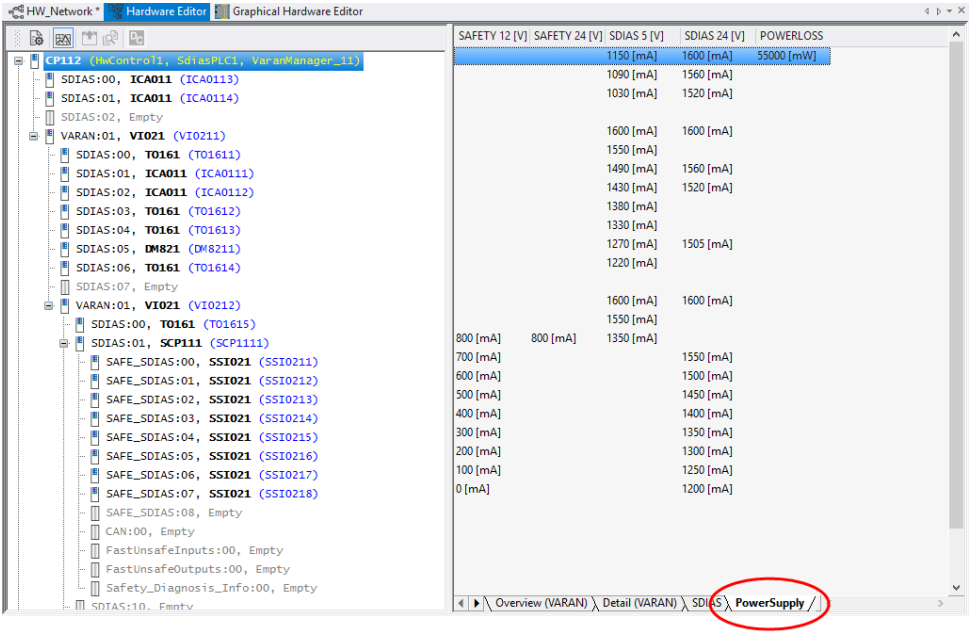
  The bar gets red, if the access time is too long; the reading **accesses must be finished until the Varan IsoStartTime**.

  The **writing** accesses **start at the end of the Varan time**. The display is a **brown bar** behind the bar for the Varan time and the text **wr <Wert> µs**: 

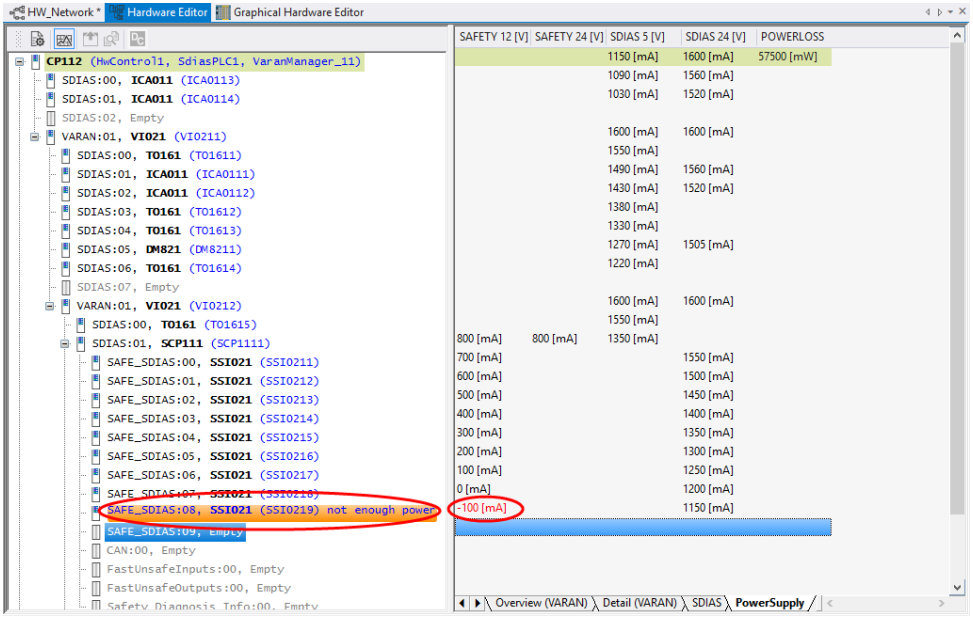
  The bar gets red, if the access time is too long; the reading **accesses must be finished until the next realtime tick**.

4.2.12.4 Power Supply

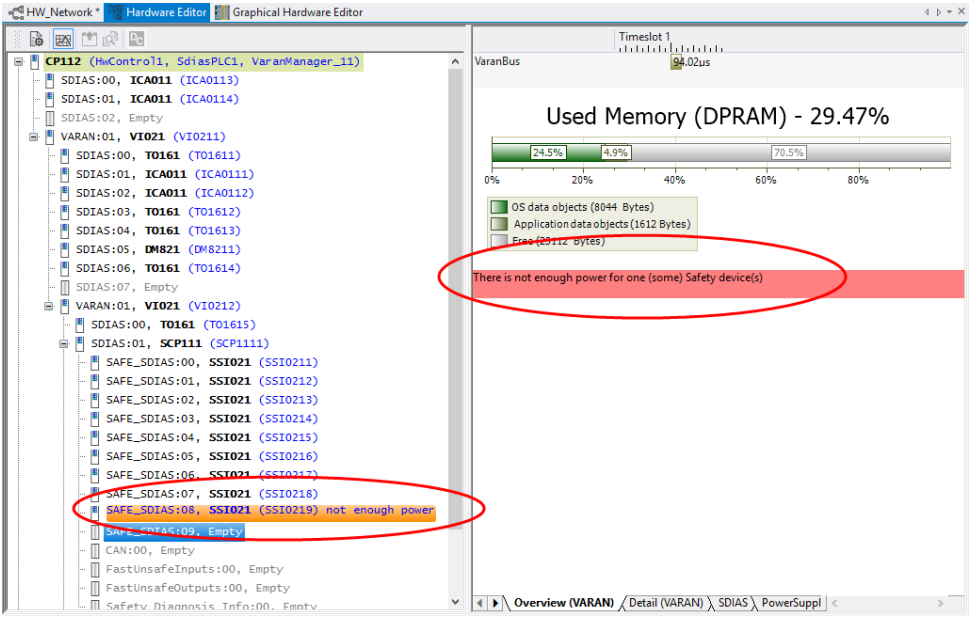
In the **PowerSupply** tab the sum current calculation incl. leakage currents is displayed. Per - in the LHD file - defined current strength and bus type, an own columns is displayed, where the current available for the next device is shown. In the last column the power loss is displayed.



If a module has too little power supply, it is colored orange, ...



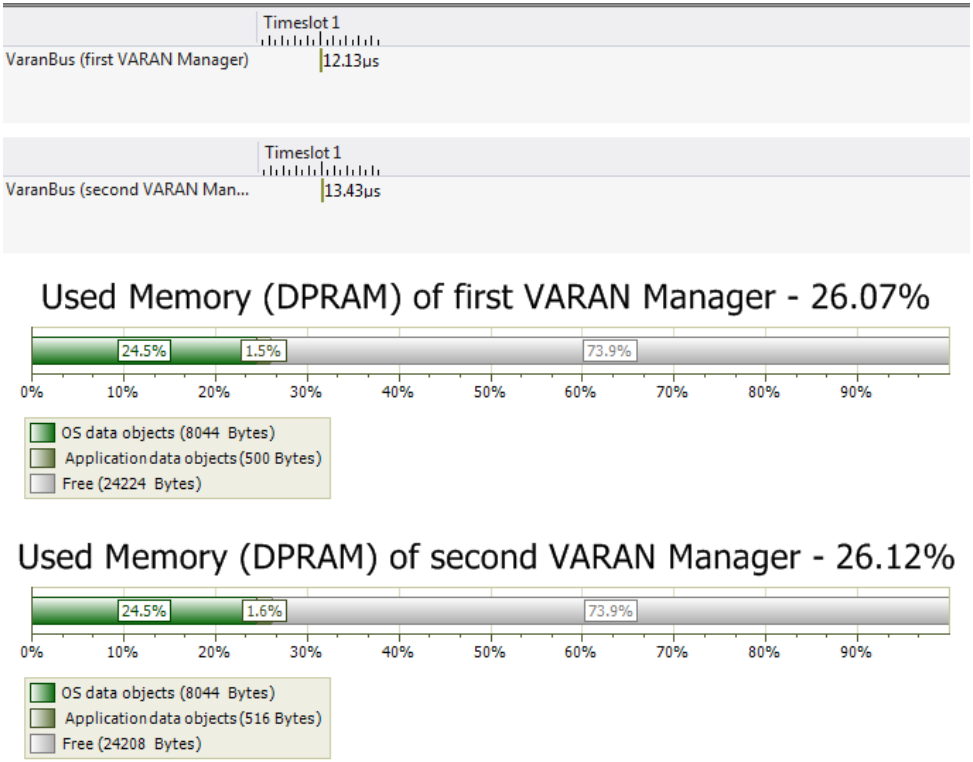
... and also in the overview an according error message is shown.


HW_Network *
Hardware Editor
Graphical Hardware Editor
CP112 (HwControl1, SdiasPLC1, VaranManager_11)
SDIAS:00, ICA011 (ICA0113)
SDIAS:01, ICA011 (ICA0114)
SDIAS:02, Empty
VARAN:01, VI021 (VI0211)
SDIAS:00, TO161 (TO1611)
SDIAS:01, ICA011 (ICA0111)
SDIAS:02, ICA011 (ICA0112)
SDIAS:03, TO161 (TO1612)
SDIAS:04, TO161 (TO1613)
SDIAS:05, DM821 (DM8211)
SDIAS:06, TO161 (TO1614)
SDIAS:07, Empty
VARAN:01, VI021 (VI0212)
SDIAS:00, TO161 (TO1615)
SDIAS:01, SCP111 (SCP1111)
SAFE_SDIAS:00, SSI021 (SSI0211)
SAFE_SDIAS:01, SSI021 (SSI0212)
SAFE_SDIAS:02, SSI021 (SSI0213)
SAFE_SDIAS:03, SSI021 (SSI0214)
SAFE_SDIAS:04, SSI021 (SSI0215)
SAFE_SDIAS:05, SSI021 (SSI0216)
SAFE_SDIAS:06, SSI021 (SSI0217)
SAFE_SDIAS:07, SSI021 (SSI0218)
SAFE_SDIAS:08, SSI021 (SSI0219) not enough power
CAN:00, Empty
FastUnsafeInputs:00, Empty
FastUnsafeOutputs:00, Empty
Timeslot 1
VaranBus
94.02us
Used Memory (DPRAM) - 29.47%
24.5%
4.9%
70.5%
0%
20%
40%
60%
80%
OS data objects (8044 Bytes)
Application data objects (1612 Bytes)
Free (29112 Bytes)
There is not enough power for one (some) Safety device(s)
Overview (VARAN)
Detail (VARAN)
SDIAS
PowerSuppl

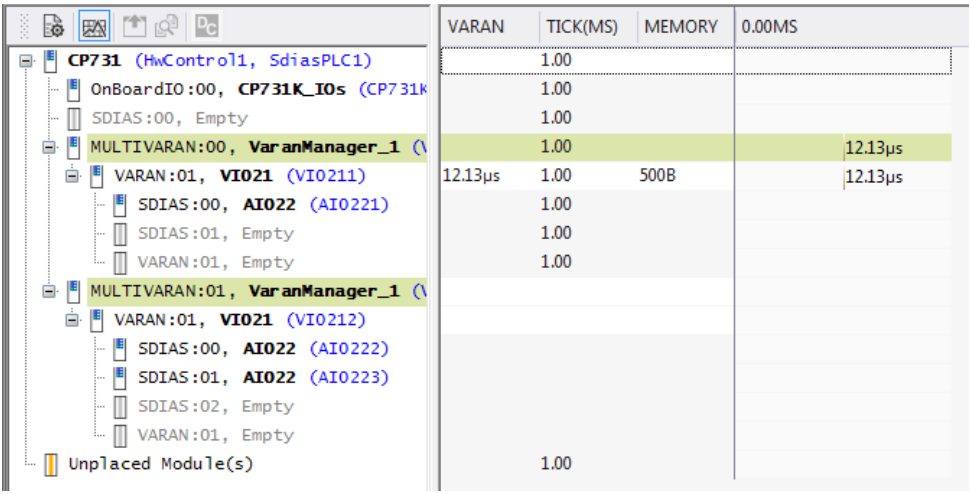
4.2.12.5 Showing the Resources for 2 VaranManagers

If the selected PLC has 2 VaranManagers, in the resource view the data are displayed separately for the two VaranManagers and separately for the CPU cores.

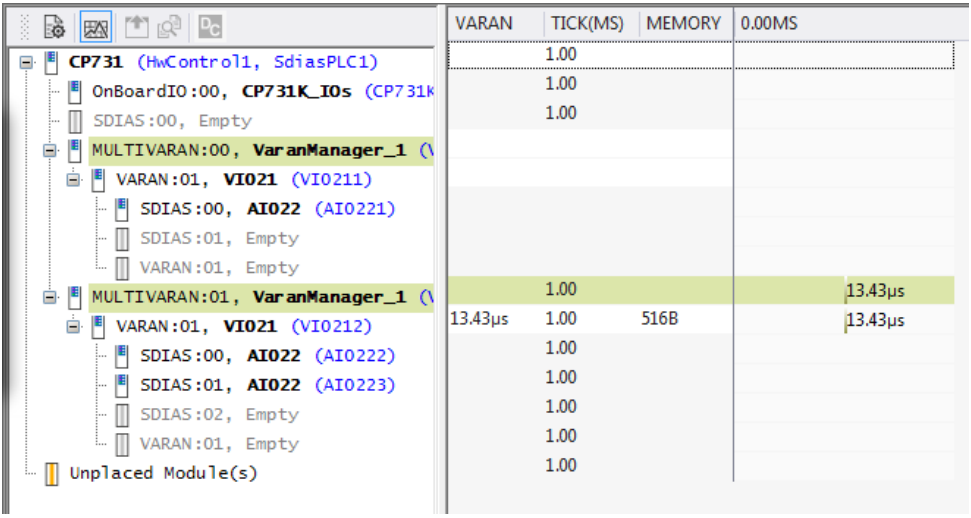
Display in the Resource Overview:



Display of VaranManager 1 in the Resource Detail:



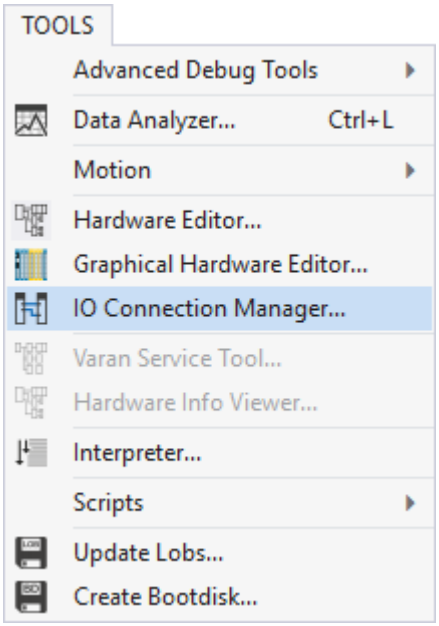
Display of VaranManager 2 in the Resource Detail:



4.2.13 IO Connection Manager

The **Connection Manager** is an organic component of the **Hardware Editor** and offers an additional view of the LASAL object relationships, which is especially suited for hardware connections. It also provides the option to create so-called temporary or **IO objects** and place them in the network. The interaction with the network view and the clarity are increased.

The Connection Manager is started using the  icon in the tool bar through the menu item **Tools / IO Connection Manager**, ...

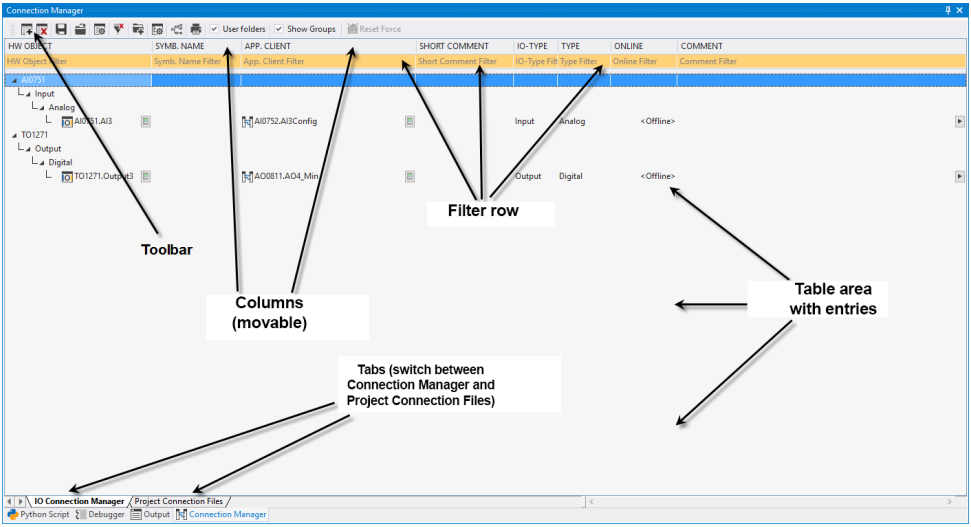


... or as previously mentioned, from the **Hardware Editor** with the context menu. With a right click on an IO object, the Connection Manager can also be started from the **network view**. A new tab opens now in the lower area next to **PythonScript**, **Debugger** and **Output**: the **IO Connection Manager** window.

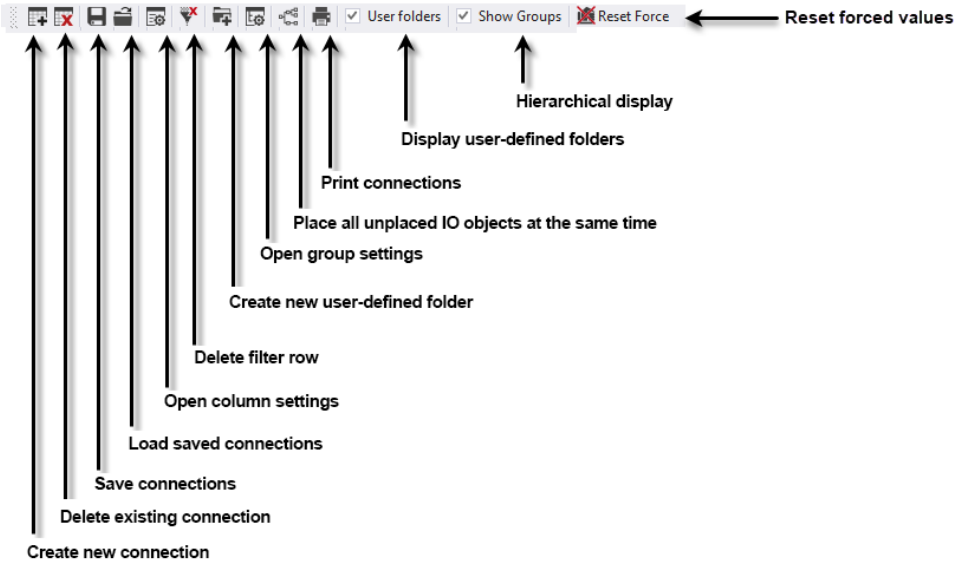
If **displayable connections** are already available for the Connection Manager, they are **automatically assumed** and displayed.

4.2.13.1 IO Connection Manager Areas

The various areas in the Connection Manager are:



The following functions are available in the **toolbar** of the Connection Manager:



Create New Connection

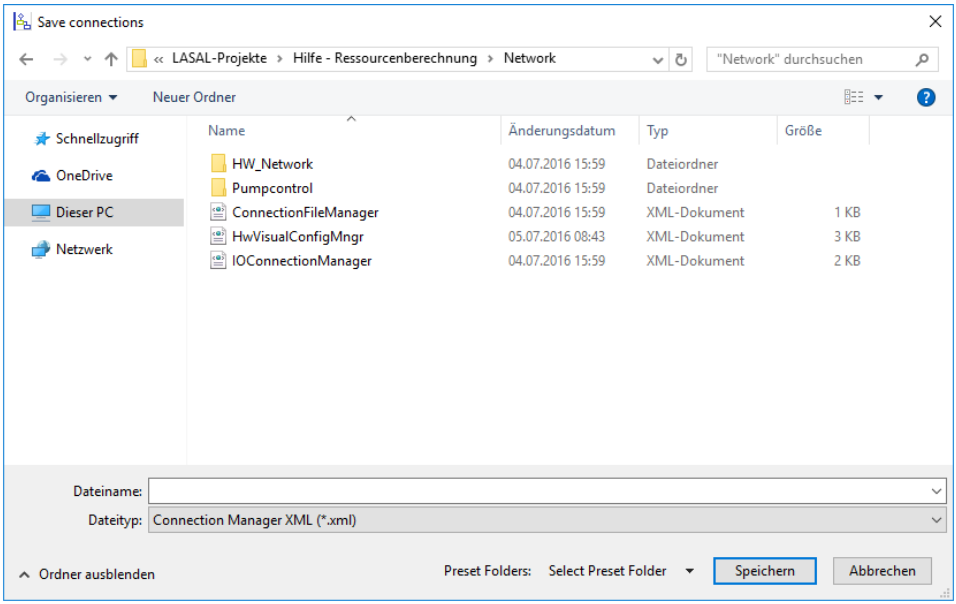
A new, empty field is created in the table area. All columns, except the Online column, can be edited.

Delete existing Connection

Delete an existing (and selected) connection. Delete connection

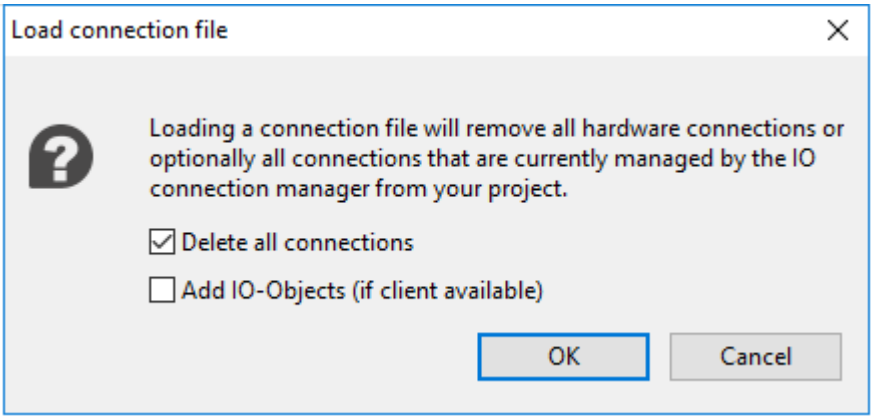
Save Connections

By clicking the  icon, the "Save connections" dialog opens. Enter the file name or select an existing file to save all entries in the Connection Manager as an *.XML file.



Load stored connections

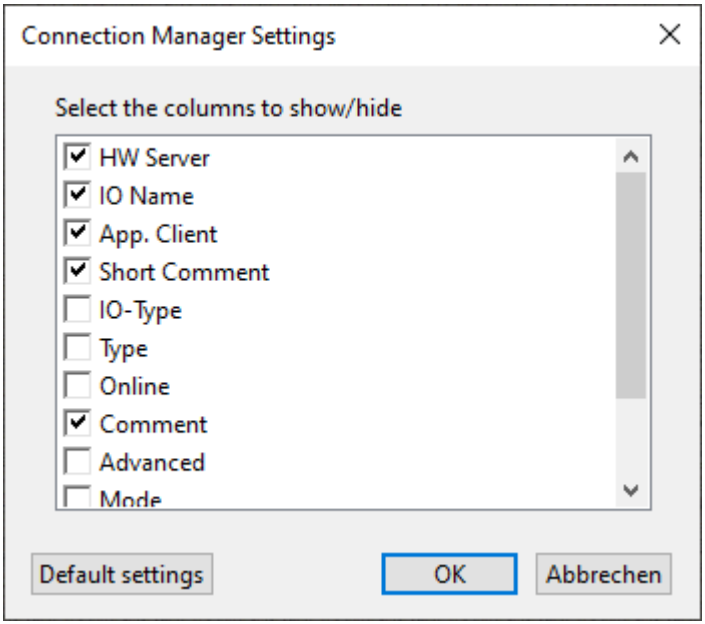
By clicking the icon  the file selection dialog opens, where you can select and load a saved connection file in *.XML format. After selecting the file, first a warning is triggered, which states that by loading connections, either all hardware connections or optionally all connections contained in the ConnectionManager (including non-hardware connections in the connection manager!) will be deleted: Additionally you can select, that the IO objects are created automatically (if the according clients exist):



The saved connections are loaded, if you confirm the query with OK.

Call column settings

Opens the column settings dialog:



In this window, all columns to display can be selected or deselected as desired. After clicking the **"Default Settings"**, only **HW Server**, **IO Name**, **App. Client** and **Short Comment** are now checked. If all columns are now deselected, the default setting is used. With **OK**, all selected settings are enabled; with **Cancel**, they are rejected.

Delete filter line

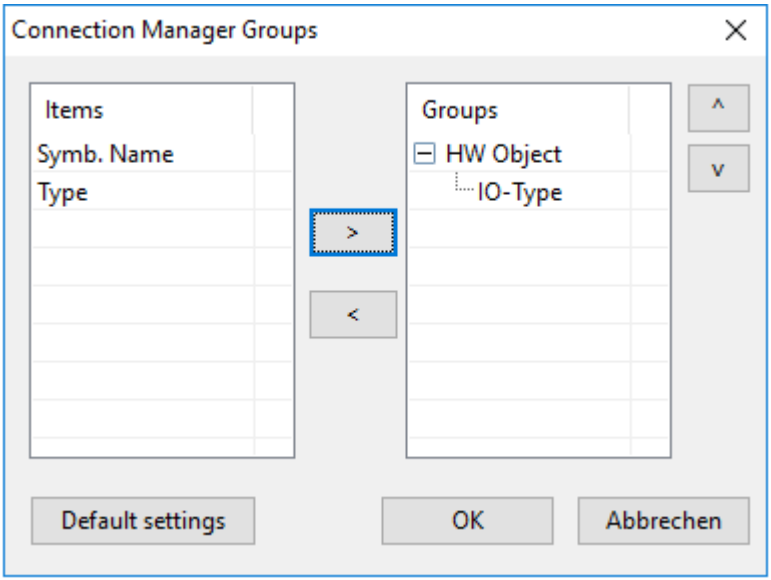
The filter settings are deleted from all columns.

Create new user-defined folder

A new user-defined folder is added. This **button is only enabled**, when the **User folders** checkbox is **checked**. User-defined folders can also be nested.

Call group settings

The dialog for selecting the displayed hierarchy levels is called. The **button is always active**, regardless of whether the **Show Groups** checkbox is checked or not.



On the **left side, all selectable hierarchy levels** are located. On the right are the selected levels. With the arrow buttons in the middle, groupings for individual levels can be added or removed. The first level added is always at the top of the hierarchy, all other

levels are added sequentially. The **selected hierarchy levels** can be **moved as desired** within the sequence using the vertical arrow button on the right, after the corresponding entry was selected in the Groups column.

The **Default settings** button moves all entries to the left in the Items column (removes the hierarchical grouping display).

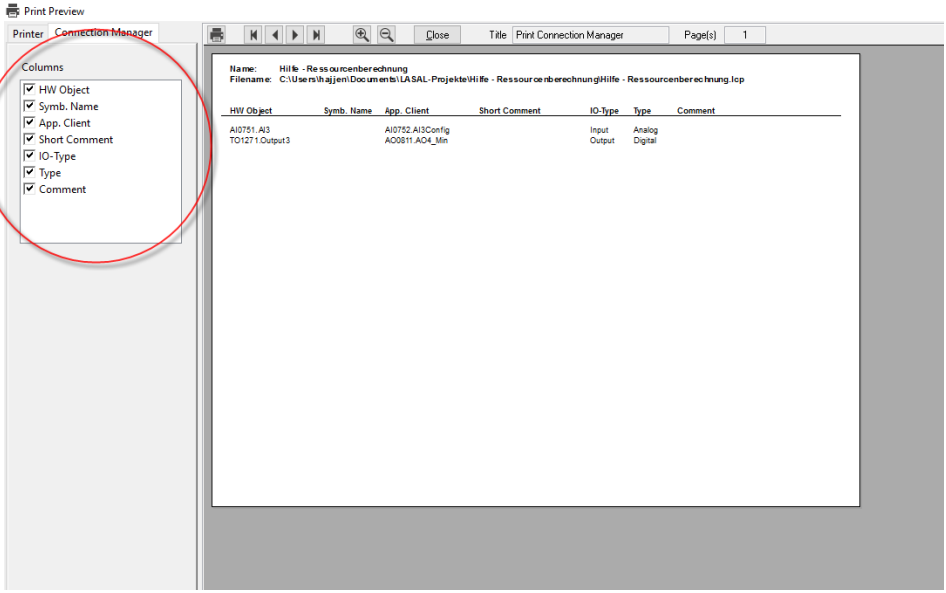
To see the defined grouping, the **Show Groups** box must be checked.

Placing All Unplaced IO Objects Simultaneously

From all connections that have already entered a symbolic name and client, but are still without an IO object, the IO objects are placed next to their clients in the object network.

Printing the CM Connections

A print dialog opens, in which all connections can be printed. Sorting, column sequence and filtering are assumed. The sequence of the columns can be set in a separate tab.

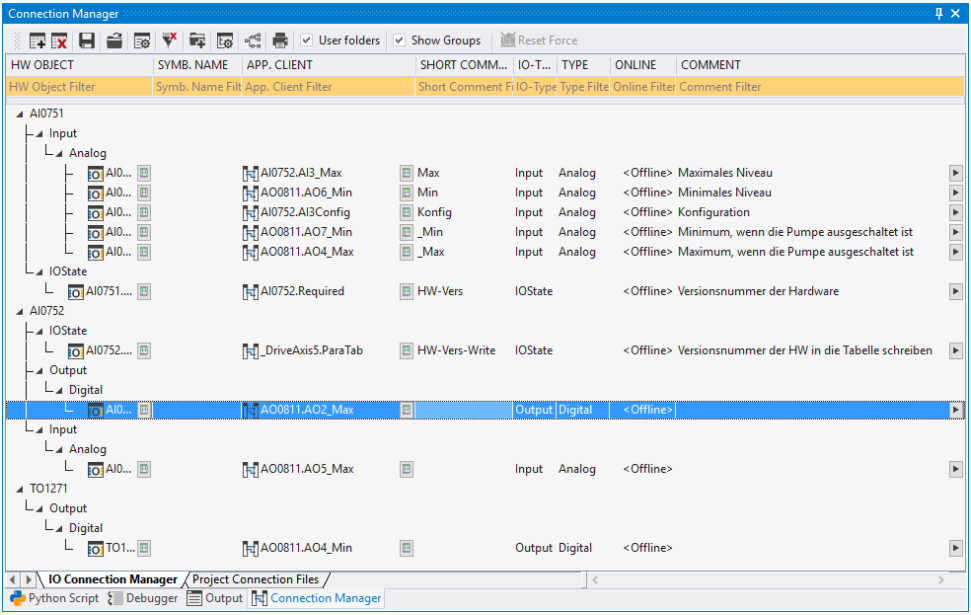


Show user-defined folders (user folders)

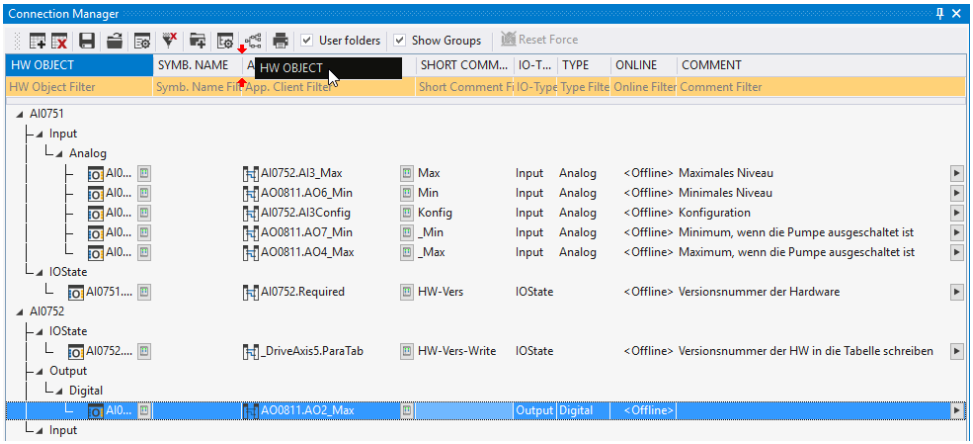
Enables/disables the display of user-defined subfolders (spaces are allowed in the folder name). Per multi-selection, several CM connections can be moved to a user-defined folder at the same time.

Hierarchical display (Show Groups)

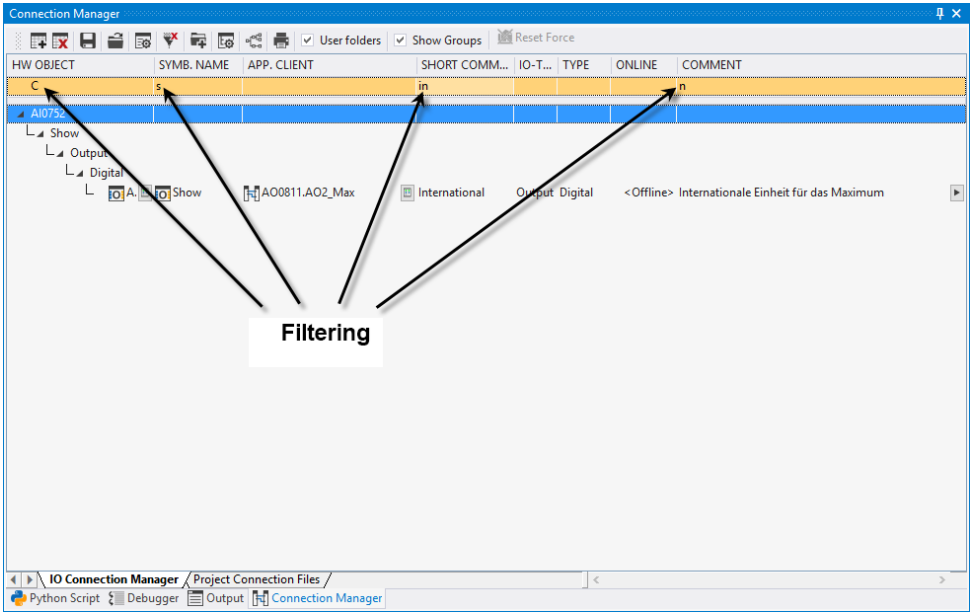
Enables/disables the hierarchical display. If one of the two checks is active, the list of connections is displayed as a tree.



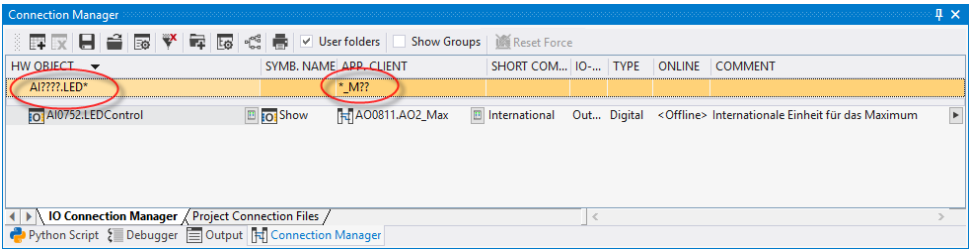
The **Column buttons** are **sortable** and can be easily **moved**. With a **click** on a column heading, the respective column can be sorted individually; clicking on the same column heading again reverses the sorting direction. To **shift a column**, click on the column header and hold the **left mouse button** while moving the mouse. The target location is indicated by a red arrow above and below the column heading.



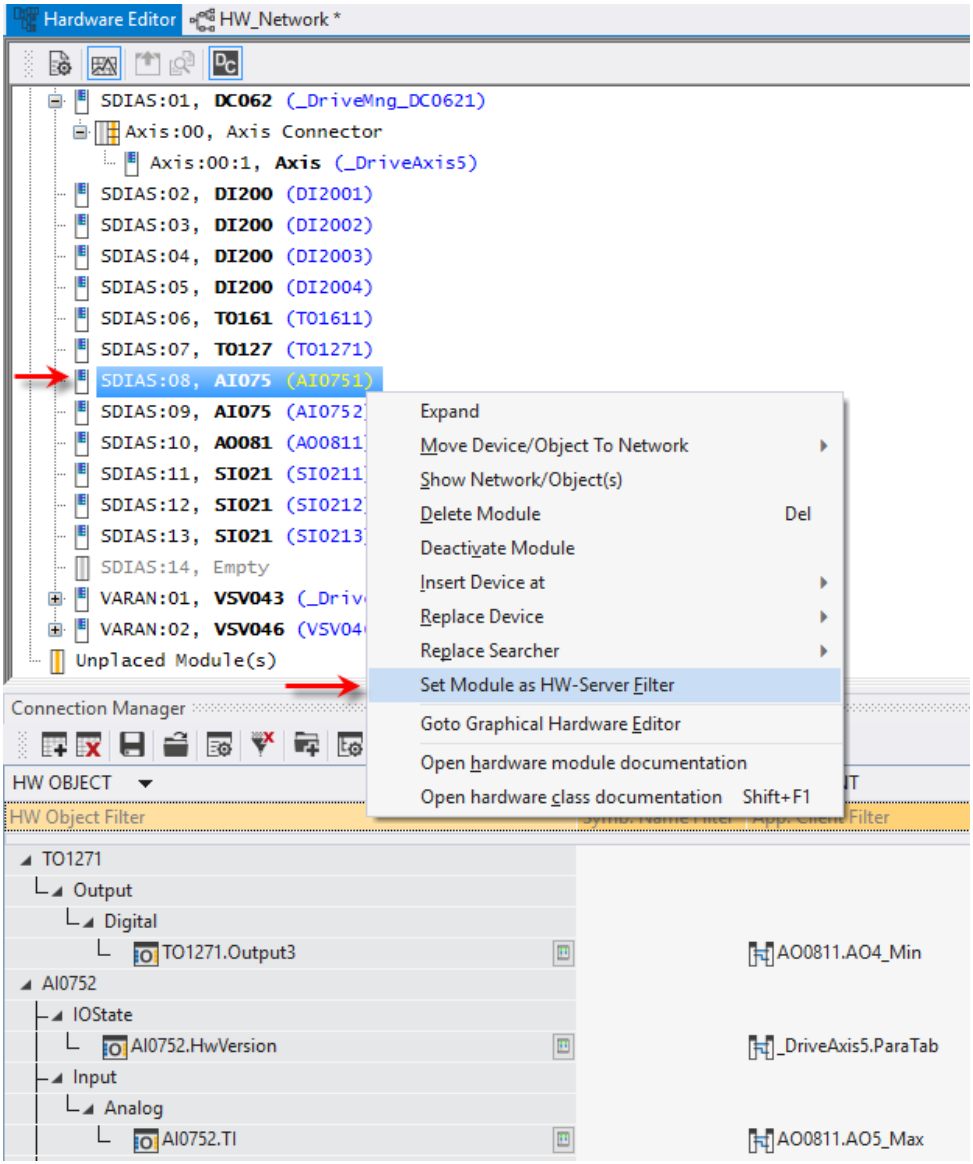
The yellow **filter line** is especially suited for filtering the correct entries out of a large number of connections with just a few clicks. **Combined filtering** – over several columns – is also possible, as well as searching by letter groups **within** the word:

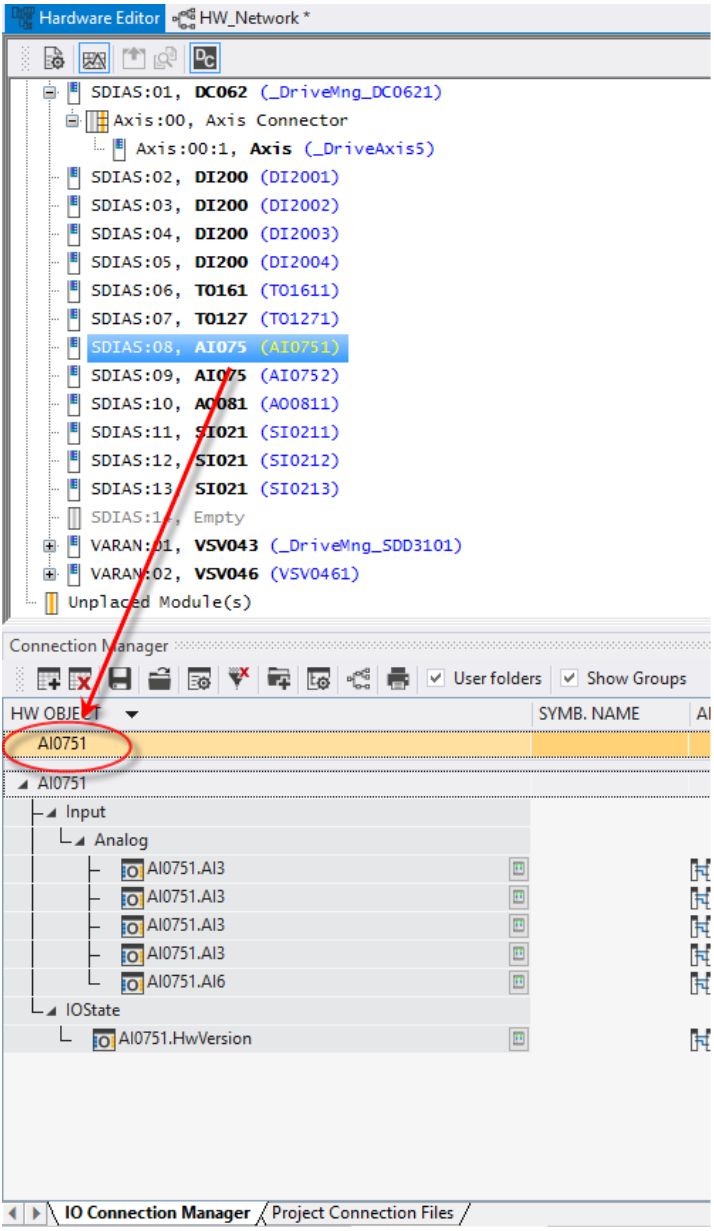


A search can also be performed with wildcards (*, ?):



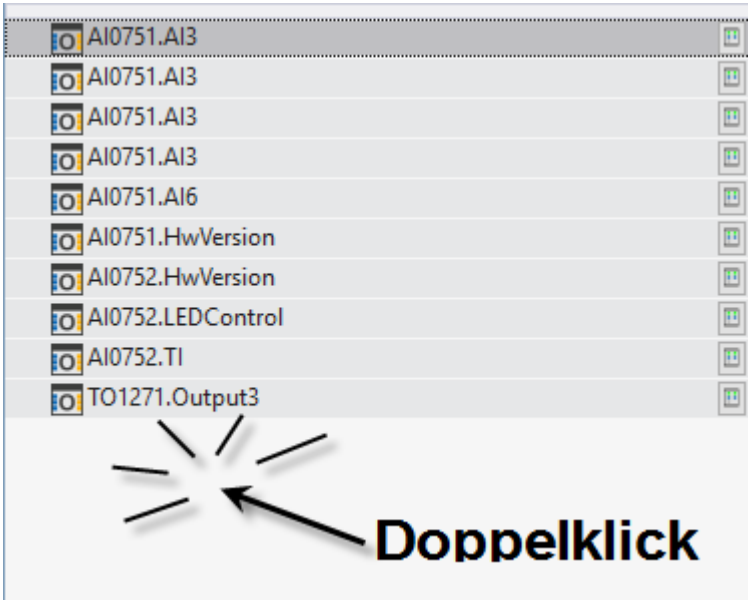
As previously mentioned, this can also be in the HW object column as a filter from the context menu of module.

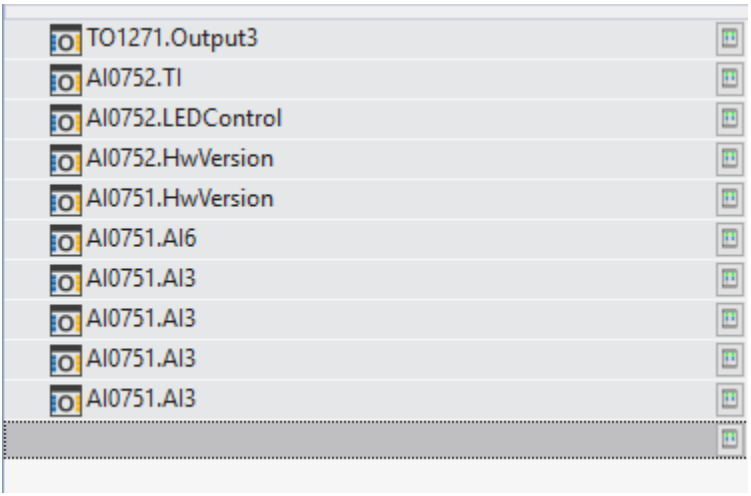

Hardware Editor
HW_Network *
SDIAS:01, DC062 (_DriveMng_DC0621)
Axis:00, Axis Connector
Axis:00:1, Axis (_DriveAxis5)
SDIAS:02, DI200 (DI2001)
SDIAS:03, DI200 (DI2002)
SDIAS:04, DI200 (DI2003)
SDIAS:05, DI200 (DI2004)
SDIAS:06, TO161 (TO1611)
SDIAS:07, TO127 (TO1271)
SDIAS:08, AI075 (AI0751)
SDIAS:09, AI075 (AI0752)
SDIAS:10, AO081 (AO0811)
SDIAS:11, SI021 (SI0211)
SDIAS:12, SI021 (SI0212)
SDIAS:13, SI021 (SI0213)
SDIAS:14, Empty
VARAN:01, VSV043 (_Driv
VARAN:02, VSV046 (VSV04
Unplaced Module(s)
Expand
Move Device/Object To Network
Show Network/Object(s)
Delete Module Del
Deactivate Module
Insert Device at
Replace Device
Replace Searcher
Set Module as HW-Server Filter
Goto Graphical Hardware Editor
Open hardware module documentation
Open hardware class documentation Shift+F1
Connection Manager
HW OBJECT
HW Object Filter
TO1271
Output
Digital
TO1271.Output3
AO0811.AO4_Min
AI0752
IOState
AI0752.HwVersion
_DriveAxis5.ParaTab
Input
Analog
AI0752.TI
AO0811.AO5_Max


Hardware Editor
HW_Network *
SDIAS:01, DC062 (_DriveMng_DC0621)
Axis:00, Axis Connector
Axis:00:1, Axis (_DriveAxis5)
SDIAS:02, DI200 (DI2001)
SDIAS:03, DI200 (DI2002)
SDIAS:04, DI200 (DI2003)
SDIAS:05, DI200 (DI2004)
SDIAS:06, TO161 (TO1611)
SDIAS:07, TO127 (TO1271)
SDIAS:08, AI075 (AI0751)
SDIAS:09, AI075 (AI0752)
SDIAS:10, AO081 (AO0811)
SDIAS:11, SI021 (SI0211)
SDIAS:12, SI021 (SI0212)
SDIAS:13, SI021 (SI0213)
SDIAS:14, Empty
VARAN:01, VSV043 (_DriveMng_SDD3101)
VARAN:02, VSV046 (VSV0461)
Unplaced Module(s)
Connection Manager
User folders
Show Groups
HW OBJECT
SYMB. NAME
AI0751
AI0751
Input
Analog
AI0751.AI3
AI0751.AI3
AI0751.AI3
AI0751.AI3
AI0751.AI6
IOState
AI0751.HwVersion
IO Connection Manager
Project Connection Files

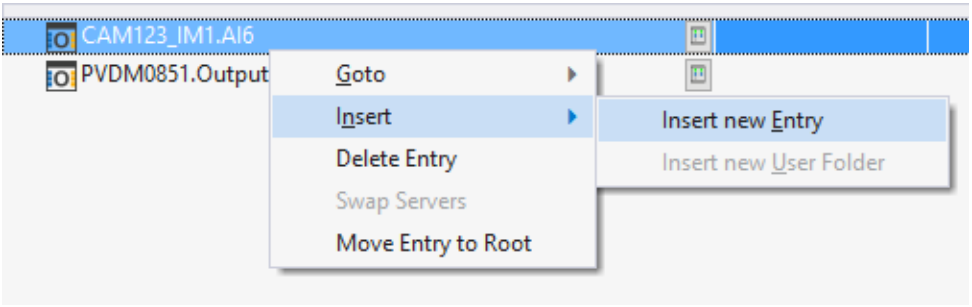
4.2.13.2 Entering Connections in the IO Connection Manager

There are also **other options** than clicking on the  icon to create a new connection in the Connection Manager. With a **double click** in the empty area at the **end of the table**, a new connection is automatically created at the end of the list.

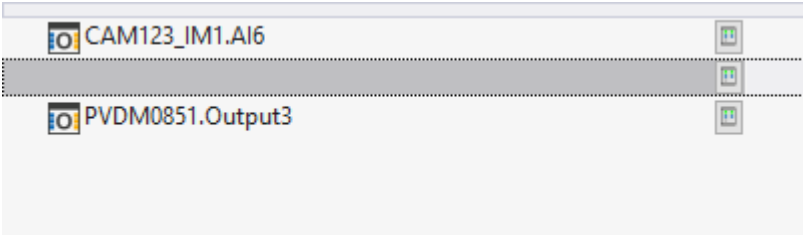




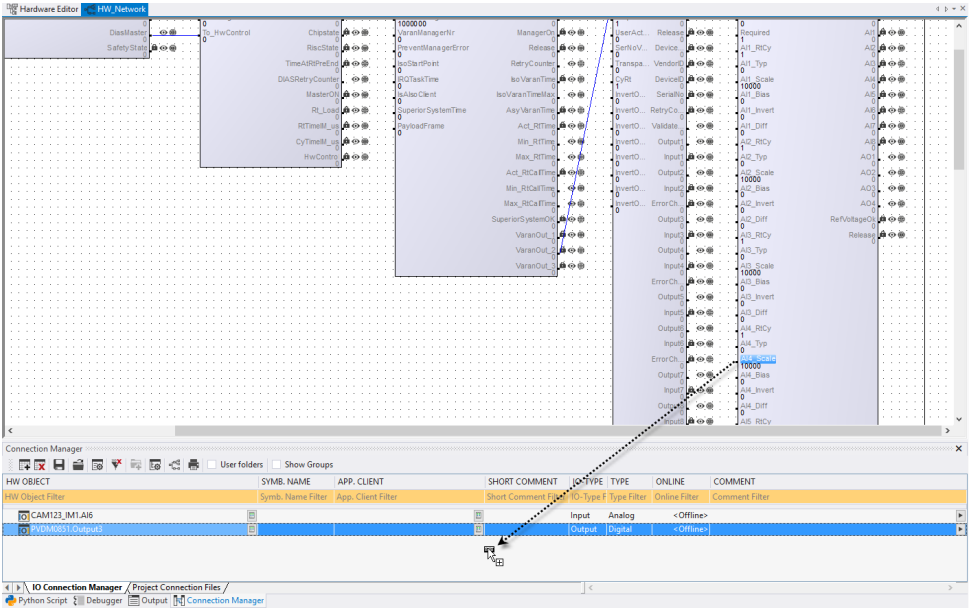
With a right click on any connection, select "**Insert / Insert new Entry**" **from the context menu** and thereby create a new connection (directly under the selected entry, if no sorting was executed before) in any location.

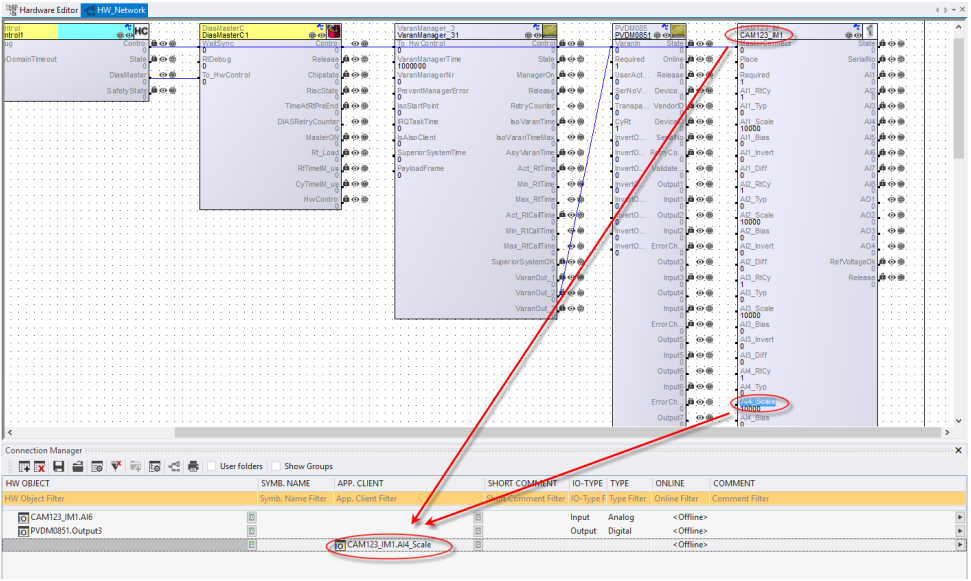


The new connection is added directly under the selected line - if it was not sorted for another criteria before. The same can be achieved, when the **Add button** is pressed for the selected line.



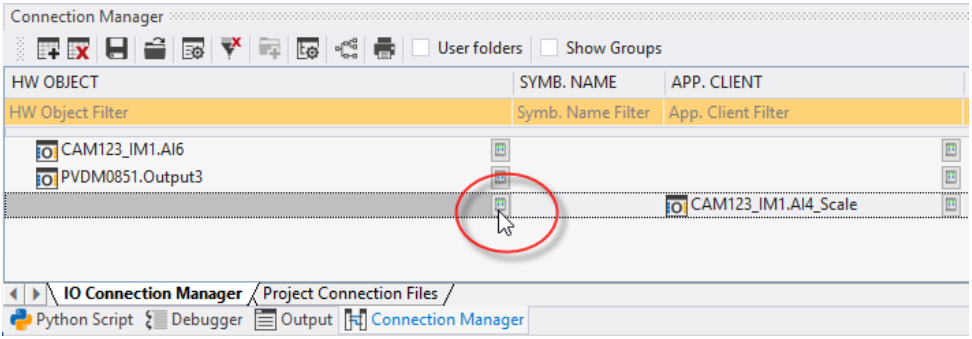
A fourth option to generate a new line in the connection manager is to **drag a valid server / client from the network view** and drop it in a free space in the Connection Manager.

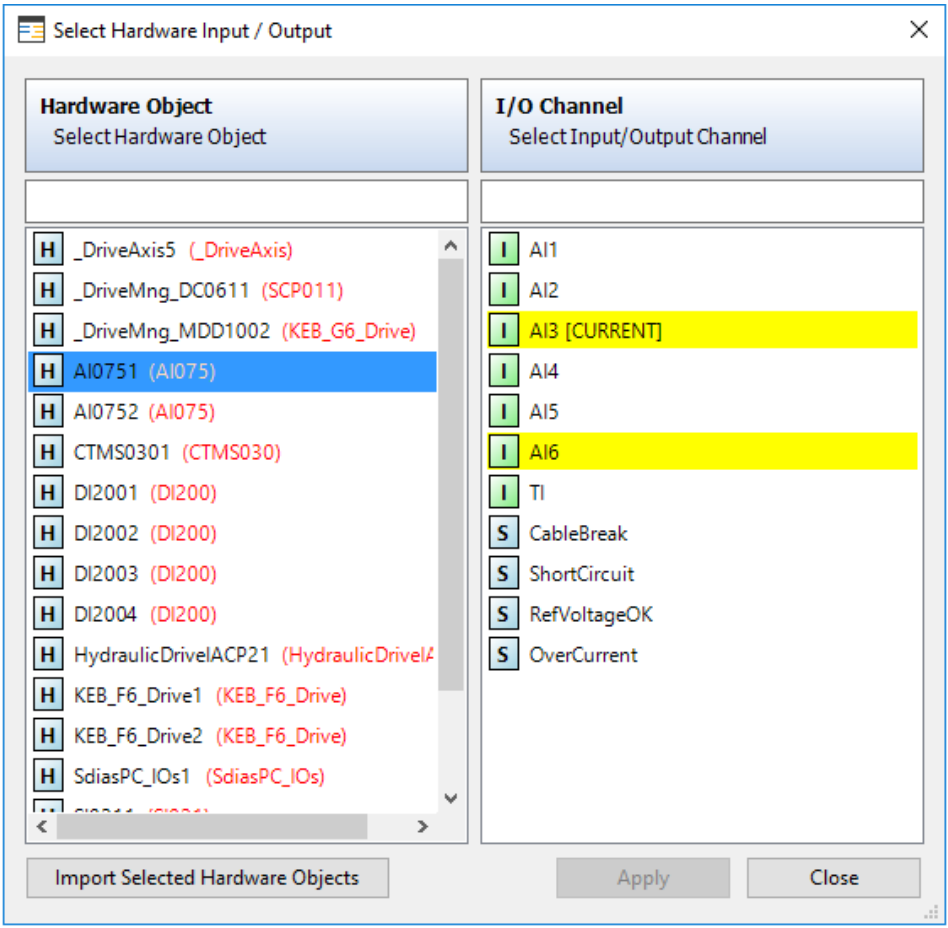




If a client is **placed on the symbolic name** of an existing connection, **a new connection is created**. If the client is **dropped** over the client field however, the **Client connection is replaced**.

Once a new entry is created, the server, client and IO names, etc. can be entered. For servers and clients, there is a **pre-select** that can be made with a click on the **select button**.

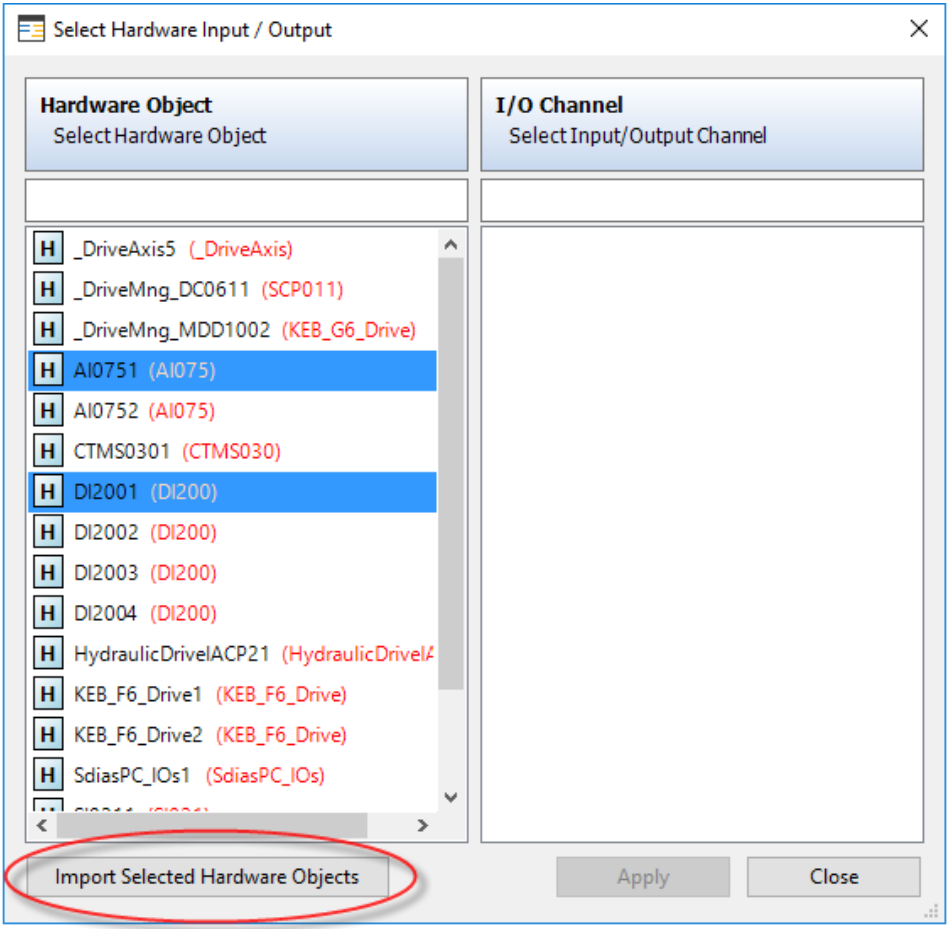




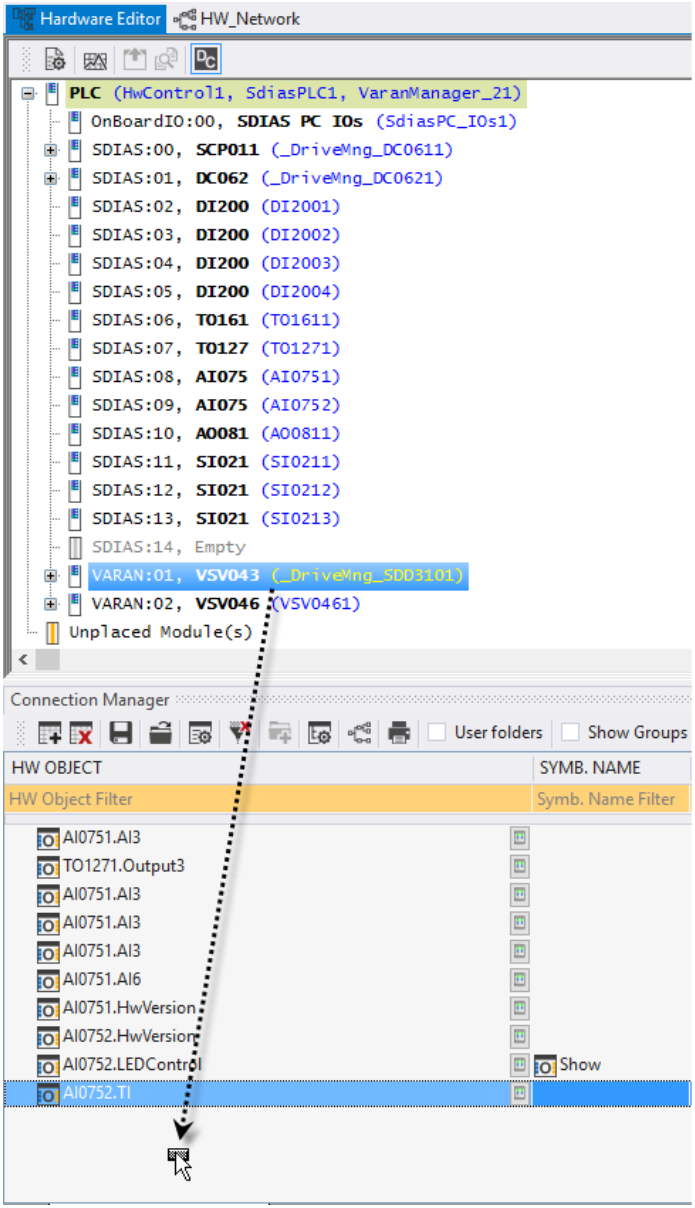
On the **left side** (in this example: sever selection), all **HW objects** that can provide a valid sever are listed. On the **Right**, only the **valid servers of the selected HW objects** are visible. **The connected servers are marked with yell**ow. If the selection is called with an available server, it is no longer marked yellow. Instead, it labeled with the word **[CURRENT]**.

Multi-selection of HW objects is also possible. In this case, no servers are shown on the right side. This mode is enabled when the **Import Selected Hardware Objects** button

is clicked. **All selected HW objects** then **imported** into the Connection Manager, which means a **new connection is created for each in and output**.

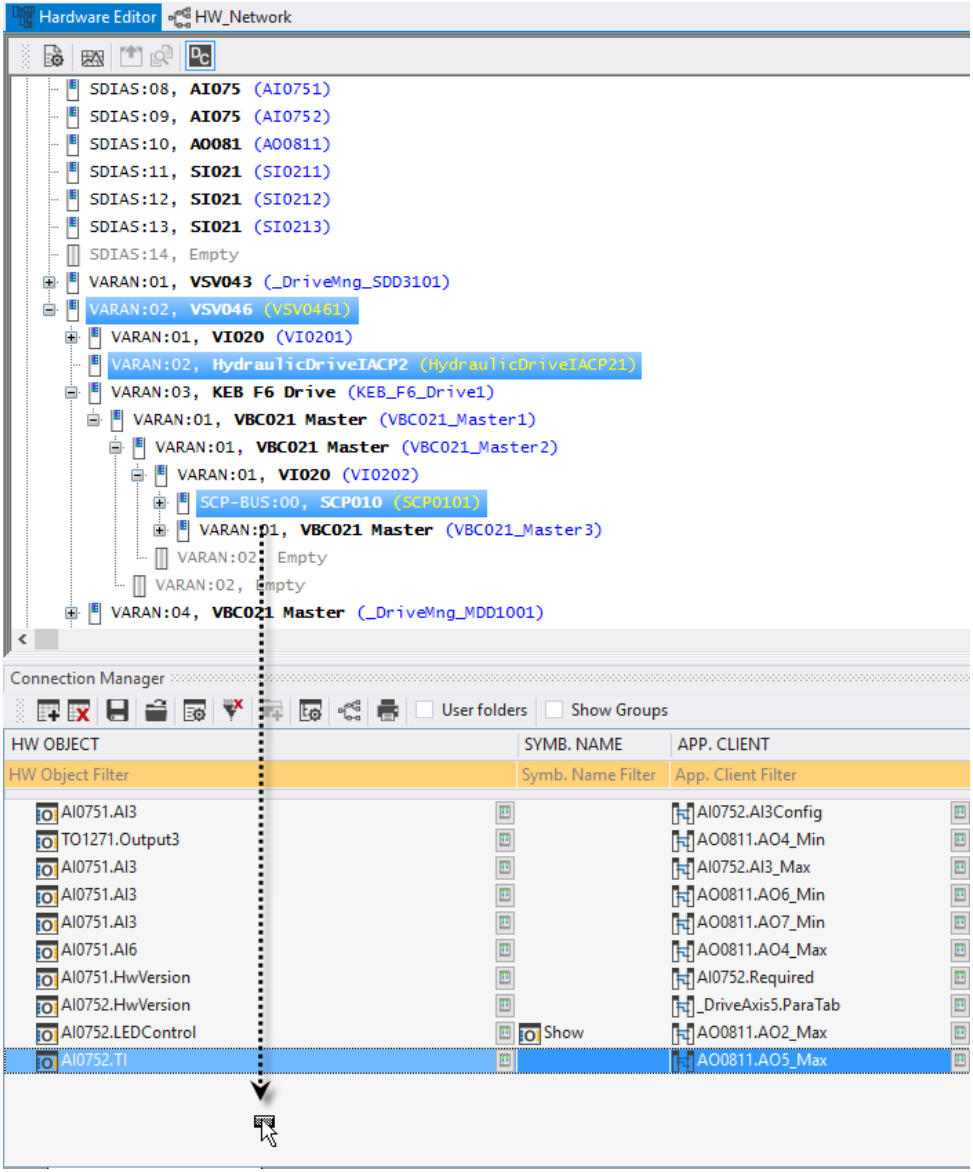


The same result is achieved by **dragging and dropping the selected modules** into the **Connection Manager**:


Hardware Editor
HW_Network
PLC (HwControl1, SdiasPLC1, VaranManager_21)
OnBoardIO:00, SDIAS PC IOs (SdiasPC_IOs1)
SDIAS:00, SCP011 (_DriveMng_DC0611)
SDIAS:01, DC062 (_DriveMng_DC0621)
SDIAS:02, DI200 (DI2001)
SDIAS:03, DI200 (DI2002)
SDIAS:04, DI200 (DI2003)
SDIAS:05, DI200 (DI2004)
SDIAS:06, TO161 (TO1611)
SDIAS:07, TO127 (TO1271)
SDIAS:08, AI075 (AI0751)
SDIAS:09, AI075 (AI0752)
SDIAS:10, AO081 (AO0811)
SDIAS:11, SI021 (SI0211)
SDIAS:12, SI021 (SI0212)
SDIAS:13, SI021 (SI0213)
SDIAS:14, Empty
VARAN:01, VSV043 (_DriveMng_SDD3101)
VARAN:02, VSV046 (VSV0461)
Unplaced Module(s)
Connection Manager
User folders
Show Groups
HW OBJECT
SYMB. NAME
HW Object Filter
Symb. Name Filter
AI0751.AI3
TO1271.Output3
AI0751.AI3
AI0751.AI3
AI0751.AI3
AI0751.AI6
AI0751.HwVersion
AI0752.HwVersion
AI0752.LEDControl
Show
AI0752.TI

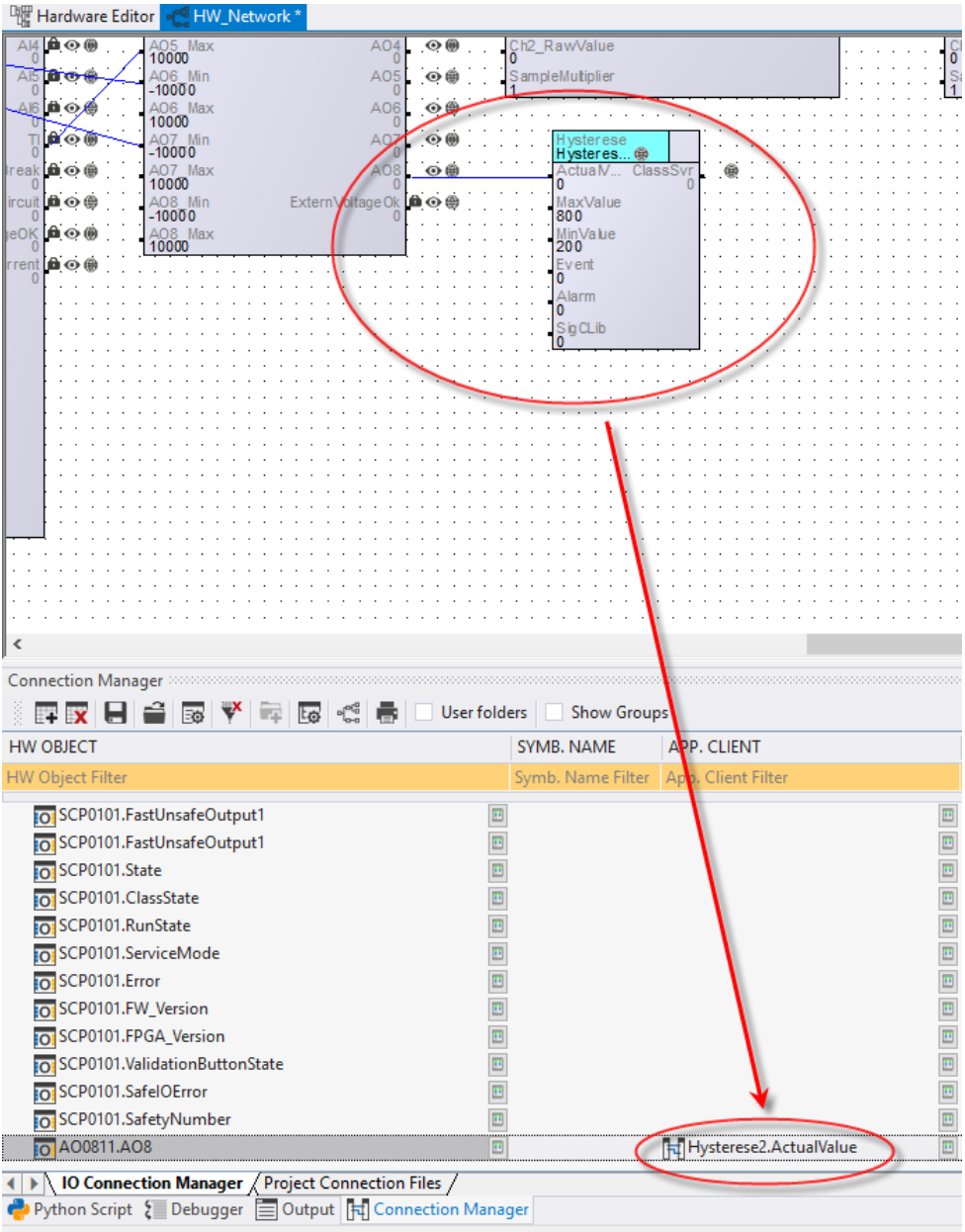
When the left mouse button is released, the corresponding in and outputs are imported.

A selection of unrelated servers from one or more HW objects can also be imported in the same way from the HW Editor through multi selection and Drag & Drop into the Connection Manager.

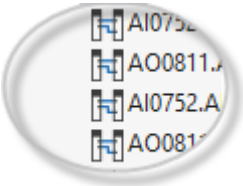

Hardware Editor
HW_Network
SDIAS:08, AI075 (AI0751)
SDIAS:09, AI075 (AI0752)
SDIAS:10, AO081 (AO0811)
SDIAS:11, SI021 (SI0211)
SDIAS:12, SI021 (SI0212)
SDIAS:13, SI021 (SI0213)
SDIAS:14, Empty
VARAN:01, VSV043 (_DriveMng_SDD3101)
VARAN:02, VSV046 (VSV0461)
VARAN:01, VI020 (VI0201)
VARAN:02, HydraulicDriveIACP2 (HydraulicDriveIACP21)
VARAN:03, KEB F6 Drive (KEB_F6_Drive1)
VARAN:01, VBC021 Master (VBC021_Master1)
VARAN:01, VBC021 Master (VBC021_Master2)
VARAN:01, VI020 (VI0202)
SCP-BUS:00, SCP010 (SCP0101)
VARAN:01, VBC021 Master (VBC021_Master3)
VARAN:02, Empty
VARAN:02, Empty
VARAN:04, VBC021 Master (_DriveMng_MDD1001)
Connection Manager
User folders
Show Groups
HW OBJECT
SYMB. NAME
APP. CLIENT
HW Object Filter
Symb. Name Filter
App. Client Filter
AI0751.AI3
AI0752.AI3Config
TO1271.Output3
AO0811.AO4_Min
AI0751.AI3
AI0752.AI3_Max
AI0751.AI3
AO0811.AO6_Min
AI0751.AI3
AO0811.AO7_Min
AI0751.AI6
AO0811.AO4_Max
AI0751.HwVersion
AI0752.Required
AI0752.HwVersion
_DriveAxis5.ParaTab
AI0752.LEDControl
Show
AO0811.AO2_Max
AI0752.TI
AO0811.AO5_Max

Alternatively, the server / client can also be entered **manually**.

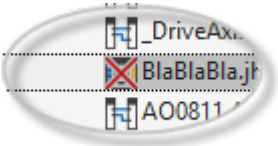
Starting with LASAL CLASS 2 version 02.02.089, connections between classes that have nothing to do with the hardware control can now be managed in the Connection Manager.


Hardware Editor
HW_Network *
AO5_Max
10000
AO6_Min
-10000
AO6_Max
10000
AO7_Min
-10000
AO7_Max
10000
AO8_Min
-10000
AO8_Max
10000
AO4
AO5
AO6
AO7
AO8
ExternVoltageOk
Ch2_RawValue
SampleMultiplier
Hysterese
Hysteres...
ActualV...
ClassSvr
MaxValue
800
MinValue
200
Event
Alarm
SigCLib
Connection Manager
User folders
Show Groups
HW OBJECT
SYMB. NAME
APP. CLIENT
HW Object Filter
Symb. Name Filter
App. Client Filter
SCP0101.FastUnsafeOutput1
SCP0101.FastUnsafeOutput1
SCP0101.State
SCP0101.ClassState
SCP0101.RunState
SCP0101.ServiceMode
SCP0101.Error
SCP0101.FW_Version
SCP0101.FPGA_Version
SCP0101.ValidationButtonState
SCP0101.SafeIOError
SCP0101.SafetyNumber
AO0811.AO8
Hysterese2.ActualValue
IO Connection Manager
Project Connection Files
Python Script
Debugger
Output
Connection Manager

If a **valid connection** has been entered (server and client are legal), the **"Connected" icon** appears in front of the client:

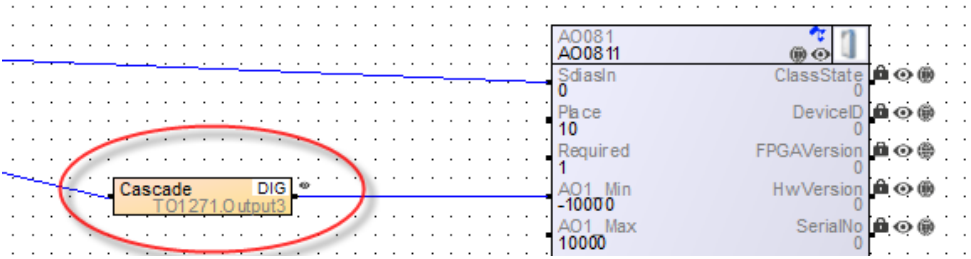


If an **invalid connection** is entered – e.g. **illegal / not existent client** -, an icon appears that is **crossed out** with a red line to indicate the defective connection.



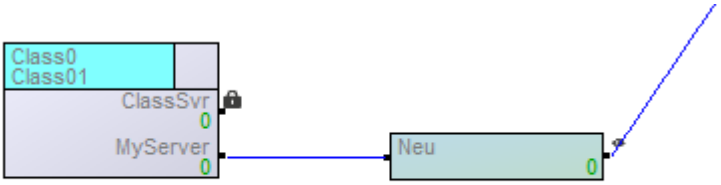
If the **Symb. Name** column is filled, an icon with a red x is shown before the Symb. names, since it does not yet correspond to an **IO object** in the network. The names can now be placed in the network. An **IO object** is thereby created. This view increases the readability of large networks, since the (remote) servers can be placed near the client as an **IO object**. The icon before the IO name is no longer crossed out:



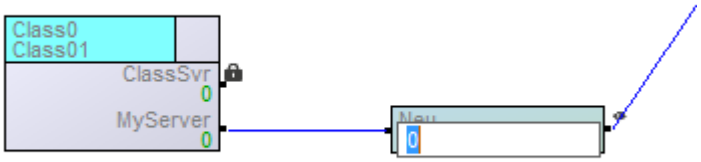


The color and label of the IO object is based on the IO type and the type of connection.

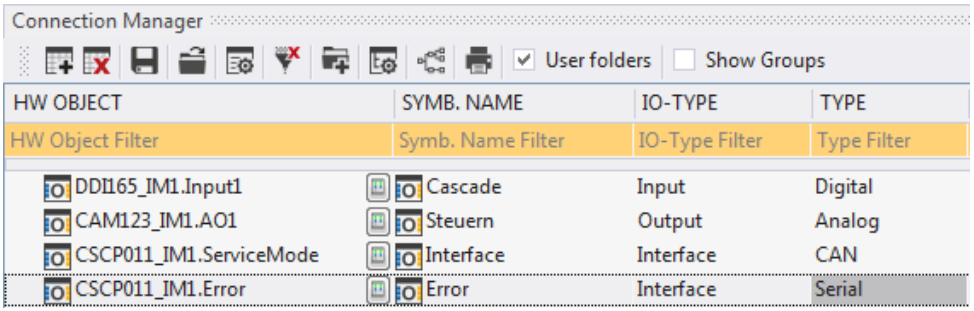
In the lower right of the object, servers and their values are displayed. The latter is shown in green when in online mode

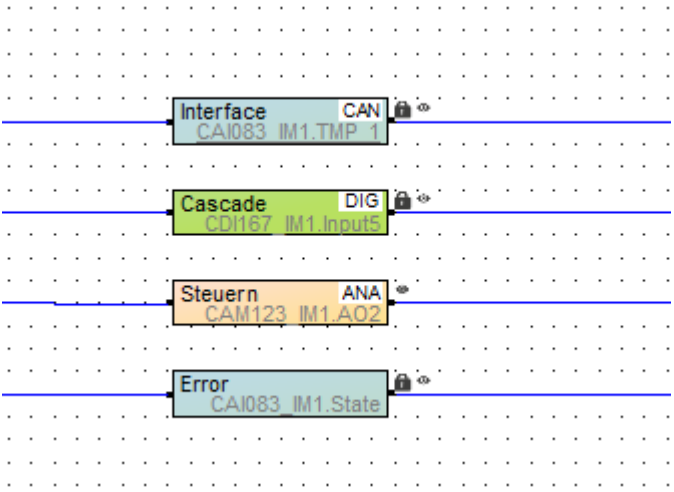


and can be edited in servers that are not write-protected.

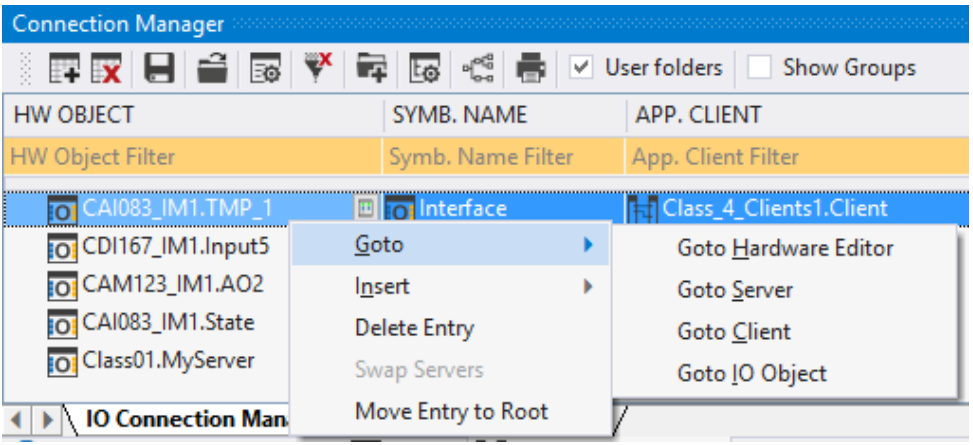


All objects with the IO type Input are green; those with the IO type Output are yellow. All objects with the IO types Interface of IO Stage are blue. The first three letters of the type are displayed in the upper right corner of the IO object. In the following image, the different color and label variations of the IO objects are shown:





With a right click on a connection in the connection manager, the following context menu is opened:



The menu items **Swap Servers** and **Move Entry to Root** are only enabled with specially selected entries (see below).

Goto

Goto Hardware Editor

Program jumps to the server in the hardware editor.

Goto Server

The program jumps to the server in the network view.

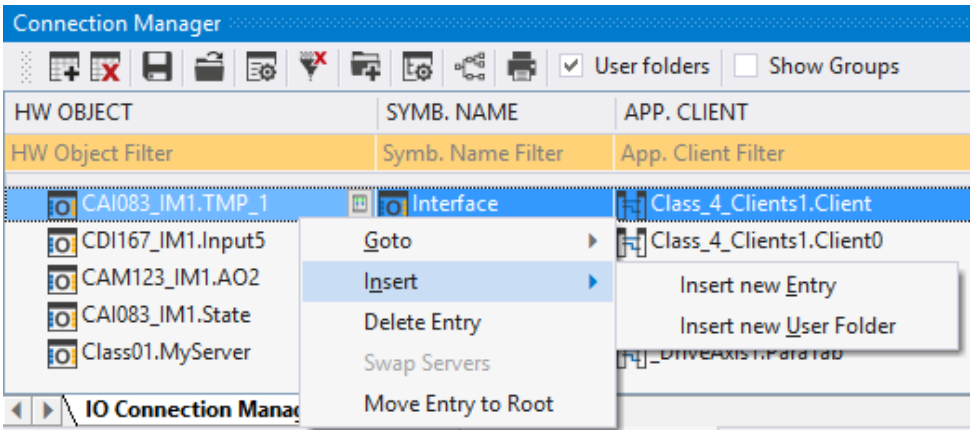
Goto Client

The program jumps to the client in the network view.

Goto IO Object

The program jumps to the IO object (blue, pink or green) in the network.

Insert



Insert new Entry

At that location, a new empty line is inserted (the same as pressing the **Ins** key).

Insert new User Folder

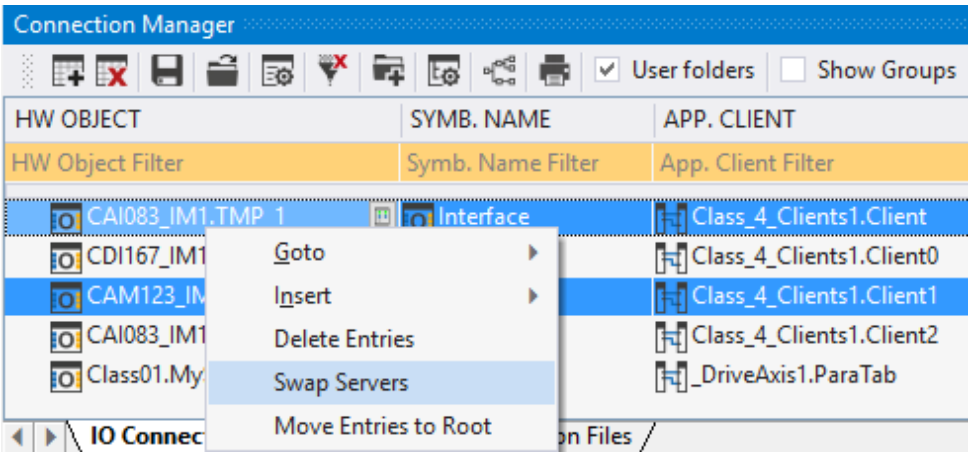
A new user-defined folder is inserted.

Delete Entry

This menu item deletes a selected entry (same as pressing the **Del** key).

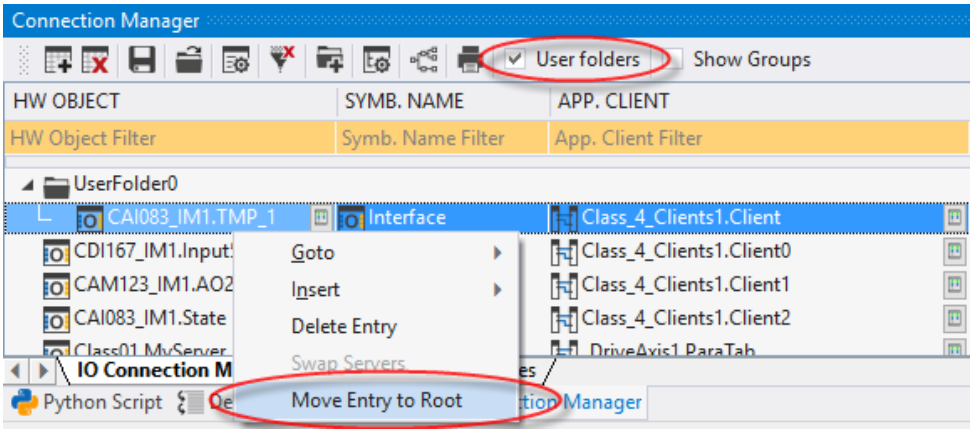
Swap Servers

If exactly two connections in the Connection Manager are selected, the menu item **"Swap Servers"** is enabled and the servers of the two connections can be easily exchanged.:

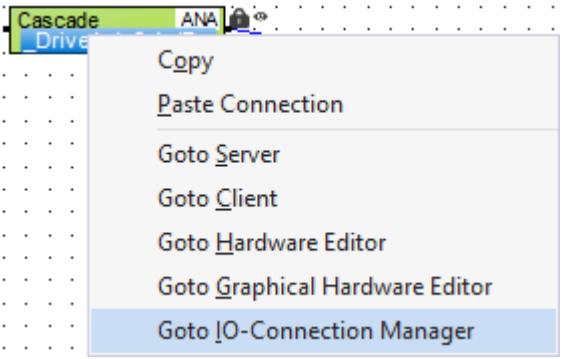


Move Entry to Root

This menu item is only enabled the **User folders** Checkbox is activated and a connection is available in a **user-defined folder**. It moves the connection from the user-defined folder to the root directory. Here, the nesting depth plays no role.

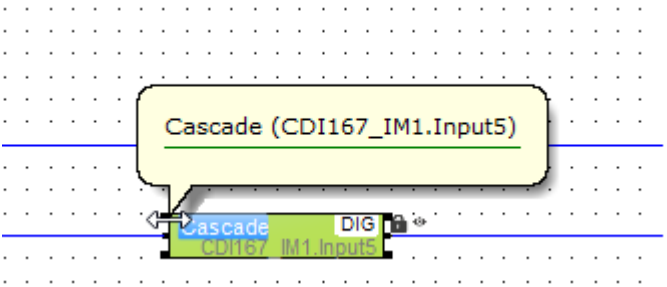


It is also possible to jump to the Connection Manager from the **Network View**, by clicking on an existing **IO object** and selecting the menu item Goto **IO-Connection Manager**.

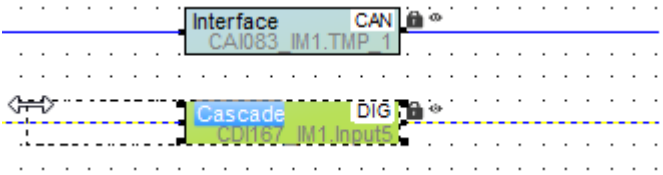


In this menu, the IO object can also be deleted (**Delete**), the size of content automatically adjusted (**Autosize**), the object copied or cut and inserted into another network (**Copy Element**, **Cut Element**). It is also possible to jump to the server (**Goto Server**), Hardware Editor (**Goto Hardware Editor**), Graphical Hardware Editor (**Goto Graphical Hardware Editor**) or Connection Manager (**Goto Connection Manager**).

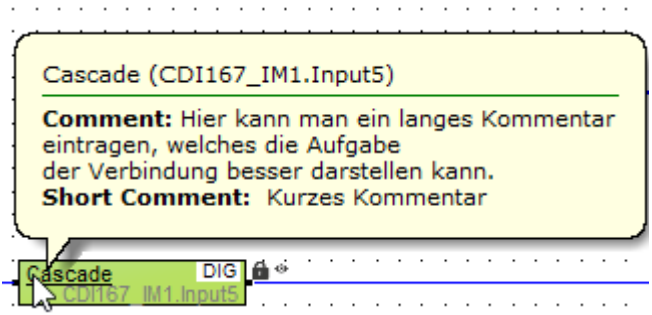
The IO object can – similar to the class object – not only be moved, but the size also changed (Resize). Here, the must be positioned so that the Resize arrow is visible.



...and expanded to the desired size with the left mouse button pressed.



If commentary is entered in the Connection Manager and the mouse pointer is held over the IO object, it is also displayed in a Tooltip:

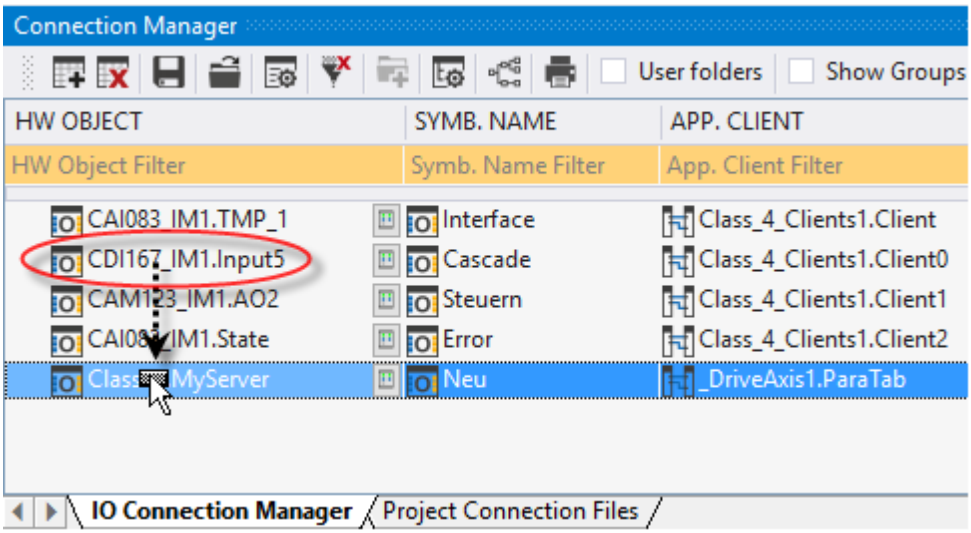


If an object name, server name, client name or the object itself is changed, the change **is immediately assumed in all other relevant views of LASAL**. When, in our example, the server is changed in the connection with the IO name "Cascade", the change is immediately visible in the hardware editor as well as in the network. If the name of a client user-defined class is changed that is part of a connection, the change is made immediately in the Connection Manager, network view, class view, properties window and in the hardware editor.

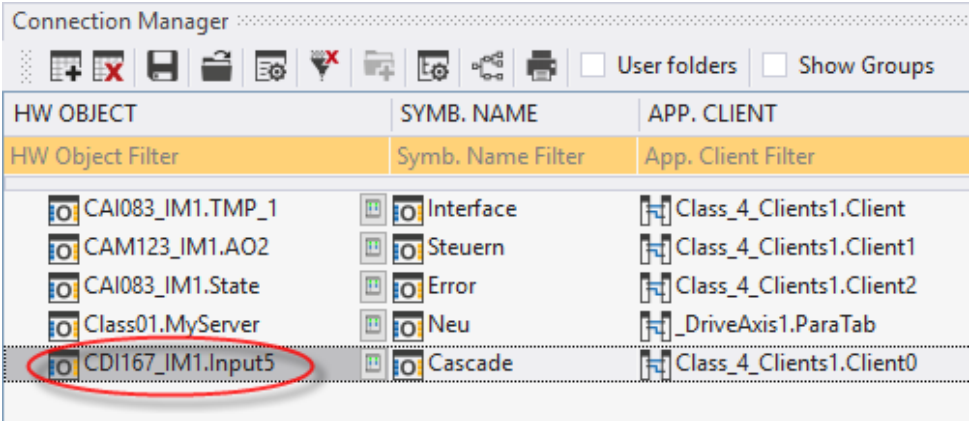
When deleting, the Connection Manager (CM) responds as follows: if the IO object is deleted from the network, all CM parameters are retained. Only the Symb. name icon is crossed out. If the connection is deleted from the network, only the client is removed from the CM. If both (IO object and connection) are deleted from the network, the server, Symb. name – with icon crossed out in red -, IO type, type and commentary remain in the CM. To completely delete a line from the CM, the **Del key** in the CM must be pressed or you select **Delete Entry** from the context menu.

With the button combinations **Ctrl + F**, text (server IO name, type, client, comment...) can be searched at any time.

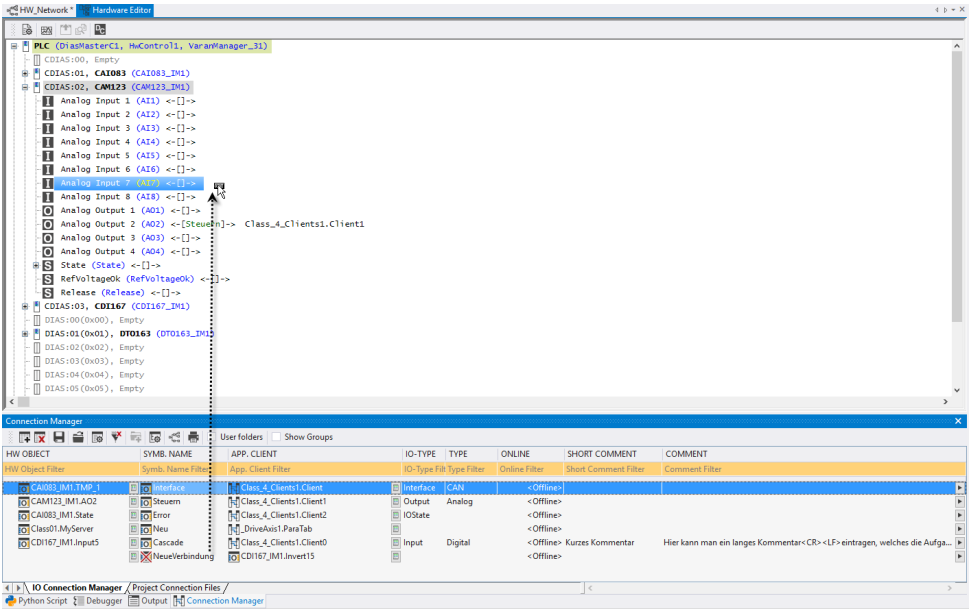
The sequence of the connections can be changed using the Drag & Drop method in the HW as usual. Simply click a connection and hold the mouse button …



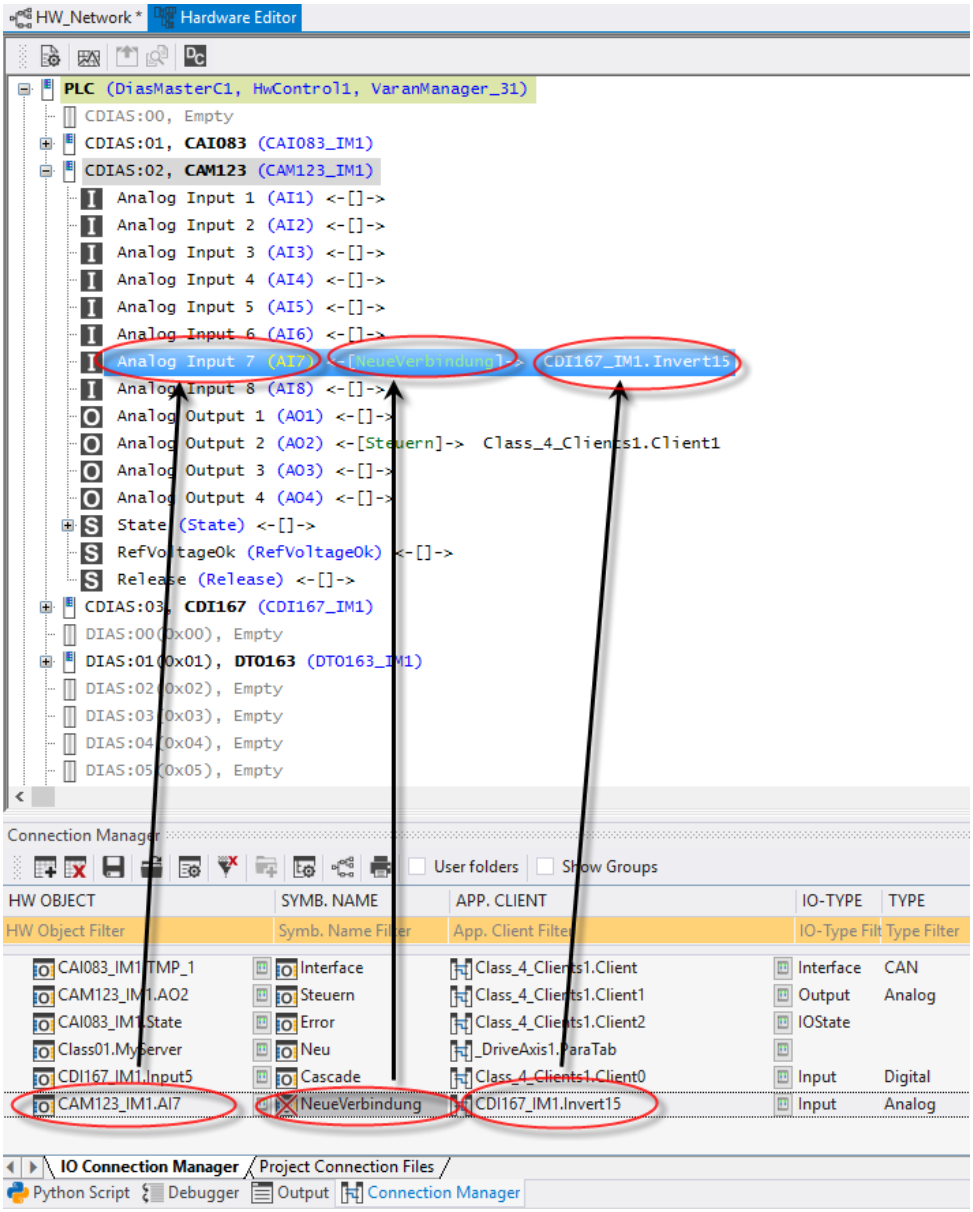
... and the release.



With an opened module and displayed servers, the option to Drag & Drop a connection from the Connection Manager into the HW editor without entering a server.



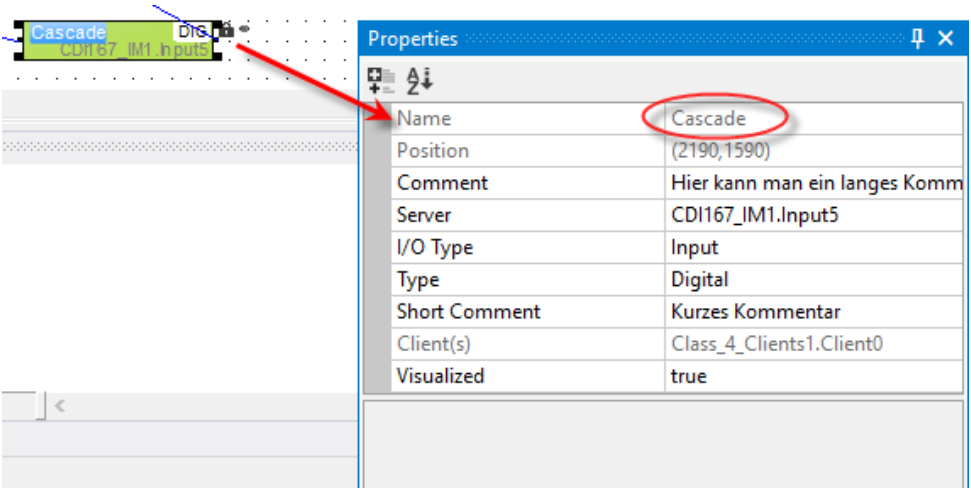
When released, the connection is placed in the HW editor and in the Connection Manager, the selected server is now entered.


HW_Network *
Hardware Editor
PLC (DiasMasterC1, HwControl1, VaranManager_31)
CDIAS:00, Empty
CDIAS:01, CAI083 (CAI083_IM1)
CDIAS:02, CAM123 (CAM123_IM1)
Analog Input 1 (AI1) <-[]->
Analog Input 2 (AI2) <-[]->
Analog Input 3 (AI3) <-[]->
Analog Input 4 (AI4) <-[]->
Analog Input 5 (AI5) <-[]->
Analog Input 6 (AI6) <-[]->
Analog Input 7 (AI7) <-[NeueVerbindung]-> CDI167_IM1.Invert15
Analog Input 8 (AI8) <-[]->
Analog Output 1 (AO1) <-[]->
Analog Output 2 (AO2) <-[Steuern]-> Class_4_Clients1.Client1
Analog Output 3 (AO3) <-[]->
Analog Output 4 (AO4) <-[]->
State (State) <-[]->
RefVoltageOk (RefVoltageOk) <-[]->
Release (Release) <-[]->
CDIAS:03, CDI167 (CDI167_IM1)
DIAS:00(0x00), Empty
DIAS:01(0x01), DTO163 (DTO163_IM1)
DIAS:02(0x02), Empty
DIAS:03(0x03), Empty
DIAS:04(0x04), Empty
DIAS:05(0x05), Empty
Connection Manager
User folders
Show Groups
HW OBJECT
SYMB. NAME
APP. CLIENT
IO-TYPE
TYPE
HW Object Filter
Symb. Name Filter
App. Client Filter
IO-Type Filt
Type Filter
CAI083_IM1.TMP_1
Interface
Class_4_Clients1.Client
Interface
CAN
CAM123_IM1.AO2
Steuern
Class_4_Clients1.Client1
Output
Analog
CAI083_IM1.State
Error
Class_4_Clients1.Client2
IOState
Class01.MyServer
Neu
_DriveAxis1.ParaTab
CDI167_IM1.Input5
Cascade
Class_4_Clients1.Client0
Input
Digital
CAM123_IM1.AI7
NeueVerbindung
CDI167_IM1.Invert15
Input
Analog
IO Connection Manager
Project Connection Files
Python Script
Debugger
Output
Connection Manager

This only functions however, as long as a client and a **symbolic name** are entered in the Connection Manager!

4.2.13.3 Properties of an IO object

If an **IO object** is clicked in the network view, the following parameters show in the properties window:

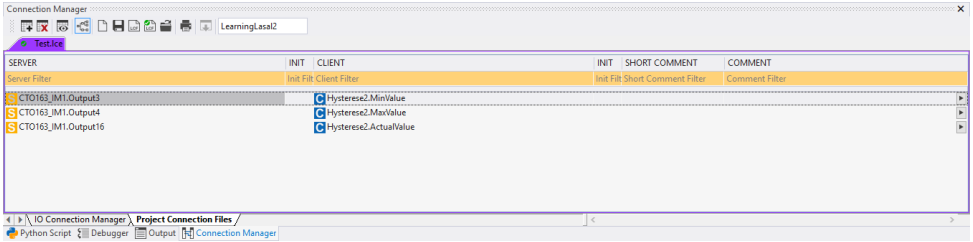


The fields **Name**, **Position** and **Client(s)** only show information.

The **server connection** can be edited here (like in the Connection Manager in the connections), also the **I/O Type** and the two **Comments (short and long)**. The property **Visualized** can only be entered here and defines, if this **IO object** should be visible in the LASAL SCREEN Editor (**true**) or not (**false**).

4.2.13.4 The Project Connection Files Window

Via the **Project Connection Files** tab, a view that looks very similar to the IO Connection Manager opens, but covers an entirely different range of functions.

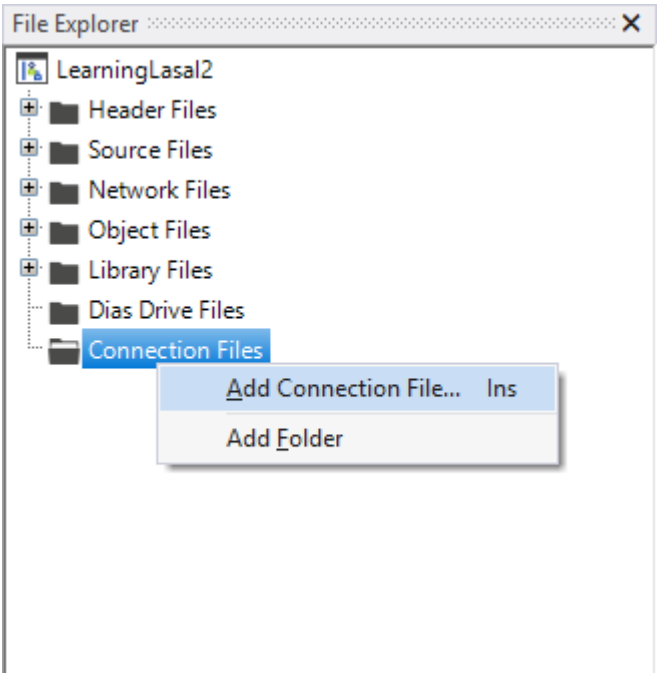


Contrary to the Connection Manager, the Connection File Editor provides the option to add connections without having to change or compile the project. The connections assigned here can be stored in a connection file, which can then be loaded in the PLC.

Generally, this connection file is stored with the project (the connection files available in the project are listed in the File tree under "Connection Files") but they can also be saved individually using the Save button in the Connection File Editor toolbar. By creating several connection files for a project, the user has the option to manage several configurations or expansion stages with a base project.

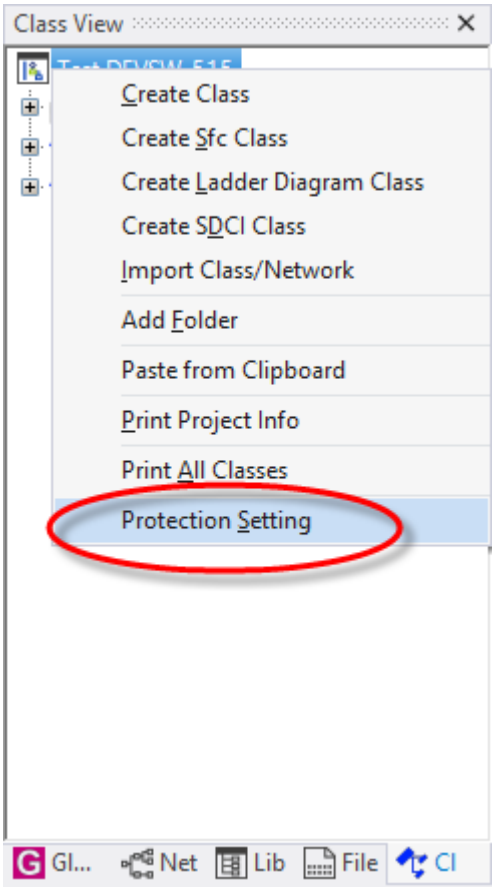
All connections in the base configuration are thereby stored in the project and the desired additional configuration is added later through a connection file. This provides the advantage of not having to manage an individual project version for each expansion stage. Instead, they can be comfortably configured through connection files.

In order to work with the **Project Connection Files** view, at least one connection file must already be available and added to the **Connection Files** folder in the **file explorer**. For this purpose, either press the icon "Create new File"  in the task bar of the Project Connection Files tab; or right-click on the "**Connection Files**" folder in the File Explorer and select "**Add Connection File...**".

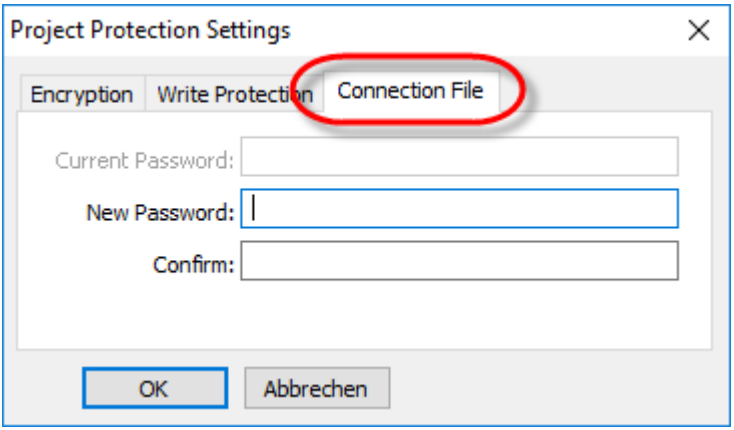


In both cases, a file dialog opens, in which a file with the ending *.lce (= LASAL connection file) should be selected.

At this point, the difference between ***.lce and *.lcf files** must be noted: a connection file (Project Connection File) is stored **on the Windows PC in *.lce** format. This always affects the actual *.lce file, which is currently displayed and can be stored using the  button. An *.lce is opened using the  button If "**Download active connection files**"  is pressed, an ***.lcf file** is generated and loaded **directly into the control**. This can also be done in encrypted form by right-clicking on the project in the tree and selecting "Protection Settings":



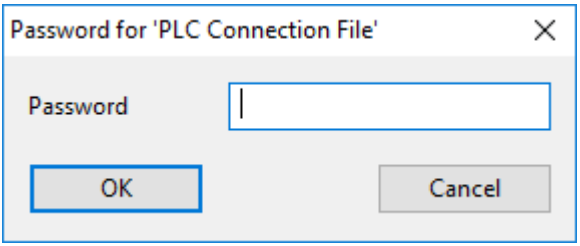
Select the third tab **Connection File** here:



After entering and confirming a password the connection file is now **encrypted**. The file is also **downloaded to the control in encrypted form** - here you need **at least Loader version 02.02.201**, so that the operating system can handle encrypted *.lcf files. If it is an **RTK operating system**, the version should be **at least 01.03.130**. For the **Salamander OS**, the minimum required version depends on the kernel: the Salamander version with **kernel version 02** must be **at least 09.02.050;** with **kernel version 03 at least 09.03.120**.

If a LASAL CLASS 2 version, which already contains this feature, and a Loader / OS, in which these changes are not yet implemented, is used, the project **cannot be loaded** on the control, the **CPU does not start up**!

If you go **online on a control** with an **encrypted connection file while the project is running**, you will be asked for this password immediately after going online.

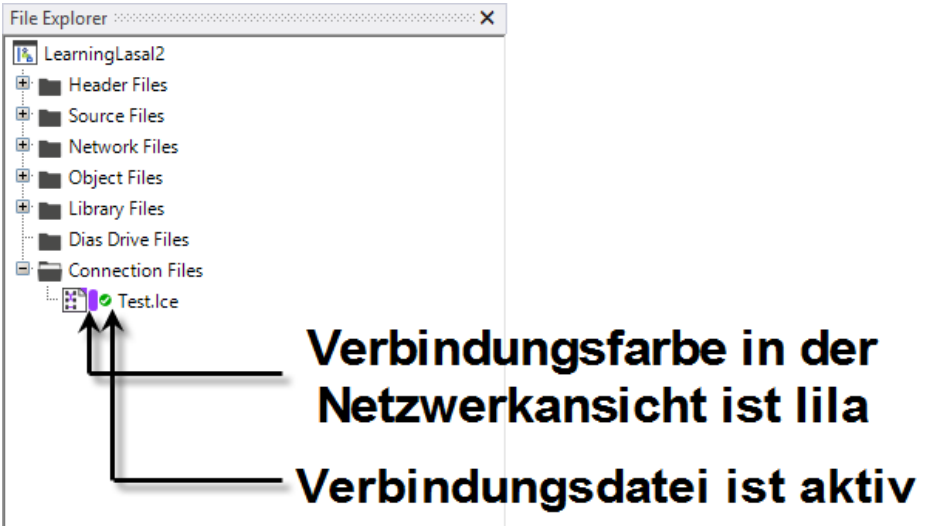


There is **always only one *.lcf file on the control**! It will be overwritten every time you download it. It is located on the control under **C:\lsldata** and is called **prjcfg.lcf**.

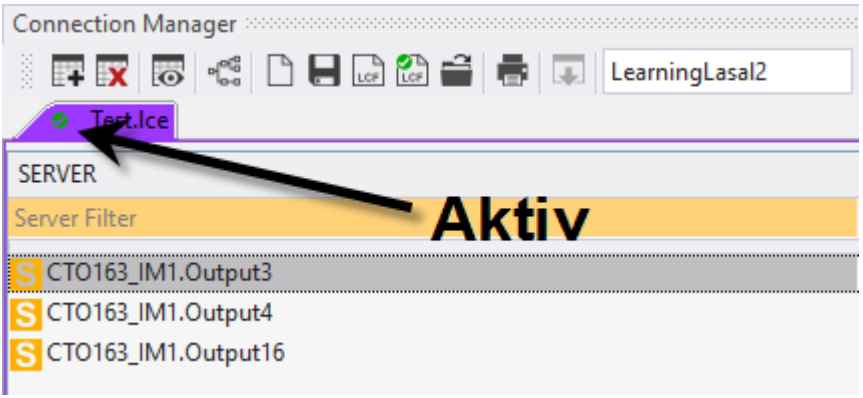
The option is also available to generate and store ***.lcf files on the Windows PC**. These can then – via the File Commander for example – be stored in a folder in the control. Using the **ConfigFileManager** a user-defined *.lcf file can now be loaded and while starting the project, the default.*lcf file overwritten.

In the task bar, the two LCF icons can be used to select whether this ***.lcf file** should only be generated from the **currently displayed** , or **all active** **Connection Files**.

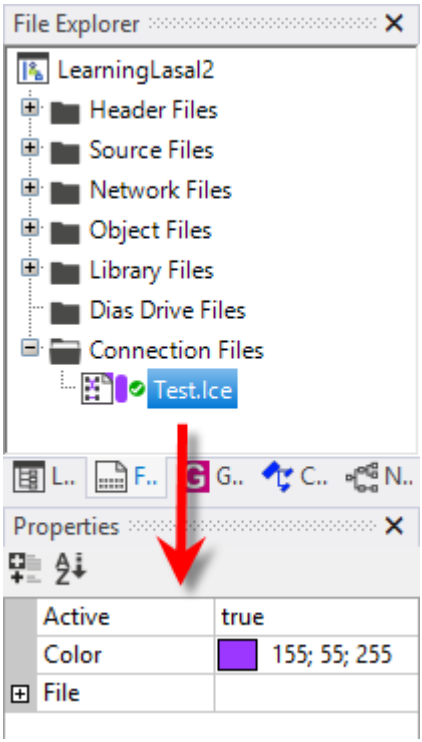
Now, the newly added file appears in the **Connection Files** folder.



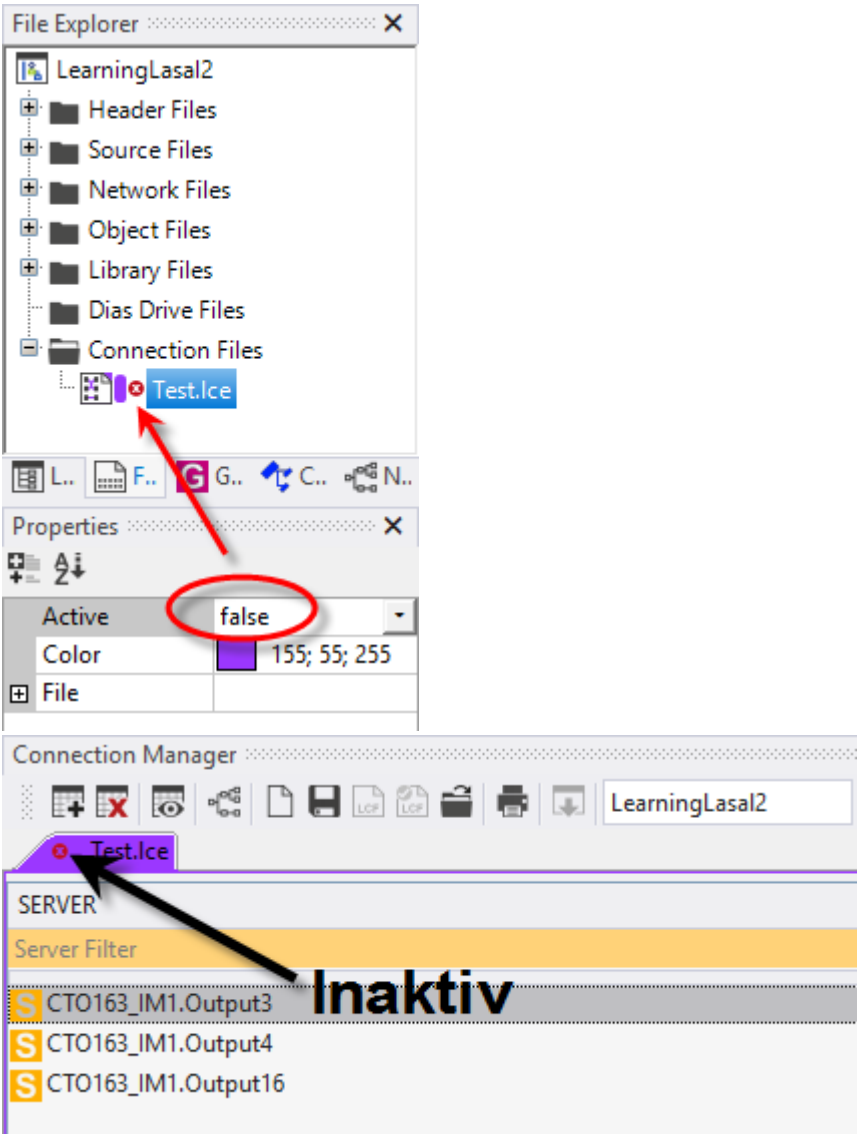
It is enabled by default (= marked in green) and has a color strip (default: purple) in the icon that indicates the color in which the connections in this file are displayed in the network view. The same marking (green = enabled, red = disabled) is found in the register tab display of the **Project Connection Files**.



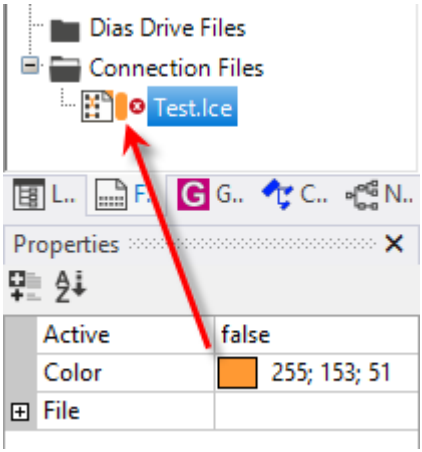
To disable the connection file or change the color of the connection in the network view, simply left-click on the file in the tree to open the properties in the properties window.



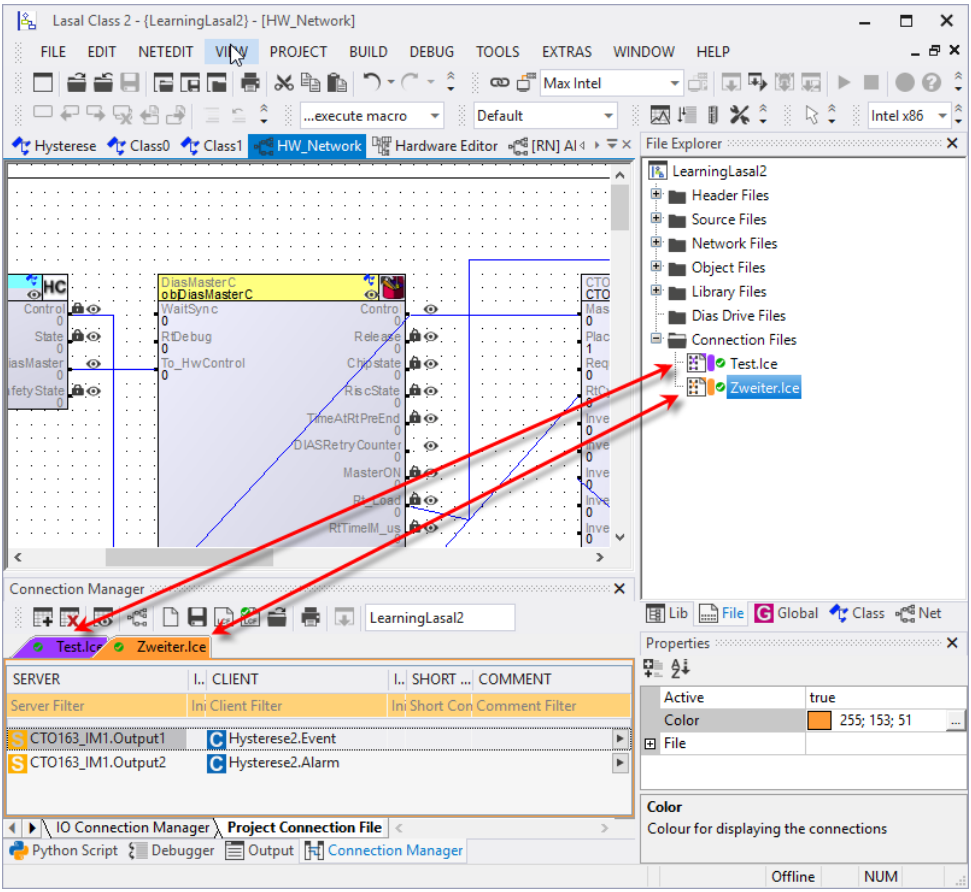
If the Active property is changed to **false**, the display in the icon and the register tab changes immediately.



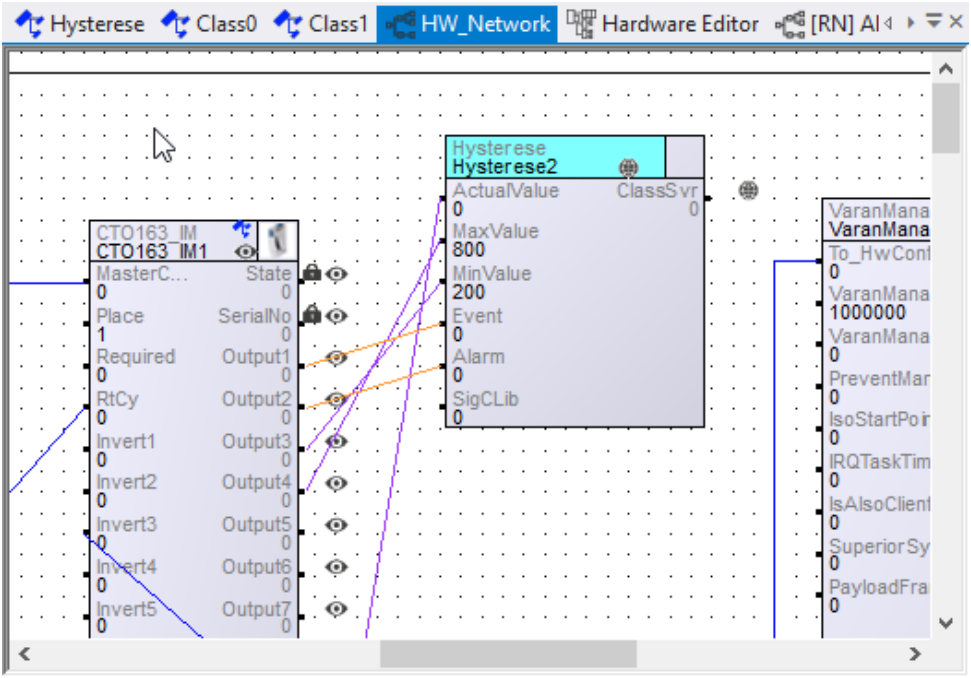
If the color is changed, the color strip in the icon ...



... the color of the register tab (for multiple Project Connection Files, different colors can be set for better recognition):

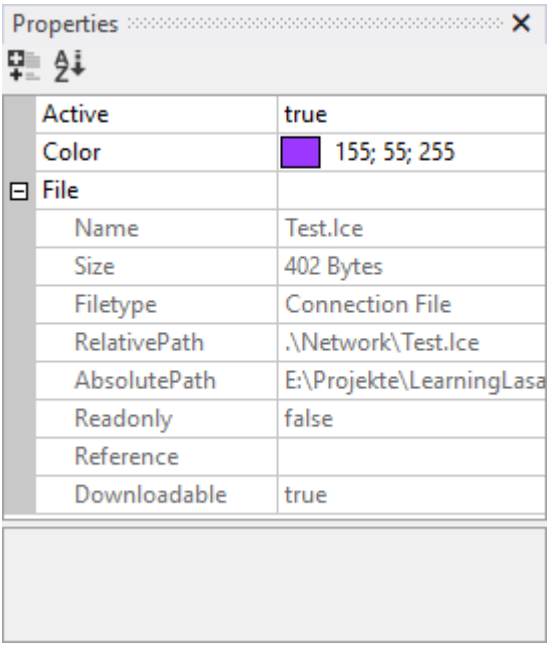


... and the color of the connections in the network view are changed.



The overview with multiple active connection files – especially in the network view – is not lost.

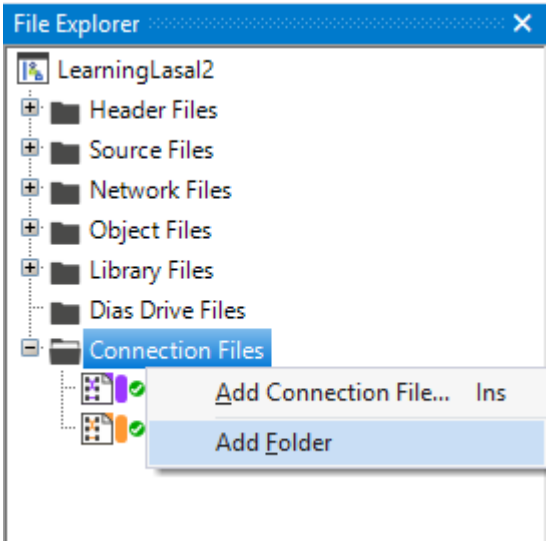
If in the properties, the plus symbol in front of **File** is clicked, further information on the connection file is displayed (path, file size etc.).



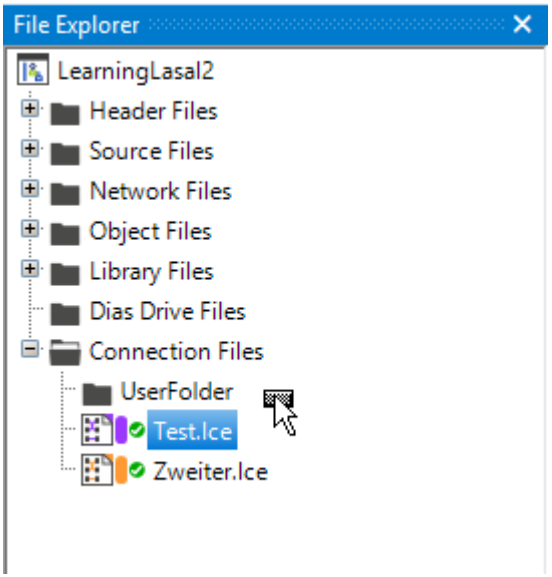
This is a **display-only function**, no properties can be edited under file.

The connection files can – similar to those in the class tree – can be managed in the **User Folders**. Individual files, as well as the **User Folders** can – as previously discussed – can be **enabled** or **disabled**. This results in various management options: **entire groups can be enabled/disabled**, for which only a mouse click and restart of the project in the control is required.

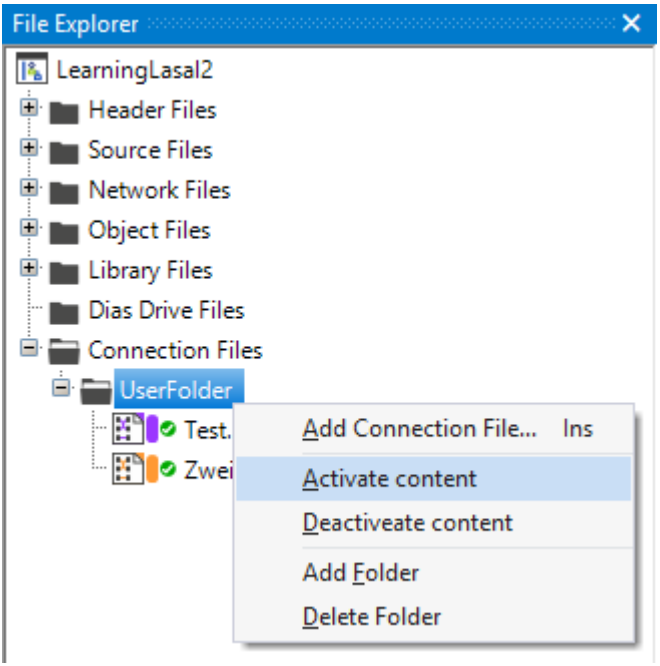
To add a **User Folder**, the context menu **of the Connection File** folder is called with a right-click in the **file explorer**.



Connection files can be moved to the newly added folder ...



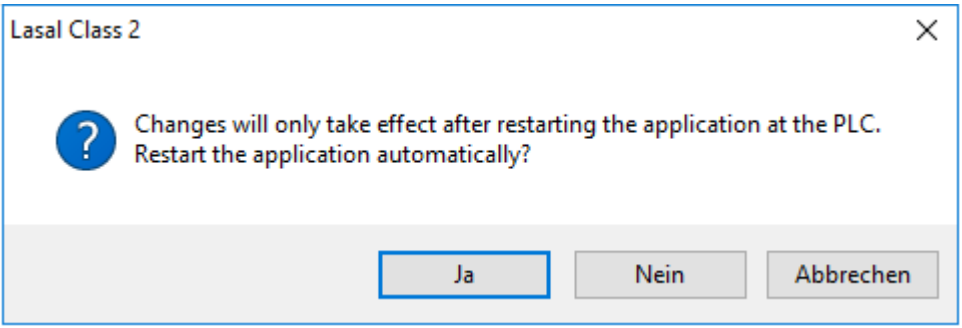
... and the entire folder content in context menu **enabled or disabled with only one click**:



Adding a connection file under Connection Files, as well as **enabling/disabling** or **deleting** an existing file is also possible with the corresponding commands **via scripting** (see chapter).

There are two options to load a connection file into the PLC:

- **Download Button**: If in the Connection File Editor, the Download button  is clicked and the checkbox "Add active project connection files" is checked in the download dialog box that appears, the connection file is loaded into the control



after confirmation in the dialog box that appears. If Yes is selected here, the active project in the control is stopped and immediately restarted so that the loaded connection file is enabled.

- **Debug Menu:** Previously stored connection files can be loaded to the PLC through Debug Menu => File Transfer => Download Connection File.

A connection function is assigned in the same way as in the Connection Manager, except for the missing preselect and the additional Init columns. Alternatively, the a legal server / client can be dragged from the network view analog to the Connection Manager. Here, an IO object can also be entered manually as a server or client. Or a connection with an IO object can be made from the Connection Manager via Drag and Drop in the Project Connection Files window.

The Init columns is used to set the sever /client inputs during start-up of the loader. When starting the application in the PLC, the data contained in the connection file is analyzed by the loader, which then creates the required connections. If a client is connected in the project as well as in the connection file, the loader **replaces** the connection in the project with the one in the connection file.

To use the connection file, the project name in the specified file must match the name of the application in the PLC and it must be possible to assign all connections (client and server side available).

If the project name does not match, the connection file is ignored (message in the Debugger Trace window). If no connections can be assigned, the application cannot be started and the CPU goes into the Loader Syntax Error status (message in Debugger Trace window).

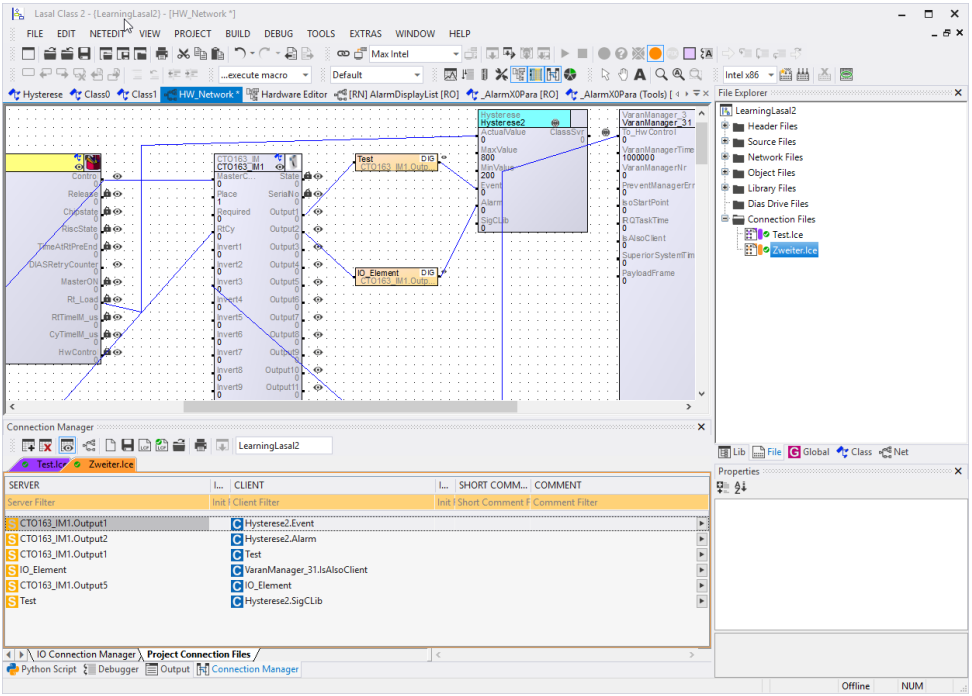
In the online mode (RUN RAM only) or by clicking the **Show connections in network** button  , the additional connections are displayed in object network with the color

defined in the "Color" property of the connection file. Project connections that were overwritten by the connection file are not shown in the online mode (RUN RAM only).

In the status line, the project filter field can be found next to the download button.

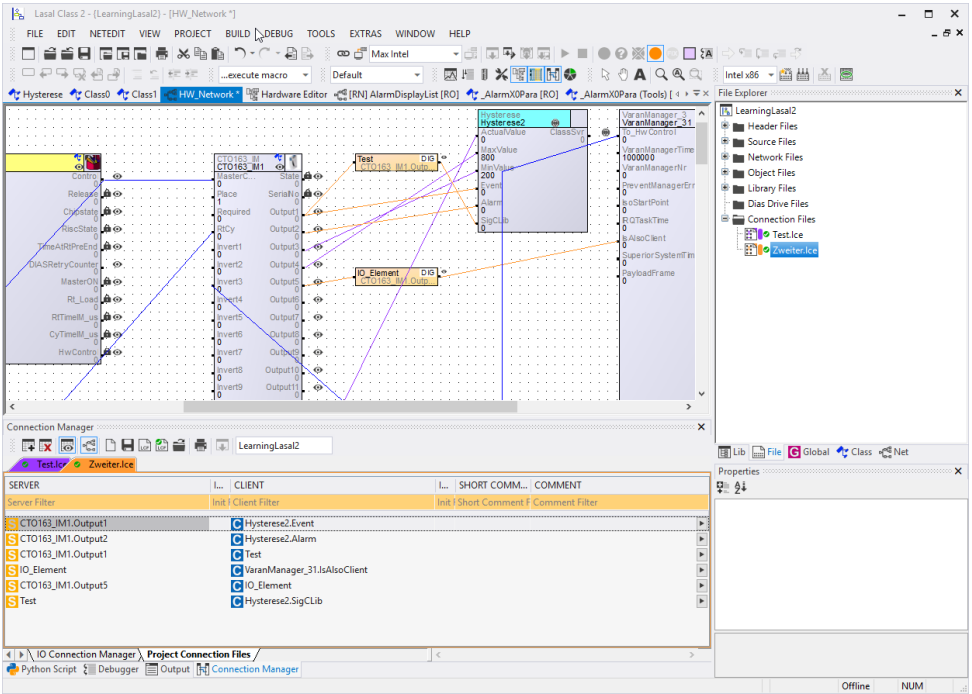


In this field, the name of the current project is entered by default, however, different project names can also be filtered here. Wild cards are thereby allowed: the star character (*) stands for the number of number of characters; the question mark (?) for a random character; the hash character (#) for a random number. E.g. you could filter with "Lear*" to also find the current project name (LearningLasal2).

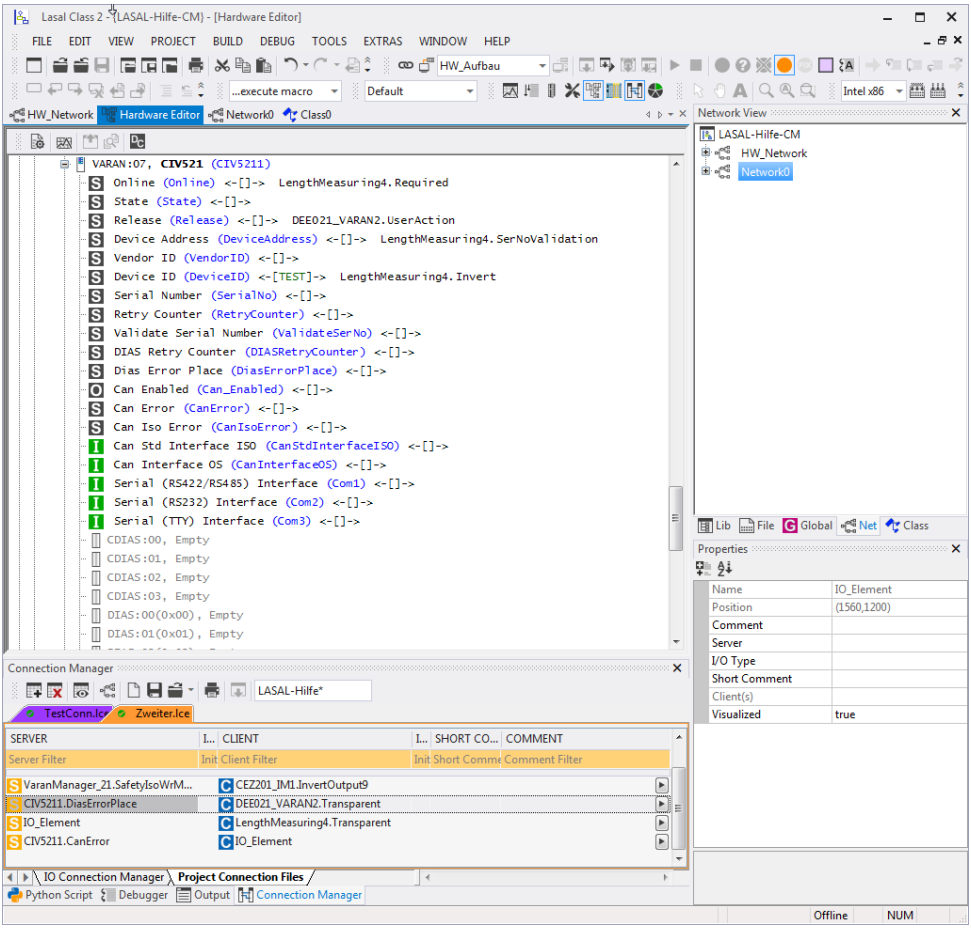


In the image above, the project is not yet running and the Show Connections in network button was also not pressed. The connection entered in the **Project Connection Files** window are not yet visible in the object network, only those available in the **IO Connection Manager** (in the project).

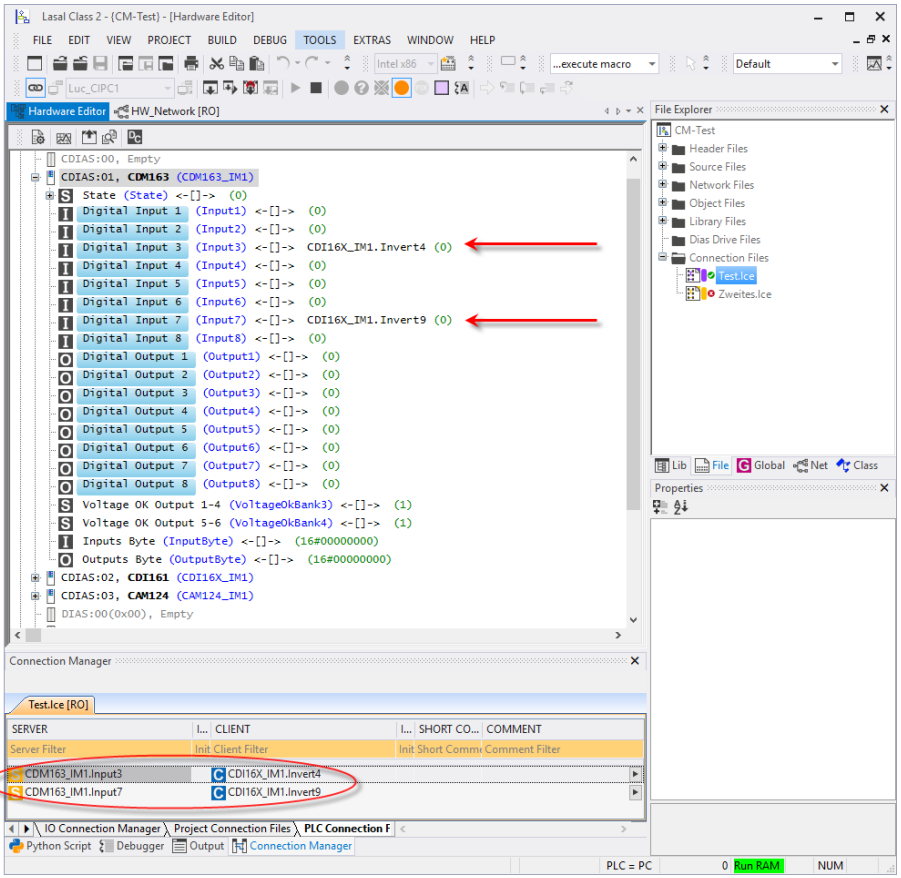
In the following image the Show Connections in network button is pressed, the temporary connection entered in the **Project Connection Files** window are visible in the object network (in purple or yellow). Alternatively, online mode would also be possible for this purpose.



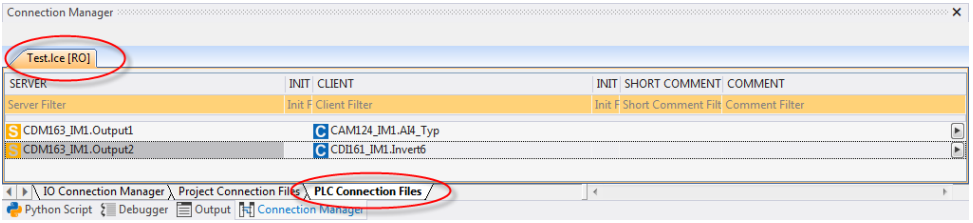
The can be observed in the HW tree: in the following image, the temporary connections are not visible in the HW tree because the project was not started or the Project Connection File was not activated.



When the project is started, the Project Connection File activated or the **Show connections in network** button is pushed in offline mode, the temporary connections are also entered into the HW tree:



If – as in the previous image – the control is in Run Ram mode and a Connection File is already loaded in the PLC, an additional tab appears: **“PLC Connection Files“**. When clicked, which connection file currently in the control is valid. (In our example: Test.lce)



A comparison with the **Project Connection Files window** can be easily made.



If **multiple initializations** or **Clients with multiple connections** are created during download, the compiler sends an **error message** and the **Download is canceled**!

4.2.13.5 Advanced IO functionality

- Overview
- Functionality
- Simulation modes
- Switching on the Advanced IO functionality
- Description of the additional columns
  - Advanced
  - Force Enable
  - Force Value
    - Special features of the "Forced mode"
  - Sim(ulation) Enable
  - Sim(ulation) Counter
  - Sim(ulation) Value
  - Hw Value
- ImportantNotes

Overview



Note

From the **release Q1/2023** (LASAL CLASS 2 **version 02.02.225**) and onwards, it is possible to create **Advanced IO objects** and thus also to use **simulation**. In order to use the **full functionality** of Advanced IO, the **loader version 02.02.244** or newer must be used!

The Advanced IO functionality makes it possible to work **with simulation environments** in LASAL CLASS 2. This makes it possible to test parts of a hardware setup or entire

machines **in advance, i.e. before the HW setup**. This saves development costs and time, since possible malfunctions can already be detected and corrected at this time.

The Advanced IO functionality is implemented in LASAL CLASS 2 **in the project or application properties** as well as **in the object network** and **in the Connection Manager**.

The realization in LASAL CLASS 2 is designed in such a way that in principle any simulation can be seamlessly integrated, but currently **only the export of iPhysics** (Machineering) **is supported**.

Functionality

To enable a simulation, **IOs with the same name must be created in Lasal CLASS 2 and iPhysics**. It is important that **in Lasal CLASS 2 all IOs** must be available which iPhysics should use, but **in iPhysics not all IOs** of the Lasal CLASS 2 project must be available.

A **codegen file can then be created** based on the simulation created in iPhysics. This codegen file is required by the iPhysics simulation class as a **configuration file** and **must be downloaded to the controller**. However, this happens automatically when the file is selected in the **Virtual Commissioning Manager** -Properties of an iPhysics commissioning.

Several simulations can run in parallel in LASAL CLASS 2 - a simulation object is created by LASAL CLASS 2 for each of these simulations. The simulation objects are managed **in the Connection Manager** on the new tab "**Virtual Commissioning Manager**".

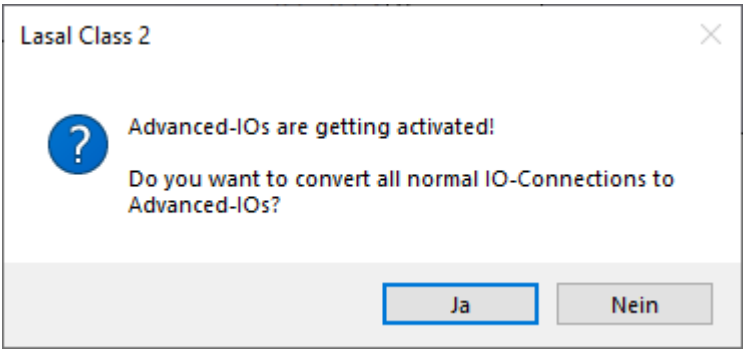
Simulation modes

There are three simulation modes:

1. **No Simulation** – input and output variables are **passed to the application / to the hardware without any interaction**.
2. **Simulation** – the **hardware exists only virtually**, the output values of the application are **passed to the simulation program** where they are processed. The input values in LASAL CLASS 2 then come from the simulation program, and are passed on to the application.
3. **Digital Twin** – the values **are passed on to the machine (hardware) as well as to the simulation program**, which runs in parallel.

Switching on the Advanced IO functionality

In the project properties, the functionality can be enabled for the project with the property "**Use Advanced IO**". A dialog immediately asks whether the LASAL CLASS 2 project should be converted to the new format for the use of Advanced IOs:

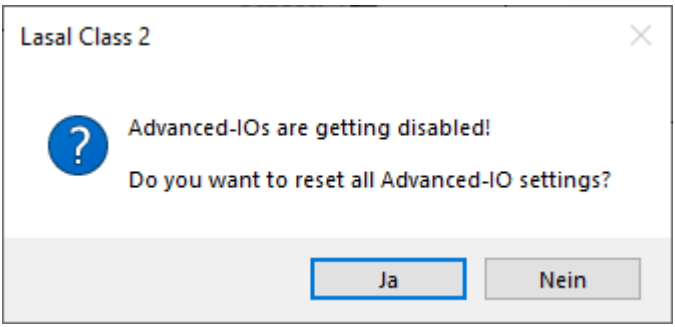


If the question is answered with yes, **all IO objects are converted to Advanced-IOs**, which can also be seen directly in the object network (the IOs are displayed larger and have additional servers). The checkboxes of all entries in the **Advanced** column are set.

Attentiong!

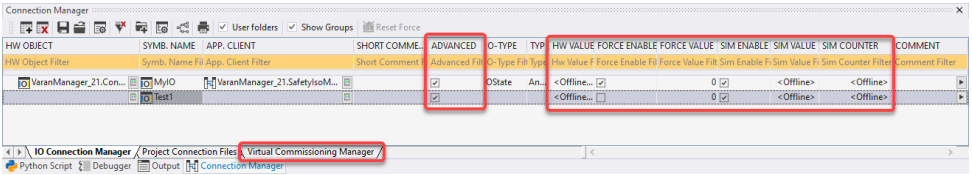
If the columns **Advanced, Force Enable, Force Value, Sim Enable, Sim Value, Sim Counter** or **Hw Value** are not visible, they must be made visible in **Column settings** !

This conversion **can be undone at any time** by setting the property "**Use Advanced-IO**" to **false**. Here, too, a query is made:



Thereupon all IO objects are made "**simple**" again, i.e. the **advanced** checkboxes are all deselected.

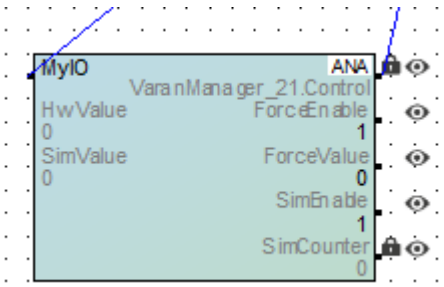
If **Use Advanced-IO = true**, the **Advanced** column in the Connection Manager is made **ready for input**, and if checked, the **two Force columns are also enabled**. Depending on the column visibility setting, the columns for the simulation (**SIM ENABLE, SIM COUNTER, SIM VALUE**) and **HW VALUE** are also made visible. The Virtual Commissioning Manager **tab is also displayed**:



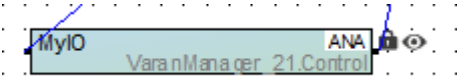
Description of the additional columns:

Advanced

This checkbox can only be entered if the property **Use Advance-IO** has been set to true in the project settings. By selecting the checkbox, this IO object becomes an **Advanced IO object** with additional servers: (**Force Enable, Force Value, SimEnable** and **SimCounter)** as well as the conditional informal values **HwValue** and **SimValue**.



Conversely, the checkbox can be removed at any time and then it becomes a simple IO object; thus losing the servers **Force Enable, Force Value, SimEnable** and **SimCounter** as well as the conditional informal values **HwValue** and **SimValue:**



Force Enable

This checkbox can **only be entered in online mode** and only if the **Advanced** checkbox is selected. It switches on the **Force mode**, i.e. the value specified in the **Force Value** column is passed directly to the controller / simulation program.

Force Value

This field can **only be entered in online mode**, and only if the **Advanced** and **Force Enable** checkboxes are selected. The value entered here is passed directly to the controller / simulation program.



Attention

If **Advanced IOs** are used **without hardware**, the **IO type must be set manually**, otherwise the Advanced IO objects cannot function properly!



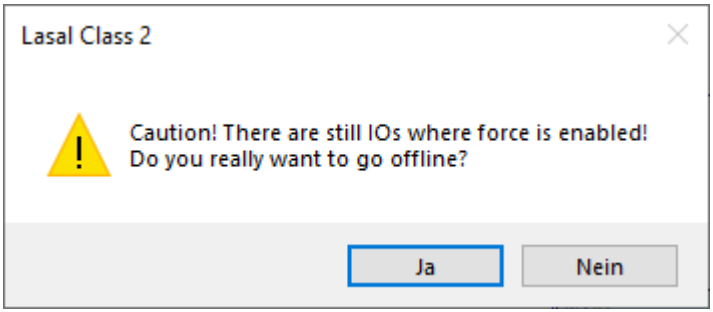
Hint

If the IO-Type column is not visible, it must be enabled in the Column Settings (accessible via the corresponding toolbar button in the Connection Manager).

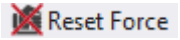
Special features of the "Forced mode"

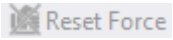
By setting the **Force Enable check mark**, the "**Forced Mode**" is activated, i.e. the current value on the hardware / in the simulation program **is overwritten by the value in the Force Value column**. This mode allows the user **to intervene manually at any time**. Due to their function, the two Forced columns are **only ready for input in online mode**. They **correspond to** the two identically named **servers in the Advanced IO object**.

If **Force Enable** has been set in Advanced IO objects in the object network and you go offline, you will be made aware of this by means of a warning message:



If you don't want this message to appear every time you go offline, turn it off under Extras / Options with the "**Show Force Enable Warning Dialog**" item.

With the button in the toolbar of the IO Connection Manager 

(enabled) 

(disabled) all Force Enable checkmarks can be deleted (reset) at once. Thus the **forcing is completely switched off**. Attention: **the values in the Force Value column remain unchanged!** The **Reset Force** button, like the two Force columns, **can only be used in online mode**.

The "**Forced Mode**" is **independent of the simulation mode**, so **it can be used in all three kinds of simulation**.

Sim(ulation) Enable

This checkbox **only has an effect if a simulation** (Virtual Commissioning) **is running**. Here you can set **for each individual IO object whether the simulation is to be operated or not**. The choice made here **corresponds to the server SimEnable** for an AdvancedIO object.

Sim(ulation) Counter

This field **only shows something when a simulation** (Virtual Commissioning) **is running**. A counter is incremented here **every time a value comes from the simulation**. The value shown here **corresponds to the server SimCounter** for an AdvancedIO object.

There are also two informal values, which appear in the place where clients are otherwise displayed:

Sim(ulation) Value

This field **only shows something when a simulation** (Virtual Commissioning) **is running**. The displayed **value comes from the simulation** and **corresponds to the informal value SimValue** at an AdvancedIO object.

Hw Value

This field **shows something only in online mode**. The displayed **value comes from the hardware**, and **corresponds to the informal value HwValue** at an AdvancedIO object.

Important notes



Attention

If a **dot occurs in the name of an Advanced IO object**, it is **replaced by an underscore** in the LASAL SCREEN EDITOR, and should also be used there in this way. Thus, "MyIOObject.Ver1" becomes --> "MyIOObject_Ver1".



Attention

Currently **only the connection type Data Channel is supported** for Advanced IO objects. Connections of the type Object Channel or Command Channel **cannot be used**! This fact is also pointed out by means of an error message:

OutputTableGen: [ERROR] Channel 'Net4.ComplexConsumer.Client' is not a Data Channel therefore it can't be connected to an Advanced IO! Please use a regular IO instead.

IO Connection Manager Areas
Properties of an IO Object
The Project Connection Files Window
Project Connection Files
The Virtual Commissioning Manager

IO Connection Manager

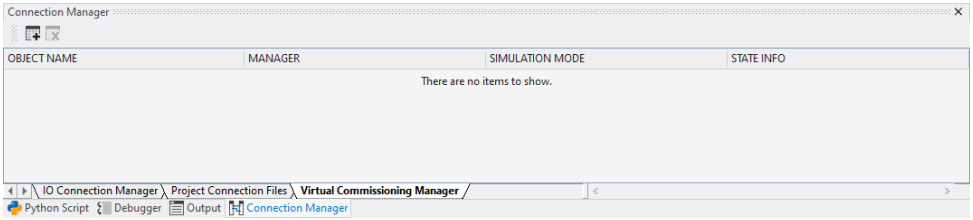
4.2.13.6 The Virtual Commissioning Manager

Attention

The Virtual Commissioning tab is only available if the Use Advanced-IO **Use Advanced-IO** property in the project properties is set to **true**!

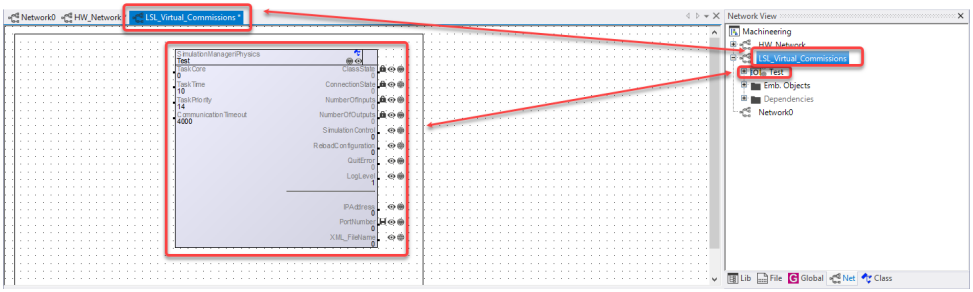
General information

Several different simulations can run simultaneously in the Connection Manager. These are managed in the **Virtual Commissioning Manager** tab by creating entries (simulation objects) here, **each with its own class and simulation mode**, and assigning them a name. The object created in the background can be addressed via this name.



Thus, the Virtual Commissioning Manager **is responsible for managing the simulation classes and objects**, which are responsible for the connection between the Advanced IOs and the simulation software. Currently SIGMATEK supports the simulation software **iPhysics** from the company **Macheneering**.

When creating the first simulation object, LASAL CLASS 2 creates **its own network** for this object: **LSL_Visual_Commissions**. Each simulation object is automatically created in this object network:



Attention

The name of the Visual Commissioning network **should not be renamed**, because LASAL CLASS 2 will try to regenerate the Virtual Commissioning network with the given name for each new simulation object!

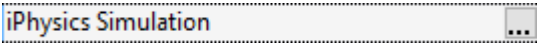
In the toolbar old familiar items can be seen again: with the  -icon a new virtual commissioning can be **created**, with the  -icon a selected commissioning can be **deleted**. There are three columns:

Objectname

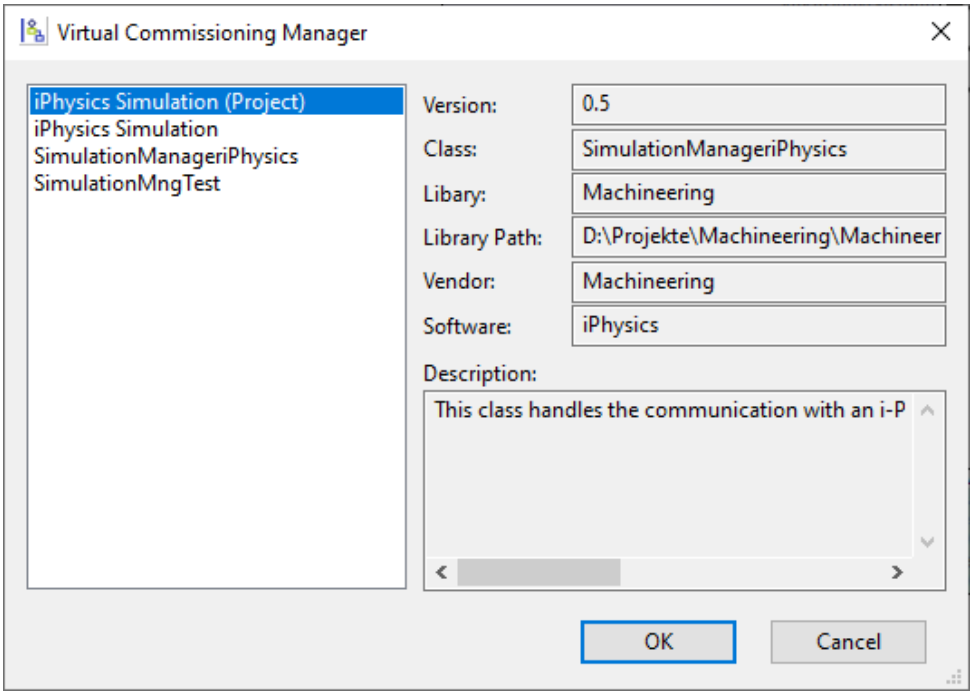
Hereby a **freely selectable name** is assigned, which however must be unique in the entire Connection Manager. With this name an object is created in the background.

Manager

In this field **the simulation manager or the class can be selected**. The available **properties** of the entry **depend on this selection**. If a new entry is created, **iPhysics Simulation** is entered in this field by default. The fact that it is possible to choose between several different classes becomes visible by the More button with the three dots:

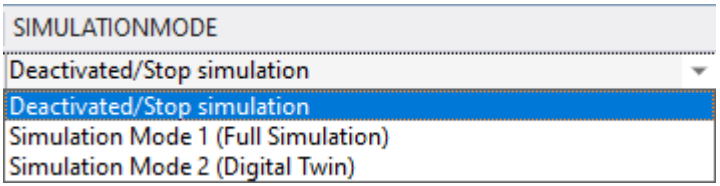


If you click on this More button, a selection dialog opens, in which you can choose between the classes shown on the left side. On the right side you can see the properties of the selected class:



Simulationmode

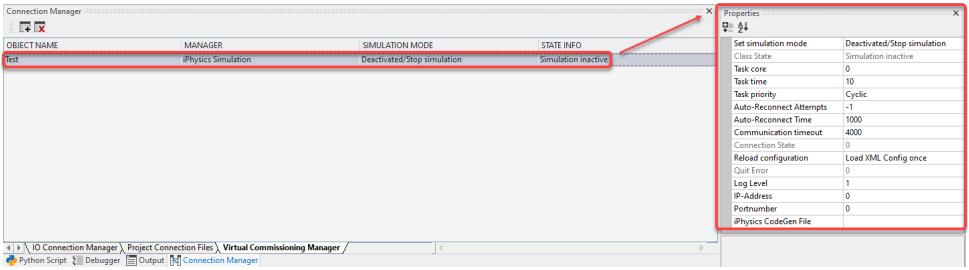
In this field **the type of simulation can also only be selected**: (The entries in the drop-down list are each dependent on the class selected in the previous column).



State info

The **status of the currently running simulation** is displayed. This is a **pure display field** and **cannot be edited**.

If a virtual commissioning is selected, **special properties** are displayed in the properties window. **The final configuration of the simulation takes place here:**



The displayed properties are related to the selected simulation manager / class.

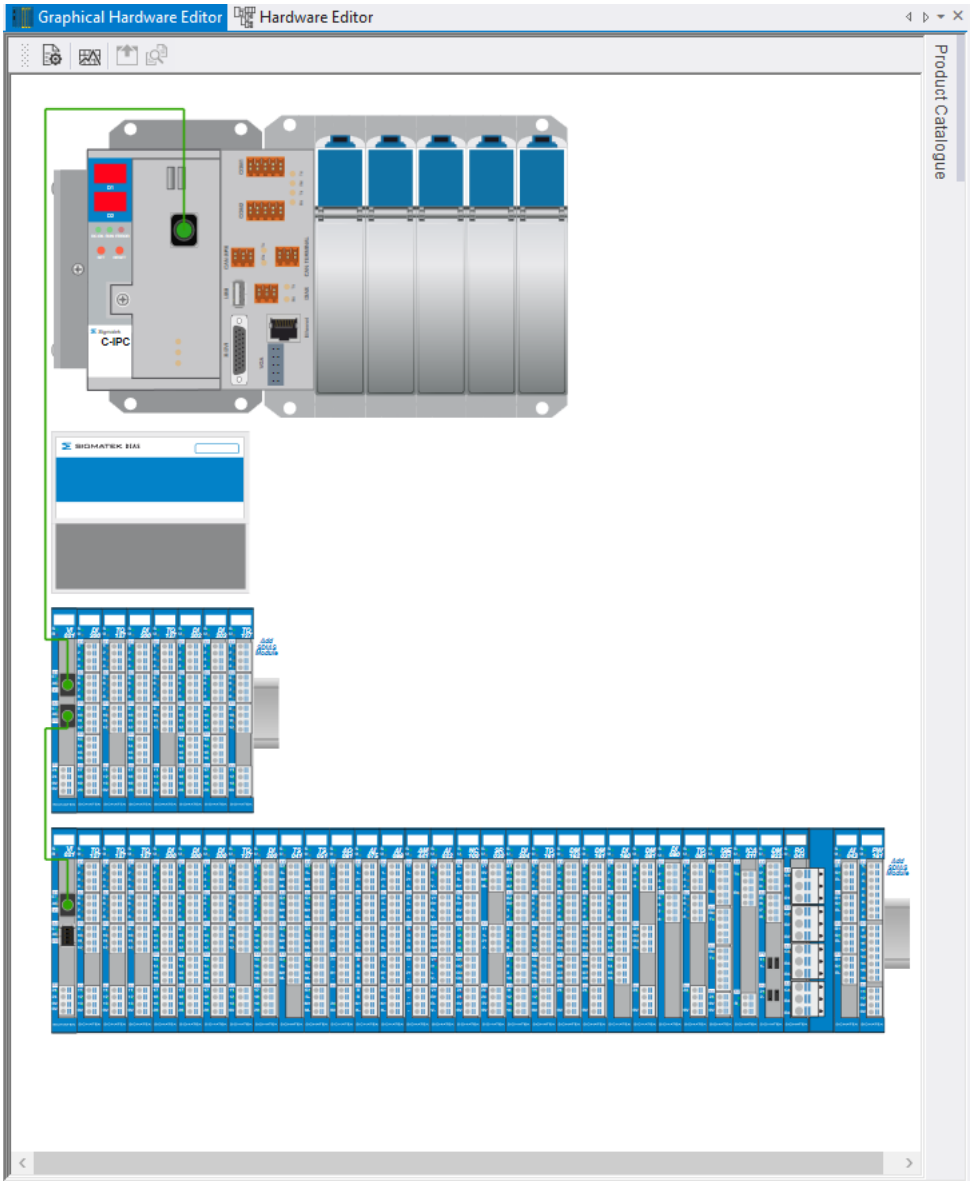
4.2.14 Graphical Hardware Editor

In addition to the HW editor, there is a completely graphic display variant: the graphic HW editor (GHE). The goal of this display form is to simulate the physical hardware configuration as detailed as possible. The GHE is fully integrated into the HW editor / object network / connection manager structure, which means the changes in one display are made to all others in real time. There is also a one-sided, indirect connection to the network tree (from the context menu via **Show Return** in the in the object network and **Goto Graphical Hardware Editor** in the GHE).

There are several features / limitations, which must be taken into consideration when using the GHE:

- **User-defined classes cannot be displayed**, classes can only be selected from the product catalog included in delivery.
- **GHE supports SDIAS** and **VARAN**.

The GHE – analog to the HW editor and object network – is displayed in the workspace.


Graphical Hardware Editor
Hardware Editor
Product Catalogue
C-IPC

The window is divided into three sections: The toolbar, the tab for the product catalog and the actual workspace.

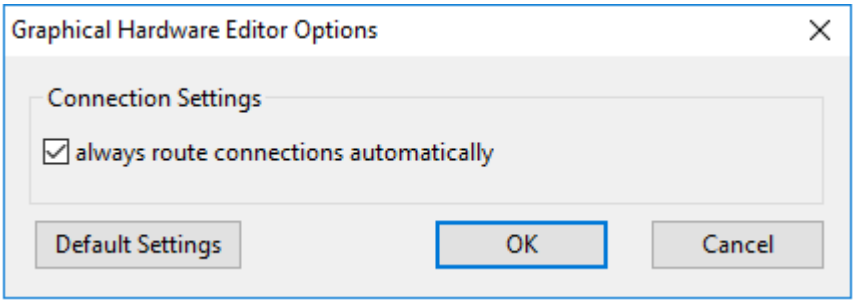
4.2.14.1 The Toolbar



The two icons on the left side are always active, the two on the right side are only active in online mode. Starting from left to right:

Hardware Editor Configuration Settings

With a right click, the following dialog opens:

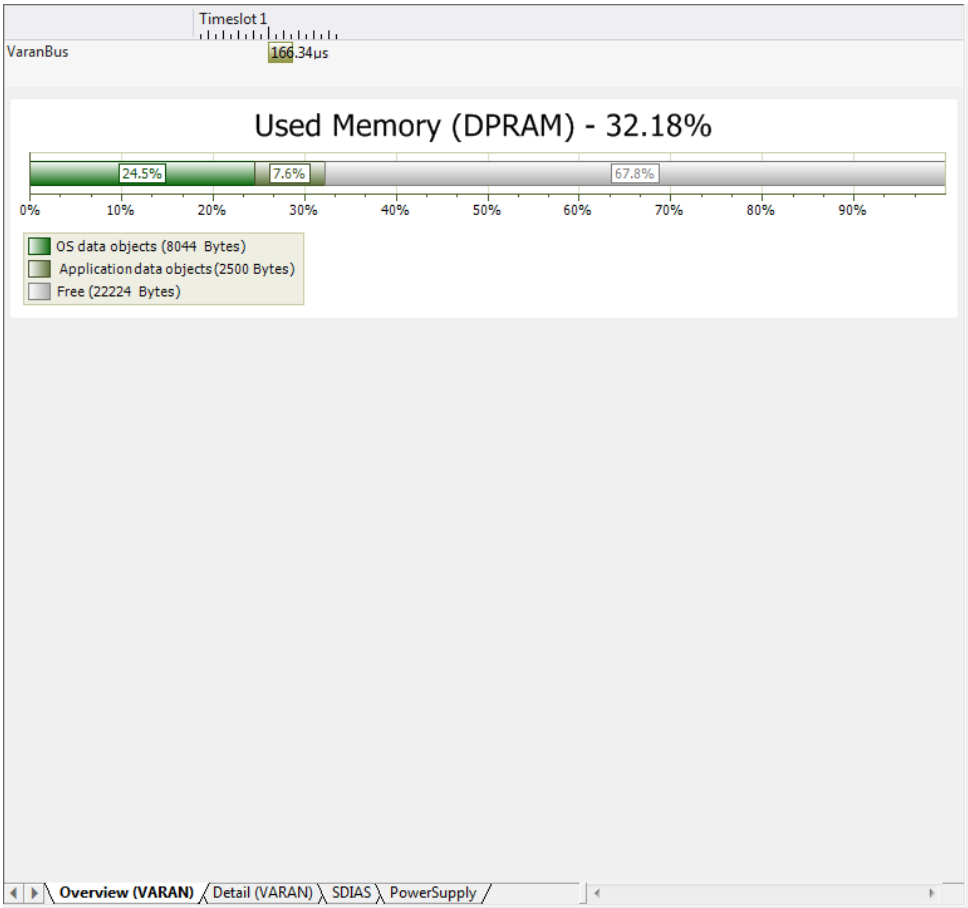


The **always route connections automatically** checkbox enables the recalculation and display of the connections after each change in the workspace (moving, adding, deleting etc.). This is default setting. If not desired, the checkbox can be deselected.

With a left-click on **Default Settings**, the checkbox is reselected. With **OK**, the settings are accepted. With **Cancel**, they are discarded.

Calculate Resources of Project

With this link the following area opens at the right side of the workspace:



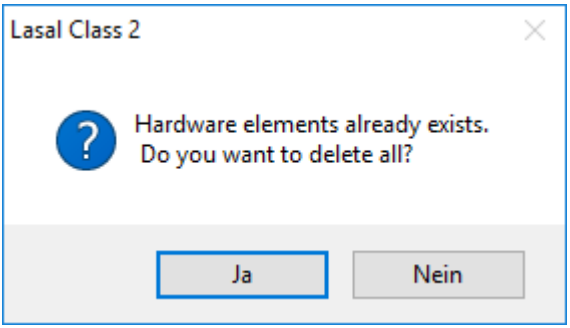
In the tab **Overview (VARAN)** the **Iso start point of the VARAN communication** depending on the defined VARAN manager time (**top**) as well as the **consumption of the single data objects** and the available memory (**bottom**) is displayed.

If the tab **Detail (VARAN)** at the bottom is clicked, the times as well as the DPRAM memory resources of the VARAN objects are shown in detail. In the **SDIAS** tab the resource data of the single SDIAS modules is displayed; under **Power Supply** the sum current calculation incl. the leakage currents is displayed.

For more information see the chapter Resources Calculation.

Upload Hardware Tree from PLC

This icon is only active in online mode and with a loaded project. If the hardware is already in the GHE, two options are then provided in a dialog: With “yes”, all HW elemets are deleted before uploading or with no, only those that don’t match the PLC configuration are deleted. The other – including those available in the PLC – HW elements retain their current configuration in this case.



Compare Hardware Tree of Project and PLC \ Reset Online Compare

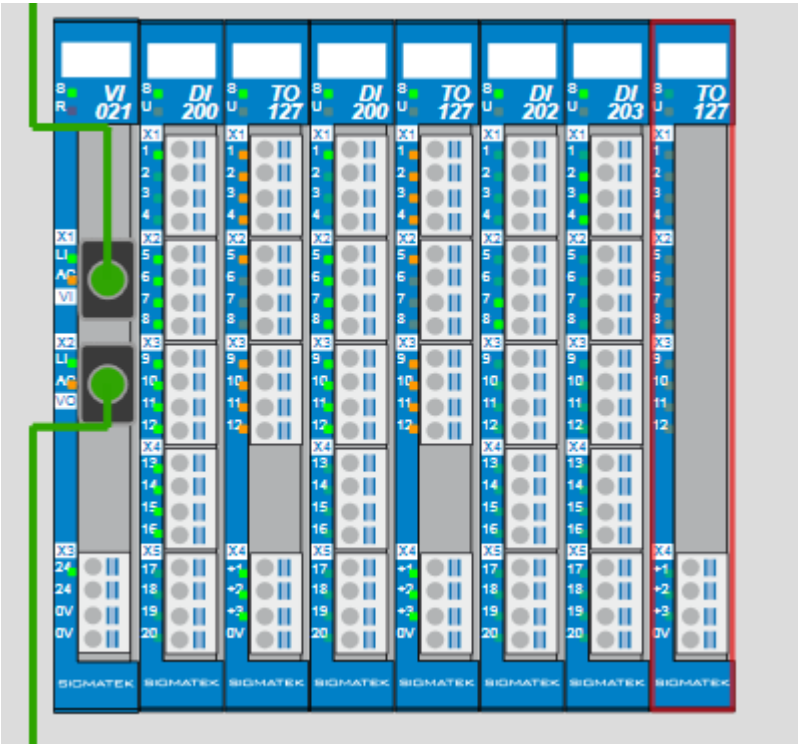
This icon is also only active in online mode. If selected, whether the project and the PLC configuration are identical is monitored. If so, no further action is taken; except a logging entry. If differences are detected however, the excessive modules in the project are displayed in red in the GHE ...



... and in the output window of the **“Find Results 2”** tab, an error message in the following form is triggered:

[14:34:43 (ERROR) FindResult2] Module 'ICA011' (Module 'ICA0112') configured in project is not available at the PLC.

If the hardware from the PLC is not available in the GHE however, the module– also framed in red – is inserted into the GHE ...



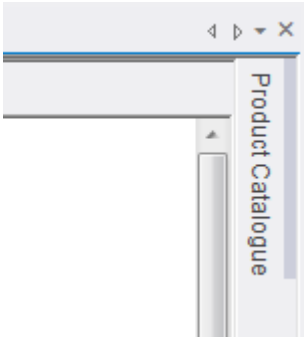
... and error messages are displayed in the output window of the **Find Results 2** tab:

[14:41:23 (ERROR) FindResult2] Module 'TO127' found on PLC

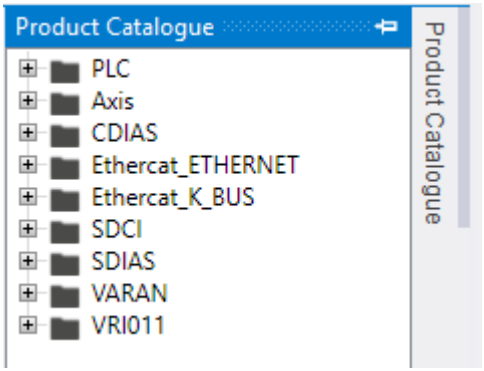
[14:41:23 (ERROR) FindResult2] Module(s) at PLC are not configured in project.

4.2.14.2 Product Catalog

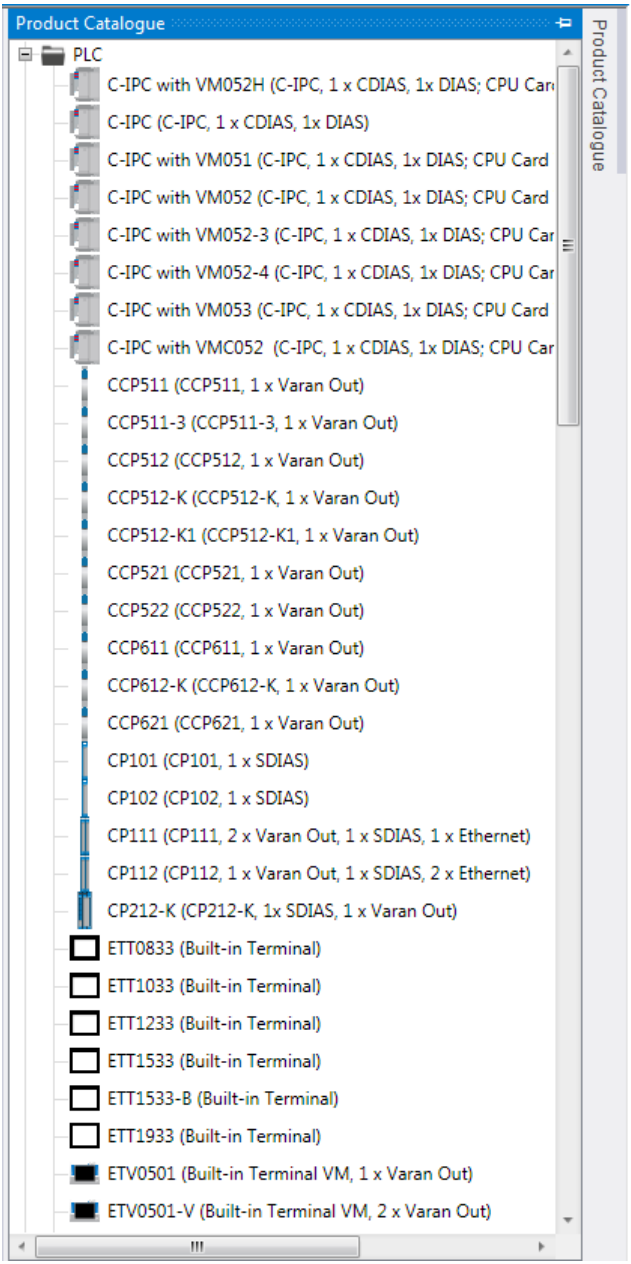
The product catalog contains all SIGMATEK components resp. modules, which can be displayed in the Graphical Hardware Editor. It is hidden behind a vertical tab:



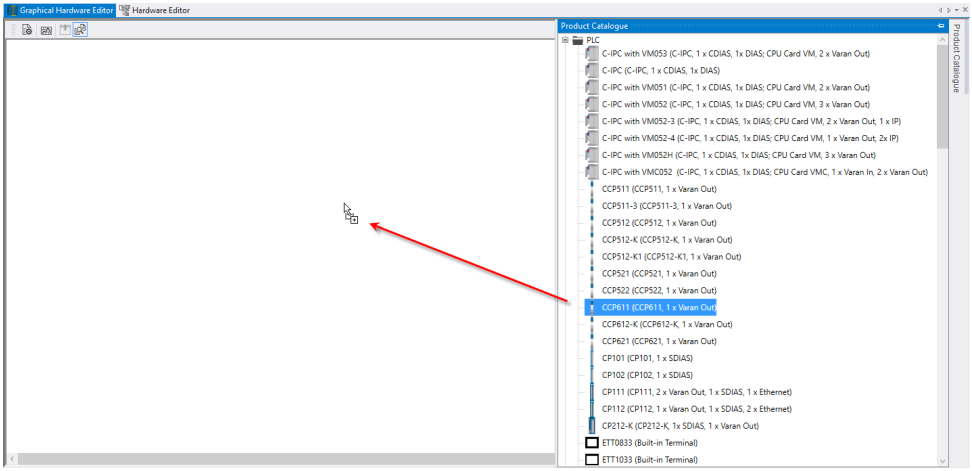
If this tab is clicked, the modules are shown in groups:



With a further click, the groups are expanded and the individual modules are then visible:

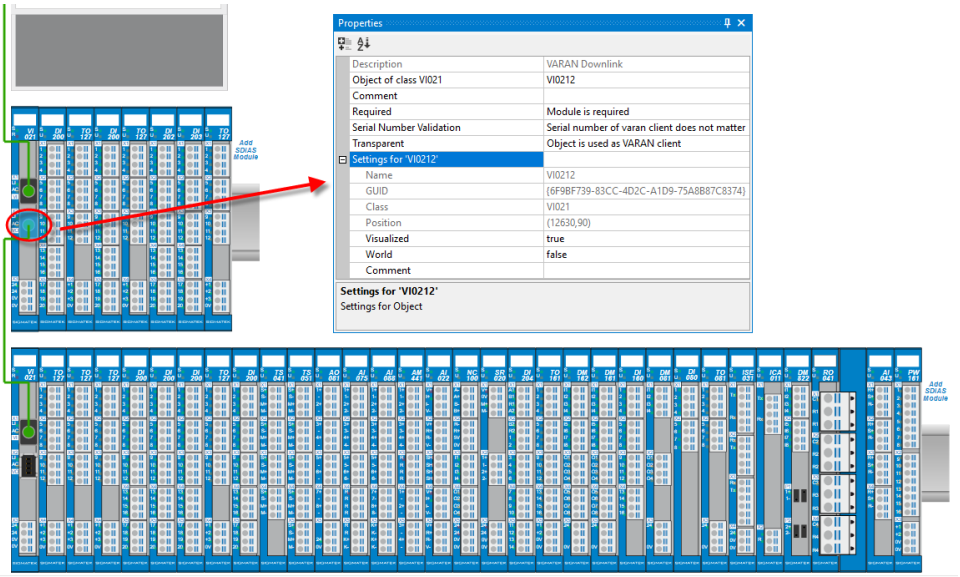

Product Catalogue
PLC
C-IPC with VM052H (C-IPC, 1 x CDIAS, 1x DIAS; CPU Car
C-IPC (C-IPC, 1 x CDIAS, 1x DIAS)
C-IPC with VM051 (C-IPC, 1 x CDIAS, 1x DIAS; CPU Card
C-IPC with VM052 (C-IPC, 1 x CDIAS, 1x DIAS; CPU Card
C-IPC with VM052-3 (C-IPC, 1 x CDIAS, 1x DIAS; CPU Car
C-IPC with VM052-4 (C-IPC, 1 x CDIAS, 1x DIAS; CPU Car
C-IPC with VM053 (C-IPC, 1 x CDIAS, 1x DIAS; CPU Card
C-IPC with VMC052 (C-IPC, 1 x CDIAS, 1x DIAS; CPU Car
CCP511 (CCP511, 1 x Varan Out)
CCP511-3 (CCP511-3, 1 x Varan Out)
CCP512 (CCP512, 1 x Varan Out)
CCP512-K (CCP512-K, 1 x Varan Out)
CCP512-K1 (CCP512-K1, 1 x Varan Out)
CCP521 (CCP521, 1 x Varan Out)
CCP522 (CCP522, 1 x Varan Out)
CCP611 (CCP611, 1 x Varan Out)
CCP612-K (CCP612-K, 1 x Varan Out)
CCP621 (CCP621, 1 x Varan Out)
CP101 (CP101, 1 x SDIAS)
CP102 (CP102, 1 x SDIAS)
CP111 (CP111, 2 x Varan Out, 1 x SDIAS, 1 x Ethernet)
CP112 (CP112, 1 x Varan Out, 1 x SDIAS, 2 x Ethernet)
CP212-K (CP212-K, 1x SDIAS, 1 x Varan Out)
ETT0833 (Built-in Terminal)
ETT1033 (Built-in Terminal)
ETT1233 (Built-in Terminal)
ETT1533 (Built-in Terminal)
ETT1533-B (Built-in Terminal)
ETT1933 (Built-in Terminal)
ETV0501 (Built-in Terminal VM, 1 x Varan Out)
ETV0501-V (Built-in Terminal VM, 2 x Varan Out)
Product Catalogue

The modules can be placed in the workspace via Drag & Drop.

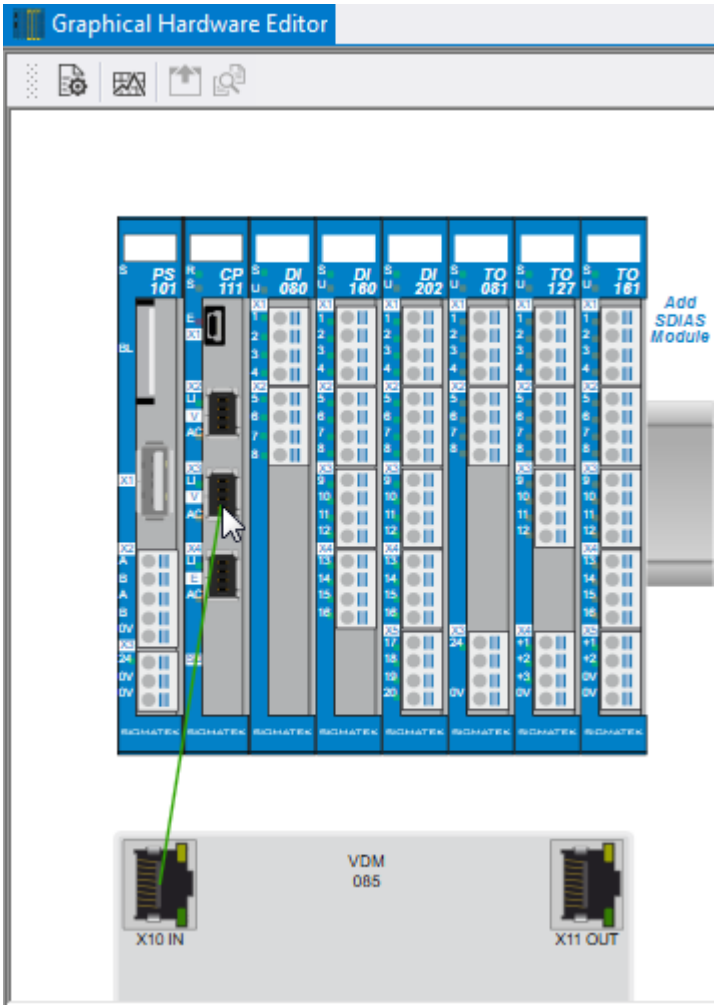


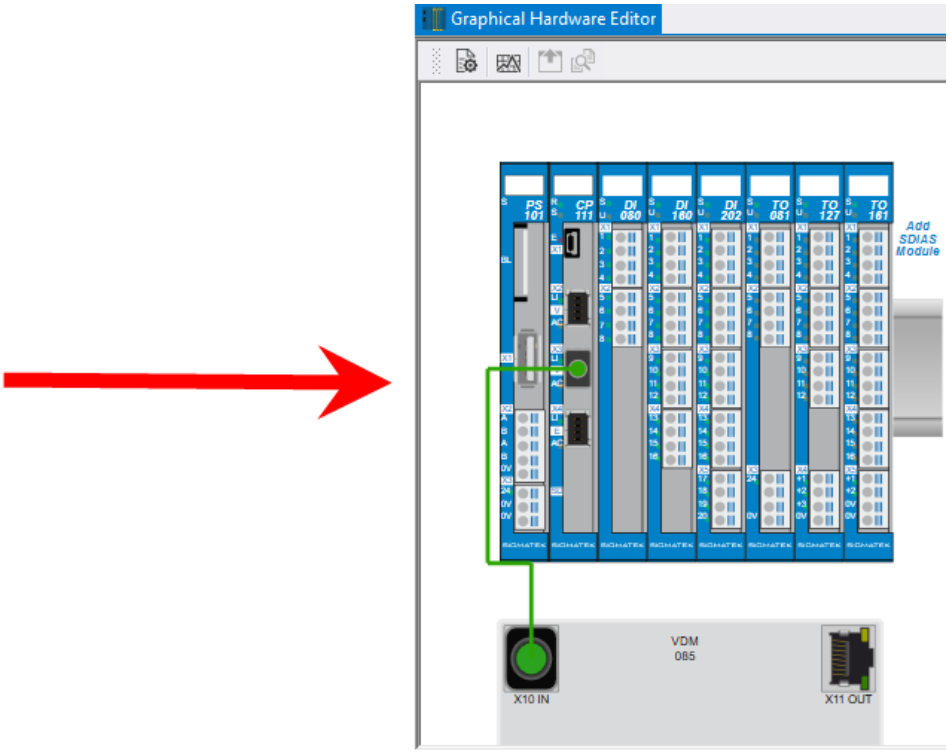
4.2.14.3 The Workspace

Here the hardware is simulated in detail. Not only does the appearance correspond to the reality – the logic and connection options as well. If an in / output or connection option in a module / module is clicked, the corresponding properties – when a connection exists; even that of the connected module – are shown in the properties window:

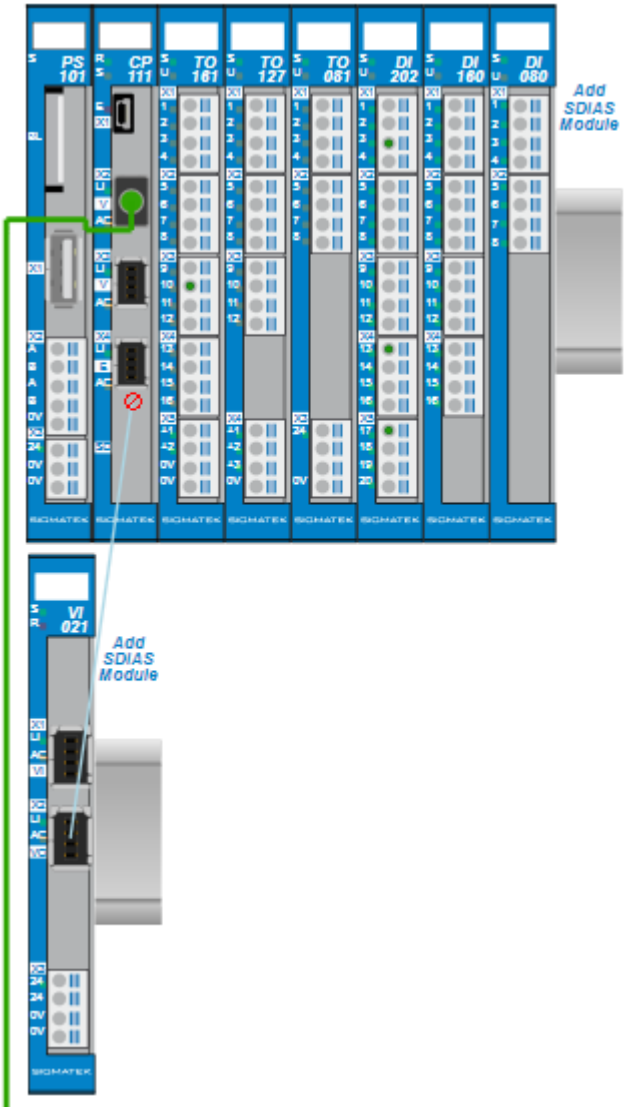


If a connection option is clicked and moved while holding the moust button, a connection can be made directly in the GHE – provided that the counterpart is available.

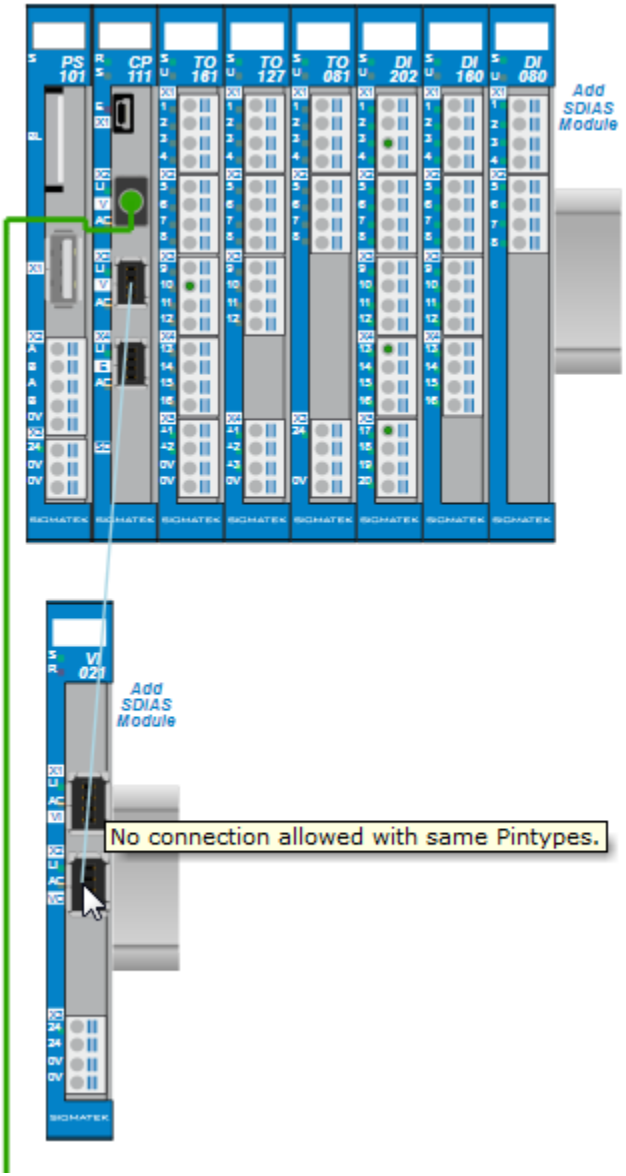

Graphical Hardware Editor
PS 101
CP 111
DI 080
DI 160
DI 202
TO 081
TO 127
TO 161
Add SDIAS Module
VDM 085
X10 IN
X11 OUT



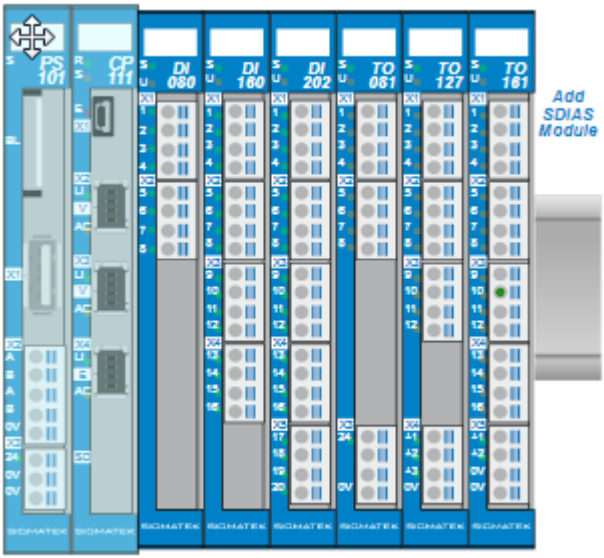
If the line interface does not meet the requirements, this is indicated by a change ...



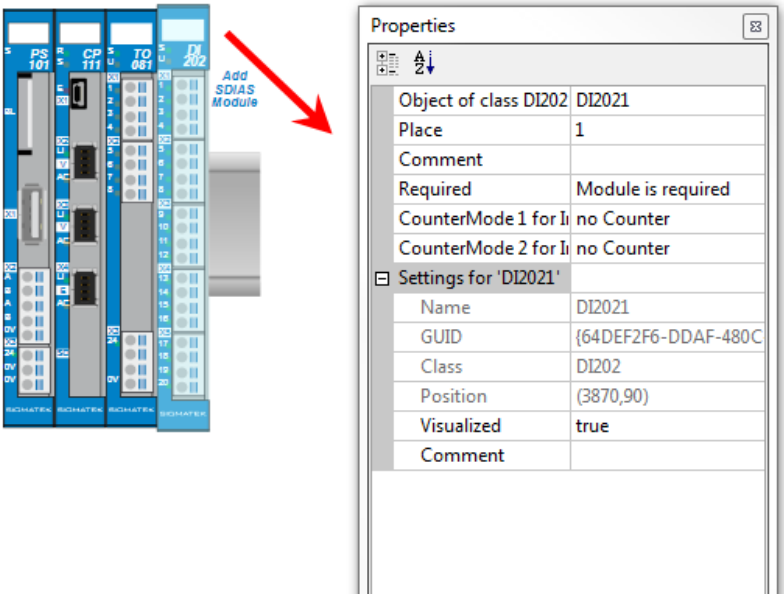
... in the mouse pointer or with a Tooltip:


PS 101
CP 111
TO 161
TO 127
TO 081
DI 202
DI 160
DI 080
Add SDIAS Module
VI 021
Add SDIAS Module
No connection allowed with same Pintypes.

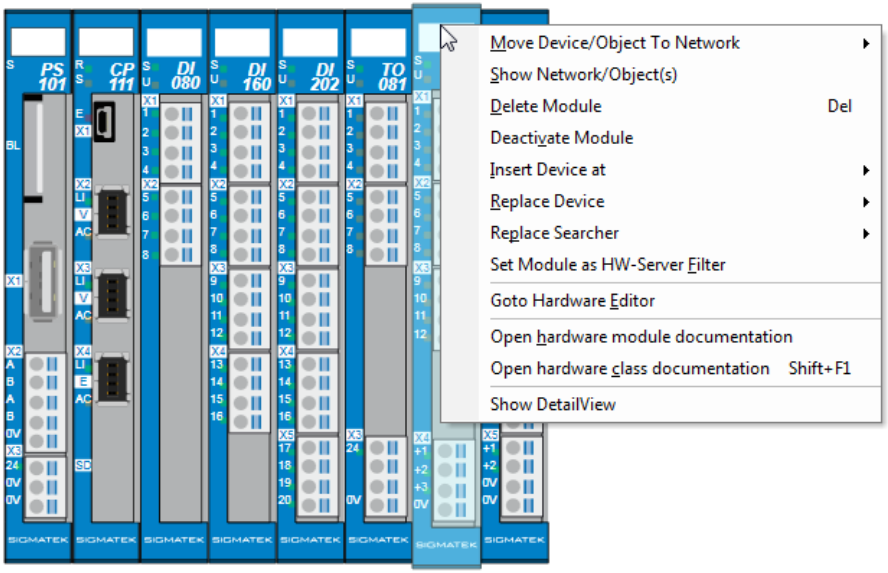
The modules can be moved by selecting them and a location where there are no connection options “touched” with the mouse. This is shown with the “move mouse pointer” common in LASAL:

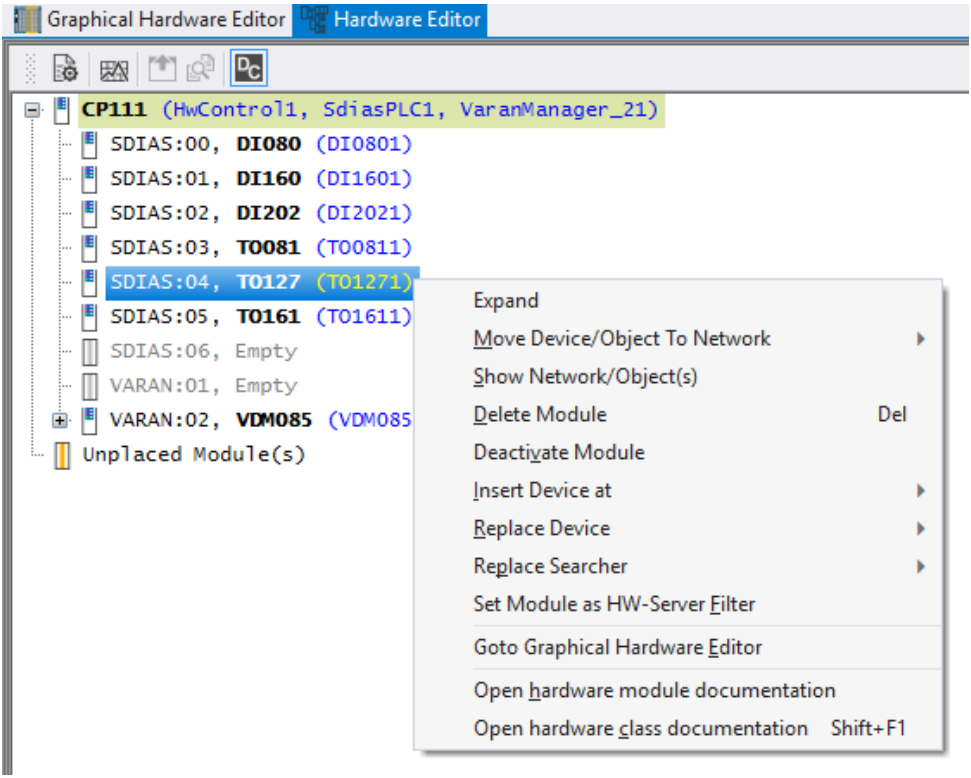


If a module is selected, its properties are displayed in the properties window, where they can also be edited:

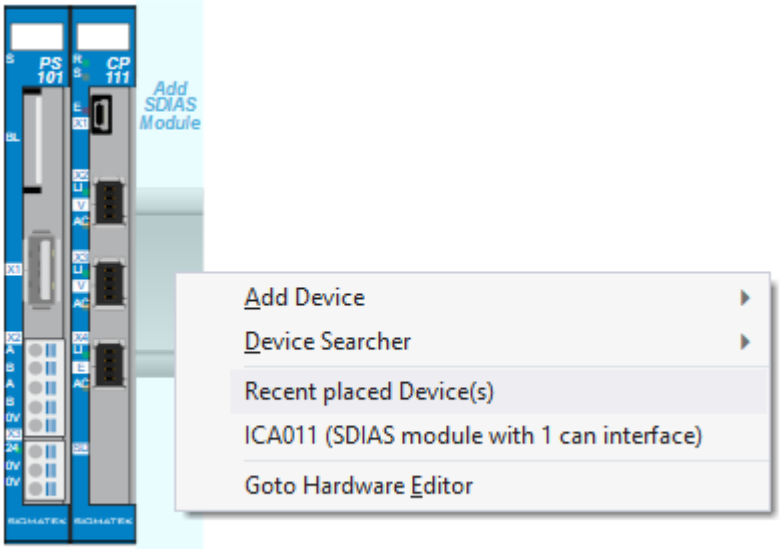


With exception of the menu item **Goto Graphical Hardware Editor** or **Goto Hardware Editor**, the **Expand** option available only in the HW editor and the **Show DetailView** available only in the GHE, the context menu of a module corresponds exactly to that of elements in the hardware editor. Also visible here, is that these two display forms are only for **different views** of the same configuration.


PS 101
CP 111
DI 080
DI 160
DI 202
TO 081
Move Device/Object To Network
Show Network/Object(s)
Delete Module Del
Deactivate Module
Insert Device at
Replace Device
Replace Searcher
Set Module as HW-Server Filter
Goto Hardware Editor
Open hardware module documentation
Open hardware class documentation Shift+F1
Show DetailView

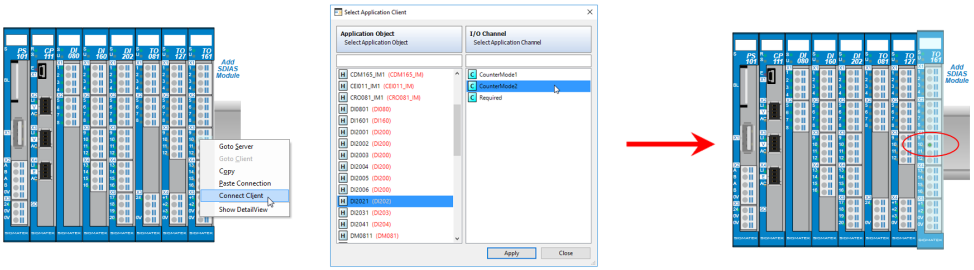


The context menu of a connection interface – similar to the hardware editor – allows a direct connection to a module without having to first Drag & Drop it in the workspace:



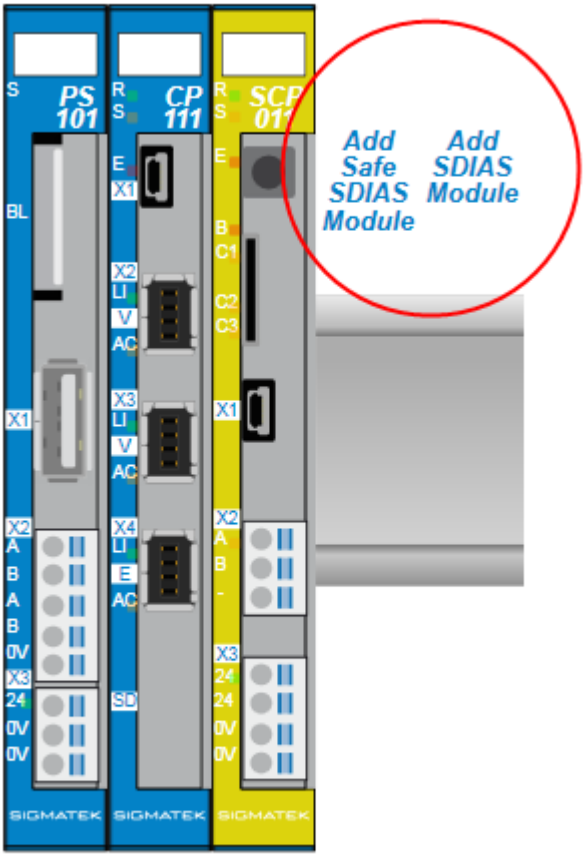
The menu items Add Device, Device Searcher or the direct selection under Recent Placed Devices function as accustomed from the hardware editor and allow the direct connection of a new module.

In the context menu of an input, output or state interface, familiar menu items are found. Connections can thereby be made, as in the object network, using **Copy** and **Paste Connection**. However, there is a more comfortable way: With **Connect Client**, a selection – similar to that in the Connection Manager – the available clients are shown that can now be selected and connected:

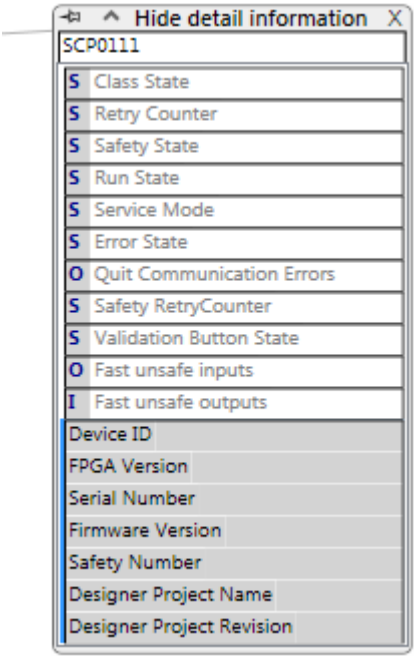


Connected in / outputs as well as down or uplinks are generally displayed in green.

If a Safety CPU was already added, there two free interfaces at the front: one for Safety and one for "normal" S-DIAS modules.

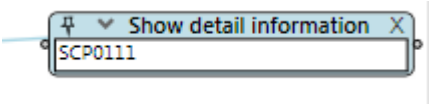


In the context menu of most modules - and several interfaces - the item **Show DetailView** is available. If selected, a detail window opens containing the in and outputs, as well as general information on the module / the connection; however, not input-ready:



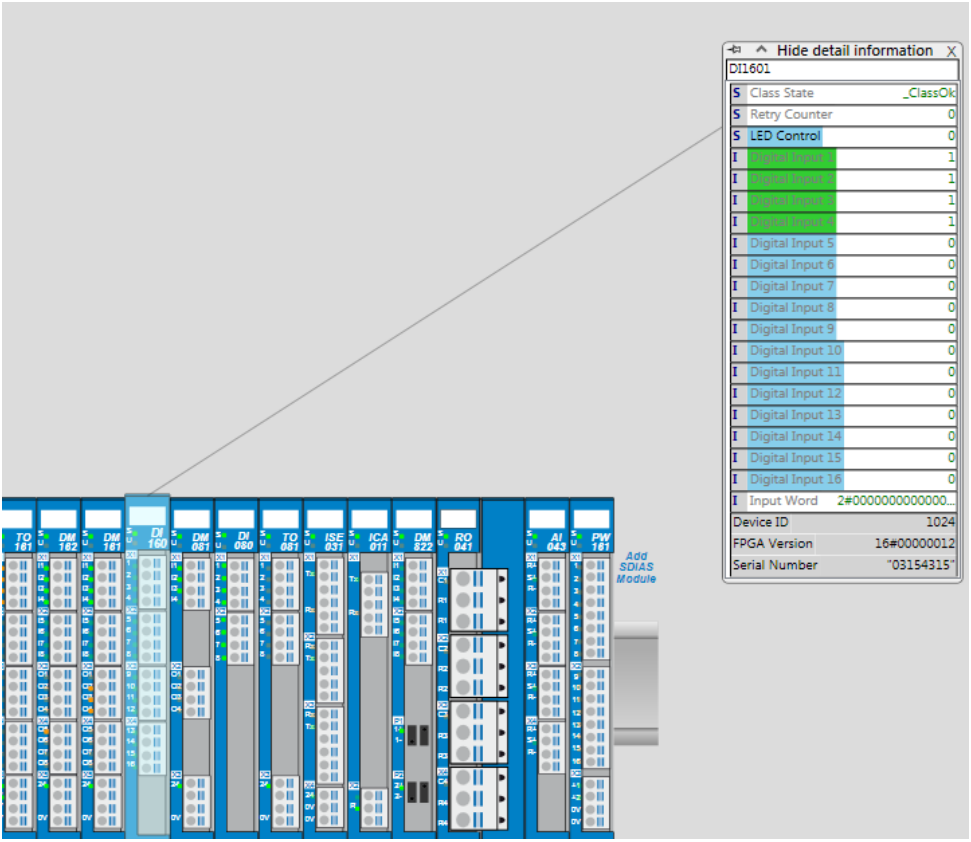
If the DataAnalyzer is active, the **in-/outputs** shown here can be moved to the **DataAnalyzer tree with Drag & Drop**.

In the title bar, this window has a "thumbtack" with which it can be "pinned". When pinned, the window cannot be moved outside of the view in any direction. If **Hide detail information** is clicked, the display is reduced to one line.

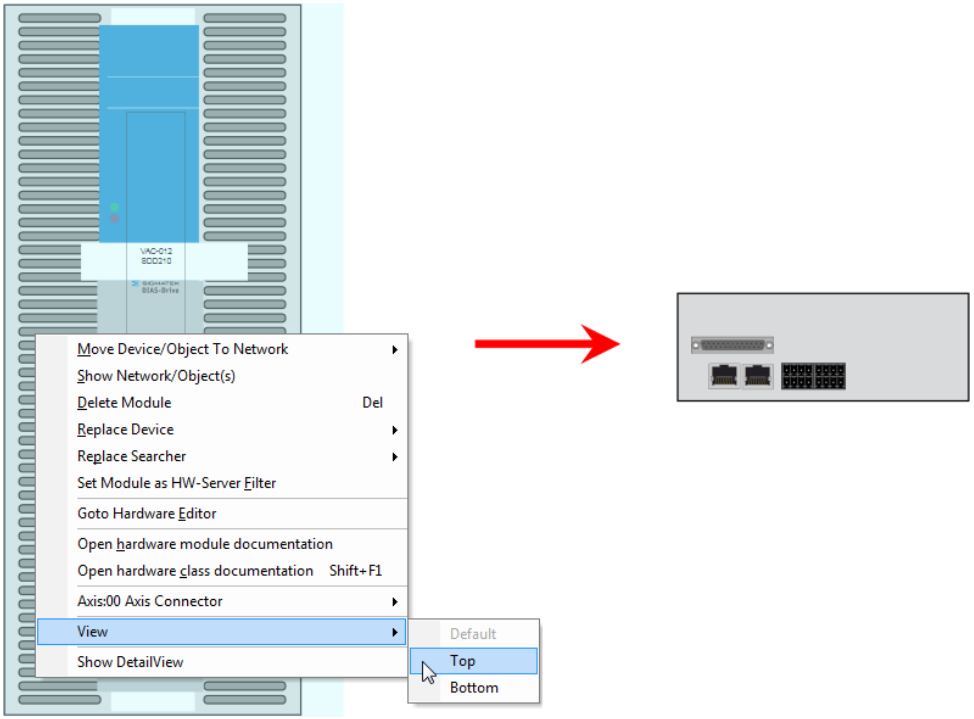


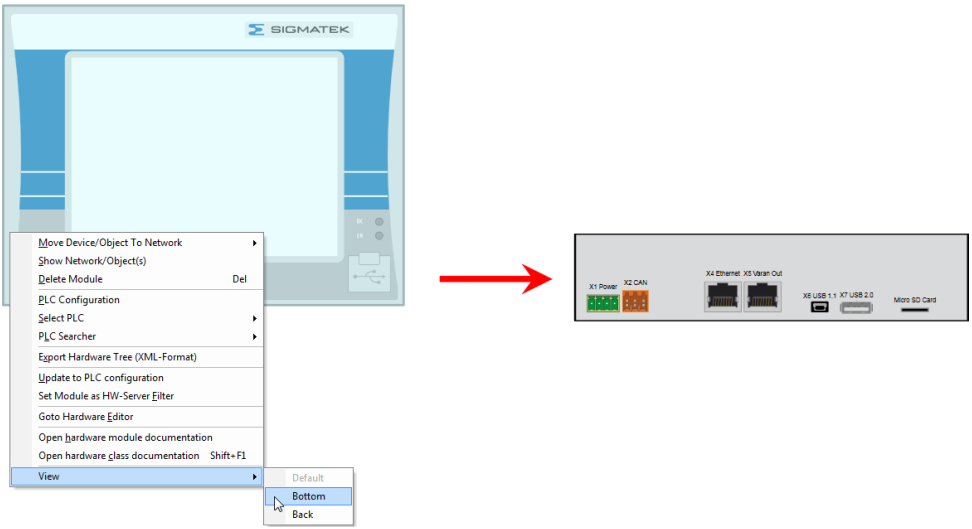
To display the detail view again, **Show detail information** must be clicked.

If online, the current values are shown and if necessary, marked by color:



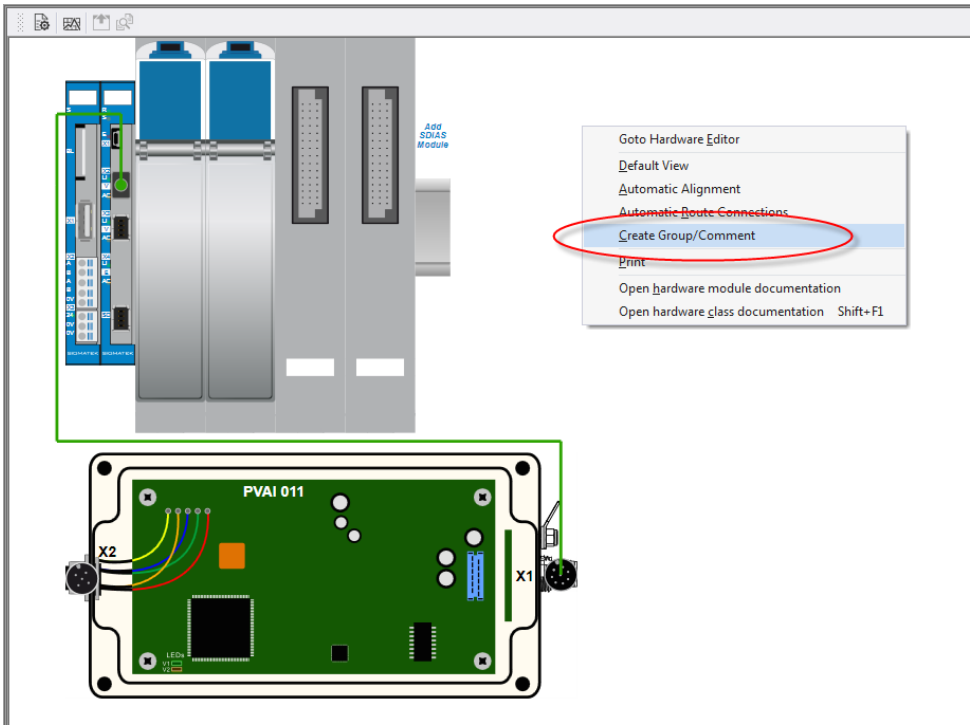
Modules, in which the interfaces are not visible or only partially visible in the standard view, have an additional item in the context menu: **View**. Via the different views (Default, Top, Bottom, ...), the interface to connect can always be made visible.


Move Device/Object To Network
Show Network/Object(s)
Delete Module Del
Replace Device
Replace Searcher
Set Module as HW-Server Filter
Goto Hardware Editor
Open hardware module documentation
Open hardware class documentation Shift+F1
Axis:00 Axis Connector
View
Show DetailView
Default
Top
Bottom

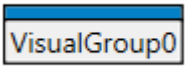


The module can therefore be connected to another component more easily. The view of this module can – also after the connection – be changed at any time.

The option is available to create **commentary** and add components to them (**grouping function**). For this purpose, the context menu is called with a right click on an empty surface and **Create Group/Comment** is selected:

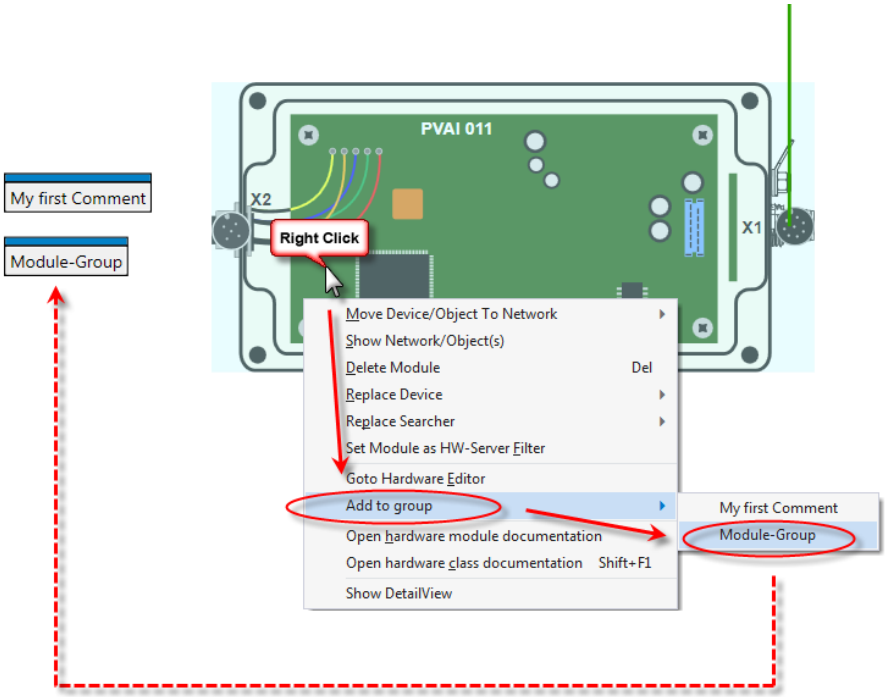


The **group/commentary field** then appears:

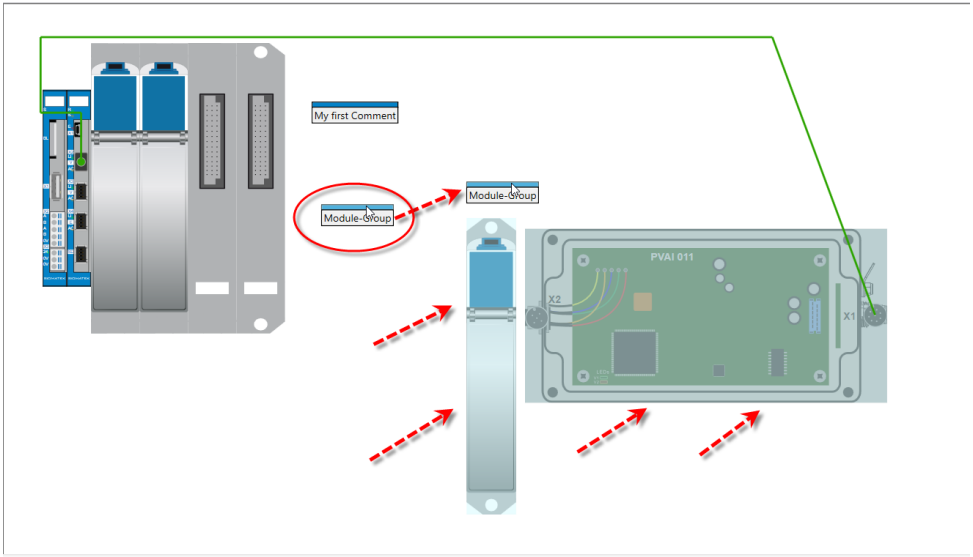


With a **double-click** on the light **text field**, the **content** can now be **edited**.

If comments should not only be created, but also **grouped** as well, **components** can be **added** to the field. For this purpose, **right-click** on the component and select **"Add to Group" in the context menu**. Under this menu item, all available fields are shown. The desired field can now be selected.

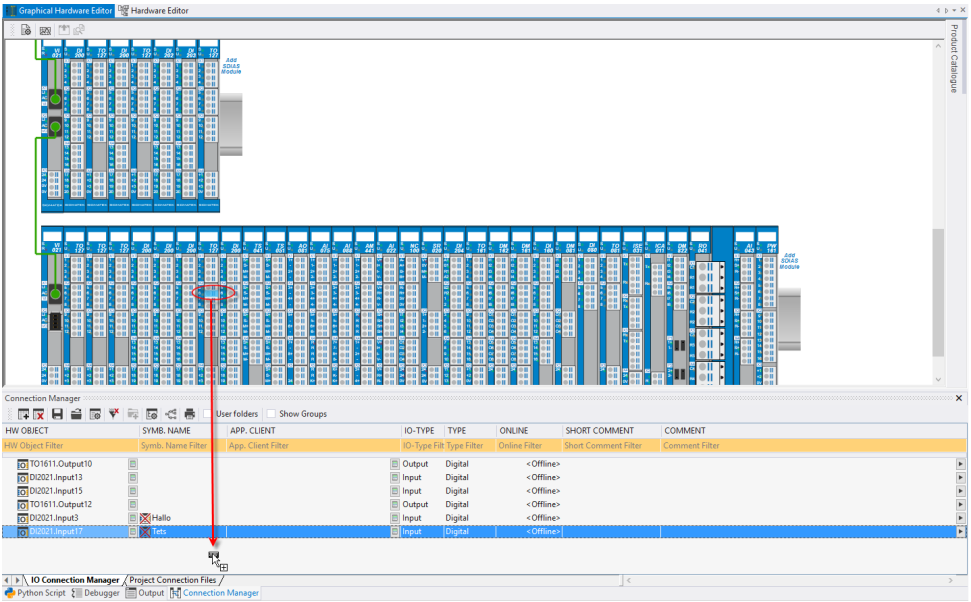


After all desired elements of the group have been added, **all components** can be **selected at the same time** (left-click on the blue header bar of the group field) **and moved** (click and hold left mouse button to move the entire group). The **relative position of the components** for the group field is **maintained** as long as the group is being moved.



If a **component** is clicked **separately**, its **position** can be **changed** as desired. If the group field is clicked, all corresponding components are moved together maintaining their relative position. It is also possible to move the **group field only**, while all components keep their absolute position. For this purpose, press the **Ctrl key** while moving the blue header bar of the group field.

As in the hardware editor, an in or output can be selected in the GHE and moved to the Connection Manager by pressing and holding the left mouse button. Connections can thereby be made more comfortably.


Graphical Hardware Editor
Hardware Editor
Product Catalogue
Connection Manager
User folders
Show Groups
HW OBJECT
SYMB. NAME
APP. CLIENT
IO-TYPE
TYPE
ONLINE
SHORT COMMENT
COMMENT
TO1611.Output10
DI2021.Input13
DI2021.Input15
TO1611.Output12
DI2021.Input3
DI2021.Input17
Hallo
Tets
IO Connection Manager
Project Connection Files
Python Script
Debugger
Output
Connection Manager

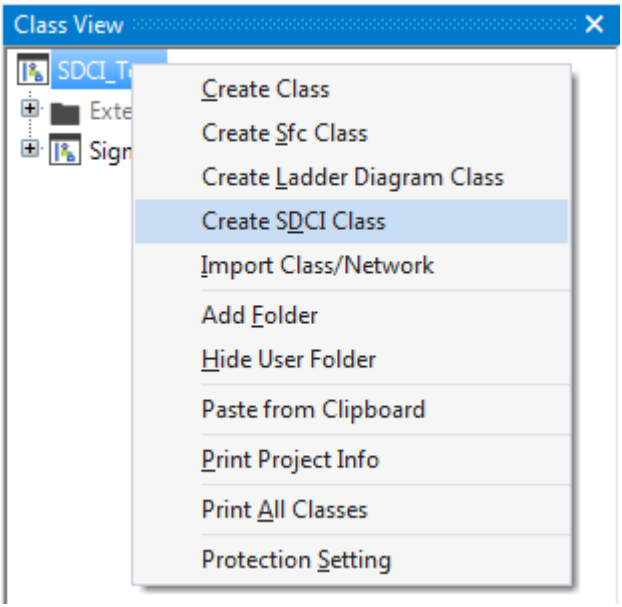
4.3 Connection of SDCI Devices

Via the connection of SDCI devices, it is possible to integrate modules (actuators, Sensors) that have SDCI communication into the LASAL CLASS 2 hardware configuration. Here, an SDCI class is created that contains all the interface functions required for setting parameters as well as running processes and event data of an SDCI device.

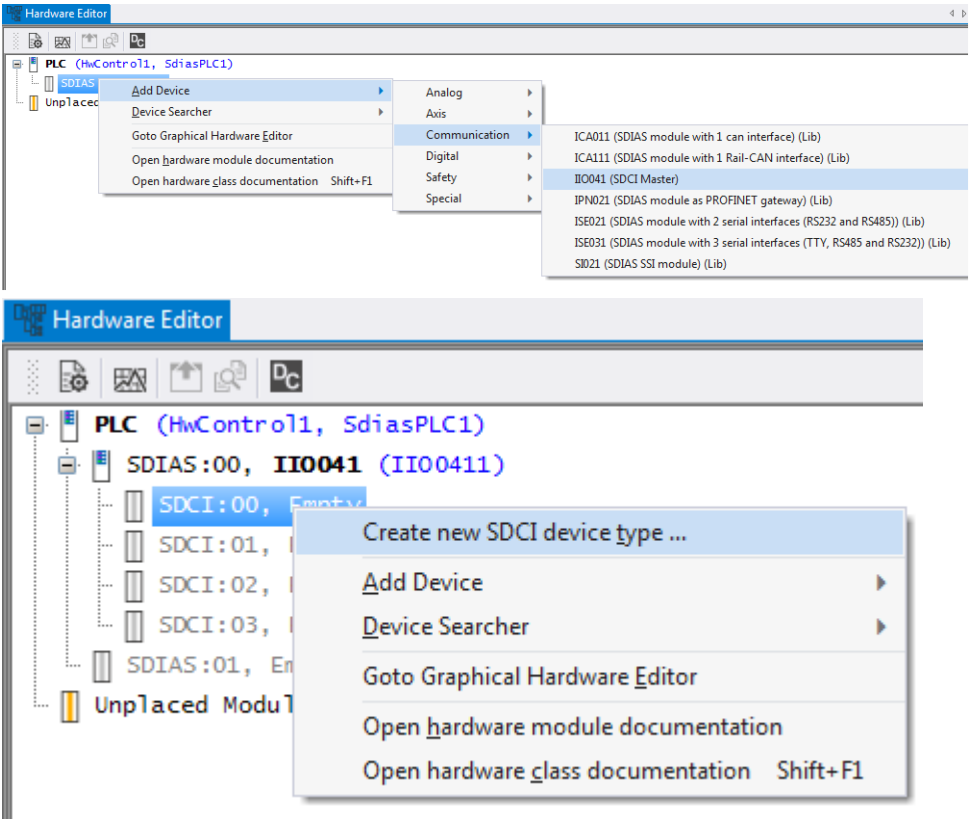
In the following chapters the integration of SDCI devices in LASAL CLASS 2 is explained; more detailed information on using SDCI classes can be found in the documentation of the hardware library in the chapter SDCI.

4.3.1 Creating a SDCI Device Class

An SDCI device class can be generated in the Class tree in the context menu of a project ...

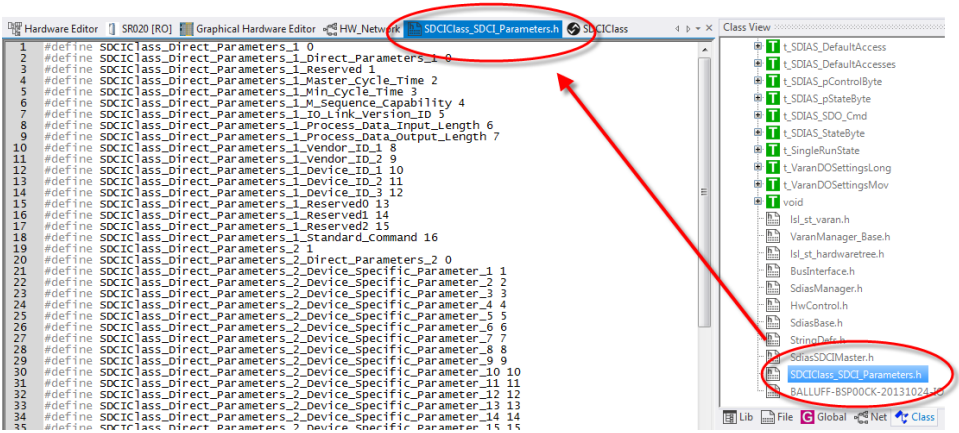


... or via the hardware editor.



The generated SDCI class is derived from a special hardware class for this purpose. This class contains all functions required to establish communication with the SDCI device. The exact description can be found in the hardware classes.

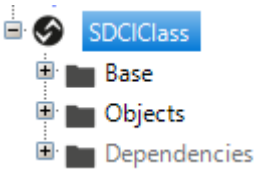
Also a header file with the name **SDCIClass_SDCI_Parameters.h** is created automatically. In the beginning it is empty. When a SDCI device is imported (see chapter Connecting a SDCI Device) and the project is saved, this header file is filled with defines, which allow the access to the single (sub)parameters via *ClassName_ParameterName(_SubparameterName)*.




```
#define SDCIClass_Direct_Parameters_1 0
#define SDCIClass_Direct_Parameters_1_Direct_Parameters_1 0
#define SDCIClass_Direct_Parameters_1_Reserved 1
#define SDCIClass_Direct_Parameters_1_Master_Cycle_Time 2
#define SDCIClass_Direct_Parameters_1_Min_Cycle_Time 3
#define SDCIClass_Direct_Parameters_1_M_Sequence_Capability 4
#define SDCIClass_Direct_Parameters_1_IO_Link_Version_ID 5
#define SDCIClass_Direct_Parameters_1_Process_Data_Input_Length 6
#define SDCIClass_Direct_Parameters_1_Process_Data_Output_Length 7
#define SDCIClass_Direct_Parameters_1_Vendor_ID_1 8
#define SDCIClass_Direct_Parameters_1_Vendor_ID_2 9
#define SDCIClass_Direct_Parameters_1_Device_ID_1 10
#define SDCIClass_Direct_Parameters_1_Device_ID_2 11
#define SDCIClass_Direct_Parameters_1_Device_ID_3 12
#define SDCIClass_Direct_Parameters_1_Reserved0 13
#define SDCIClass_Direct_Parameters_1_Reserved1 14
#define SDCIClass_Direct_Parameters_1_Reserved2 15
#define SDCIClass_Direct_Parameters_1_Standard_Command 16
#define SDCIClass_Direct_Parameters_2 1
#define SDCIClass_Direct_Parameters_2_Direct_Parameters_2 0
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_1 1
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_2 2
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_3 3
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_4 4
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_5 5
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_6 6
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_7 7
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_8 8
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_9 9
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_10 10
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_11 11
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_12 12
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_13 13
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_14 14
#define SDCIClass_Direct_Parameters_2_Device_Specific_Parameter_15 15
```

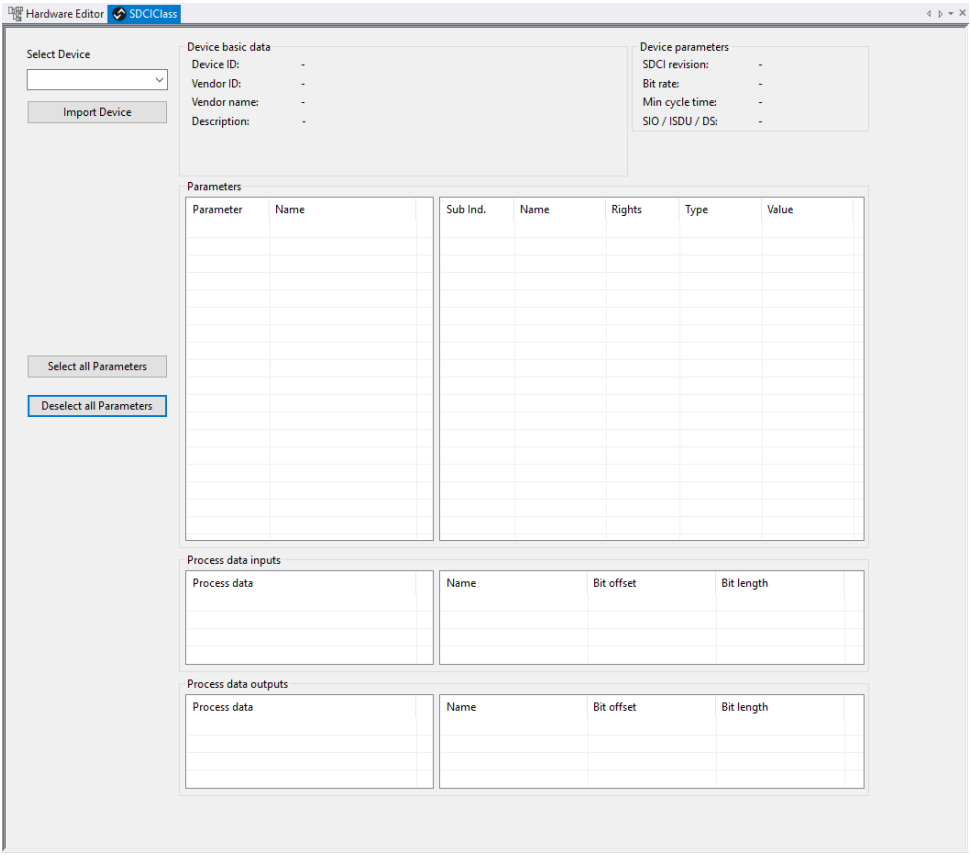
These defines can only be used after compiling!

In the project tree, no clients and servers are displayed in such a class.



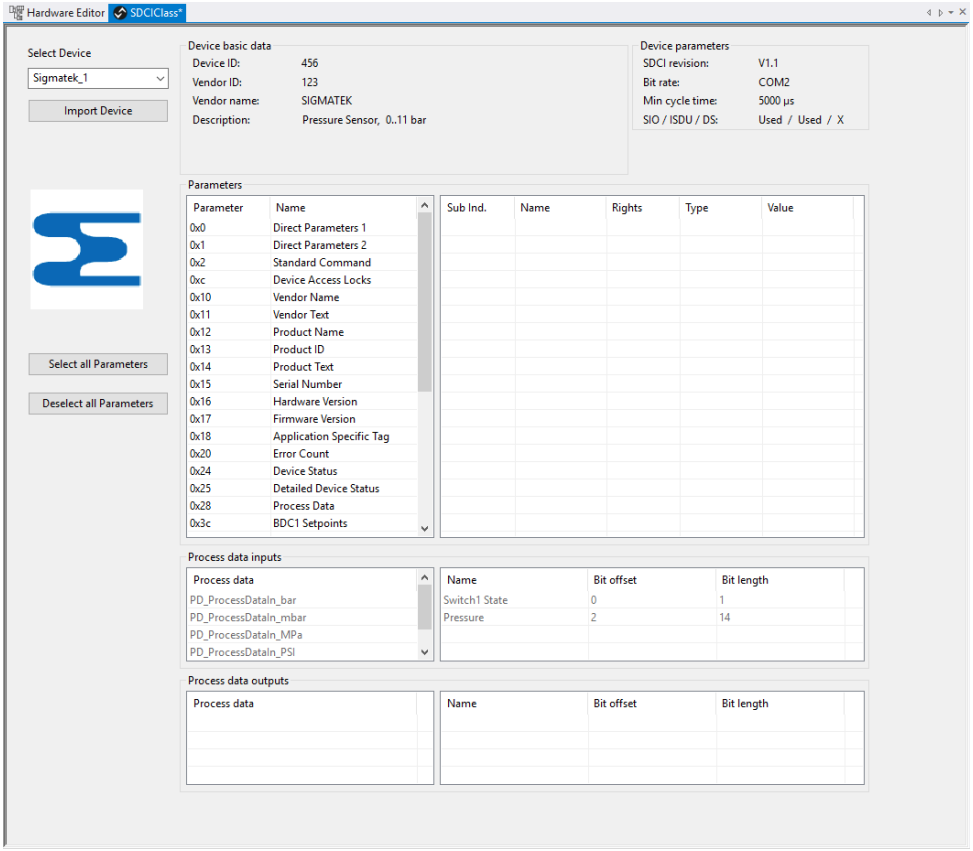
4.3.2 Connecting a SDCI Device

With a double click on the class, the following window opens in the workspace:



With a click on "Import Device", a corresponding IODD file (IO Device Description) in *.xml format is loaded. Only one file can be loaded per class. The file contains information for a single SDCI device. In the Combobox "Select Drive", all SDCI devices supported by the IODD file are shown. From this Combobox, an SDCI device can be selected for the SDCI class.

A header file with the name **SDCIClass_SDCI_Parameters.h** is automatically created. In it the parameters and subparameters of the SDCI device are written as defines, so that they can be accesses directly with their names after saving and compiling.

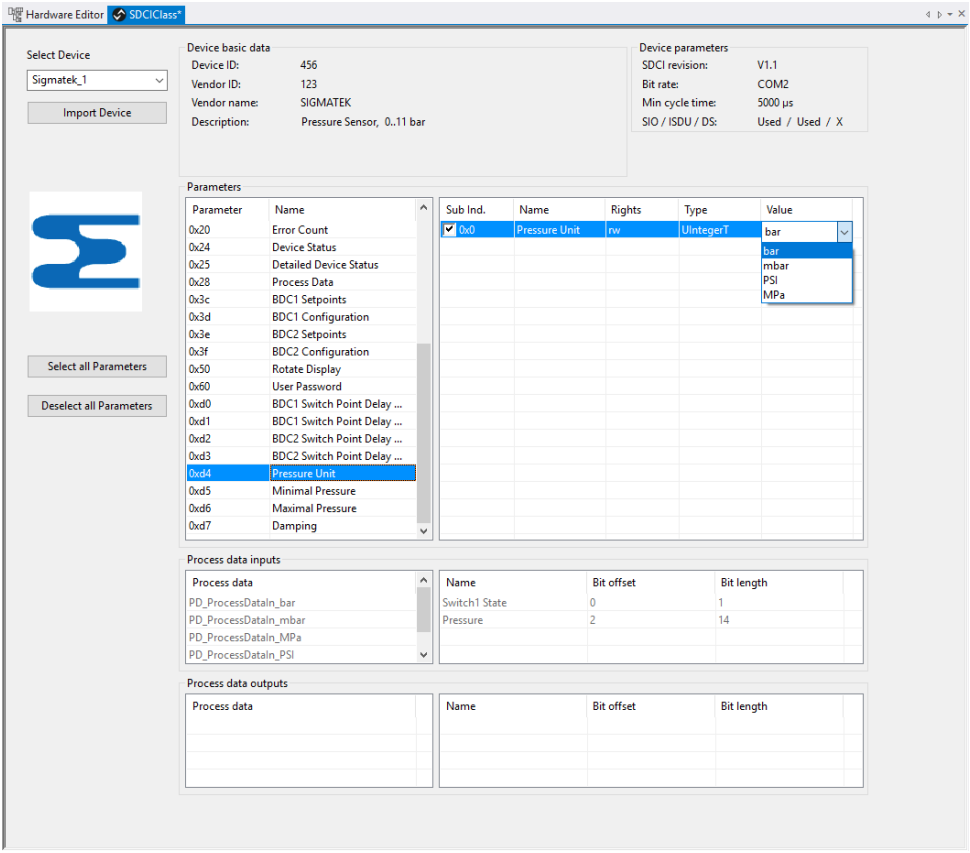


Through the selection of an SDCI device, only the corresponding device data is shown in the dialog. The basic (e.g. Device ID, Vendor ID...) and device information (e.g. revision, bit rate...) are listed, as well as the individual parameters of the device.

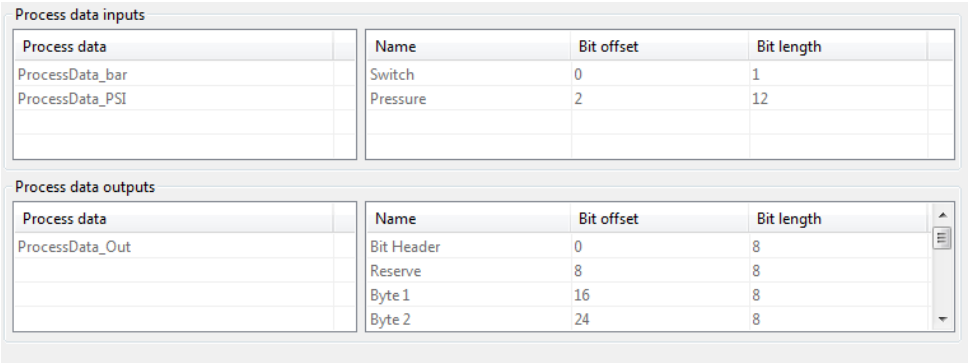
The parameters can be partially changed by the user (depending on the settings in the IODD file). Parameters, which cannot be changes are greyed. The changes are made either in a Combobox or a text editor with the value of the sub-parameter. The validity of the input is checked and in the event of an error, the value is displayed in red. The allowable values for text editors are shown in a tooltip. Which values of the SDCI device should be written can also be specified. For this purpose, the checkbox in the Sub-Indices column must be set. The checkbox is only visible when a change is possible. The

checkboxes for all parameters can also be activated or deactivated at the same time using the buttons 'Select all Parameters' and 'Deselect all Parameters'.

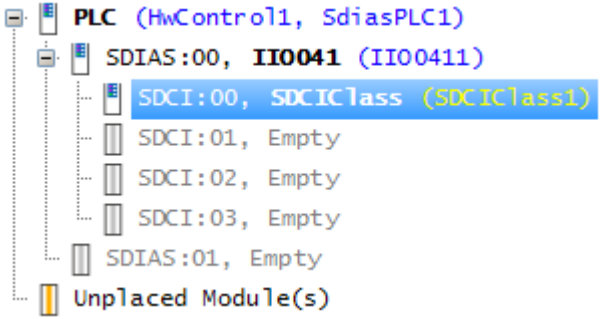
Default parameters can now be defined:



Additionally, the process data for the inputs and outputs are displayed in the dialog. This display serves as information for the structure of the process data. The names, the bit offsets and the bit lengths of single process data are listed. Further information can be found in the data sheets of the SDCI device manufacturers.



If the SDCI device class is generated through a free SDCI slot in the hardware, it is then placed at the selected slot:



The newly created class can now, as with SIGMATEK hardware classes, be placed in the hardware editor and reused. The classes generated via this process can also be managed in a library.

If the project is now saved, all SDCI data (SDCI device class, selected SDCI devices, parameters, ...) are automatically included.

4.4 DiasDrive

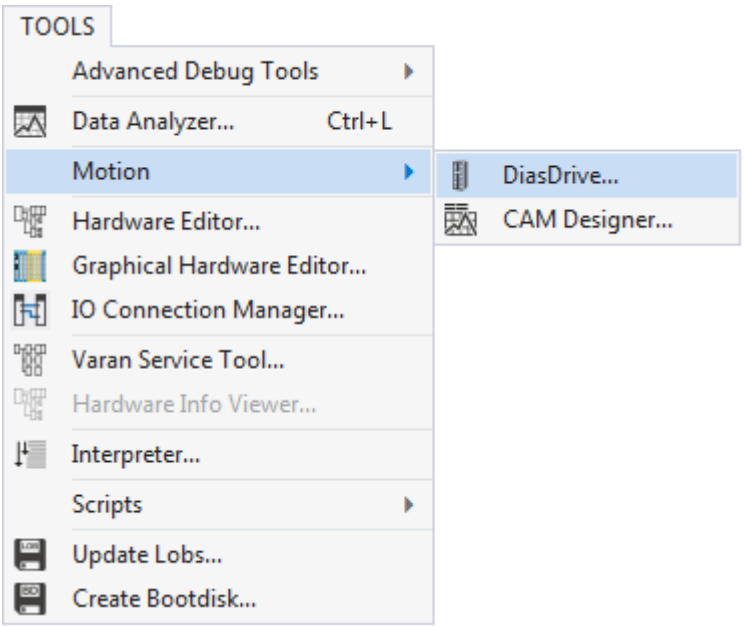
The DiasDrive area in LASAL CLASS 2 contains the configuration and calibration of Servo Drive hardware modules, which are included into an operation system via VARAN bus. The integration of servo information to a LASAL CLASS 2 project is also included in this area.

4.4.1 What is VARAN?

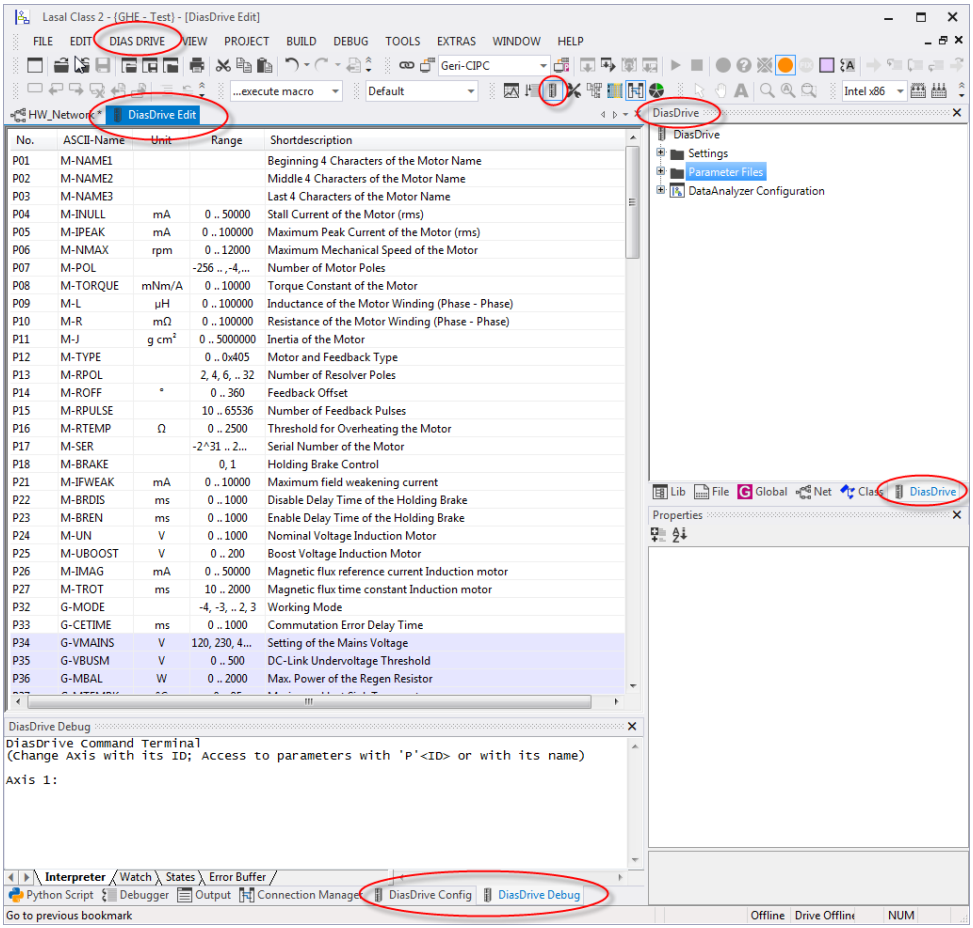
See http://www.varan-bus.net/

4.4.2 Accessibility

For activation of the Dias Drive environment select the menu entry **Tools, Motion, DiasDrive** or the  tool in the toolbar.



Thereby different windows are opened automatically. Additionally the menu entry **Dias Drive** is installed in the menu bar.



Through closing the **DiasDrive Edit window**, or via selecting the menu entry **Tools, DiasDrive** (or the **DiasDrive symbol** ), this environment (window and menu) is closed.

4.4.2.1 DiasDrive Edit

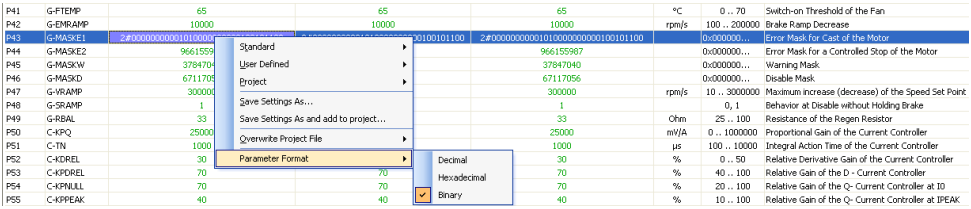
In this window different information of the single parameters are listed. If LASAL CLASS 2 is connected to a Dias Drive (online mode) the value to the parameters of each axis is displayed.

DiasDrive Edit

No.	ASCII-Name	Unit	Range	Shortdescription
P01	M-NAME1			Beginning 4 Characters of the Motor Name
P02	M-NAME2			Middle 4 Characters of the Motor Name
P03	M-NAME3			Last 4 Characters of the Motor Name
P04	M-INULL	mA	0 .. 50000	Stall Current of the Motor (rms)
P05	M-IPEAK	mA	0 .. 100000	Maximum Peak Current of the Motor (rms)
P06	M-NMAX	rpm	0 .. 12000	Maximum Mechanical Speed of the Motor
P07	M-POL		-256 .. ,-4,...	Number of Motor Poles
P08	M-TORQUE	mNm/A	0 .. 10000	Torque Constant of the Motor
P09	M-L	µH	0 .. 100000	Inductance of the Motor Winding (Phase - Phase)
P10	M-R	mΩ	0 .. 100000	Resistance of the Motor Winding (Phase - Phase)
P11	M-J	g cm²	0 .. 5000000	Inertia of the Motor
P12	M-TYPE		0 .. 0x405	Motor and Feedback Type
P13	M-RPOL		2, 4, 6, .. 32	Number of Resolver Poles
P14	M-ROFF	°	0 .. 360	Feedback Offset
P15	M-RPULSE		10 .. 65536	Number of Feedback Pulses
P16	M-RTEMP	Ω	0 .. 2500	Threshold for Overheating the Motor
P17	M-SER		-2^31 .. 2...	Serial Number of the Motor
P18	M-BRAKE		0, 1	Holding Brake Control
P21	M-IFWEAK	mA	0 .. 10000	Maximum field weakening current
P22	M-BRDIS	ms	0 .. 1000	Disable Delay Time of the Holding Brake
P23	M-BREN	ms	0 .. 1000	Enable Delay Time of the Holding Brake
P24	M-UN	V	0 .. 1000	Nominal Voltage Induction Motor
P25	M-UBOOST	V	0 .. 200	Boost Voltage Induction Motor
P26	M-IMAG	mA	0 .. 50000	Magnetic flux reference current Induction motor
P27	M-TROT	ms	10 .. 2000	Magnetic flux time constant Induction motor
P32	G-MODE		-4, -3, .. 2, 3	Working Mode
P33	G-CETIME	ms	0 .. 1000	Commutation Error Delay Time

Here a detailed help function is installed: regardless where one clicks in the window, with F1 the help for the clicked area / paramter opens.

Because there are numeric fields with which hexadecimal or binary display is more intuitive (in the example graphic, there are mask fields that are binary coded), it is possible to switch the number format per field.



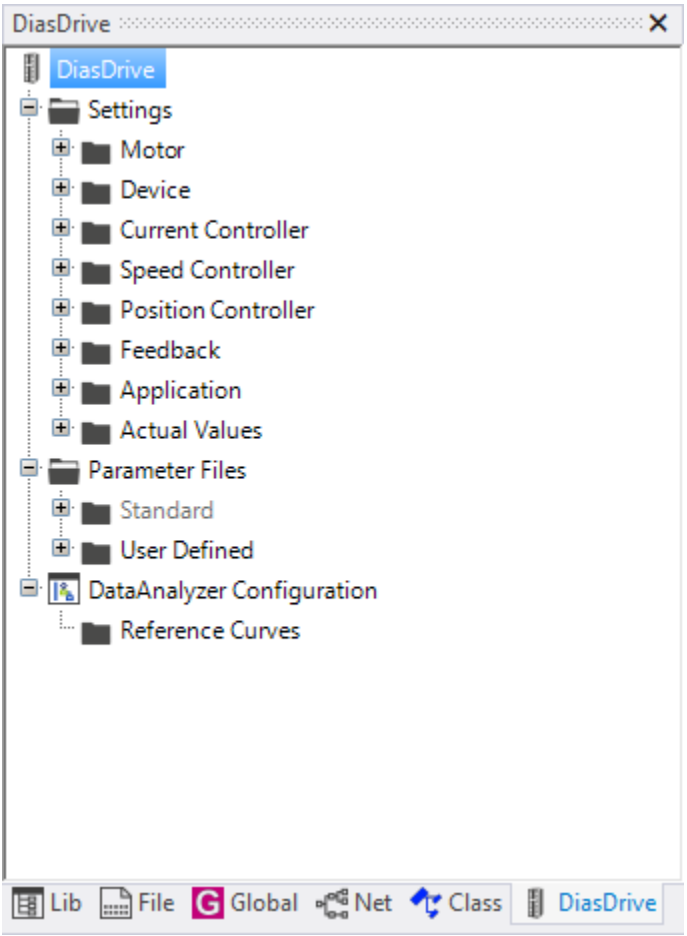
See also:

DiasDrive Tree

DiasDrive Config and DiasDrive Debug Window (Overview)

4.4.2.2 DiasDrive Tree

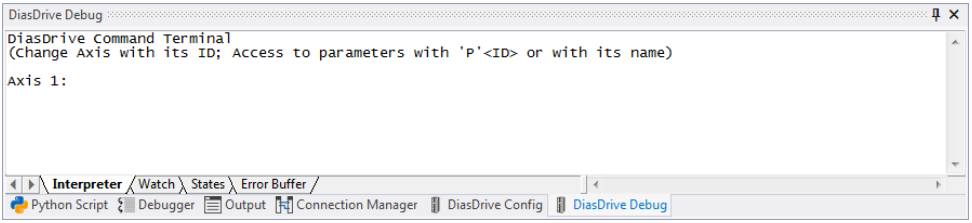
This tree includes an overview of the configuration: groupwise organized parameters, pre-configured parameter files and DataAnalyzer for the DiasDrive.



4.4.2.3 DiasDrive Config and DiasDrive Debug Window (Overview)

Both DiasDrive windows contain additional windows (Interpreter, Watch, States, Error Buffer, Controller Setup Error Configuration, Motor Configuration, Device Configuration und Comm. Setup) to monitior and configure individual parameters or parameter groups.

DiasDrive Debug Window



The description of the individual fields / functions can be found in the online help: **Most fields / variables are explained in ToolTips or in the integrated Motion help when selected and F1 is pressed.**

The individual register tabs:

Interpreter

In this window, the parameters are queried or set when commands are entered. See Interpreter Window.

Watch

Here, parameters can be set for better monitoring. See Watch Window.

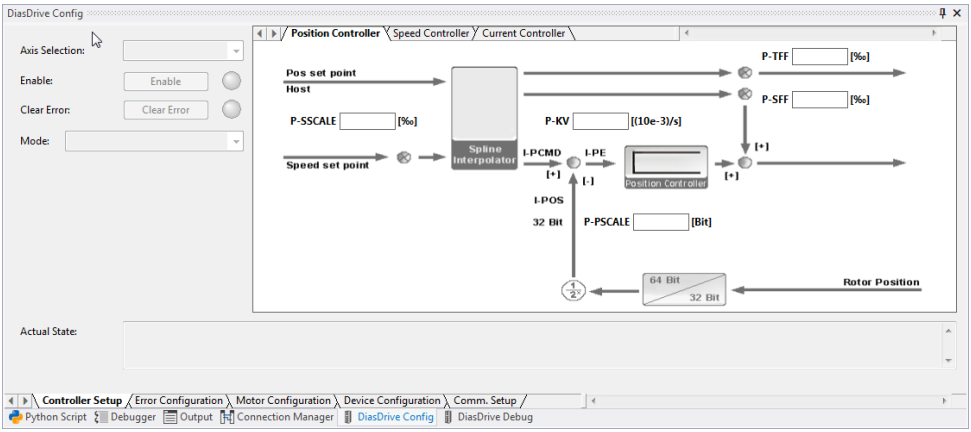
States

Overview of the status for all axes of a drive. See State Window.

Error Buffer

Displays error buffer See Error Buffer.

Dias Drive Config Window



The description of the individual fields / functions can be found in the online help: **Most fields / variables are explained in ToolTips or in the integrated Motion help when selected and F1 is pressed.**

The individual register tabs:

Controller Setup

Used to set the individual controllers of the DiasDrives. See Controller Setup Window.

Error Configuration

Determines the servo response in the event of an error. See Error Configuration Window.

Motor Configuration

Used to set motor parameters. See Motor Configuration Window.

Device Configuration

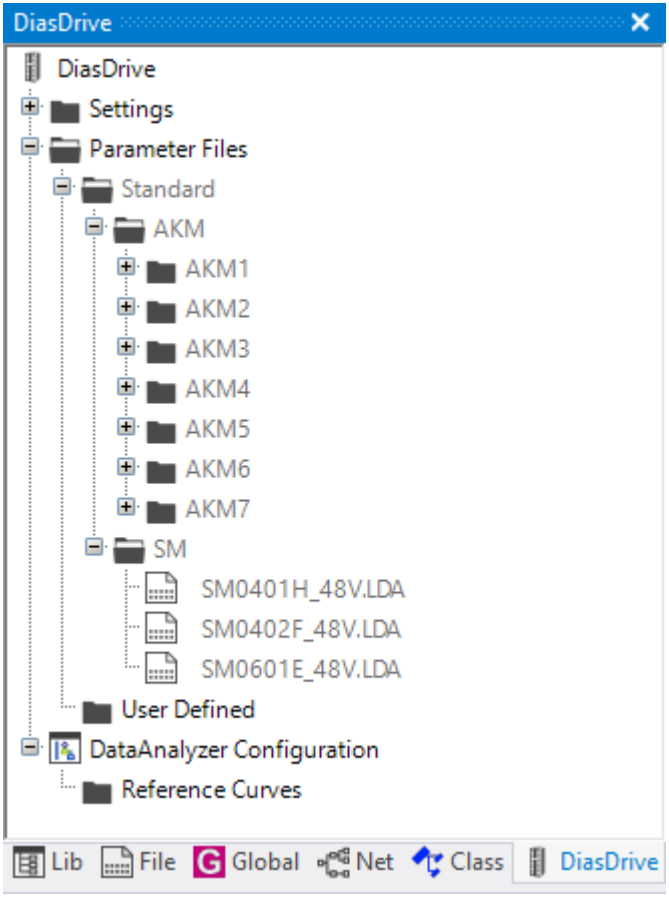
Used to set motor parameters. See Device Configuration Window.

Comm. (Communication) Setup

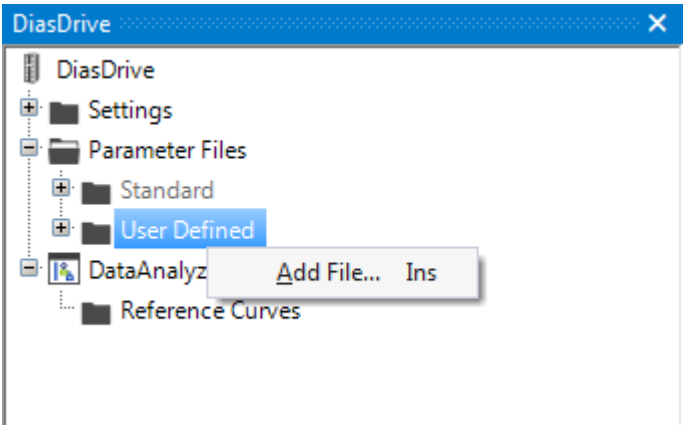
Used to set the communication parameters. See Comm.(unication) Setup Window.

4.4.3 Parameter Files Processing

Standardized LASAL CLASS 2 is delivered with numerous pre-configured parameter files. These files can be seen under **Parameter Files – Standard** in the DiasDrive tree.



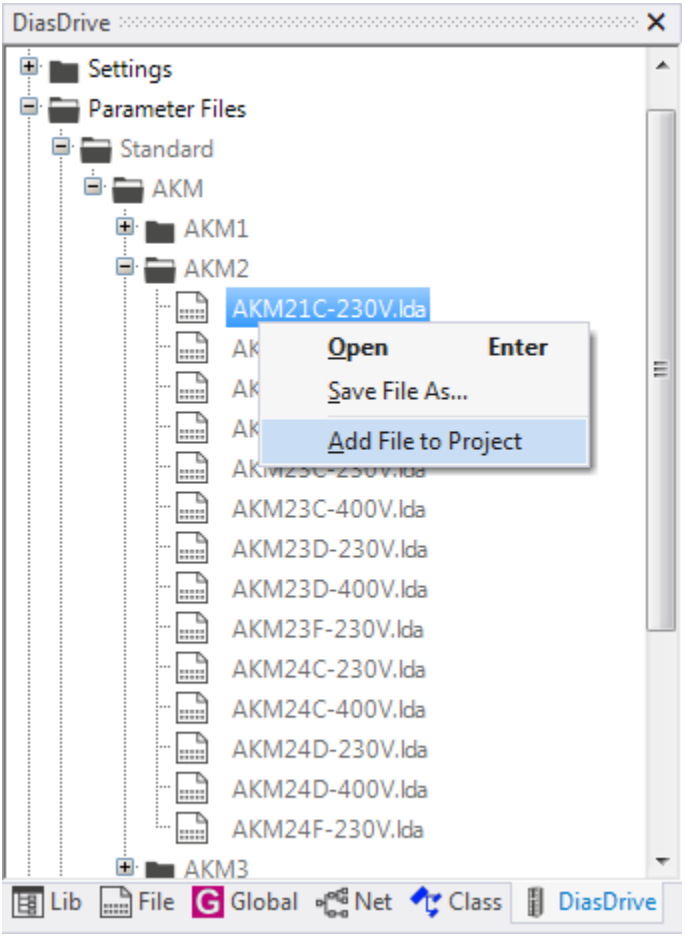
User files can be stored in the **User Defined** folder. A Drive file can be added through selecting **Add File...** in the context menu of the folder.



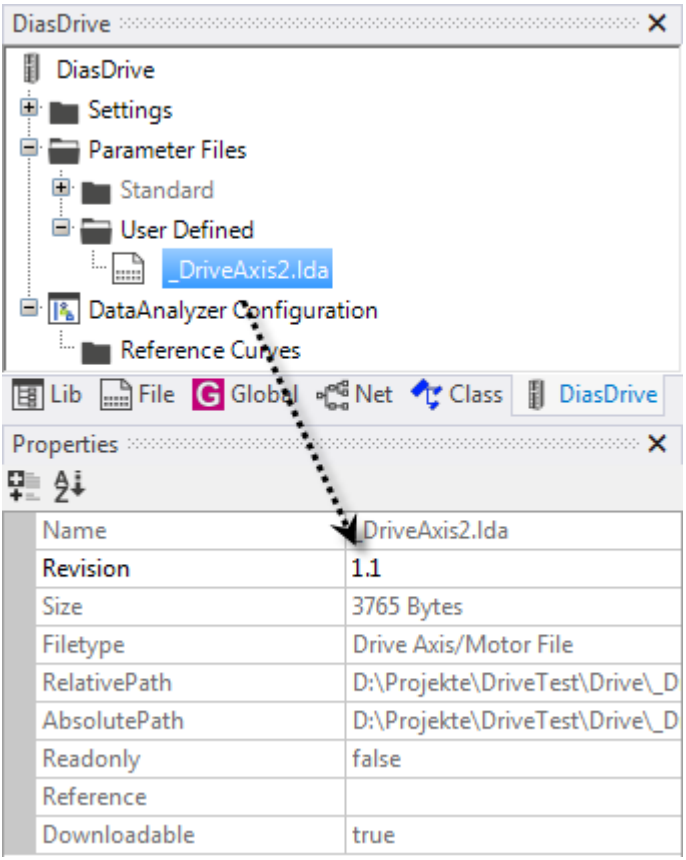
With a double click on the file or via the context menu entry **Open** the file can be opened with an editor.

Files listed in the **Standard** folder can only be opened write-protected. A file in this folder can be stored to the User Defined folder by selecting the context menu entry **Save File As…**. The file can be edited afterwards.

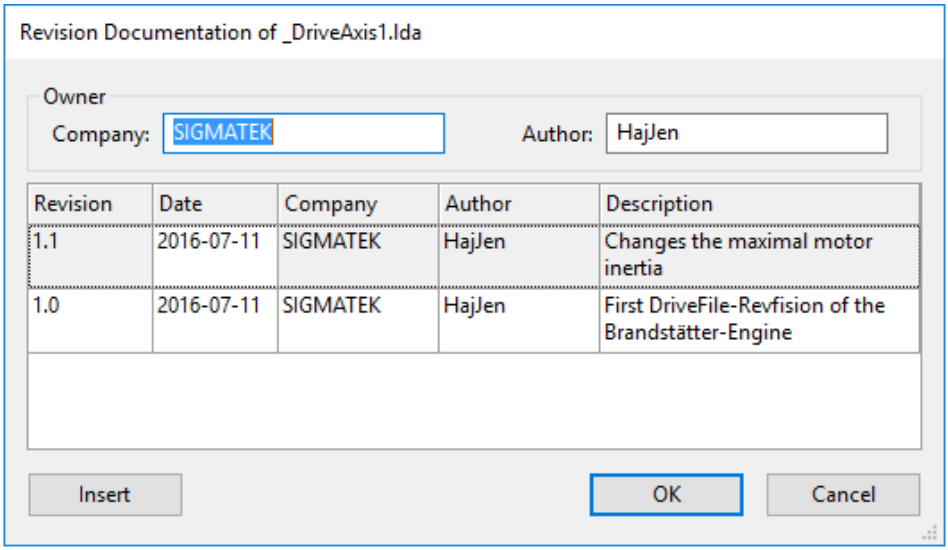
If a LASAL CLASS 2 project is loaded, a parameter file can be added to it via the context menu entry **Add File to Project**. Afterwards the parameter file, of course, can also be edited.



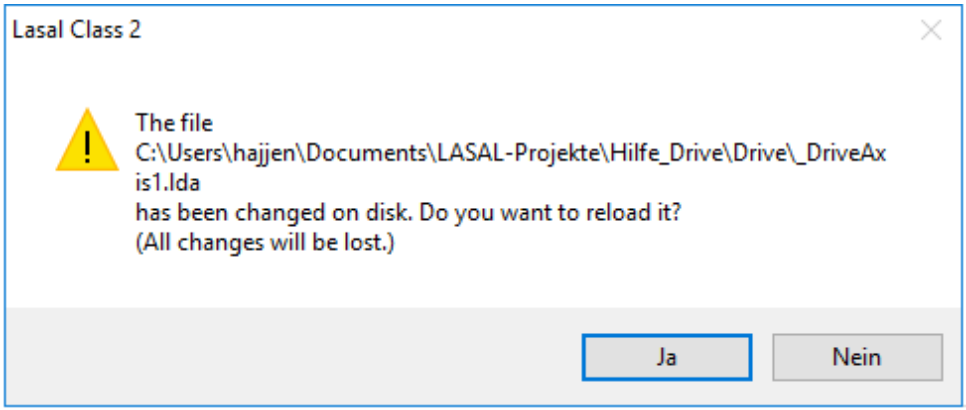
Similer to classes, revision documentation can be maintained for the drive file. If the file is selected, the revision number is included as an editiable property in the Properties window.



With the More button ( ) the revision documentation can be managed in this field:



If a file is changed externally (outside of LASAL CLASS 2), it is automatically loaded at reactivation of the editor. If modifications have already been made, a request appears asking whether the changes should be deleted or not.

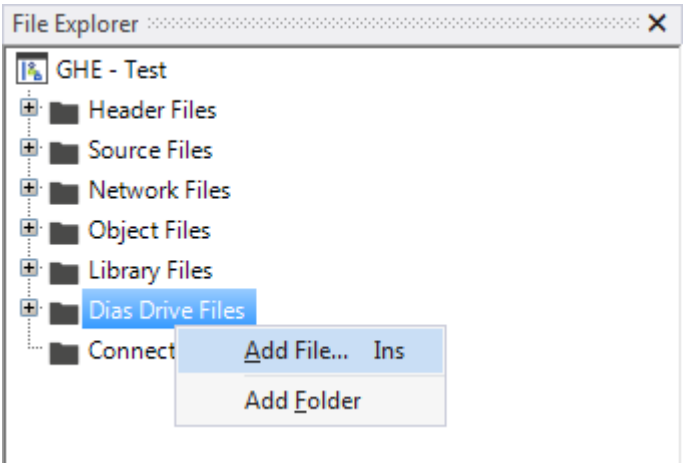


Deletion of a file in the **User Defined** folder only causes removal from the list but not from the hard disk.

Through the menu entry **File, Print...** a window with a page preview can be opened in order to print a file. In this window it can be set, which columns and parameter groups should be printed (only parameters can be printed, which are existent in the corresponding file).

4.4.4 Implementation of Parameter Files to a LASAL Class 2 Project

Via the project tree File existing Drive parameter files can be added to LASAL CLASS 2 projects. Thereby these files are copied into the particular project directory and can be corrected if necessary, without any influence to the master file. A file can be added through the context menu entry **Add File…** under **Dias Drive Files** in the project tree (**View – Trees – File Explorer**).



With Add Folder, user folders – as in many other sections in the LASAL tree – can be created and the parameter files grouped.

Also through dragging from the DiasDrive tree to this folder files can be added. Furthermore via selecting **Add File to Project** in the context menu of the parameter file, it can be added to the project.

The adding of files and therefore the change of the project is only possible when no active connection to the control exists (offline mode).

A file contains information of exactly one axis of the Drive. If more axes should be configured then more files are attached. If the same file is selected repeatedly, so an index (e.g. **MainEngine_0**) is attached to the file added to the project.

The order of the files corresponds to the order of the axes. To change the order a file can be shifted within the project folder via the context menu or via Drag & Drop.

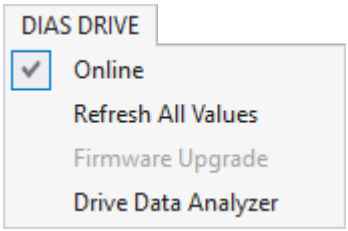
During the building process (**Build**) a StructuredText table is created from the information of each parameter file. This StructuredText table can then be accessed from the project. The table name is put together of **DRIVETAB** and a two-digit index number (Example: DRIVETAB00, DRIVETAB01, ...).

Alternative, at each parameter file in the project an object of the **SDDIntfc** class (or derived from it) and an axis can be assigned via the properties window. Afterwards the table name consists of **DRIVETAB_**, the name of the assigned object (whereby at complex classes the structure tags are replaced by underlines) and a two-digit axis identification (Example: Object name "SDDINTFC1" Axis "Axle2" - DRIVETAB_SDDINTFC1_02).

If the LASAL CLASS 2 project is sent to the control, these automatically created tables are also sent.

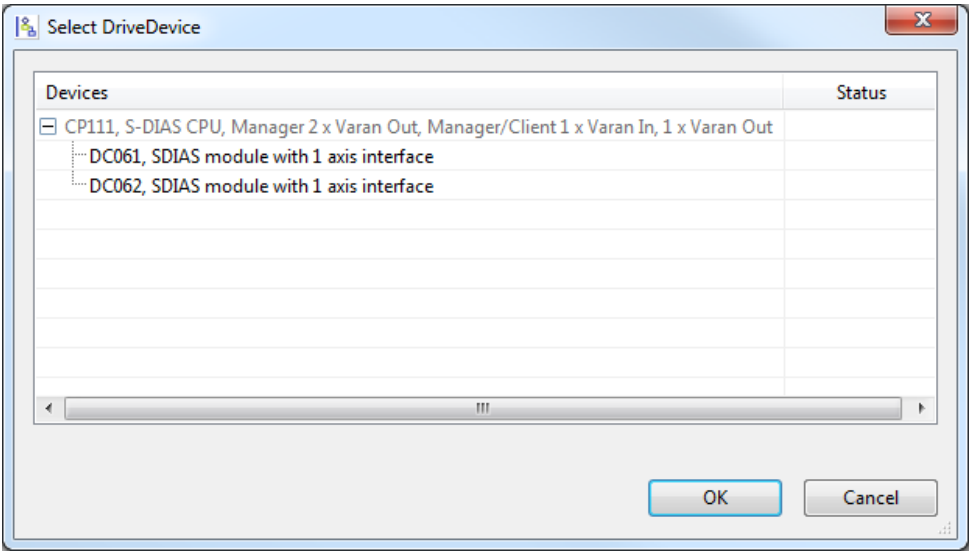
4.4.5 Online Configuration

To build-up a VARAN connection with a DiasDrive, it is necessary to connect to a so-called "VARAN Master" (C-IPC, LARS, ...) via the online dialog. Afterwards a dialog, listing a selection of possible target devices, can be opened via the menu **DiasDrive, Online**.

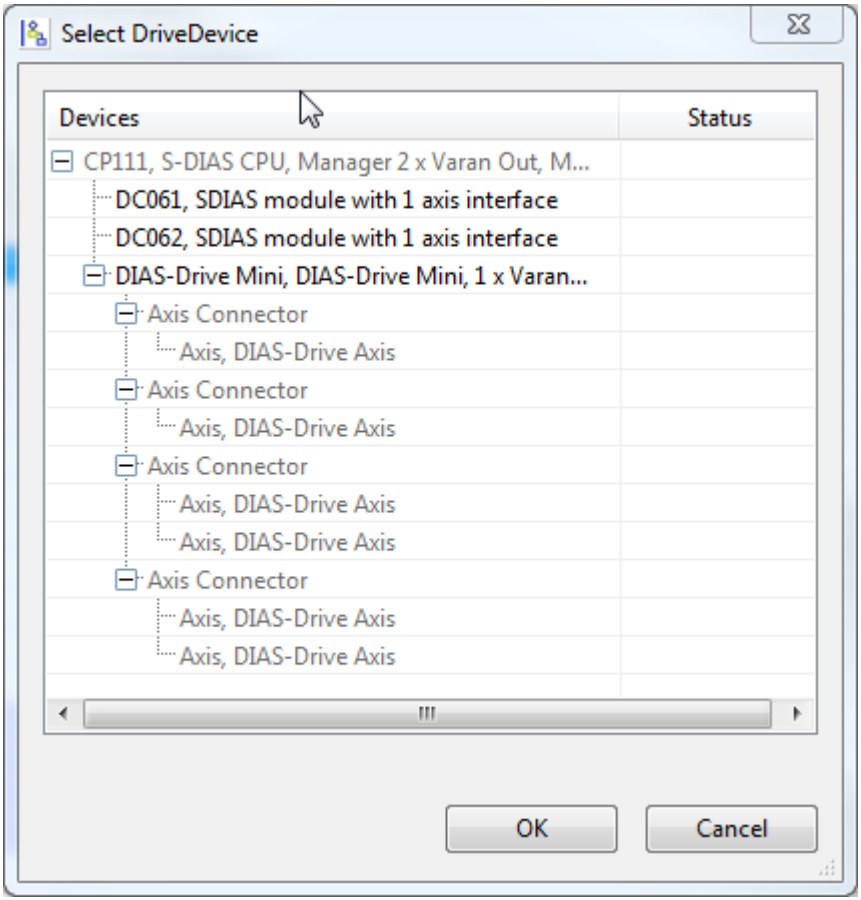


The connection to the DiasDrive is canceled if the **VARAN Master CPU** is in Run mode (Run-RAM, Run-ROM, SingleStep, …).

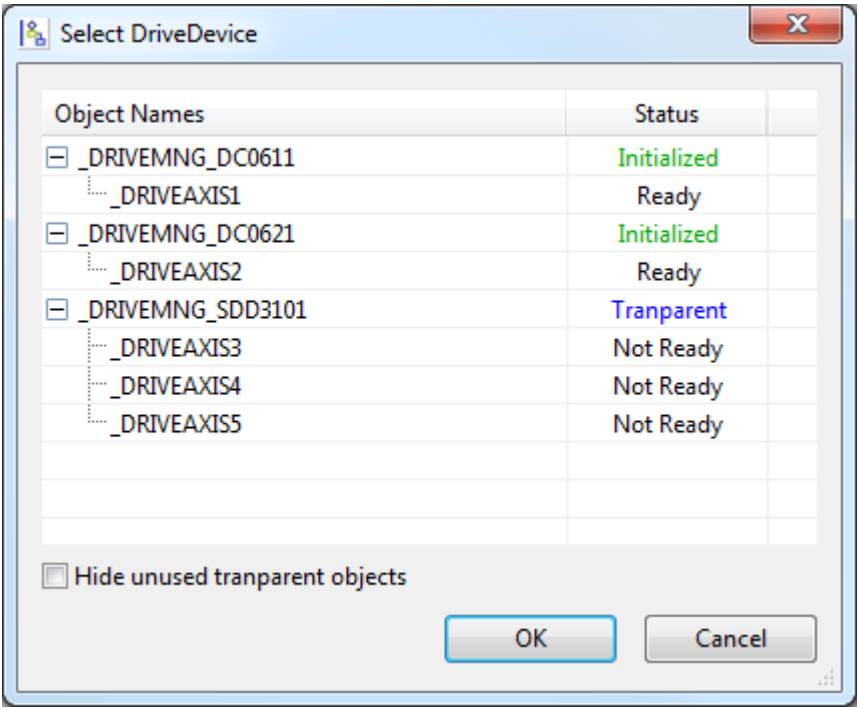
After selecting a valid target device and closing the dialog with OK, in the **DiasDrive Edit window** an additional column for axes is inserted and the parameter values (displayed in green) are uploaded automatically. If the control is not in one of the RUN modes, the modules are displayed:



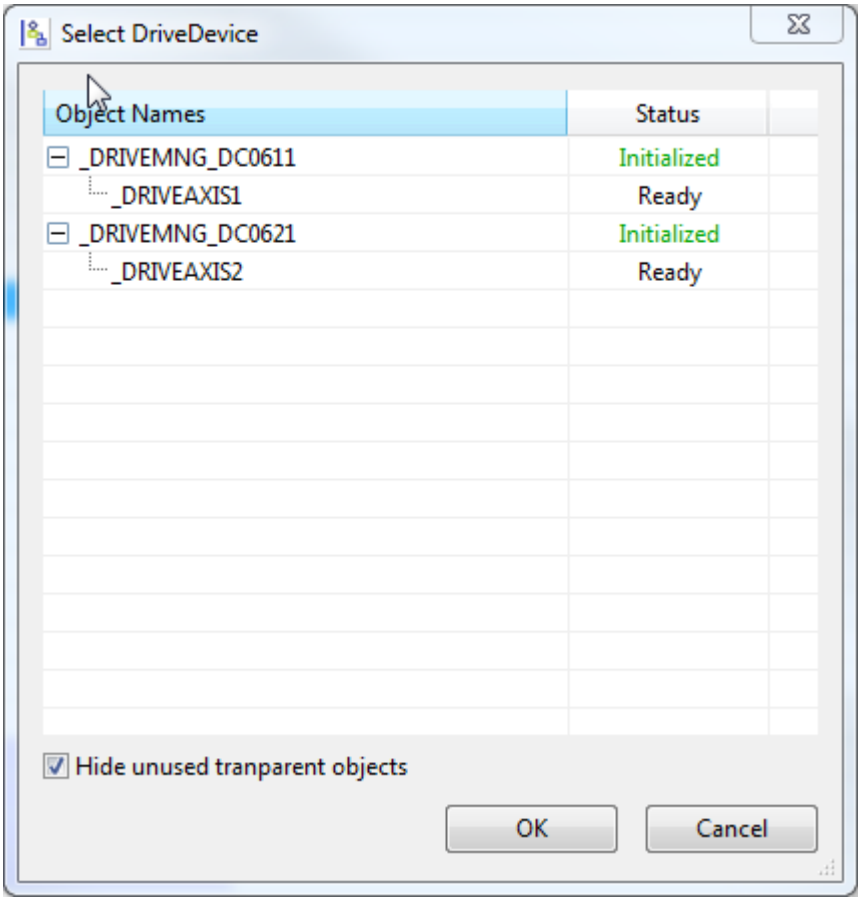
(Starting with the operating system LASAL OS 01.03.010, all axes are also displayed):



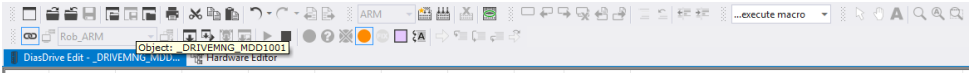
If however, one of the RUN modes is active, the status of the individual objects is shown, as well as the respective axes with whicn an online connection is possible::



If the checkbox **Hide unused transparent objects** is set, all transparent objects that are not used are hidden.



Transparent objects are only recognized as such since revision 1.41 of the class _DrvMngBase!

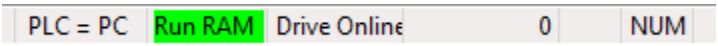


No.	ASCII-Name	Axis1	Axis2	Axis3	Axis4	Axis5	Axis6	Unit	Range	Shortdescription
P01	M-NAME1									Beginning 4 Characters of the Motor Name
P02	M-NAME2									Middle 4 Characters of the Motor Name
P03	M-NAME3									Last 4 Characters of the Motor Name
P04	M-INULL	5000	5000	5000	5000	5000	5000	mA	0 .. 50000	Stall Current of the Motor (rms)
P05	M-IPEAK	10000	10000	10000	10000	10000	10000	mA	0 .. 100000	Maximum Peak Current of the Motor (rms)
P06	M-NMAX	3000	3000	3000	3000	3000	3000	rpm	0 .. 12000	Maximum Mechanical Speed of the Motor
P07	M-POL	6	6	6	6	6	6		-256 .. ,-4,...	Number of Motor Poles
P08	M-TORQUE	1700	1700	1700	1700	1700	1700	mNm/A	0 .. 10000	Torque Constant of the Motor
P09	M-L	1000	1000	1000	1000	1000	1000	µH	0 .. 100000	Inductance of the Motor Winding (Phase - Phase)
P10	M-R	1000	1000	1000	1000	1000	1000	mΩ	0 .. 100000	Resistance of the Motor Winding (Phase - Phase)
P11	M-J	10	10	10	10	10	10	g cm²	0 .. 5000000	Inertia of the Motor
P12	M-TYPE	0	0	0	0	0	0		0 .. 0x405	Motor and Feedback Type
P13	M-RPOL	2	2	2	2	2	2		2, 4, 6, .. 32	Number of Resolver Poles
P14	M-ROFF	0	0	0	0	0	0	°	0 .. 360	Feedback Offset
P15	M-RPULSE	1024	1024	1024	1024	1024	1024		10 .. 65536	Number of Feedback Pulses
P16	M-RTEMP	1500	1500	1500	1500	1500	1500	Ω	0 .. 2500	Threshold for Overheating the Motor
P17	M-SER	0	0	0	0	0	0		-2^31 .. 2...	Serial Number of the Motor
P18	M-BRAKE	0	0	0	0	0	0		0, 1	Holding Brake Control
P21	M-IFWEAK	0	0	0	0	0	0	mA	0 .. 10000	Maximum field weakening current
P22	M-BRDIS	10	10	10	10	10	10	ms	0 .. 1000	Disable Delay Time of the Holding Brake
P23	M-BREN	100	100	100	100	100	100	ms	0 .. 1000	Enable Delay Time of the Holding Brake
P24	M-UN	0	0	0	0	0	0	V	0 .. 1000	Nominal Voltage Induction Motor
P25	M-UBOOST	0	0	0	0	0	0	V	0 .. 200	Boost Voltage Induction Motor
P26	M-IMAG	0	0	0	0	0	0	mA	0 .. 50000	Magnetic flux reference current Induction motor
P27	M-TROT	20	20	20	20	20	20	ms	10 .. 2000	Magnetic flux time constant Induction motor
P32	G-MODE	3	3	3	3	3	3		-4, -3, .. 2, 3	Working Mode
P33	G-CETIME	50	50	50	50	50	50	ms	0 .. 1000	Commutation Error Delay Time
P34	G-VMAINS	230	230	230	230	230	230	V	120, 230, 4...	Setting of the Mains Voltage
P35	G-VBUSM	100	100	100	100	100	100	V	0 .. 500	DC-Link Undervoltage Threshold
P36	G-MBAL	0	0	0	0	0	0	W	0 .. 2000	Max. Power of the Regen Resistor
P37	G-MTEMPK	85	85	85	85	85	85	°C	0 .. 85	Maximum Heat Sink Temperature
P38	G-MTEMPE	85	85	85	85	85	85	°C	0 .. 85	Maximum Internal Temperature

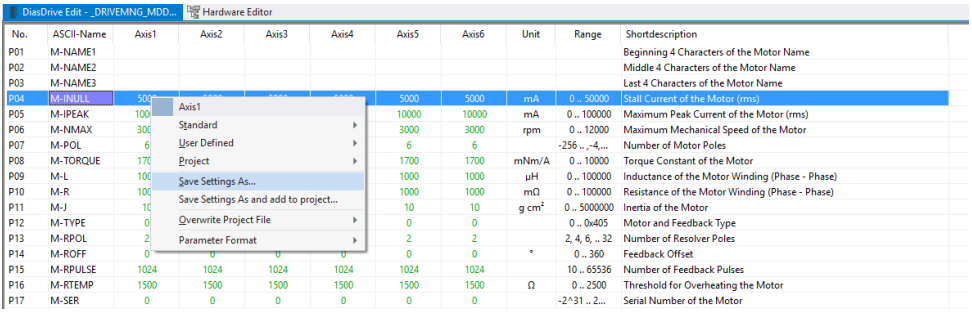
The name of the selected target device is shown on the tab as well as in a Tooltip bubble when the mouse pointer placed on it.

If values cannot be loaded three question marks (???) are displayed. These question marks also appear when the connection to the DiasDrive is canceled.

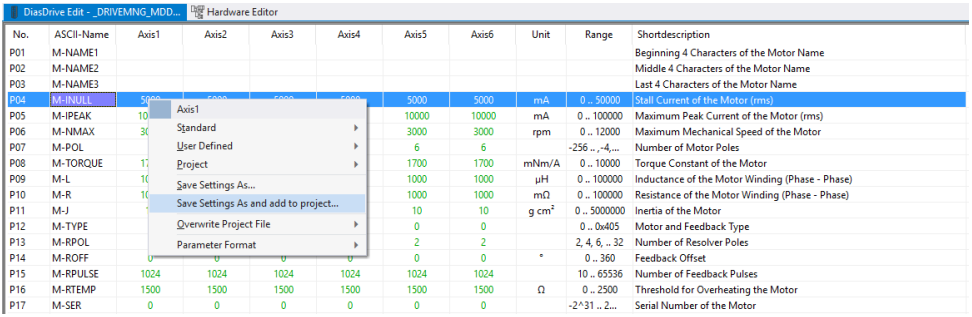
The current state of the connection (Online, Offline) can be seen in the status line.



Select a value to change it. Press F2 (Enter or double-click) to enter a new value. Afterwards the new value is sent to the Drive immediately. To save the set data of an axis to a file, select **Save settings as** and **Save Settings As and add to project...** from the context menu of the axis column.

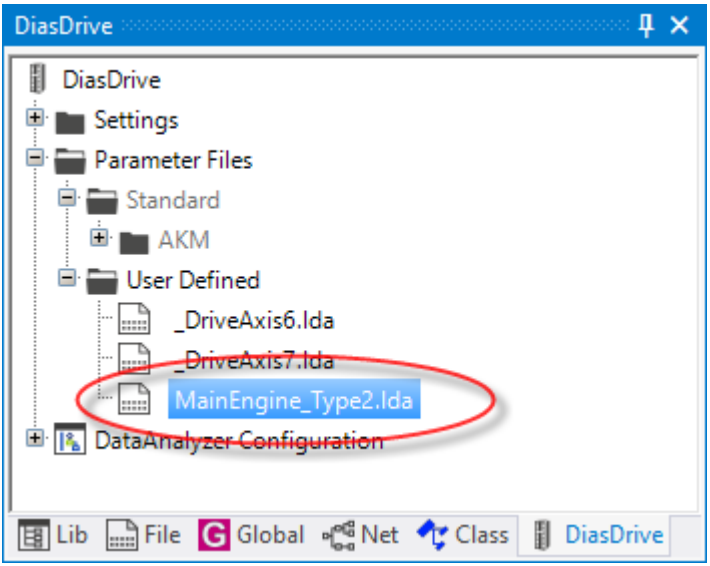


No.	ASCII-Name	Axis1	Axis2	Axis3	Axis4	Axis5	Axis6	Unit	Range	Shortdescription
P01	M-NAME1									Beginning 4 Characters of the Motor Name
P02	M-NAME2									Middle 4 Characters of the Motor Name
P03	M-NAME3									Last 4 Characters of the Motor Name
P04	M-INULL	5000				5000	5000	mA	0 .. 50000	Stall Current of the Motor (rms)
P05	M-IPEAK	100				10000	10000	mA	0 .. 100000	Maximum Peak Current of the Motor (rms)
P06	M-NMAX	300				3000	3000	rpm	0 .. 12000	Maximum Mechanical Speed of the Motor
P07	M-POL	6				6	6		-256 .. -4,...	Number of Motor Poles
P08	M-TORQUE	170				1700	1700	mNm/A	0 .. 10000	Torque Constant of the Motor
P09	M-L	100				1000	1000	µH	0 .. 100000	Inductance of the Motor Winding (Phase - Phase)
P10	M-R	100				1000	1000	mΩ	0 .. 100000	Resistance of the Motor Winding (Phase - Phase)
P11	M-J	10				10	10	g cm²	0 .. 5000000	Inertia of the Motor
P12	M-TYPE	0				0	0		0 .. 0x405	Motor and Feedback Type
P13	M-RPOL	2				2	2		2, 4, 6, .. 32	Number of Resolver Poles
P14	M-ROFF	0	0	0	0	0	0	°	0 .. 360	Feedback Offset
P15	M-RPULSE	1024	1024	1024	1024	1024	1024		10 .. 65536	Number of Feedback Pulses
P16	M-RTEMP	1500	1500	1500	1500	1500	1500	Ω	0 .. 2500	Threshold for Overheating the Motor
P17	M-SER	0	0	0	0	0	0		-2^31 .. 2...	Serial Number of the Motor

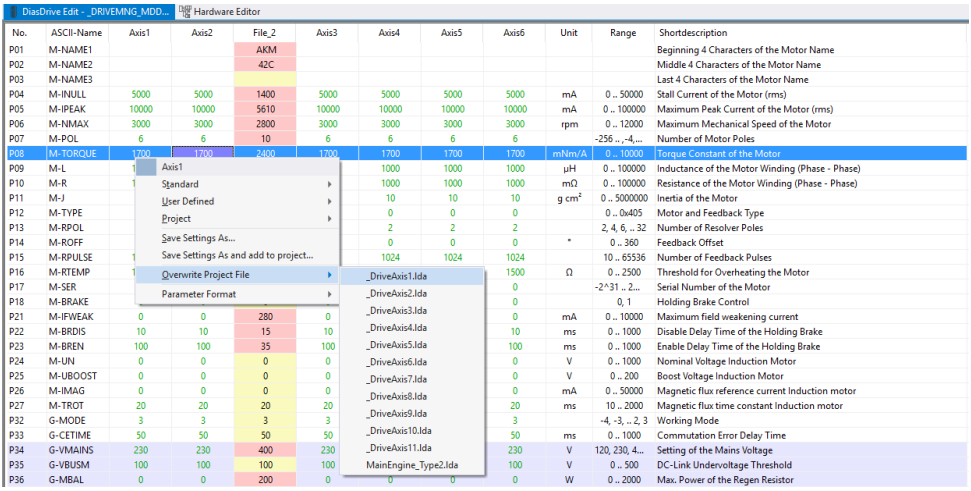


No.	ASCII-Name	Axis1	Axis2	Axis3	Axis4	Axis5	Axis6	Unit	Range	Shortdescription
P01	M-NAME1									Beginning 4 Characters of the Motor Name
P02	M-NAME2									Middle 4 Characters of the Motor Name
P03	M-NAME3									Last 4 Characters of the Motor Name
P04	M-INULL	5000				5000	5000	mA	0 .. 50000	Stall Current of the Motor (rms)
P05	M-IPEAK	10				10000	10000	mA	0 .. 100000	Maximum Peak Current of the Motor (rms)
P06	M-NMAX	30				3000	3000	rpm	0 .. 12000	Maximum Mechanical Speed of the Motor
P07	M-POL					6	6		-256 .. -4,...	Number of Motor Poles
P08	M-TORQUE	17				1700	1700	mNm/A	0 .. 10000	Torque Constant of the Motor
P09	M-L	10				1000	1000	µH	0 .. 100000	Inductance of the Motor Winding (Phase - Phase)
P10	M-R	10				1000	1000	mΩ	0 .. 100000	Resistance of the Motor Winding (Phase - Phase)
P11	M-J	1				10	10	g cm²	0 .. 5000000	Inertia of the Motor
P12	M-TYPE					0	0		0 .. 0x405	Motor and Feedback Type
P13	M-RPOL					2	2		2, 4, 6, .. 32	Number of Resolver Poles
P14	M-ROFF	0	0	0	0	0	0	°	0 .. 360	Feedback Offset
P15	M-RPULSE	1024	1024	1024	1024	1024	1024		10 .. 65536	Number of Feedback Pulses
P16	M-RTEMP	1500	1500	1500	1500	1500	1500	Ω	0 .. 2500	Threshold for Overheating the Motor
P17	M-SER	0	0	0	0	0	0		-2^31 .. 2...	Serial Number of the Motor

In the following dialog the position and the name of the new parameter file can be selected (here: **MainEngine_Type2**). After saving process this file is automatically arranged to the **User Defined files** in the DiasDrive tree.



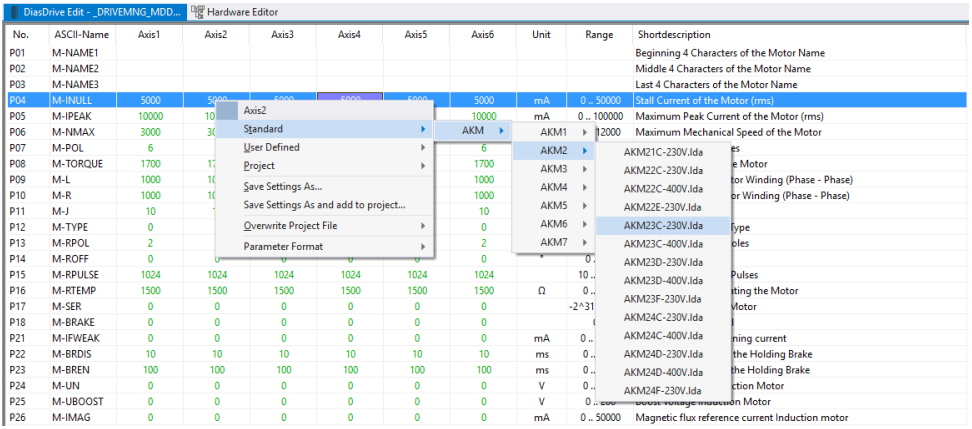
With the **Overwrite Project File** item in the context menu, changed parameter values in an existing parameter (here: **Axle1.lda**) can be overwritten.



No.	ASCII-Name	Axis1	Axis2	File_2	Axis3	Axis4	Axis5	Axis6	Unit	Range	Shortdescription
P01	M-NAME1			AKM							Beginning 4 Characters of the Motor Name
P02	M-NAME2			42C							Middle 4 Characters of the Motor Name
P03	M-NAME3										Last 4 Characters of the Motor Name
P04	M-INULL	5000	5000	1400	5000	5000	5000	5000	mA	0 .. 50000	Stall Current of the Motor (rms)
P05	M-IPEAK	10000	10000	5610	10000	10000	10000	10000	mA	0 .. 100000	Maximum Peak Current of the Motor (rms)
P06	M-NMAX	3000	3000	2800	3000	3000	3000	3000	rpm	0 .. 12000	Maximum Mechanical Speed of the Motor
P07	M-POL	6	6	10	6	6	6	6		-256 .. ,-4,...	Number of Motor Poles
P08	M-TORQUE	1700	1700	2400	1700	1700	1700	1700	mNm/A	0 .. 10000	Torque Constant of the Motor
P09	M-L	1				1000	1000	1000	µH	0 .. 100000	Inductance of the Motor Winding (Phase - Phase)
P10	M-R	1				1000	1000	1000	mΩ	0 .. 100000	Resistance of the Motor Winding (Phase - Phase)
P11	M-J					10	10	10	g cm²	0 .. 5000000	Inertia of the Motor
P12	M-TYPE					0	0	0		0 .. 0x405	Motor and Feedback Type
P13	M-RPOL					2	2	2		2, 4, 6, .. 32	Number of Resolver Poles
P14	M-ROFF					0	0	0	°	0 .. 360	Feedback Offset
P15	M-RPULSE	1				1024	1024	1024		10 .. 65536	Number of Feedback Pulses
P16	M-RTEMP	1						1500	Ω	0 .. 2500	Threshold for Overheating the Motor
P17	M-SER							0		-2^31 .. 2...	Serial Number of the Motor
P18	M-BRAKE							0		0, 1	Holding Brake Control
P21	M-IFWEAK	0	0	280	0			0	mA	0 .. 10000	Maximum field weakening current
P22	M-BRDIS	10	10	15	10			10	ms	0 .. 1000	Disable Delay Time of the Holding Brake
P23	M-BREN	100	100	35	100			100	ms	0 .. 1000	Enable Delay Time of the Holding Brake
P24	M-UN	0	0	0	0			0	V	0 .. 1000	Nominal Voltage Induction Motor
P25	M-UBOOST	0	0	0	0			0	V	0 .. 200	Boost Voltage Induction Motor
P26	M-IMAG	0	0	0	0			0	mA	0 .. 50000	Magnetic flux reference current Induction motor
P27	M-TROT	20	20	20	20			20	ms	10 .. 2000	Magnetic flux time constant Induction motor
P32	G-MODE	3	3	3	3			3		-4, -3, .. 2, 3	Working Mode
P33	G-CETIME	50	50	50	50			50	ms	0 .. 1000	Commutation Error Delay Time
P34	G-VMAINS	230	230	400	230			230	V	120, 230, 4...	Setting of the Mains Voltage
P35	G-VBUSM	100	100	100	100			100	V	0 .. 500	DC-Link Undervoltage Threshold
P36	G-MBAL	0	0	200	0	0	0	0	W	0 .. 2000	Max. Power of the Regen Resistor

In addition to individual parameters, parameters from files can also be sent to the drive. For this purpose, a file is selected from the axis column of the context menu (here: Axle2),

which appears in a column with a yellow background to the right of the axis (here : File_2).



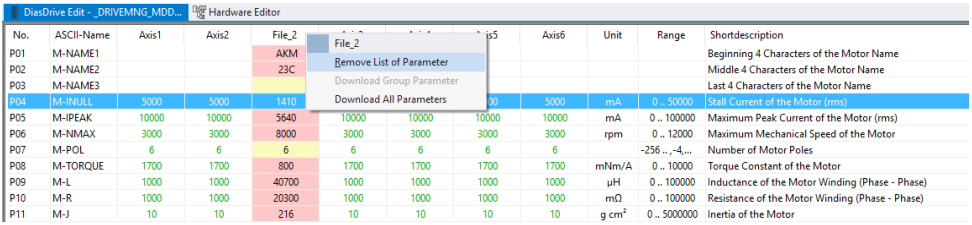
No.	ASCII-Name	Axis1	Axis2	Axis3	Axis4	Axis5	Axis6	Unit	Range	Shortdescription
P01	M-NAME1									Beginning 4 Characters of the Motor Name
P02	M-NAME2									Middle 4 Characters of the Motor Name
P03	M-NAME3									Last 4 Characters of the Motor Name
P04	M-INULL	5000	5000	5000	5000	5000	5000	mA	0 .. 50000	Stall Current of the Motor (rms)
P05	M-IPEAK	10000	10				10000	mA	0 .. 100000	Maximum Peak Current of the Motor (rms)
P06	M-NMAX	3000	3						12000	Maximum Mechanical Speed of the Motor
P07	M-POL	6					6			es
P08	M-TORQUE	1700	17				1700			e Motor
P09	M-L	1000	10				1000			tor Winding (Phase - Phase)
P10	M-R	1000	10				1000			or Winding (Phase - Phase)
P11	M-J	10					10			
P12	M-TYPE	0					0			Type
P13	M-RPOL	2					2			oles
P14	M-ROFF	0	0	0	0	0	0		0	
P15	M-RPULSE	1024	1024	1024	1024	1024	1024		10 ..	Pulses
P16	M-RTEMP	1500	1500	1500	1500	1500	1500	Ω	0 ..	ting the Motor
P17	M-SER	0	0	0	0	0	0		-2^31	Motor
P18	M-BRAKE	0	0	0	0	0	0		(	
P21	M-IFWEAK	0	0	0	0	0	0	mA	0 ..	ning current
P22	M-BRDIS	10	10	10	10	10	10	ms	0 ..	the Holding Brake
P23	M-BREN	100	100	100	100	100	100	ms	0 ..	the Holding Brake
P24	M-UN	0	0	0	0	0	0	V	0 ..	ction Motor
P25	M-UBOOST	0	0	0	0	0	0	V	0 .. 200	Boost Voltage Induction Motor
P26	M-IMAG	0	0	0	0	0	0	mA	0 .. 50000	Magnetic flux reference current Induction motor

Differences to the actual value are identified with a **red** background.

No.	ASCII-Name	Axis1	Axis2	File_2	Axis3
P01	M-NAME1			AKM	
P02	M-NAME2			23C	
P03	M-NAME3				
P04	M-INULL	5000	5000	1410	5000
P05	M-IPEAK	10000	10000	5640	10000
P06	M-NMAX	3000	3000	8000	3000
P07	M-POL	6	6	6	6
P08	M-TORQUE	1700	1700	800	1700
P09	M-L	1000	1000	40700	1000
P10	M-R	1000	1000	20300	1000
P11	M-J	10	10	216	10
P12	M-TYPE	0	0	0	0
P13	M-RPOL	2	2	2	2
P14	M-ROFF	0	0	0	0
P15	M-RPULSE	1024	1024	1024	1024
P16	M-RTEMP	1500	1500	300	1500
P17	M-SER	0	0	0	0
P18	M-BRAKE	0	0	0	0
P21	M-IFWEAK	0	0	282	0
P22	M-BRDIS	10	10	18	10
P23	M-BREN	100	100	20	100
P24	M-UN	0	0	0	0
P25	M-UBOOST	0	0	0	0
P26	M-IMAG	0	0	0	0
P27	M-TROT	20	20	20	20
P32	G-MODE	3	3	3	3
P33	G-CETIME	50	50	50	50
P34	G-VMAINS	230	230	230	230
P35	G-VBUSM	100	100	100	100
P36	G-MBAL	0	0	66	0
P37	G-MTEMPK	85	85	85	85
P38	G-MTEMPE	85	85	85	85

If a specific group of parameters was selected from the DiasDrive tree, the context menu of the column can be used to specify whether to send only the parameters of the group or all parameters.

The context menu item Remove List of Parameter removes the column with the loaded parameters.



No.	ASCII-Name	Axis1	Axis2	File_2	Axis5	Axis6	Unit	Range	Shortdescription
P01	M-NAME1			AKM					Beginning 4 Characters of the Motor Name
P02	M-NAME2			23C					Middle 4 Characters of the Motor Name
P03	M-NAME3								Last 4 Characters of the Motor Name
P04	M-INULL	5000	5000	1410	5000	5000	mA	0 .. 50000	Stall Current of the Motor (rms)
P05	M-IPEAK	10000	10000	5640	10000	10000	mA	0 .. 100000	Maximum Peak Current of the Motor (rms)
P06	M-NMAX	3000	3000	8000	3000	3000	rpm	0 .. 12000	Maximum Mechanical Speed of the Motor
P07	M-POL	6	6	6	6	6		-256 .. -4,...	Number of Motor Poles
P08	M-TORQUE	1700	1700	800	1700	1700	mNm/A	0 .. 10000	Torque Constant of the Motor
P09	M-L	1000	1000	40700	1000	1000	µH	0 .. 100000	Inductance of the Motor Winding (Phase - Phase)
P10	M-R	1000	1000	20300	1000	1000	mΩ	0 .. 100000	Resistance of the Motor Winding (Phase - Phase)
P11	M-J	10	10	216	10	10	g cm²	0 .. 5000000	Inertia of the Motor

The values displayed can be printed over the menu item File, Print…. In the print dialog, which columns and groups to print can be selected. If only one group in the editor is selected, only this group can be printed.

4.4.5.1 Interpreter Window

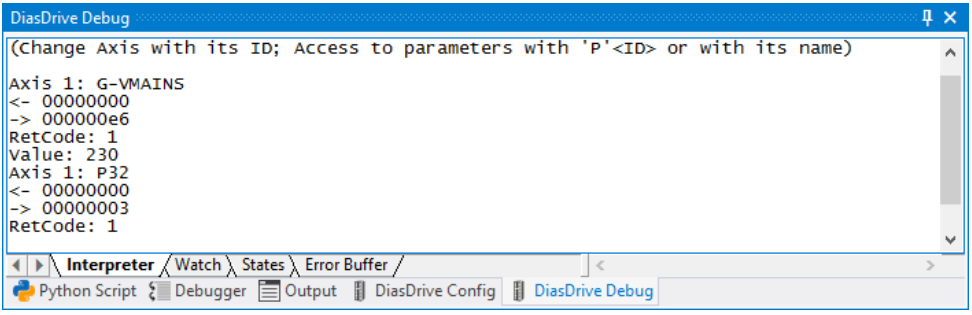
In this command-line-orientated window parameters can be used and set. Here also parameters can be used that are not displayed in the overview, because they are not necessary for the configuration process.

The parameters are selected either by parameter name (in G-VMAINS example below) or with 'P', followed by the parameter number (in the example it is 32 for G-MODE)

```
Axis 1: G-VMAINS
```

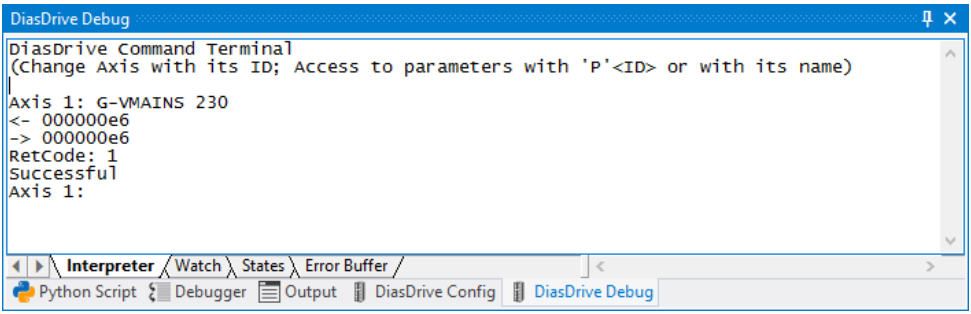
or

```
Axis 1: P32
```



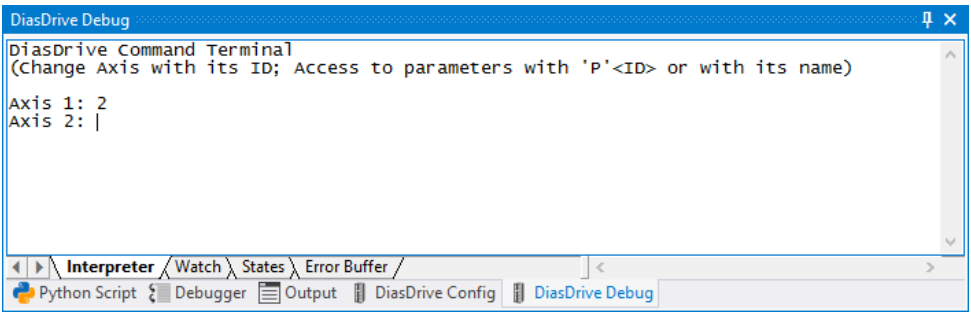
If a value should be written, it is written after a space:

```
Axis 1: G-VMAINS 230
```

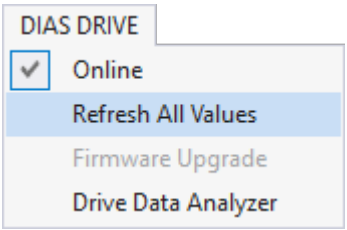


If an axis should be changed, the desired axis number is written:

```
Axis 1: 2
```

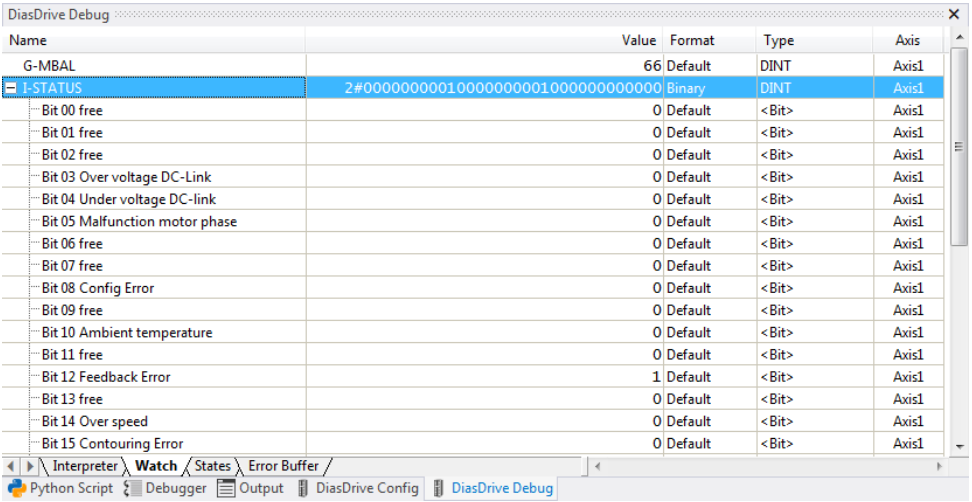


If a parameter has been changed that way, it will not be corrected automatically. Not until selection of the menu entry Dias Drive, Refresh All Values the changes are also corrected in all other views.



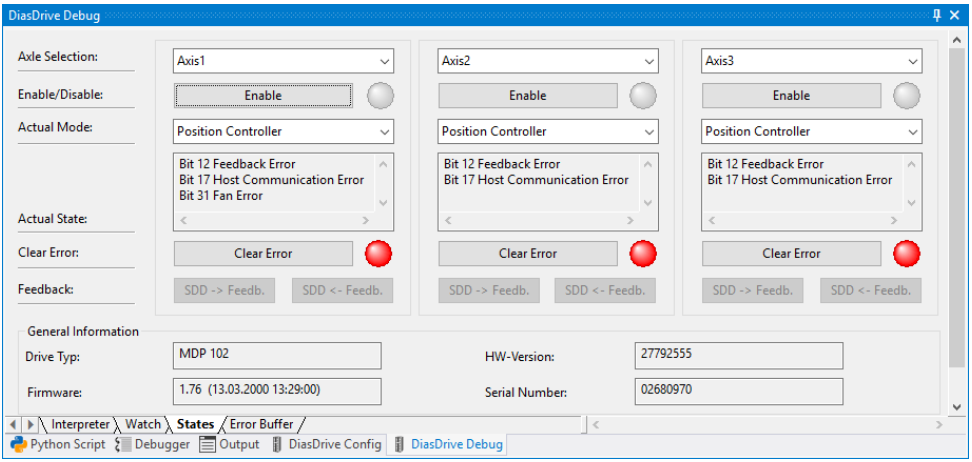
4.4.5.2 Watch Window

This window serves the specific surveillance of certain parameters. Parameters listed here are reread periodically by the Drive. Also the display format of the value can be changed in this view, in order to easier interpret values. Only few parameters, like I-STATUS or I-DERROR, can be displayed with the names of the single bits.



4.4.5.3 State Window

The State window provides an overview of the actual status of all possible axes and general information on the drive. For each axis, an enable button is available (Enable) as well as list with error messages, which can be cleared with an additional button (Clear Error). The actual axis control mode can then be converted.

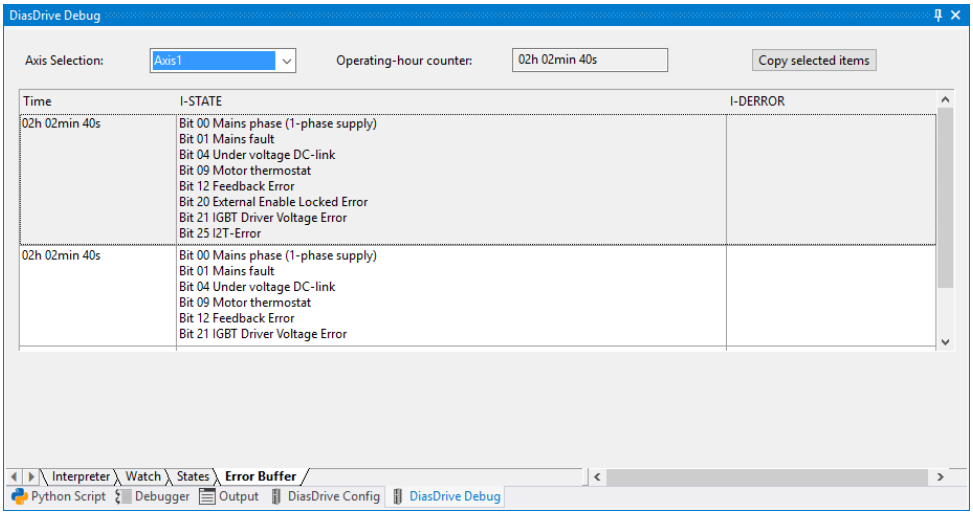


Up and download encoder settings

With the SDD -> Feedb. and SDD <- Feedb. the feedback settings can be up and downloaded into the encoder.

In the **"Serial Number"** field, either the serial number of the device or the head station is displayed.

4.4.5.4 Error Buffer

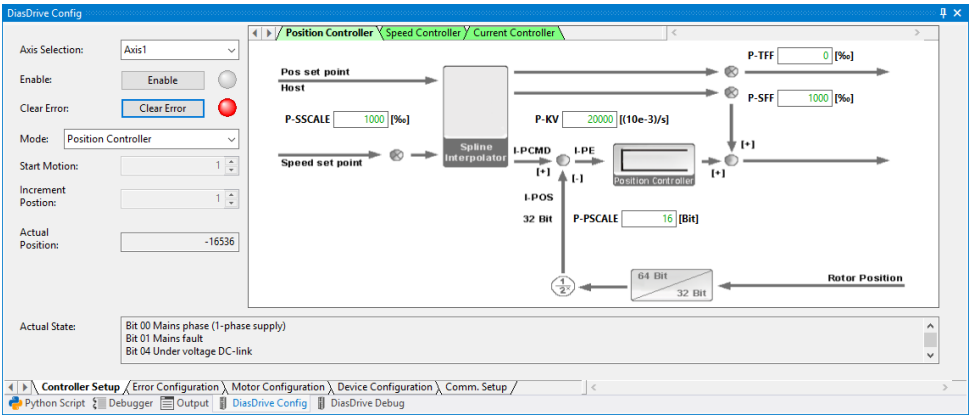


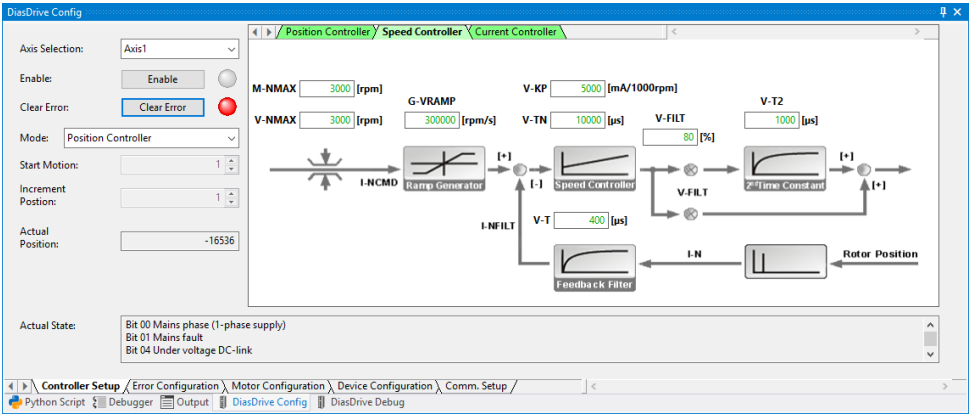
Here, the errors are stored. In the left column, the duration for which the drive has been running is entered. In the middle column is the motor status and in the right column, the internal drive error.

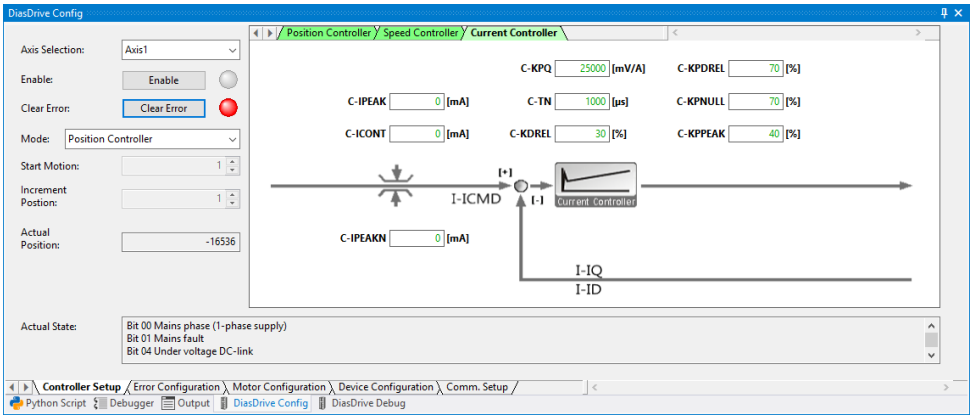
4.4.5.5 Controller Setup Window

Here parameters, axes and controllers can be set specifically. On the left side the axis to be controlled can be selected. Under it there is the axis-enabling button and a button to clear error messages, which are displayed in the bottom window. Under the **Clear Error button** the operation mode of the axis can be selected. Following there are parameters depending on the set operation mode. The switching of the controller view has no influence on the setpoint value parameter. A green coloration of the controller register shows the active controller in the operation mode set.

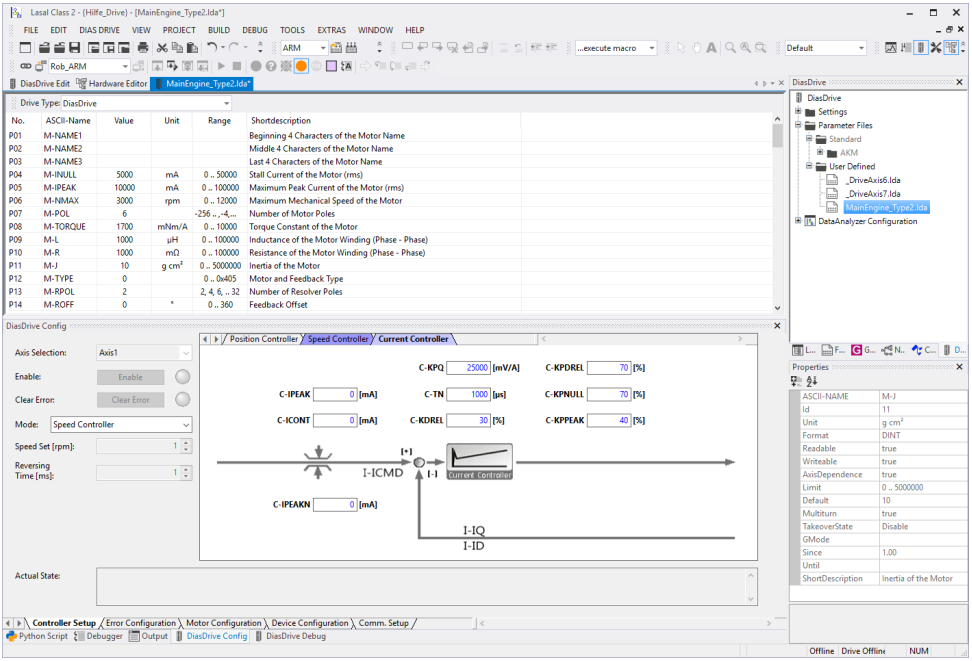
There are three sub-windows available: **Position Controller** (position limit), **Speed Controller** (speed limit) and **Current Controller** (current limit):


DiasDrive Config
Axis Selection:
Axis1
Enable:
Enable
Clear Error:
Clear Error
Mode:
Position Controller
Start Motion:
Increment Postion:
Actual Position:
-16536
Position Controller
Speed Controller
Current Controller
Pos set point
Host
Speed set point
P-SSCALE
1000
[‰]
Spline Interpolator
I-PCMD
I-PE
I-POS
32 Bit
P-KV
20000
[(10e-3)/s]
P-TFF
0
[‰]
P-SFF
1000
[‰]
Position Controller
P-PSCALE
16
[Bit]
64 Bit
32 Bit
Rotor Position
Actual State:
Bit 00 Mains phase (1-phase supply)
Bit 01 Mains fault
Bit 04 Under voltage DC-link
Controller Setup
Error Configuration
Motor Configuration
Device Configuration
Comm. Setup
Python Script
Debugger
Output
DiasDrive Config
DiasDrive Debug

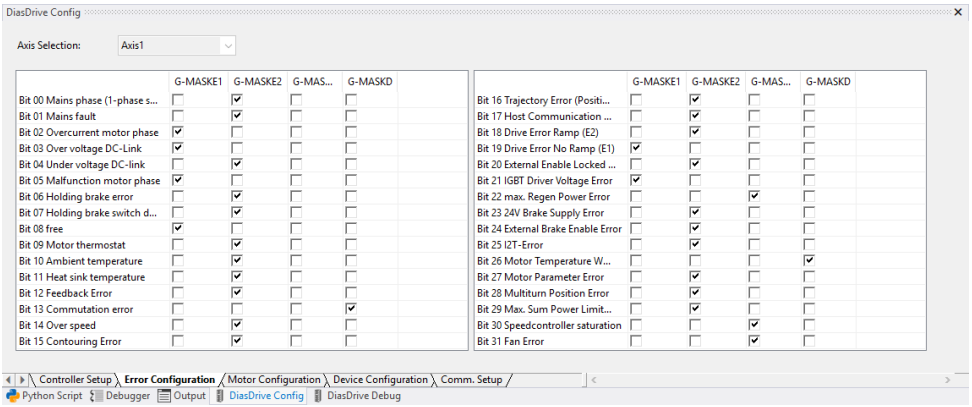

DiasDrive Config
Axis Selection:
Axis1
Enable:
Enable
Clear Error:
Clear Error
Mode:
Position Controller
Start Motion:
Increment Postion:
Actual Position:
-16536
Position Controller
Speed Controller
Current Controller
M-NMAX
3000
[rpm]
V-NMAX
3000
[rpm]
G-VRAMP
300000
[rpm/s]
V-KP
5000
[mA/1000rpm]
V-TN
10000
[µs]
V-FILT
80
[%]
V-T2
1000
[µs]
I-NCMD
Ramp Generator
Speed Controller
V-FILT
Z-Time Constant
I-NFILT
V-T
400
[µs]
I-N
Feedback Filter
Rotor Position
Actual State:
Bit 00 Mains phase (1-phase supply)
Bit 01 Mains fault
Bit 04 Under voltage DC-link
Controller Setup
Error Configuration
Motor Configuration
Device Configuration
Comm. Setup
Python Script
Debugger
Output
DiasDrive Config
DiasDrive Debug



The **green numbers** mean that the user is continuously online and the values are current. A Motor / Drive can also be configured **offline**; all **digits** are then shown in **blue** (gray in online mode). This can be done by opening a motor file, for example, in **DiasDrive => Parameter Files => User Defined**.



4.4.5.6 Error Configuration Window



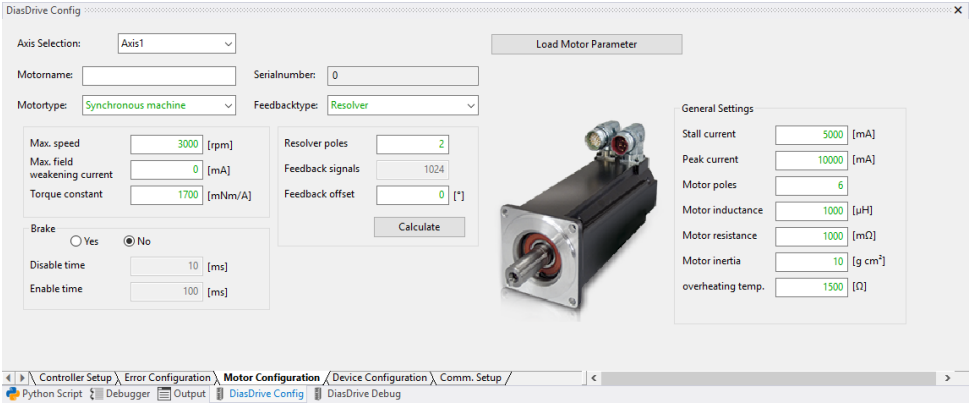
In this table the behavior at occurrence of a certain error can be set for each axis. One error can be assigned to at most on behavior (per axis).

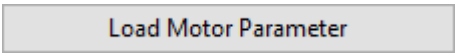
The parameters determine the behavior at fault:

- G-MASKE1: Error leads to run down of the drive.
- G-MASKE2: Error leads to a controlled switch-off.
- G-MASKW: Error is displayed as warning.
- G-MASKD: Error is suppressed.

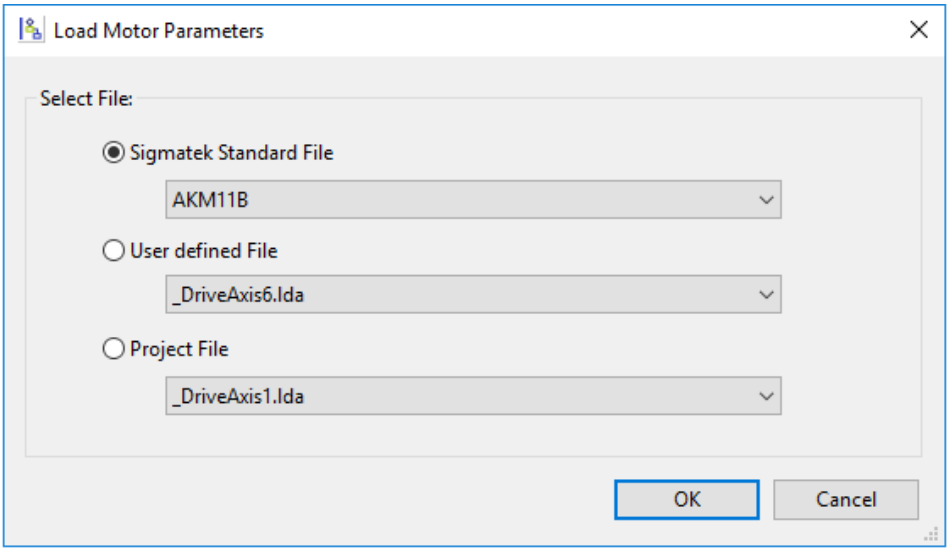
4.4.5.7 Motor Configuration Window

Used to set motor parameters.



With the 

button, all motor and current controller parameters are loaded from a file. The file can be selected dialog box that appears after the click.



Standard Drive File

Motor files (currently AKM only) included in LASAL.

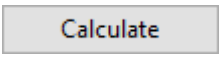
User defined File

Motor files listed under **DiasDrive => Parameter Files => User Defined**.

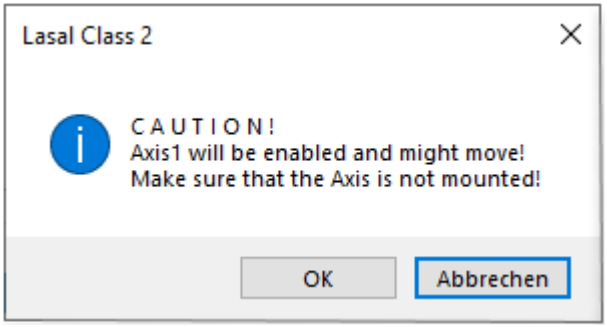
Project File

Motor files already added to the project (an axis object).

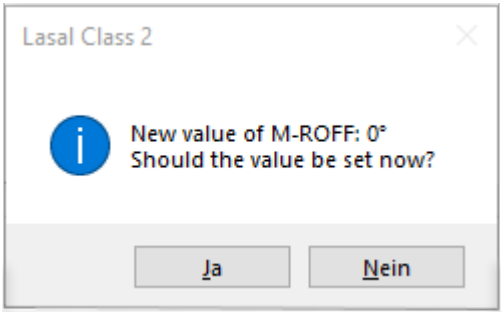
Calculate

With the button 

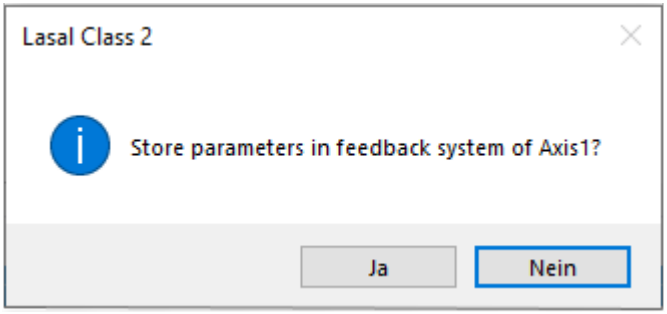
the value M-ROFF (Feedback offset) can be recalculated. Since this can be associated with risks, it is pointed out in message boxes. First, a message is displayed that the selected axis is enabled and therefore could move (hanging loads). To avoid this, it should be demounted:



If you confirm the dialog with **OK**, the determination is carried out. Then you are asked whether the M-ROFF variable (i.e. the feedback offset) should actually be changed to the calculated value:



If this question was also answered with **Yes**, the result is taken over into the table. If the feedback encoder is a Hyperface or EnDat, you are asked whether the values should actually be written to the encoder-internal memory of the axis:

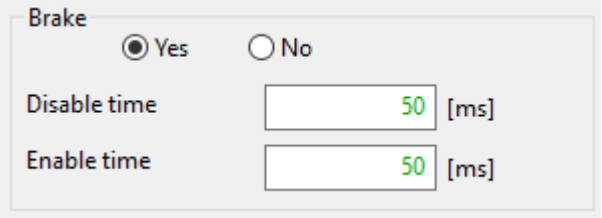


Attention

If this question is answered with **Yes, all the M-parameters** are written to the axis!

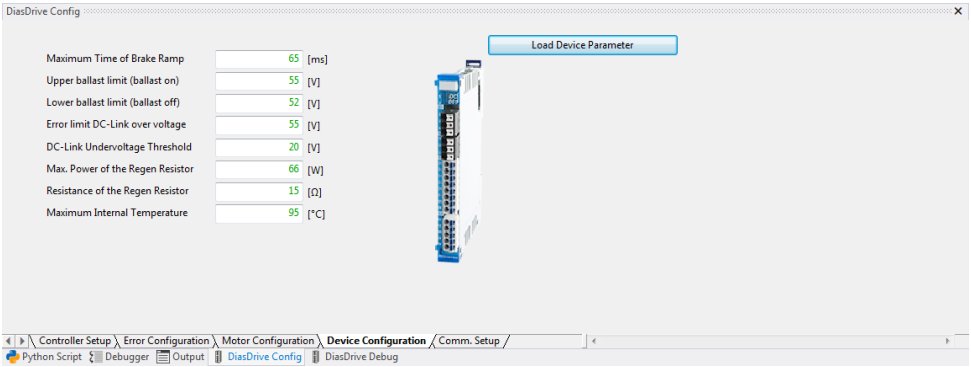
Brake

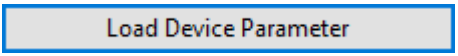
In the Motor configuration window, a possibly existing brake can also be switched on and off (Yes / No, this selection corresponds to the value of M-BRAKE) and the brake release time (Disable time) and brake application time (Enabled time) can be specified:



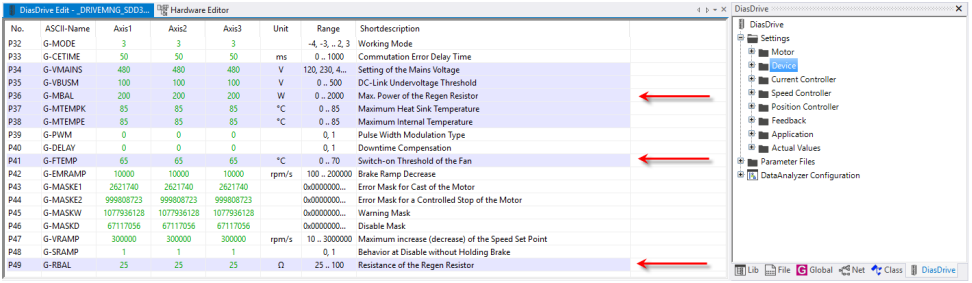
4.4.5.8 Device Configuration Window

Used to set motor parameters.



The 

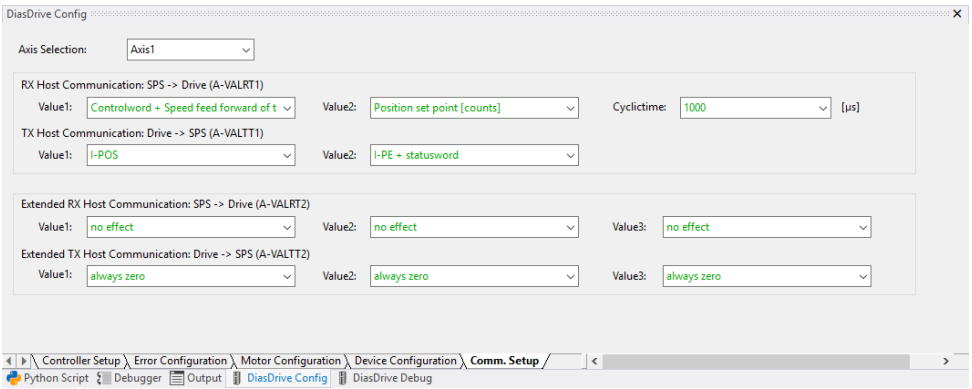
button loads all (axis-independent) parameters from the device group shown here (axis-independent). This are also highlighted with color when the device folder in the DiasDrive tree is selected:



The file can be selected – exactly the same as in the motor configuration – from the dialog box that appears after the click.

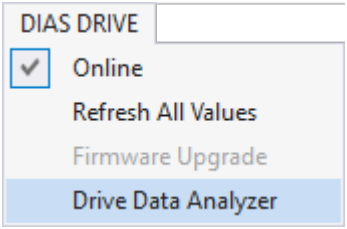
4.4.5.9 Comm.(unication) Setup Window

Used to set the communication parameters **A-VALRT1**, **A-VALRT2**, **A-VALTT1** and **A-VALTT2**. For more information see the Drive documentation.

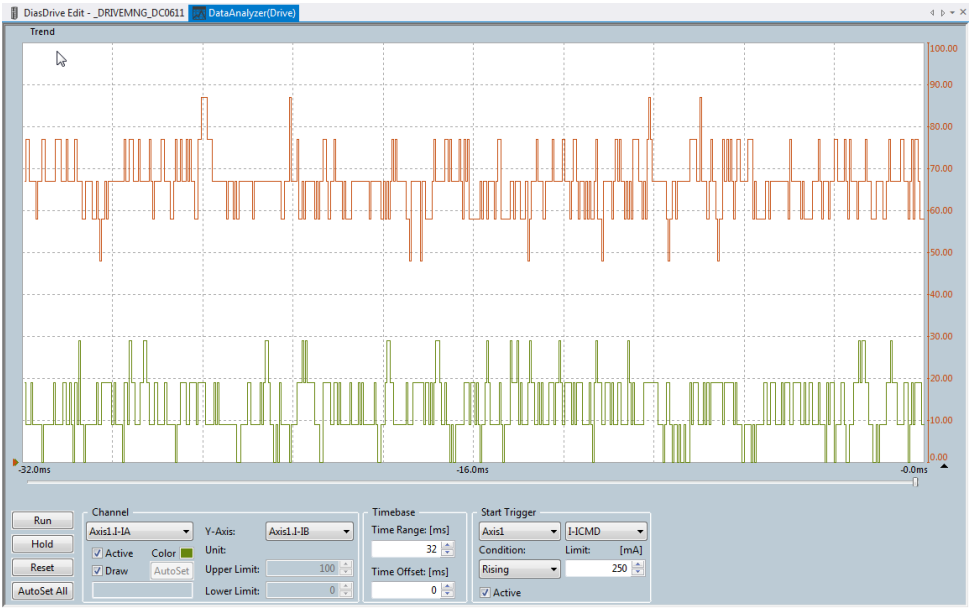


4.4.6 DataAnalyzer for DiasDrive

The DataAnalyzer for DiasDrive can be started via the menu entry **DiasDrive, Drive DataAnalyzer** or with a double click on the tree entry **DataAnalyzer Configuration**.



This special DataAnalyzer is characterized with the (**Drive**) add-on.

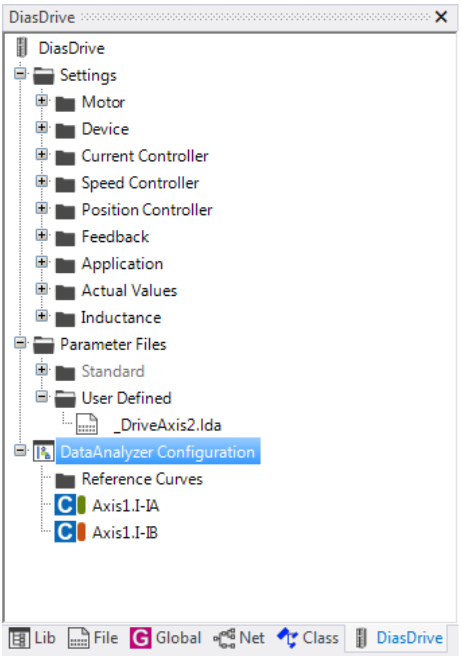


The basic operation of the DataAnalyzer is described in the **DataAnalyzer** chapter.

In contrast to the general DataAnalyzer only a maximum of 3 record channels can be installed at the DataAnalyzer for the Drive. Furthermore the selection of the sample rate and the possible parameters for the record is fix.

Furthermore the name of the recording channel is automatically adapted to the parameter and axis name.

The configuration of the trigger includes raising or falling edge at the condition. Moreover the number of sample points before trigger event can be entered at **Presamples** parameter.



Properties

Sample time [µs]	62.5
TriggerStart	
Active	true
Axis	Axis1
Parameter	I-ICMD
Type	Start
Condition	Rising
Value	250
Presamples	100%
Comment	
TimeRange	32 ms
TimeOffset	0 ms
ScreenView	TREND
Accessories	Zero line
Prescale	1
Comment	
Filename	

TriggerStart
Trigger which starts recoding

The record comprises 512 sample points, which are started via the **Run button** and stored in the Drive until completeness. Not until the data memory is full the data are automatically loaded to the DataAnalyzer. Already during transfer the received information are graphically displayed.

The state of transfer can be seen in the status line via a bar. Through pressing the **Stop key** the transfer is canceled. The **Hold key** only stops the diagram but not the transfer.



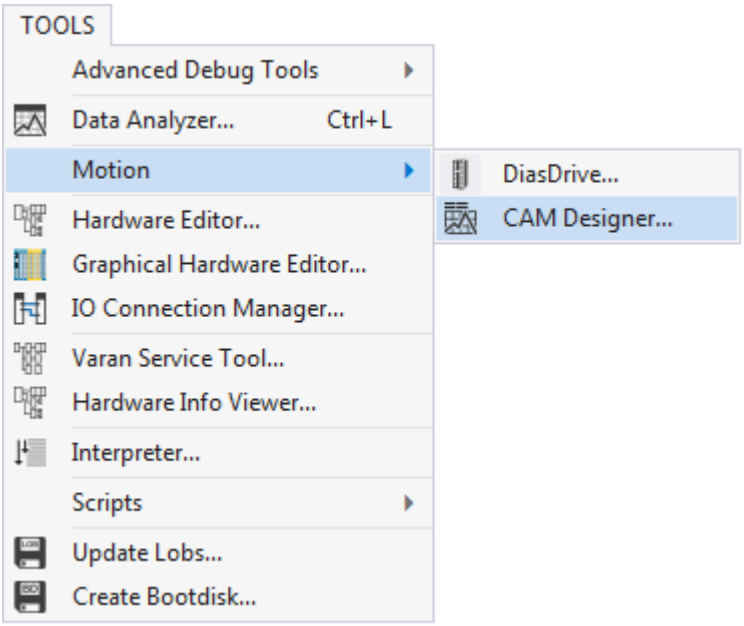
The record of values can only takes place with an online connection to the DiasDrive. The configuration of the DataAnalyzer, however, can also be done in offline mode.

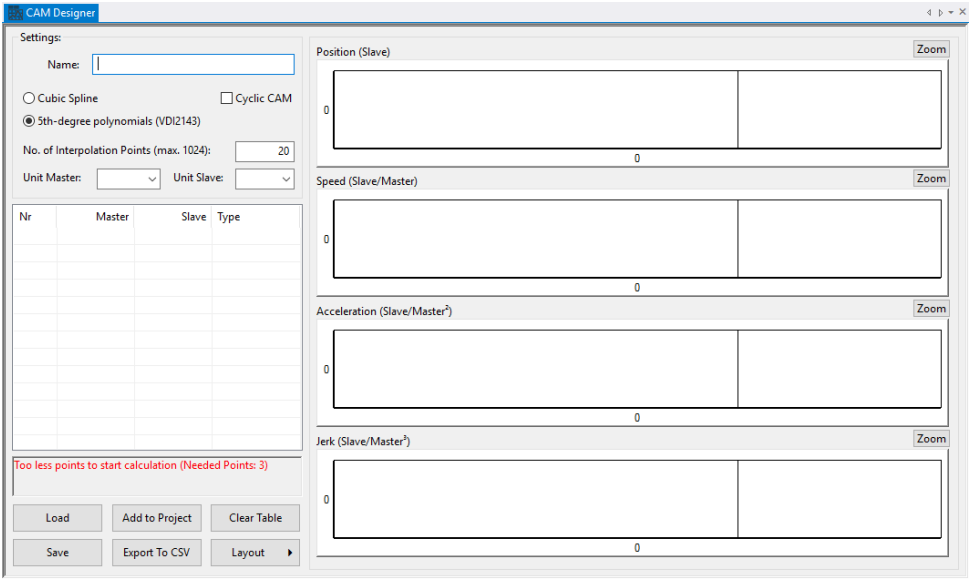
4.5 CAM-Designer

This application allows the display of position, speed, acceleration and jerk curves with two axes.

4.5.1 Accessibility

In Tools menu, the electronic CAM editor is started by selecting the menu item, **Motion / CAM Designer**.





The window is divided into two areas: On the left, the input and control area is located; to the right is the display area.

4.5.2 The Input and Control Area (left side)

The left area is further divided into four sub sections.

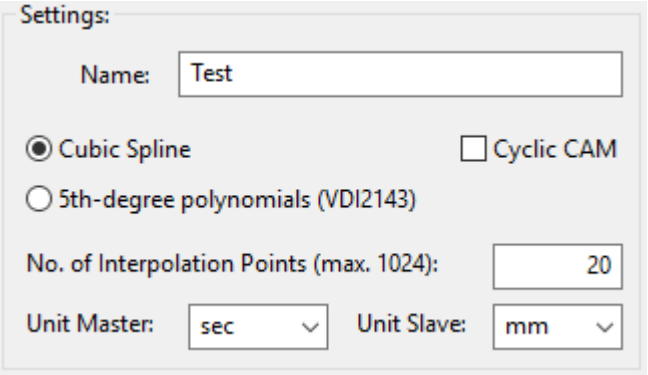
Settings

The Input Table

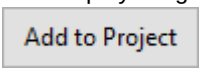
Error Message Area

Control Area

4.5.2.1 Settings



In this area, the file name, order of the polynomials, number of interpolation points and the axis units are given, as well as whether on not the motion is cyclically repeated.

At first, the **name** does not play a significant roll and can be arbitrary. If the curve is added to the project with 

however, a class is created with the entered name. If for any reason, the name conflicts with LASAL conventions, a compiler error message is generated.

If the class name already exists, only the table is added or if required, the existing table data is overwritten. Other functions of the class are unaffected!

The automatically generated CAM table is always named `CAMTAB01`. If the class is newly created via the CAM Designer, a read() method that returns the CAM table is automatically created.

Using **“Cubic Spline”** or **“5th-degree polynomials (VDI2143)”**, the degree of the polynomial (3rd or 5th degree) can be defined for the curve display. For more information on 5th degree polynomials, see the VDI 2143 standard (VDI = Verein Deutscher Ingenieure = Organization of German Engineers).



At least 3 points must be defined so that the CAM Editor can draw a curve.

If the control box **“Cyclic CAM”** is selected, the electronic CAM editor assumes a cyclically repeated motion.

When “Cubic Spline” and “Cyclic CAM” are selected at the same time, the Y values of the first and last points must be identical – otherwise an error message is triggered.

If “softer, rounder” curves with fewer corners are desired, a higher number can be entered in **“No. Of Interpolation Points“**.

To make imputing values into the table more comfortable, the desired units of measurement can be defined under **Units**. The units provided by SIGMATEK can be selected directly, however, user defined input is also possible. In this case, the unit of measurement should be defined in a project file. Otherwise, a compiler error will occur.

4.5.2.2 The Input Table

Nr	Master	Slave	Type
0	0	0	Stop
1	5	-2	Stop
2	8	7	Stop
3	10	12	Stop
4	20	0	Stop

Here, the values for both axes are entered. The first column (**Nr**) cannot be changed; the other columns can be edited.

If no values (or only 1 or 2 lines) are visible in the table, no curve is displayed. In the error message area, the error



is visible. A double click in the free surface is enough to immediately add a line with start values. This can be changed immediately.

To exit the error message, at least 3 values are needed to calculate and display a curve.

Numbers entered In the **Master** column must be **whole numbers** (anything other will not be accepted), **unique** (otherwise an error messaged) and **ascending** (otherwise the table is resorted automatically). They represent the **horizontal axis** in the displayed curve.

The numbers in the **Slave** curve can be entered freely. They must be **whole numbers** only, but can **also be negative**. They represent the vertical axis in the position curve.

The last column (**Type**) can only be changed / is functional when **5th degree polynomials** (**VDI2143**) are selected.

Here, between four different point types is distinguished:

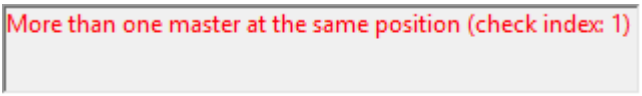
1. Stop (= stop point)
2. Speed (= speed point)
3. Reversal (= reverse point)
4. Move (= motion point)

For more information on the individual point types, see the **guideline VDI 2143**.

Alternatively to the table input, it is also possible to enter an additional point (= table line) directly in the **position curve**. This is done in the curve display through the **context menu** (right click) on the desired location and selecting **“Add New Point“**. The new point is added immediately – and in the graphic as well as in the table.

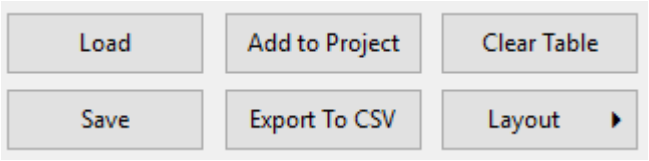
A **change** in the table values can be made at **any time** and is affects the curve display in real time. A line is **deleted** by pressing the **Del** button; a **new line** can be added by **double clicking** in the **free area** of the table.

4.5.2.3 Error Message Area

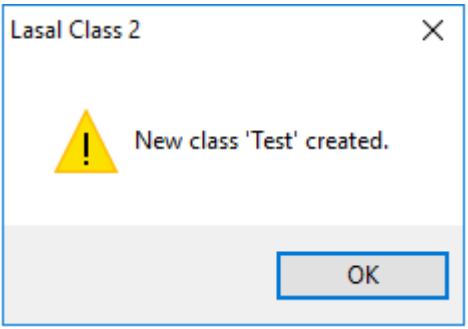


As mentioned above, **error messages** are shown. As the same time, the messages are also output /recorded In the Output and logging tabs. The message above indicates that the line or master number is not unique.

4.5.2.4 Control Area



In this area, the user can issue the application **commands**. The curve can be saved (**Save**; an **.xml file** with the ending ***.cam** is created) and loaded (**Load**). As previously mentioned, they can be added to the project (**Add to Project**). A class with the specified name is thereby generated.



With a double click on the class name, the generated table is visible in the editor in the class view.

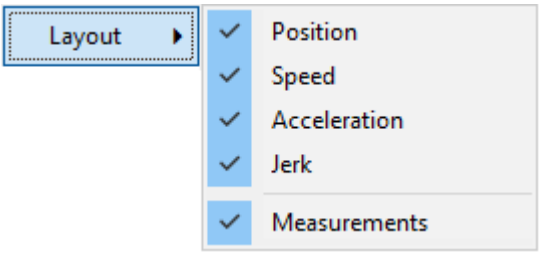
CAM Designer | Test* [RO]

Debugging Context:

```
1
2
3   FUNCTION VIRTUAL GLOBAL Test::ClassSvr::Read
4     VAR_OUTPUT
5       output (EAX)   : DINT;
6     END_VAR
7
8     output$pVoid := (#CAMTAB01())+sizeof(DINT);
9
10  END_FUNCTION
11
12
13  FUNCTION TAB Test::CAMTAB01
14    0000020$UDINT, //Size of table (lines)
15    TO_DINT(0.0000$REAL sec), TO_DINT(0.0000$REAL mm),
16    TO_DINT(0.3158$REAL sec), TO_DINT(0.7116$REAL mm),
17    TO_DINT(0.6316$REAL sec), TO_DINT(1.3700$REAL mm),
18    TO_DINT(0.9474$REAL sec), TO_DINT(1.9221$REAL mm),
19    TO_DINT(1.2632$REAL sec), TO_DINT(2.3271$REAL mm),
20    TO_DINT(1.5789$REAL sec), TO_DINT(2.6280$REAL mm),
21    TO_DINT(1.8947$REAL sec), TO_DINT(2.9016$REAL mm),
22    TO_DINT(2.2105$REAL sec), TO_DINT(3.2191$REAL mm),
23    TO_DINT(2.5263$REAL sec), TO_DINT(3.5865$REAL mm),
24    TO_DINT(2.8421$REAL sec), TO_DINT(3.9672$REAL mm),
25    TO_DINT(3.1579$REAL sec), TO_DINT(4.3241$REAL mm),
26    TO_DINT(3.4737$REAL sec), TO_DINT(4.6202$REAL mm),
27    TO_DINT(3.7895$REAL sec), TO_DINT(4.8182$REAL mm),
28    TO_DINT(4.1053$REAL sec), TO_DINT(4.8810$REAL mm),
29    TO_DINT(4.4211$REAL sec), TO_DINT(4.7715$REAL mm),
30    TO_DINT(4.7368$REAL sec), TO_DINT(4.4526$REAL mm),
31    TO_DINT(5.0526$REAL sec), TO_DINT(3.8871$REAL mm),
32    TO_DINT(5.3684$REAL sec), TO_DINT(3.0708$REAL mm),
33    TO_DINT(5.6842$REAL sec), TO_DINT(2.0792$REAL mm),
34    TO_DINT(6.0000$REAL sec), TO_DINT(1.0000$REAL mm),
35  END_FUNCTION
36
```

The data can also be exported as an excel table (**Export To CSV**), the table reset (Clear Table) and how the curve(s) is shown in the display area (**Layout =>**) can be decided.

If the **Layout button** is clicked, the following options are provided:



Here, curves are shown or hidden, or whether to show the units in the position curve can be selected.

The four display options for curves are:

1. Position curve
2. Speed curve (= first derivation)
3. Acceleration curve (= second derivation)
4. Jerk curve (= third derivation)

The checked options are displayed, the others are not.

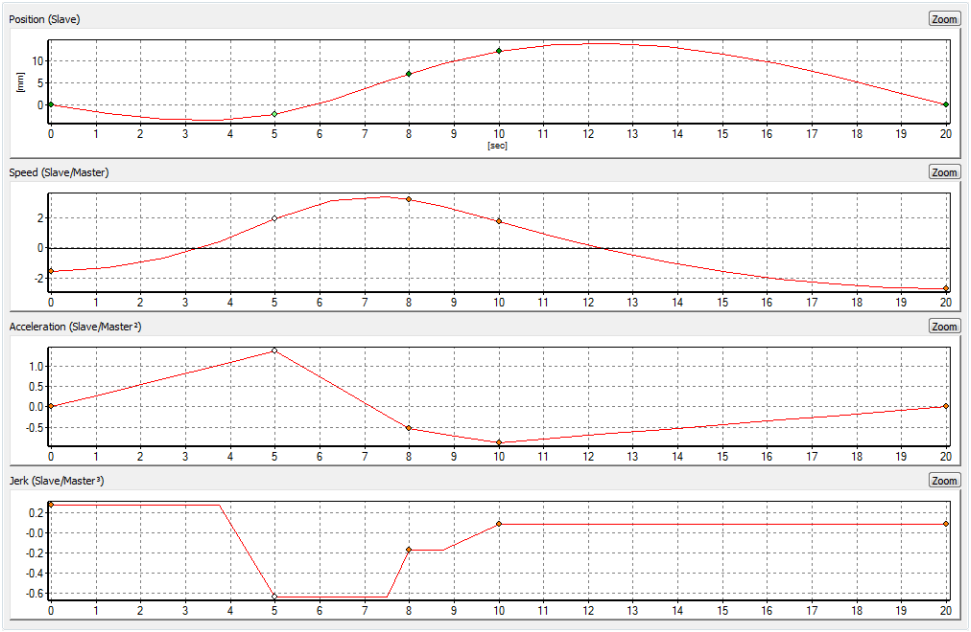
The last option (measurement) controls how the axis units are displayed in the position curve. Disabling this parameter saves space to display the curve.

4.5.3 The Display Area (right side)

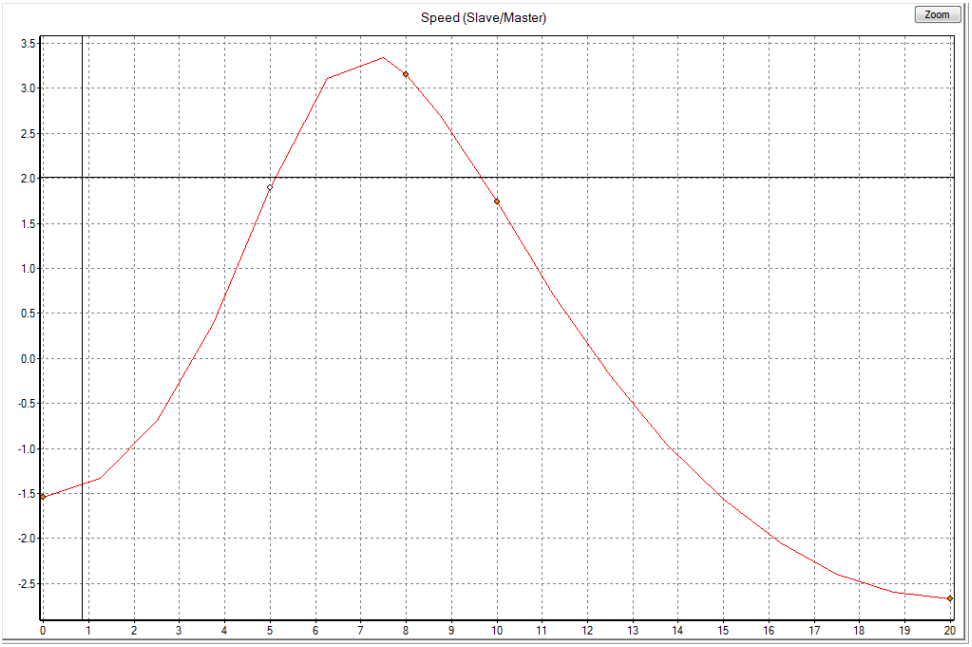
4.5.3.1 Curve Display and Zoom

The **display area** is located on the right half or the CAM Designer.

In the default setting, all four curve displays are shown.



Each image has a small zoom button in the upper right corner. If this button is clicked with the left mouse button, a curve is englarged to the full display area size.



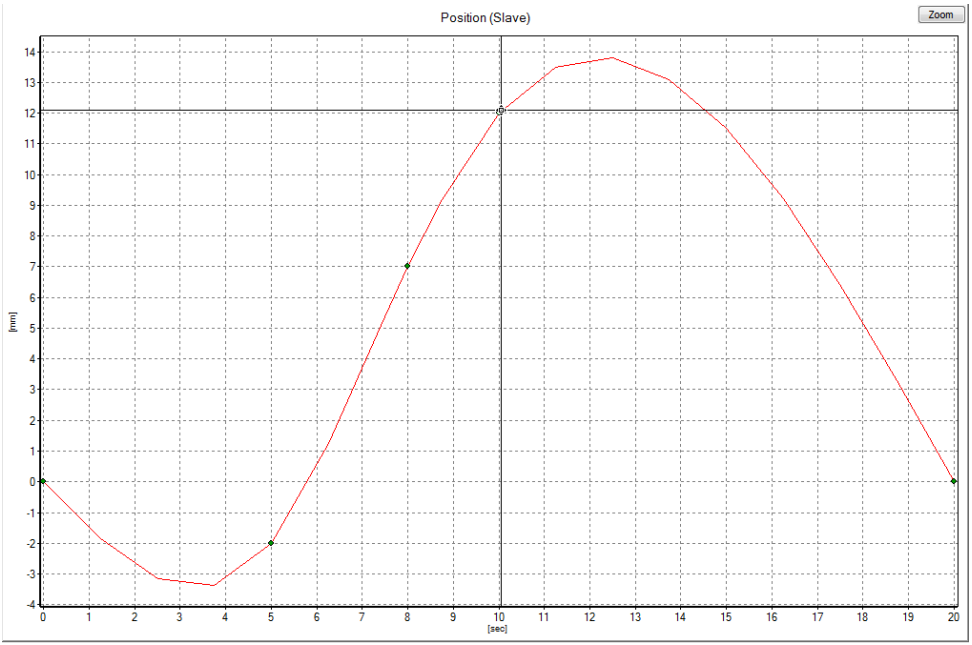
With a new click on the zoom button the image is reset to the original size.

In both the curve display and the table, points can be selected. In the other display, the marking shifts at the same time.

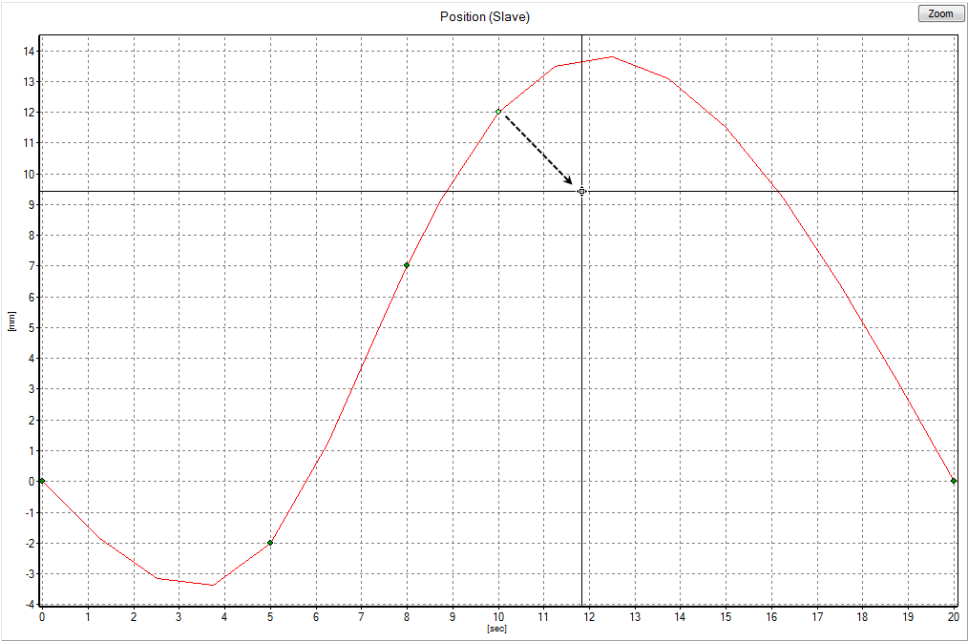
While the **selected point** is marked with a **dark blue background** in the **table**, in the **curve display**, the **selected point is lighter** than the others.

4.5.3.2 Change of Data

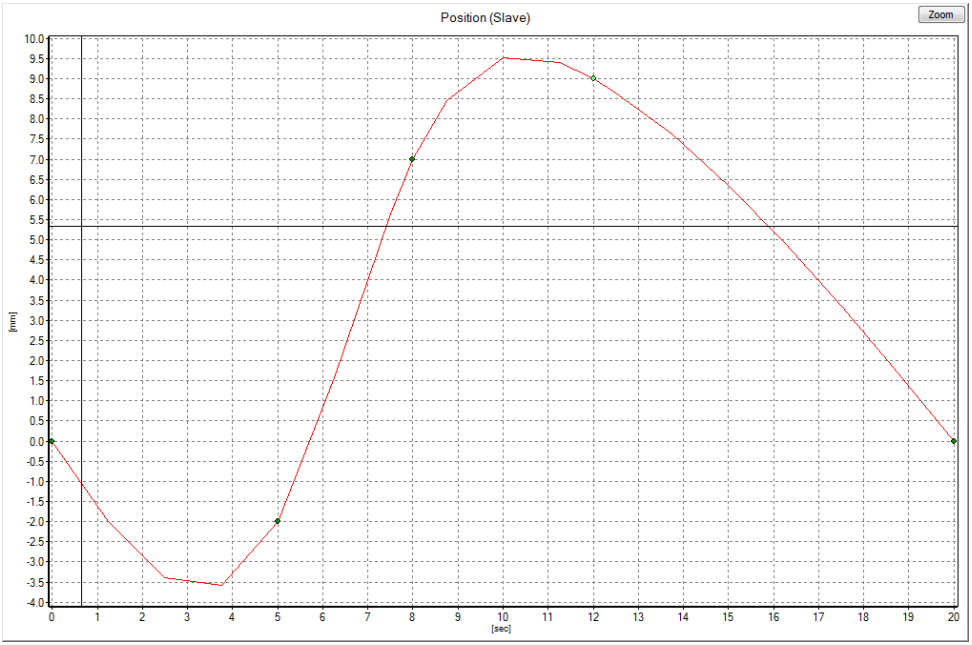
In the **position curve** (and **only there**), it is possible to **shift an existing point**. Here, the mouse must be move so close to the point so that the mouse pointer changes. **In the other curve displays**, the point can be selected (whereby the marking in the table is moved accordingly), but not shifted.



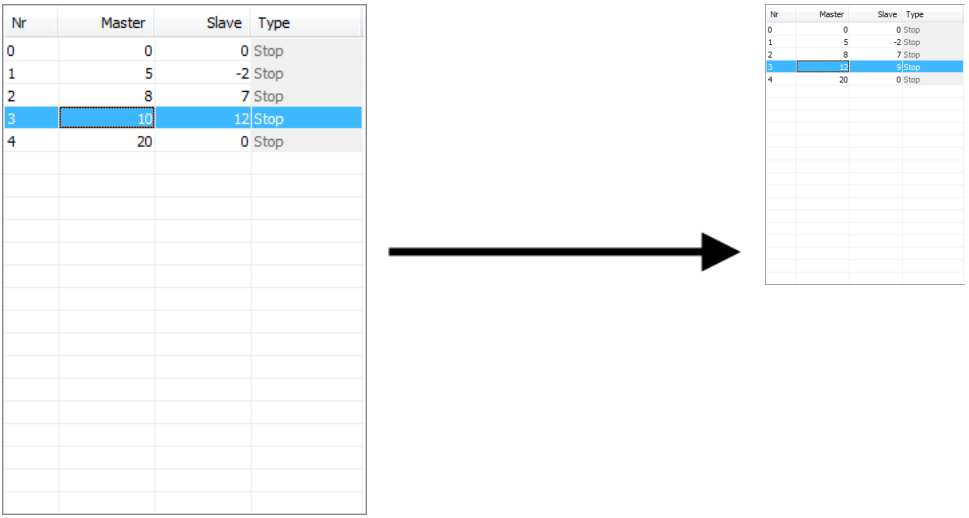
Afterwards, the point can be moved when the mouse button is pressed and held.



After releasing the left mouse button, the new value is assumed – in the curve



as well as in the table.



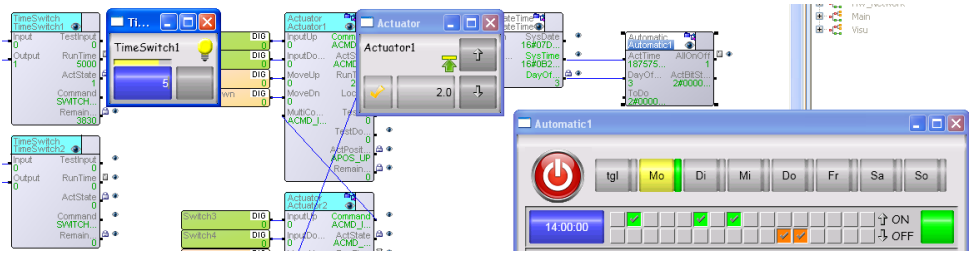
Nr	Master	Slave	Type
0	0	0	Stop
1	5	-2	Stop
2	8	7	Stop
3	10	12	Stop
4	20	0	Stop

It should be noted that the display I the position curve – scanning, value limits etc. - can automatically adjust to the new data.

Of course, the same effect can be achieved by changing the data in the table accordingly.

4.6 VOV (Visual Object View)

A VOV is used to visually simulate (online operation) and set parameters (offline operation) of one or more objects in the object network of LASAL CLASS.



The **VOV objects** are created in LSE with the menu item **„Export VOV“**. Here, the property must be set to **"Visualized = true"** in appropriate **objects and servers**. In both the object network and the project tree, this is indicated with the eye symbol. After **compilation**, the **MaeExp.txt** file is automatically created and must then be inked to LSE (see LSE help).

4.6.1 VOV-(Visual Object View-) Attaching Files to Classes

As previously mentioned, a *.VOV file can be added to a class by selecting "Add File..." in the class context menu. If a file in the Files sub folder is double clicked, LASAL searches for a lse.exe file:

1. first under the path defined in Extras/Options/LSE Path, when nothing is found here,
2. the LSE installation path is searched, if unsuccessful,
3. the relative LSE path to LASAL CLASS 2/Bin is then searched.

If LASAL still could not find an executable LSE file, an attempt is made to open the *.VOV file with a file association.

If LSE is installed correctly and objects are available in the *.VOV file, LSE opens the visualization.

Changes in LSE are assumed when closing. All views in LASAL CLASS 2 must first be closed!

If the *.VOV file is now added added to the class, from which an object is then placed in the object network, the *.VOV file can be opened through the object's context menu using the menu item "Open Visual Object View".

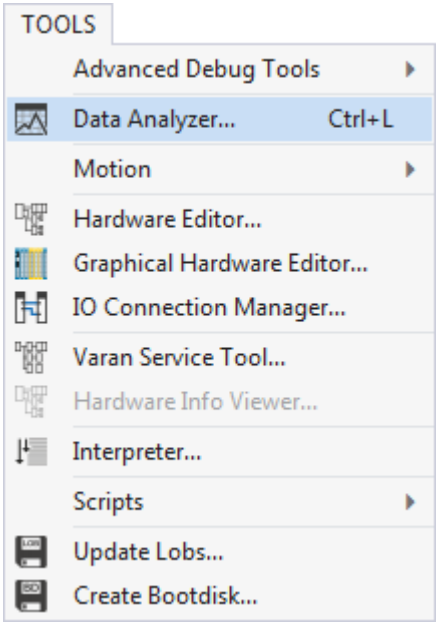
It is also possible to add multiple VOV's to a class, the desired VOV can then be selected to open.

4.7 DataAnalyzer

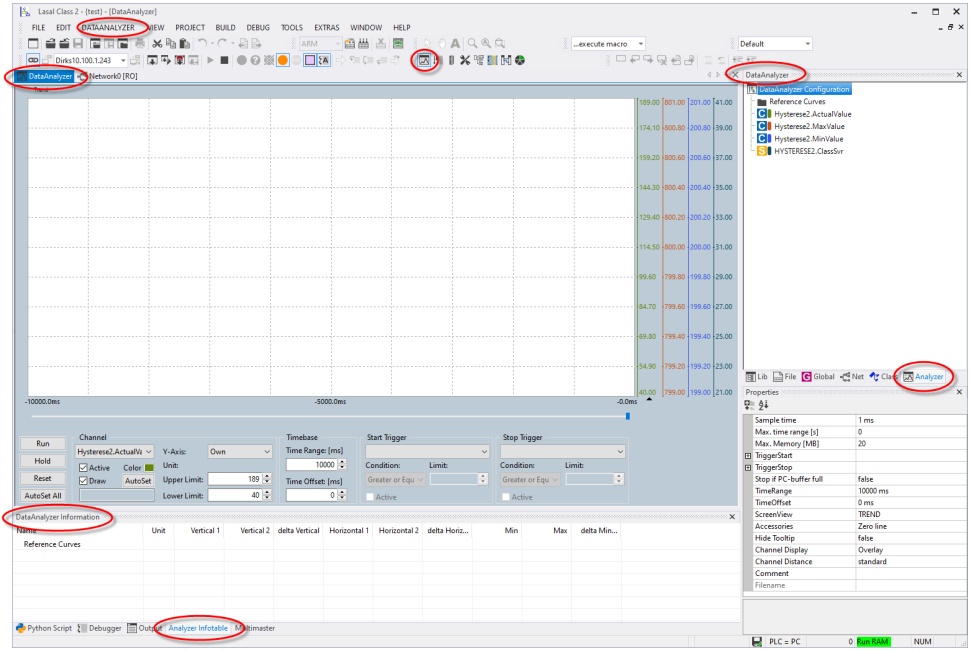
The DataAnalyzer in LASAL CLASS 2 serves the recording and graphic display of measuring data, made on the PLC.

4.7.1 DataAnalyzer User Interface

To activate the DataAnalyzer user interface in LASAL CLASS 2 select the menu entry **Debug, Tools, Data Analyzer** or the  symbol in the **Toolbox toolbar**.



Thereby several windows (**DataAnalyzer window** and **DataAnalyzer tree**) are automatically opened. Additionally the **DataAnalyzer** menu entry is installed in the menu bar, which is only visible at displayed **DataAnalyzer** window:

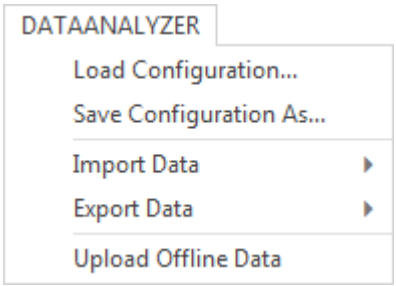


Through closing the **DataAnalyzer window** or through repeated selection of the **Debug, Tools, Data Analyzer** menu entry as well as the **DataAnalyzer** symbol, the DataAnalyzer user interface (window and menu) will be closed again.

To operate the **DataAnalyzer**, in most cases no LASAL CLASS 2 project is necessary. Exceptions are, among other things, system variables, global variables and variables of objects.

4.7.1.1 DataAnalyzer Menu

Via this menu different configurations can be saved (**Save Configuration** and **Save Configuration As**) and loaded (**Load Configuration**) again. Furthermore recorded measurements can be exported (**Export Data**) to a file (measuring points are separated through commas) and reloaded (**Import Data**) again.Further details can be found in the chapter Importing / Exporting Data.



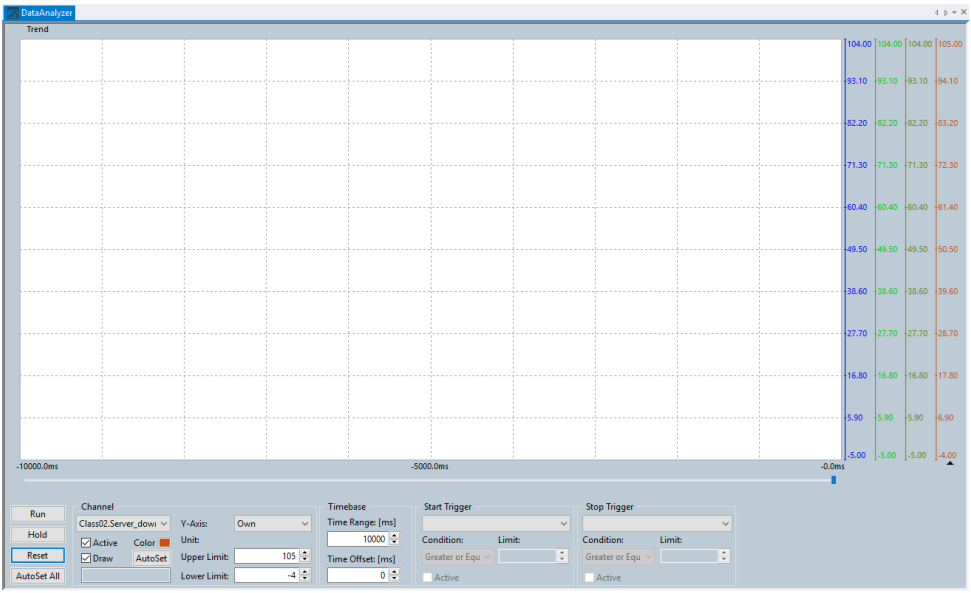
If a recording has been started on a control and then you went offline, the meanwhile recorded data can be uploaded and displayed with the menu entry **Upload Offline Data** after going online again with the same configuration.

The following rules specify how/which configuration is stored/loaded depending, among other things, whether the DataAnalyzer is open while saving/loading:

1.	Load project: DA not opened	Config. from project (load)
2.	Load project: DA opened	Config. remains unchanged
3.	New channel is assigned	Cfg b. Proj. (speichern)
4.	Konfiguration oder Scope laden	Not configured with project (save)
5.	Change channel info, axis or parameter	Configured with project (save)
6.	Load DA configuration from project (new context menu)	Configured with project. (ld./sp.)
7.	Save DA configuration with project (context menu)	Configured with project (save)

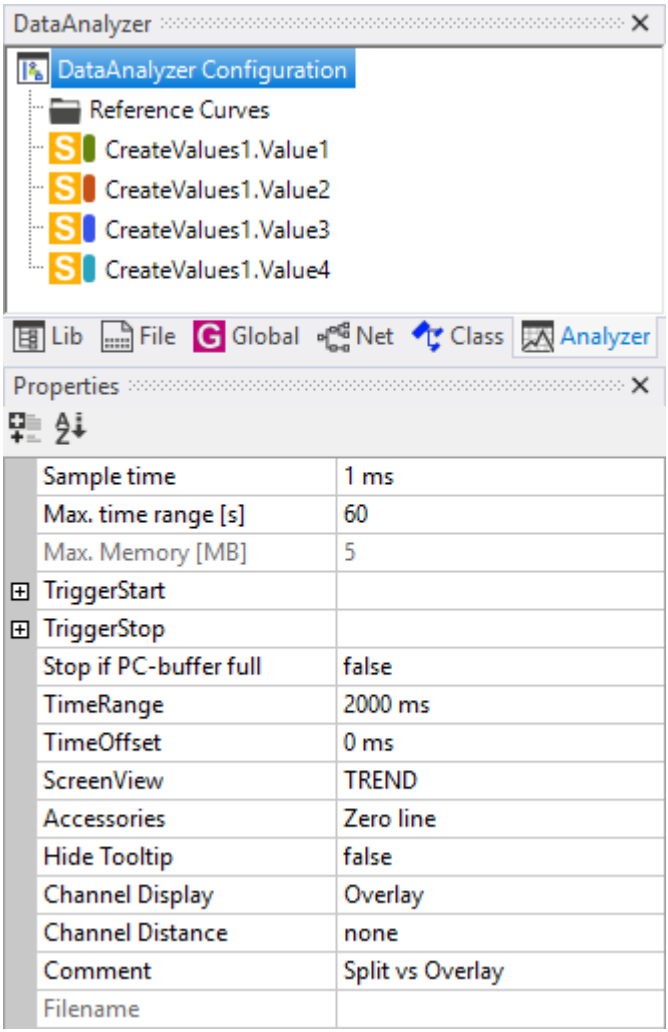
4.7.1.2 DataAnalyzer Main Window

This window shows the graphic display of the measuring data and control elements for starting the recording and configuring the diagram.



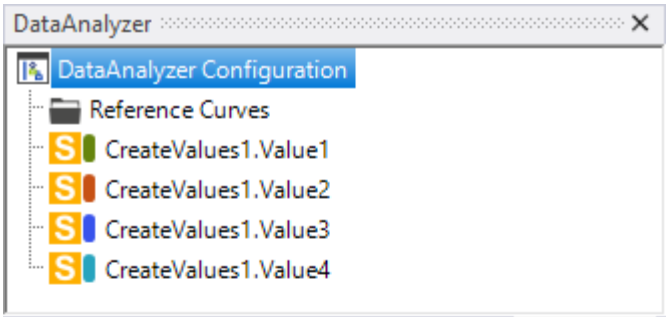
4.7.1.3 DataAnalyzer Tree

This tree includes the complete configuration of the DataAnalyzer: Setting the sample rate, recording channels, Trigger and so on.



4.7.2 DataAnalyzer Configuration

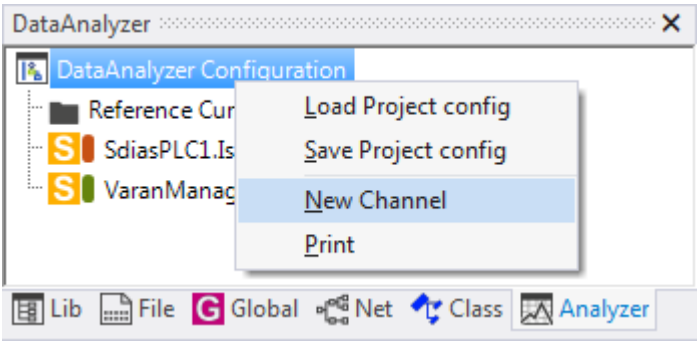
The configuration of the DataAnalyzer takes place in the tree view and the Properties window. If the topmost element of the DataAnalyzer tree is selected (root), then the general settings of the Analyzer, like sample rate, time resolution, Trigger and so on, are displayed in the Properties window:



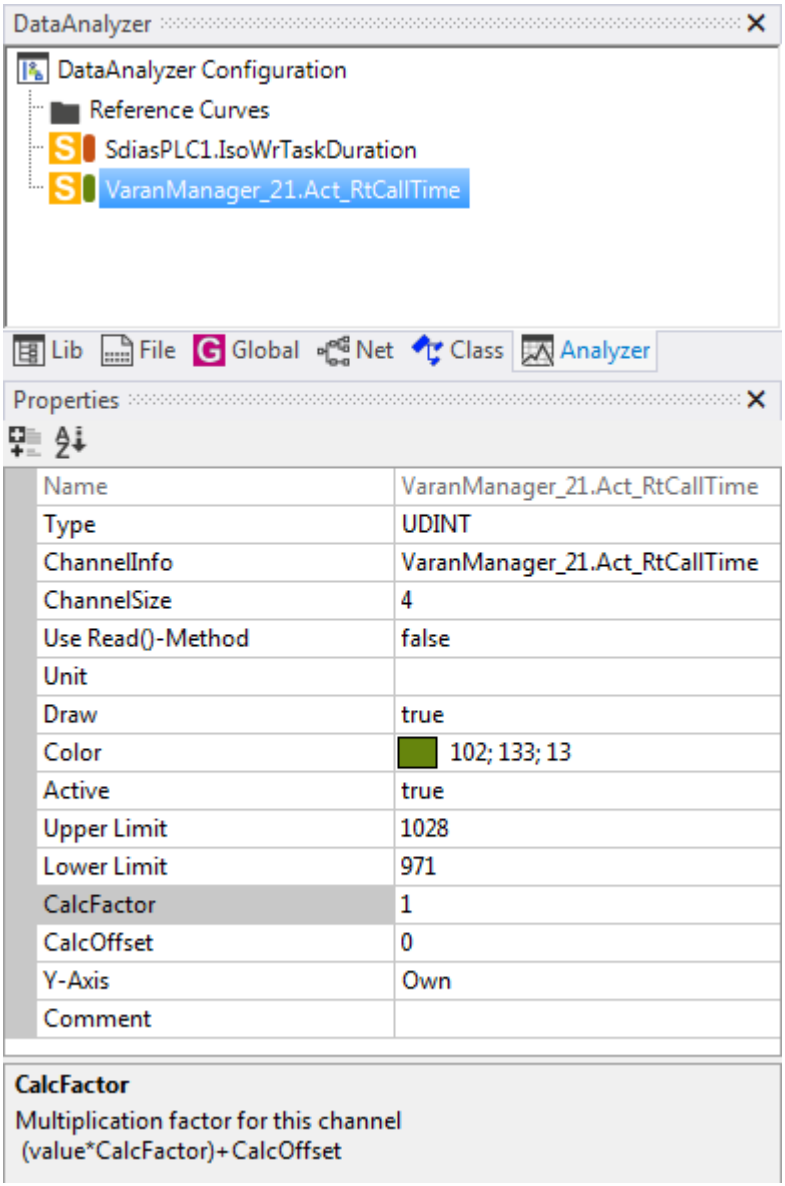
Properties

Sample time	1 ms
Max. time range [s]	60
Max. Memory [MB]	5
⊟ TriggerStart	
Active	false
Type	Start
Condition	Greater
Channel	TriggerStart.sActive
Value	1
Comment	StartTrigger<CR><LF>
⊟ TriggerStop	
Active	false
Type	Stop
Condition	Greater
Channel	TriggerStop.sActive
Value	1
Comment	StopTrigger<CR><LF>
Stop if PC-buffer full	false
TimeRange	2000 ms
TimeOffset	0 ms
ScreenView	TREND
Accessories	Zero line
Hide Tooltip	false
Channel Display	Overlay
Channel Distance	none

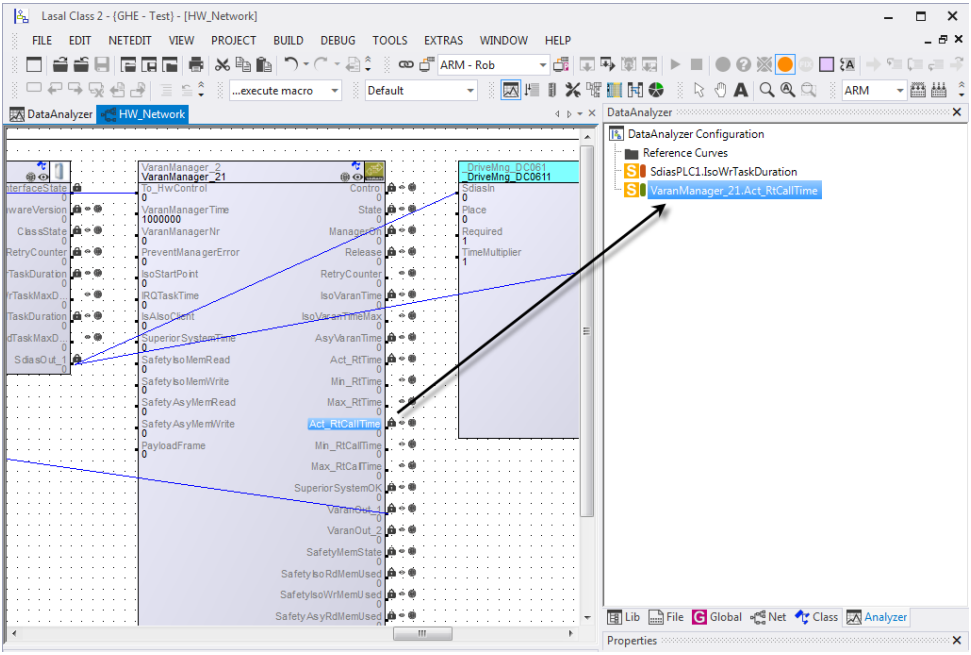
There are several ways to install a new recording channel. First of all in the tree, the context menu of the topmost entry (DataAnalyzer Configuration) has to be opened with a right mouse click (or the corresponding key on the keyboard). Here the menu entry **New Channel** has to be selected to create a new channel.



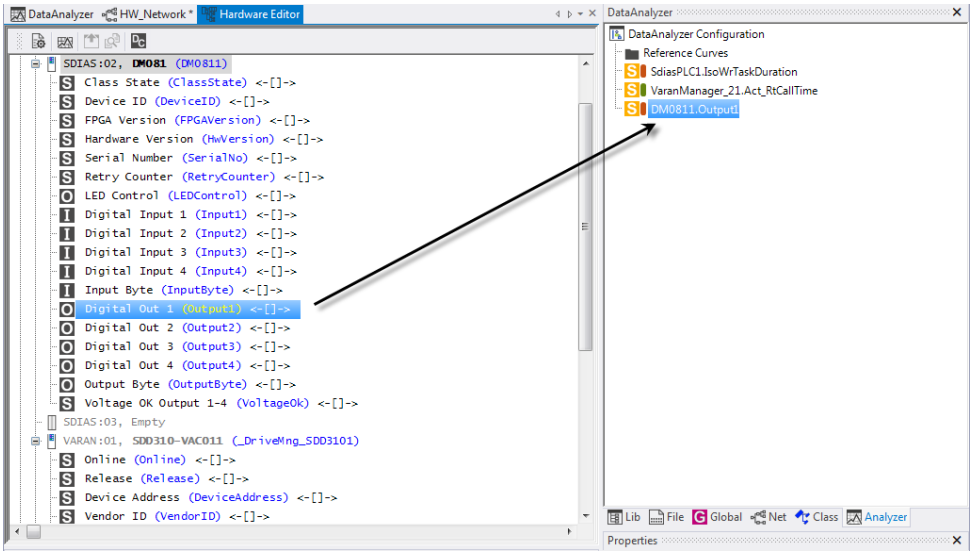
The configuration of the new channel takes place again via the **Properties window**. This type of channel configuration is used for recording of memory addresses or (global) variables.


DataAnalyzer
DataAnalyzer Configuration
Reference Curves
SdiasPLC1.IsoWrTaskDuration
VaranManager_21.Act_RtCallTime
Lib
File
Global
Net
Class
Analyzer
Properties
Name
VaranManager_21.Act_RtCallTime
Type
UDINT
ChannelInfo
VaranManager_21.Act_RtCallTime
ChannelSize
4
Use Read()-Method
false
Unit
Draw
true
Color
102; 133; 13
Active
true
Upper Limit
1028
Lower Limit
971
CalcFactor
1
CalcOffset
0
Y-Axis
Own
Comment
CalcFactor
Multiplication factor for this channel
(value*CalcFactor)+CalcOffset

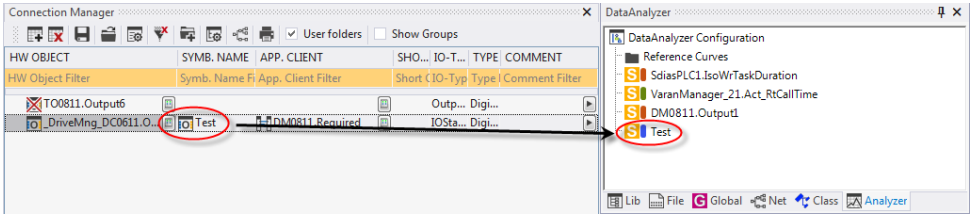
Channel configuration can also be done via **Drag & Drop**. Here a channel of a ONE object (Client or Server channel) is dragged to the topmost element in the DataAnalyzer tree with the mouse. The necessary settings are made automatically.



A different method is to **Drag & Drop** in or outputs **directly from the HW Editor** into the DataAnalyzer:



It is also possible to Drag & Drop a connection from the connection manager into the DataAnalyzer:



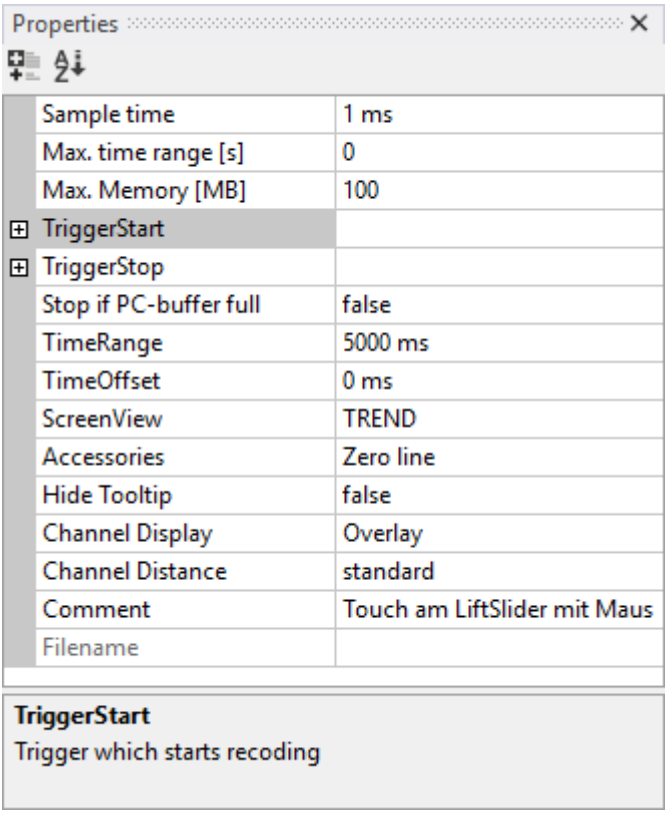
IO objects can also be moved to the DataAnalyzer. The greatest advantage of this method, is that the server of this connection can be changed at a later time, without having to make changes in the DataAnalyzer. Since here, only the IO object is accessed, the connected server is recorded.

If a connection with an IO object is moved to the DataAnalyzer from the Connection Manager via Drag & Drop, the IO object is automatically used as a channel!

Another possibility of channel configuration is, to move the in-/outputs from the **Show DetailView window** in the **Graphical Hardware Editor** to the DataAnalyzer tree with Drag & Drop.

4.7.2.1 DataAnalyzer Properties

The displayed properties can differ through the display mode. This means, in different views properties are hidden. They have no meaning for the set view.



- **Sample time**: Here the sample rate in µs can be set.



If the set sample rate is not possible for the used operating system on the control (to low, not in the time period, …), then this value is automatically adapted at recording start!

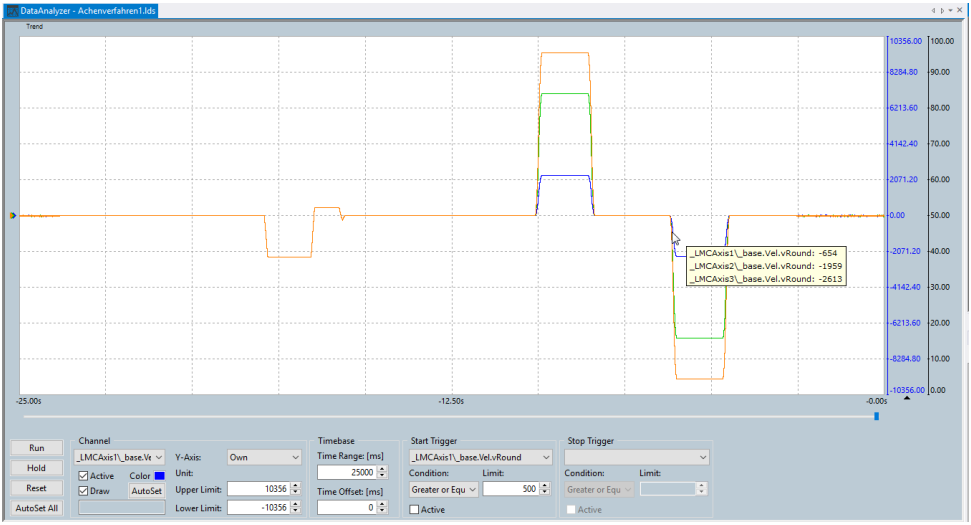


Pay attention to the fact that the recorded values can also be transferred at slow connections between PC and control.

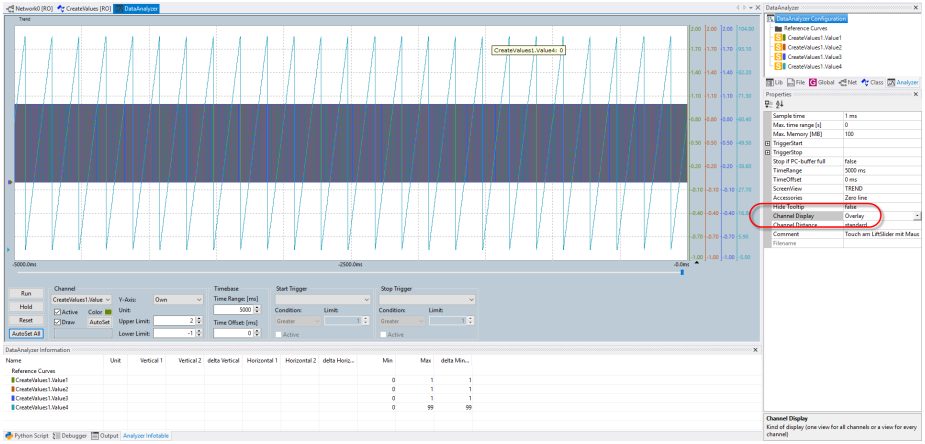
Max. time range [s]	Maximum recording time in seconds to store data in the PC. In combination with the sample rate, this value affects the RAM needed to store data in the PC. The higher this time value is, the amount of RAM needed increases. The can negatively affect the performance of the PC!
Max. Memory [MB]	The maximum PC-RAM available for storing data.
TriggerStart	Start trigger properties. This trigger can also be set over the control panel of the main view.
TriggerStop	Stop trigger properties.
Stop if PC-buffer full	If TRUE is selected, date recording is stopped as soon as the reserved memory in the PS is full. If FALSE is selected, the memory is overwritten.
TimeRange	X-axis resolution in ms. In the properties window, the time unit can also be entered (s, sec, ms, h etc.)
TimeOffset	X-axis offset in ms. In the properties window, the time unit can also be entered (s, sec, ms, h etc.)
ScreenView	Setting of different views: **Trend:** Current value stands on the right; old values are shifted to the left. **Oscilloscope**: Classical oscilloscope diagram. Display starts on the left border, newer values are attached on the right. If the right border is reached, the display starts on the left border again.
Afterglow (only displayed at ScreenView = OSCILLOSCOPE!)	Display of old curves in Oscilloscope view. Up to 3 old curves are shown in faded color.
Accessoires	Enable/disable various accessories: **Zero line:** In the left margin, the arrows are displayed in colors, which correspond to the curve that show the zero line of the matching curves. **Current value:** Shows the position of the actual measurement value of the stopped trace with an empty arrow in the left margin.
Hide Tooltip	Switches on (**true**) or off (**false**) the tooltip value display at the mouse pointer when moving over one or more curves.
Channel Display	Determines whether several curves are displayed one above the other (overlay) or each in its own area (split). If **Split** is selected here, **Relative Height** is displayed in the curve properties

	(see there). Effective only after additional pressing of the button **AutoSet All**!
Channel Distance	If **Split** was selected in Channel Display, the distance between the individual display areas can be set here. There are three possibilities: **none** (no distance), **standard** (little distance) and **big** (greater distance). Effective only after additional pressing of the button **AutoSet All**!
Comment	Any comment

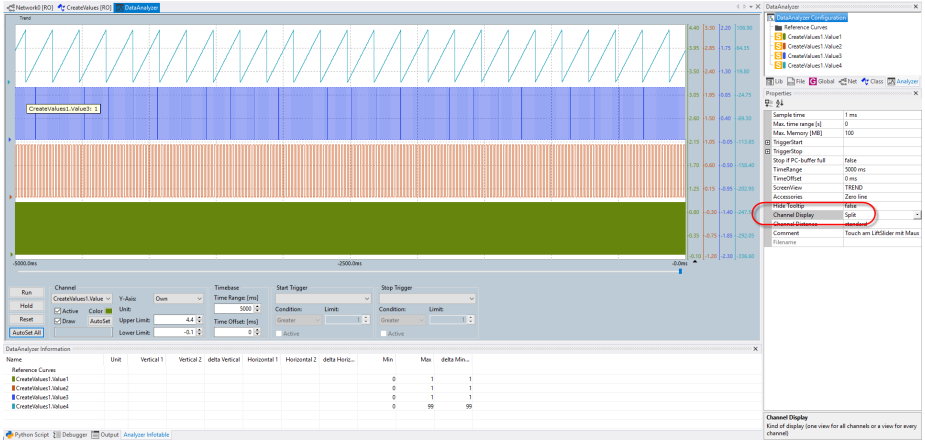
Tooltip:



Overlay:

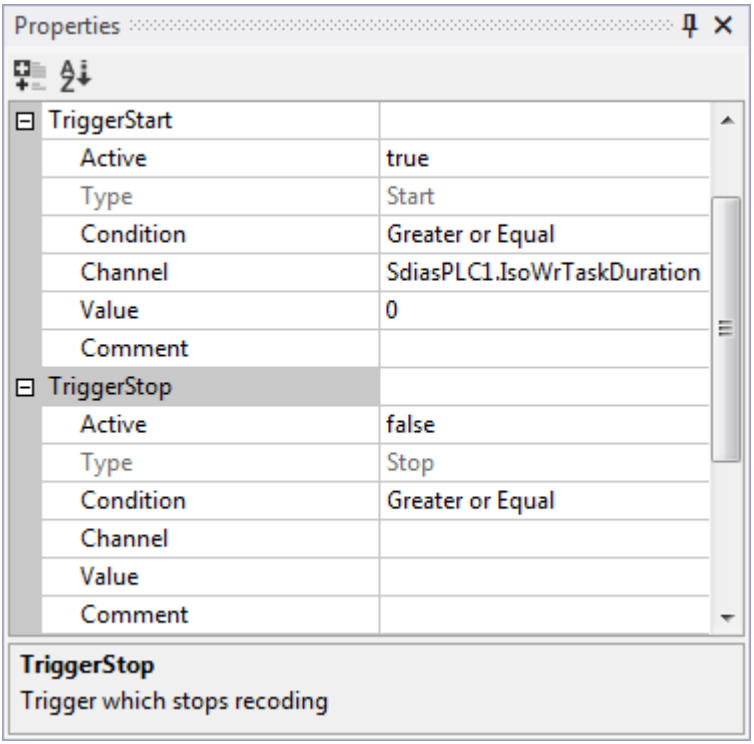


Split:



4.7.2.2 Trigger Properties

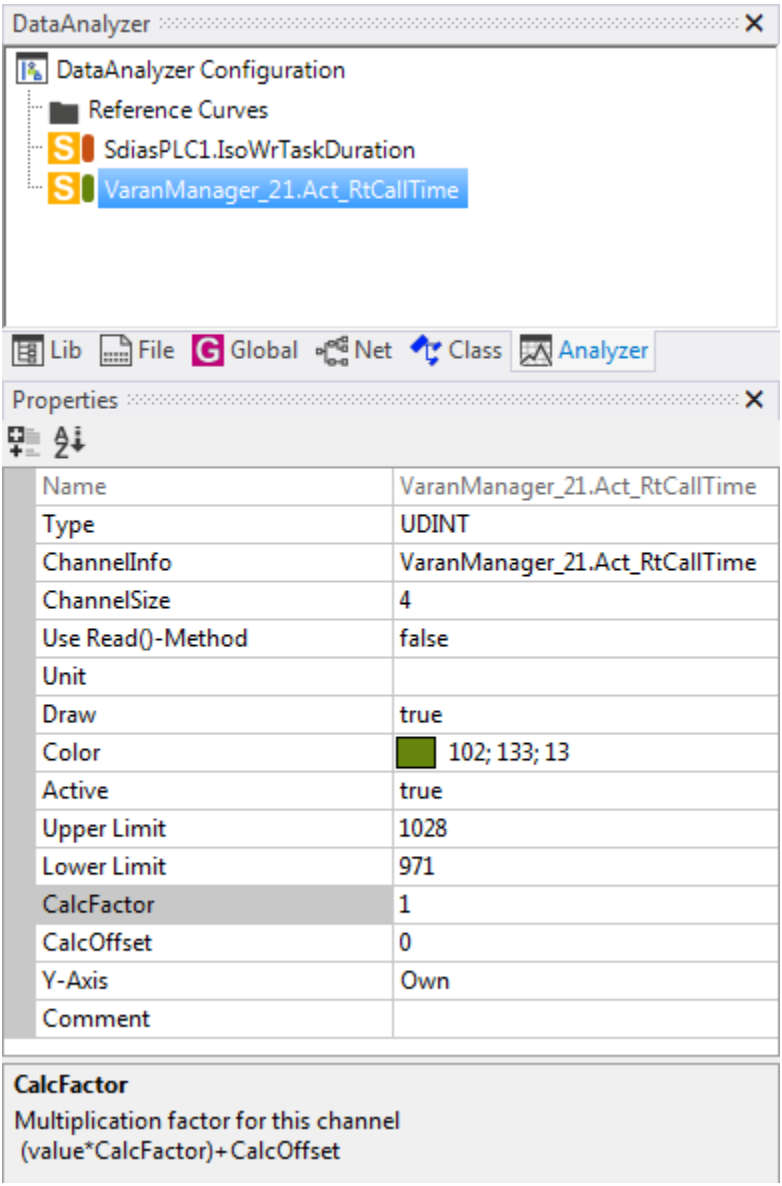
The Trigger is configured via the properties TriggerStart and TriggerStop in the Properties window of the DataAnalyzer:



Active	Activation or deactivation of the complete Trigger
Type	Display, whether the Trigger serves to start or to stop the recoding (parameter is write-protected)
Condition	Selection, which condition has to be fulfilled, so that the Trigger releases
Channel	Measuring channel that should be used for the monitoring of the Trigger
Value	Trigger level. The value compared with the measuring value via Condition
Comment	Any comment

4.7.2.3 Properties of a Measuring Channel

In the properties window the properties of the measuring channel, selected in the DataAnalyzer tree, are displayed:


DataAnalyzer
DataAnalyzer Configuration
Reference Curves
SdiasPLC1.IsoWrTaskDuration
VaranManager_21.Act_RtCallTime
Lib
File
Global
Net
Class
Analyzer
Properties
Name
VaranManager_21.Act_RtCallTime
Type
UDINT
ChannelInfo
VaranManager_21.Act_RtCallTime
ChannelSize
4
Use Read()-Method
false
Unit
Draw
true
Color
102; 133; 13
Active
true
Upper Limit
1028
Lower Limit
971
CalcFactor
1
CalcOffset
0
Y-Axis
Own
Comment
CalcFactor
Multiplication factor for this channel
(value*CalcFactor)+CalcOffset

Name	Name of the measuring channel. Can only be changed in the tree
Type	Type of the measuring value
ChannelInfo	Info, which value should be recorded. o **Server:** "Objectname.Servername" o **Client:** "Objectname.Clientname" o **Object variable:** "Objectname.Variablename" (requires loaded LASAL CLASS 2 project) o **Global variable:** "Variablename" (requires loaded LASAL CLASS 2 project) o **Storage cell:** "%mAddress"
ChannelSize	Size of the measuring value (1 byte, 2 bytes or 4 bytes)
Use Read()-Method	Determines whether or not the standard read methods for this channel should be called
Unit	A unit for each curve can be optically defined, which is shown over the Y-scale to the right. All units, including user-configured units, are available for selection.
Draw	Here, whether the curve should be visible or not is defined. A recording is always made.
Color	Color of the gradient
Active	Activation or deactivation of the recording of the selected channel
Upper Limit	Maximum value at which the Y-axis can be displayed
Lower Limit	Minimum value at which the Y-axis can be displayed
CalcFactor	Multiplication factor for this channel (value * CalcFactor) + CalcOffset
CalcOffset	Adds a constant value to this channel
Y-Axis	Here, which Y-axis scale to use as a reference can be defined: o **None:** No scale is shown for this channel o **Own:** Only the Y-axis scale is shown o **Selected channel:** The scale for the selected channel is shown

Relative Height	This property is only displayed if "Channel Display" is set to **Split** in the DataAnalyzer properties. *
Comment	Any comment to the selected channel

* **Relative Height**: When the DataAnalyzer "Channel Display" property is set to **Split**, this Channel property is displayed. It determines the height

in the main window, which is made available in split mode for the individual curve. The default value is 10. This value is not absolute, but is calculated as follows:

All "Relative Height"-values are added together and then converted to the screen height. The following example illustrates how this works:

Channel	Relative Height
Channel 1	10
Channel 2	20
Channel 3	10
results in	
Sum:	40.

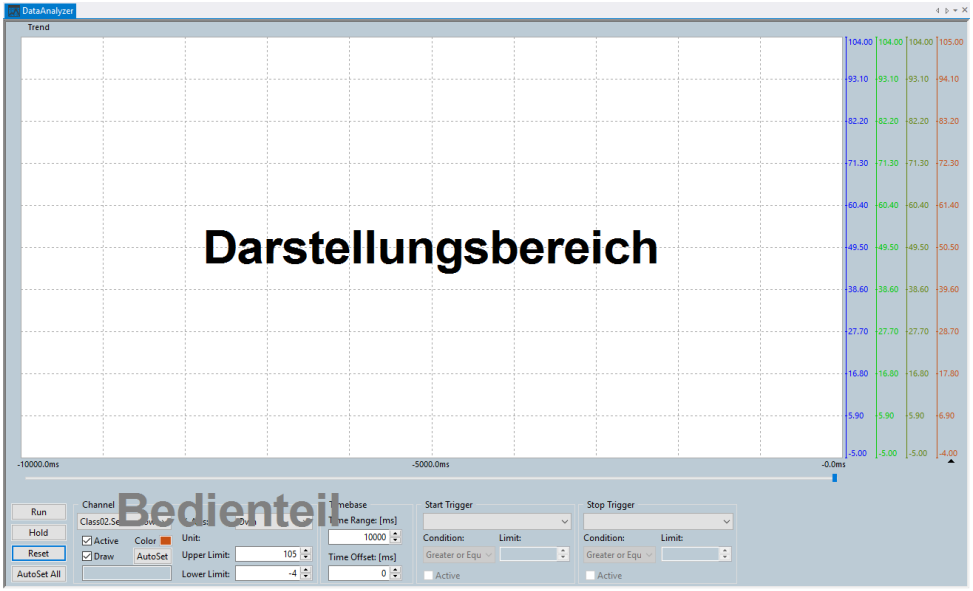
If now the free display height is 100%, channel 1 gets 100 / 40 * 10 = 25% of the height, also channel 3. Channel 2 gets 100 / 40 * 20 = 50%. If now Channel 3 is deleted, the sum of all relative heights remains 30. In this case Channel 1 gets 100 / 30 * 10 = 33,333...% and Channel 2 gets 100 / 30 * 20 = 66,666... % of the screen height.

The number of configurable measuring channels is not limited. Only the bandwidth of the transfer may require the reduction of the channels.

4.7.2.4 Saving and Loading the Configuration

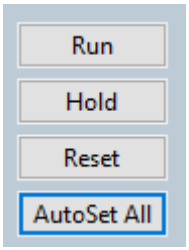
At start of the DataAnalyzer the last made configuration is used automatically. Via the **DataAnalyzer** menu entry a configuration can be saved to any location (**Save Configuration As**), respectively, loaded again from there (**Load Configuration**).

4.7.3 Recording with the DataAnalyzer window



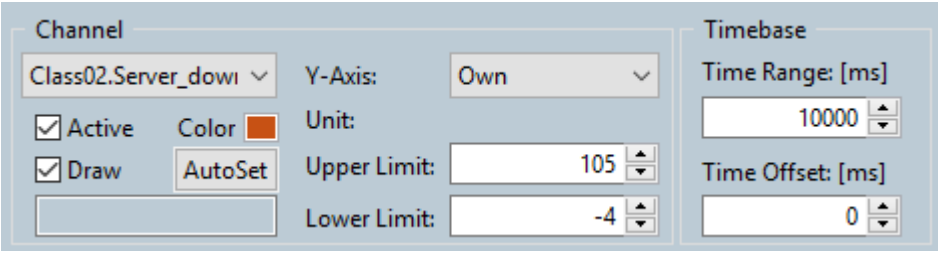
4.7.3.1 Control Unit

On the left border 4 buttons are positioned. These serve to control different functionalities of the DataAnalyzer:



- **Run:** Starts the recording. During recording the button description and functionality change to **Stop**, which serves to stop the recording.

- **Hold:** Pauses the update of the diagram in the meantime. The recording goes on in the background. After activation of this button its description changes to Resume. At activation of the **Resume** button the curves are updated again.
- **Reset:** Deletes all recorded measuring values.
- **AutoSet All:** Positions all recorded curves on the visible area of the display (automatically configuration of **Resolution** and Offset of each channel). Following the area for measuring channel configuration is situated. Via the **Channel select list** a certain channel can be selected. The selected channel can then be set via **check boxes**:

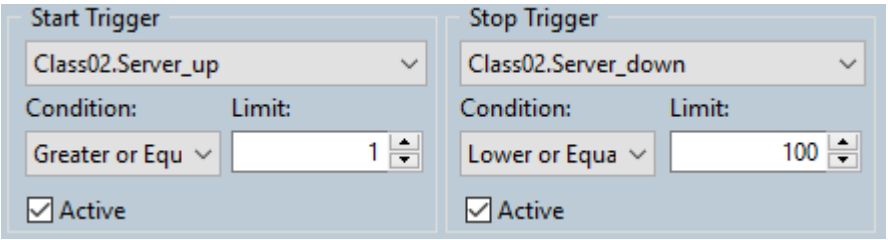


- **Active:** If the channel should be recorded (taken on at start of the recording!)?
- **Draw:** Gradient is displayed. However, at deactivated check box the measuring values are recorded.
- **Color:** Only shows, in which color the gradient is drawn. Not here, but in the properties window the color can be set.
- **AutoSet:** Automatically adaptation of the gradient (only this channel).
- **Upper Limit / Lower Limit:** Resolution setting of the Y-axis.

Furthermore there are functionalities to adjust the time base (X-axis):

- **Time Range [ms]:** Resolution setting of the X-axis.
- **Time Offset [ms]:** Setting of the X-axis offset

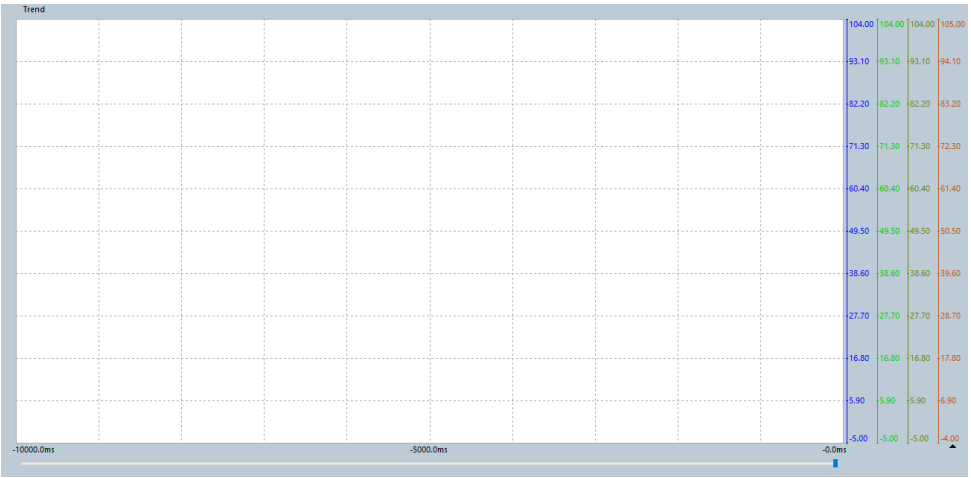
The third group is used to configure the trigger (start and stop trigger):



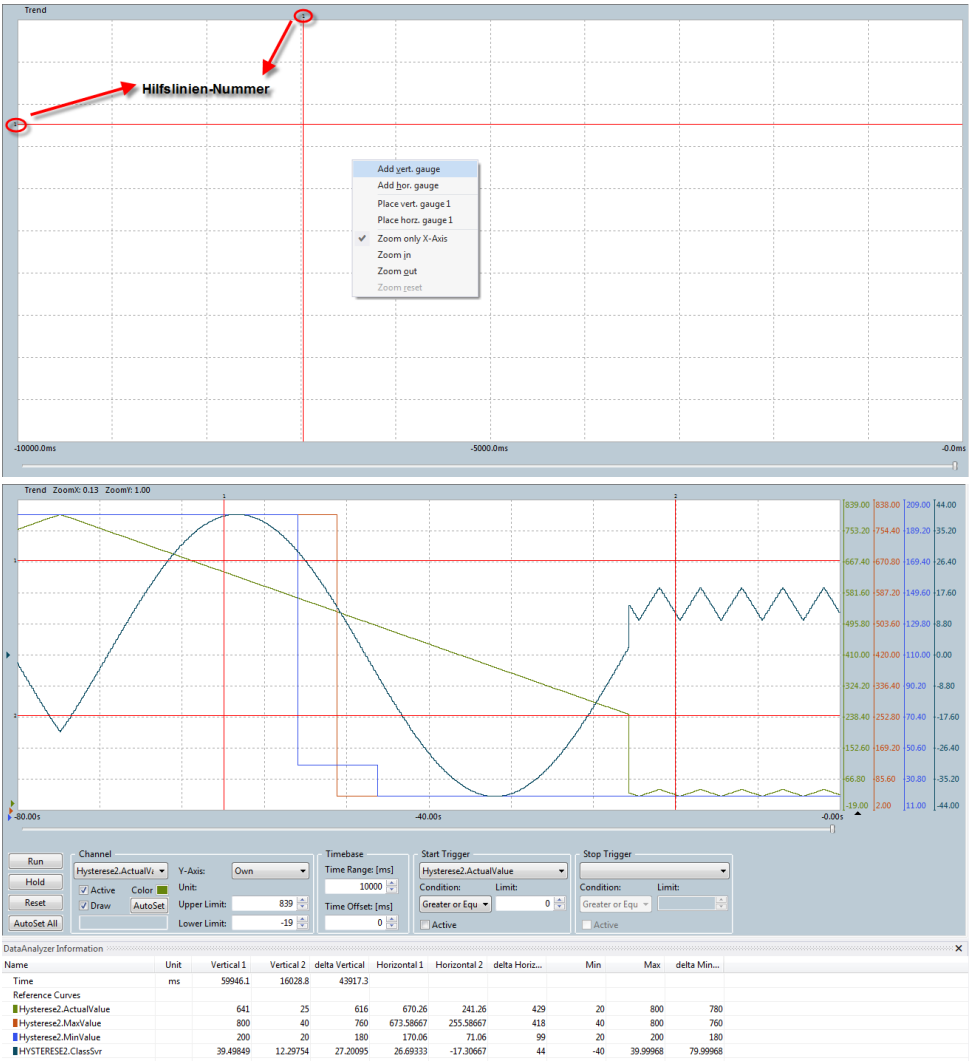
After setting the measurement channel that should be monitored (here: Channel0), condition: and the desired size (Limit:) can be set. With the control box Active, all triggers can be activated or deactivated.

4.7.3.2 Display Area

The display area is situated above the control unit and takes the superior part of the main view. The complete display area is divided into **10 horizontal** and **10 vertical** sections for better orientation:



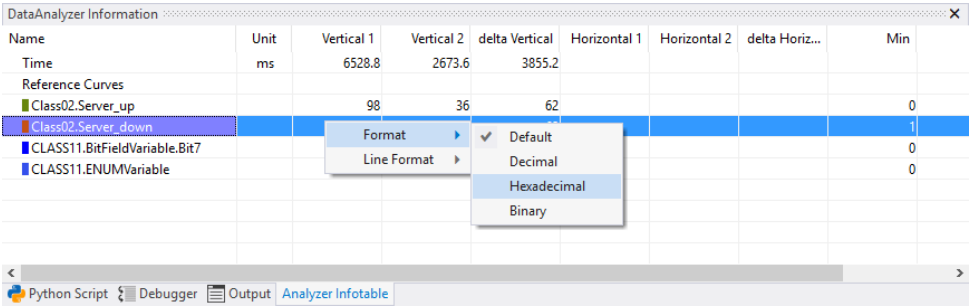
For a better definition of the measuring values 2 vertical and 2 horizontal help lines marked with numbers can be fade in through the context menu of the display area:



Name	Unit	Vertical 1	Vertical 2	delta Vertical	Horizontal 1	Horizontal 2	delta Horiz...	Min	Max	delta Min...
Time	ms	59946.1	16028.8	43917.3						
Reference Curves										
Hysterese2.ActualValue		641	25	616	670.26	241.26	429	20	800	780
Hysterese2.MaxValue		800	40	760	673.58667	255.58667	418	40	800	760
Hysterese2.MinValue		200	20	180	170.06	71.06	99	20	200	180
HYSTERESE2.ClassSvr		39.49849	12.29754	27.20095	26.69333	-17.30667	44	-40	39.99968	79.99968

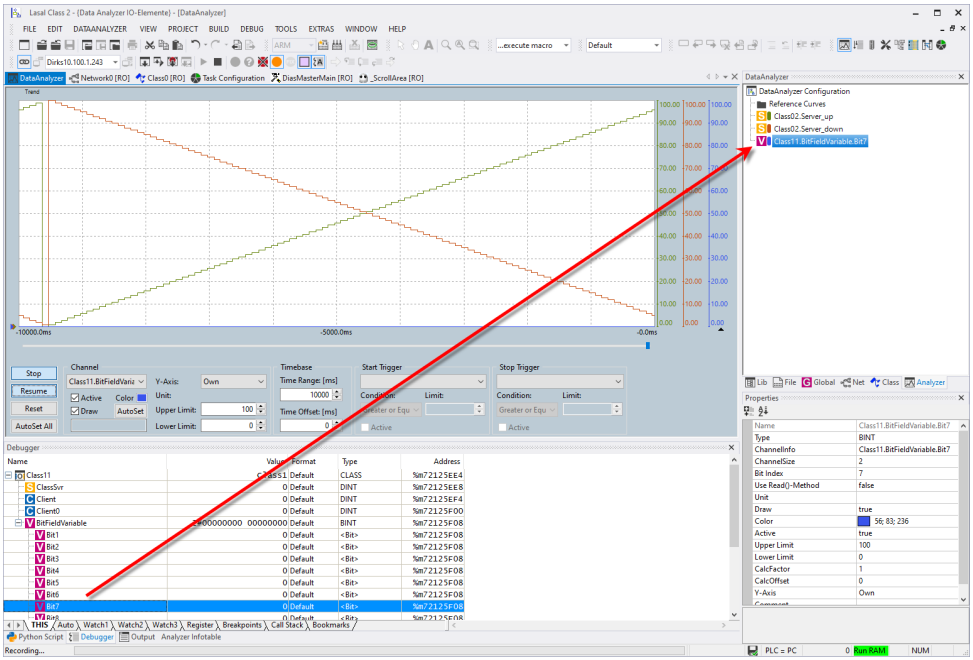
In the **DataAnalyzer Infotable**, the results can be monitored. The values of the reference curve can also be displayed. The channels are marked by color (corresponding to the DataAnalyzer tree). Each measurement unit is placed in a separate column. All values from this table can be copied to the clipboard with Ctrl+C.

With a right-click on one of the displayed fields, the context menu is called:

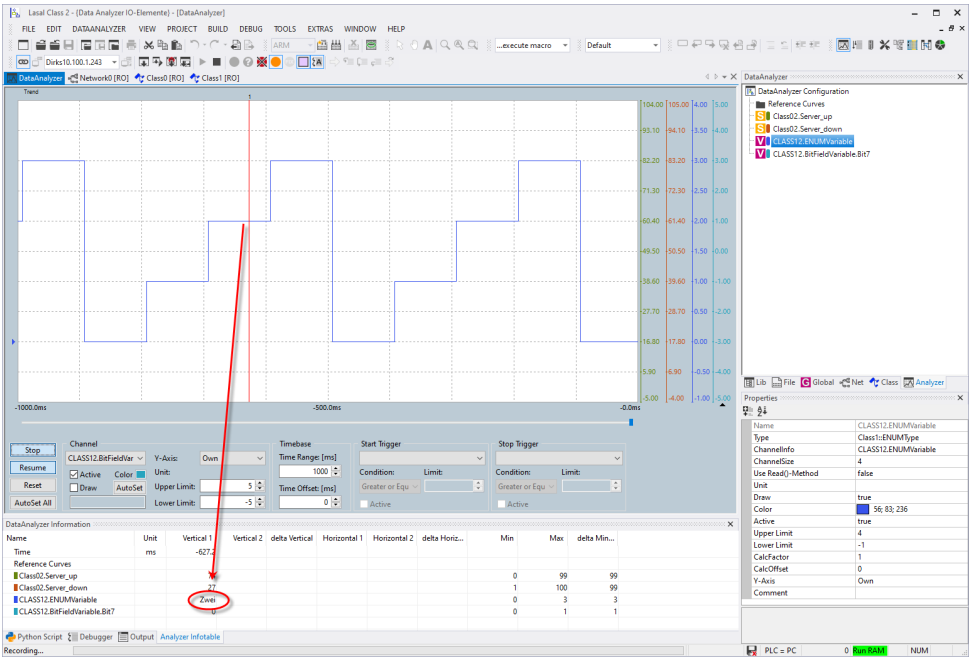


Here the option is available to define the display format of the corresponding field (Format) or all fields on the line (Line Format). In addition to "Default", three options are available: Decimal, Hexadecimal and Binary.

It is also possible to record individual bits; the variable must be of type BINT, BSINT or BDINT. In the Debugger, the variable can be opened and the bit placed in the Data Analyzer tree:



If channel(s) or type ENUM are recorded and the user moves over the recorded curve in the display area with the vertical help line, so that the actual value corresponds to a defined ENUM item, the value is shown in the DataAnalyzer as a character string.

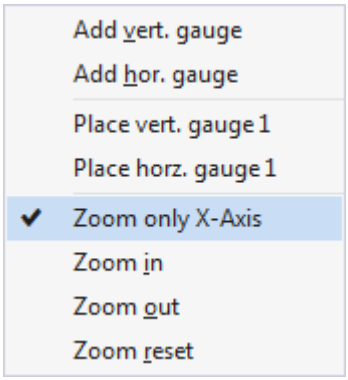


The moment of the measurement is displayed at the **vertical artificial line**. If the artificial line overlaps one or more gradients, then the sample values are displayed (till the next measuring point):

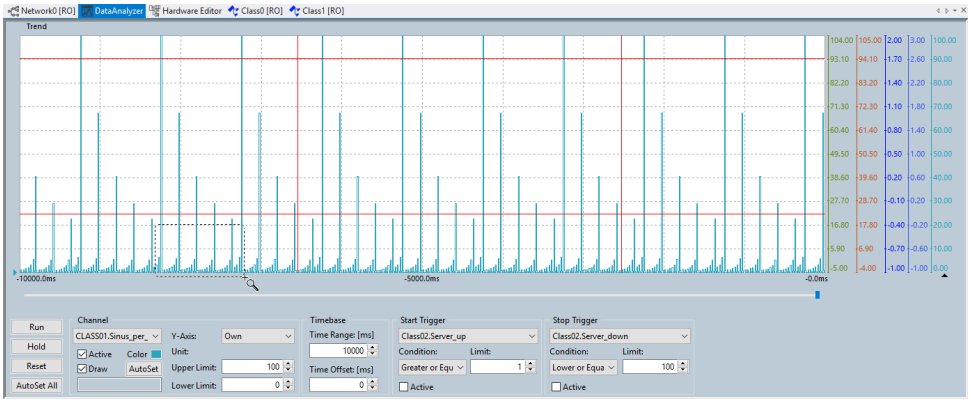
The **horizontal artificial lines** always show the mathematical value of each channel. At two horizontal artificial lines, the offset between both lines for the channel is calculated additionally.

Zooming the display area

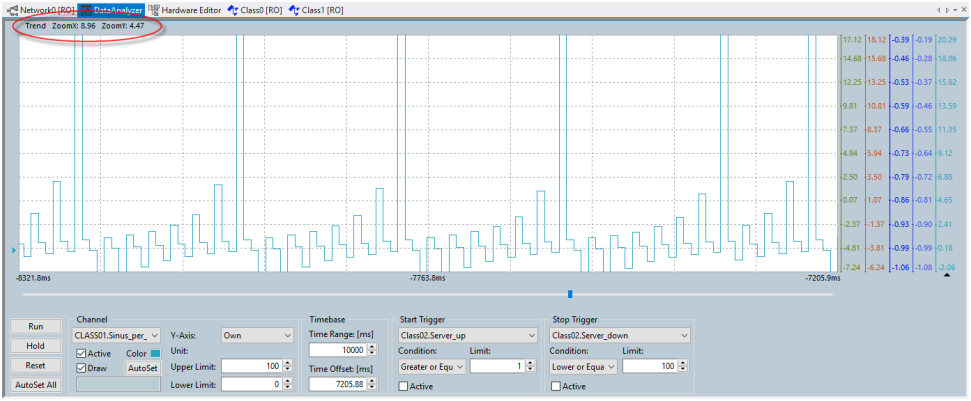
A small area of the traced curve can be zoomed, by which one simple used the mouse to expand the desired field. In the context menu you can select, whether only the X axis (default) or both the x as well as the Y axis should be zoomed.



With the mouse, click on the upper left corner of the desired section and move the mouse with the button pressed to form a rectangle, and then release the mouse button:



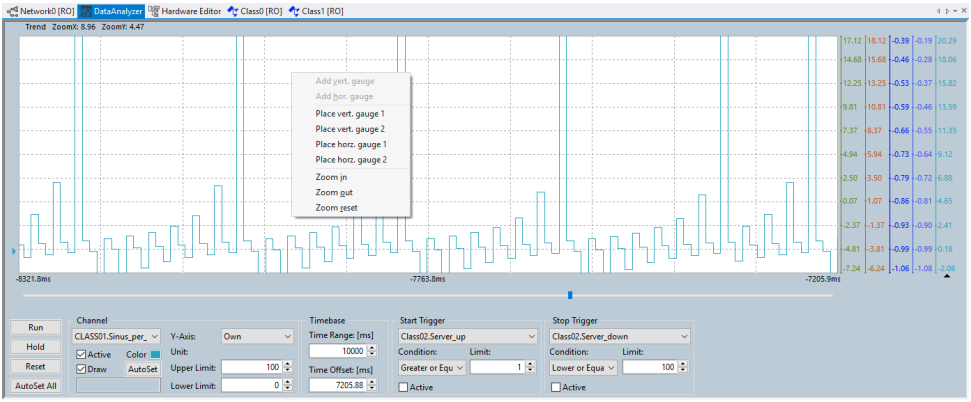
The specified section is then shown:



In the upper left area, the zoom factors (ZoomX, ZoomY) are displayed. In this example the context menu entry “Zoom only X-Axis“ has **not** been selected.

Zoom only X-Axis, Zoom In, Zoom out, Zoom reset

With a right mouse click in the display area, additional functions can be found.



Zoom only X-Axis

When zoomoing, with this function – with free zooming like described above as well as with the menu entries “Zoom-In” and “Zoom-Out” – you can choose between “Zoom only

X-Axis" (menu entry selected = default) or "Zoom X- and Y-Axis" (menu entry not selected).

Zoom In

With this function, the selected display can be zoomed further by double the actual value.

Zoom Out

With this function, the display is reduced.

Zoom Reset

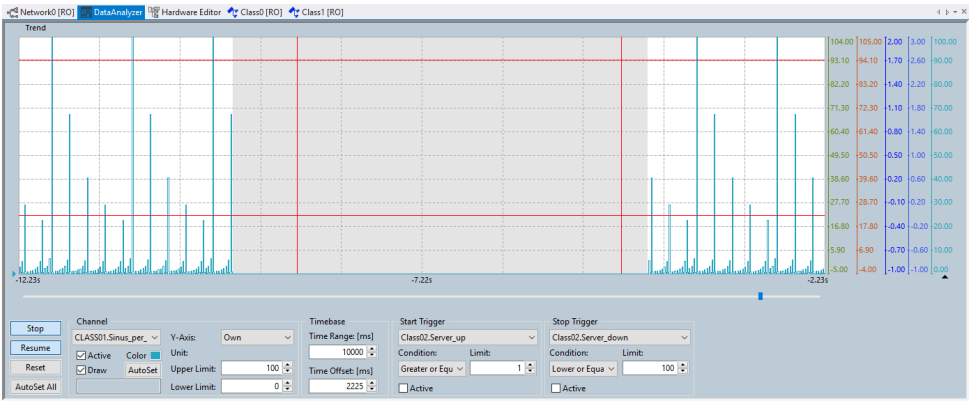
This function is used to return to the original display size.

4.7.3.3 Start Measuring

Through pressing the **Run button** in the **control unit** of the DataAnalyzer the recording of the measuring values is started. However, before start of the recording at least one measuring channel has to be configured and an online connection to the desired control has to be existent. At incorrect setting of a channel (e.g. client name has changed), it will not be recorded.

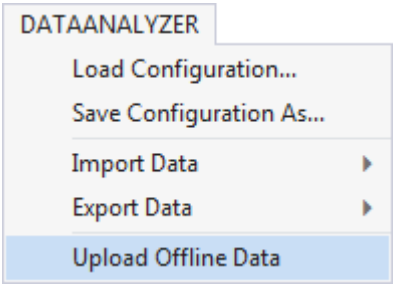
If a **Start Trigger** has been set, the recording starts not until it has been released.

If no values are recorded for a time – e.g. because the online connection was disrupted or due to a buffer overflow - the corresponding area in the the curve display is marked in grey:



4.7.3.4 Measuring and/or Recording in OFFLINE Status

The DataAnalyzer can continue to record measurement values although it is temporarily offline with the control. Data is temporarily stored in a ring buffer in the control. With the menu command Upload Offline Data, the data in the DataAnalyzer is reloaded into Lasal after the online connection has been restored. Retrieving the values is also possible during Reset. This function is recommended for slow sample rates only, as otherwise the ring buffer is immediately filled or overwritten.



Data recording response in the ring buffer

- No trigger set: ring buffer is continuously overwritten
- StartTrigger set: buffer is filled once then recording stops
- StopTrigger set only: buffer is overwritten up to the trigger

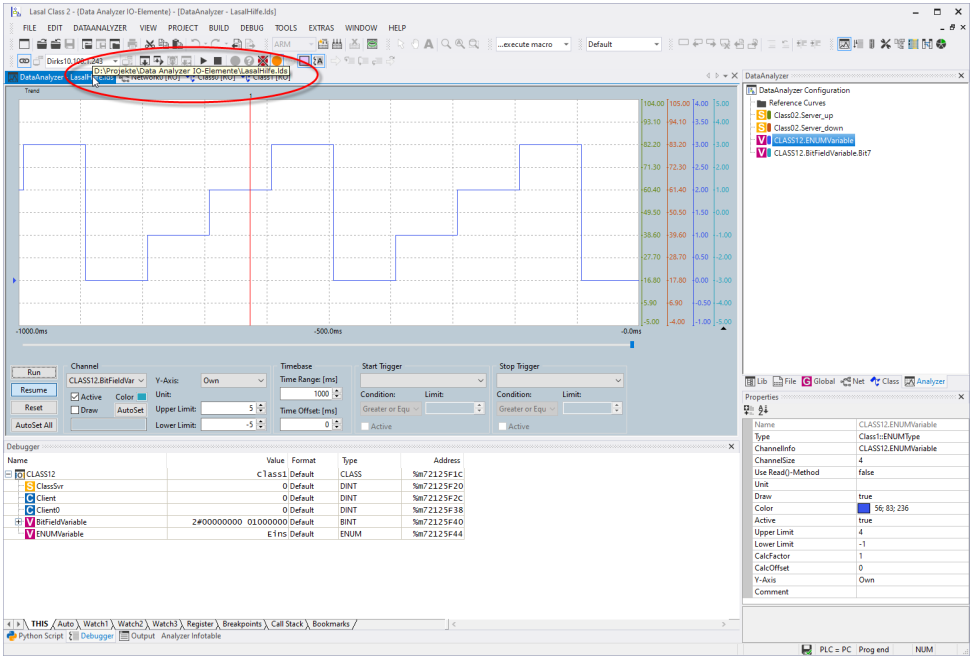
4.7.3.5 Saving and Loading Measuring Values

Through selecting **DataAnalyzer, Export Data** in the menu bar, the recorded measuring values can be saved to a file. Thereby the single values are separated by a comma (decimal comma is displayed as dot).

In the first line the names of the particular channels are listed after the sample rate. In the first column the index of the measurement is listed (at transfer troubles, some lines could be ignored).

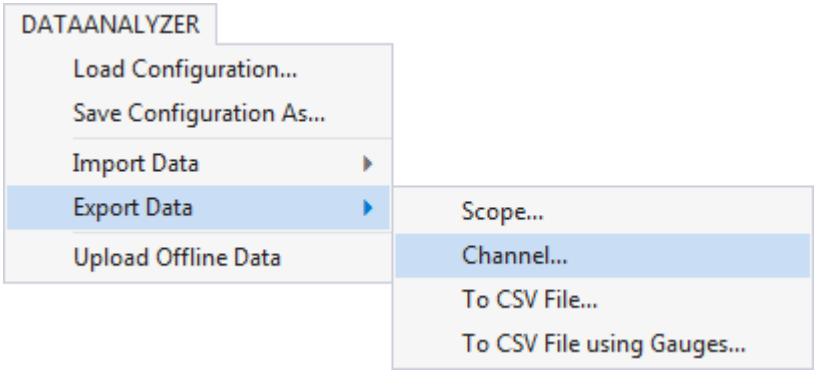
The menu entry **DataAnalyzer, Import** is used to load saved data. Thereby attention has to be paid to load the correct configuration of the DataAnalyzer (assignment of values). The loaded values are graphically displayed automatically.

If a Scope file is saved or just loaded, the file name is shown in the tab and when the mouse cursor is placed over the tab, the complete path is shown as a tool tip.

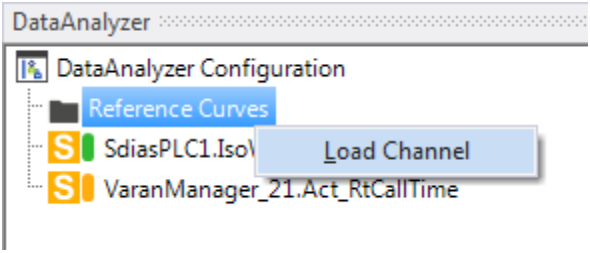


4.7.3.6 Importing / Exporting Data

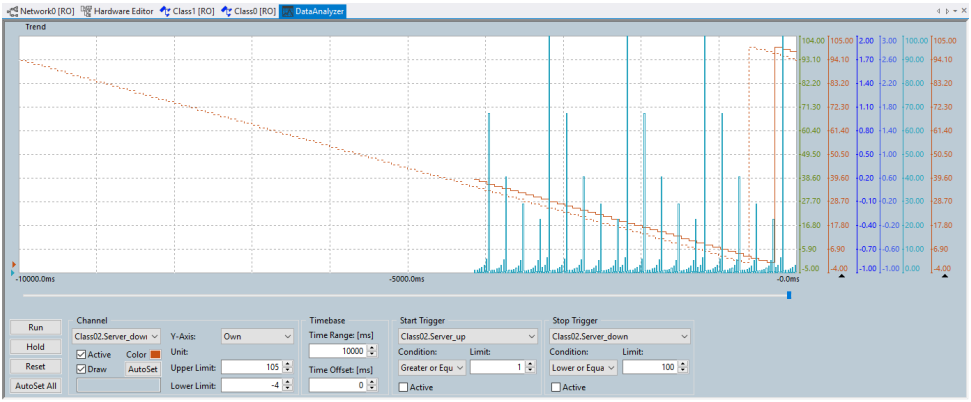
Using the menu item **DataAnalyzer/Export Data/Channel...**, it is possible to export channel data,…



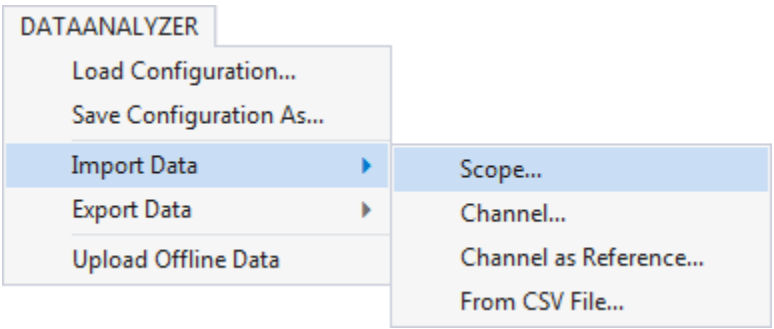
...which can later be loaded as a **reference curve**.



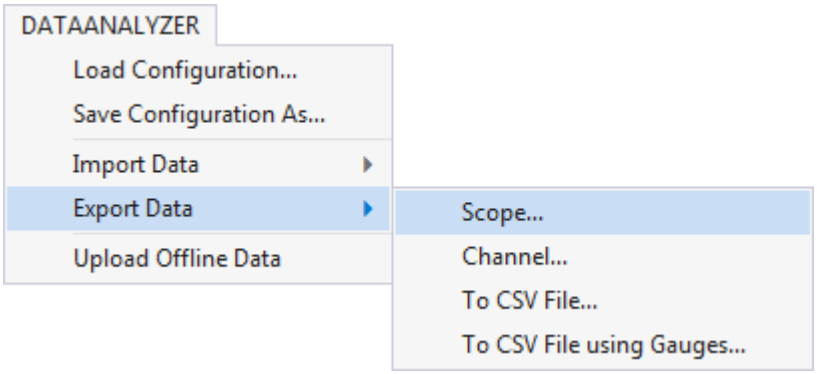
The reference curves are shown as dotted lines in the DataAnalyzer and can be compared with the recently recorded values.



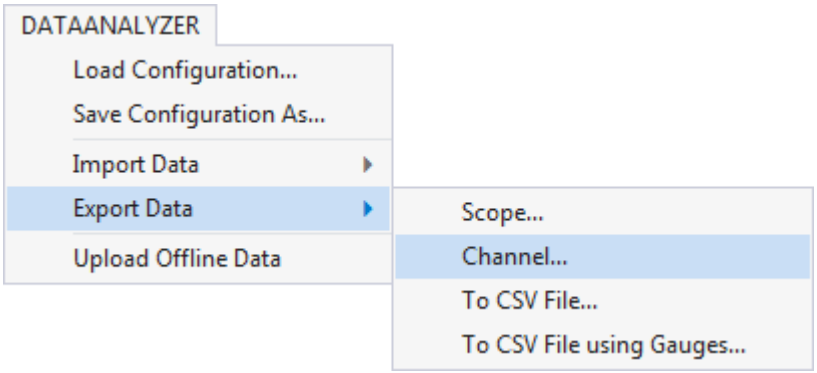
Scope files can be imported and exported. If "**Import Data/Scope...**" is selected, all data is imported.



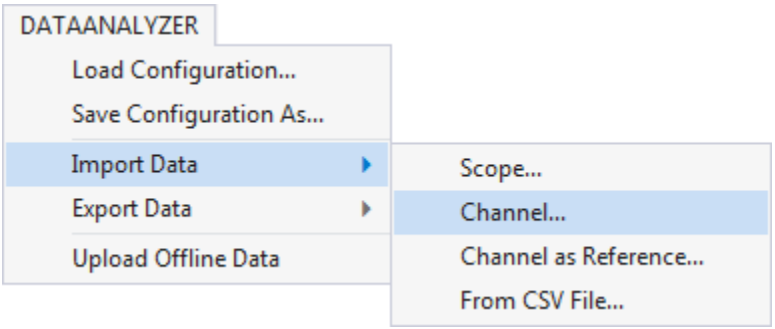
The same is **true** for "**Export Data/Scope...**"



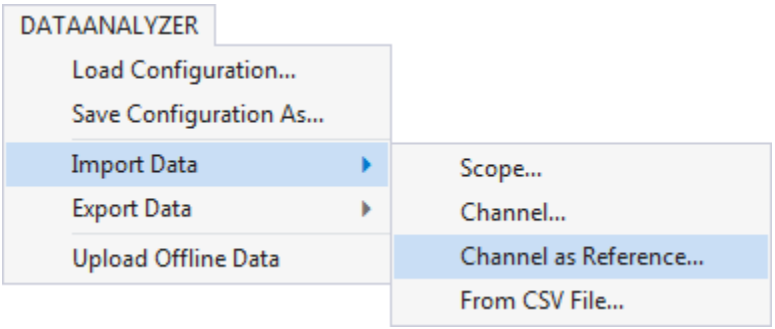
If Export "**Data/Channel...**" is selected however, only the channel data is exported.



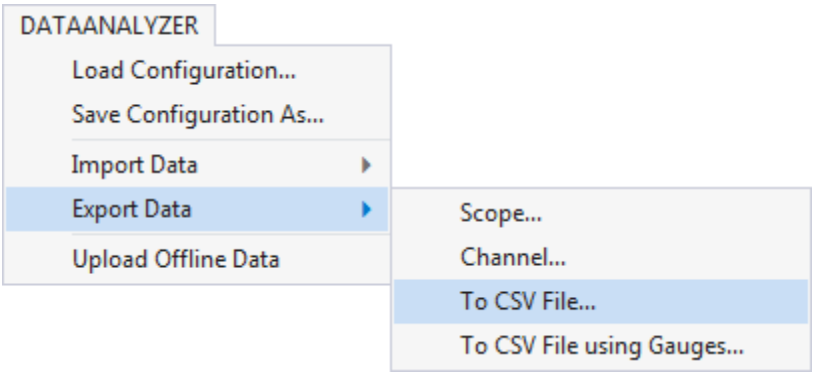
If the "**Import Data/Channel...**"the channel is imported as a "normal channel and is continuously traced – the curve disappears from the display when in the control field, "**Reset**" is pressed.



When "**Import Data Channel as Reference...**" is selected however, the imported channel is entered as a reference curve (corresponding to the context menu item "Load Channel") as described earlier in this section. In this case, the dotted curve is displayed even when "**Reset**" is clicked.

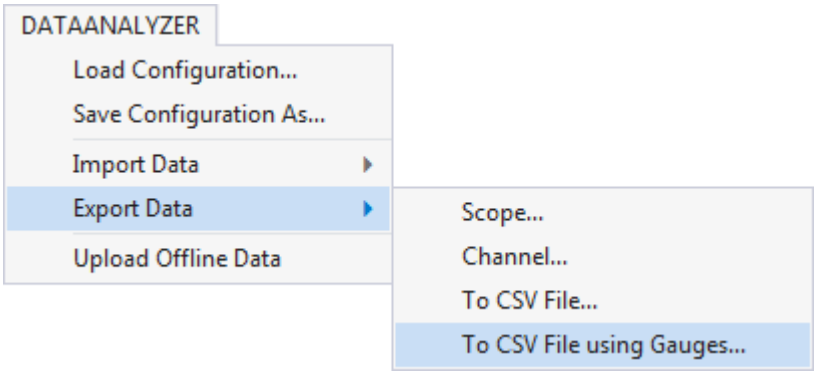


Scope data can also be exported as CSV files.



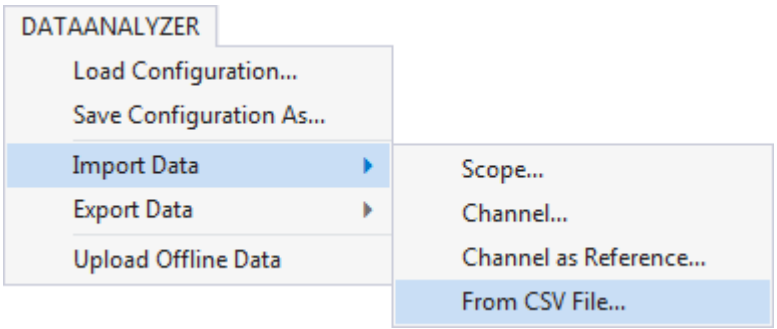
If "**Export Data/To CSV File using Gauges...**" is used to export, the following applies:

There is no measurement line	Everything is saved
A measurment line is given	The measurement line is the start up to the actual value
There are two measurement lines	Measurement 1 line as start, measurement line 2 as stop



The index of the measurement values is maintained. (Measurement data in the CSV file don't necessarily begin with index 1!)

These – or manually created – CSV files can also be imported.

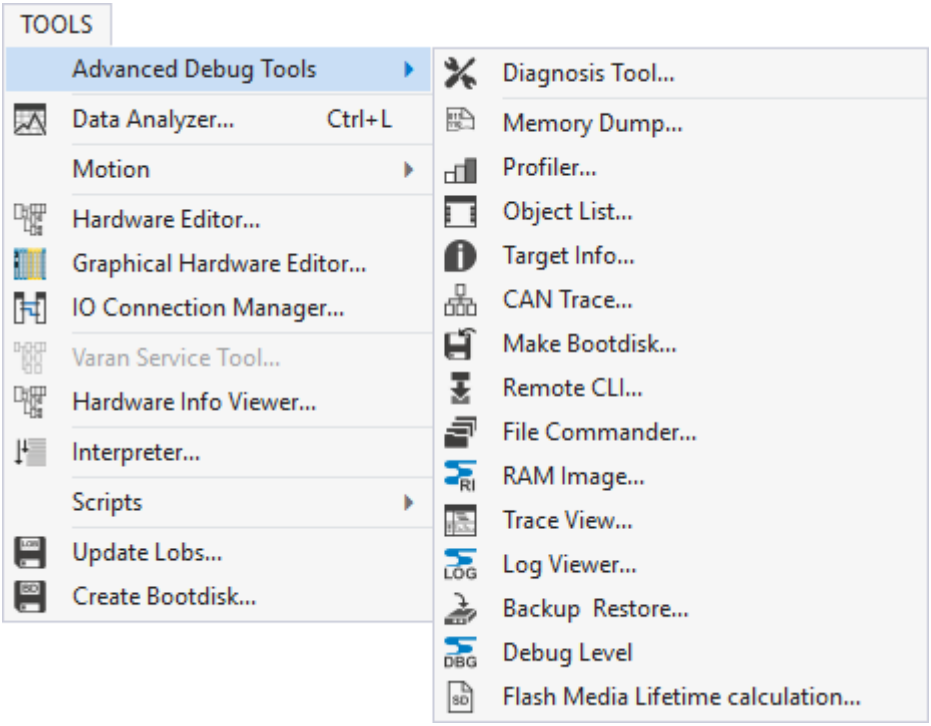


4.8 Advanced Debug Tools

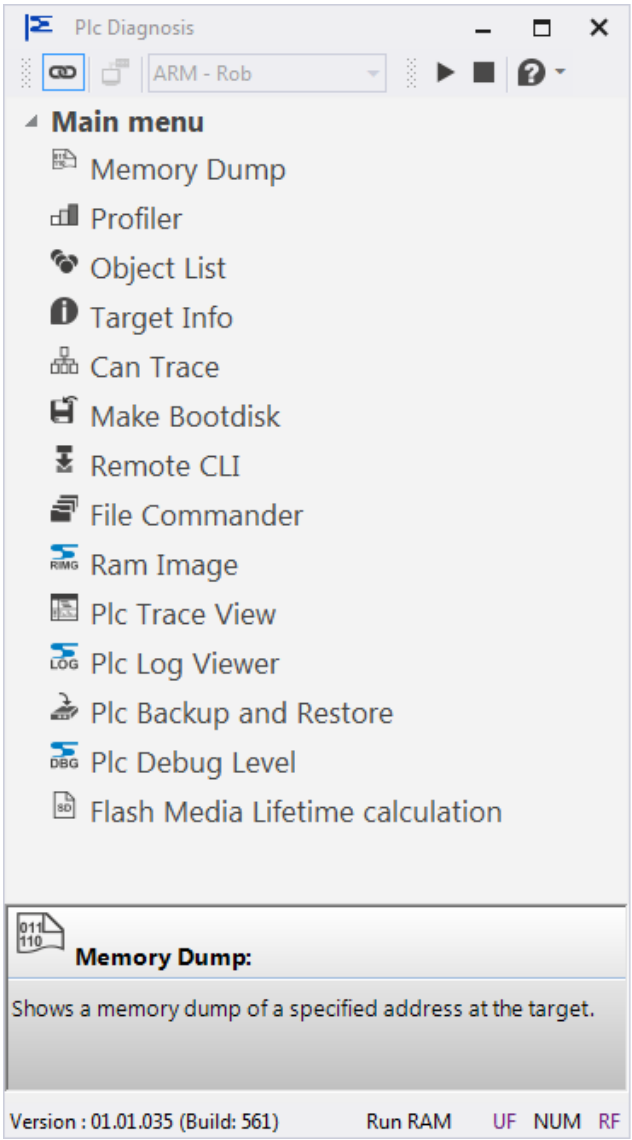
With **Advanced Debug Tools** the user gets a manifold tool for managing, monitoring and analyzing the control.

4.8.1 Accessibility

The menu entry **Tools → Advanced Debug Tools** opens a submenu. First **Diagnosis Tool...** can be selected.



Select this menu entry to open the included **Plc Diagnosis Tool** application:

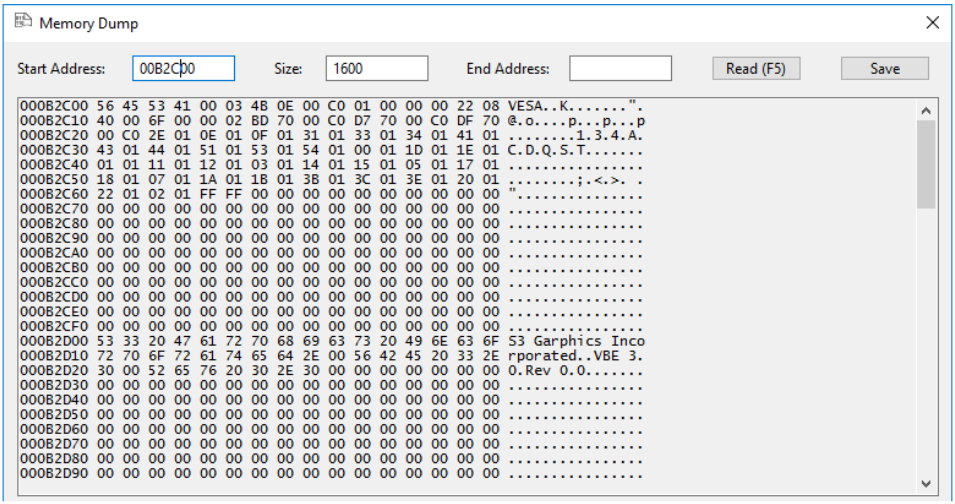

Plc Diagnosis
ARM - Rob
Main menu
Memory Dump
Profiler
Object List
Target Info
Can Trace
Make Bootdisk
Remote CLI
File Commander
Ram Image
Plc Trace View
Plc Log Viewer
Plc Backup and Restore
Plc Debug Level
Flash Media Lifetime calculation
Memory Dump:
Shows a memory dump of a specified address at the target.
Version : 01.01.035 (Build: 561)
Run RAM
UF
NUM
RF

All of the following menu entries are also available in the PLC Diagnosis program package. This can also be opened through a mouse click on the Toolbar icon  .

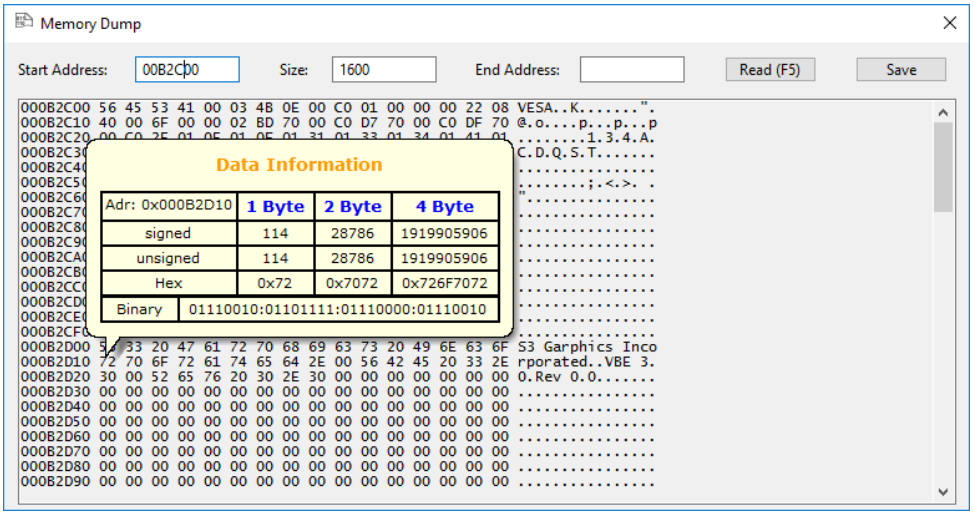
4.8.2 The Individual Debug Tools

4.8.2.1 Memory Dump

Shows the memory image with the predefined start address and size. The Dump can be stored in a text file.

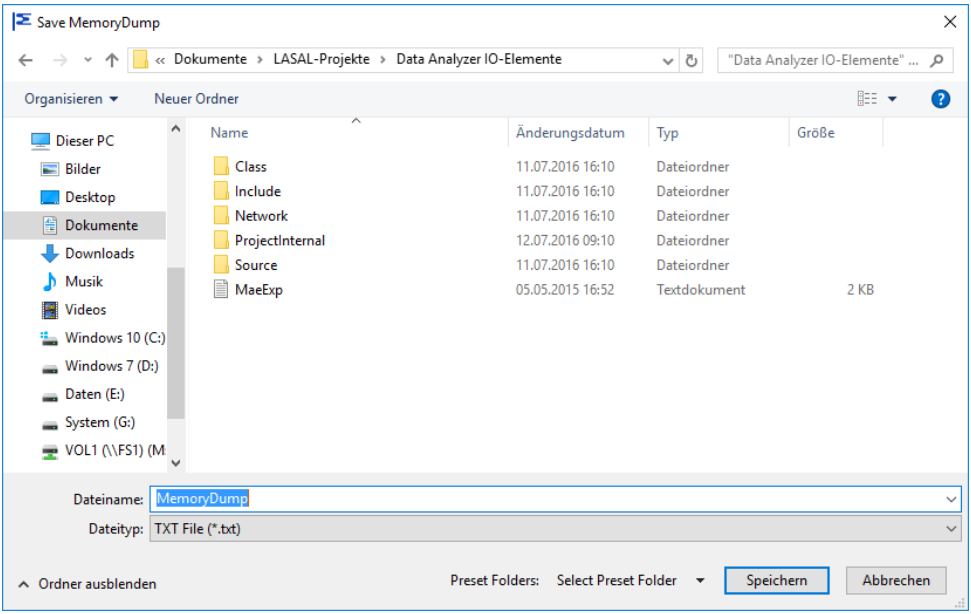


If the user stops on a memory address with the mouse, the data value of the memory for 1, 2 and 4-byte lengths signed, unsigned, hexadecimal and binary are shown in an information tool tip.



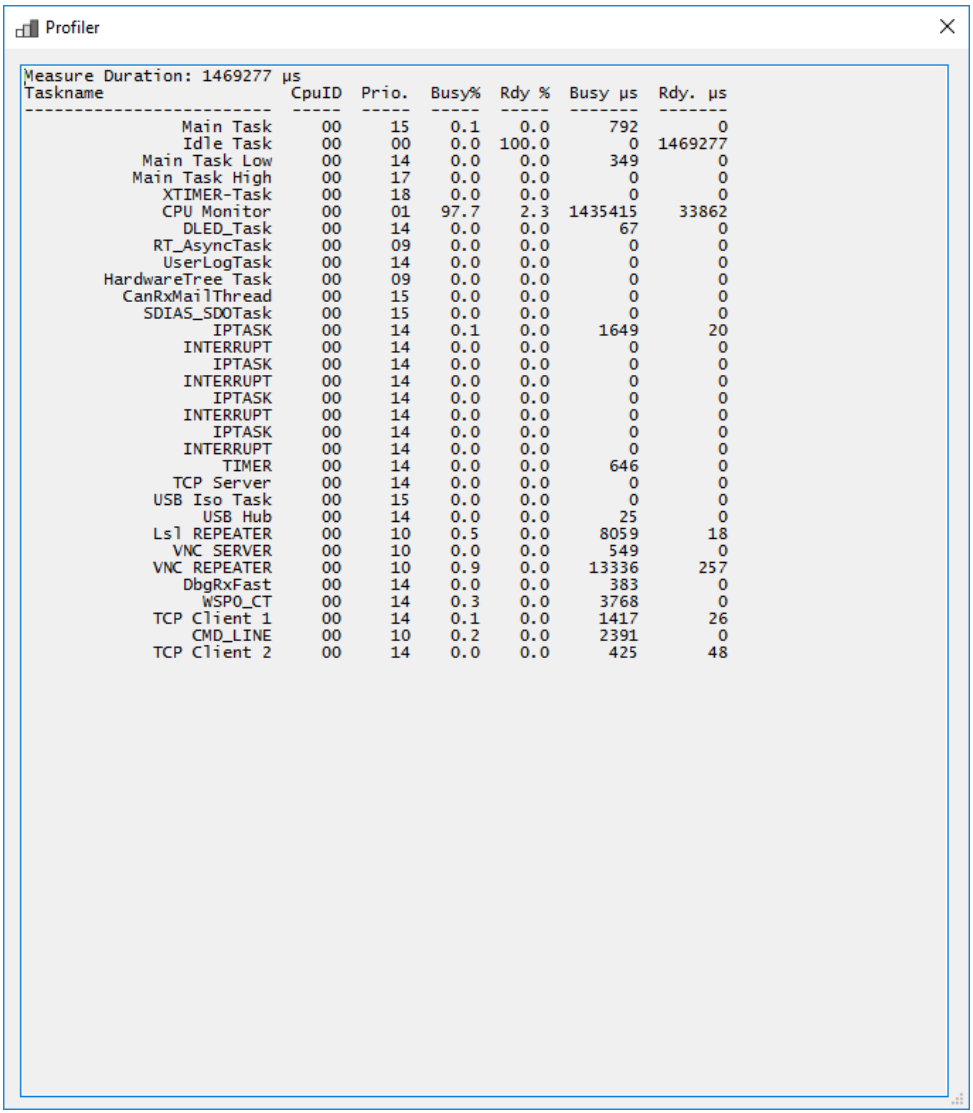
Instead of the memory size also an End address can be defined.

Weiters besteht die Möglichkeit mit dem Button **Save** den Memorydump in ein Text-File abzuspeichern.


Save MemoryDump
Dokumente › LASAL-Projekte › Data Analyzer IO-Elemente
"Data Analyzer IO-Elemente" ...
Organisieren
Neuer Ordner
Dieser PC
Bilder
Desktop
Dokumente
Downloads
Musik
Videos
Windows 10 (C:)
Windows 7 (D:)
Daten (E:)
System (G:)
VOL1 (\\FS1) (M
Name
Änderungsdatum
Typ
Größe
Class 11.07.2016 16:10 Dateiordner
Include 11.07.2016 16:10 Dateiordner
Network 11.07.2016 16:10 Dateiordner
ProjectInternal 12.07.2016 09:10 Dateiordner
Source 11.07.2016 16:10 Dateiordner
MaeExp 05.05.2015 16:52 Textdokument 2 KB
Dateiname: MemoryDump
Dateityp: TXT File (*.txt)
Ordner ausblenden
Preset Folders: Select Preset Folder
Speichern
Abbrechen

4.8.2.2 Profiler

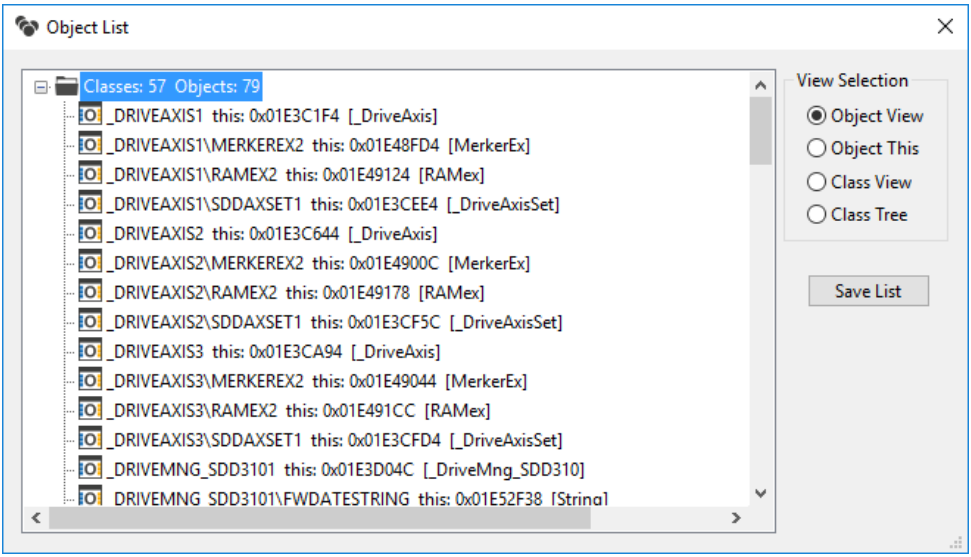
Dynamic task record of the target (updated approximately every 3 seconds).



Taskname	CpuID	Prio.	Busy%	Rdy %	Busy µs	Rdy. µs
Main Task	00	15	0.1	0.0	792	0
Idle Task	00	00	0.0	100.0	0	1469277
Main Task Low	00	14	0.0	0.0	349	0
Main Task High	00	17	0.0	0.0	0	0
XTIMER-Task	00	18	0.0	0.0	0	0
CPU Monitor	00	01	97.7	2.3	1435415	33862
DLED_Task	00	14	0.0	0.0	67	0
RT_AsyncTask	00	09	0.0	0.0	0	0
UserLogTask	00	14	0.0	0.0	0	0
HardwareTree Task	00	09	0.0	0.0	0	0
CanRxMailThread	00	15	0.0	0.0	0	0
SDIAS_SDOTask	00	15	0.0	0.0	0	0
IPTASK	00	14	0.1	0.0	1649	20
INTERRUPT	00	14	0.0	0.0	0	0
IPTASK	00	14	0.0	0.0	0	0
INTERRUPT	00	14	0.0	0.0	0	0
IPTASK	00	14	0.0	0.0	0	0
INTERRUPT	00	14	0.0	0.0	0	0
IPTASK	00	14	0.0	0.0	0	0
INTERRUPT	00	14	0.0	0.0	0	0
TIMER	00	14	0.0	0.0	646	0
TCP Server	00	14	0.0	0.0	0	0
USB Iso Task	00	15	0.0	0.0	0	0
USB Hub	00	14	0.0	0.0	25	0
Lsl REPEATER	00	10	0.5	0.0	8059	18
VNC SERVER	00	10	0.0	0.0	549	0
VNC REPEATER	00	10	0.9	0.0	13336	257
DbgRxFast	00	14	0.0	0.0	383	0
WSPO_CT	00	14	0.3	0.0	3768	0
TCP Client 1	00	14	0.1	0.0	1417	26
CMD_LINE	00	10	0.2	0.0	2391	0
TCP Client 2	00	14	0.0	0.0	425	48

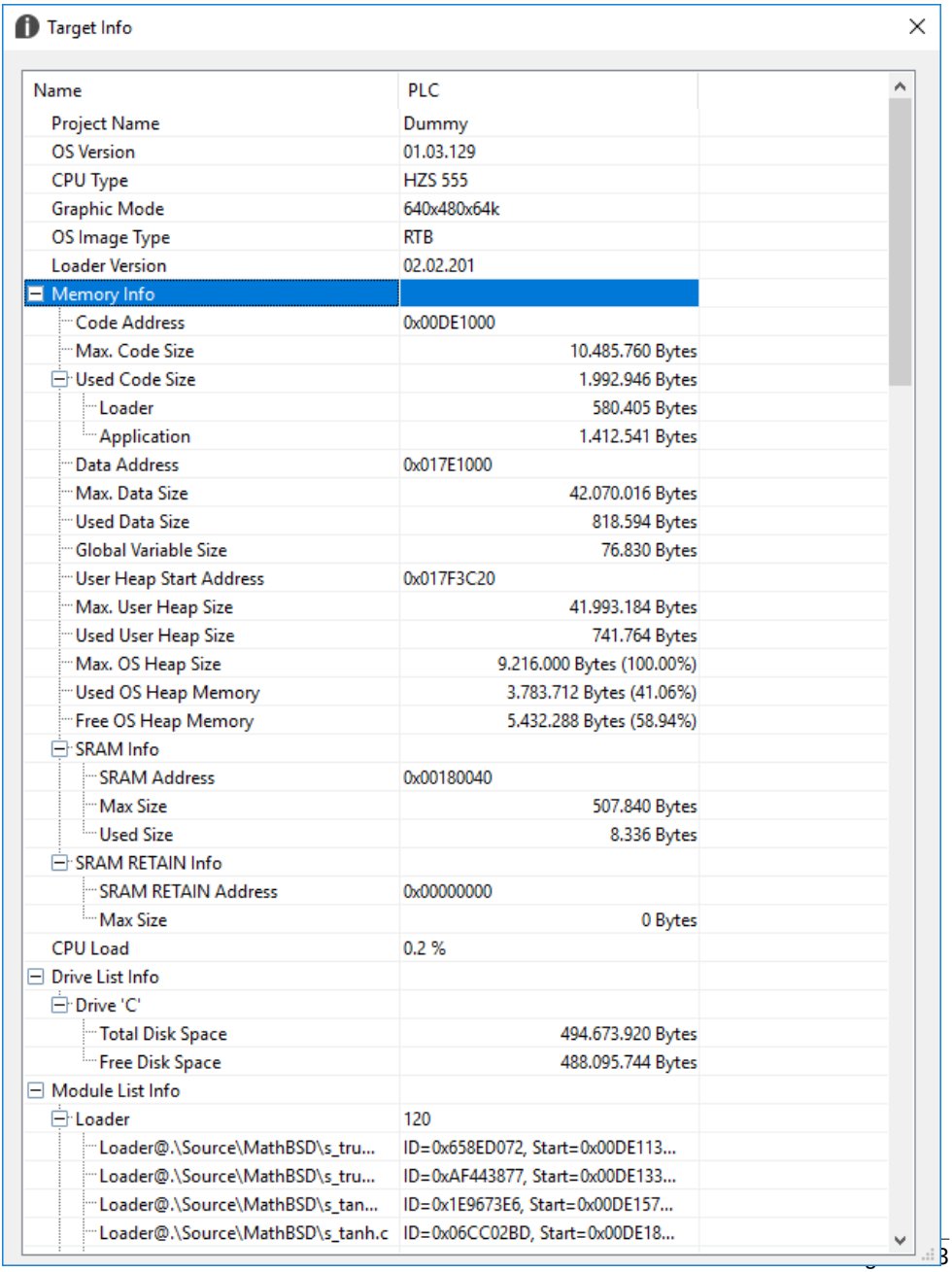
4.8.2.3 Object List

Shows all objects and classes located in the target, including their corresponding addresses.



4.8.2.4 Target Info

General information on the target system such as versions and memory overview.


Target Info
Name
PLC
Project Name
Dummy
OS Version
01.03.129
CPU Type
HZS 555
Graphic Mode
640x480x64k
OS Image Type
RTB
Loader Version
02.02.201
Memory Info
Code Address
0x00DE1000
Max. Code Size
10.485.760 Bytes
Used Code Size
1.992.946 Bytes
Loader
580.405 Bytes
Application
1.412.541 Bytes
Data Address
0x017E1000
Max. Data Size
42.070.016 Bytes
Used Data Size
818.594 Bytes
Global Variable Size
76.830 Bytes
User Heap Start Address
0x017F3C20
Max. User Heap Size
41.993.184 Bytes
Used User Heap Size
741.764 Bytes
Max. OS Heap Size
9.216.000 Bytes (100.00%)
Used OS Heap Memory
3.783.712 Bytes (41.06%)
Free OS Heap Memory
5.432.288 Bytes (58.94%)
SRAM Info
SRAM Address
0x00180040
Max Size
507.840 Bytes
Used Size
8.336 Bytes
SRAM RETAIN Info
SRAM RETAIN Address
0x00000000
Max Size
0 Bytes
CPU Load
0.2 %
Drive List Info
Drive 'C'
Total Disk Space
494.673.920 Bytes
Free Disk Space
488.095.744 Bytes
Module List Info
Loader
120
Loader@.\Source\MathBSD\s_tru...
ID=0x658ED072, Start=0x00DE113...
Loader@.\Source\MathBSD\s_tru...
ID=0xAF443877, Start=0x00DE133...
Loader@.\Source\MathBSD\s_tan...
ID=0x1E9673E6, Start=0x00DE157...
Loader@.\Source\MathBSD\s_tanh.c
ID=0x06CC02BD, Start=0x00DE18...

4.8.2.5 Can Trace

The tool allows monitoring the CAN bus communication. In the area **Interface** you can select between BU105 and BU106 depending on the existing hardware resp. the LPT socket can be defined. Attention: The driver for BU105 and BU106 must be installed herefore! Additionally the **Baudrate** can be selected in the according field. It is also possible to filter the CAN packets. The filter mask can only be used with the BU105 and can be set to a maximum of 0x7EF. Mask and filter are connected with a logical AND. With the checkbox **Invert Filter** the filter can be inverted, with the button **Filter Result** it can be **used** for the values shown in the table.

The entries are listed in the table at the bottom left. The column **ID** shows the CAN packet ID, **Len** the length of the packet. In the column **Data** you find the data (max. 8 bytes), to the right under **Time Stamp** there is the time stamp of the recording. This starts at 0x0000 and is increased to 0xEA5F before it is reset to 0x0000. The value 0xEA5F corresponds to exactly 60,000 milliseconds, i.e. one minute. This should be more than sufficient for most systems.

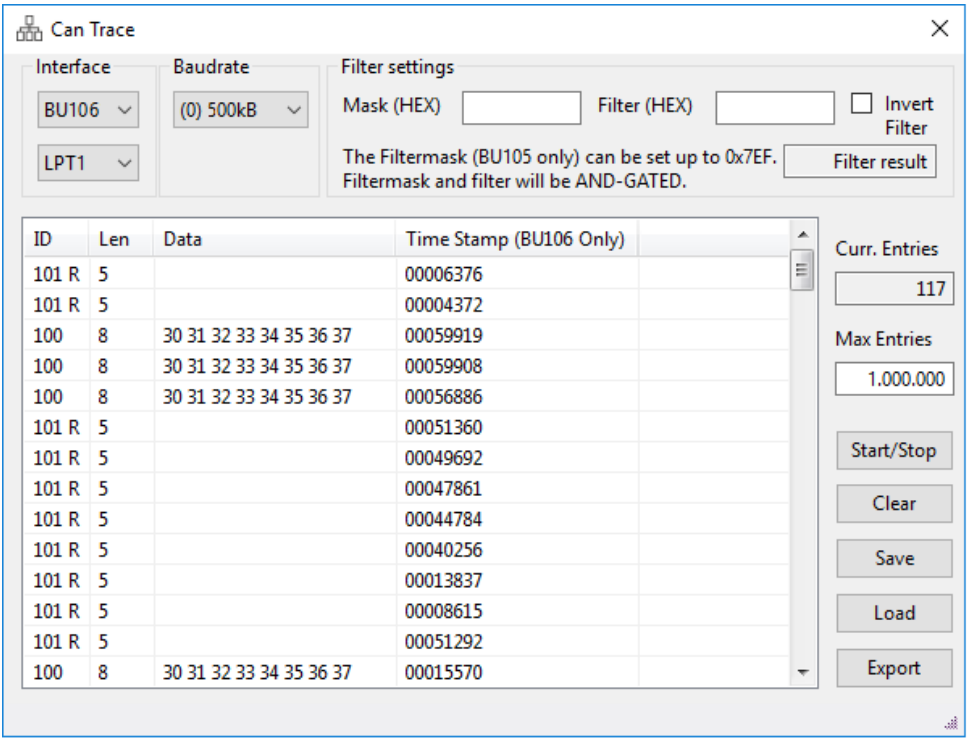


Attention

TimeStamp is only available with the hardware BU106!

If RTR entries (remote frames) exist, this is indicated with a “**R**” behind the ID. The data of remote frames are not displayed, only their expected length in the column **Len**.

On the right side beside the table the number of messages is displayed in the list in the field **Curr. Entries**. The field can be selected in order to copy the value. The maximum number of entries to be recorded can be set with **Max. Entries**. With the **Start/Stop** button the recording is started/stopped; with Clear the table is deleted. With **Save** the recording is saved as a *.bin file, with **Load** such a saved file can be loaded. With **Export** the recording is exported to a *.txt file.



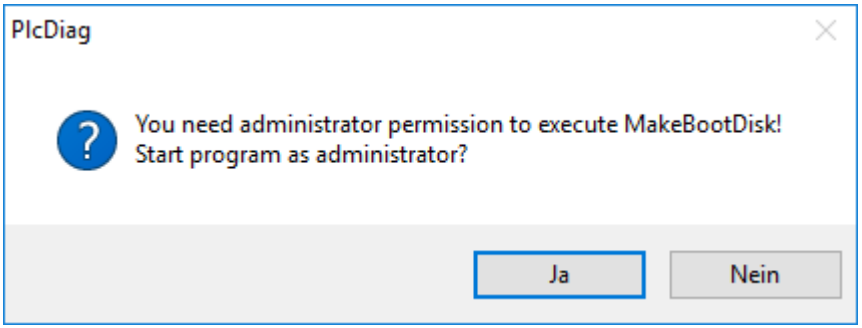
4.8.2.6 Make Bootdisk

When using Make Bootdisk with a card in SIGMATEK format, if the message appears – depending on the Windows version used – that there is no card in the corresponding drive or the card is not formatted, you should always select "Cancel", "Close", or "Ignore".

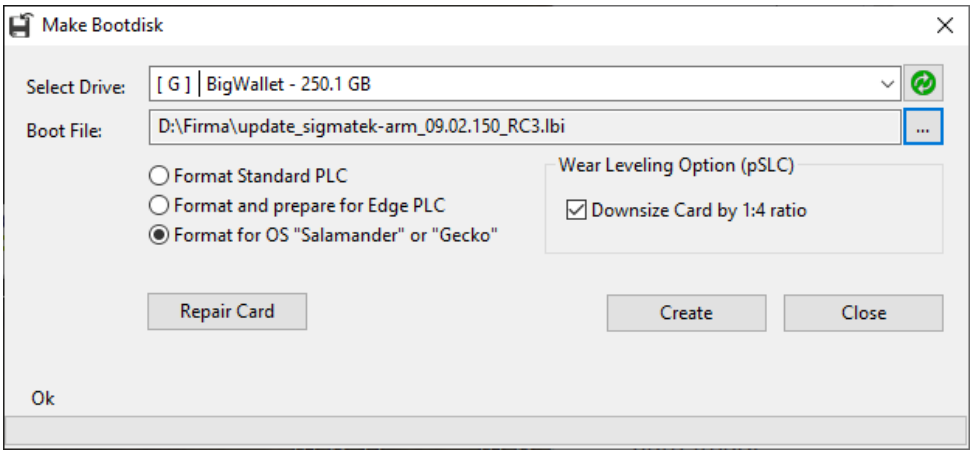
Please note:

Never select "Format disk", as this will destroy the data on the card!

The bootdisk is created from an ***.lbi** file (operating system file). The older ***.rtb** format is also supported. When calling this function, a message box opens – depending on the Windows version – in which the login as administrator is required to run all functions of the tool.



If the question is answered with **No**, the tool cannot be run (except for the Close button, everything is disabled).



When selecting the boot file, the recommended format is set automatically, but the user can change the selection as desired.

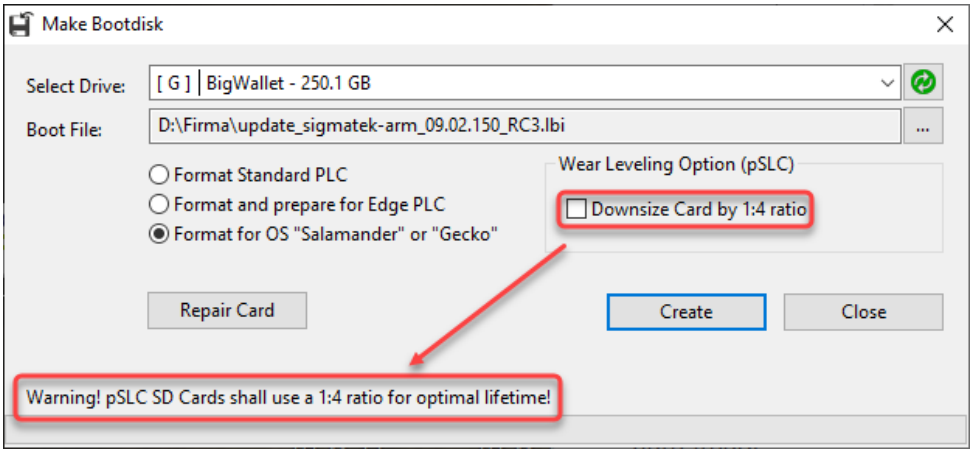
With the radio buttons you can choose between three different formats: for **Standard PLC**, for **Edge PLC**, or for the operating systems **Salamander or Gecko**.

The checkbox "**Downsize Card by 1:4 ratio**" is only enabled if the physical available space on the card is >= 4 GB and is used for Wear Leveling if selected.

Wear Leveling is a method to extend the lifetime of solid-state drives. Solid-state storage (which includes also SD cards) consists of microchips that store data in blocks. Each of these memory blocks can tolerate a certain (and limited) number of programs-erase cycles before becoming unreliable. Wear Leveling arranges the data so that write/erase

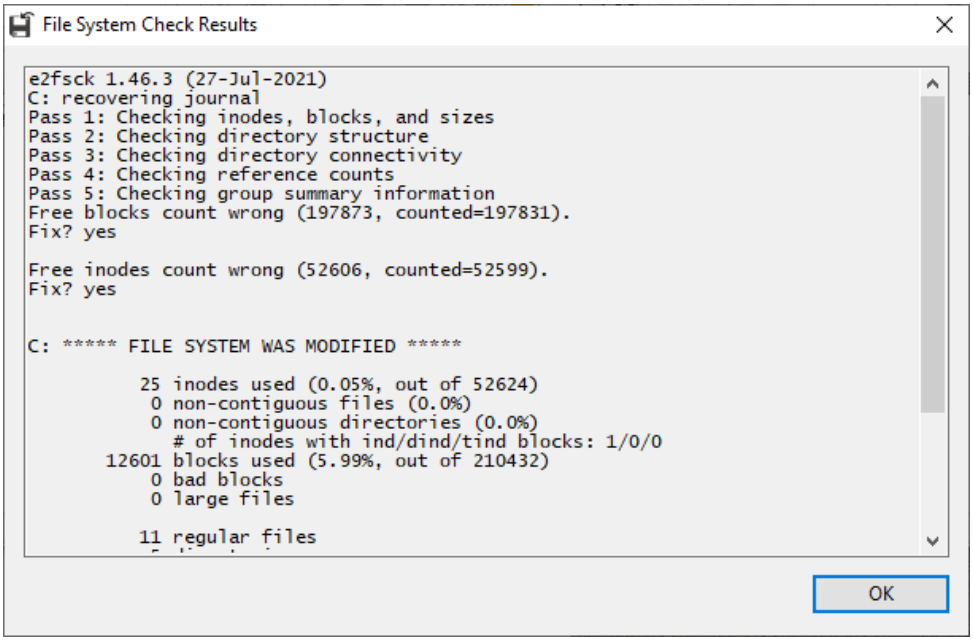
cycles are **evenly distributed across all blocks of the medium**. Here, with the checkbox selected, in the case of a 4 GB pSLC card, 1 GB of the available space will be formatted - the remaining space will be used for Wear Leveling, that is **for increasing the lifetime of the SD card**.

If the "**Downsize Card by 1:4 ratio**" checkbox is deselected, a note flashes in the status bar to alert the user to the Wear Leveling option:



The  button updates the list of the available destination drives (e.g. after removal or reconnection of card readers / memory cards).

The "Repair Card" button will help if the card fails to boot for some reason. After completing the repair, the result is displayed:



The card should now be bootable again.

Please note:

Only the user partition is repaired! If there are errors in system files, the card cannot be repaired with this function!

The option is available to set parameters for running Make Bootdisk (e.g. through a DOS box) and create a storage medium directly. The syntax is written as follows:

PlcDiag/SubProg:7 [/Drive:Driveletter] [/Format:Dateiformat] [/pSLC:FULL, 1GB] [/Bootfile:Filename] [Autostart: 0,1]

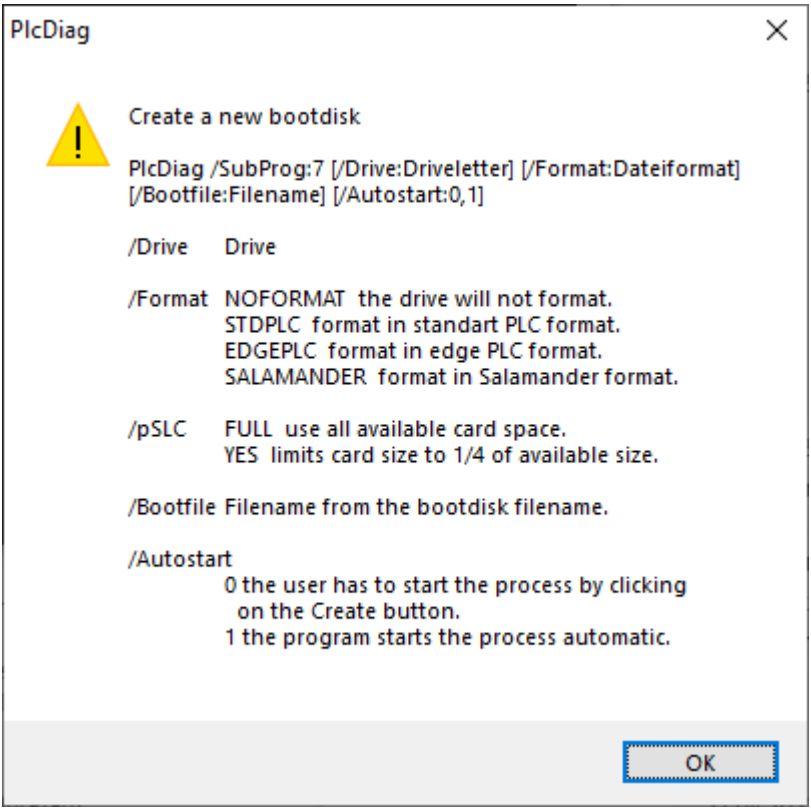
Description of individual parameters:

/Drive	Destination drive with the writable storage medium.
/Format	NONE – no formatting (default value) STDPLC – formatting in standard PLC format EDGEPLC – formatting in Edge PLC format SALAMANDER – formatting in Salamander or Gecko format

/pSLC	FULL - The entire available memory space on the card is used YES - It will limit the card size to 1/4 of the available memory (the rest will be used for wear leveling)	
/Bootfile	Path and file name for the *.lbi file (operating system file)	
/AutostartButton	0	the user starts the process by clicking on “Create”
	1	the progam starts the process automatically

This description is also displayed as help when navigating to the bin folder of the PlcDiag application in the CommandLine application (cmd, or Run) and then typing the following: `PlcDiag /Help /SubProg:7`

The following window is displayed:





Please note:

For the display of the help in the Command Line application to work, the "cmd" program **must already be started as administrator!**

4.8.2.7 Remote CLI

Sends CLI commands to the target system and displays on the screen. Most command line instructions are supported. But they can be different depending on the operating system and the component. For a more detailed description of the supported command please read the documentation of the operating system!

Not supported commands are among others:

BOOT
IF
GOTO
FORMAT
PAUSE
ECHO
TOUCHCFG
DELAY
REM
LISTTS
LABEL
PARTITION
CHKDSK
MEMDUMP
QUCMD
TASKS
TASKS1
TASKS2
PROJECT
REXX

EXIT

As well as the constellations

SETENV FILENAMES

LINK INFO

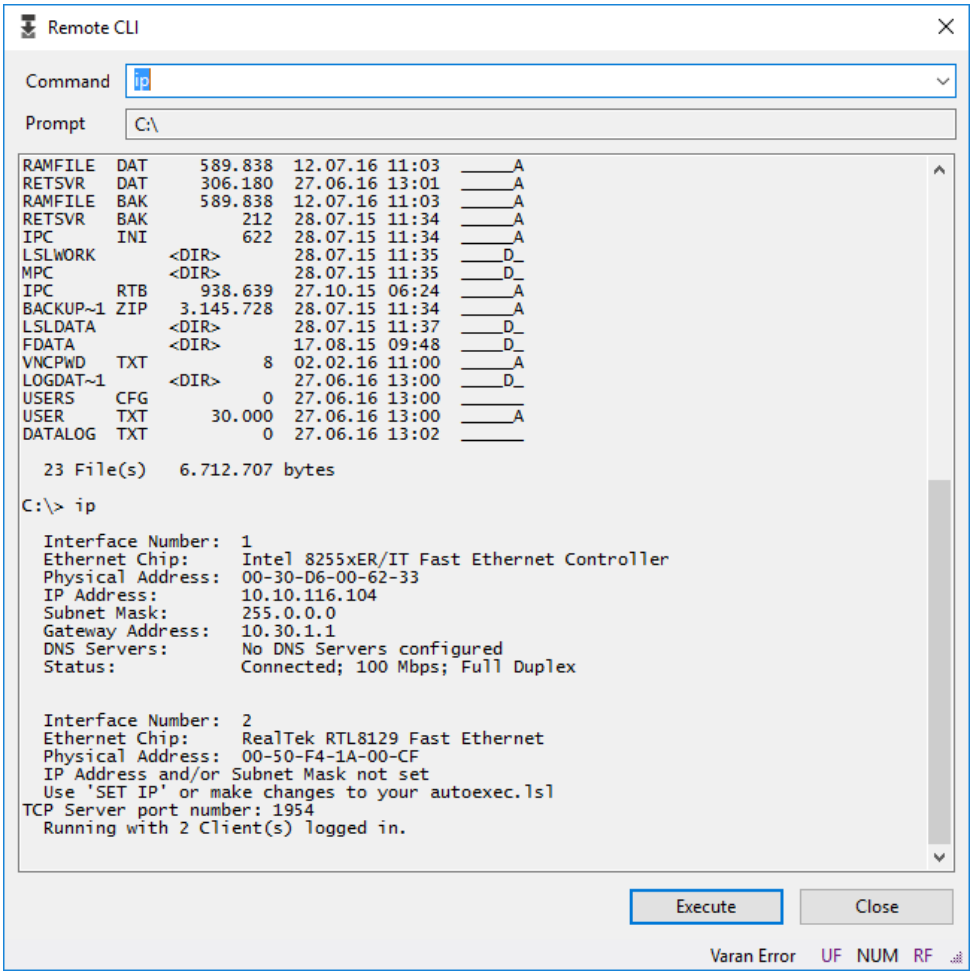
SET with the parameters DATE, TIME, APPCODE or APPDATA.

With the commands TREE and IP all parameters are cut.

It is explicitly pointed out, that using the Remote CLI tool can result in unwanted effects on the control resp. in a running program with improper use!

With the command TYPE the contents of a text file is written directly in the display under the prompt. The command VIEW opens an own dialog with some comfortable functions (scrolling, reloading, handling of different end of line formats). EDIT shows the same dialog like VIEW, but now the file can also be edited and be saved under any name. (e.g. “edit autoexec.lsl”).

With the button Execute or by pressing the Enter key the CLI command is executed, with Close the tool is closed.

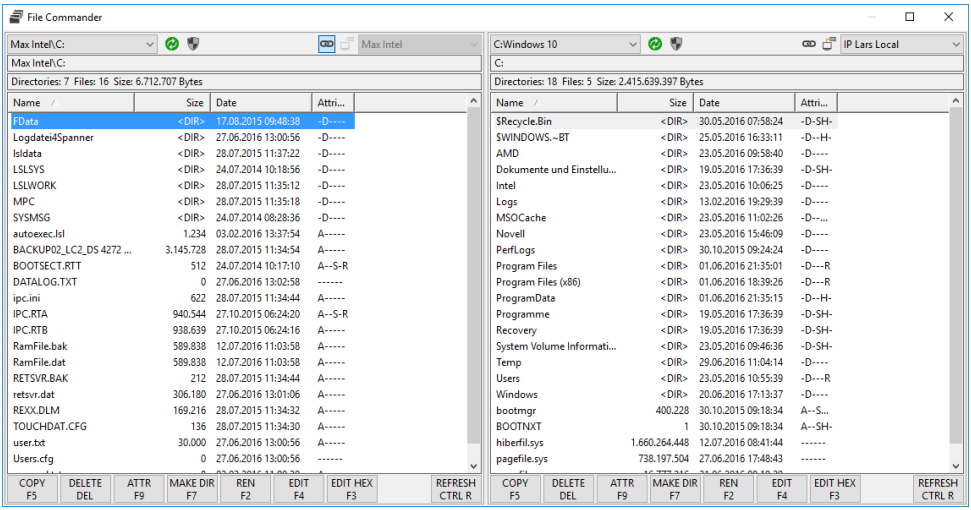


4.8.2.8 File Commander

File explorer for the target and PC. A powerful tool, which – similar to the Windows Explorer – provides many useful functions. It allows e.g. file transformations between two different controls or between a control and a PC. For this the screen is divided into two halves. Depending on the selection, it is possible to work in both directions.

In the toolbar, directly beside a selection box for the drives to be displayed, there are two icons. The first ( ) updates the drive selection (e.g. after removing a Flash card); the second ( ) allows it to execute operations as an administrator. Some actions are only possible in this way (deleting/renaming protected system files, changing file properties, etc.).

To the right, there is the telephone book with the online button and the button for quick settings – already known from the LASAL CLASS toolbar.



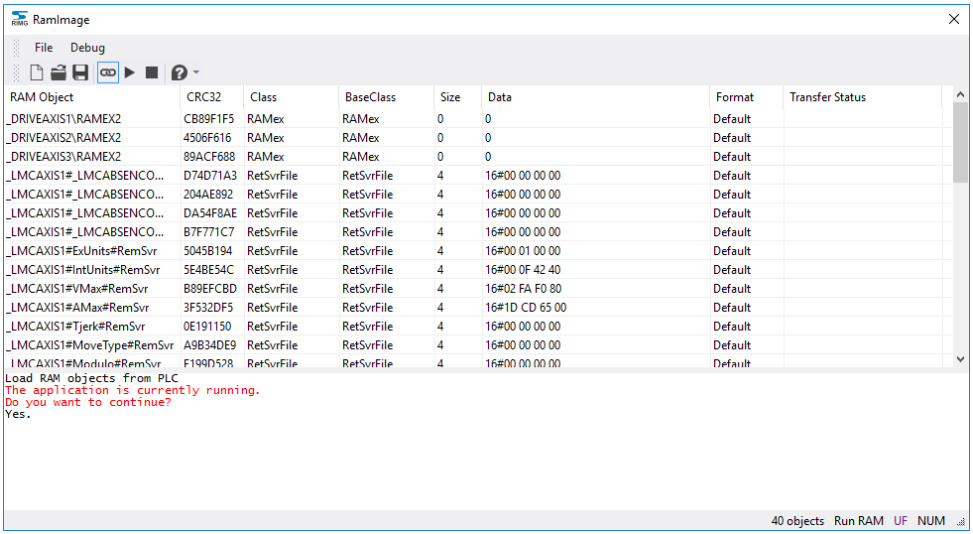
If you select **IP Lars Local** in the drive selection box, the drive tree of the **own PC** is displayed (in the image above on the right side). If a control is online, its name and the drive are displayed – here on the left side.

Below there is useful information about the structure and the memory of the displayed directory.

Then there is the table with the data and column headers that allow sorting. Multiple selection is possible here. The buttons below the table refer to the selected file/s in the active half of the screen – except **REFRESH** and **MAKE DIR**. When pressing the COPY button the selection is copied **from the** half with the active selection **to the** other half!

Beside copying, also deleting files (button **DELETE**), changing the attributes/properties (button **ATTR**), creating a new folder (button **MAKE DIR**), renaming files (button REN) and editing – text files (button **EDIT**) or in hexadecimal format (button **HEX EDIT**) is possible. The button **REFRESH** updates the display in the active half of the screen.

4.8.2.9 Ram Image



With the Ram Image Tool remanent data of the control can be transferred to, respectively, loaded from the target system. Moreover functionalities for verification and reparation of these data are available.

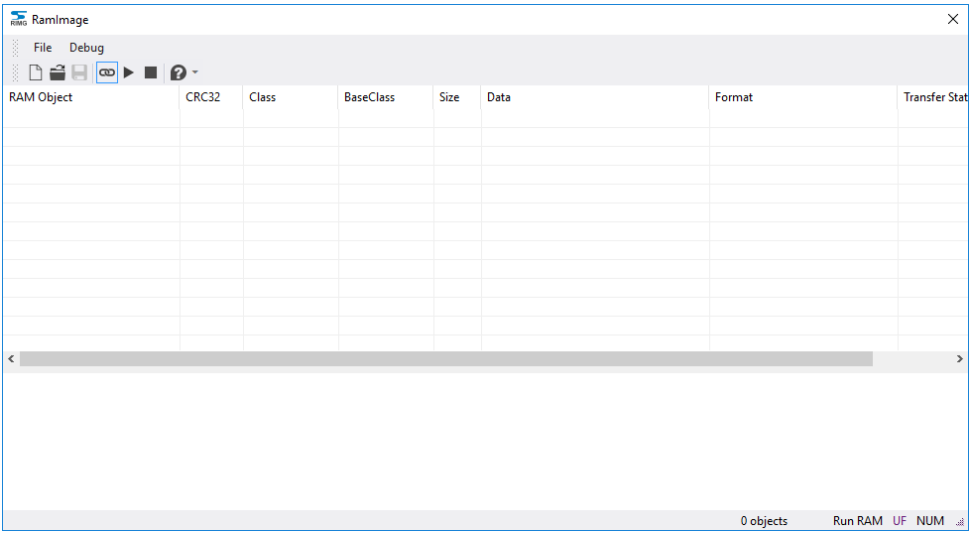
A few words regarding the SRAM format: In the C-DIAS CPU, SET RAM FORMAT NEW' is set by default. With these CPUs, a distinction is made between CPUs with and without a file system.

In CPUs without a file system, the environmental variable SETENV SRAMFORMAT 1' is set. With this format, both the data and the file system are stored in the SRAM (battery buffered).

In CPUs with a file system (own FLASH card), the environmental variable SETENV SRAMFORMAT 2' is set. In this case, only the data is stored in SRAM. The file system is located in the RAMFILE.DAT file.

There is still an older SRAM format, which can be selected using the ‚SET RAM FORMAT OLD' command. Since this format is no longer supported, it is not discussed here.

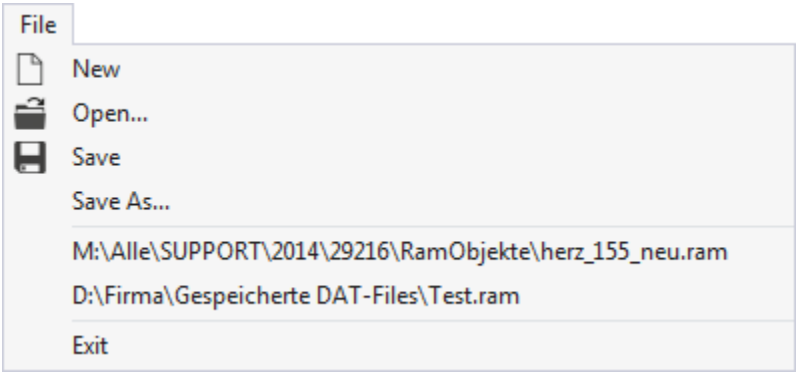
The current LASAL CLASS 2 setting is used as online adjustment. The toolbar includes, beside the standard buttons, buttons for the online connection as well as for starting and stopping the control.



In the upper area information to the respective data objects are displayed, whereby the Data column shows the included value. This value can also be changed and transferred to the control again, respectively, saved to a file.

In the lower area status messages are displayed concerning the latest menu command.

File



New

Deletes the objects from the list.

Open

Opens a saved file with remanent data objects ending with ".ram".

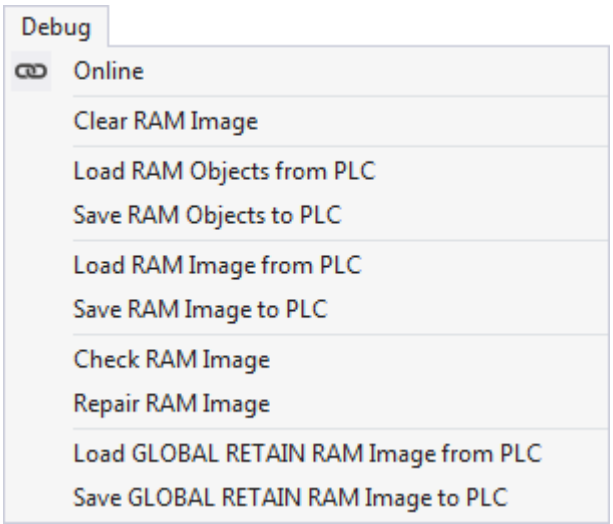
Save

Saves the remanent data objects currently displayed.

Save As...

Saves the remanent data objects currently displayed under a desired name.

Debug



Online

Establishes an online connection to the target system.

Clear RAM Image

Deletes all remanent data in a way that the initial values or 0 (when no initial value is available) are used at next start of the control program.

Load RAM Objects from PLC

Loads all remanent data objects and shows them in the list. For this action, the control must be in **Run** mode.

Save RAM Objects to PLC

Saves the remanent data objects displayed in the list to the control. For this action, the control must be in Run mode or has been **run** at least once and is now in **reset** mode.

- **RETENTIVE SERVER** (Sram and File): If these objects loaded while the control is Run mode, a **Reset** and **Run** must be performed one after the other.

Load RAM Image from PLC

Differing to the object commands the complete image of the data is transferred, but not the single remanent data objects. After this command a file window is opened, in which the storage location of the ".dat" file has to be defined. In the user interface no objects are displayed. Therefore it is not possible to influence the values of the respective objects.

This may be a disadvantage, but, however, the command is faster than the transfer of the single objects and can also be executed at Reset mode of the control.
If the control is in **Run** mode, this command cannot be executed.

Save RAM Image to PLC

Equipollent to the Load Image command an image file (*.dat) has to be selected here after execution. This image of the remanent data is transferred to the control afterwards.
If the control is in **Run** mode, this command cannot be executed.
The SRAM format must match (set with SET SRAMFORMAT 1 or SET SRAMFORMAT 2).

Check RAM Image

Here an existent Ram Image file (*.dat) can be checked, whether the data in the file are valid or not.

Repair RAM Image

Repairs a broken Ram Image file (*.dat), if possible.

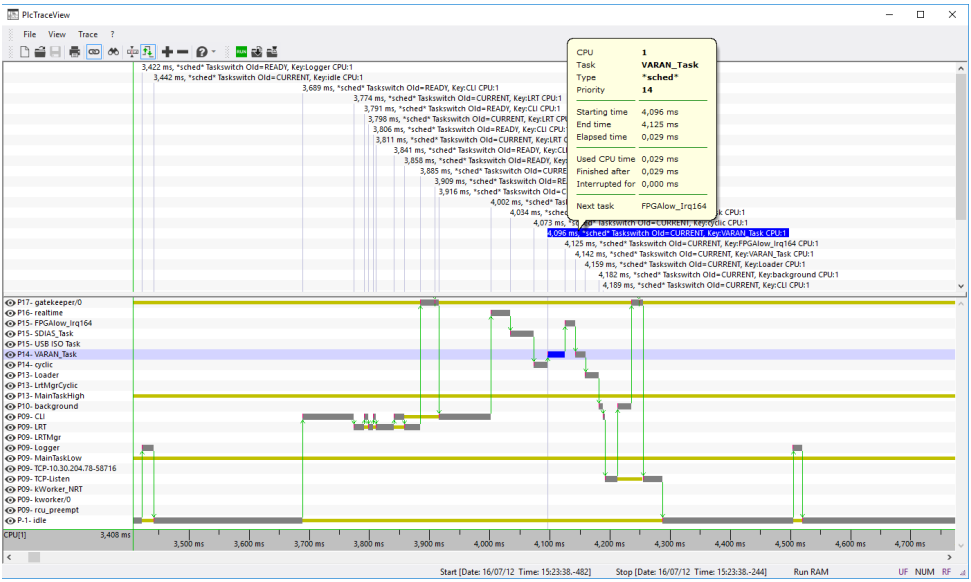
Load GLOBAL RETAIN RAM Image from PLC

Loads the global variables from the null voltage protected SRAM. These variable were declared with VAR_GLOBAL RETAIN and stored in a separate SRAM space, which must be configured with ‚SET SRAMRETAIN <size>'. The data are handled separately from the other SRAM data.

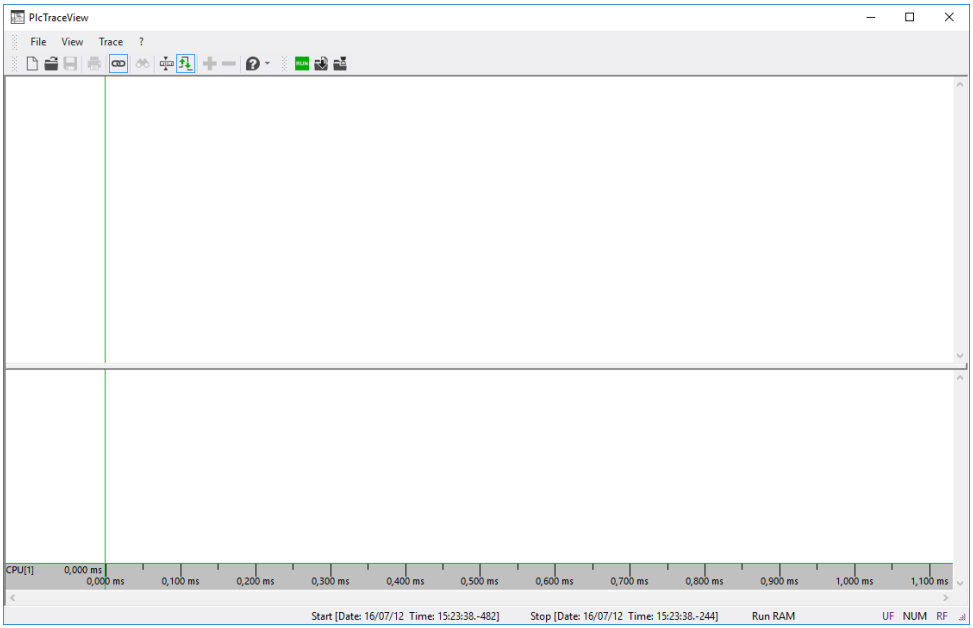
Save GLOBAL RETAIN RAM Image to PCL

Save the global variables from the null voltage protected area of the SRAM in the PLC. These variable were declared with VAR_GLOBAL RETAIN and stored in a separate SRAM space, which must be configured with ‚SET SRAMRETAIN <size>'. The data are handled separately from the other SRAM data.

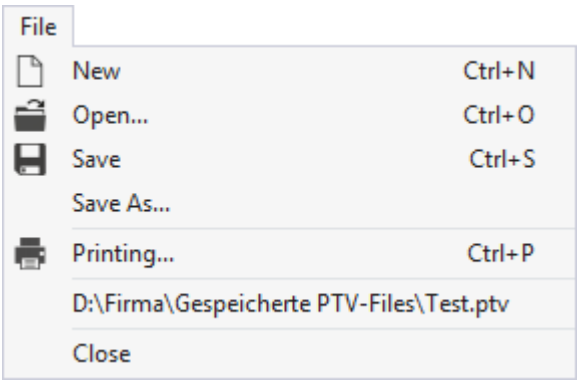
4.8.2.10 PLC Trace View



With the **PlcTraceView** tool records about the time response of individual tasks of a project can be made. Therefore this tool can be used for error search, analysis, diagnosis, respectively, for time measurement. With different settings the record start can be triggered (immediately, after a defined time, after certain events, aso).


PlcTraceView
File View Trace ?
CPU[1]
0,000 ms
0,000 ms 0,100 ms 0,200 ms 0,300 ms 0,400 ms 0,500 ms 0,600 ms 0,700 ms 0,800 ms 0,900 ms 1,000 ms 1,100 ms
Start [Date: 16/07/12 Time: 15:23:38.-482] Stop [Date: 16/07/12 Time: 15:23:38.-244] Run RAM
UF NUM RF

File


File
New Ctrl+N
Open... Ctrl+O
Save Ctrl+S
Save As...
Printing... Ctrl+P
D:\Firma\Gespeicherte PTV-Files\Test.ptv
Close

New

New record.

Open

Opens an existing record.

Save

Saves the opened record.

Save As...

Saves the current record under a defined name.

Printing...

Opens the printer menu.

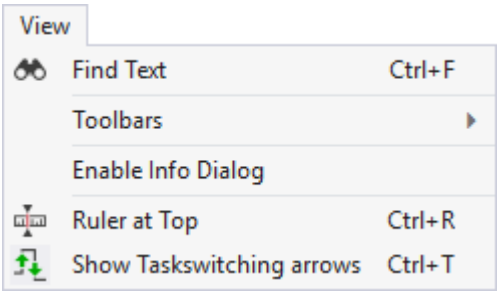
Recent File List

The last opened projects are listed here.

Close

Closes the program.

View

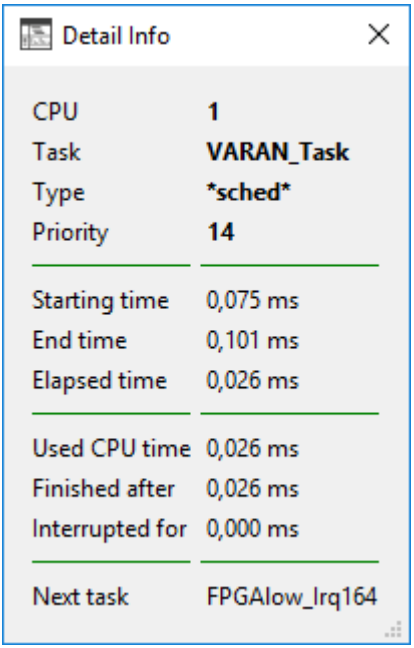


Find Text

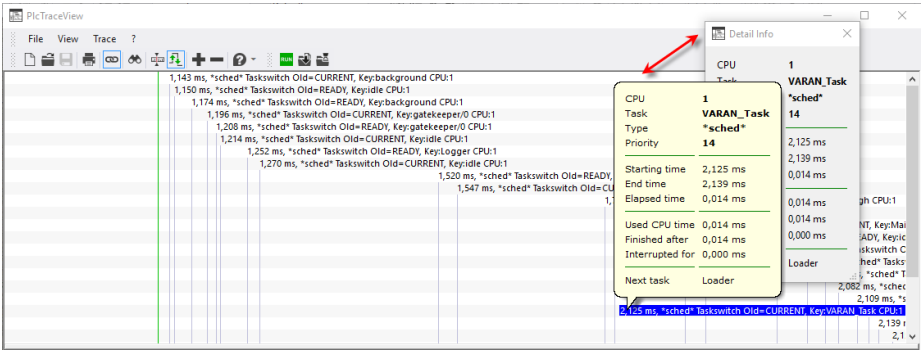
Opens the **Find Text window** to search for text in the **PlcTraceView window**.

Enable Info Dialog

During activation, a small separate dialog window is shown when a task is clicked as seen below.



When the mouse cursor is placed over the task, the same information is shown in a Tooltip:



The info dialog then makes sense: While the Tooltip changes in real time to fit the displayed task as the mouse moves over it, the information in the info dialog remains stationary until the next **click** and can therefore be used for comparison.

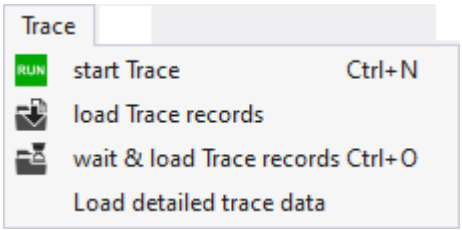
Ruler at Top

The timeline in the output window can be moved to the upper position with Ruler at Top.

Show Taskswitching arrows

Here the green arrows between the single task can be fade in, respectively, fade out.

Trace



start Trace

Opens the window (**Trace starten window**) to prepare the record settings.

load Trace records

Loads the recorded data.

wait & load Trace records

Waits for valid data and then starts automatically with the recording of PLC data.

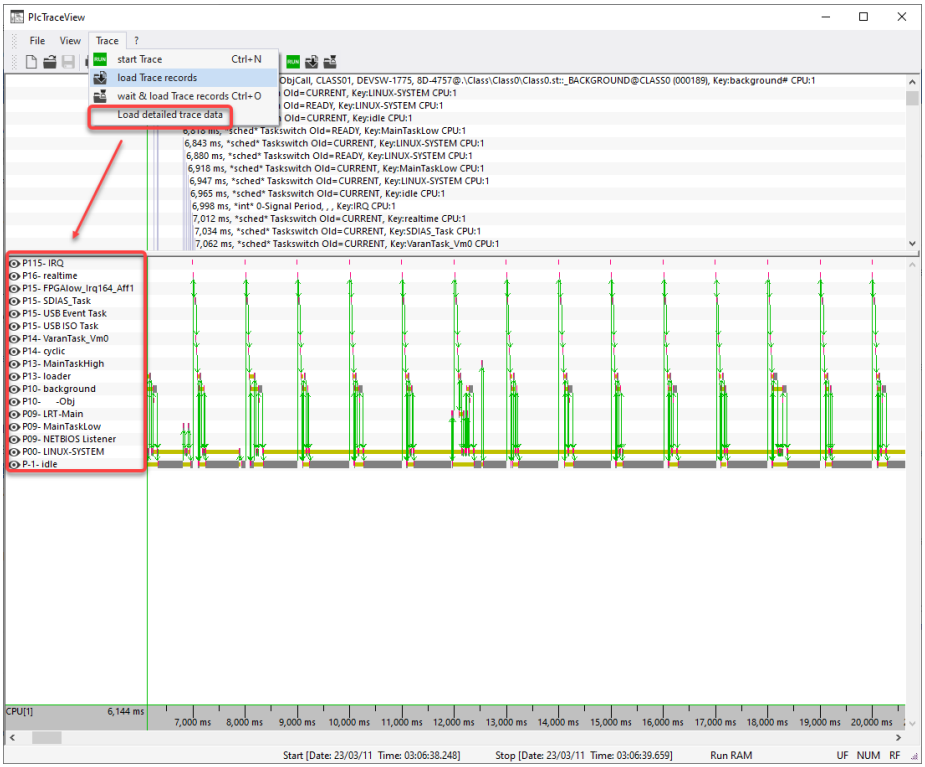
Load detailed trace data



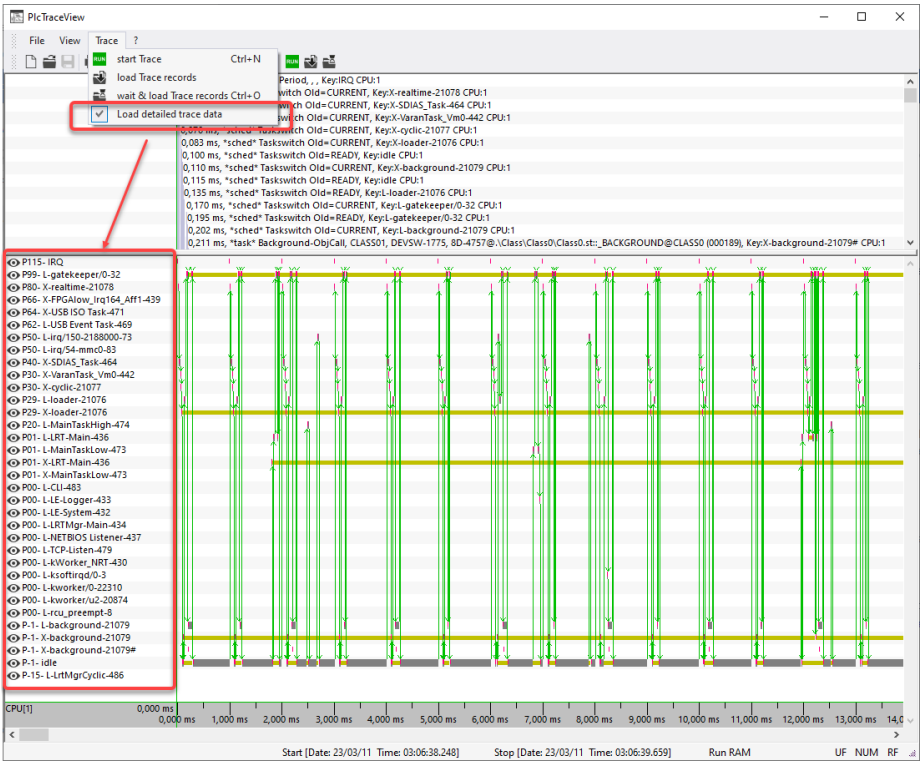
Attention!

This menu item is **not selectable** if there is an online connection to a CPU **with RTK operating system**!

Here, a checkbox can be used to select how many tasks are to be loaded. If this **checkbox is not selected**, the **tasks are loaded "compressed"**, i.e. in a similar form as in the RTK operating system:

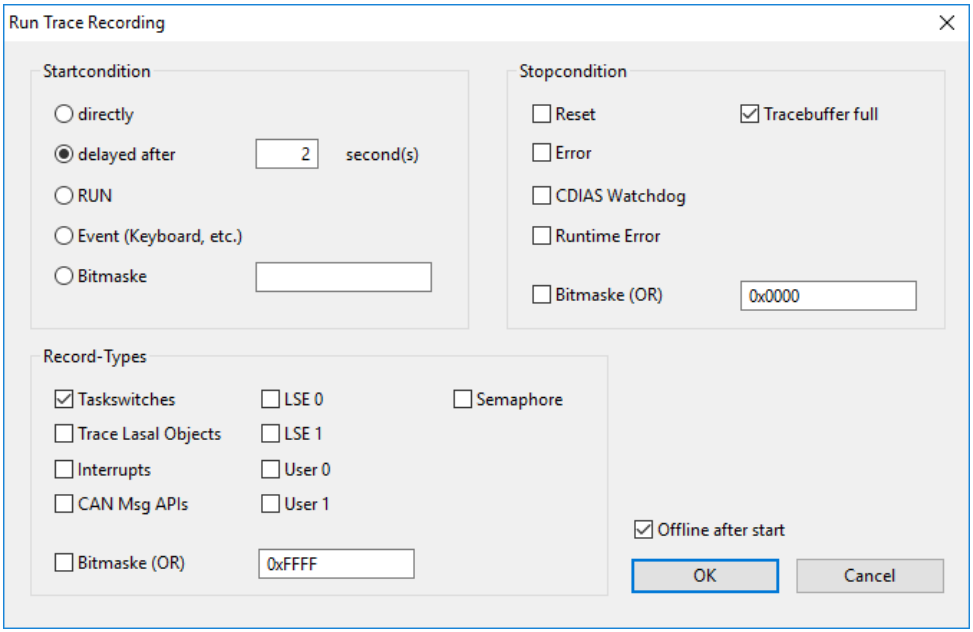


If the **checkbox is selected**, **all tasks** - also those belonging to the LINUX system - **are loaded**:


PlcTraceView
File
View
Trace
?
start Trace Ctrl+N
load Trace records
wait & load Trace records Ctrl+O
Load detailed trace data
P115- IRQ
P99- L-gatekeeper/0-32
P80- X-realtime-21078
P66- X-FPGAlow_Irq164_Aff1-439
P64- X-USB ISO Task-471
P62- L-USB Event Task-469
P50- L-irq/150-2188000-73
P50- L-irq/54-mmc0-83
P40- X-SDIAS_Task-464
P30- X-VaranTask_Vm0-442
P30- X-cyclic-21077
P29- L-loader-21076
P29- X-loader-21076
P20- L-MainTaskHigh-474
P01- L-LRT-Main-436
P01- L-MainTaskLow-473
P01- X-LRT-Main-436
P01- X-MainTaskLow-473
P00- L-CLI-483
P00- L-LE-Logger-433
P00- L-LE-System-432
P00- L-LRTMgr-Main-434
P00- L-NETBIOS Listener-437
P00- L-TCP-Listen-479
P00- L-kWorker_NRT-430
P00- L-ksoftirqd/0-3
P00- L-kworker/0-22310
P00- L-kworker/u2-20874
P00- L-rcu_preempt-8
P-1- L-background-21079
P-1- X-background-21079
P-1- X-background-21079#
P-1- idle
P-15- L-LrtMgrCyclic-486
CPU[1]
0,000 ms
1,000 ms
2,000 ms
3,000 ms
4,000 ms
5,000 ms
6,000 ms
7,000 ms
8,000 ms
9,000 ms
10,000 ms
11,000 ms
12,000 ms
13,000 ms
Start [Date: 23/03/11 Time: 03:06:38.248]
Stop [Date: 23/03/11 Time: 03:06:39.659]
Run RAM
UF
NUM
RF

4.8.2.101 Run Trace Recording Window

If you are online and the recording has been started with **start Trace**, the Run Trace Recording window opens. In it start and end conditions can be set and it can be selected, which kind of entries should be recorded.



Startcondition

Directly	Record is started immediately after confirmation with OK.
Delayed after	Record is started after a defined time.
RUN	Record is started at program start.
Event (Keyboard, etc.)	Record is started after certain events (Key, Touch, Mouse,...)
Bitmaske	Reserved for future expansions.

Stopcondition

Reset	Record is stopped with the Reset command.
Error	Record is stopped at a general error (AppliMemHeap, Access,…)
CDIAS Watchdog	Record is stopped at occurrence of a C-DIAS watchdog.
Runtime Error	Record is stopped at occurrence of a runtime error.
Bitmaske (OR)	Reserved for future expansions.
Tracebuffer full	Record is stopped after the buffer is full (active). Otherwise the records are written into a ring buffer (inactive).

Record-Types

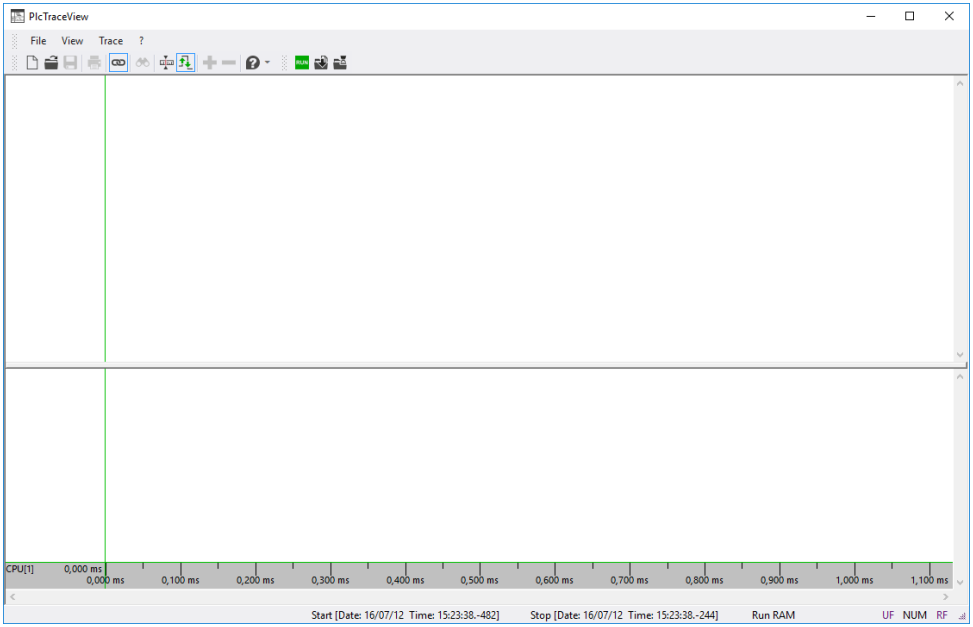
Taskswitches	The task switch of the operating system is recorded.
Trace Lasal Objects	The calls of the RtWork-, CyWork- and Bg methods are recorded.
Interrupts	Presently not available.
CAN Msg APIs	Presently not available.
Bitmaske (OR)	Reserved for future expansions.
LSE 0	Records of LSE.
LSE 1	Records of LSE.
User 0	Records of the user about operating system functions.
User 1	Records of the user about operating system functions.
Semaphore	Recording of the operating system flag

Offline after start

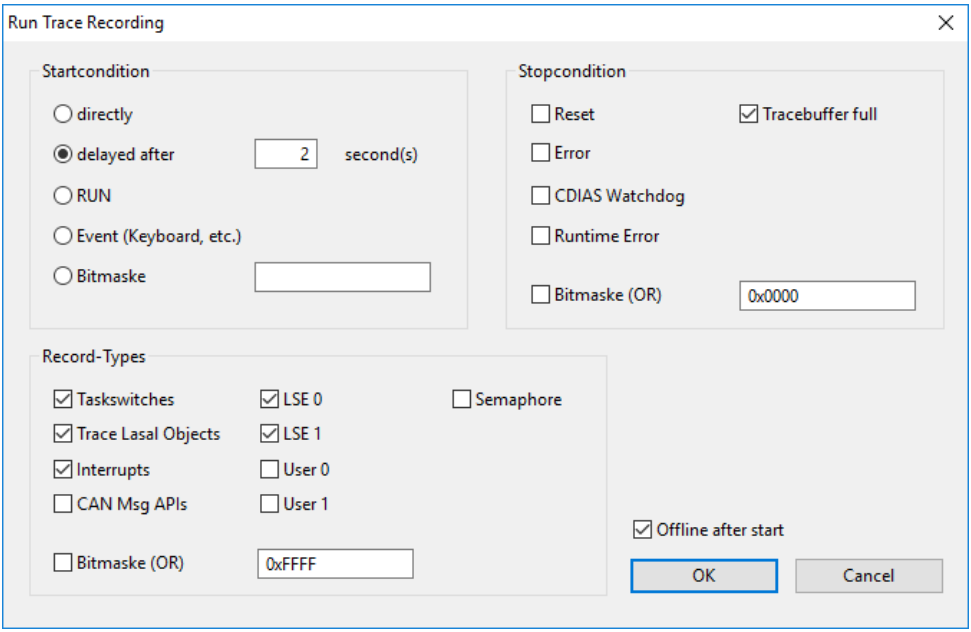
At activation it is switched to the offline mode at start. However to access the records (Menu Trace, load Trace records) an online connection has to be made.

4.8.2.102 Record of Data

Start the **PlcTraceView window** and establish an online connection with the control with the icon  in order to execute a recording.



Afterwards select **Trace, start Trace** and set the record criteria in the **Run Trace Recording window**.



Start the recording with **OK**.

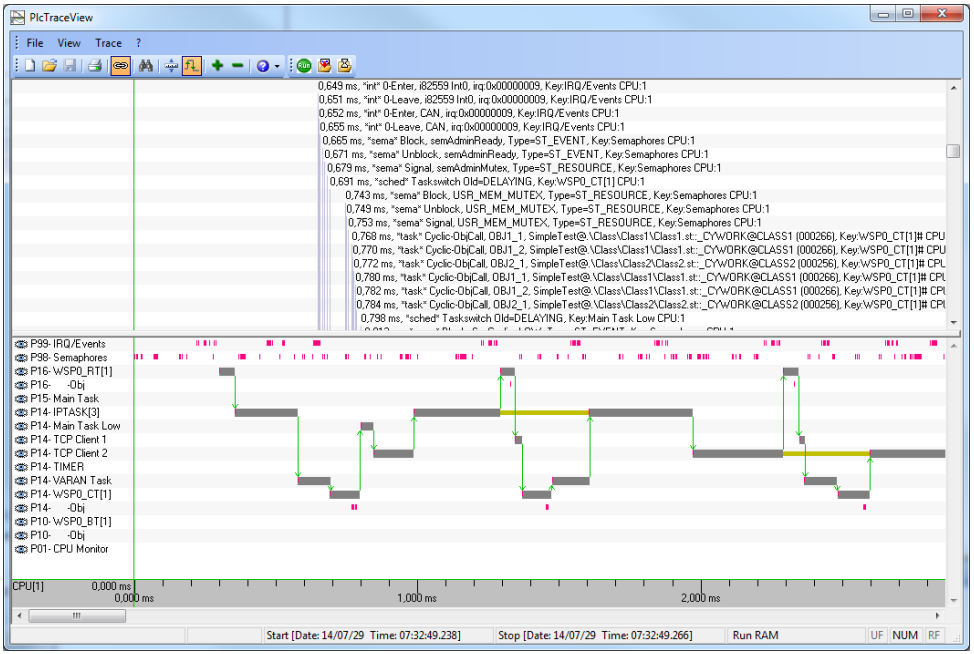


It is switched to the offline mode after start with **OK** at activation of the checkbox next to **Offline after start**.

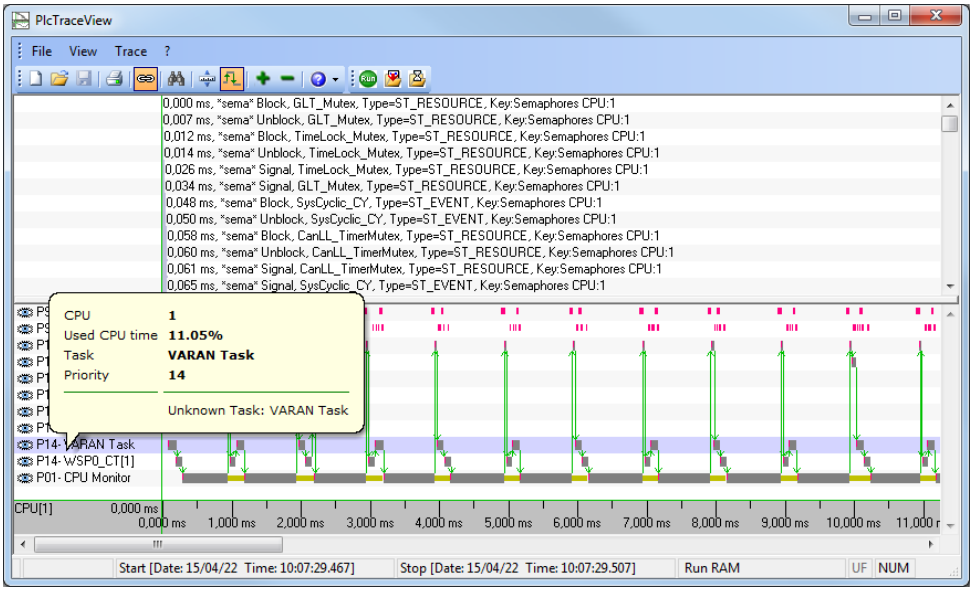
To call, respectively, display the records an online connection with the control has to be established again. Afterwards select **Trace, load Trace records** in the menu to show the records.

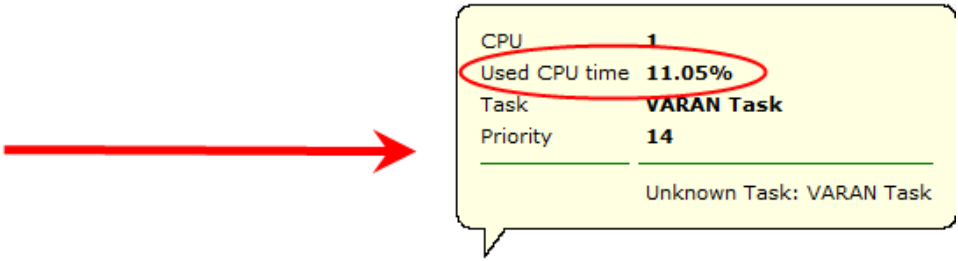
The following window shows the records of a project, as the individual tasks are called, interrupted, stopped and called again.

With plus and minus  the time line can be enlarged, respectively, minimized.



The information in the upper section of the recording window shows when the individual tasks are called. The lower section of the window shows the individual task calls. Here, all tasks can be seen on the left side. If the mouse pointer is placed on one, the CPU number, task name and priority, as well as the CPU time required is shown:





In the status bar, the start and end time of the recording can be read.

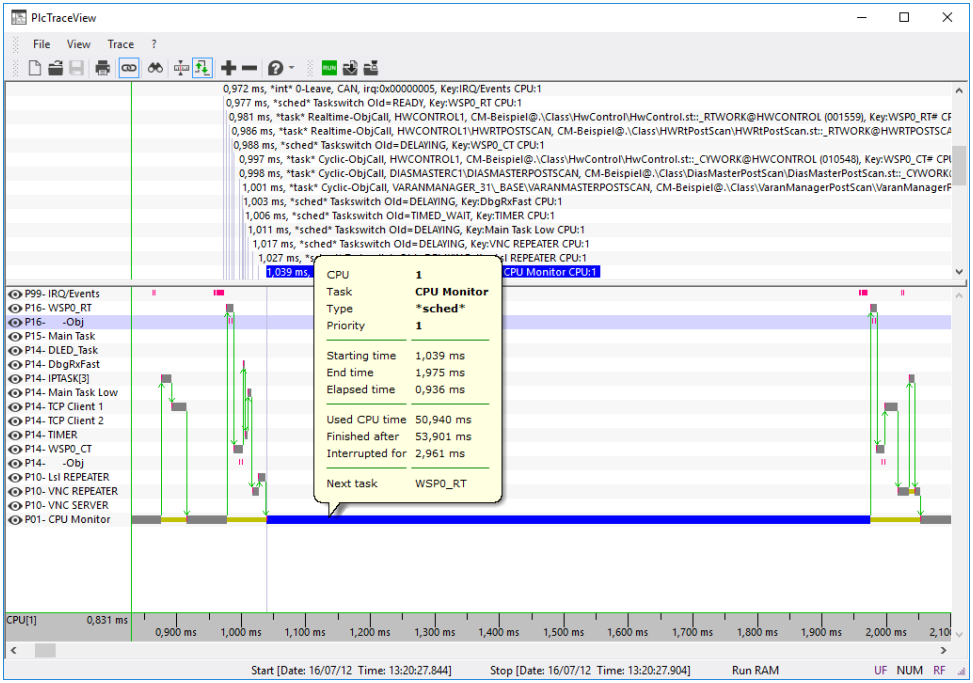
Task Explanation

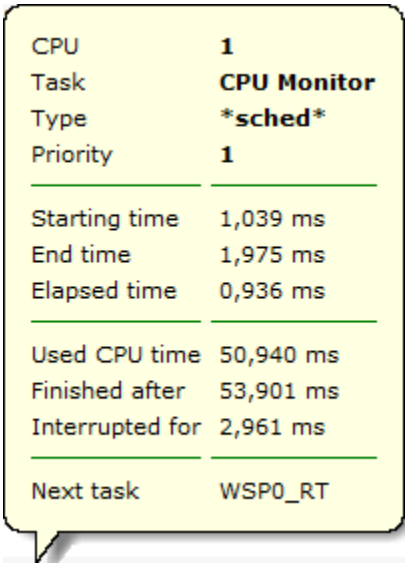
WSPO_RT	Realtime task
MainTask	Operating system task
DbgRxFast	Serial online connection
IPTASK	Internal task for the IP stack (per network interface)

TCP Client 1	TCP Online connection
TIMER	Internal task for the IP stack
WSPO_CT	Cyclic task
WSPO_BT	Background task
LSE0	Lse Task0
LSE1	Lse Task1
CPU Monitor	Operating system task (for the calculation of the CPU time)

Further explanations of tasks you can find in the **LASAL OS Tasks documentation**.

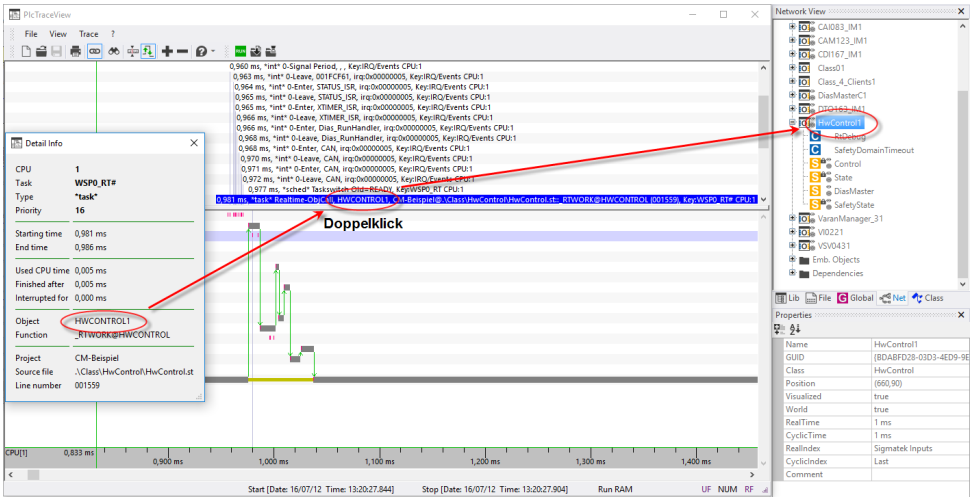
If the mouse remains over an entry in the upper area or a graphic representation of a task in the lower area, a tooltip appears containing all the essential information on the task.





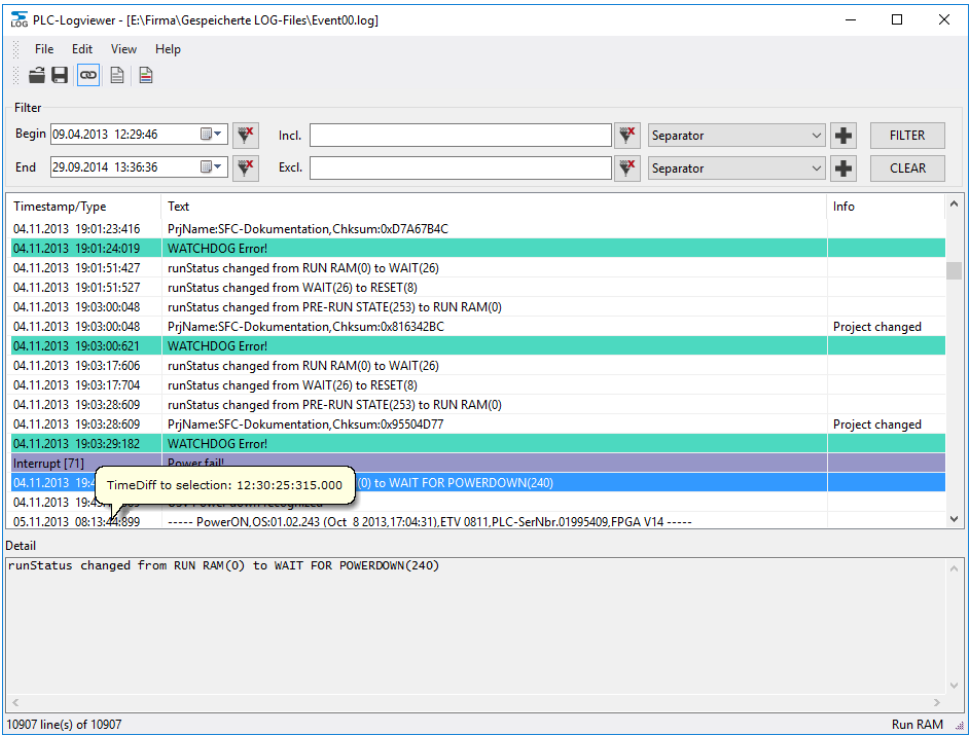
CPU	CPU number (for multicore processors, this can deviate from 1)
Task	Task name
Type	Task group
Priority	Task priority
Starting time	Start time of the task call.
End time	End time of the task call.
Elapsed Time	Time elapsed between the stat and end of the task call.
Used CPU time	The total runtime of the task.
Finished after	The entire task was ended after this time.
Interrupted for	The entire task was interrupted for this time period.
Next Task	Name of the next task called.

If an object entry in the display area is double clicked, the object opens in the corresponding tree:


PlcTraceView
File View Trace ?
Detail Info
CPU 1
Task WSP0_RT#
Type "task"
Priority 16
Starting time 0,981 ms
End time 0,986 ms
Used CPU time 0,005 ms
Finished after 0,005 ms
Interrupted for 0,000 ms
Object HWCONTROL1
Function _RTWORK@HWCONTROL
Project CM-Beispiel
Source file .\Class\HwControl\HwControl.st
Line number 001559
Doppelklick
Network View
CAI083_IM1
CAM123_IM1
CDI167_IM1
Class01
Class_4_Clients1
DiasMasterC1
HwControl1
SafetyDomainTimeout
Control
State
DiasMaster
SafetyState
VaranManager_31
VI0221
VSV0431
Emb. Objects
Dependencies
Properties
Name HwControl1
GUID {BDABFD28-03D3-4ED9-9E
Class HwControl
Position (660,90)
Visualized true
World true
RealTime 1 ms
CyclicTime 1 ms
RealIndex Sigmatek Inputs
CyclicIndex Last
Comment

4.8.2.11 PLC Log Viewer

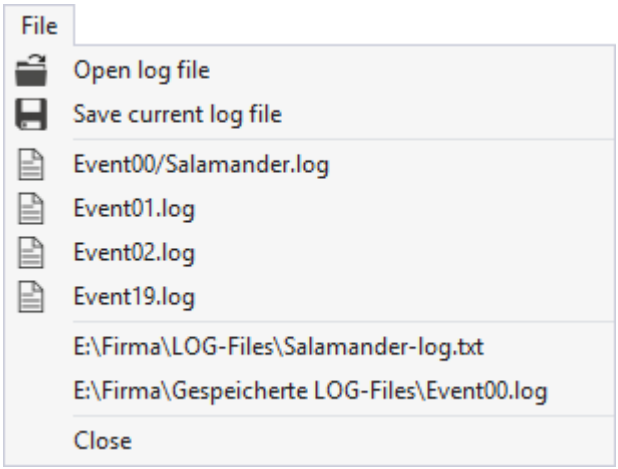
A viewer for the log files on the PLC is available: the PLC Log Viewer. It offers a comfortable possibility to open, read, filter and save the EventLog files of the PLC.



The Menu Bar

File

In this menu, the log files can be selected, opened, the currently displayed log file saved or closed.


File
Open log file
Save current log file
Event00/Salamander.log
Event01.log
Event02.log
Event19.log
E:\Firma\LOG-Files\Salamander-log.txt
E:\Firma\Gespeicherte LOG-Files\Event00.log
Close

Open log file

Opens an existing log file.

Save current log file

Saves the currently displayed log file in the original format.

Event00.log, Event01.log, Event02.log, Event19.log

Opens the respective log file in the PLC. These menu items can only be selected in online mode!

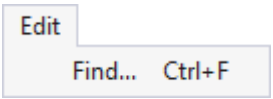
Recent File List

Here, the file opened most recently can be reopened.

Close

Closed the PLC Log Viewer.

Edit

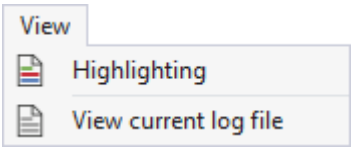


Find

Opens the search dialog.

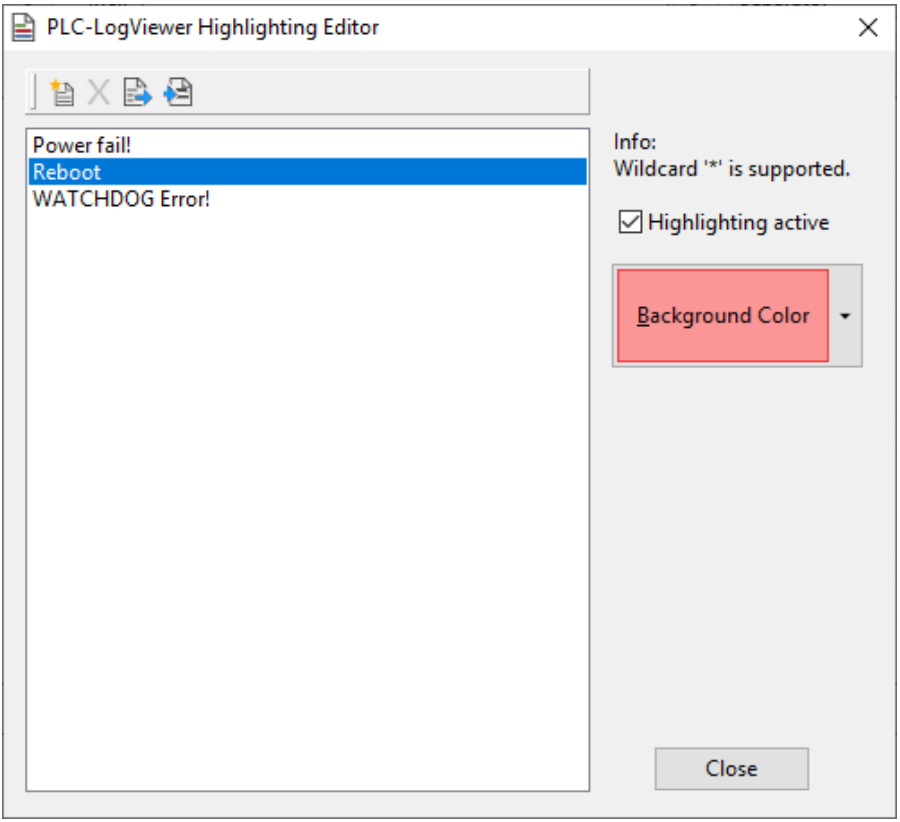
View

In this menu the selection between various views is made.



Highlighting

Here, the Highlighting dialog is opened. In this dialog box, the background color for the individual lines can be defined for the text in the "Text" column.



With the  icon, a new text can be entered for highlighting, with the  icon, existing text is deleted.

Press  to export the highlighting information as a *.xml-file, press  to import from such a file.



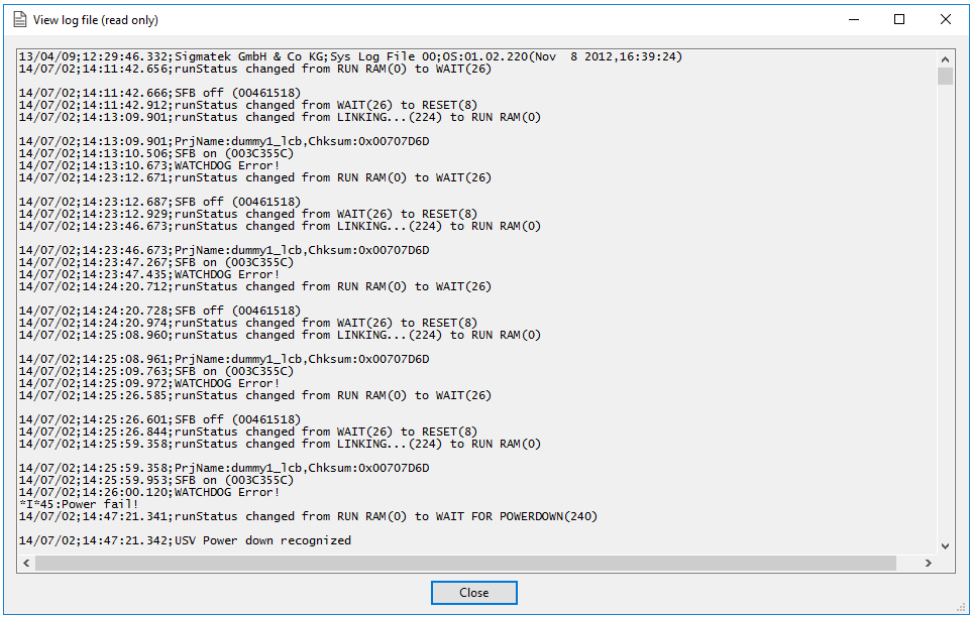
Attention!

Neither before saving as a file, **nor** before importing from a saved file is a security query displayed! This means that an existing file is overwritten during the export or the changes already made in the dialog are overwritten during the import without confirmation!

In the right half of the dialog box, the highlighting can be activated and the background color selected.

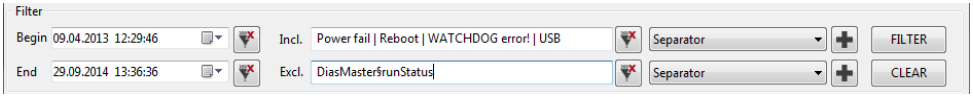
View current log file

Here, the log file can be viewed in the original format (without interpretation and classification.



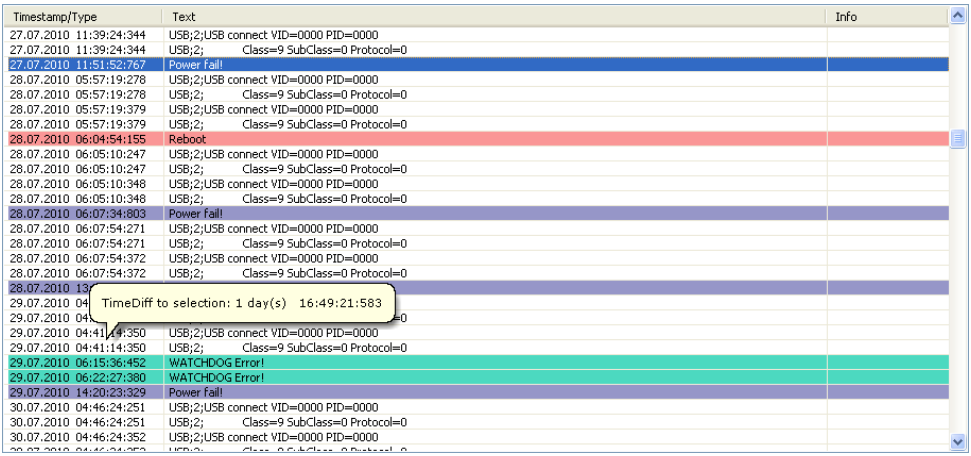
In the Toolbar  are all other buttons corresponding to one of the previously discussed menu items (**Open log file, Save current log file, View current log file, Highlighting**), with the exception of the **Online icon**  . The Icons are identical to those in the menu.

The area below is for filtering the displayed data.



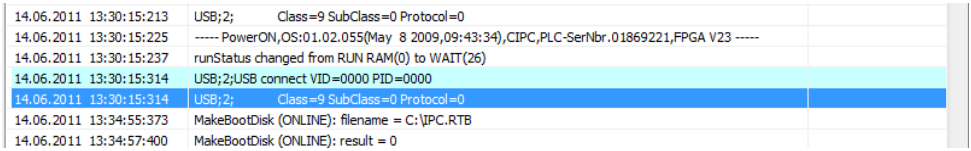
On the left, the display time of an entry can be limited; in the center, the entry can be filtered by text (inclusive and exclusive). To the right, the fields for combined filters are located (separator = OR gate, represented in the upper field by "|" – in the lower field by "§"). On the far right, the filter can be enabled with the Filter button (also with the enter key) or reset all fields with the **Clear** button.

In the display section, the selected log file entries can now be seen.



The entries activated in the **Highlighting dialog** appear with a colored background.

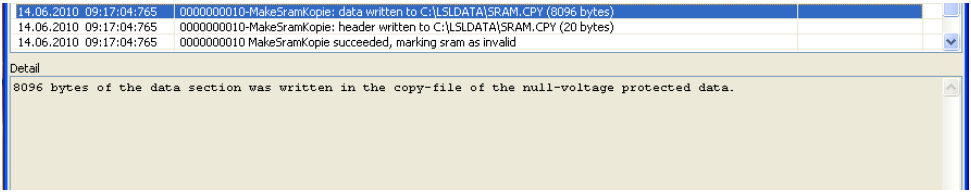
If several lines belong together, the additional (unmarked) lines are shown with a light blue background:



The tool tip shows the time difference between the selected (or selection shortage of the first) entry and the entry to which the mouse points.

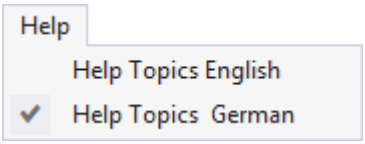
With a **double-click** on a line or with **F1**, the specific logfile online help opens.

In the detail field, the details of the selected entry are shown.



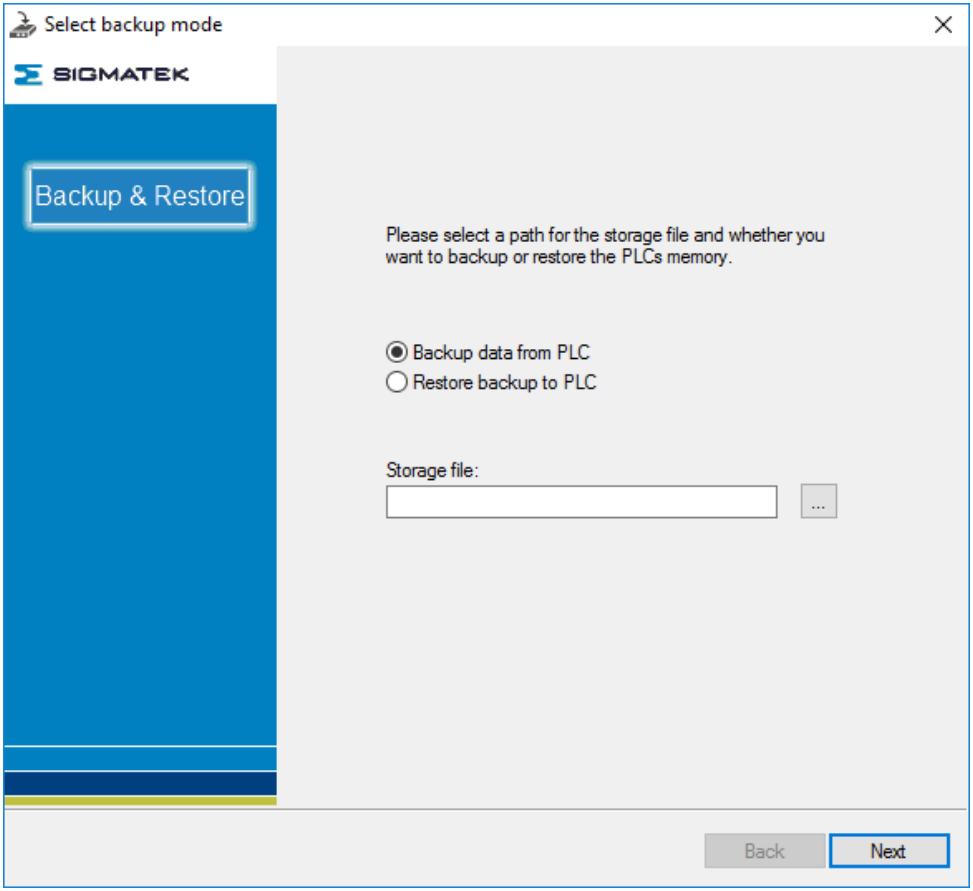
Help

In this menu, the German and English Online Help can be displayed.

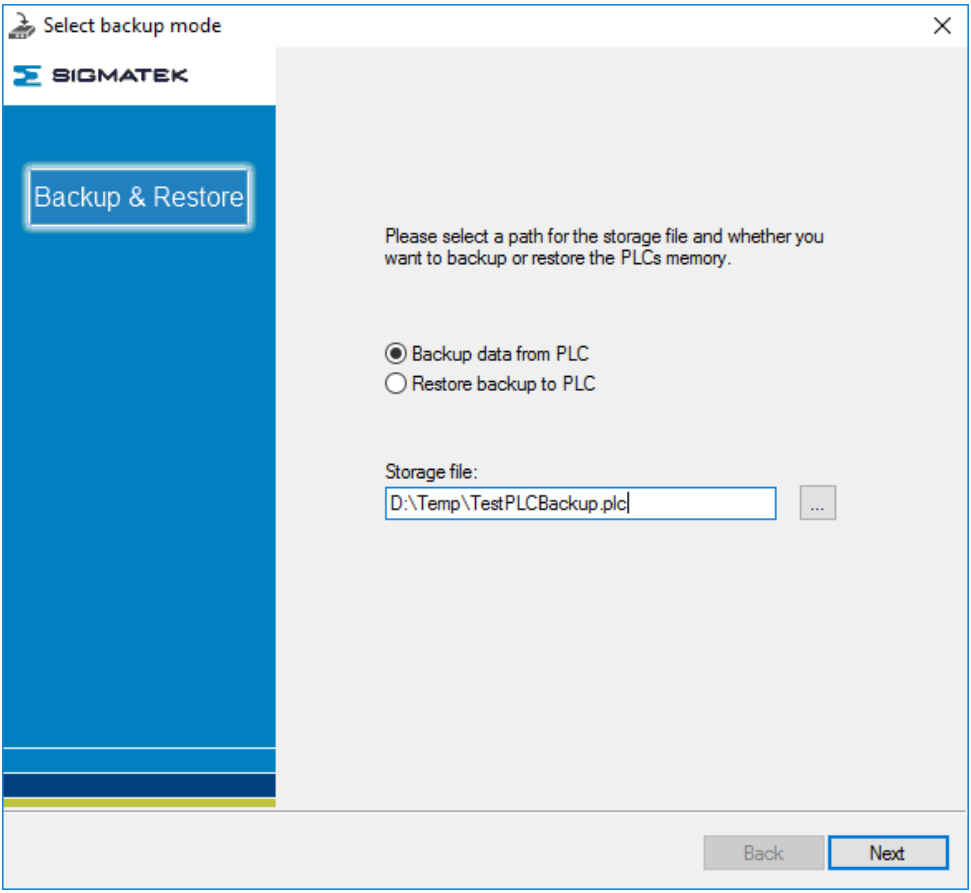


4.8.2.12 PLC Backup and Restore

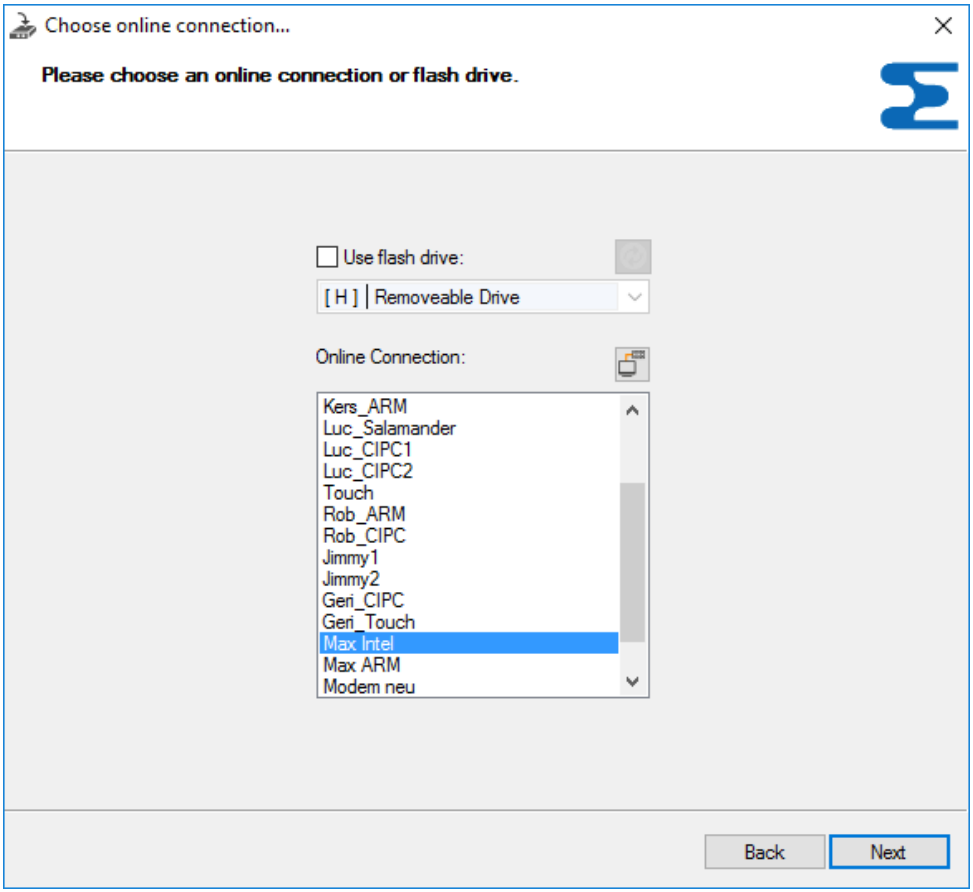
With this tool it is possible to save all configurations of a PLC resp. to restore the backup.



Note that **only PLCs with a file system** can be backed up. The project selection is always the project stored on the PLC with Save Project on PLC. This is not necessarily the **same as the project currently running on the PLC**!

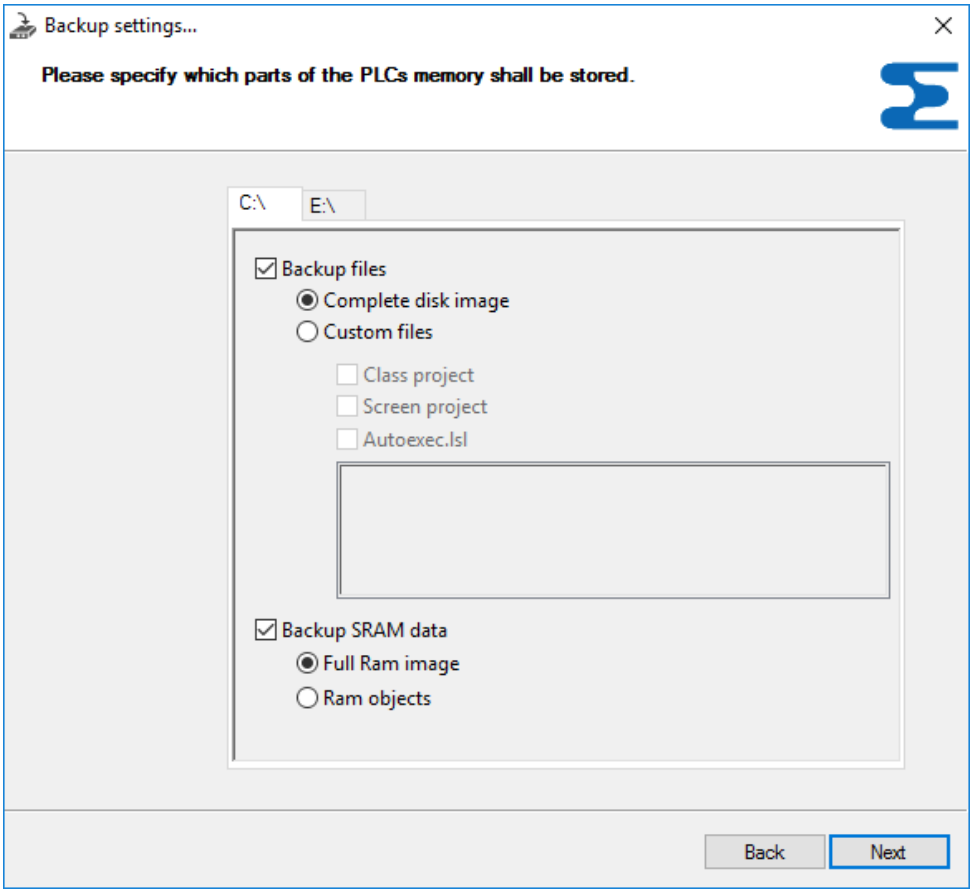


With the **Next** button, a dialog opens in which - if there is no existing connection - an online connection or card reader drive can be selected.

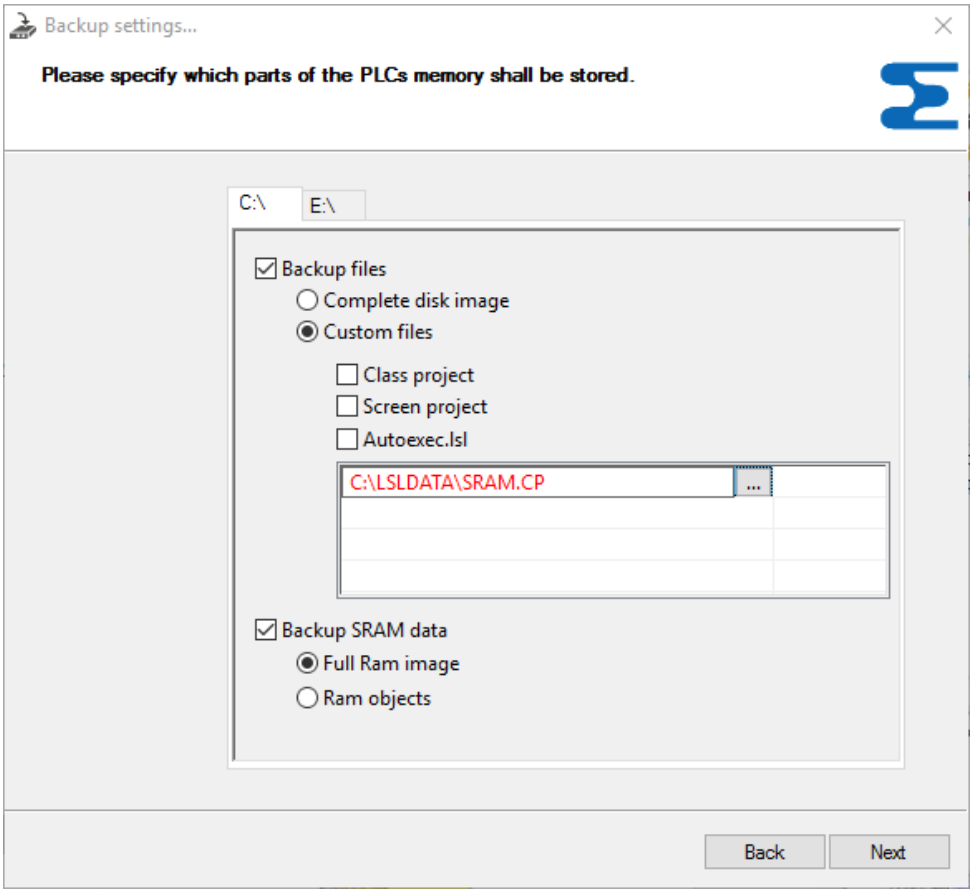


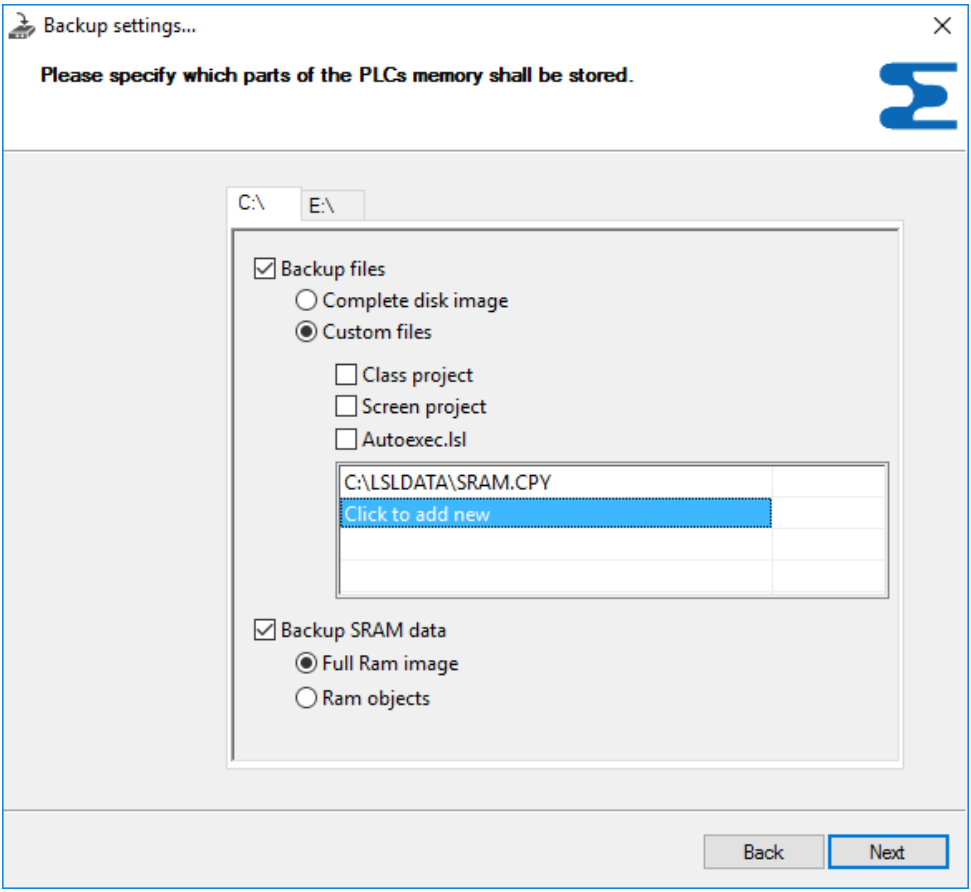
With the **Next** button, a further dialog opens in which the contents to back up can be configured.

If an online selection is now selected and several drives are available on the control, they are displayed as individual tabs:

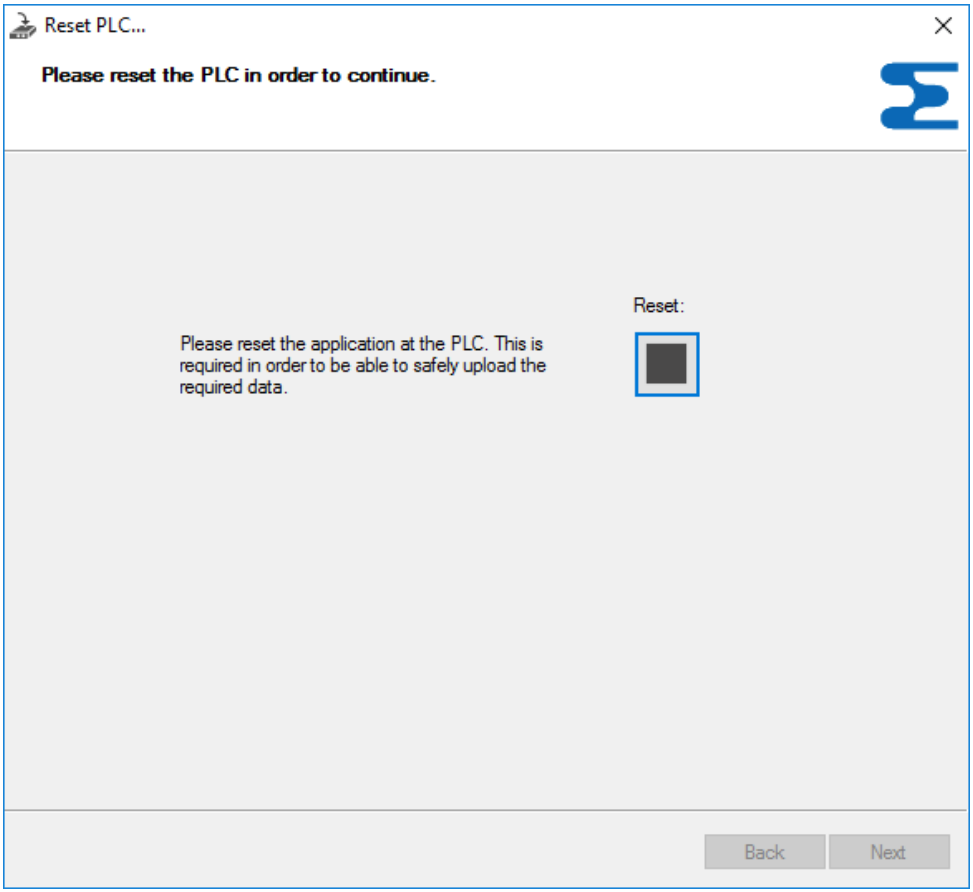


If **Custom files** is selected, specific files can be stored in addition to the pre-configure options. When the file name is entered, the path is shown in red (this means the input was not assumed) until the last character of an existing file path including the file name and extension is input correctly:

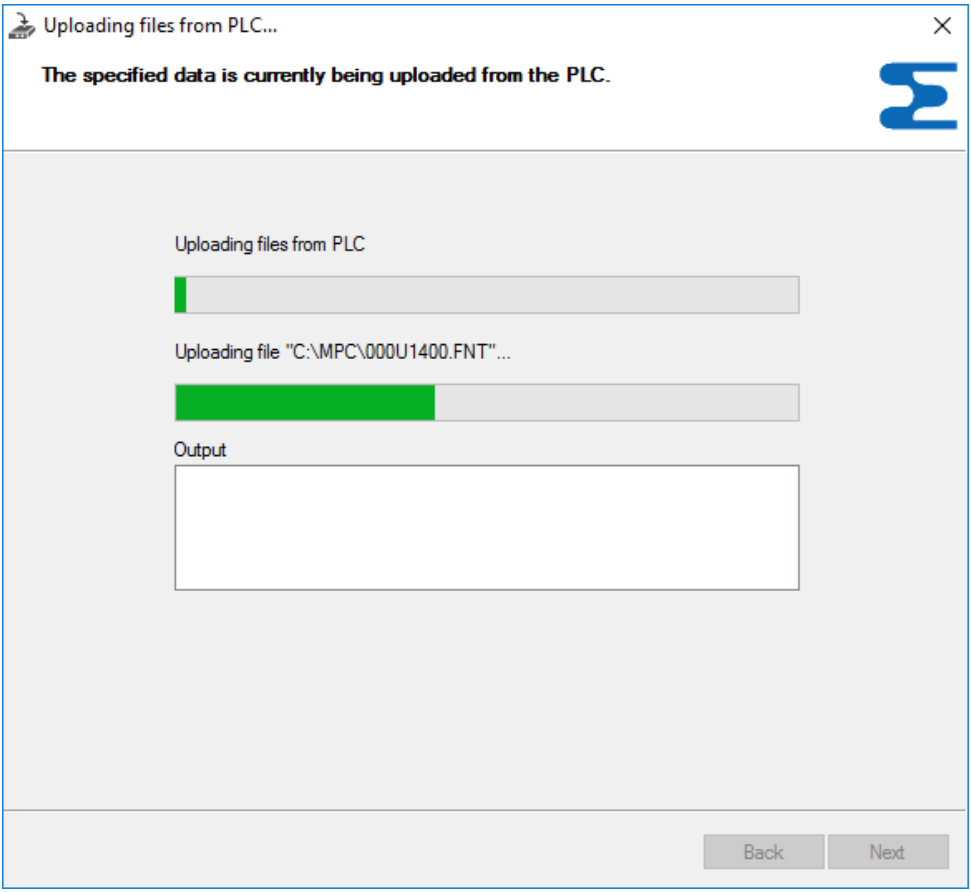

Backup settings...
Please specify which parts of the PLCs memory shall be stored.
C:\
E:\
Backup files
Complete disk image
Custom files
Class project
Screen project
Autoexec.lsl
C:\LSLDATA\SRAM.CP
...
Backup SRAM data
Full Ram image
Ram objects
Back
Next



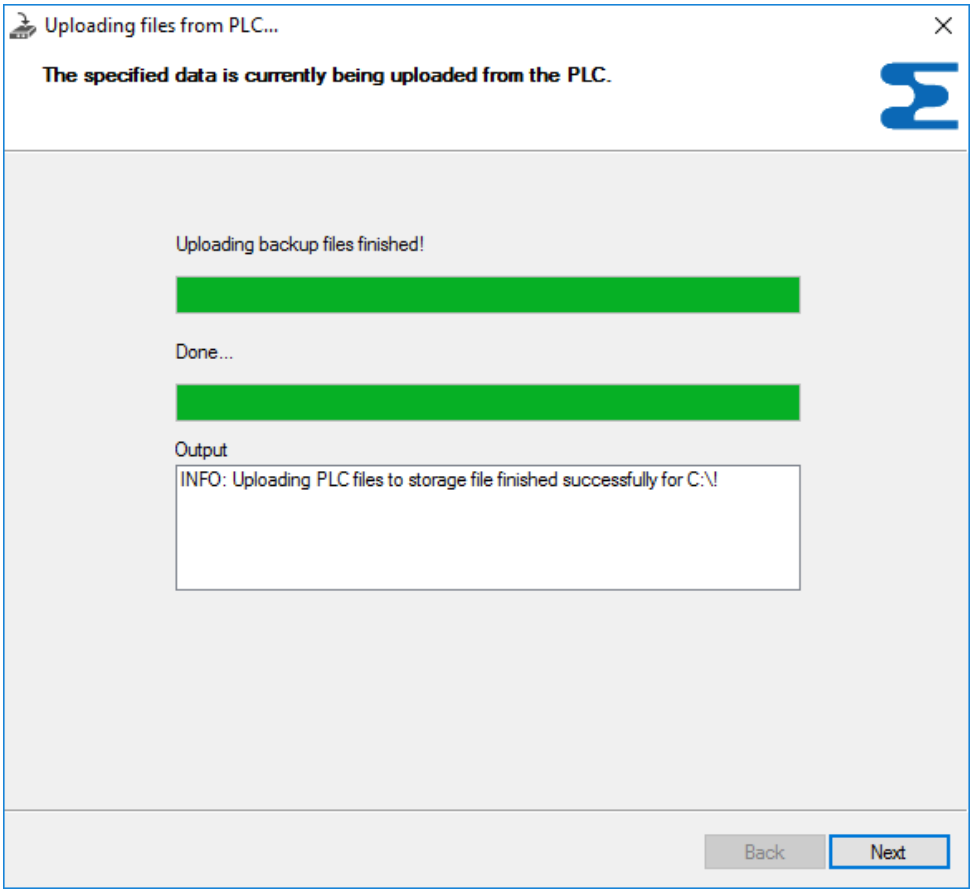
After Next - and in cast the project is running on the Control – the user is instructed to restart the PLC:



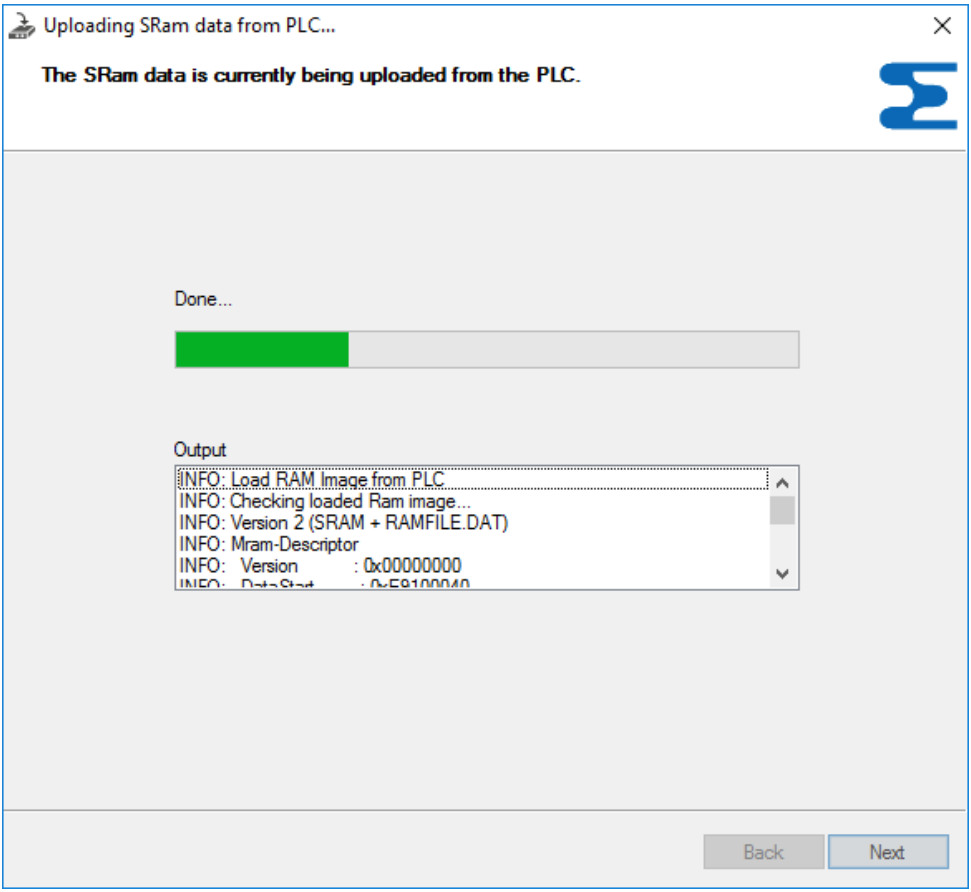
With **Next**, the backup / restoration is stored.



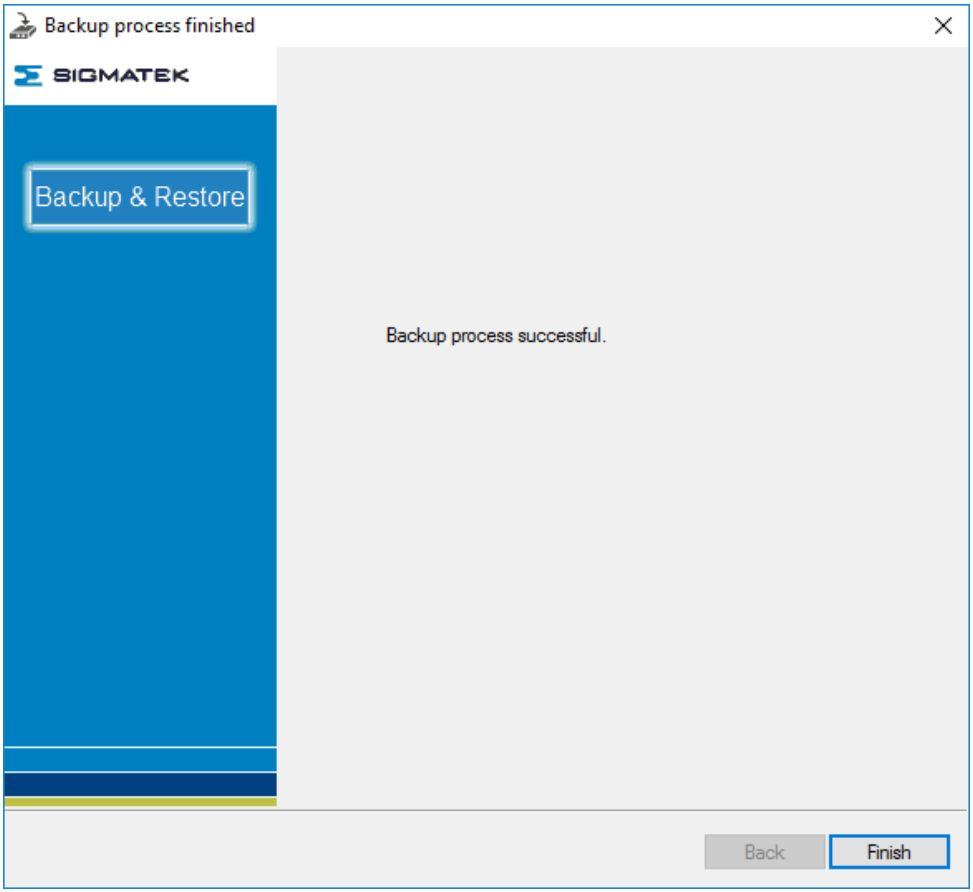
If the backup is successful, the following message appears:



If the Backup SRAM Data option is selected, the following dialog also appears:



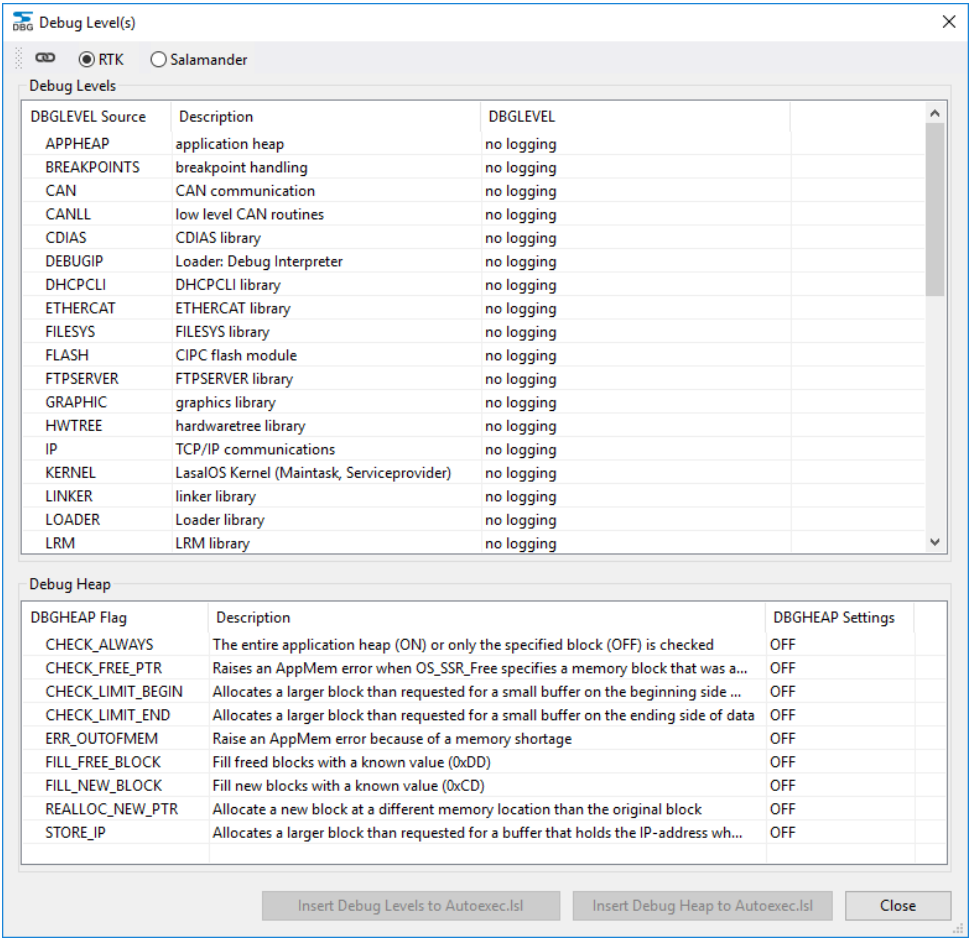
At the end of the procedure, the **Backup process finished** dialog appears:



Press **Finish** to complete the backup.

4.8.2.13 PLC Debug Level

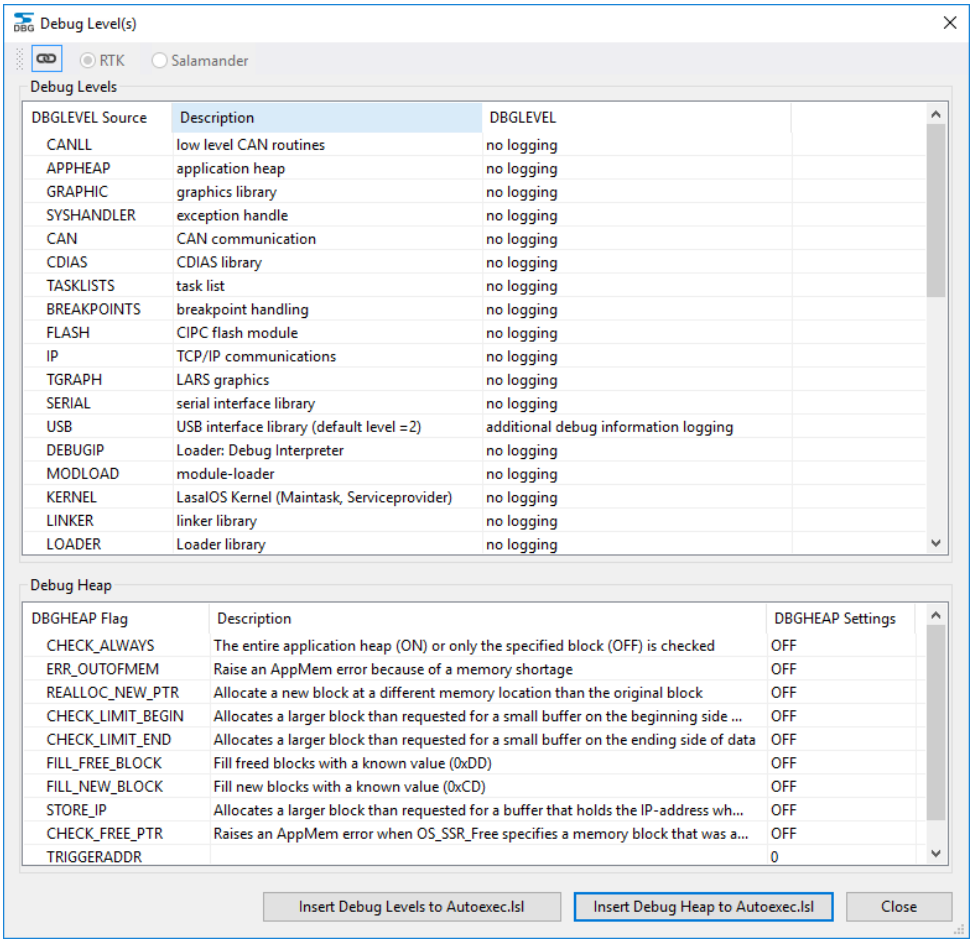
With this tool it is possible to define the debug levels and to set the debug heap flags.



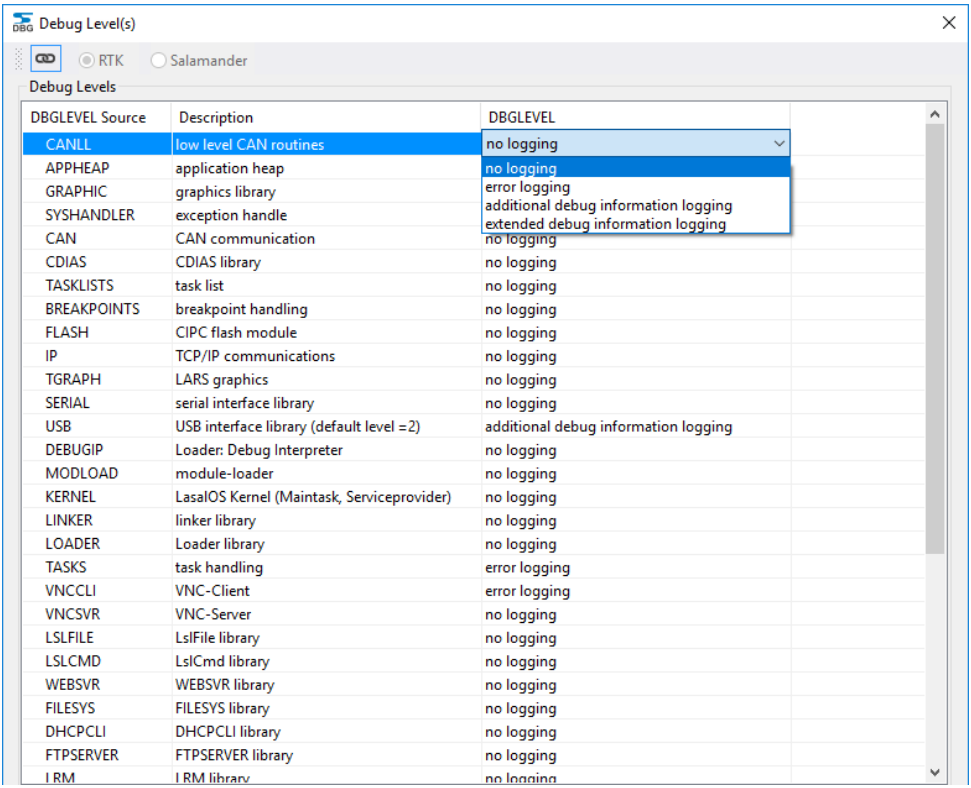
In the upper part of the dialog we see the already known online button and radio buttons for RTK and Salamander.



In offline status you can switch between the operating systems RTK and Salamander. If you are online, the radio buttons are grayed and automatically the button of the connected hardware is set.



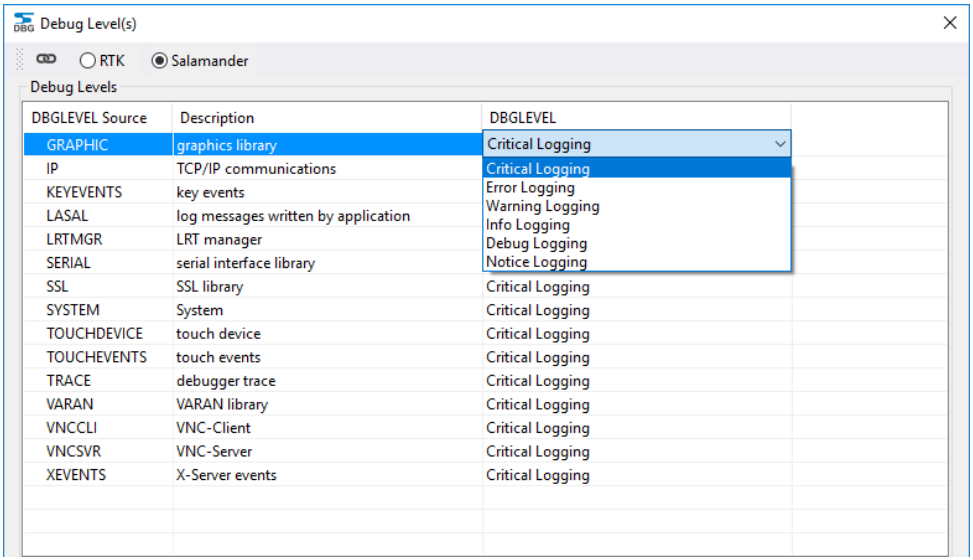
In the next part (Debug Levels), the debug levels can be configured. Only the third column (DBGLEVEL) can be edited.



Debug levels in an Rtk PLC

The available options for Rtk PLCs are:

- no protocol (Level 0)
- protocol errors only (Level 1)
- also protocol additional debug information (Level 2)
- protocol extended debug information (Level 3)

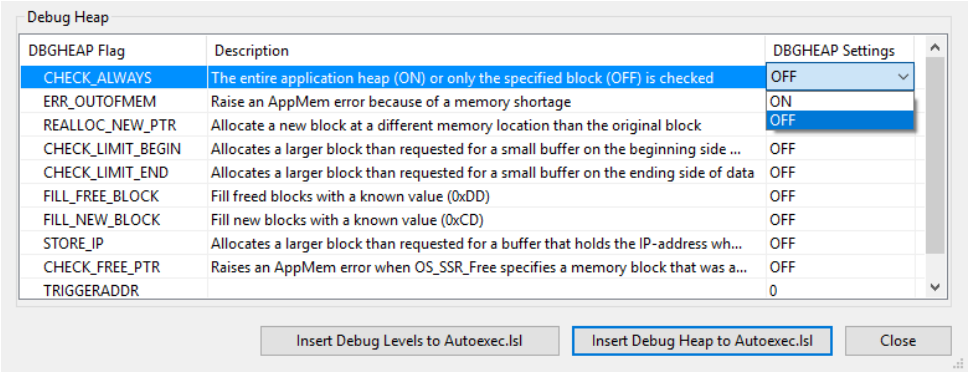


Debug-Levels bei Salamander-PLCs

The available options for Salamander PLCs are:

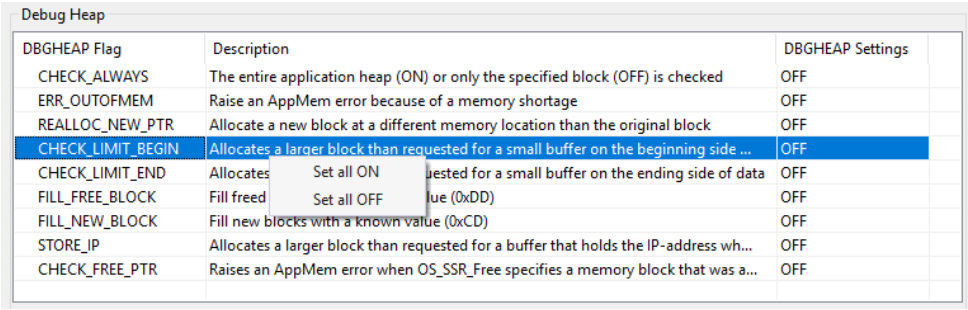
- only critical error protocols (Level 0)
- protocol of all alarms and errors (Level 1)
- include warning protocol (Level 2)
- include comment protocol (Level 3)
- include protocol of additional debug information (Level 4)
- protocol of expanded debug information (Level 5)

In the lower part, the debug heap levels can be set.



As these are flags, there is only ON resp. OFF as values.

It is possible to set (ON) or reset (OFF) all DBGHEAP flags at once by right-clicking on one of the entries.



These flags are only used for debugging. They can help to find memory errors, among other things:

- Realloc or release with an invalid pointer, or a pointer to a memory block that has already been released.

- Insufficient memory (if not handled correctly by the application!)

- An application that uses the old pointer and returns another one to realloc.

- Write before, or after an assigned memory block

- Writing to a memory block that has already been released

Since the application heap is checked for errors every time (alloc, realloc, free) during operation, the use of heap operations can be very time-consuming. If the application uses several heap operations, it can be slowed down considerably - even more: the excessive use of heap operations can possibly even lead to a runtime error! Especially when using

visualization, the display and the reaction speed of screens or Windows can be slowed down considerably by using heap operations.

If the heap is used extensively, it is recommended to use SET DBGHEAP ALL ON when developing an application. If the development was finished and no further memory errors could be found, the heap check should be deactivated for production.

It is possible to set (ON) or reset (OFF) all DBGHEAP flags at once by right-clicking on one of the entries. (In the CLI form it is the ALL option, see below).

Activating DBGHEAP flags can have various effects on the runtime behavior and memory consumption (see the description of the individual flags).

Syntax of Flag Commands **in CLI Format: SET DBGHEAP Flag Name ON|OFF**

SET DBGHEAP without option (CLI-Format only)

Description: Shows the current settings of the DBGHEAP flags.

ALL (CLI-Format only)

Description

ON: Activates all the above DBGHEAP options.

OFF: Disables all above DBGHEAP options.

Runtime behavior:

ON: the demand of the individual options adds up.

OFF: no effect.

Memory consumption:

ON: the demand of the individual options adds up.

OFF: no effect.

CHECK_ALWAYS

Description

ON: The entire application heap is checked for integrity at each heap function call.

OFF: Only the block specified in OS_SSR_ReAlloc or OS_SSR_Free is checked against the other DBGHEAP settings.

Runtime behavior:

With each reserving and releasing of a memory block, the entire heap is checked for consistency. This increases the CPU load. The amount of the additional load depends on the number of already reserved memory blocks, the frequency of reserving and releasing memory blocks and the activation of the flags CHECK_LIMIT_BEGIN,

CHECK_LIMIT_END and FILL_FREE_BLOCK. Especially in big projects this setting can lead to a considerably higher CPU load.

Memory usage:

No effect

CHECK_FREE_PTR

Description

ON: The OS triggers an AppMem error when OS_SSR_Free specifies a memory block that has already been freed.

Runtime behavior:

No effect

Memory usage:

No effect

CHECK_LIMIT_BEGIN

Description

ON: The heap manager reserves a block larger than the required size to check the area **before** the actual data for overwrites.

Runtime behavior:

Slightly higher CPU load when reserving and releasing a memory block. The additional load is constant and does not depend on the size or number of reserved blocks.

Memory usage:

4 additional bytes per memory block.

CHECK_LIMIT_END

Description

ON: The heap manager reserves a block larger than the required size to check the area for overwriting **after** the actual data.

Runtime behavior:

Slightly higher CPU load when reserving and releasing a memory block. The additional load is constant and does not depend on the size or number of reserved blocks.

Memory usage:

4 additional bytes per memory block.

ERR_OUTOFMEM

Description

ON: The OS triggers an AppMem error if OS_SSR_Alloc or OS_SSR_ReAlloc fails due to a lack of memory.

Runtime behavior:

No effect

Memory usage:

No effect

FILL_FREE_BLOCK

Description:

ON: The heap manager fills shared blocks with a known value (0xDD) to check if the shared memory is still being written to.

Runtime behavior:

Higher CPU load when reserving and releasing a memory block. The additional load depends on the size of the memory block.

Memory usage:

No effect

FILL_NEW_BLOCK

Description:

ON: New blocks are filled with the value 0xCD during assignment.

Runtime behavior:

Higher CPU load when reserving and releasing a memory block. The additional load depends on the size of the memory block.

Memory usage:

No effect

REALLOC_NEW_PTR

Description:

ON: The OS_SSR_ReAlloc function **always** reserves a new block in a different location than the original one.

Runtime behavior:

Can lead to a higher CPU load when reserving and releasing a memory block because of a higher internal fragmentation of the heap. The additional load depends on the number of reserved blocks.

Memory usage:

Can lead to a higher memory usage when reserving and releasing a memory block because of a higher internal fragmentation of the heap. The additional load depends on the number of reserved blocks.

STORE_IP

Description:

ON: The heap manager reserves a block larger than the required size to store in the additional area the instruction pointers (IP) of the functions that called the heap function. In case of an AppMem error, the IP of the faulty block is written to the system log.

Runtime behavior:

Slightly higher CPU load when reserving and releasing a memory block. The additional load is constant and does not depend on the size or number of reserved blocks.

Memory usage:

8 additional bytes per memory block.

TRIGGERADDR <addr>

Description:

If <addr> is not equal to zero, the functions MALLOC and REALLOC write the CallStack into the event log file. This is useful, for example, if the address of a damaged block is known and the function that reserved it is to be determined. The code can then be examined to determine what happens to the memory block.

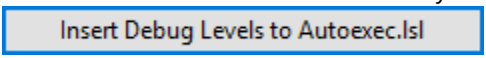
Runtime behavior:

If **this** memory block is currently allocated, an entry is made in the event log file. If the block is then released - by the user - and allocated again, an entry is of course created again. The used runtime can add up noticeably under these conditions.

Memory consumption:

No effect.

When the settings have been defined, they can then - but only in offline mode - be entered into the Autoexec.lsl file by clicking the corresponding button



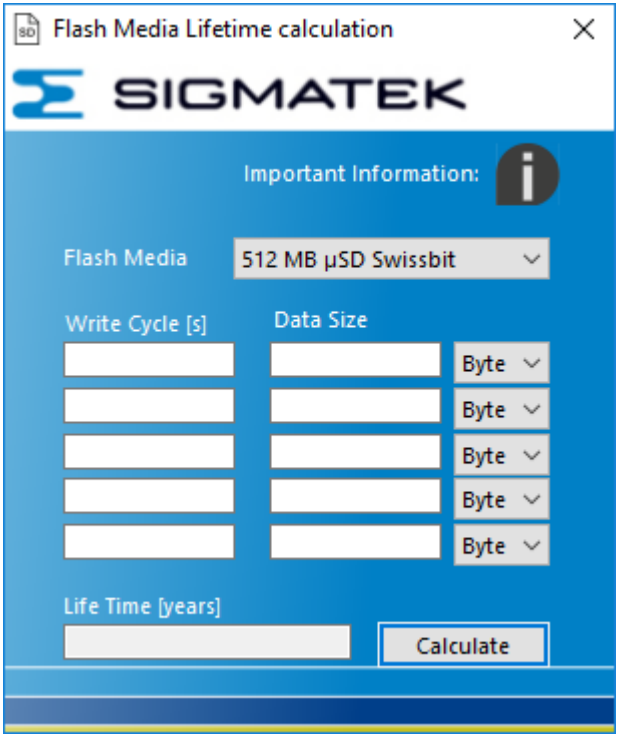
resp.

Insert Debug Heap to Autoexec.lsl. From here on, the settings are used with every start.

The debug heap functions are available from LasalOS 5.52.
The option TRIGGERADDR is available for LasalOS 1.1.51 and higher.

4.8.2.14 Flash Media Lifetime Calculation

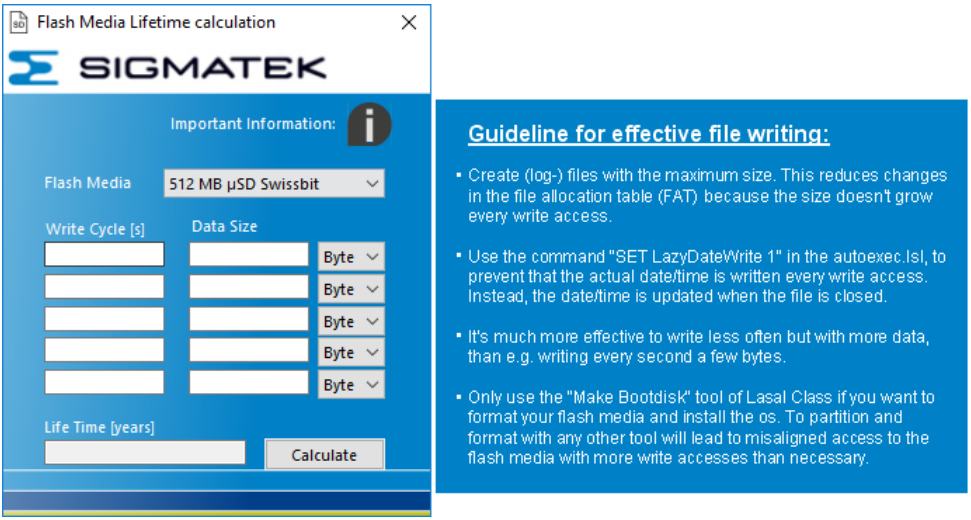
With this tool, the approximate life expectancy or the Flash memory card or NAND memory (permanently soldered memory) can be calculated.



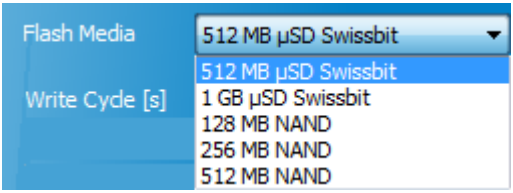
Memory cards or NAND memory are not infinitely writable. The expected lifespan depends on several factors: manufacturer, rated memorc capacity, page and block size; the number and length of the write access are only a few of the parameters to consider.

SIGMATEK supplies µSD memory cards from Swissbit in the sizes 512 MB and 1 GB, as well as hardwired NAND memory in three different sizes. For these card sizes, the manufacturer must provide the information required to calculate the expected lifespan.

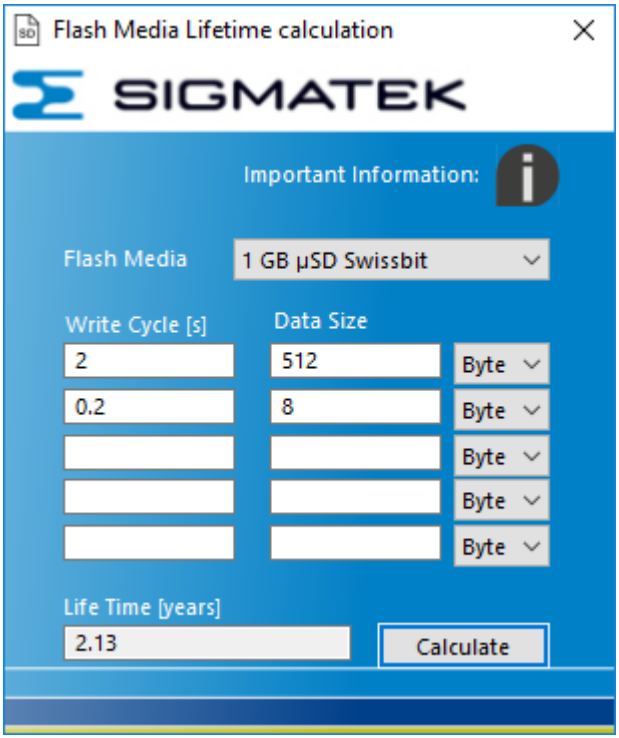
When the  button is clicked, useful tips on how the increase the lifespan of the card with efficient writing are provided:



In the field “Flash Media”, the memory size is selected:



After entering the write cycles with the respective file size and pressing the calculate button, the expected lifespan is displayed (in years).


Flash Media Lifetime calculation
SIGMATEK
Important Information:
Flash Media
1 GB µSD Swissbit
Write Cycle [s]
Data Size
2
512
Byte
0.2
8
Byte
Byte
Byte
Byte
Life Time [years]
2.13
Calculate

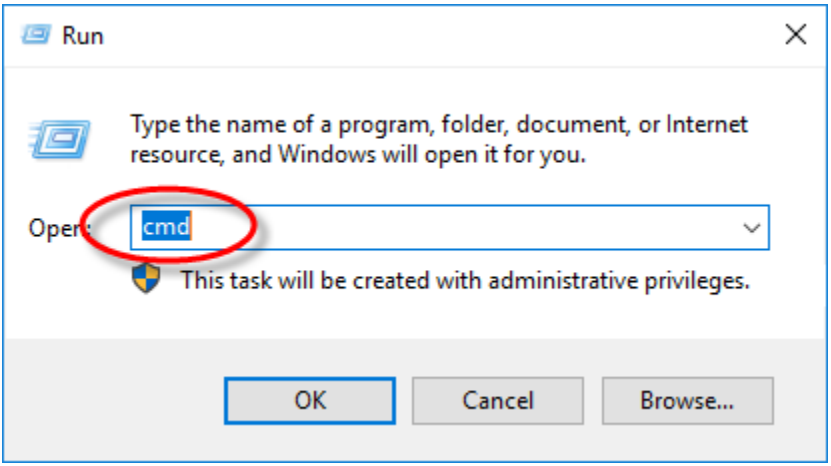
4.9 The Encryption Tool "Encrypter.exe"

In the chapter "The Project Connection Files Window" the possibility was discussed to **encrypt files** in the project tree under the class context menu item **Protection Settings** via the third tab **Connection Files**. But this can be done without this detour, directly from the command shell. For this purpose the tool **Encrypter.exe** was developed, which is located in the installation directory of LASAL CLASS 2, in the same \bin folder where the file Lasal2.exe is located. To start Encrypter.exe, proceed as follows:

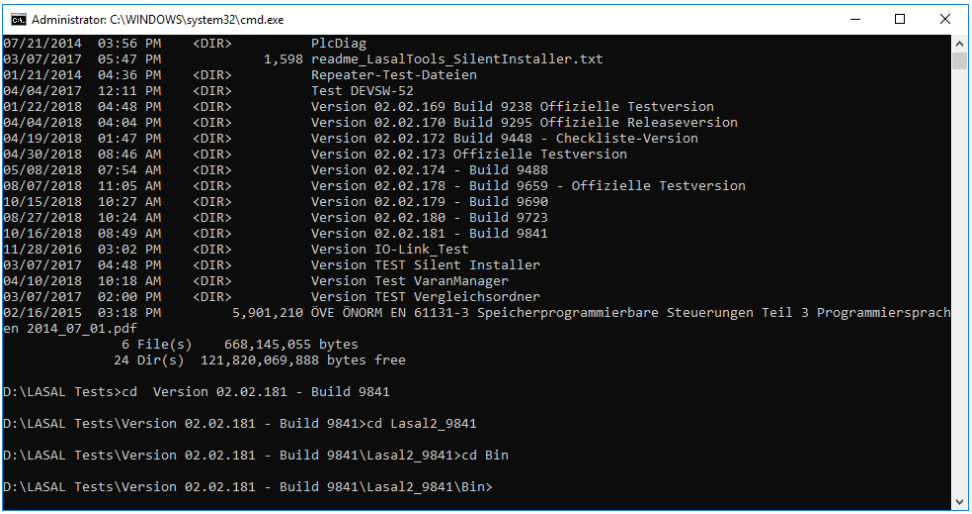
- Start the command shell by right-clicking the Windows icon on the left side of the taskbar and selecting **Run**:


Apps and Features
Power Options
Event Viewer
System
Device Manager
Network Connections
Disk Management
Computer Management
Windows PowerShell
Windows PowerShell (Admin)
Task Manager
Settings
File Explorer
Search
Run
Shut down or sign out
Desktop

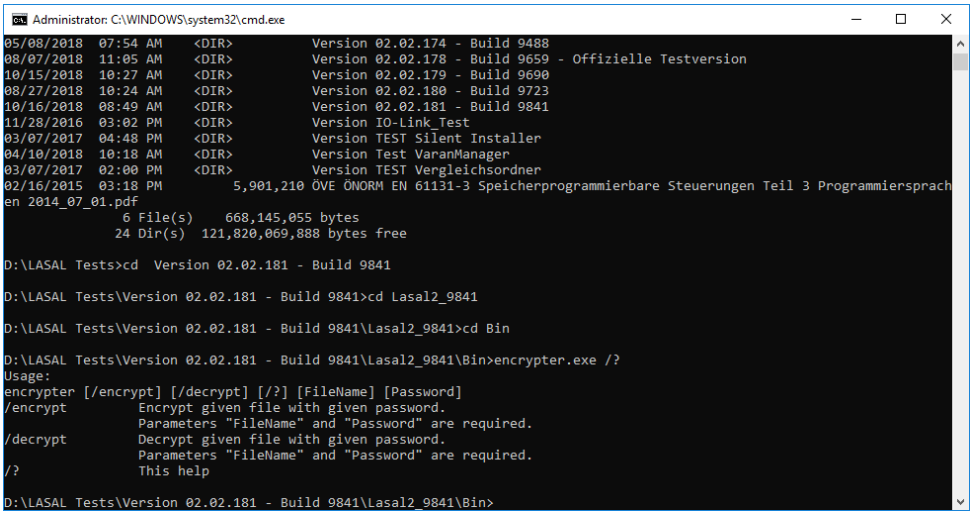
- Type **cmd** in the field in the window that opens.



- In the command shell, navigate to your LASAL CLASS 2 installation directory and then to the ...\bin\ directory.



- Start Encrypter.exe with the help attribute /?.



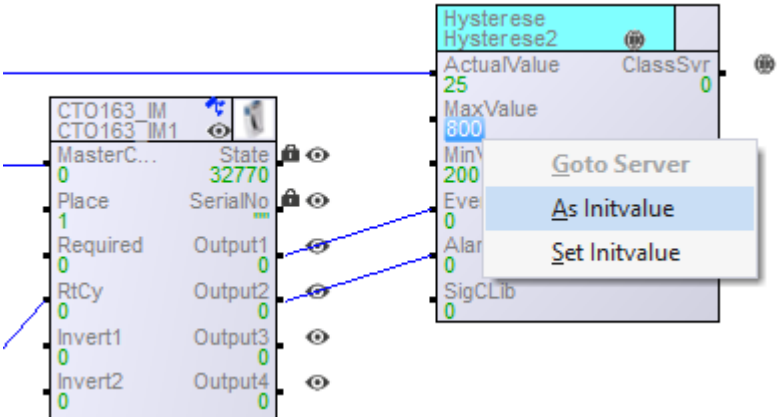
Now the syntax is explained. **encrypter /encrypt MyFileName MyPassword encrypts** the specified file using the specified password; with **encrypter /decrypt MyFileName MyPassword** it is also **decrypted**, where both the **filename** and the **password** are **mandatory** for both processes.

5 TIPS AND HELPS FOR WORKING WITH LASAL CLASS 2

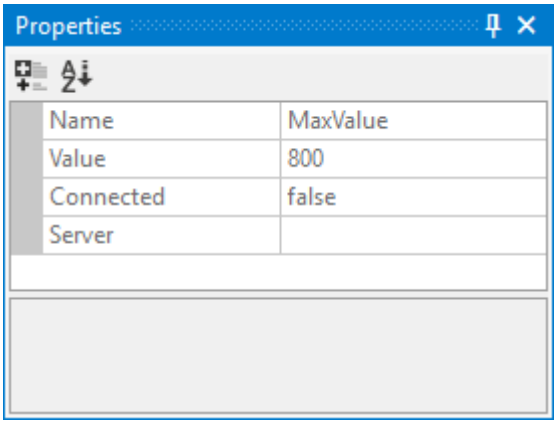
5.1 Using Online Values as Initial Values

In LASAL CLASS 2, it is possible to enter initial values in clients and servers during online operation. These values remain stored even after the control has be stopped and restarted. It is also possible to overwrite the online values with those stored in the class.

To enter an initial value in a client during online operation, select **As Initvalue** in the context menu of the client.

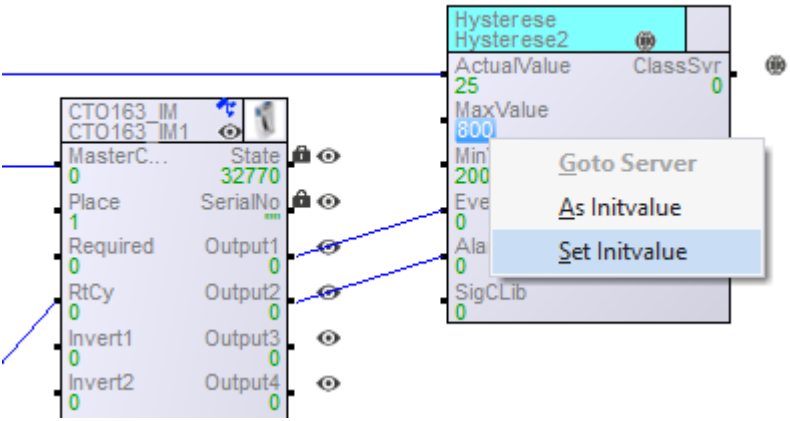


The entry specified as the initial value is then stored in LASAL CLASS 2.

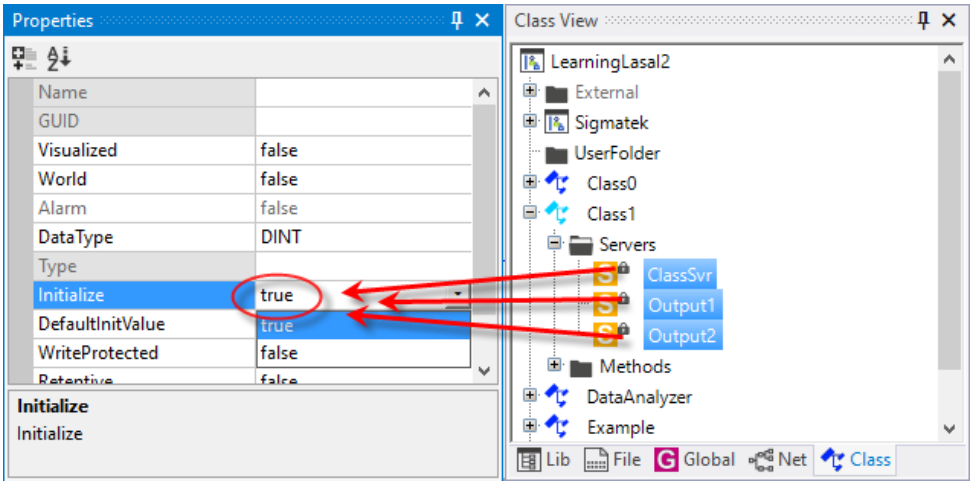


So that the stored value is loaded to the target system, the project must first be compiled and the changes transferred to the target system. If the program in the target system is the stopped and restarted the stored values are retained.

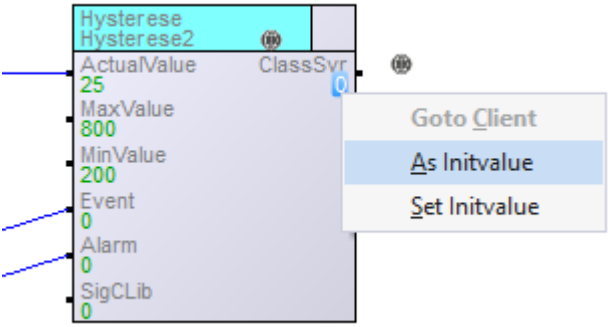
The opposite is also possible: the online value is overwritten with that in the class server via the menu item **Set Initvalue**.



To define initial values for servers during online operation, Initialize must be set to **True** for each server in the **Properties window**. This can also functions at the same time: simply mark sever server at the same time!



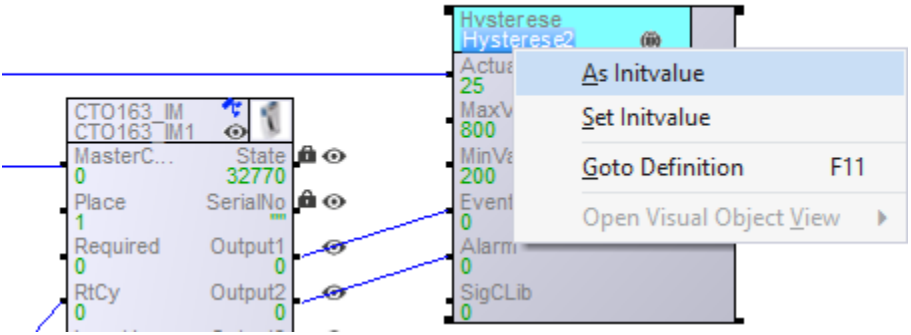
As in the client, the Init value of a server is stored by selecting **As Initvalue** in the context menu or with **Set Initvalue**, replaced by the value stored in the class.



With a right-click on an object Initvalues can also be adopted for the complete object **(As Initvalue)** or by the Init value overwritten from the class (Set Initvalue)

Caution!

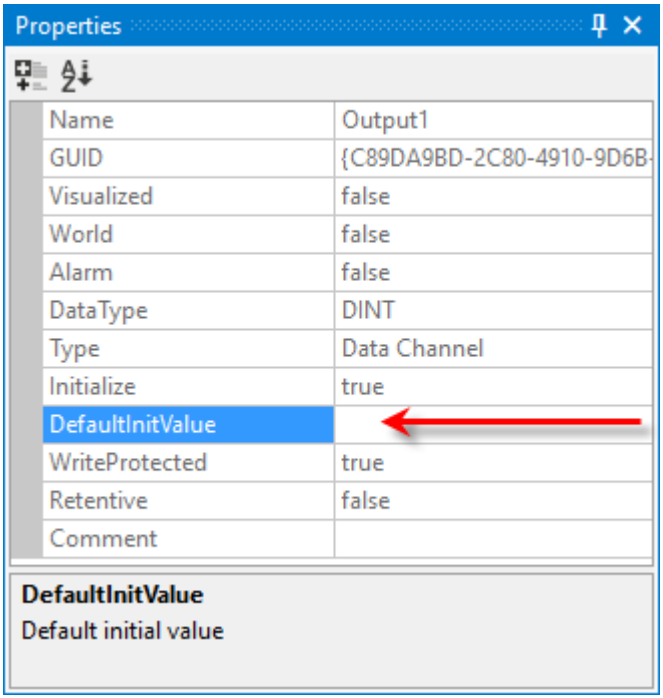
All clients and initializable servers assume the actual online values as initialization values (**As Initvalue**) or are overwritten by the class values (**Set Initvalue**)!



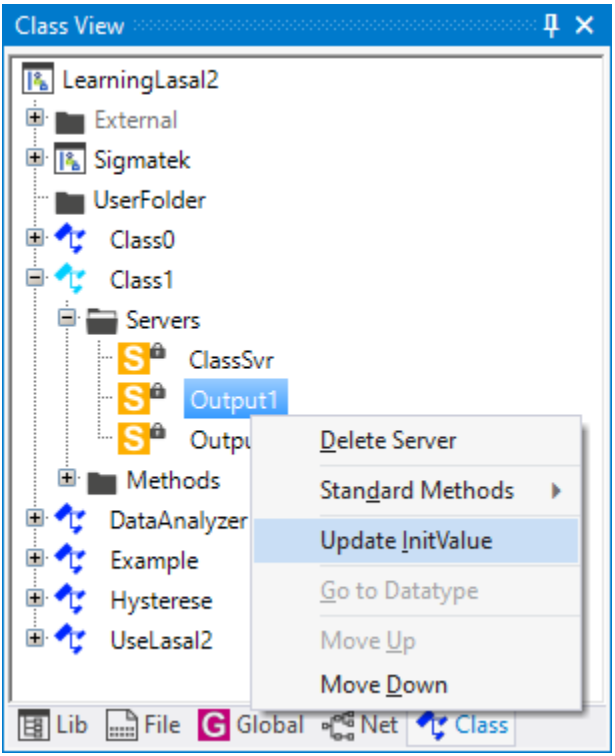
As with servers /clients, objects can also be changed using **As/Set Initvalue** via the multiple selection of several values.

5.2 Transfering Init Values from Servers/Clients to Classes of all Existing Objects

In offline mode, a server / client of a class can be initialized.

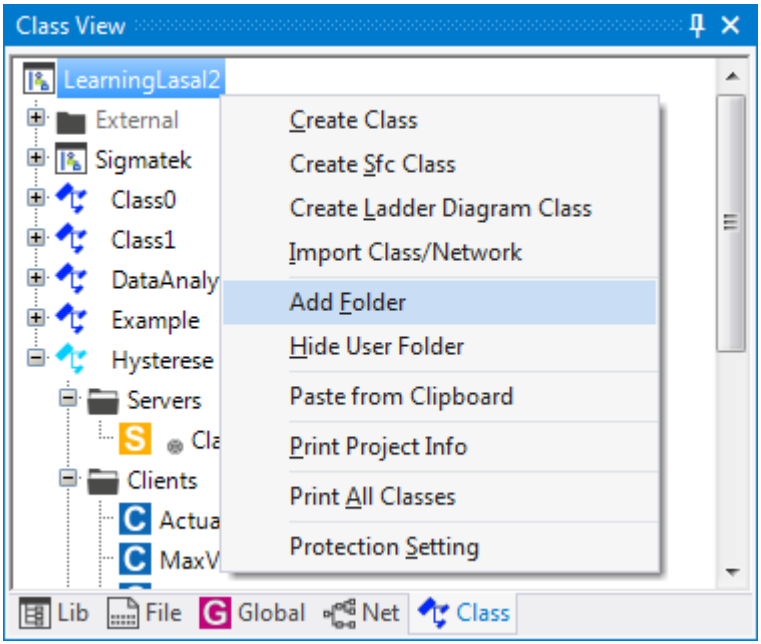


Afterwards, it is possible with a right click to transfer the Init value to the server/client of all existing instances.

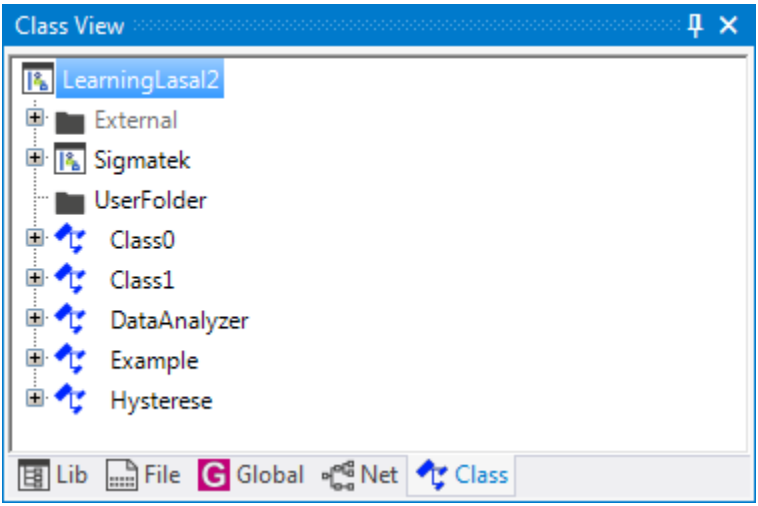

Class View
LearningLasal2
External
Sigmatek
UserFolder
Class0
Class1
Servers
ClassSvr
Output1
Outpu
Methods
DataAnalyzer
Example
Hysterese
UseLasal2
Delete Server
Standard Methods
Update InitValue
Go to Datatype
Move Up
Move Down
Lib
File
Global
Net
Class

5.3 User-defined Folders in the Project Tree

If there are several classes in a project, they can be divided into folders for clarity. To create a folder, select **Add Folder** from the context menu of the project.

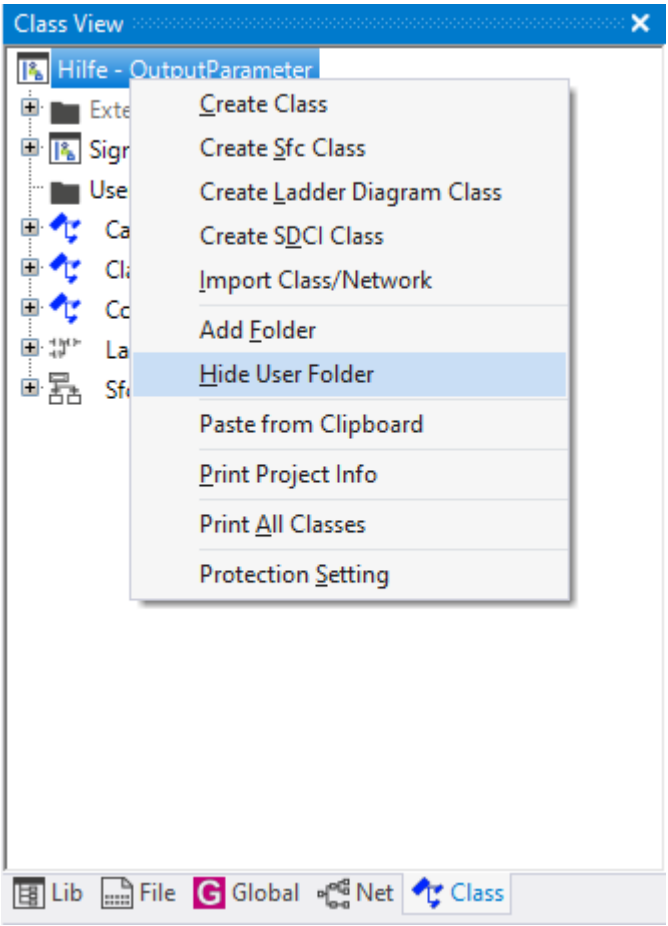


Next assign a name to the folder (here: **Complex**) then **Drag & Drop** the desired class in the new folder (here: **Application** and **Connection**).



To remove a class from a folder and place it back into the original project structure, select the desired class and place it on the project name.

If you like to hide the user-defined folders temporarily in order to alphabetically sort the classes in a hierarchy level, simply select the **Hide User Folder** command in the context menu of the project. In this view classes in the tree can be easily activated through input of the initial letters.



In the network tree and with global types in the global tree, folders can also be created to group elements. It is also possible to create connection file folder in the Dias Drive Files folder in the file tree.

5.4 Removing of Duplicate GUIDs

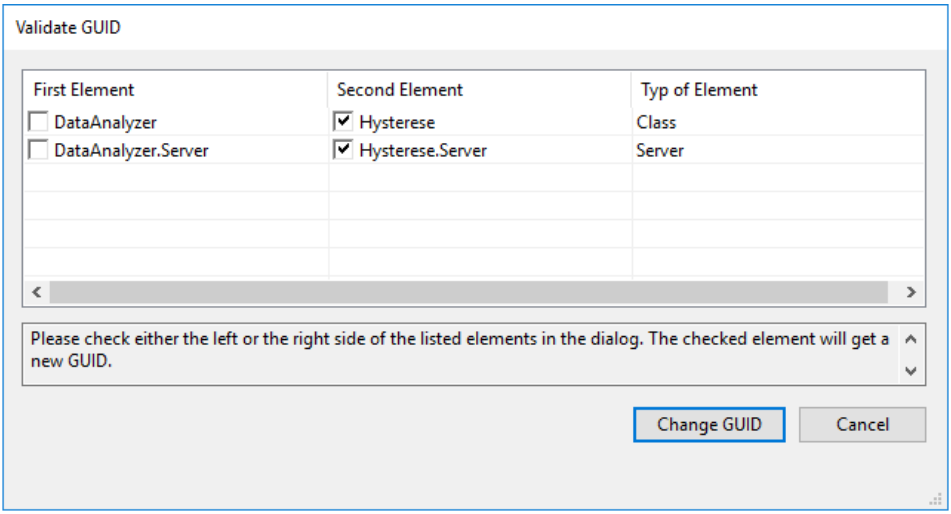
In LASAL CLASS 2 each class, each server of a class as well as each object has a GUID (Globally Unique Identifier). The GUID is assigned once at creation of the corresponding element and consecutively remains equal, also at renaming of an element. This GUID is never used for LASAL CLASS 2 processes but enables the server import to the LASAL SCREEN Editor (LSE), also with renamed elements.

A GUID defined from Windows is always unique. Although at the following processes duplicate GUIDs can appear:

- Create class X in project A
- Copy class X in project B
- Rename class X to Y in project B
- Copy class Y in project A

This process results two classes (X and Y) in project A. Both classes have the same GUID (also the server of this class have the same GUID).

LASAL CLASS 2 checks this cases at project loading and class import and displays a warning in the output window at positive result. The warning includes the instruction to solve the duplicate GUIDs, even through the **Validate GUID** dialog opened via the **Project** menu.

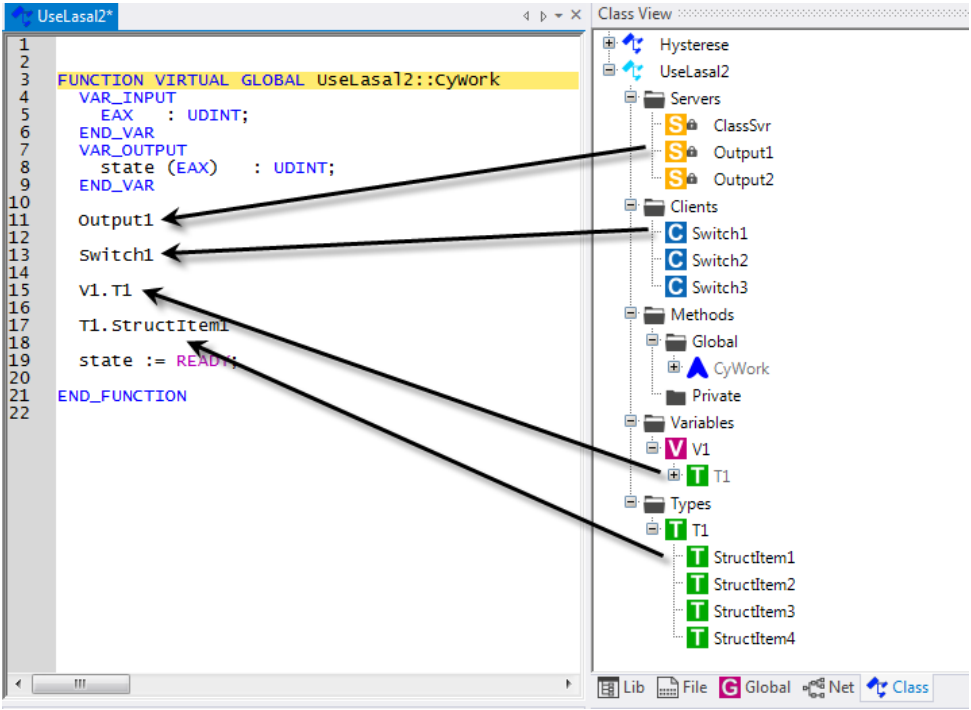


A double click on the error message in the Output window does the same.

Please select the first or second element per line via the checkbox and assign the element a new GUID through the **Change GUID** button. The GUIDs of linked classes and SIGMATEK classes cannot be changed.

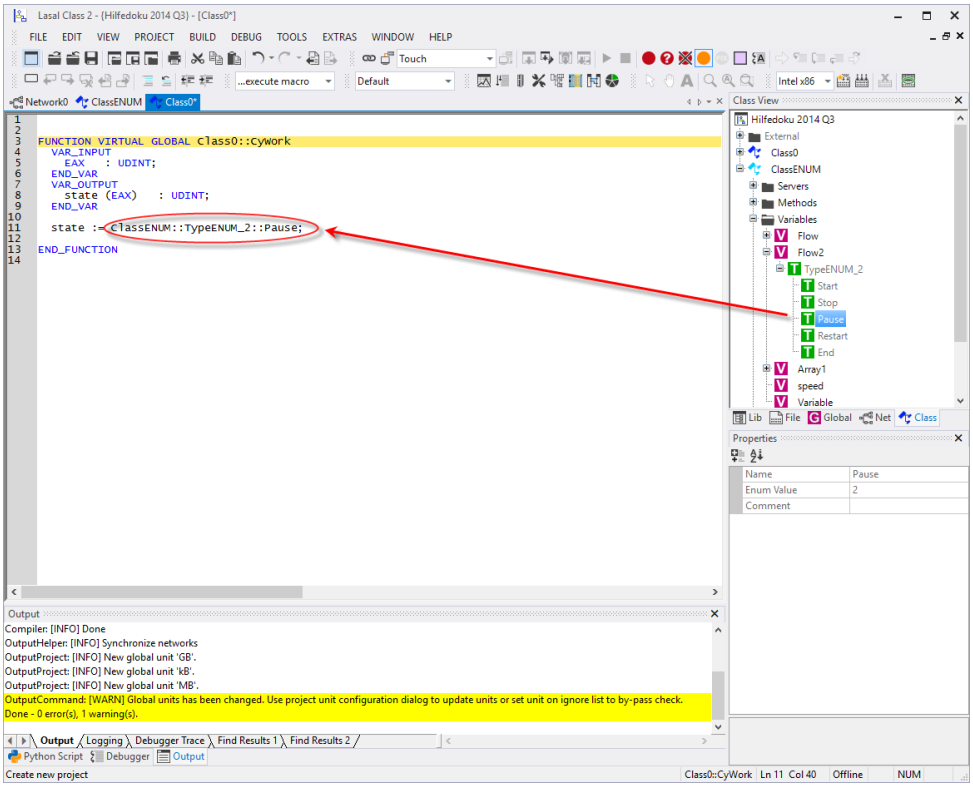
5.5 Drag & Drop in the Program Code Editor

An important additional help to avoid typing errors when creating program code is Dragging & Dropping components directly from the project tree into the program code editor.



Simply mark the required element with the left mouse button (Client, Server, Variable,...) and pull it into the program code editor at the desired location.

If an ENUM item from a different class is placed in the code via Drag & Drop, the class and Type scopes are also added.


Lasal Class 2 - [Hilfedoku 2014 Q3) - [Class0*]
FILE EDIT VIEW PROJECT BUILD DEBUG TOOLS EXTRAS WINDOW HELP
Network0 ClassENUM Class0*
FUNCTION VIRTUAL GLOBAL Class0::CyWork
VAR_INPUT
EAX : UDINT;
END_VAR
VAR_OUTPUT
state (EAX) : UDINT;
END_VAR
state := ClassENUM::TypeENUM_2::Pause;
END_FUNCTION
Class View
Hilfedoku 2014 Q3
External
Class0
ClassENUM
Servers
Methods
Variables
Flow
Flow2
TypeENUM_2
Start
Stop
Pause
Restart
End
Array1
speed
Variable
Properties
Name Pause
Enum Value 2
Comment
Output
Compiler: [INFO] Done
OutputHelper: [INFO] Synchronize networks
OutputProject: [INFO] New global unit 'GB'.
OutputProject: [INFO] New global unit 'kB'.
OutputProject: [INFO] New global unit 'MB'.
OutputCommand: [WARN] Global units has been changed. Use project unit configuration dialog to update units or set unit on ignore list to by-pass check.
Done - 0 error(s), 1 warning(s).
Create new project
Class0::CyWork Ln 11 Col 40 Offline NUM

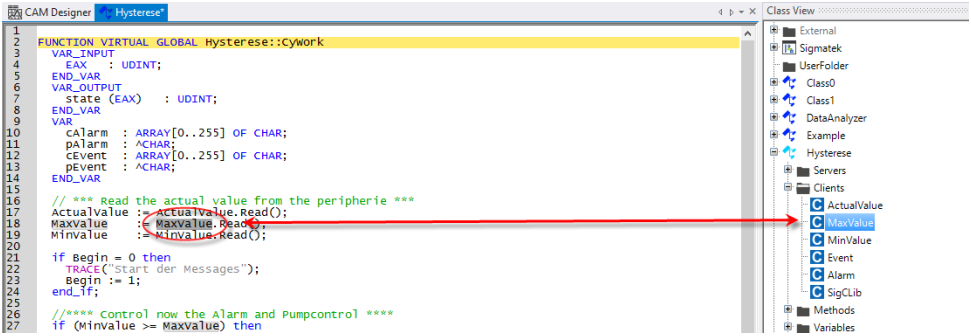
5.6 Go To Definition

With the **Go To Definition** function, program elements used in the program code editor can be located. Simply position the cursor on the data element that should be displayed and open the context menu with a right click.

```
1
2   FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
3     VAR_INPUT
4       EAX    : UDINT;
5     END_VAR
6     VAR_OUTPUT
7       state (EAX)    : UDINT;
8     END_VAR
9     VAR
10      cAlarm   : ARRAY[0..255] OF CHAR;
11      pAlarm   : ^CHAR;
12      cEvent   : ARRAY[0..255] OF CHAR;
13      pEvent   : ^CHAR;
14    END_VAR
15
16    // *** Read the actual value from the peripherie ***
17    ActualValue := ActualValue.Read();
18    MaxValue    := MaxValue.Read();
19    MinValue    := MinValu
20
21    if Begin = 0 then
22      TRACE("Start der Mes
23      Begin := 1;
24    end_if;
25
26    //**** Control now the
27    if (MinValue >= MaxVal
28      Alarm := 1;
29    else
30      Alarm := 0;
31    end_if;
32
33    if (ActualValue >= Max
34      Event := 1;
35    elsif (ActualValue <=
36      Event := 0;
37    end_if;
38
39    pAlarm := sigclib_itoa
40    pEvent := sigclib_itoa
41
42    //**** Messaging ****
43
44    if Alarm = 1 then
```

Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete
Select All Ctrl+A
Go To Definition F11
Toggle Breakpoint F4
Toggle Bookmark Ctrl+F2
Mark Text
Clear Marks
Toggle Function Folding
Insert Client-Update Source Code...

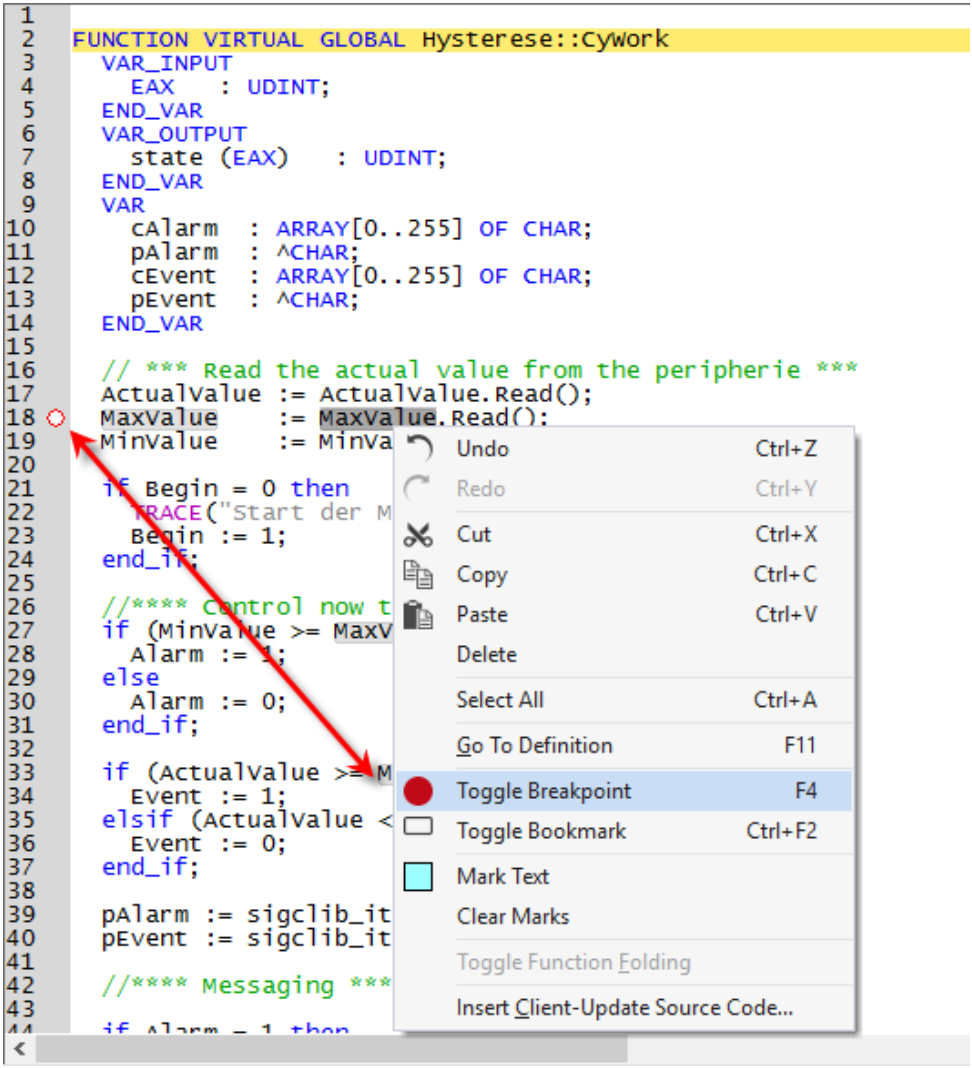
In the context menu, select the **Go To Definition** entry and the data element is located and marked (here: the Client **sMaxFuell** in the project tree).



This way, elements in the program code can be easily located; especially in projects with numerous class elements. The shortcut for **Go To Definition** is **F11**. After **F11** is successful, it is then possible to return to the output position in the program code editor with **Shift + F11**.

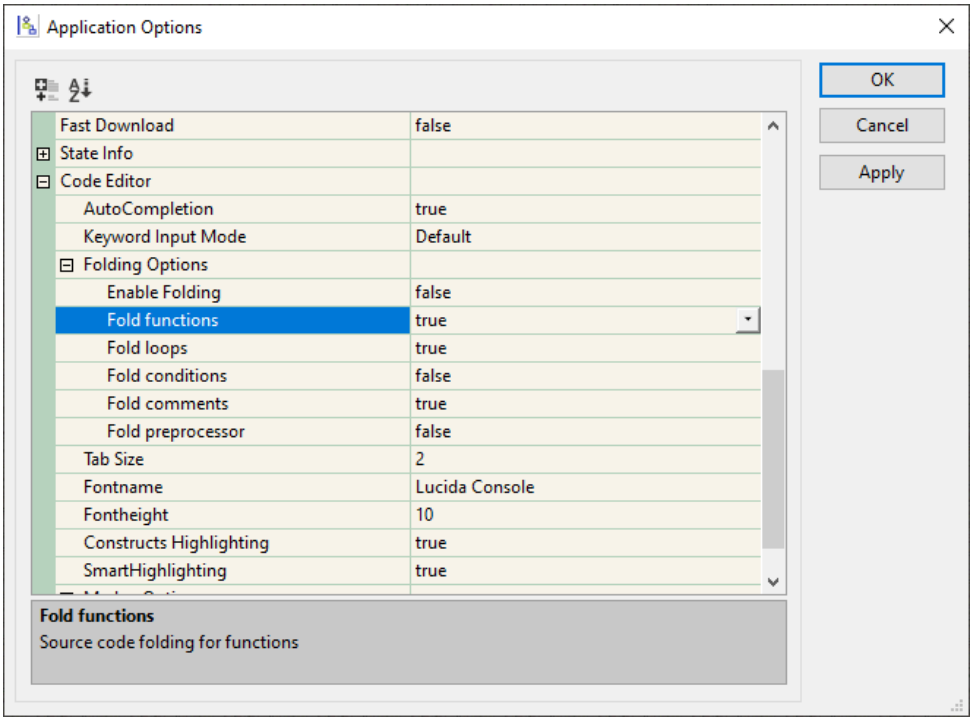
5.7 Toggle Breakpoint / Toggle Bookmark

These functions correspond to the respective functions in the breakpoint (toggle breakpoint or F4) or edit toolbar (toggle bookmark or Ctrl+F2). In the actual code line, the status of the breakpoint / bookmark is changed (enable / disable).

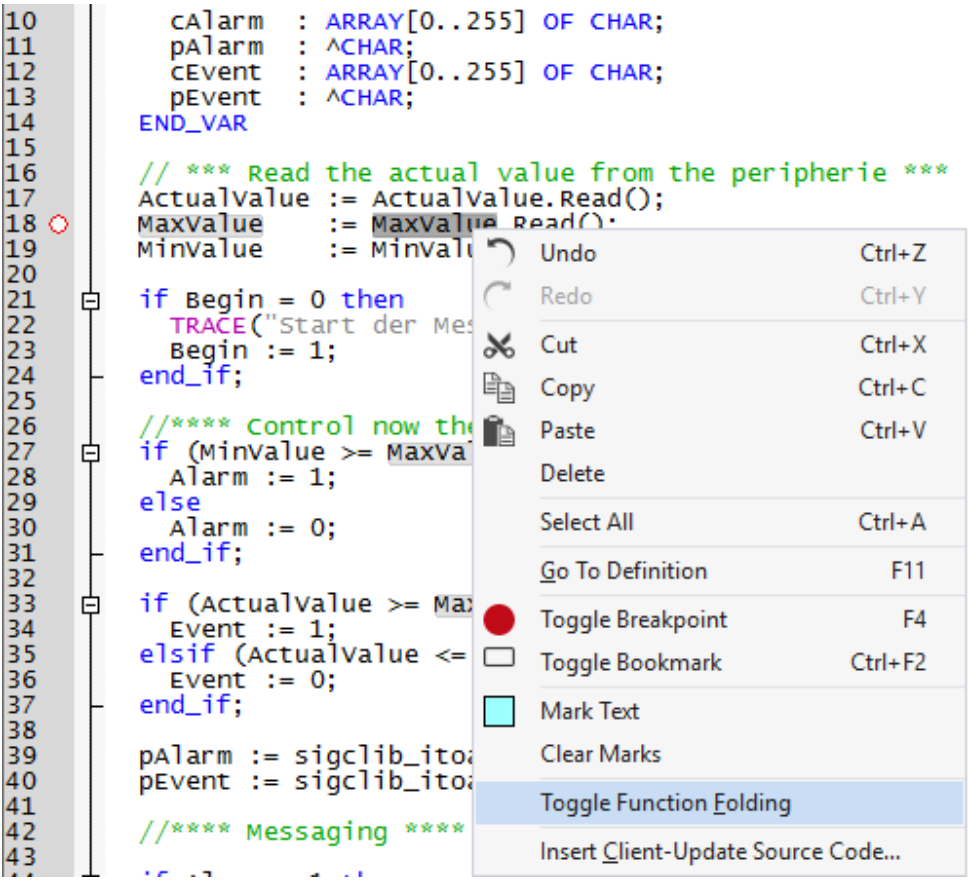

FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
VAR_INPUT
EAX : UDINT;
END_VAR
VAR_OUTPUT
state (EAX) : UDINT;
END_VAR
VAR
cAlarm : ARRAY[0..255] OF CHAR;
pAlarm : ^CHAR;
cEvent : ARRAY[0..255] OF CHAR;
pEvent : ^CHAR;
END_VAR
// *** Read the actual value from the peripherie ***
ActualValue := ActualValue.Read();
MaxValue := MaxValue.Read();
MinValue := MinVa
if Begin = 0 then
TRACE("Start der M
Begin := 1;
end_if;
//**** Control now t
if (MinValue >= MaxV
Alarm := 1;
else
Alarm := 0;
end_if;
if (ActualValue >= M
Event := 1;
elsif (ActualValue <
Event := 0;
end_if;
pAlarm := sigclib_it
pEvent := sigclib_it
//**** Messaging ***
Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete
Select All Ctrl+A
Go To Definition F11
Toggle Breakpoint F4
Toggle Bookmark Ctrl+F2
Mark Text
Clear Marks
Toggle Function Folding
Insert Client-Update Source Code...

5.8 Toggle Function Folding

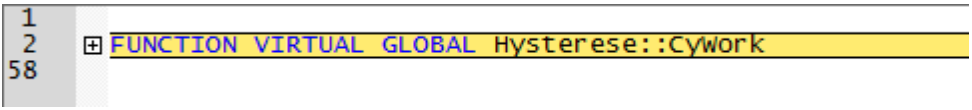
If under Extras => Options => Application Options => Code editor => Folding Options => Fold functions, **true** is entered,



the Item **Toggle Function Folding** in the context menu of the code editor is enabled.

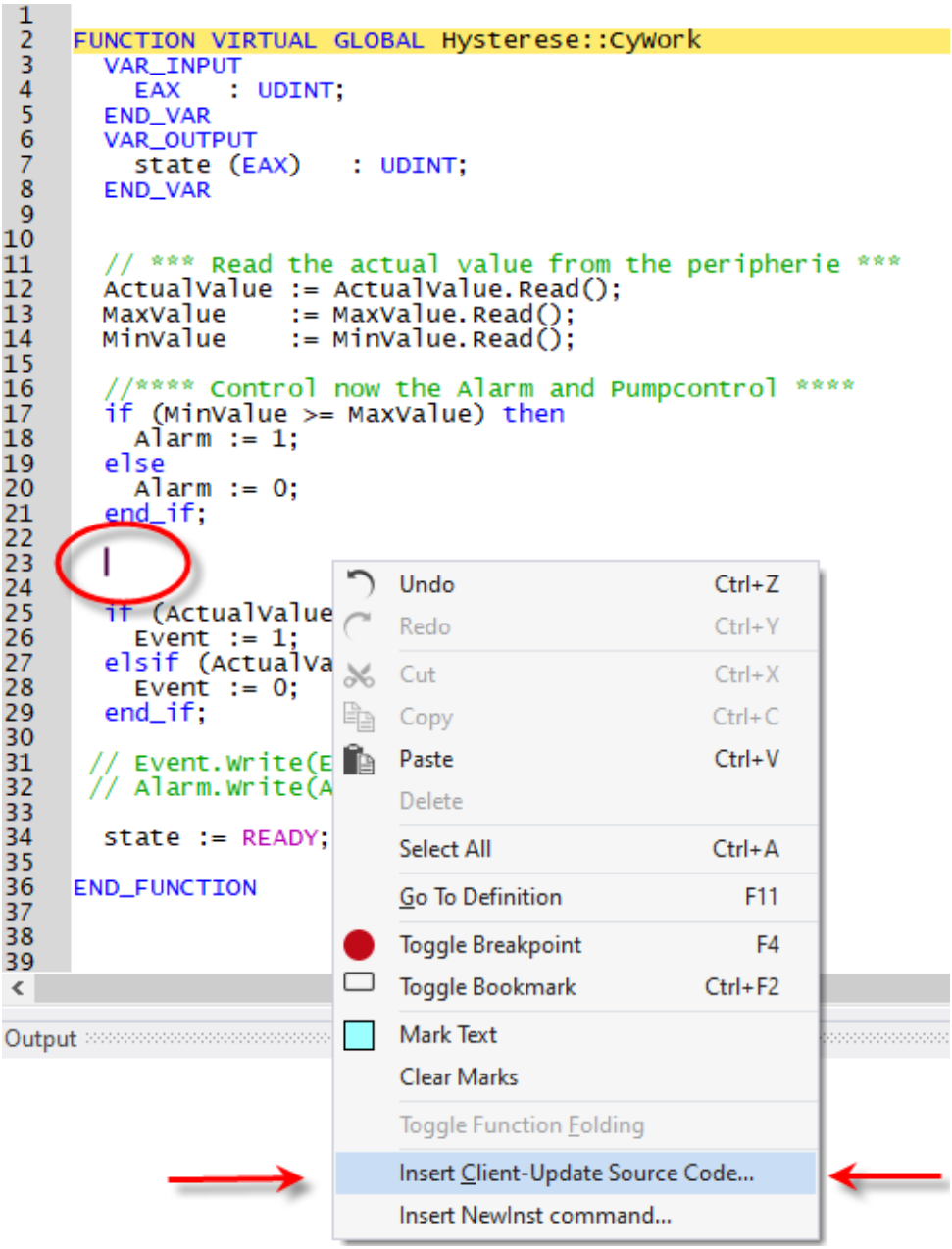


I can be used to enable/disable the folding functions – and that directly in the code editor:

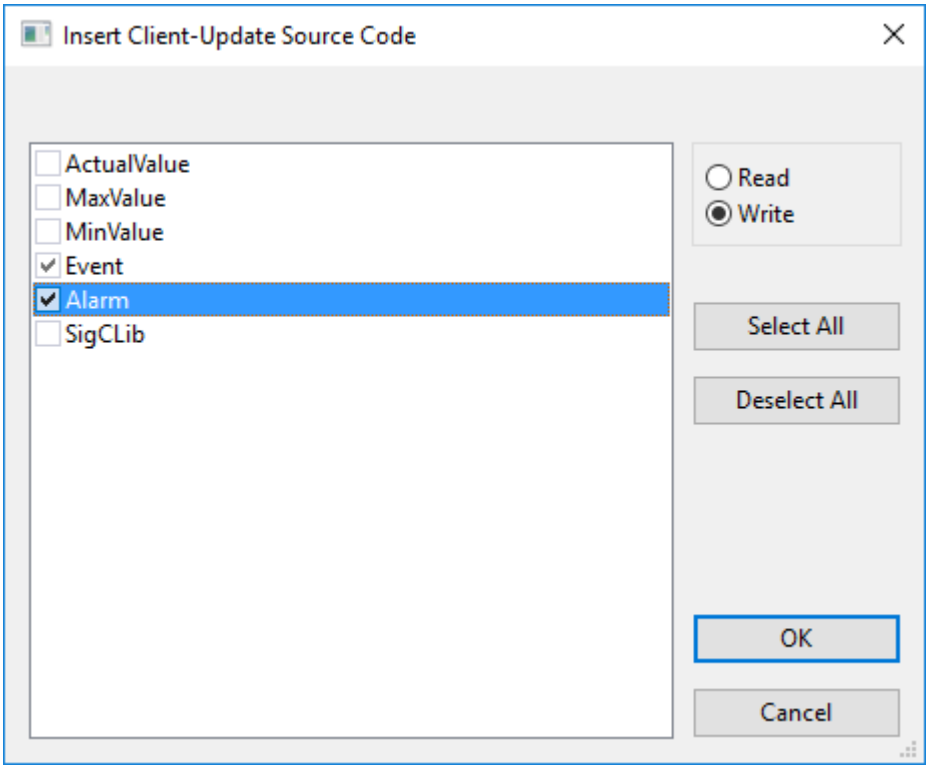


5.9 Insert Client-Update Source Code...

This function allows the automatic insertion of read or write methods. Simply position the cursor in a line where the code should be inserted and open the context menu with a right click.


FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
VAR_INPUT
EAX : UDINT;
END_VAR
VAR_OUTPUT
state (EAX) : UDINT;
END_VAR
// *** Read the actual value from the peripherie ***
ActualValue := ActualValue.Read();
MaxValue := MaxValue.Read();
MinValue := MinValue.Read();
//**** Control now the Alarm and Pumpcontrol ****
if (MinValue >= MaxValue) then
Alarm := 1;
else
Alarm := 0;
end_if;
if (ActualValue
Event := 1;
elsif (ActualVa
Event := 0;
end_if;
// Event.Write(E
// Alarm.Write(A
state := READY;
END_FUNCTION
Output
Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete
Select All Ctrl+A
Go To Definition F11
Toggle Breakpoint F4
Toggle Bookmark Ctrl+F2
Mark Text
Clear Marks
Toggle Function Folding
Insert Client-Update Source Code...
Insert NewInst command...

The following window is then opened.



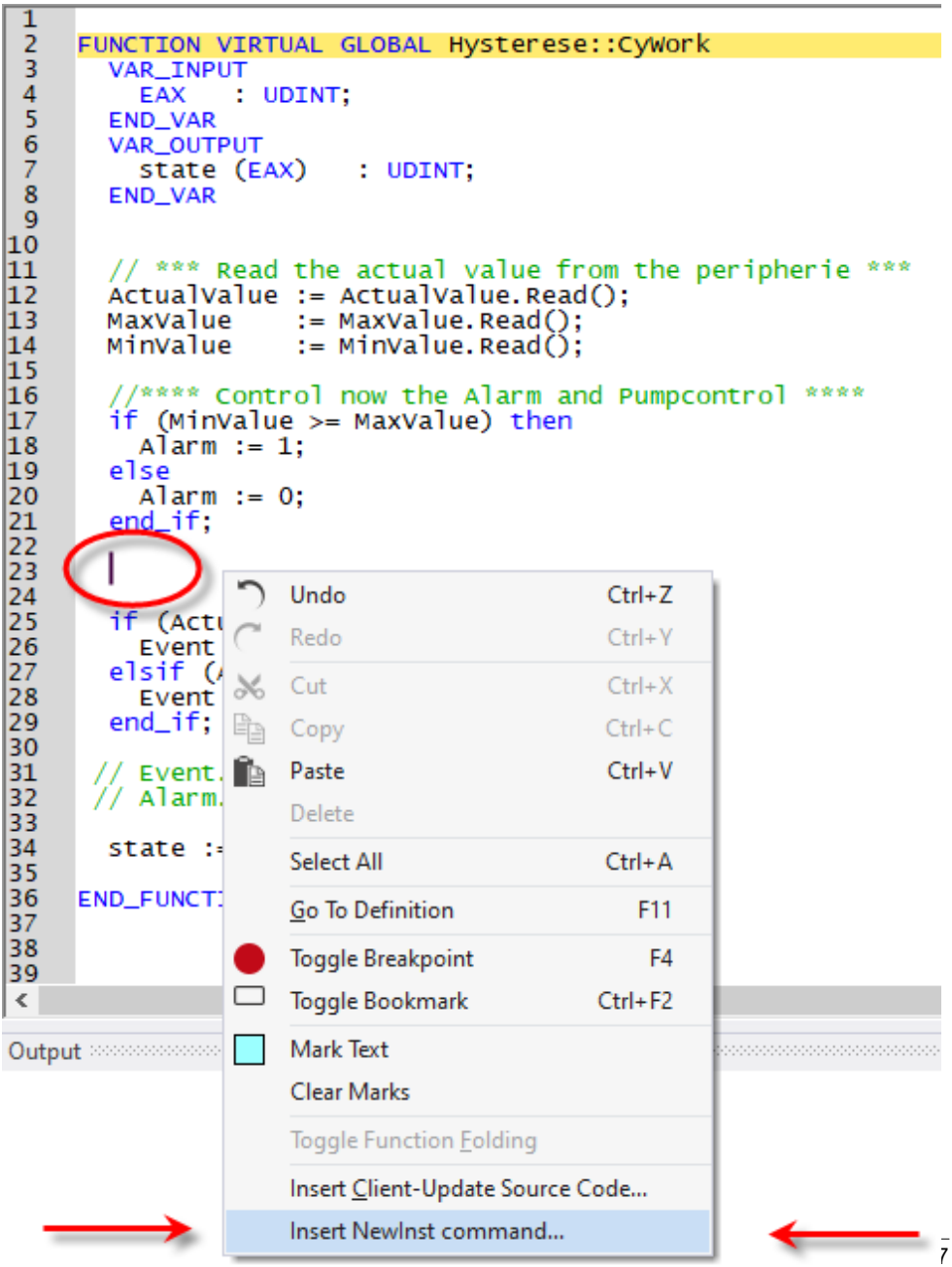
Here, select which client(s) whose read or write methods are to be inserted and click the OK button.

Only the specified program code is inserted where the cursor is located.

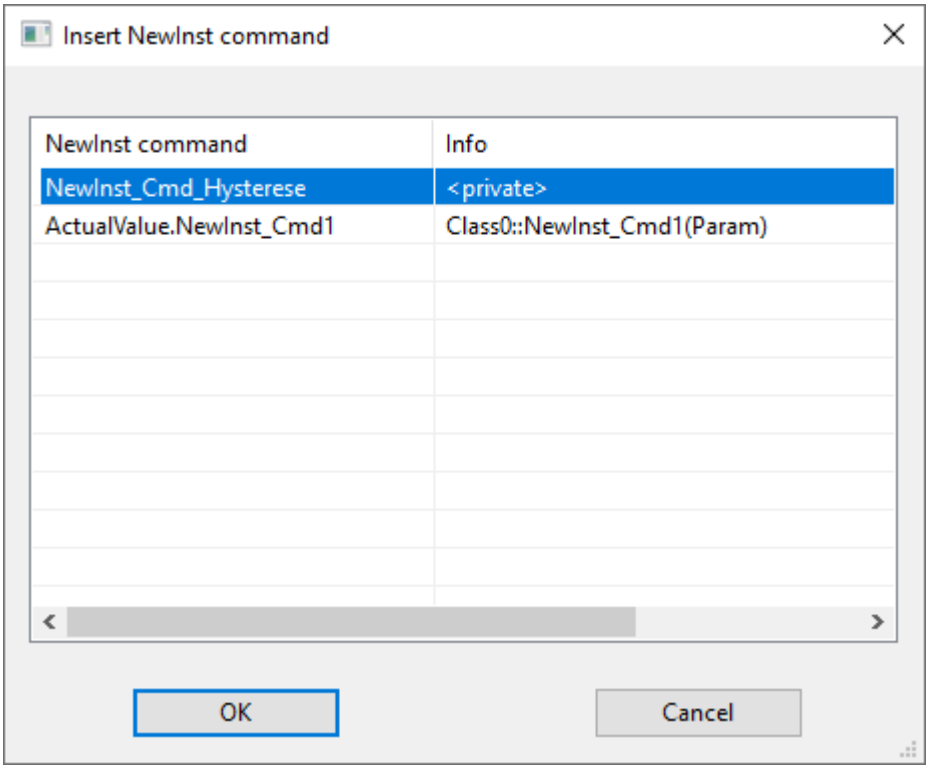
```
16    // *** Read the actual value from the peripherie ***
17    ActualValue := ActualValue.Read();
18    MaxValue    := MaxValue.Read();
19    MinValue    := MinValue.Read();
20
21    if Begin = 0 then
22      TRACE("Start der Messages");
23      Begin := 1;
24    end_if;
25
26    Event.Write (Event);
27    Alarm.Write (Alarm);
28
29    //**** Control now the Alarm and Pumpcontrol ****
30    if (MinValue >= MaxValue) then
31      Alarm := 1;
32    else
33      Alarm := 0;
34    end_if;
35
36    if (ActualValue >= MaxValue) then
37      Event := 1;
38    elsif (ActualValue <= MinValue) then
39      Event := 0;
40    end_if;
41
42    pAlarm := sigclib_itoa(Alarm, #cAlarm[0], 10);
43    pEvent := sigclib_itoa(Event, #cEvent[0], 10);
```

5.10 Insert NewInst command

This comfort function enables the automatic insertion of NewInst commands in the code editor. Simply position the cursor in the line where the code should be inserted and open the context menu with a right click.


FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
VAR_INPUT
EAX : UDINT;
END_VAR
VAR_OUTPUT
state (EAX) : UDINT;
END_VAR
// *** Read the actual value from the peripherie ***
ActualValue := ActualValue.Read();
MaxValue := MaxValue.Read();
MinValue := MinValue.Read();
//**** Control now the Alarm and Pumpcontrol ****
if (MinValue >= MaxValue) then
Alarm := 1;
else
Alarm := 0;
end_if;
Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete
Select All Ctrl+A
Go To Definition F11
Toggle Breakpoint F4
Toggle Bookmark Ctrl+F2
Mark Text
Clear Marks
Toggle Function Folding
Insert Client-Update Source Code...
Insert NewInst command...
Output

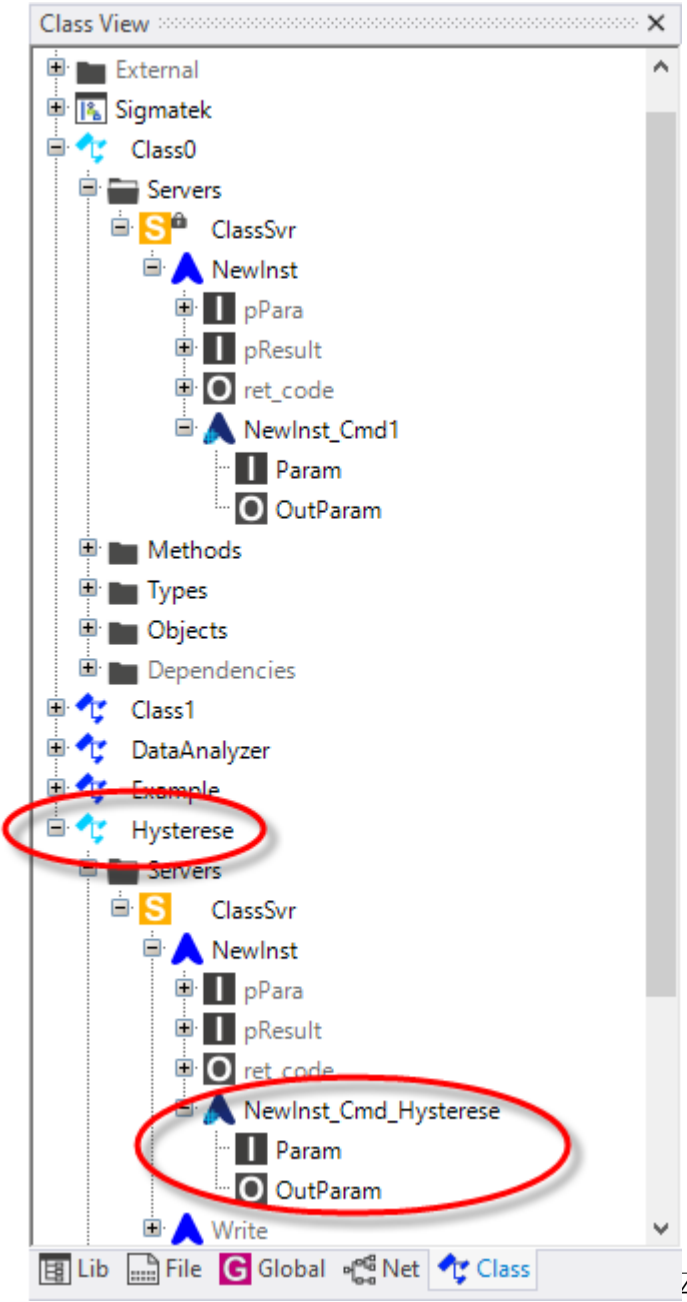
The following window then opens:



It is possible to insert class-specific NewInst commands from this list (the NewInst command marked in the previous screen):

```
1
2   FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
3     VAR_INPUT
4       EAX    : UDINT;
5     END_VAR
6     VAR_OUTPUT
7       state (EAX)    : UDINT;
8     END_VAR
9
10
11    // *** Read the actual value from the peripherie ***
12    ActualValue := ActualValue.Read();
13    MaxValue    := MaxValue.Read();
14    MinValue    := MinValue.Read();
15
16    //**** Control now the Alarm and Pumpcontrol ****
17    if (MinValue >= MaxValue) then
18      Alarm := 1;
19    else
20      Alarm := 0;
21    end_if;
22
23    NewInst_Cmd_Hysterese(Param:=);
24
25    if (ActualValue >= MaxValue) then
26      Event := 1;
27    elsif (ActualValue <= MinValue) then
28      Event := 0;
29    end_if;
30
31   // Event.Write(Event);
32   // Alarm.Write(Alarm);
33
34    state := READY;
35
36  END_FUNCTION
37
38
39
```

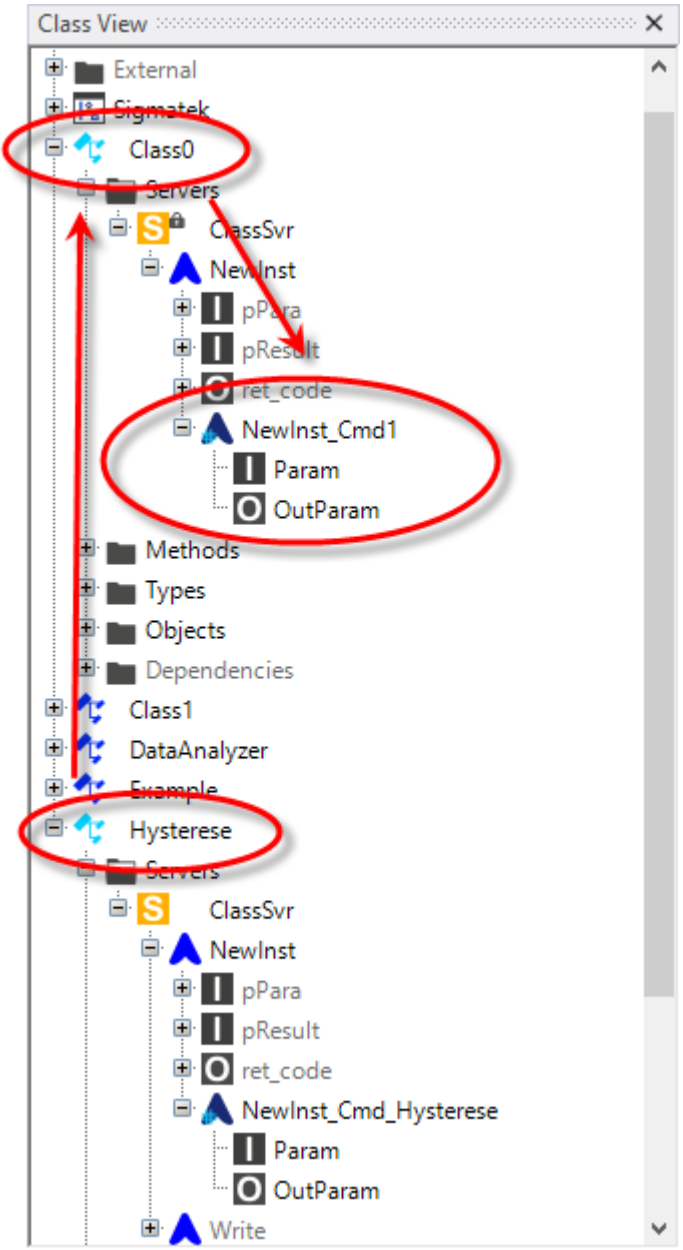
Output


Class View
External
Sigmatek
Class0
Servers
ClassSvr
NewInst
pPara
pResult
ret_code
NewInst_Cmd1
Param
OutParam
Methods
Types
Objects
Dependencies
Class1
DataAnalyzer
Example
Hysterese
Servers
ClassSvr
NewInst
pPara
pResult
ret_code
NewInst_Cmd_Hysterese
Param
OutParam
Write
Lib
File
Global
Net
Class

...or a NewInst command of a connected server (the unmarked NewInst command):

```
1
2   FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
3     VAR_INPUT
4       EAX    : UDINT;
5     END_VAR
6     VAR_OUTPUT
7       state (EAX)    : UDINT;
8     END_VAR
9
10
11    // *** Read the actual value from the peripherie ***
12    ActualValue := ActualValue.Read();
13    MaxValue    := MaxValue.Read();
14    MinValue    := MinValue.Read();
15
16    //**** Control now the Alarm and Pumpcontrol ****
17    if (MinValue >= MaxValue) then
18      Alarm := 1;
19    else
20      Alarm := 0;
21    end_if;
22
23    _call_NewInst_ActualValue_1Class0_2NewInst_Cmd1(Param:=);
24
25    if (ActualValue >= MaxValue) then
26      Event := 1;
27    elsif (ActualValue <= MinValue) then
28      Event := 0;
29    end_if;
30
31   // Event.Write(Event);
32   // Alarm.Write(Alarm);
33
34    state := READY;
35
36  END_FUNCTION
37
38
39
```

Output


Class View
External
Sigmatek
Class0
Servers
ClassSvr
NewInst
pPara
pResult
ret_code
NewInst_Cmd1
Param
OutParam
Methods
Types
Objects
Dependencies
Class1
DataAnalyzer
Example
Hysterese
Servers
ClassSvr
NewInst
pPara
pResult
ret_code
NewInst_Cmd_Hysterese
Param
OutParam
Write

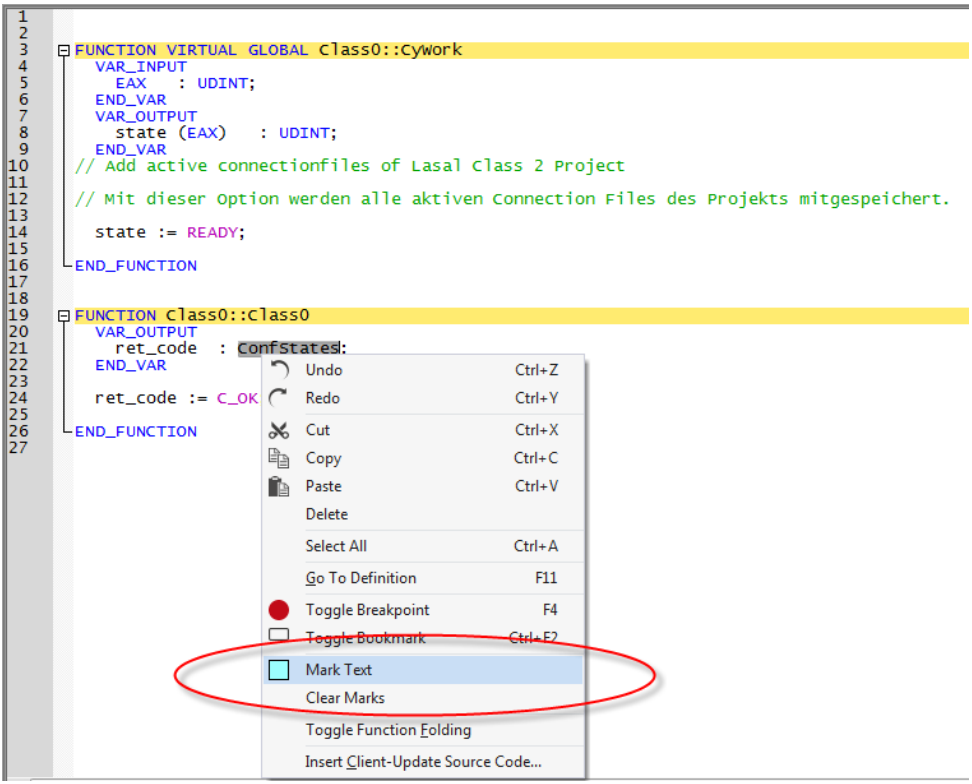
The latter variant is a call to a help function, which was automatically created by clicking on the OK button:

```
44
45  FUNCTION Hysterese::NewInst_Cmd_Hysterese
46    VAR_INPUT
47      Param    : DINT;
48      OutParam  : ^DINT;
49    END_VAR
50    VAR_OUTPUT
51      ret_code  : IprStates;
52    END_VAR
53
54    ret_code := READY;
55  END_FUNCTION
56
57
58  FUNCTION Hysterese::_call_NewInst_ActualValue_1Class0_2NewInst_Cmd1
59    VAR_INPUT
60      Param    : DINT;
61    END_VAR
62    VAR_OUTPUT
63      ret_code  : IprStates;
64    END_VAR
65    VAR
66      Para : CmdStruct;
67      Result : results;
68    END_VAR
69
70    // call NewInst command
71    // ret_code:IprStates = Class0::NewInst_Cmd1(Param)
72
73    // set parameter
74    Para.uiCmd := Class0::NewInst_Cmd1; // Class0::NewInst_Cmd1
75    Para.aPara[0] := Param; // Param
76
77    // command call
78    ret_code := ActualValue.NewInst(#Para, #Result);
79
80  END_FUNCTION
81
```

Output

5.11 Mark Text and Clear Marks Function

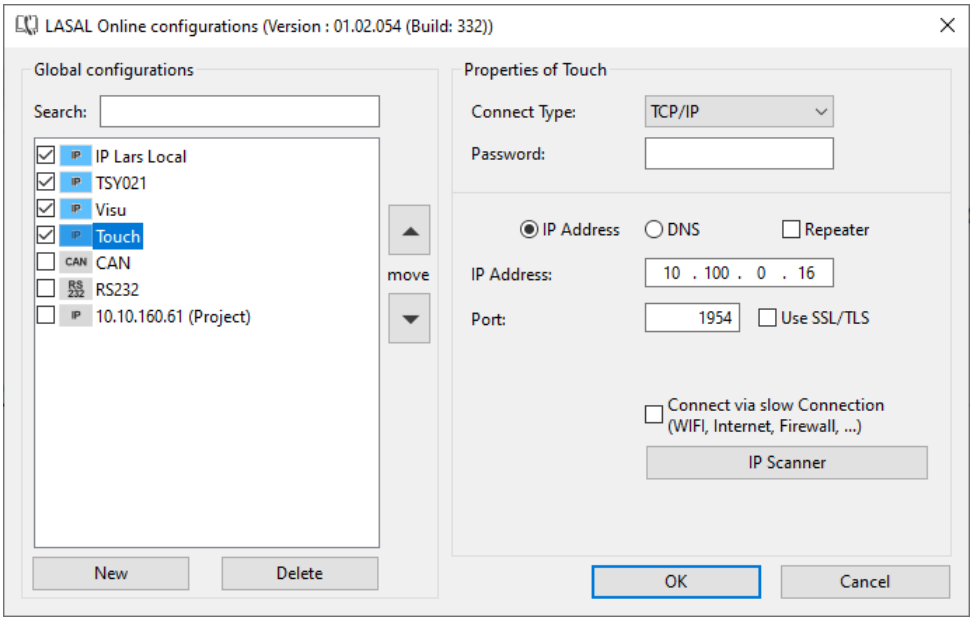
These functions are in the context menu of the Code Editor. This allows on the one hand permanent marking of all appearances of the word / area at the cursor position - this function can also be used several times, all words / areas are marked then -, and on the other hand deleting all set marks.



For more information see the chapter Mark.

5.12 Quick Change of the Online Settings

Independent if a project is loaded or not, with the "Change Online Settings" symbol in the toolbar - next to the Online symbol - the LASAL Online configurations window can be opened:

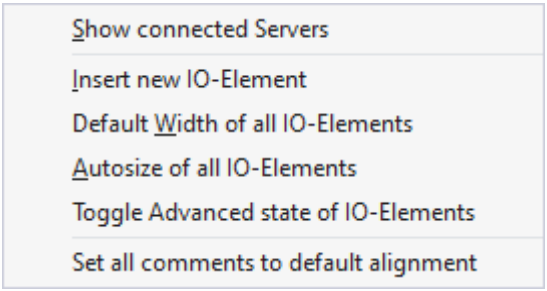


Therefore the online settings can be set, respectively, changed any time. If this action is performed by with a loaded project, the selected online setting is only assumed temporarily. This means the project is not changed.

For more information about the online settings **see the according chapter**.

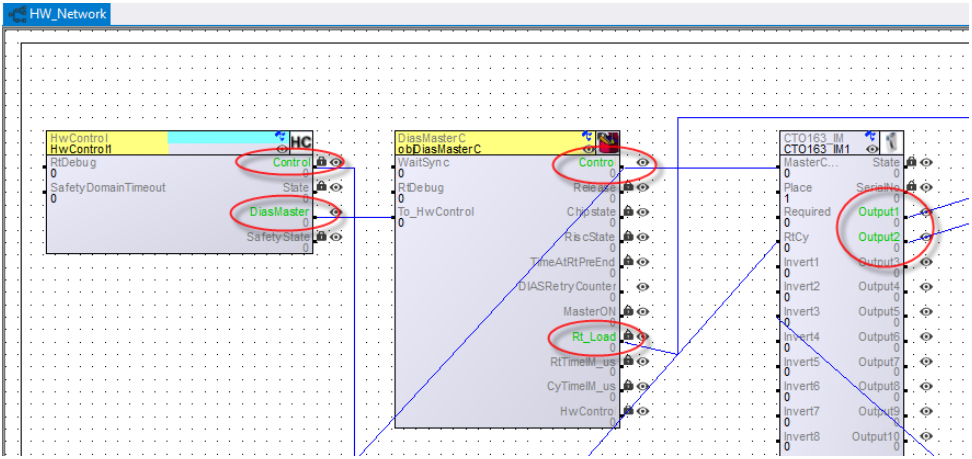
5.13 Context Menu of the Network View: Highlighting Connected Server, Creating a New IO Object

With a right-click on an empty area in the network the **context menu** can be opened.



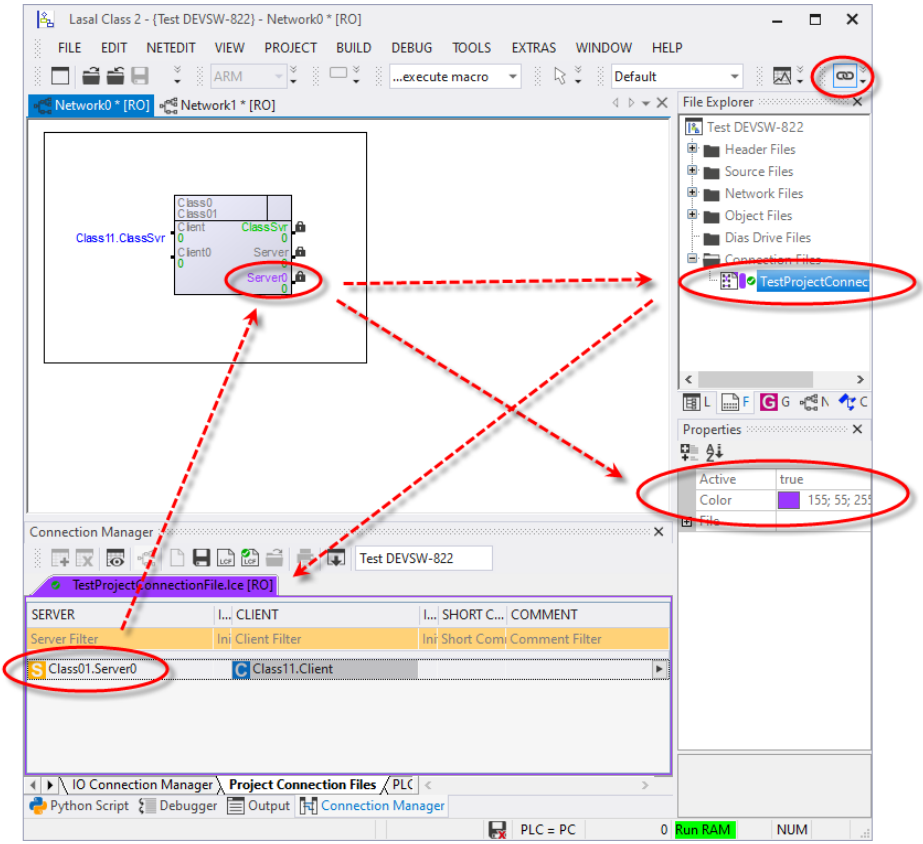
Show Connected Servers

After execution of this command, servers that are connected to clients in other networks are highlighted in terms of color.



In the menu item **Tools/Options**, all connected servers can be permanently displayed when the item **Show Connected Servers** is set to **true**.

If this option is permanently active, connected servers of connections, which are available in the Project Connection File on the controller, are marked in the set color as soon as one is online:



Insert new IO-Element

An IO object is herewith generated directly in the object network. This instruction is equivalent to the following process:

1.) Generate a new connection in ConnectManager,

2.) Assign an IO name and

3.) Drag the created IO object into the network.

For additional information on IO objects, see: the chapter, ConnectionManager

Default Width of all IO-Elements

Herewith all **IO objects** are set to standard width at once.

Autosize of all IO-Elements

Herewith all **IO object** are set to a width at once, where all texts are fully displayed.

Toggle Advanced state of IO-Elements

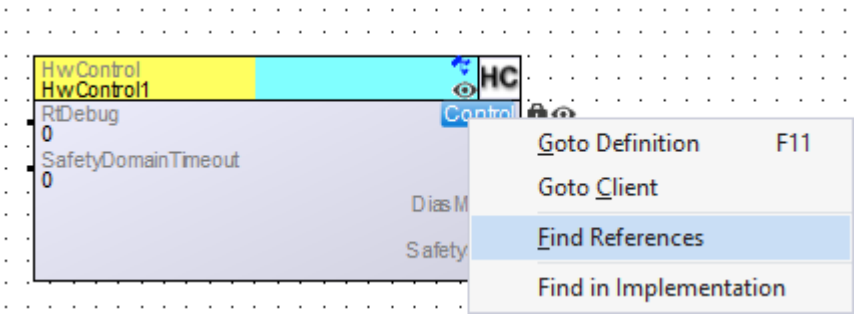
This enables / disables the Advanced checkbox - in the ADVANCED column in the Connection Manager - **for all IO objects at once**. For more detailed information about the Advanced IO Elements, see chapter Advanced IO Functionality.

Set all comments to default alignment

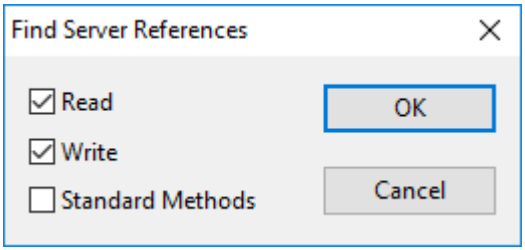
This aligns **all comments** in networks according to the setting under Extras / Options / Network Comment Alignment.

5.14 Run Find References for a Server

For servers in the object network, **Find References** can be run. Here, read/write or global method calls are found in classes connected to the object.

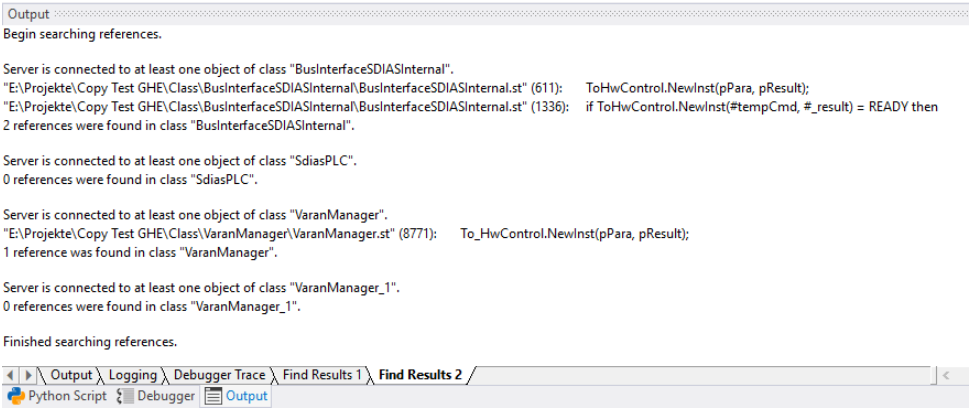


The following dialog is displayed:



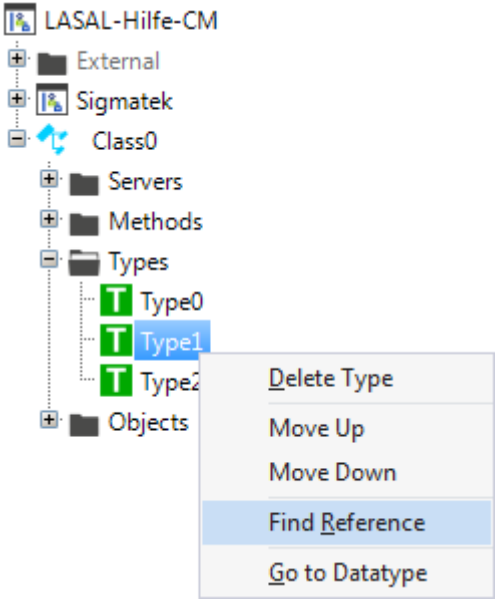
Here the methods to be searched in can be selected.

Output

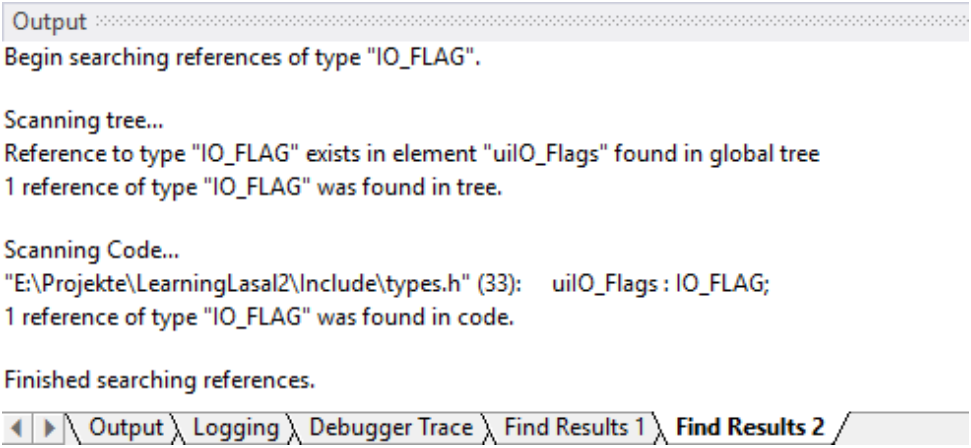

Output
Begin searching references.
Server is connected to at least one object of class "BusInterfaceSDIASInternal".
"E:\Projekte\Copy Test GHE\Class\BusInterfaceSDIASInternal\BusInterfaceSDIASInternal.st" (611): ToHwControl.NewInst(pPara, pResult);
"E:\Projekte\Copy Test GHE\Class\BusInterfaceSDIASInternal\BusInterfaceSDIASInternal.st" (1336): if ToHwControl.NewInst(#tempCmd, #_result) = READY then
2 references were found in class "BusInterfaceSDIASInternal".
Server is connected to at least one object of class "SdiasPLC".
0 references were found in class "SdiasPLC".
Server is connected to at least one object of class "VaranManager".
"E:\Projekte\Copy Test GHE\Class\VaranManager\VaranManager.st" (8771): To_HwControl.NewInst(pPara, pResult);
1 reference was found in class "VaranManager".
Server is connected to at least one object of class "VaranManager_1".
0 references were found in class "VaranManager_1".
Finished searching references.
Output
Logging
Debugger Trace
Find Results 1
Find Results 2
Python Script
Debugger
Output

5.15 Run Find References for a Type

By right clicking on a type, the Find References function can be run. This function locates all applications of the type in the tree and code, and then displays them in the Find Results 2 window.

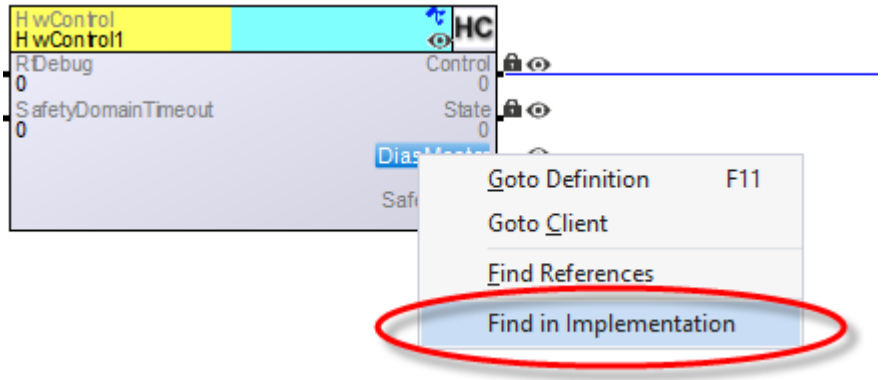


Output


Output
Begin searching references of type "IO_FLAG".
Scanning tree...
Reference to type "IO_FLAG" exists in element "uiIO_Flags" found in global tree
1 reference of type "IO_FLAG" was found in tree.
Scanning Code...
"E:\Projekte\LearningLasal2\Include\types.h" (33): uiIO_Flags : IO_FLAG;
1 reference of type "IO_FLAG" was found in code.
Finished searching references.
Output
Logging
Debugger Trace
Find Results 1
Find Results 2

5.16 Finding Clients / Server in the Implementation

In the object network, the context menu item “Find in Implementation” can be used to find the locations of a client or server name (in the example below, it is the server name “Control”) in the code editor. Classes connected via Object Channel are also searched.



These are **marked with color** in the code editor. Here, **only the string** is searched. All occurrences of the client / server **name** are therefore located.

Hardware Editor | HW_Network | NewNetwork | HC HwControl [RO] | CAM123_IM [RO] | HwBaseCDIAS [RO]

```
603
604  FUNCTION VIRTUAL GLOBAL HwControl::DiasMaster::Write
605    VAR_INPUT
606      input (EAX)    : DINT;
607    END_VAR
608    VAR_OUTPUT
609      result (EAX)   : DINT;
610    END_VAR
611
612    //is it a required error from dias master?
613    if input = 0 then
614      State := _RequiredError;
615      return;
616    end_if;
617
618    //is class ok
619    if ( State = _ClassOk ) then
620
621      s_DiasMaster.p_Functions$DINT := input;    //load pointer to DiasMaster functions
622
623      //convert time into ns (to be compatible)
624      s_DiasMaster.p_Functions^.ud_DiasTime := s_DiasMaster.p_Functions^.ud_DiasTime * 1000;
625
626      //look if DiasMaster time is bigger or equal as our task time
627      if ( ( s_DiasMaster.p_Functions^.ud_DiasTime ) < GetMyTaskTime() ) then
628
629        State           := _DiasTimeError;       //Set to error state
630        s_DiasMaster.p_Functions  := NIL;
631        result            := -1;
632
633        TRACE( "HwControl: DiasMaster time is smaller than HwControl RT-Task!" );
634
635        return;
636
637      //look if DiasMaster time is a multiple time of our task time
638      elsif ( ( s_DiasMaster.p_Functions^.ud_DiasTime ) mod GetMyTaskTime() <> 0 ) then
639
640        State           := _DiasTimeError;       //Set to error state
641        s_DiasMaster.p_Functions  := NIL;
642        result            := -1;
643
644        TRACE( "HwControl: DiasMaster time is not a multiple of HwControl RT-Task!" );
645
646        return;
647
648      else
649
650        //set counter to time to execute task at first realtime
651        s_DiasMaster.ud_Counter := s_DiasMaster.p_Functions^.ud_DiasTime;
652
653        result          := 0;
654
655        DiasMaster := input := 1;        //Mark server as connected right
656
657      end_if;
```

All results are summarized in the “Find Results1” pane, whereby here, all base or derived classes are also searched and more occurrences are found.

Output

```
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (715):    //DiasMaster
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (976): //Write Function of DiasMaster
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (979): //              DiasMaster              :W        :server which is connected to DiasMaster
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (980): //[members]s_DiasMaster      :W        :pointer to DiasMaster functions and settings
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (981):  //VIRTUAL GLOBAL HwControl::DiasMaster::Write
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (999):  // if a diasmaster is available => tell him
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1042): FUNCTION VIRTUAL GLOBAL HwControl::DiasMaster::Write
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1059):    s_DiasMaster.p_Functions$DINT:= input;       //load pointer to DiasMaster functions
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1064):    //look if DiasMaster time is bigger or equal as our task time
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1071):      TRACE( "HwControl: DiasMaster time is smaller than HwControl RT-Task!" );
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1075):    //look if DiasMaster time is a multiple time of our task time
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1082):      TRACE( "HwControl: DiasMaster time is not a multiple of HwControl RT-Task!" );
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1093):      DiasMaster := input := 1;                 //Mark server as connected right
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1097):    //set RtDebug-Flag for DiasMaster
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1260):  // we need to inform either the VaranManager (if available) or the DiasMaster (if available) about it, to call the CyWork of it (because HwControl got no cyclic task)
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1278):  // check diasmaster
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1317):  // we need to inform either the VaranManager (if available) or the DiasMaster (if available) about it, to not call the CyWork of it
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1331):  // check diasmaster
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1570):    //look if DiasMaster is also used ( must be after starting realtime task )
"D:\Projekte\Tests\LearningLasal1\Class\HwControl\HwControl.st" (1573):      //make final initialization in DiasMaster

Total found: 33; Matched Files: 1; Total files searched: 4
```

5.17 Navigation in the Object Network

Especially with several resp. very big/complex networks, it is helpful to jump quickly between the end points of a connection with one click.

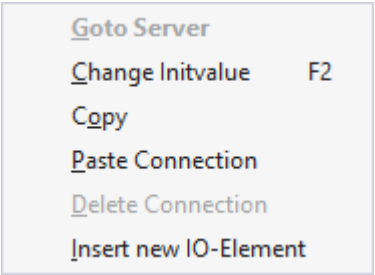
There are several ways to do this:

Context Menu of a Client/Server in the Object Network

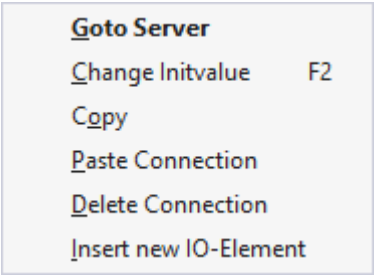
Navigation with Cursor Keys

5.17.1 Context Menu of a Client/Server in the Object Network

With a right click on the value of a client, its context menu is opened:



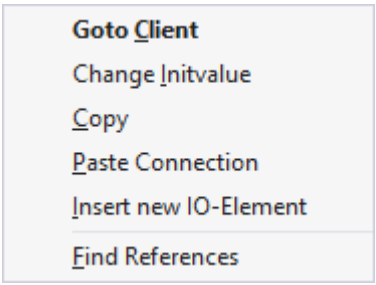
If the client is already connected to a server, all menu entries are active:



With a click on **Goto Server** the ON display jumps to the connected server and selects it.

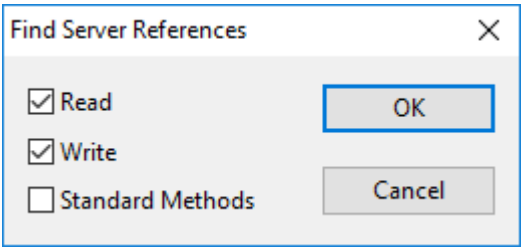
Change Initvalue changes the init value of the client, **Copy** copies the end point of a possible connection, **Paste Connection** inserts an end point already copied on the server side and thus establishes a connection. **Delete Connection** deletes an existing connection and **Insert New IO Element** creates a new IO element directly next to the client.

On the server side, the context menu looks like this:



Again, the user only has to select **Goto Client** and the ON display jumps to the selected client.

Change Initvalue also changes the init value here (the menu item is only available if in the server properties **Initialize** is set to **true**!), **Copy** and **Paste Connection** as well as **Insert New IO Element** work in the same way as the client context menu. **Find References** opens the Find References dialog and searches the entire project for the selected server methods..



The results are displayed in the **Find Results** pane:

Output

Begin searching references.

Server is connected to at least one object of class "AI046".
0 references were found in class "AI046".

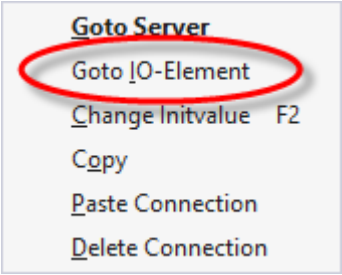
Server is connected to at least one object of class "DM046".
0 references were found in class "DM046".

Server is connected to at least one object of class "SdiasBase".
0 references were found in class "SdiasBase".

Finished searching references.

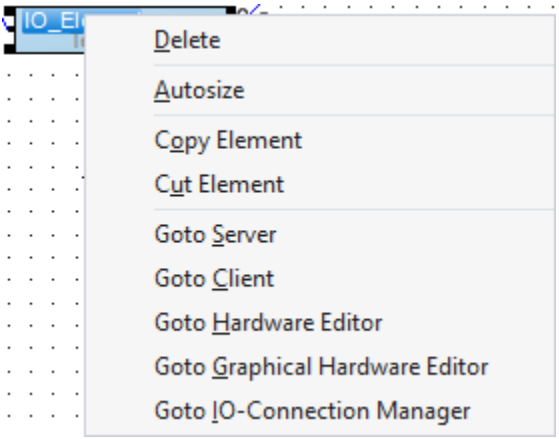
Output | Logging | Debugger Trace | Find Results 1 | **Find Results 2**

If a connection already exists through an IO element, there is the additional menu item **Goto IO Element**:



Selecting this menu entry jumps to the connected IO element.

Also the **IO element** has its own **context menu**:

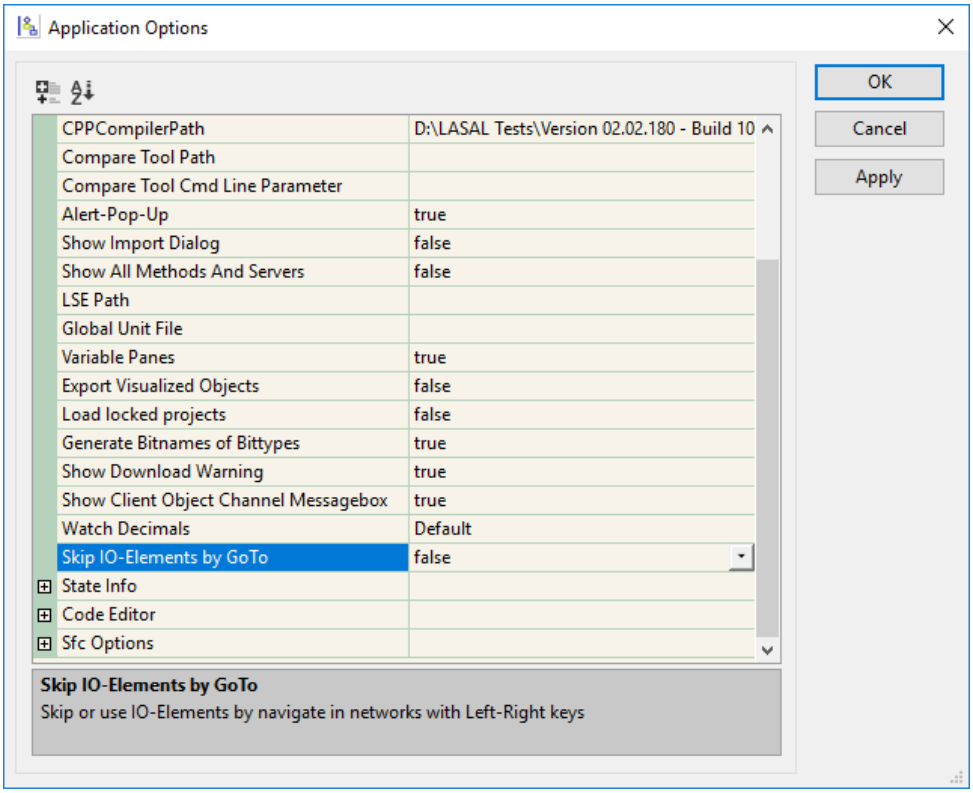


In addition to menu items already known from other network objects, such as **Delete**, **Autosize** (horizontal size of the IO element is set so that both server and client are completely readable without abbreviation), **Copy** and **Cut Element** (copy resp. cut the IO element) you can jump directly from here with the **Goto commands** to the client or server as well as the **Hardware Editor**, the **Graphical Hardware Editor** or the **Connection Manager**. (For the HW editors, however, the IO element must be attached to a server of an object from the HW library!).

5.17.2 Navigation with Cursor Keys

If a **client/server** is **already selected**, it is possible to navigate in the object network using the cursor keys. Use the **up arrow** and **down arrow keys** to navigate within the object (next or previous server / client or to the object name and class).

The **arrow-to-left** and **arrow-to-right** keys can be used to leave the object - if there is a connection to a server/client or to an IO element at the just marked position. The **Skip IO-Elements by GoTo** option in **Extras/Application Options** determines whether you switch to the IO element when navigating with the horizontal arrow keys or skip it and jump directly to the connection's end point.



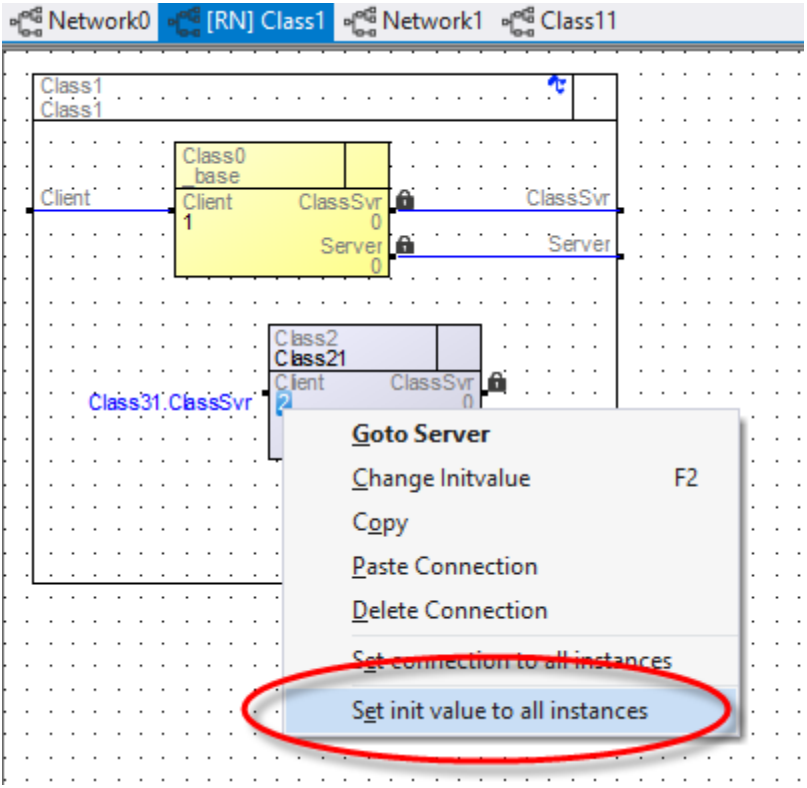
The default value here is **false** (the **IO element** is the first to be **accessed**), with **true** the **IO element** is **skipped** during cursor navigation.

5.18 Execute Set init value to all instances resp. Set connection to all instances for an embedded Client

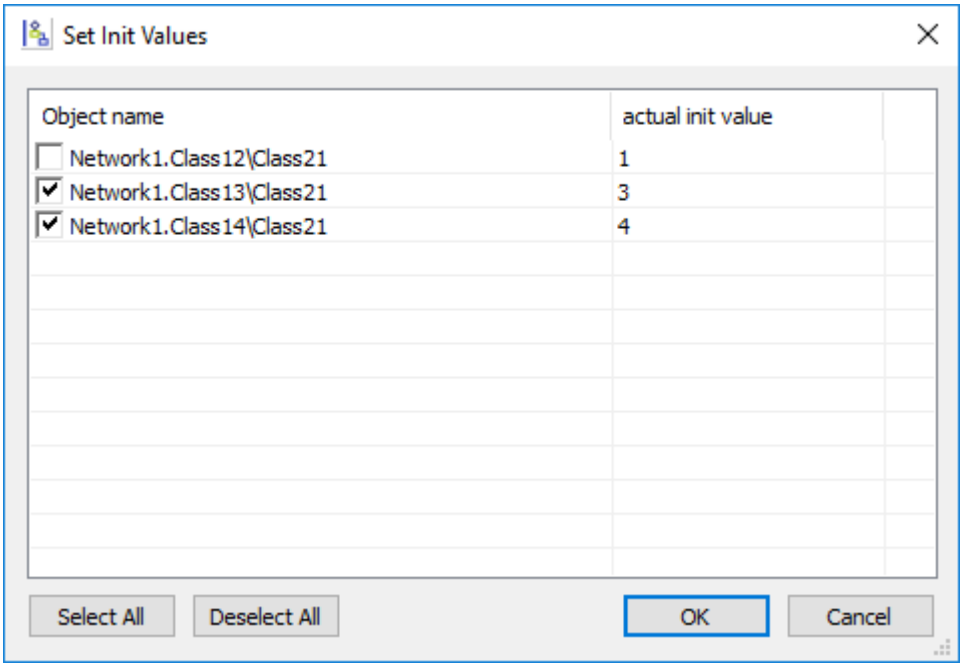
Set init value to all instances resp. **Set connection to all instances** can be executed in the object network for clients of embedded objects. This can be necessary e.g. when importing a base class from a library. From the reference network as well as from the proxy network it is possible to **adapt the initial value of the client resp. the global server connection** for all objects of the class.

Note that the context menu items **only** exist if the client actually **has an init value** (in the properties window the value field must be filled! The value can also be 0) or if it has a **global connection**.

If **Set init value to all instances** is executed,

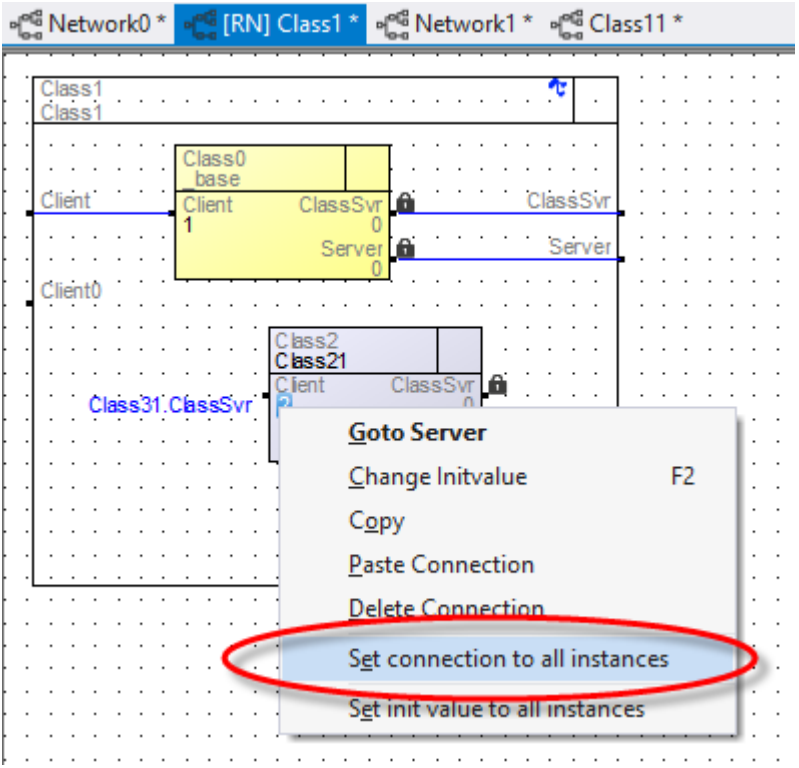


… and there is more than one object of the class and the init values of the different objects are not the same, then the dialog **Set Init Values** opens:

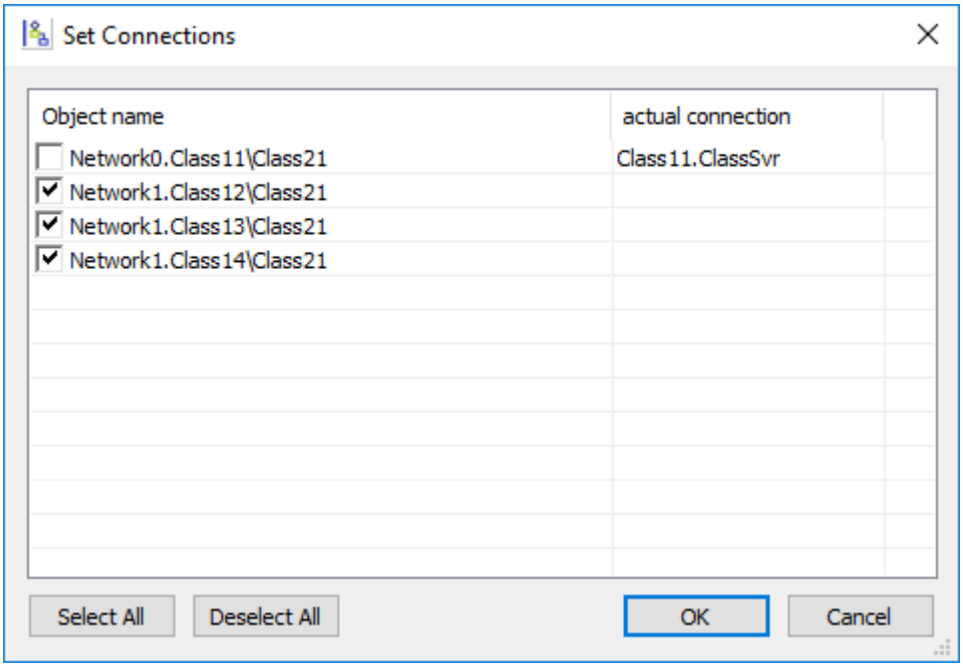


All objects of the selected class and their current init values are listed and can be selected or deselected. For those clients that are selected, the init value is set to the value of the originally selected client after a click on the OK button.

If the client originally selected has a global connection, it can also be set for all other instances. To do this, use the menu item **Set connection to all instances**.



If there are differences in the connections of the embedded clients of the respective objects, the **Set Connections** dialog is displayed:



Here you again have the option of selecting or deselecting individual instances. The global connection is reset for all selected instances.

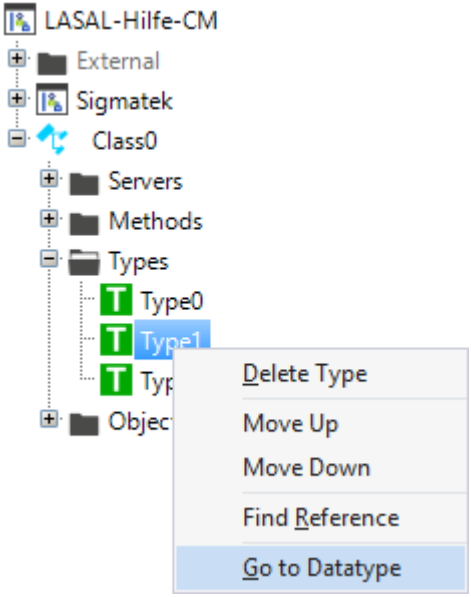
These functions can be called **from both the reference and proxy network**.

If all initial values or all global connections are already identical, an INFO line is displayed in the output window instead of the dialog:

```
OutputCommand: [INFO] Nothing to do. All connections or init values are identical.
```

5.19 Go to Datatype with Elements that can have a Type

For elements that can have a type (type, var, client, server, subtype, subvar, parameter…) can be selected with a right click on **Go to Datatype**. As long as the object is not a standard type (e.g. DINT), the program jumps directly to the type in the object tree.



5.20 Loading of projects with missing elements

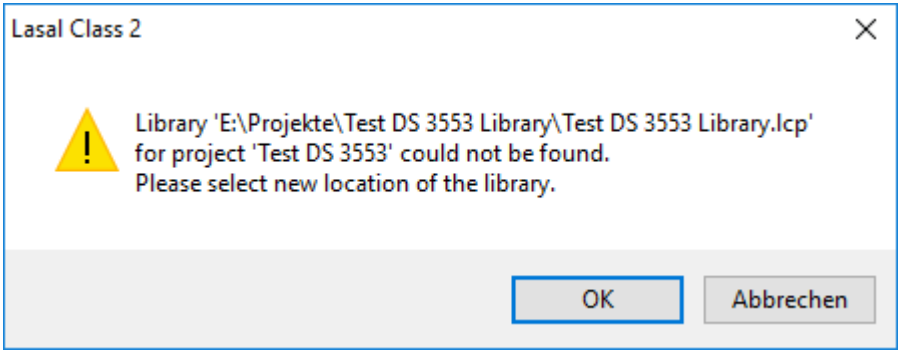
The following error messages can arise among other things at opening of projects:

- Library could not be found
- Class could not be found
- Header file could not be found

If one of the above error messages are returned, new paths, replacement classes or headers must be entered, or loading of the project stopped by pressing cancel in the "question dialog" for libraries that were not found.

5.20.1 Library could not be found

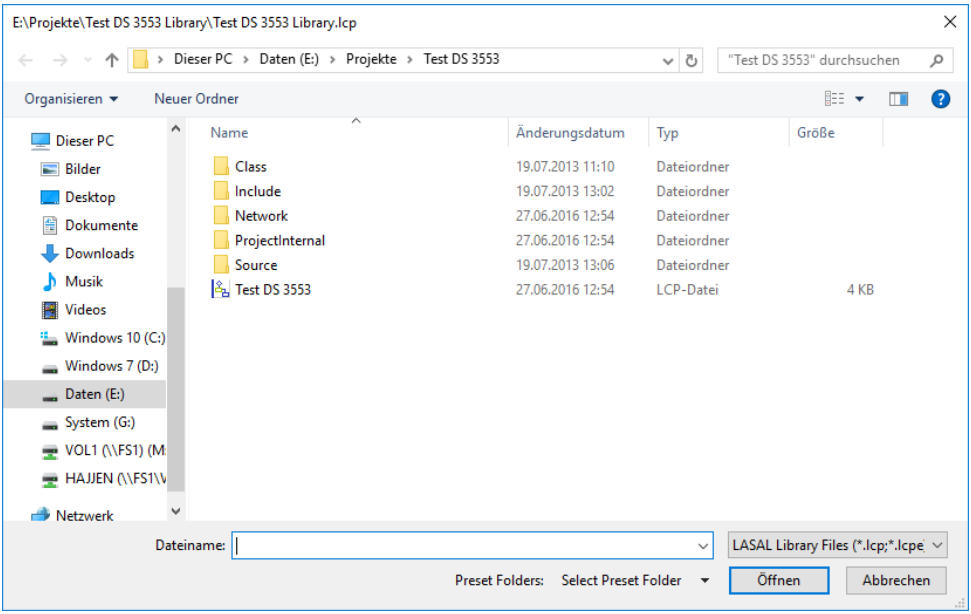
If the following window appears while opening a project, either a different path must be entered for the library or loading the project must be stopped.



This message can appear if a library has been deleted or renamed, respectively, the path of the library has changed.

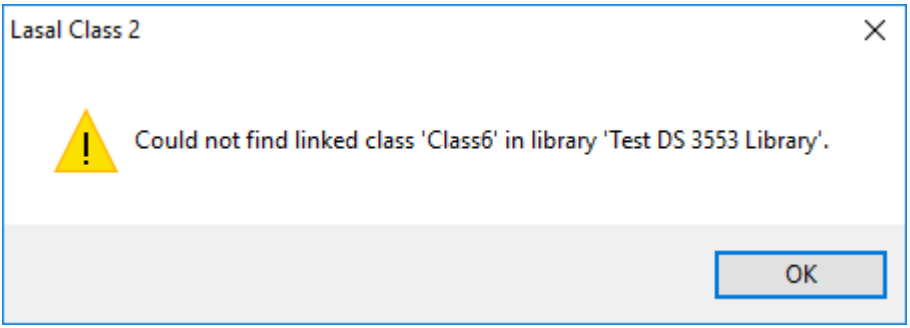
Confirm this message with OK and select the new library path from the open window, or abort loading the project.

At the heading of this window the originally defined path of the library is displayed.

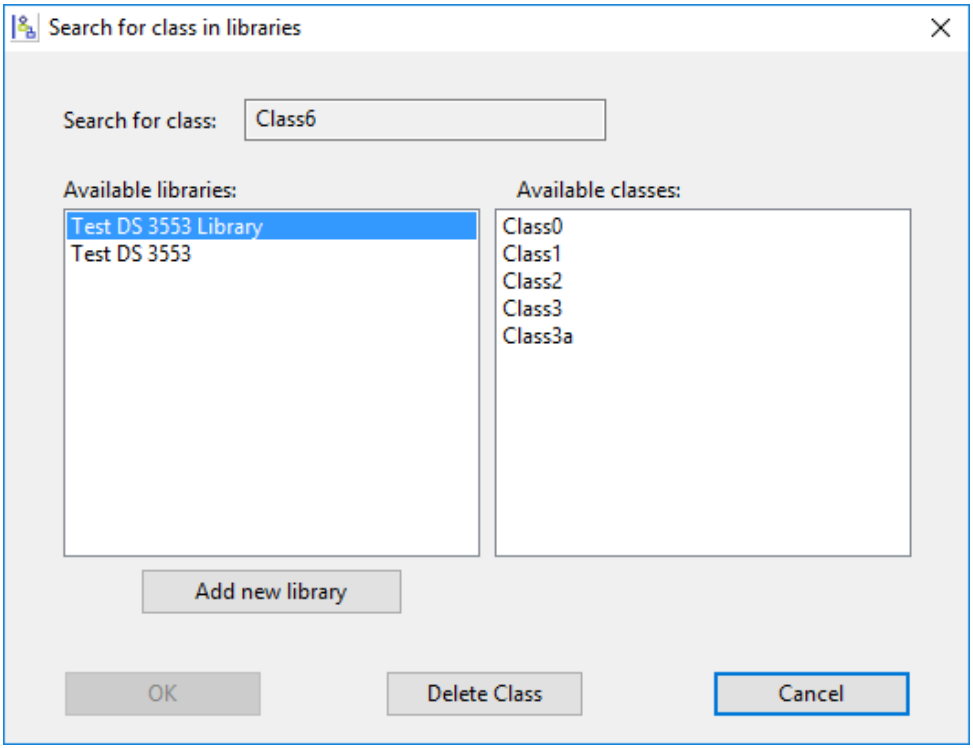


5.20.2 Class in the library could not be found

If the following window appears when opening a project, the mentioned class has been deleted or renamed.



After confirmation with **OK**, the **Search for class in libraries window** appears.

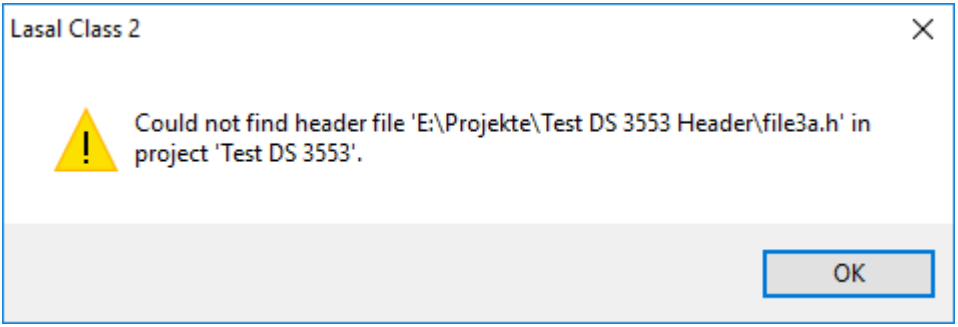


Selection possibilities

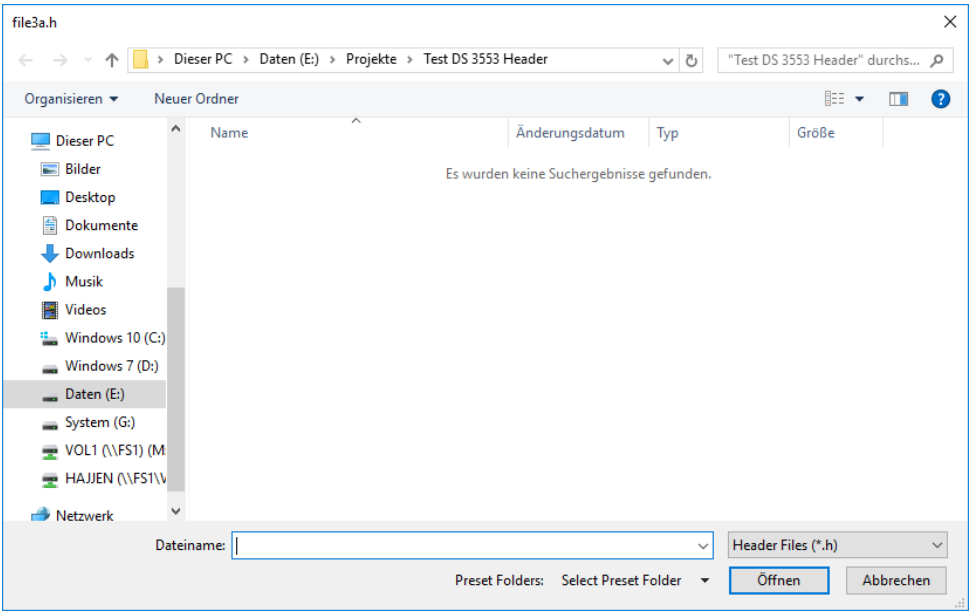
- Select a new class under **Available classes:** and confirm with **OK**.
- Delete a class (**Delete Class**) out of the library under **Available classes:**.
- With **Cancel** the class to find is deleted.

5.20.3 Header file could not be found

A header file has been deleted or renamed if the following message appears.



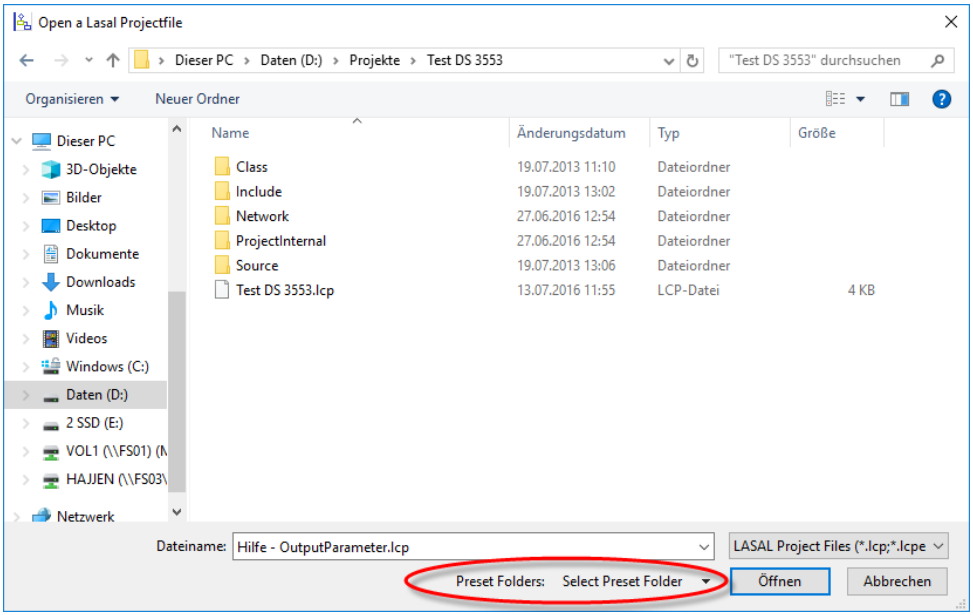
After confirmation with **OK** the following window appears:



Afterwards create a new header file or select the right one.

5.21 Open Dialogs with Quick Navigation

In the Open Dialogs (e.g. Open Project, Open File), a selection list is located at the lower frame of the dialog box in which the defined path can be selected directly.



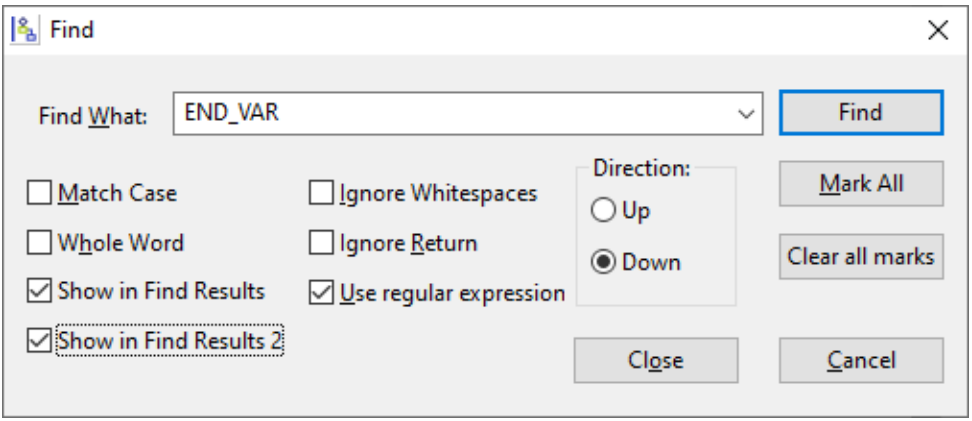
A detailed description of the Preset folders function can be found in the corresponding chapter.

5.22 Search and Replace

In LASAL CLASS 2 different kinds of searching and replacing are available.

5.22.1 Simple Search

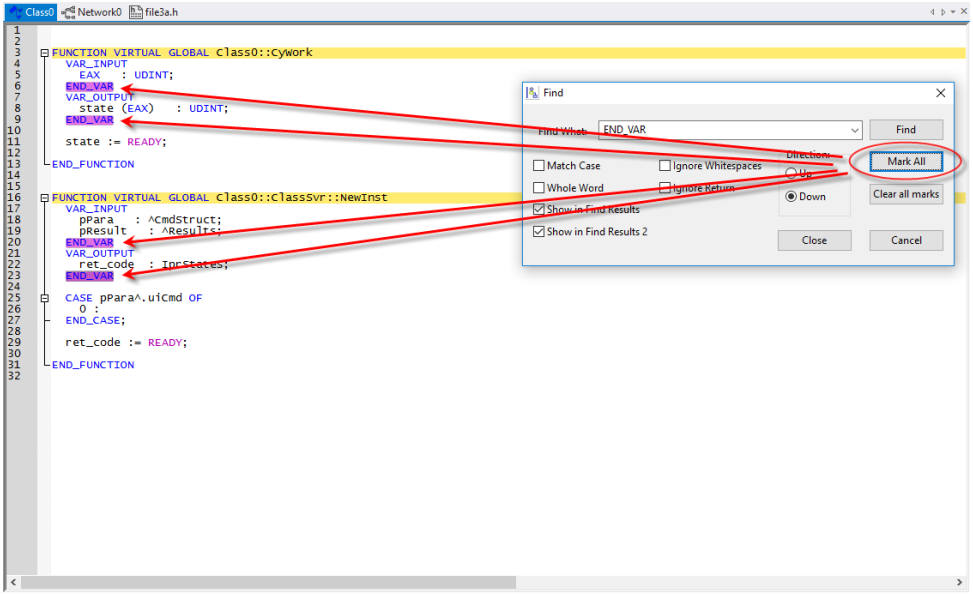
With the search function **Simple search (Find)** in LASAL CLASS 2, entire text or individual words can be located. Where exactly to search depends on which section in marked at the time of the call. The simple search is started by first selecting the Code Editor. To open the search window, use the combination **Ctrl+ F** or select **Edit, Find...** from the menu bar.



Type in the word (or text) in the **Find What:-field** and click on **Find Next** with the left mouse button or press the **enter** key. If an element / text was previously marked / selected in any one of the windows (class tree, network, code editor…), the marked text is inserted into the **Find What:** window.

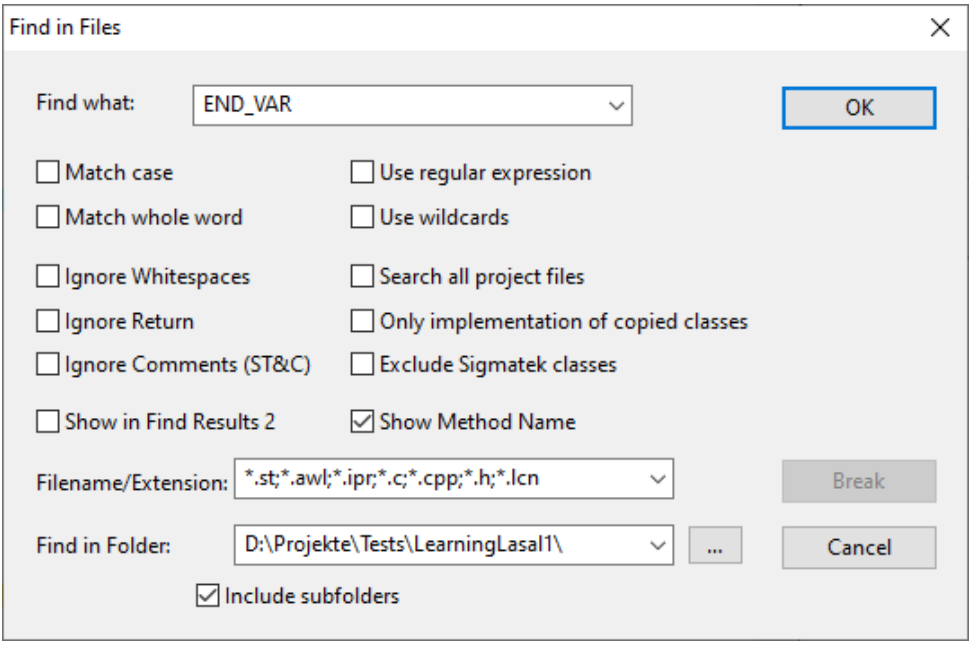
If only a **whole word** should be located activate the corresponding checkbox. Define the search direction and decide whether the search should include **upper and lower** case letters by activating/ deactivating the checkbox **Match Case**. If **Ignore Return** is selected, carriage return is ignored (you also find text spread over several lines). If you select **Ignore Whitespaces** blanks, tabs and similar special characters are ignored. The checkboxes **Show in in Find Results** resp. **Show in Find Results 2** define, where the output goes to.

With a click on "**Mark All**", **all located occurrences are permanently marked** until this setting is removed by clicking on "**Clear all marks**".


Class0
Network0
file3a.h
FUNCTION VIRTUAL GLOBAL Class0::CyWork
VAR_INPUT
EAX : UDINT;
END_VAR
VAR_OUTPUT
state (EAX) : UDINT;
END_VAR
state := READY;
END_FUNCTION
FUNCTION VIRTUAL GLOBAL Class0::ClassSvr::NewInst
VAR_INPUT
pPara : ^CmdStruct;
pResult : ^Results;
END_VAR
VAR_OUTPUT
ret_code : IprStates;
END_VAR
CASE pPara^.uiCmd OF
0 :
END_CASE;
ret_code := READY;
END_FUNCTION
Find
Find What:
END_VAR
Find
Mark All
Clear all marks
Match Case
Ignore Whitespaces
Whole Word
Ignore Return
Direction:
Up
Down
Show in Find Results
Show in Find Results 2
Close
Cancel

5.22.2 Find in Files

It is also possible, in LASAL CLASS 2 to search for words or text in other files. To start this function, select **Edit, Find in Files...** from the menu bar or use the combination **Ctrl + Alt + F**.



Enter the word or phrase in the field, **Find what:** and click **OK** with the left mouse button or simply push the **enter** key. If an element / text was previously marked / selected in any one of the windows (class tree, network, code editor…), the marked text is inserted into the **Find What:** window. However, the following decisions must first be made:

Match case

Activate this function, when the search should include **upper and lower case letters**.

Match whole word

Activate this function when only the whole word should be located.

Ignore Whitespaces

If this function is activated, coherent texts separated by spaces and special characters (such as tabs etc.) are also found.

Ignore Return

This enables texts to be found even across line boundaries.

Ignore Comments (ST & C)

This only finds texts that do not appear in ST or C comments.

Show in Find Results 2

The output is in the Show Find Results 2 window.

Use regular expression

This placeholder (Wildcards) can be assigned in the search text.

A regular expression can be a single character or a multiple character scheme. The simplest regular expression is a single character that equals itself in character string. Every letter (a to z or A To Z) or number (0 to 9) can be used as a character string.

Certain key characters can have a special meaning when used as a regular expression:

Character	Meaning
Asterisk-Symbol * (star)	Equals 0 or several sequential character schemes.
Brackets [ ]	Specifies an area of individual character strings.
Caret ^	Marks the beginning of an input string.
Dollar sigh $	Marks the end of an input string.
Hyphen	Separated the end points of an area.
Parentheses ()	Specifies a group of characters as the name of an alliance.
Period .	Corresponds to each individual character, including spaces.
Plus +	Signifies one or more sequential schemes.

Question mark ?	Null or the start of a scheme.
Under score _	Corresponds to a comma (,), left or right brace ({ }), left or right parentheses, the start of an input string, the end of an input string or a space.

Using wildcards

Place holders '*' (stars) can be used as place holders at activation. These place holders stand for an arbitrary number of characters within a line.

Search All Project Files

By activating this option, all files assigned in the project is searched. The type of file can also be selected under File types (e.g. search all header files linked in the project.). The Find in Folder field is locked.

Only Implementation of copied classes

By activating this option, the search is limited to all class files. The declaration section of the class file is then ignored and searched in the implementation section only (the part in which the user programs their methods).

Under **File types:**, enter the appropriate file ending in the field or press the  button to define the data type. After this, only the files with the defined ending are searched.

Specify the path in which each file is located with the defined file ending. Use the  symbol to select the path or simply type it in the **Find in Folder:** field. Once a path has been entered in can be reselected using the  symbol.

The defined path and file with the specified ending are then searched. Select **Include Subfolders** to include the paths of sub folders in the search.

At start up, the dialog box is closed and the search results are entered sequentially in the **Find Results window**. The search is interrupted when the dialog is re-opened and the **Break** button is pushed.

If there are any changed but not yet saved source files in the started LASAL 2 project, a request for saving appears at start of Find in Files.

Exclude Sigmatek Classes

If this checkbox is selected, SIGMATEK classes are not included in the search.

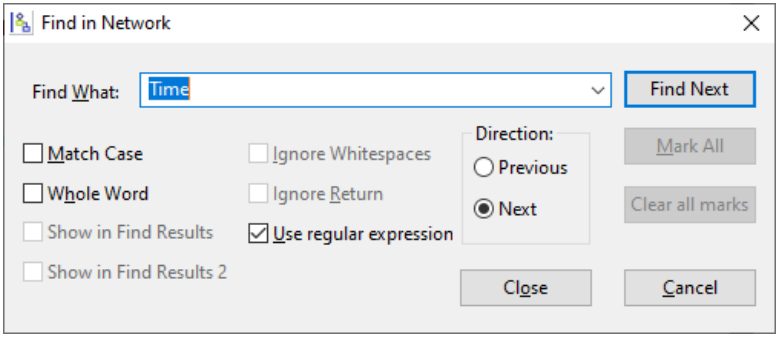
Show Method Name

If this checkbox is selected, the name of the method in which the text was found is also displayed.

5.22.3 Find in Networks

Lasal Class 2 offers the possibility to search for texts - server or client names, object names etc. - in **object networks** . To open the search window, use the keyboard shortcut **Ctrl + F** or select **Edit, Find...** from the menu bar.

The search dialog is similar to the simple search, with the difference that some checkboxes and radio buttons are grayed out and you can select Previous / Next instead of Up / Down.



Type the word (or text) you want to search for in the **Find What:** field and left-click **Find Next** or press **Enter**.

If you only want to search for an **entire word** , activate the **Whole Word**checkbox. Next, define the **search direction** under **Direction** : and with the checkbox **Match Case**, decide whether **upper and lower case** letters should be included in the search.

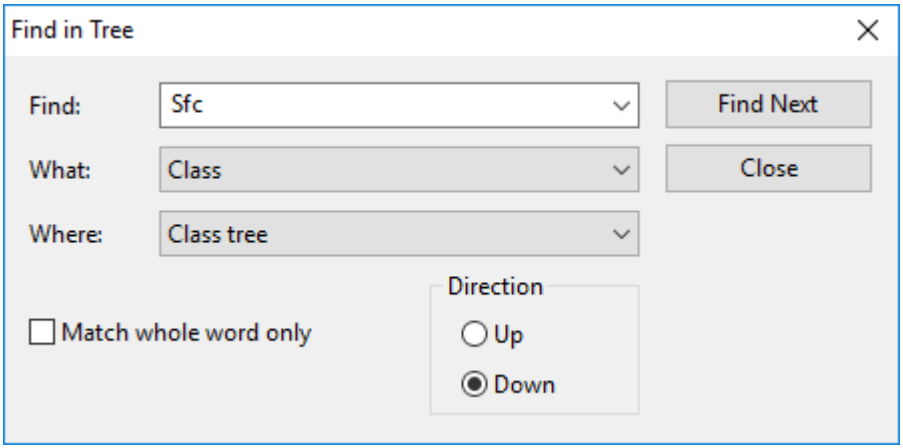
By activating **Use Regular Expression** , wildcards can be specified in the search text used. A regular expression can be a single character or a multi-character scheme. The simplest regular expression is a single line that equates to itself in the character string. Any letter (A to Z or a to z) or number (0 to 9) can be used as a character pattern Certain keyboard characters have a special meaning when used as a regular expression:

Characters	Definition
Asterisk symbol * (star)	Corresponds to 0 or several sequences of the character scheme.
Brackets [ ]	Specifies a section of a single character scheme.
Caret ^	Indicates the beginning of an input string.
Dollar sign $	Indicates the end of an input string.
Hyphen –	Separates the endpoints of a segment.
Parentheses ()	Specifies a group of characters as the name of an affiliation.
Period .	Corresponds to each individual character, including spaces.
Plus sign +	One or more sequences of a scheme
Question mark ?	Equals null or an occurrence of the scheme.
Underscore _	Corresponds to a comma (,), left/right brace (}), left/right parenthesis, the start/end of an input string or space.

As a result, the marker in the object network window jumps to the found text (client name, object name, server name...). **However** , texts e.g. in **IO objects**, **connection names** drawn to the outside and **texts outside** the currently displayed network are not found.

5.22.4 Search in Trees

With the "Find In Tree" dialog it is possible to search for tree elements. Through the find **Edit\Find in Tree** menus or with the shortcut “**Ctrl + Shift + T**”; or simply with a right-click on the tree and selecting “Find in Tree” in the context menu, the following dialog is opened.



Find

The search term is entered here.

What

Type of the element that should be searched (Class, Server, Client, Variables, Network, Method, Object, Type, File, Type element and so on, depending on the tree).

Where

Tree, in which the search should be conducted (Class Tree, Network Tree, Libraries Tree, Globals Tree, File Tree or Hardware Editor Tree).

Match whole word only

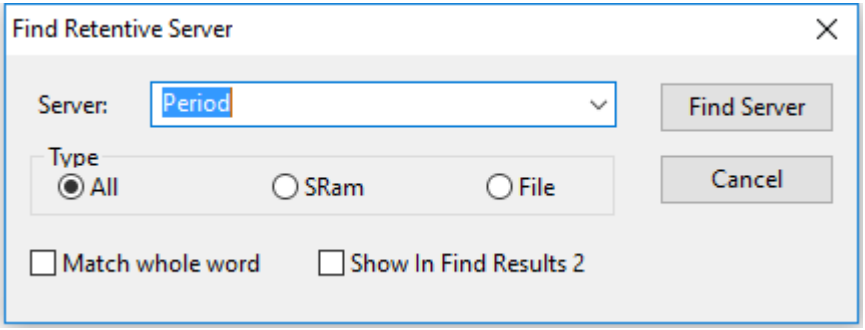
Only whole words are searched.

Direction (Up / Down)

Specifies the direction.

5.22.5 Search for Retentive Server

With the **“Find Rentive Server”** dialog, it is possible to **search for zero-voltage servers** explicitly. Through the **“Edit\Find in Retentive Server”** menu or with the shortcut **“Ctrl + Shift + E”**; or simply with a right-click on the tree and **selecting “Find Retentive Server”** in the context menu, the following dialog is opened.



Server

Here, the **search term** must be entered.

Type

Here you can determine whether **all** zero-voltage-safe servers, only those **stored in the SRAM** or only those **stored in a file** in the root directory of the control are to be found.

Match whole word

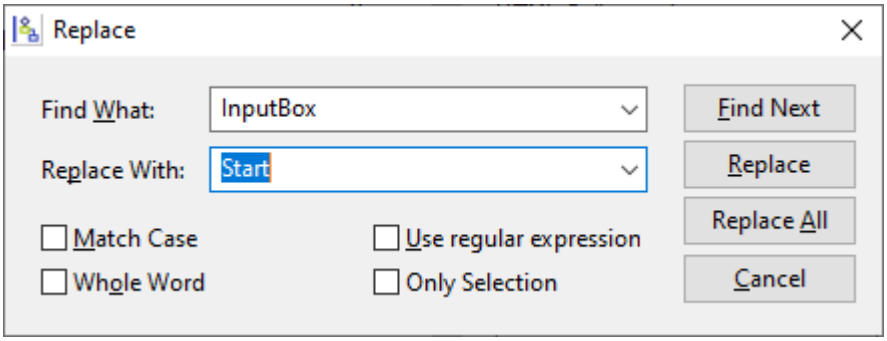
Only whole words are searched.

Show In Find Results 2

Search results are displayed in the **Find Results 1** window by default. Here, the alternative **Find Results 2** can be selected (because, for example, the results are already in **Find Results 1** and are still needed).

5.22.6 Simple Replacement

With the replace function in LASAL CLASS 2, individual words or entire text can be located and replaced. Where exactly to search and replace depends on which section in marked at the time of the call. The simple replacement is started by first selecting the Code Editor. To open the replace window, use the combination, **Ctrl+ R** or select **Edit, Replace...** in the context menu.



Enter the word or text to be located in the **Find What:**-field and in the **Replace With:**-field, enter the new word or text. Next, click on **Find Next** with the left mouse button or press the **enter** key. When the word is found, it is marked in blue. Press the **Replace** button to replace the desired word (text).

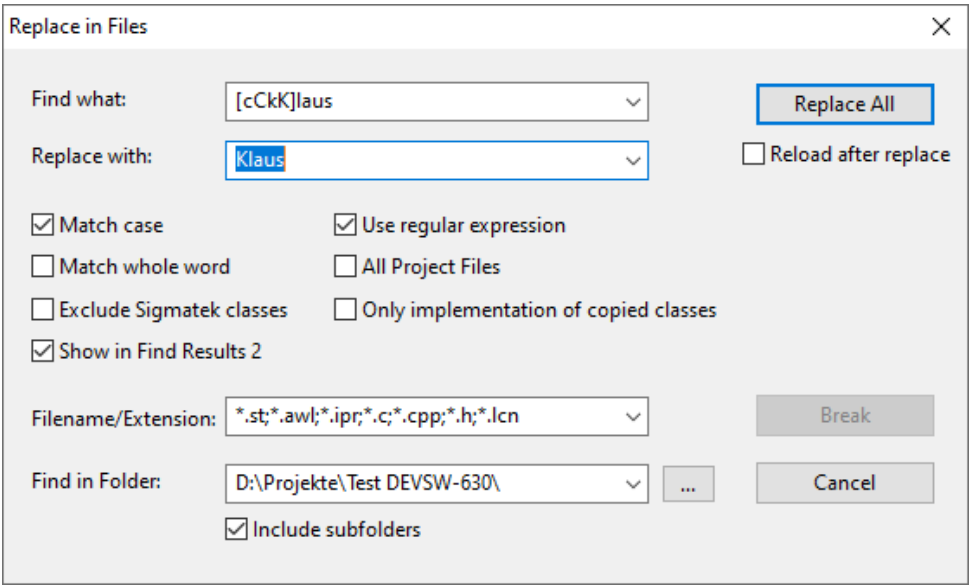
If only a **whole word** should be located activate the corresponding checkbox. Define the search direction and decide whether the search should include **upper and lower** case letters by activating/ deactivating the checkbox **Match Case**.

Press **Replace all** to replace all the words or text found in the file.

If multi-line text is selected when the dialog is opened, the **Only Selection** checkbox is set. If this option is active, the search and replace process is executed within the selection.

5.22.7 Replace in Files

It is also possible, in LASAL CLASS 2 to search for words or text in other files and replace them. To start this function, select **Edit, Replace in Files...** from the menu bar or use the combination **Ctrl + Shift + R**.



Enter the word or text in the field, **Find what:** and in the field, **Replace with:**, enter the new word or text then push the **Replace All** button or simply push the **enter** key. However, the following decisions must first be made:

Match case

Activate this function, when the search should include upper and lower case letters.

Match whole word

Activate this function when only the whole word should be located.

Exclude Sigmatek Classes

If this checkbox is selected, SIGMATEK classes are not included in the search.

Show in Find Results 2

Activate this function to follow the changes in the “Find Results 2” window.

Ignore Whitespaces

If this function is activated, related texts separated by spaces and special characters (such as tabs etc.) are found.

Ignore Return

With this function, text beyond the line limits is also found.

Show in Find Results 2

The output is shown in the Show Find Results 2 window.

Use regular expression

This placeholder (Wildcards) can be assigned in the search text.

A regular expression can be a single character or a multiple character scheme. The simplest regular expression is a single character that equals itself in character string. Every letter (a to z or A To Z) or number (0 to 9) can be used as a character string.

Certain key characters can have a special meaning when used as a regular expression:

Character	Meaning
Asterisk-Symbol * (star)	Equals 0 or several sequential character schemes.
Brackets [ ]	Specifies an area of individual character strings.
Caret ^	Marks the beginning of an input string.
Dollar sigh $	Marks the end of an input string.
Hyphen	Separated the end points of an area.
Parentheses ()	Specifies a group of characters as the name of an alliance.
Period .	Corresponds to each individual character, including spaces.
Plus +	Signifies one or more sequential schemes.
Question mark ?	Null or the start of a scheme.

Under score _	Corresponds to a comma (,), left or right brace ({ }), left or right parentheses, the start of an input string, the end of an input string or a space.

All Project Files

By activating this option, all files in the project are searched for and all matching texts are replaced - except for *.st and *.lcn files. The type of files can also be selected under **File types** (e.g. search all header files linked in a project). The **Find in Folder** field is locked.

Only Implementation of copied classes

By activating this option the search is limited to all class files. The declaration section of the class file is then ignored and only the implementation section is searched. (each section that contains user defined methods).

Under **File types:**, enter the appropriate file ending in the field or press the  button to define the data type. After this, only the files with the defined ending are searched.

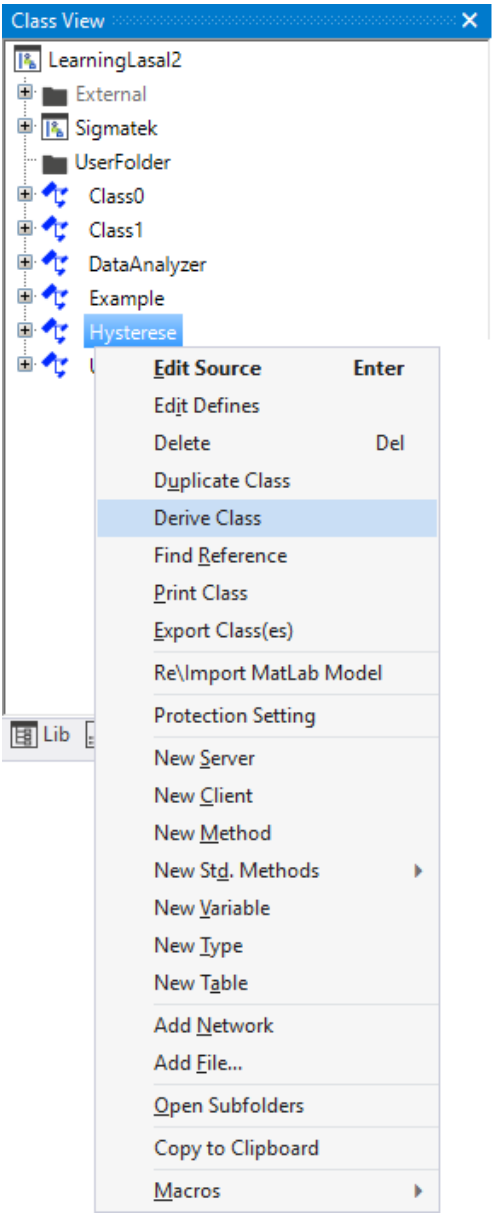
Specify the path in which each file is located with the defined file ending. Use the  symbol to select the path or simply type it in the **Find in Folder:** field. Once a path has been entered in can be reselected using the  symbol.

The defined path and file with the specified ending are then searched. Select **Include Subfolders** to include the paths of sub folders in the search.

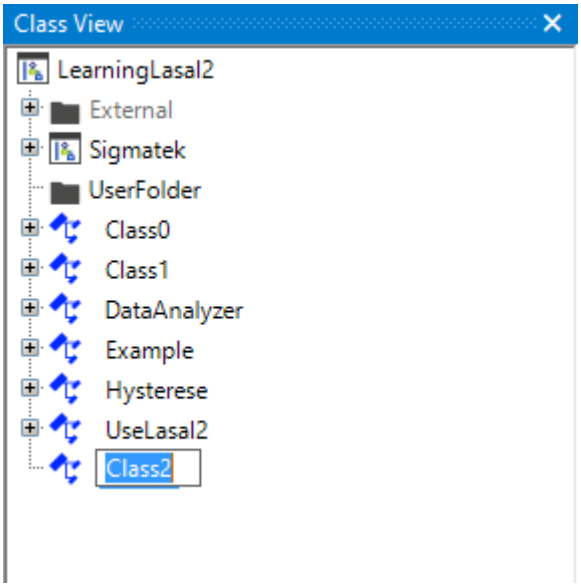
At start up, the dialog closes and the search results are entered in the **Find Results** window sequentially. The search can be aborted by re-opening the dialog box and pressing the **Break** button.

5.23 Easy Derivation of Classes

This function makes it possible to create a derivation of a class with an easy mouse click. To create a derivation of a class, simply open the context menu of a class and select **Derive Class**.


Class View
LearningLasal2
External
Sigmatek
UserFolder
Class0
Class1
DataAnalyzer
Example
Hysterese
Lib
Edit Source Enter
Edit Defines
Delete Del
Duplicate Class
Derive Class
Find Reference
Print Class
Export Class(es)
Re\Import MatLab Model
Protection Setting
New Server
New Client
New Method
New Std. Methods
New Variable
New Type
New Table
Add Network
Add File...
Open Subfolders
Copy to Clipboard
Macros

Thus you get a derivation of the **Hysterese** class (here: **Class0**).



5.24 Drag & Drop of Class Elements

5.24.1 Change of Order of the Class Elements

Via Drag & Drop the order of the following elements within a class can be changed:

- Server
- Clients
- Methods
- Variables
- Tables

At changing the order of methods pay attention to the fact that in the sub folder "Global" the virtual global methods are listed before the global methods. A global method cannot be moved in the area of virtual global methods and reverse.

5.24.2 Moving / Copying of Class Elements

Via Drag & Drop the following class elements can be moved, respectively, copied to another class:

- Server (Server methods are not copied, respectively, moved)
- Clients
- Variables
- Types

Dragging of elements to another class leads to the process **Moving**, dragging of elements with pressed Ctrl key leads to the process **Copying**.

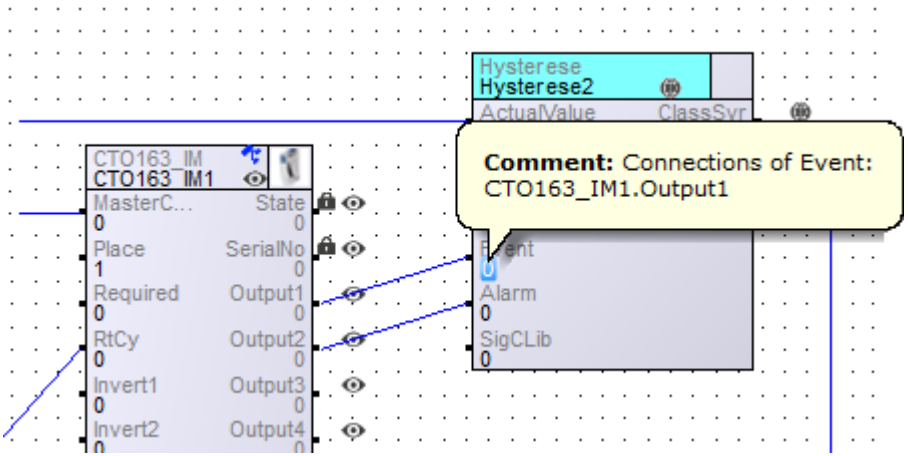
The element either can be dragged over the class name or over the corresponding element folder (e.g. class folder, clients at moving a client).

Additionally to the moving and copying possibilities mentioned above, types from classes can be moved, respectively, copied to global types and reverse.

If a type used in the original class is pulled to another class (= moved), the action is not performed. Instead, a corresponding error message is shown in the output window.

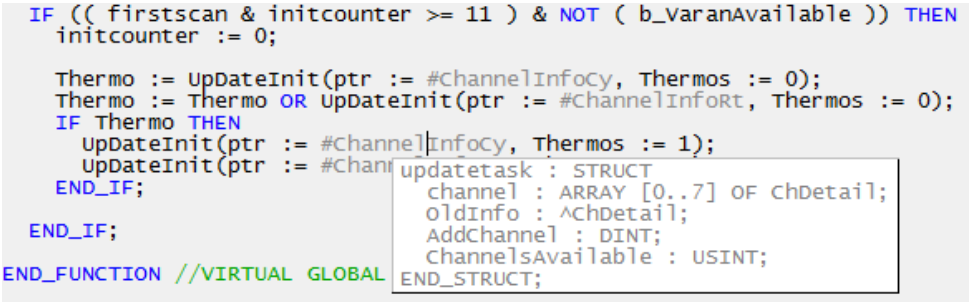
5.25 Displaying the Connections in the Object Network with Tooltip

With the data part (value display) of clients and servers in the network, completed connections are shown with tooltip.

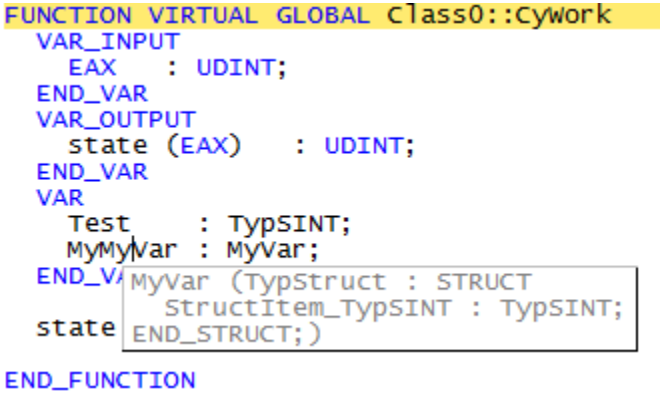


5.26 Displaying the Variable Type in the Code Editor with Tooltip

If you stop over a variable in the code editor with the mouse pointer, that has a type, it is shown in the Tooltip.

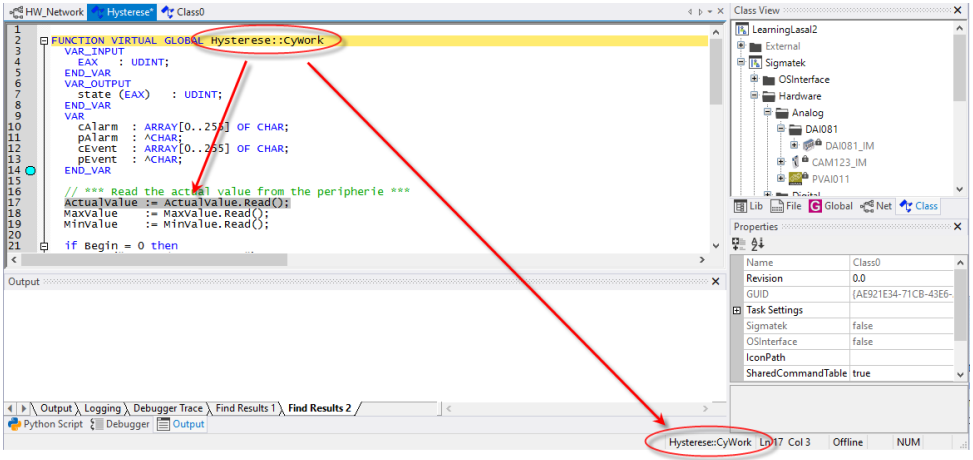


With “nested” types, the Tooltip shows all types: first the reference type then (in parentheses) the actual type:

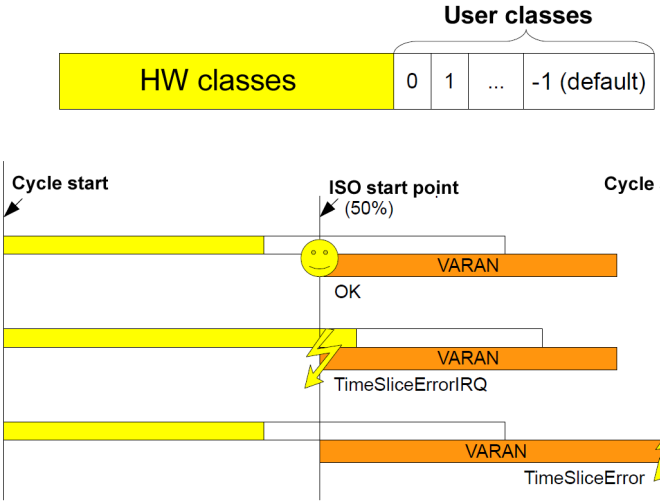


5.27 Display of Method Names in the Status Bar for Simpler Orientation

In the CodeEditor offline, the status bar shows the method at the cursor position, in Debug mode, the method is shown in which the instruction pointer is located.



5.28 Task Sequence without RtPostScan (up to Lasal 02.02.052)



Meaning of the TimeSliceErrorIRQ / TimeSliceError error messages

HW classes and VARAN Managers access the DPRAM at the same time. This can cause data Inconsistencies.

TimeSliceErrorIRQ

The processing of the HW classes has not yet been completed by the ISO start time; the isochronous VARAN task is therefore not yet released.

TimeSliceError

The processing of the VARAN Manager program (=data transfer on the VARAN bus) is not yet complete and the processing of the HW classes have already started.

Causes for Errors

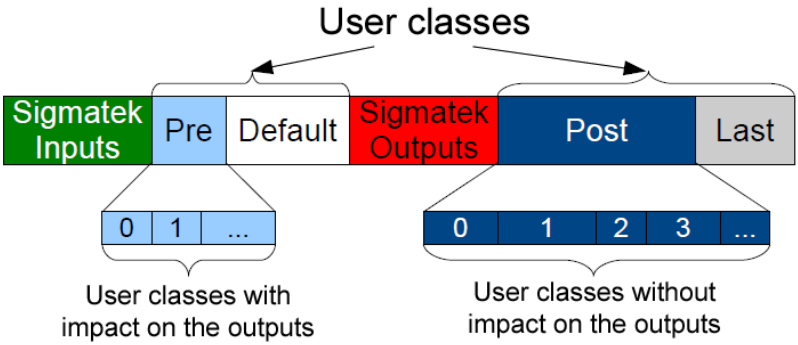
TimeSliceErrorIRQ

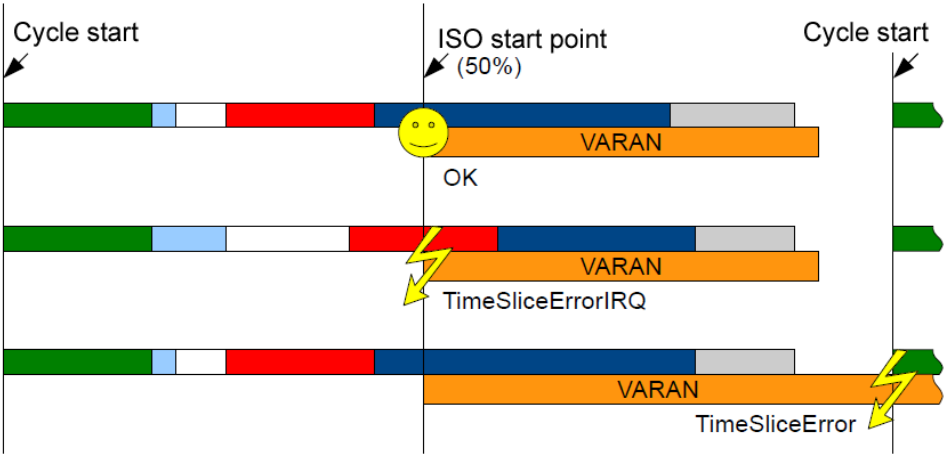
- IRQ's (Ethernet)
- The HW classes take to long

TimeSliceError

- Not required, several modules, 3 retries per module

5.29 Task Sequence with RtPostScan (with Lasal 02.02.054 and higher)





Causes for Errors

TimeSliceErrorIRQ

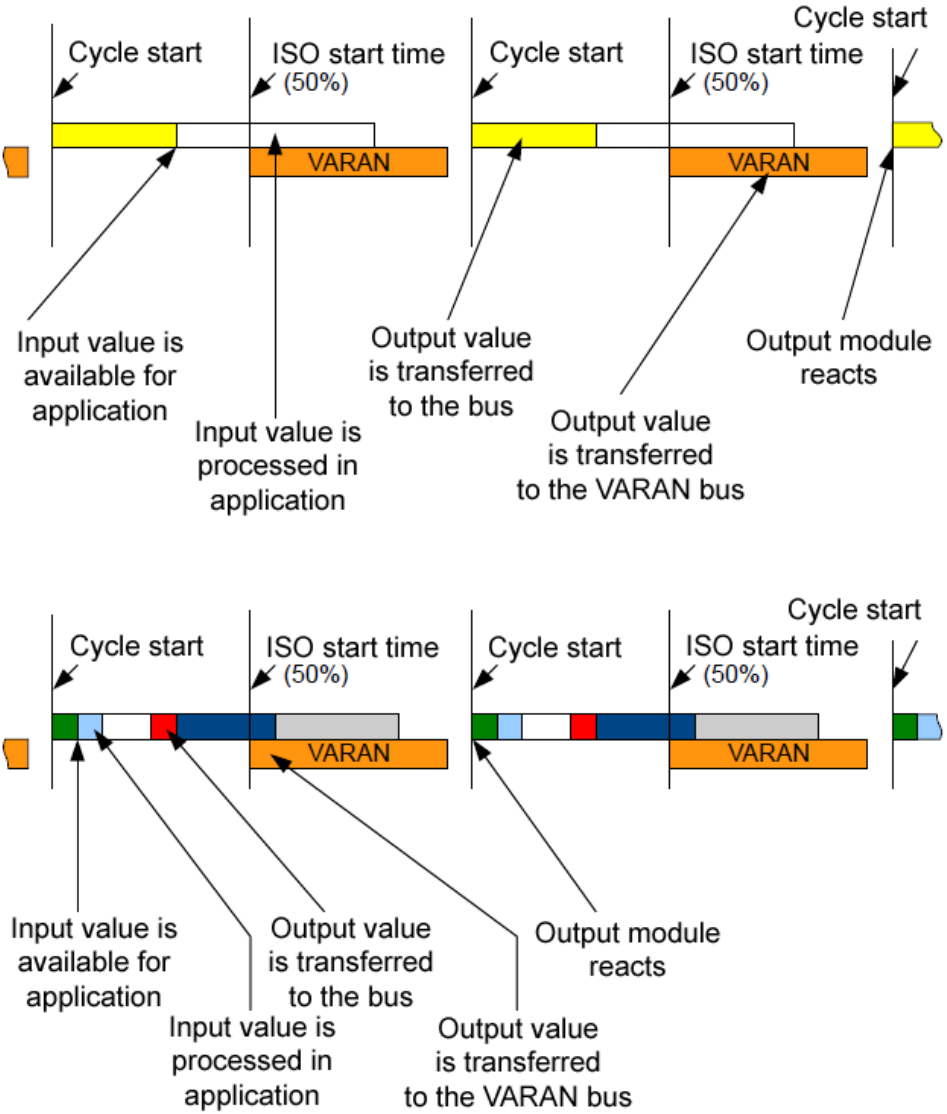
- IRQ's (Ethernet)

- The part of the application that handles the outputs takes too long (pre / default)
- The HW classes take too long (SIGMATEK Inputs / SIGMATEK Outputs)

TimeSliceError

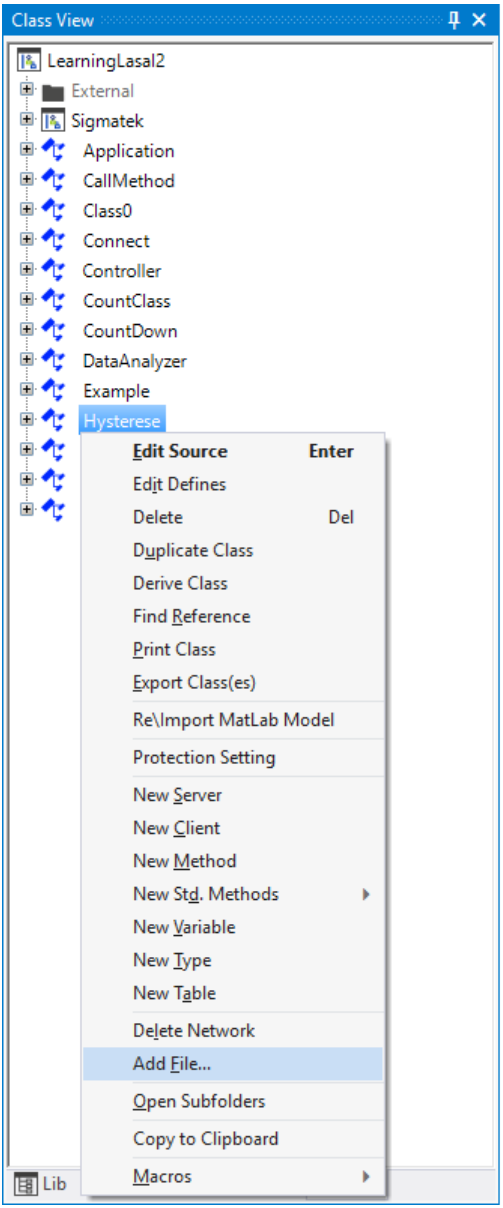
- Not required, several modules, 3 retries per module

5.30 Effects of RtPostScan

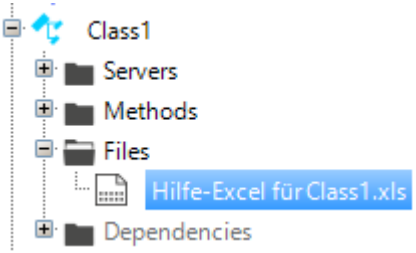


5.31 Attach Help Files to a Class

Help files can be added to normal classes as well as KOP classes. In the ClassView window, the class / LeaderClass must be clicked with the right mouse button and Add File selected. *.VOV files (Visual Object View) are added using this method.


Class View
LearningLasal2
External
Sigmatek
Application
CallMethod
Class0
Connect
Controller
CountClass
CountDown
DataAnalyzer
Example
Hysterese
Edit Source Enter
Edit Defines
Delete Del
Duplicate Class
Derive Class
Find Reference
Print Class
Export Class(es)
Re\Import MatLab Model
Protection Setting
New Server
New Client
New Method
New Std. Methods
New Variable
New Type
New Table
Delete Network
Add File...
Open Subfolders
Copy to Clipboard
Macros
Lib

The file added as described above is placed in the Files subfolder of the class / KOP.



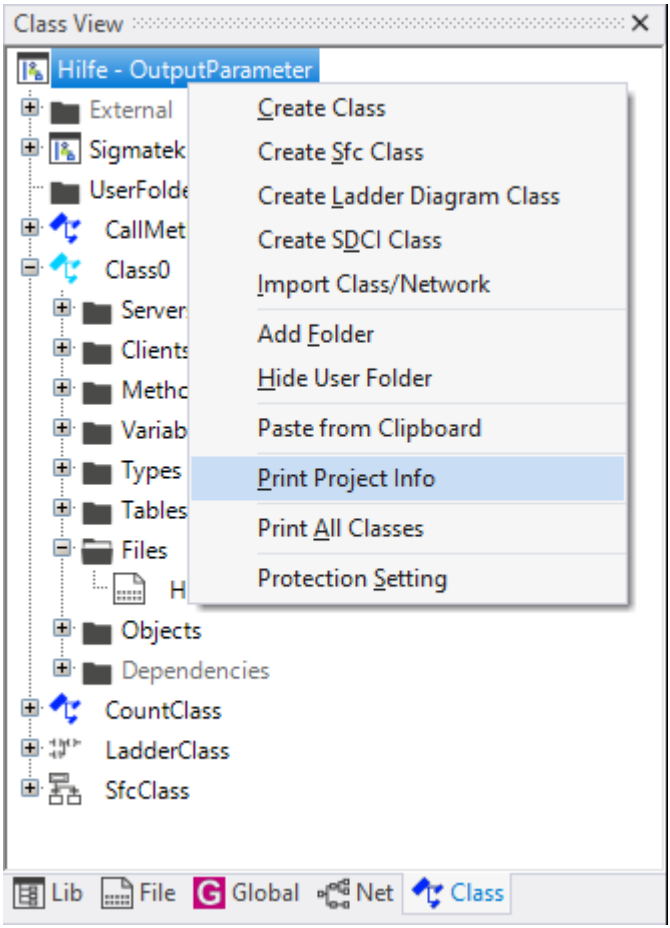
With a double click on the respective file, the corresponding application is opened. Files whose ending is unknown to LASAL (e.g. *.XLS for Excel files) are also opened automatically – provided that the appropriate display software is installed on the PC.

5.32 Print functionality

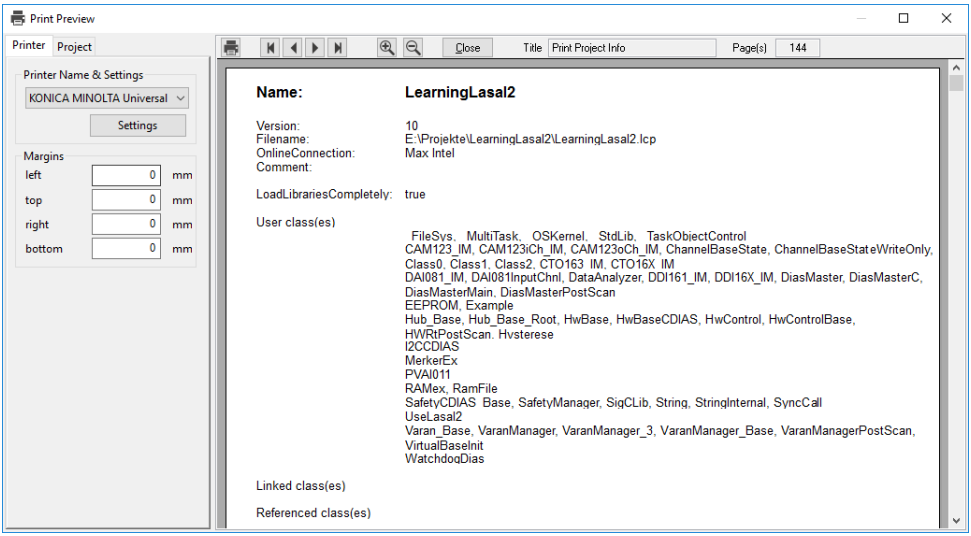
LASAL CLASS 2 allows to print out information about projects, classes and/or networks,

5.32.1 Project

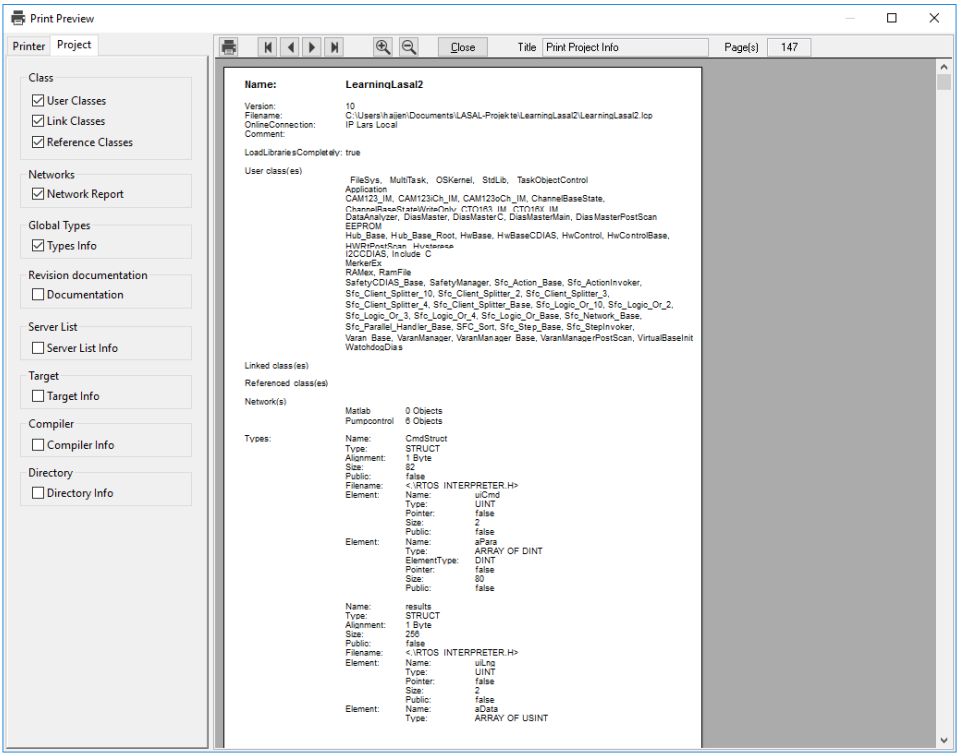
In order to print a project/ all relevant data of a project in LASAL CLASS 2, open the context menu with a right click on the project and select Print Project Info to open the **Print Preview window**.



In the **Print Preview window** under **Printer Name & Settings** you can select the desired printer. Under **Settings** you can make your print adjustments.



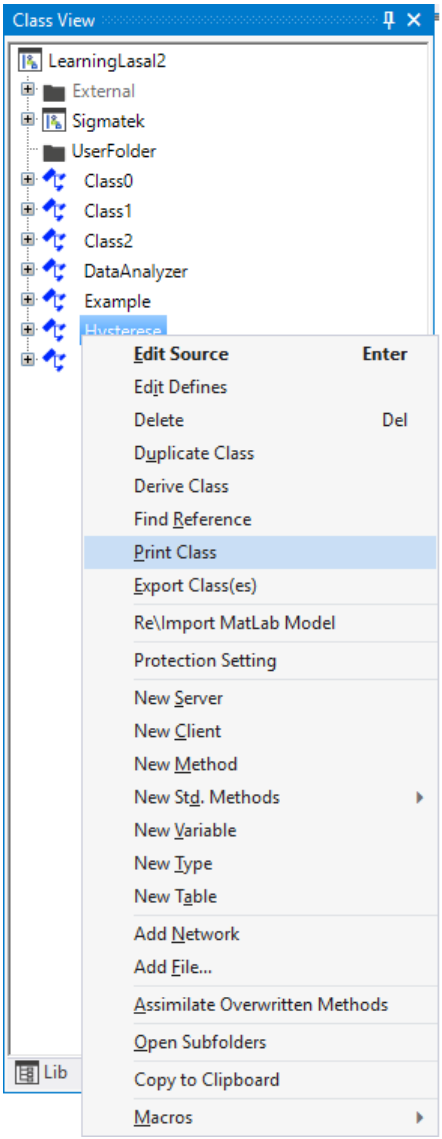
The menu **Margins** serves to define the distance to the margin in mm. Switch to register **Project** now.



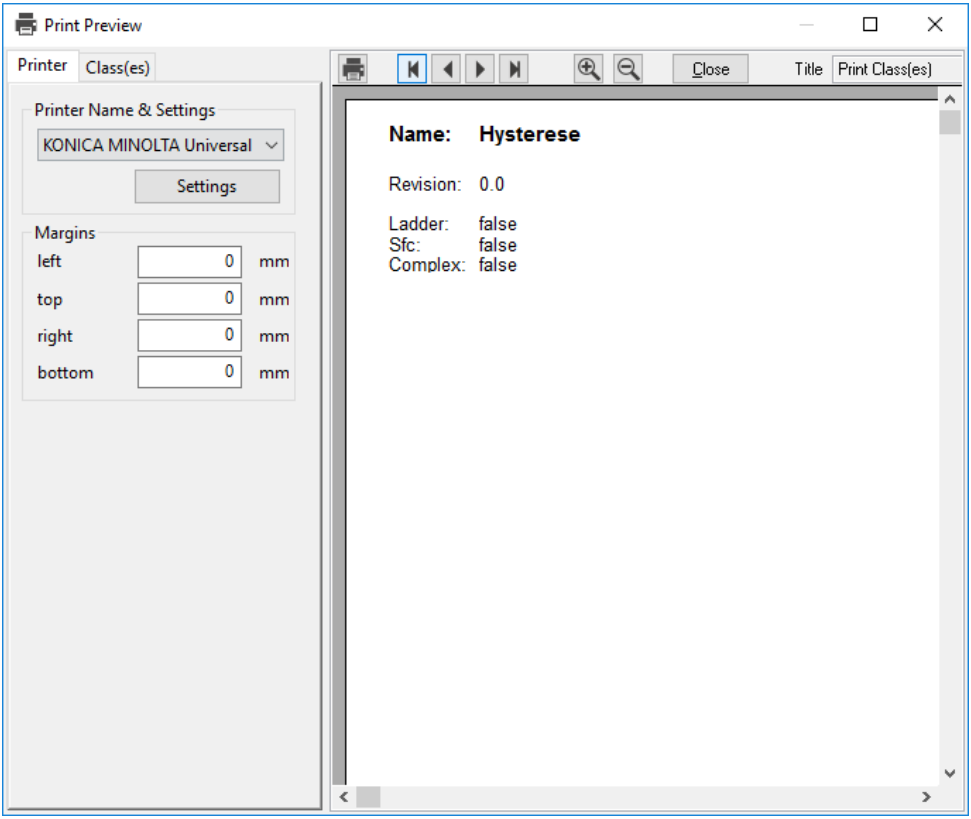
Under **Class**, **Networks** and **Global Types** and **Revision documentation** you can display and print the corresponding components of the project by ticking off the checkbox. With **Server List**, **Target**, **Compiler** and **Directory**, the respective information is printed in addition to that liste above. With  the pages can be run over. Use the buttons  and  to maximize, respectively, minimize the page display. To print the page use the  button. With  the Print Preview can be closed.

5.32.2 Class

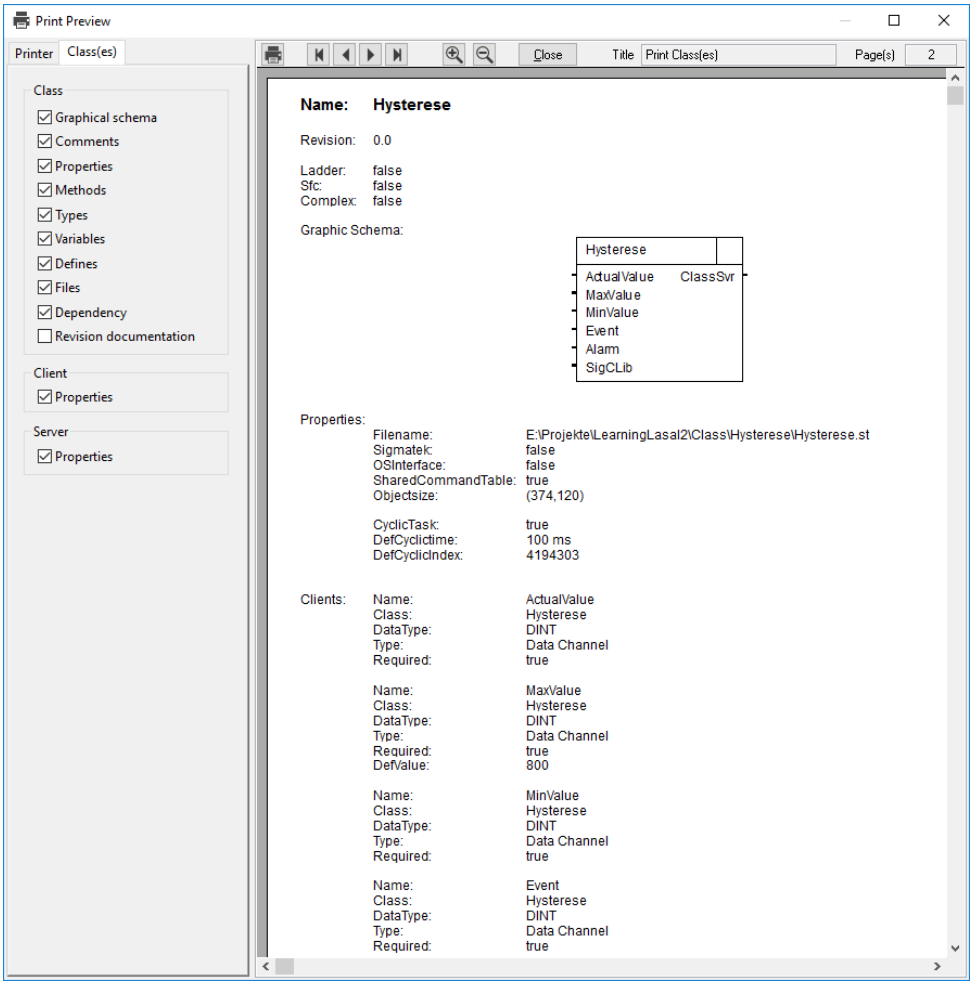
To print a class, respectively, its components select the entry **Print Class** in the context menu of the desired class (here: **Regulator**).

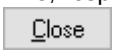

Class View
LearningLasal2
External
Sigmatek
UserFolder
Class0
Class1
Class2
DataAnalyzer
Example
Hysterese
Lib
Edit Source Enter
Edit Defines
Delete Del
Duplicate Class
Derive Class
Find Reference
Print Class
Export Class(es)
Re\Import MatLab Model
Protection Setting
New Server
New Client
New Method
New Std. Methods
New Variable
New Type
New Table
Add Network
Add File...
Assimilate Overwritten Methods
Open Subfolders
Copy to Clipboard
Macros

Thereby the Print Preview window opens. Under **Printer Name & Settings** you can select the desired printer. Under **Settings** you can make your print adjustments.



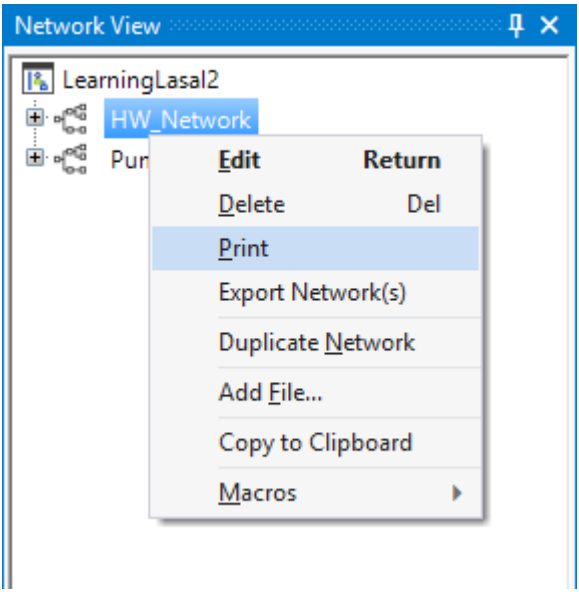
The menu Margins serves to define the distance to the margin in mm. Switch to register **Class(es)** now. Under **Class, Client** and **Server** you can display and print the corresponding components of the class by ticking off the checkbox.



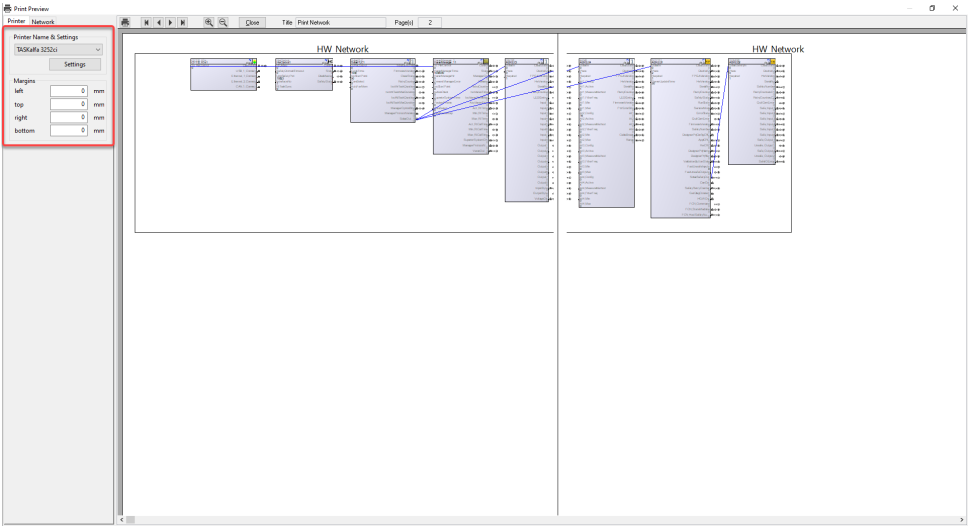
With  the pages can be run over. Use the buttons  and  to maximize, respectively, minimize the page display. To print the page use the  button. With  the Print Preview can be closed.

5.32.3 Network

To print a network select the entry **Print** in the context menu of the desired network (here: **Hardware**).

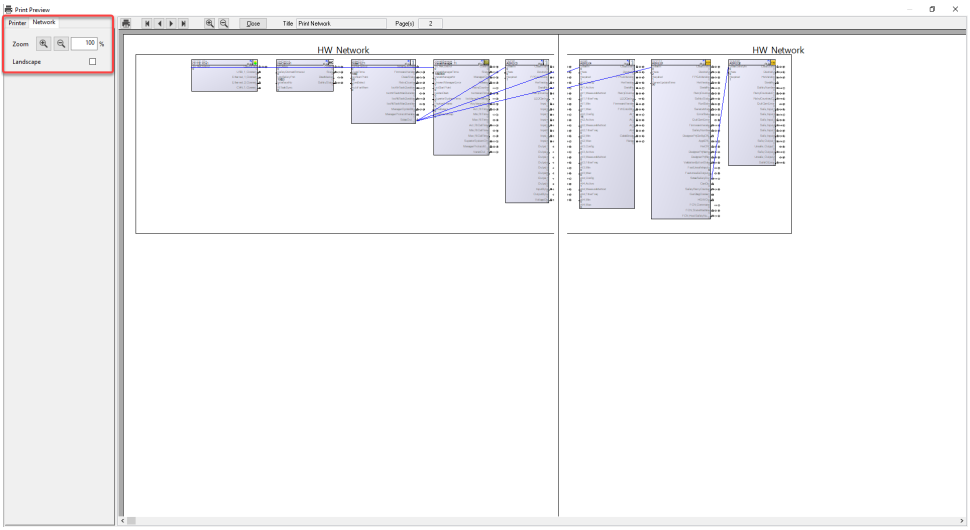


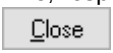
Thereby the **Print Preview window** opens. Under **Printer Name & Settings** you can select the desired printer. Under **Settings** you can make your print adjustments.



The menu **Margins** serves to define the distance to the margin in mm.

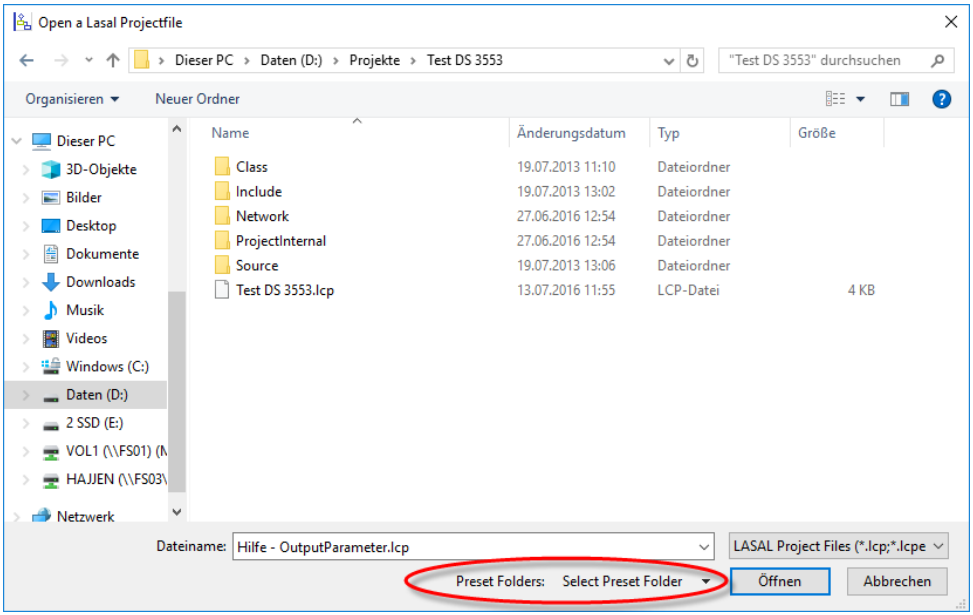
Switch to register **Network** now. With  you can maximize, respectively, minimize the graphic of the network. Activate the checkbox next to **Landscap**e to display the page in landscape format.



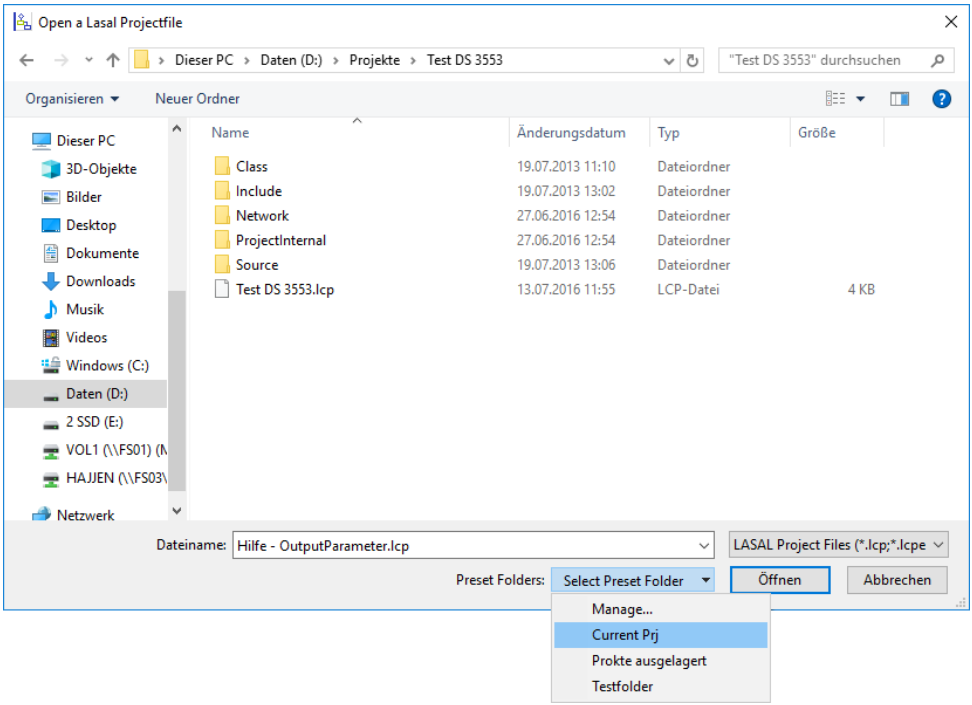
With  the pages can be run over. Use the buttons  and  to maximize, respectively, minimize the page display. To print the page use the  button. With  the Print Preview can be closed.

5.33 Preset Folders in the Open Project window

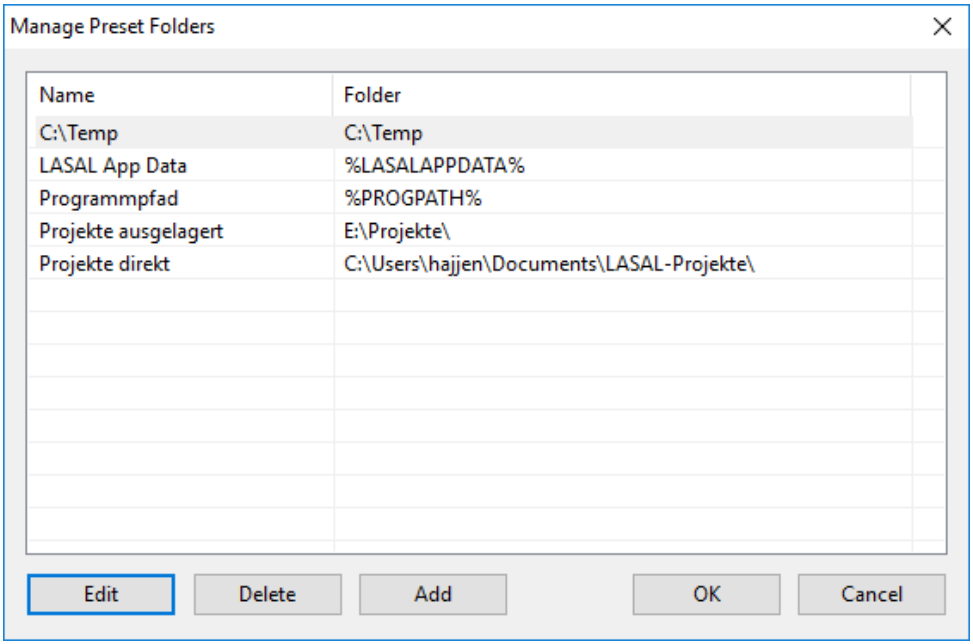
In the File Open dialog of LASAL CLASS 2 it is possible to define and manage own Preset folders, i.e. folders frequently needed.



With a left click, the selection "Preset Folders" is opened in the lower frame of the window. At the beginning of the list, two non-deletable entries are found: **Manage** and **Current Prj**. The latter is only visible when a project in LASAL CLASS is already open. With **Current Prj**, the program jumps directly to the folder containing the *lcp project file of the currently open project.

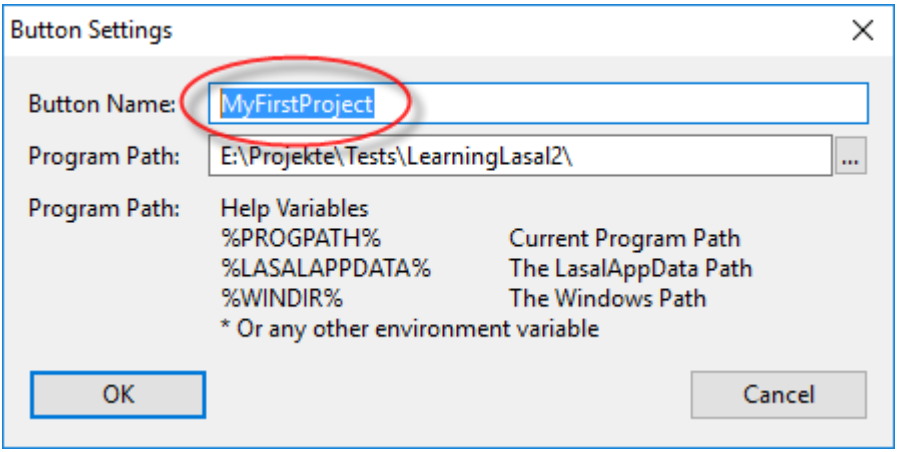


When **Manage** is selected, the **Manage Preset Folders** dialog opens:.

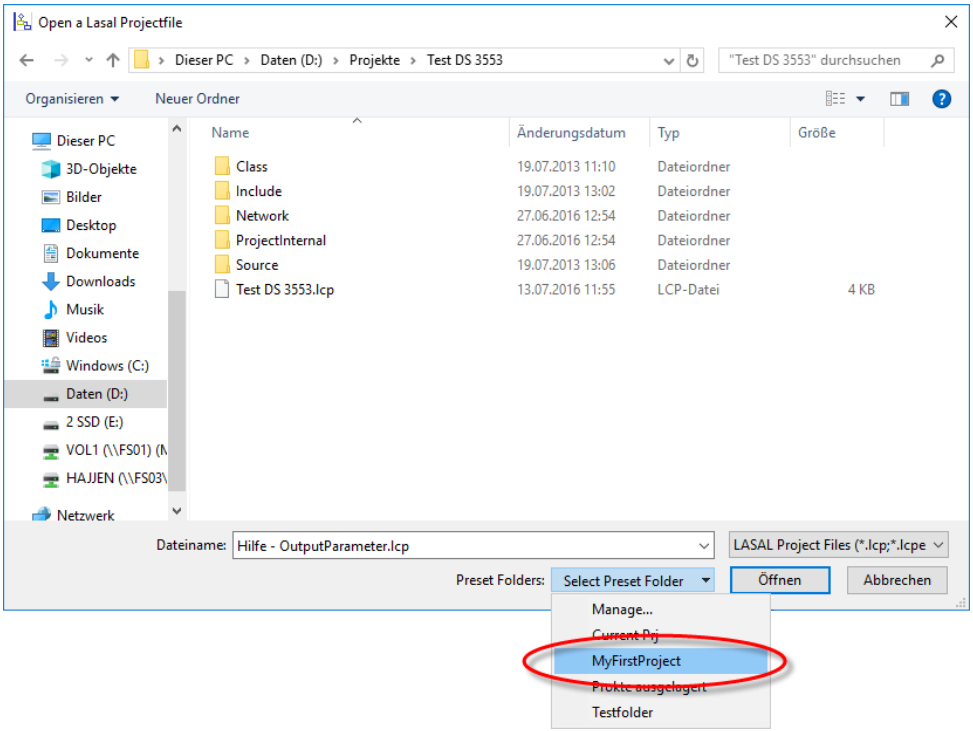


In this dialog, the preset folders can be managed. Here the options to **Add**, **Delete** or **Edit** an entry are available. If the dialog is ended with the **Cancel** button, the changes are discarded. With **OK**, the changes are saved.

With a click on the **Edit** or **Add** button, the **Button Settings** window opens:



Here, the list of displayed names, as well as which directory to open can be entered. Environment variables can be hereby used, which are shown in the dialog most often. Assign the name **MyFirstProject**, select a folder and confirm it with **OK**. The next time the Preset Folders menu is opened, the newly created folder is displayed in the list and can be accessed directly.


Open a Lasal Projectfile
Dieser PC › Daten (D:) › Projekte › Test DS 3553
"Test DS 3553" durchsuchen
Organisieren
Neuer Ordner
Dieser PC
3D-Objekte
Bilder
Desktop
Dokumente
Downloads
Musik
Videos
Windows (C:)
Daten (D:)
2 SSD (E:)
Netzwerk
Name
Änderungsdatum
Typ
Größe
Class 19.07.2013 11:10 Dateiordner
Include 19.07.2013 13:02 Dateiordner
Network 27.06.2016 12:54 Dateiordner
ProjectInternal 27.06.2016 12:54 Dateiordner
Source 19.07.2013 13:06 Dateiordner
Test DS 3553.lcp 13.07.2016 11:55 LCP-Datei 4 KB
Dateiname: Hilfe - OutputParameter.lcp
LASAL Project Files (*.lcp;*.lcpe
Preset Folders: Select Preset Folder
Öffnen
Abbrechen
Manage...
MyFirstProject
Testfolder

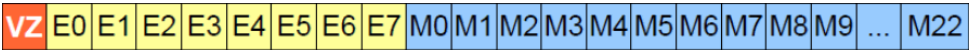
5.34 Known problems and limitations

Here we describe known problems and limitations in working with LASAL CLASS 2..

5.34.1 Real Numbers - Conversion Errors

When working with real numbers (REAL / LREAL), calculation or rounding errors may occur under certain circumstances.

5.34.1.1 Structure of a 32-bit Real Number in Memory



VZ

1 means a negative sign

0 means a positive sign

E0..E7

The exponent is calculated (E0..E7) – 127

Therefore positive and negative exponents are possible

M0..M22

The mantissa is calculated

1 + (M0 .. M22)

5.34.1.2 Example with Real Numbers

a = 100.1 …. decimal

The number 100.1 should be converted from the decimal system to the binary system.

The whole number and decimal parts are treated separately.

The following chapters describe the process in detail.

5.34.1.21 The decimal number 100 is converted to the binary system

Proceed as follows:

(1) Divide the number with remainder by 2.

(2) The remainder is the next digit (from right to left).

(3) If the (whole number) quotient is 0, the conversion is complete, otherwise take the (whole number) quotients of the new number and repeat from (1).

100 : 2 = 50 remainder: 0

50 : 2 = 25 remainder: 0

25 : 2 = 12 remainder: 1

12 : 2 = 6 remainder: 0

6 : 2 = 3 remainder: 0

3 : 2 = 1 remainder: 1

1 : 2 = 0 remainder: 1

Binary result: 1100100 = Decimal 100

5.34.1.22 The decimal break should be converted to the binary system

To obtain decimal digits, proceed as follows:

(1) Multiply the number by base 2

(2) The number before the decimal point is the next digit of the result

(3) Remove the number before the decimal point.

(4) Repeat from (1), until the remainder is 0, it starts to repeat or the desired precision is reached.

2 · 0.1 = 0.2 --> digit: 0

2 · 0.2 = 0.4 --> digit: 0

2 · 0.4 = 0.8 --> digit: 0

2 · 0.8 = 1.6 --> digit: 1

2 · 0.6 = 1.2 --> digit: 1

2 · 0.2 = 0.4 --> digit: 0

The "remainder" 0.4 already occurs in the 2nd step. After this point the decimal digits start to repeat periodically!!!

Result: 0,0001100110011001100110011001100110011001100110011...

5.34.1.23 Combining the number

Resultat: 1100100,0001100110011001100110011001100110011001100110011 …

5.34.1.24 Calculating an Exponent

Shift the decimal point until is in front of the first 1: → not the number of places

1100100,0001100110011001100110011001100110011001100110011 ...

n = 6

1,1001000001100110011001100110011001100110011001100110011 ...

Exponent = 127 + n

Exponent = 133

Binary exponent = 10000101

5.34.1.25 Determine the mantissa

Shift a number one place to the left

1,1001000001100110011001100110011001100110011001100110011

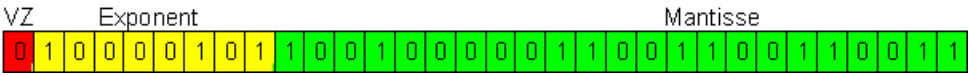
Shift

10010000011001100110011001100110011001100110011001100110

With a resolution of 32 bits, the mantissa has 23 bits in memory. Therefore only the first 23 bits are saved.

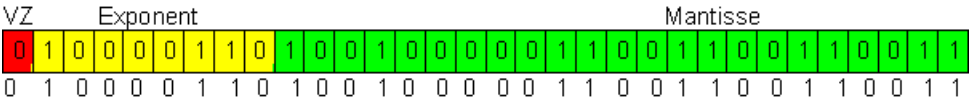
Mantissa = 10010000011001100110011real number 100.1 combined:

5.34.1.26 Real number 100.1 combined



When calculated back to decimal, the result is **100,0999984741211** = **100.1**.

5.34.1.27 Real number 200.2 combined



When calculated back to decimal, the result is **200,1999969482422** = **200.2**.

If remainder = 0 cannot be determined with the decimal places, minor calculation errors occur!

5.34.1.28 Multiplying real numbers

See Example:
https://www.wtamu.edu/academic/anns/mps/math/mathlab/beg_algebra/beg_alg_tut7_mult.htm (English)

http://www.sps-lehrgang.de/multiplikation-gebrochener-dualzahlen/ (German)

5.34.1.29 Source directory

Conversion:	http://www.arndt-bruenner.de/mathe/scripts/Zahlensysteme.htm
Calculation:	https://www.h-schmidt.net/FloatConverter/IEEE754.html
IEEE754 Norm:	http://de.wikipedia.org/wiki/IEEE_754
In detail:	http://www.validlab.com/goldberg/paper.pdf

5.34.2 Mixing Copy and Link Classes

This chapter discusses common problem that can greatly affect the reliability of your project, when you mix linked and copied classes.

5.34.2.1 Using linked classes

During software development with the LasalClass tool, it is a common practice to reuse classes from our libraries, as they provide many useful solutions to common challenges in industrial machine development.

The most common way to reuse our classes is **to link** them into your project.

The main advantage of linking a class is that the code was precompiled from the factory when LasalTools was released.

When you add a linked class, LasalClass will not recompile the class, but will only link it to the precompiled code, increasing your productivity by reducing building times.

5.34.2.2 Using copied classes

Another common way is **to copy** a SIGMATEK class into your project. There are many reasons why a class is copied into your project.

The first reason is when the user wants to extend an existing class. The class is copied and more functions can be added or the behavior of the class can be customized for your own project.

Another common case is the application of the test-driven development (TDD) or the V-model. In this type of development, it is important to closely control the project changes, otherwise too many resources are consumed in constantly testing and re-testing the engine. This becomes even more complex when the product has been on the market for a long time, as restoring the test setup requires significant effort.

According to the aforementioned practices, it is best **to decouple your project from all external influences**. To achieve this, **the best way is to copy the classes**. In a regular LasalTool upgrade, improvements for the classes are released, but according to the TDD methodology, even a single line of code change requires you to revalidate the entire system.

The **main disadvantage of copying** classes is that **compile times are significantly increased** because the source files are recompiled as needed.

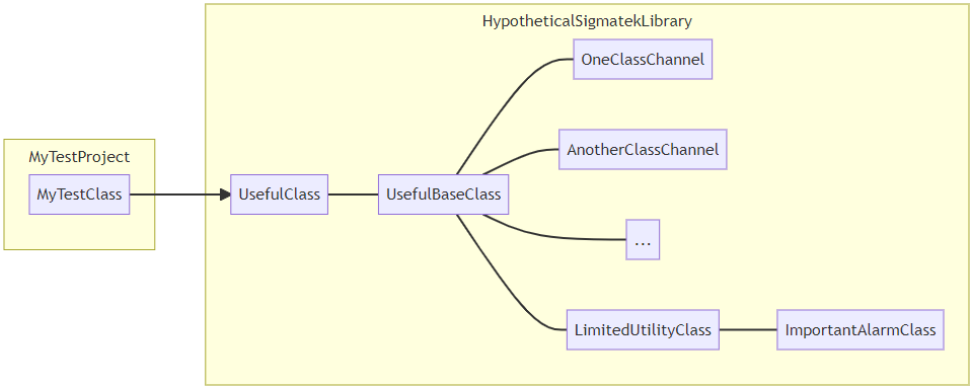
5.34.2.3 Mixing both methods

It sounds **very convenient** to mix both types of class reuse, because you can choose the best method for each individual case. This option is quite tempting because of its flexibility, but **it can have serious consequences and unpredictable side effects**.

To better understand the problem, the following example will serve:

5.34.2.31 Example

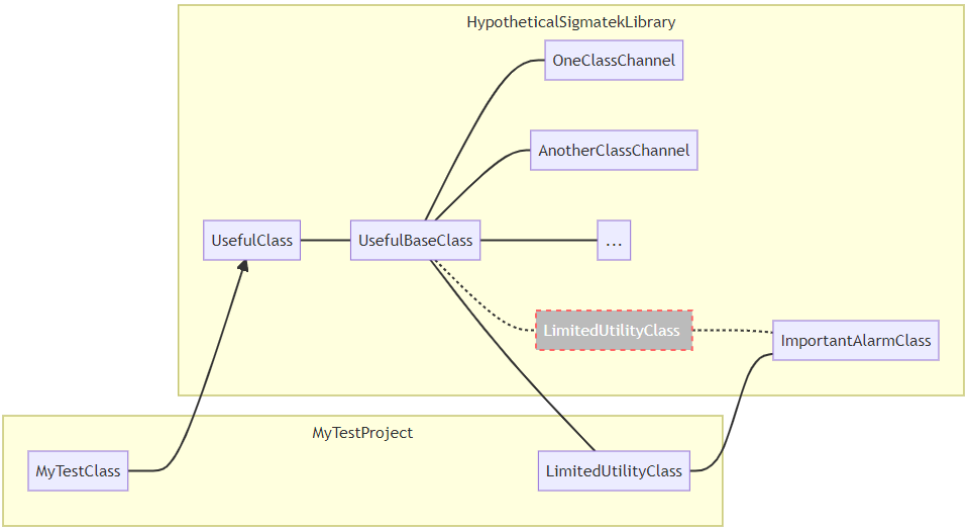
Let's imagine a hypothetical class hierarchy as in the following diagram:



The user creates the **MyTestProject** project, which has a connection to the **UsefulClass** from the HypotheticalSigmatekLibrary.

Once the project is running, the user wants to extend the **LimitedUtilityClass**. Let's say that this class has an array that can store 10 items, but the user wants to be able to store 10 more items. So he copies the class into his project to resize the array to a total of 20 elements.

Now the user thinks he has the following layout:



Well, not quite...

Indeed, the linker collects the classes as in the picture above, and you have a chance that this partially works, but one crucial point causes problems.

As described in the Using linked classes chapter, "The main advantage of linking a class is that **the code was precompiled from the factory at the time LasalTools was released**".

In our case, this means that all classes using **LimitedUtilityClass** from the **HypotheticalSigmatekLibrary** were statically created based on a version of the **LimitedUtilityClass** class with an array of 10 elements instead of 20. After all, this version was created at the factory, at the time your current LasalTools version was released.

The first clear difference is that the sizeof(**LimitedUtilityClass**) changes depending on the perspective: whether the class in the library, or the one from the project is meant.

It must also be noted that **all accesses to an object** are made with the **this** pointer. To access an element, the compiler calculates the distance between the **this** pointer and the element.

For example, let's imagine an object that has a DINT element named **myDint**, and the following assignment:

```
myObject.myDint := 55;
```

Let's assume that the **myDint** element is 20 bytes away from the **this** pointer. The compiler will therefore add 20 bytes to the **this** pointer of the **myObject** object, and store the value 55 at the resulting address.

More difficulties...

Now if you add a **virtual method**, channels included, then things get really scary. Let's just say that virtual methods are a set of function pointers added to the beginning of a class.

- The first obvious change is the value of the **sizeof()** operator.
- The second is that all elements will be shifted by 4 bytes (and 4 bytes per virtual method added). So not a single element will work as expected!
- Calling a virtual method can point to another method. Why? Let's get to the bottom of it....

Example of a virtual function table

Let's imagine that the **LimitedUtilityClass** has 3 virtual methods (**MyVirtual1**, **MyVirtual2**, **MyVirtual3**) and a 10 element DINT array.

Then an object instance in memory will look like this:

```
THIS pointer ---> +00 +-------------+
                      | MyVirtual0  |
                  +04 +-------------+
                      | MyVirtual1  |
                  +08 +-------------+
                      | MyVirtual2  |
                  +12 +-------------+
                      | MyArray[10] |
                  +52 +-------------+
```

So, if we want to transform **LimitedUtilityClass** by adding a virtual method and 10 additional elements to the array, some changes in the memory layout can be seen.

First of all, it must be noted that the order of the virtual functions is not really predictable, since some entries are reserved by the compiler for special methods.

For this example, assume that the new virtual method is called **MyNewVirtual**, and is inserted in the middle of the virtual table due to the internal rules of the compiler. This could cause the following memory layout:

```
THIS pointer ---> +00 +--------------+
                      | MyVirtual0   |
                  +04 +--------------+
                      | MyNewVirtual |
                  +08 +--------------+
                      | MyVirtual1   |
                  +12 +--------------+
```

```
    | MyVirtual2   |
+16 +--------------+
    | MyArray[20]  |
+96 +--------------+
```

Now you can observe what happened:

- the value of **sizeof()** was changed from 52 to 96 bytes
- The element **MyArray[0]** was moved in memory from offset 12 to offset 16
- A new virtual method was inserted at offset 4, which shifted two other virtual methods by 4 bytes

From your project's point of view, everything works fine because it knows the current layout of the object, and any new offsets are handled correctly.

But let's imagine that the library was compiled in the past: A call to **MyVirtual1** will point to **MyNewVirtual** instead, and a call to **MyVirtual2** will call **MyVirtual1**.

Depending on the logic of your program, and the way it uses these methods, the results will vary greatly from what we expect.

But that's not all! Let's look at offset 12 in both memory layouts. For the project, offset 12 corresponds to **MyVirtual2** and for the library it is **MyArray[0]**. When a compiled code in the library writes to the **MyArray[0]** location, it overwrites the actual virtual function **MyVirtual2**. It is very likely that the stored data at this position 0 of the array has no relation to a function pointer. In this case, when **MyVirtual2** is called from the project's point of view, an access exception will be thrown and the controller will crash.

Other known weird behaviors

A common side effect can also be observed in the LASAL CLASS 2 IDE. It happens very often that changes to the **LimitedUtilityClass** are undone. For example, new members disappear, or array boundaries are reset.

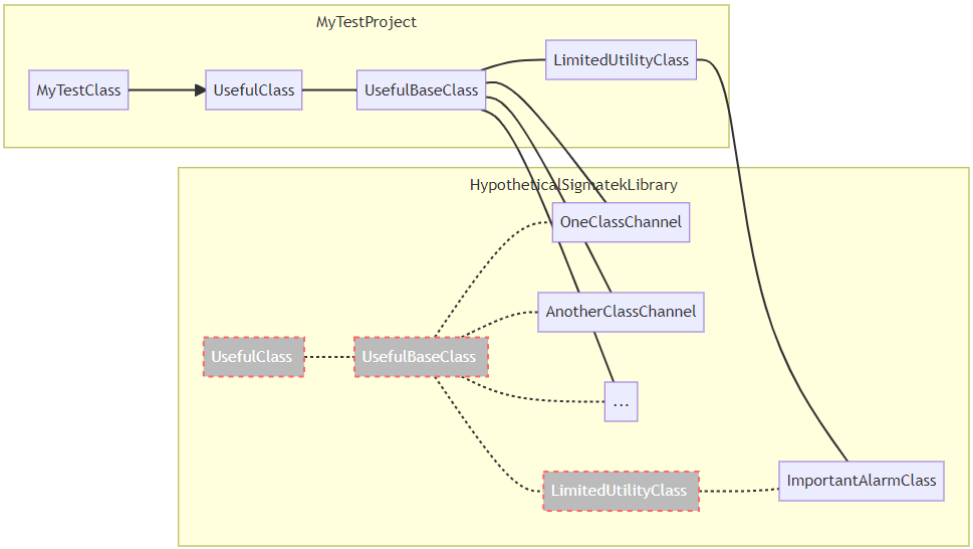
The reason for this behavior and the countermeasures are very complex, and there is no general rule, because the result depends on the compilation order, i.e. the order of all `#included` files and the dependency of the classes on each other.

During compilation, the compiler reuses the symbol tables as much as possible for performance reasons; and it is not uncommon for it to read the obsolete version of the symbol table through a library class, which explains why changes already applied are rolled back.

5.34.2.4 How to fix this issue

How to solve this problem?

To fix this problem, we need to copy all parent classes for **LimitedUtilityClass**. In other words, all classes from the library that depend on **LimitedUtilityClass**. For the example used here, we also need to copy **UsefulClass** and **UsefulBaseClass**, as shown in the diagram below:



Please note that classes that have a lower rank in the class hierarchy - like in our example **OneClassChannel,** or **ImportantAlarmClass** - do not need to be copied, as they are not bound to the **LimitedUtilityClass** (the binary code is decoupled).

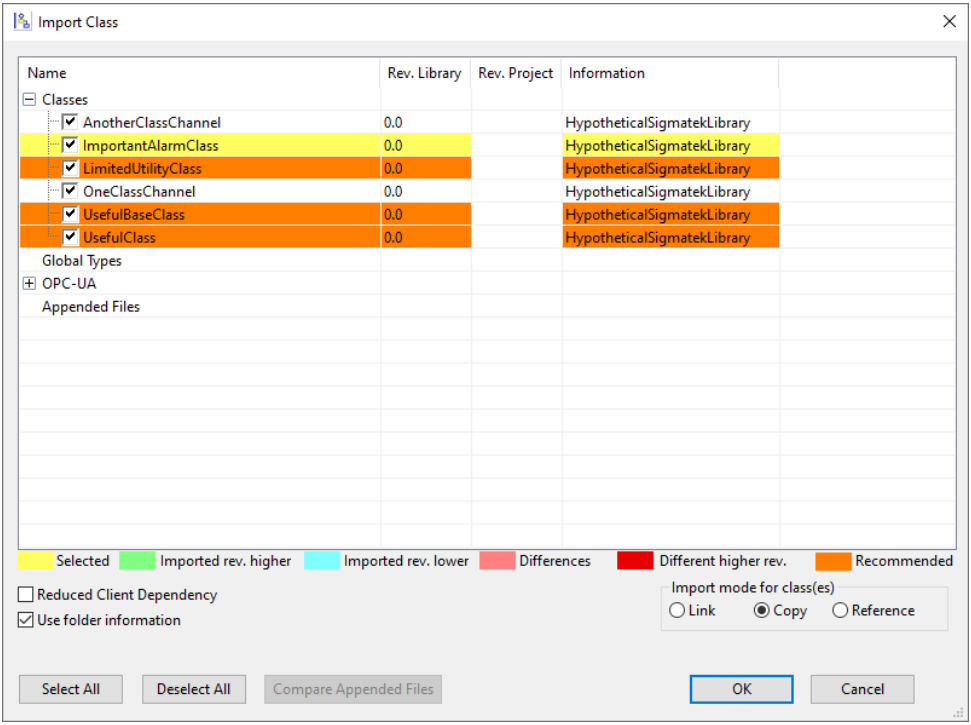
5.34.2.5 General recommendations

To summarize, if possible, one of the following rules should be applied:

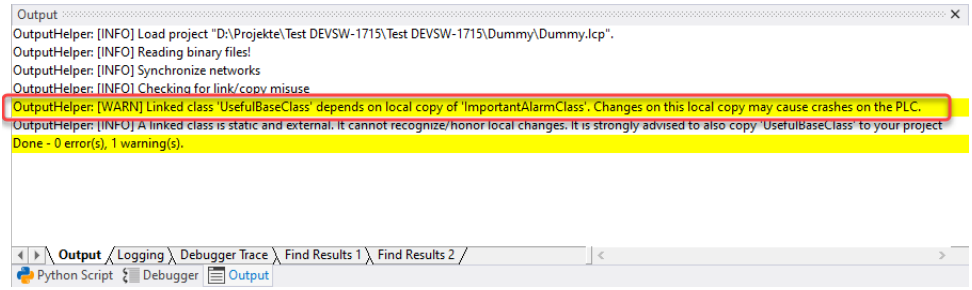
- **All** library classes are **linked**, or
- **All** library classes are **copied**

If none of the above options is possible, use **LasalClass2 version 02.02.223 or higher**, and copy the class again.

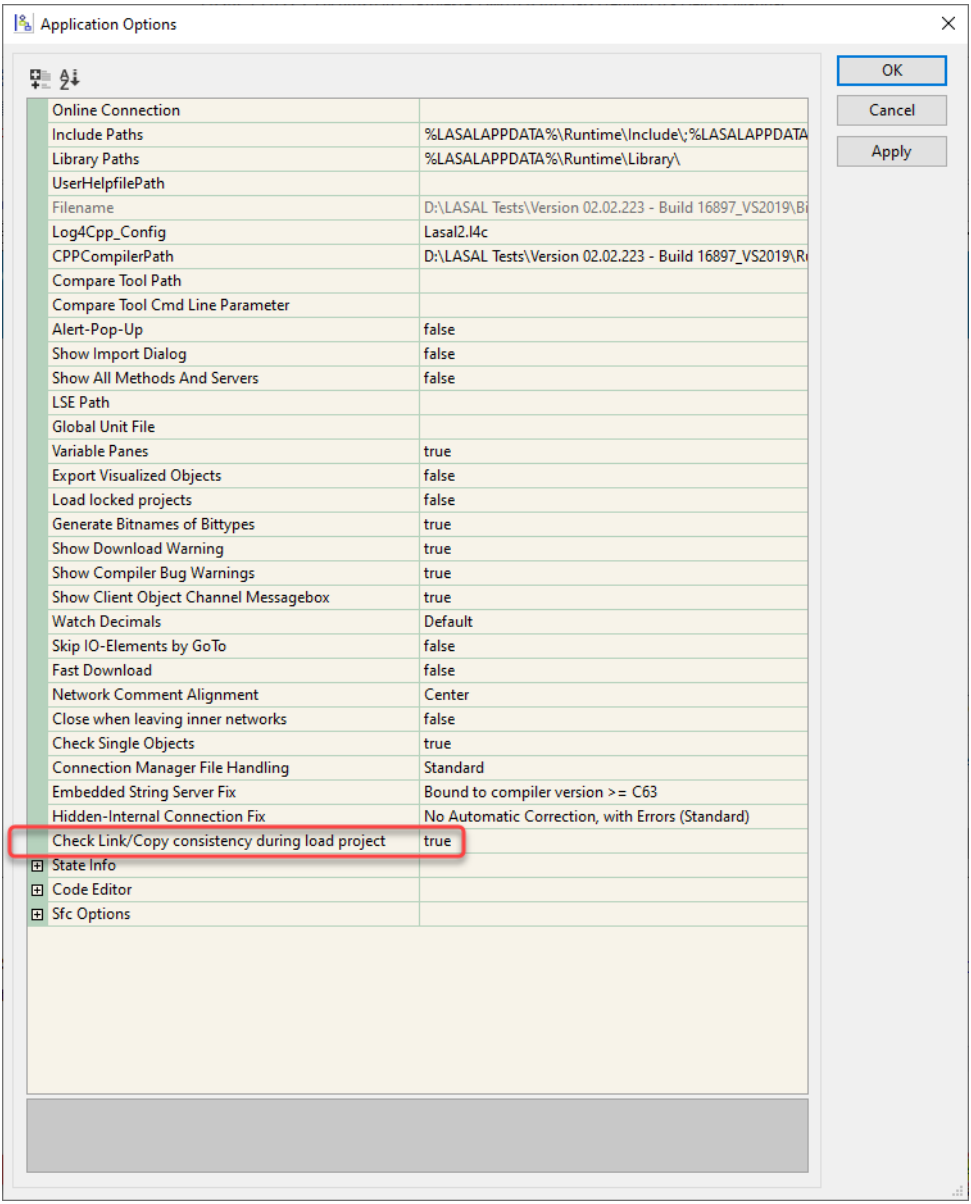
Since this version, a class dependency evaluation function has been added to show all classes that may be affected by a local change to the class selected for import. The image below is an example where a class was selected to be copied (highlighted in yellow) and all other **parent classes** (highlighted in orange) were also selected:



Since this release, projects are checked on open to detect class relationships that can cause this type of problem. Loader trace messages are displayed, as shown in the following figure:



This check can be prevented under Extras / Options with the setting "Check Link/Copy consistency during load project" = **false**. By default, this option is set to **true**:

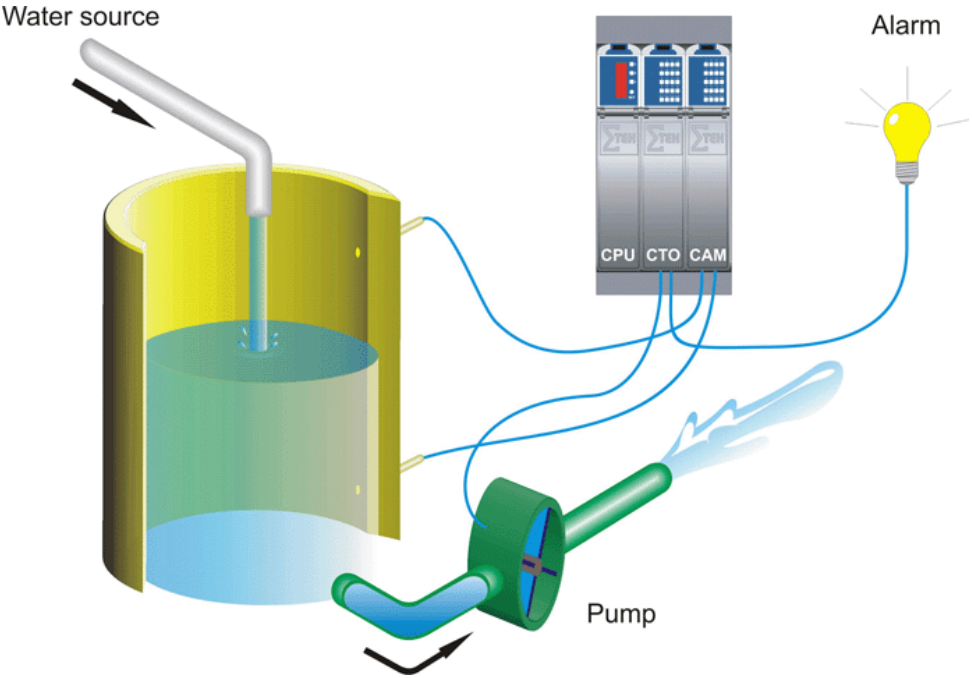

Application Options
OK
Cancel
Apply
Online Connection
Include Paths
%LASALAPPDATA%\Runtime\Include\;%LASALAPPDATA
Library Paths
%LASALAPPDATA%\Runtime\Library\
UserHelpfilePath
Filename
D:\LASAL Tests\Version 02.02.223 - Build 16897_VS2019\Bi
Log4Cpp_Config
Lasal2.l4c
CPPCompilerPath
D:\LASAL Tests\Version 02.02.223 - Build 16897_VS2019\R
Compare Tool Path
Compare Tool Cmd Line Parameter
Alert-Pop-Up
false
Show Import Dialog
false
Show All Methods And Servers
false
LSE Path
Global Unit File
Variable Panes
true
Export Visualized Objects
false
Load locked projects
false
Generate Bitnames of Bittypes
true
Show Download Warning
true
Show Compiler Bug Warnings
true
Show Client Object Channel Messagebox
true
Watch Decimals
Default
Skip IO-Elements by GoTo
false
Fast Download
false
Network Comment Alignment
Center
Close when leaving inner networks
false
Check Single Objects
true
Connection Manager File Handling
Standard
Embedded String Server Fix
Bound to compiler version >= C63
Hidden-Internal Connection Fix
No Automatic Correction, with Errors (Standard)
Check Link/Copy consistency during load project
true
State Info
Code Editor
Sfc Options

6 THE FIRST LASAL PROJECT

6.1 Requirements

On the following pages, a simple LASAL pump control project is constructed Step by Step.

As shown in the illustrations below, water flows into a tank. When the maximum fill value is reached, the pump is activated and the level in the tank begins to sink. When the minimum fill level is reached, the pump is deactivated and the tank refills.



Required Hardware

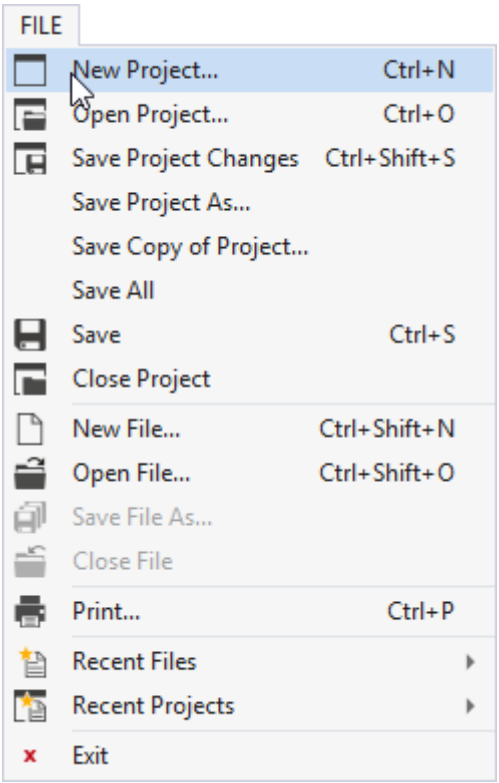
- CPU
- Digital output card
- Analog Input card

6.2 Implementation

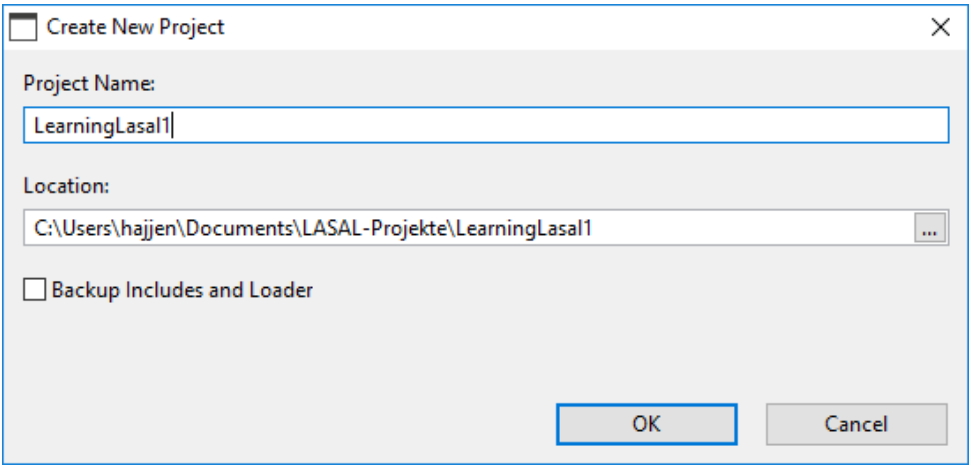
Before beginning with this project, it is recommended that the chapter The Design Environment of LASAL CLASS 2 be carefully read to become familiar with the LASAL CLASS 2 graphical user interface and to create the first project without problem.

6.2.1 Creating a New Project

To start a new project, in the menu list, select **File** then **New Project…**.



Next, the **Create New Project window** appears:



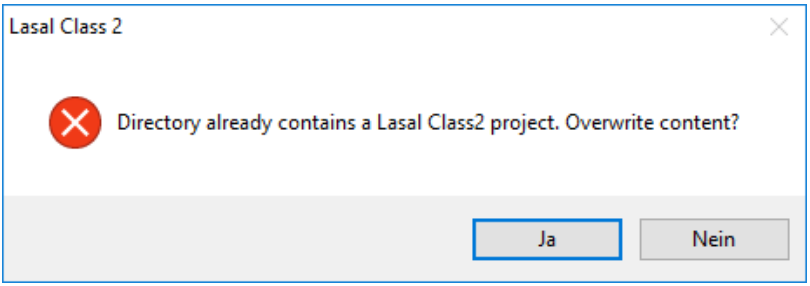
In the **Project Name:** field, assign a name for the project.

Under **Location:**, define the path to the project by entering it directly or clicking the ... symbol with the left mouse button and selecting the path.

A sub-file with the same name as the project is generated automatically.

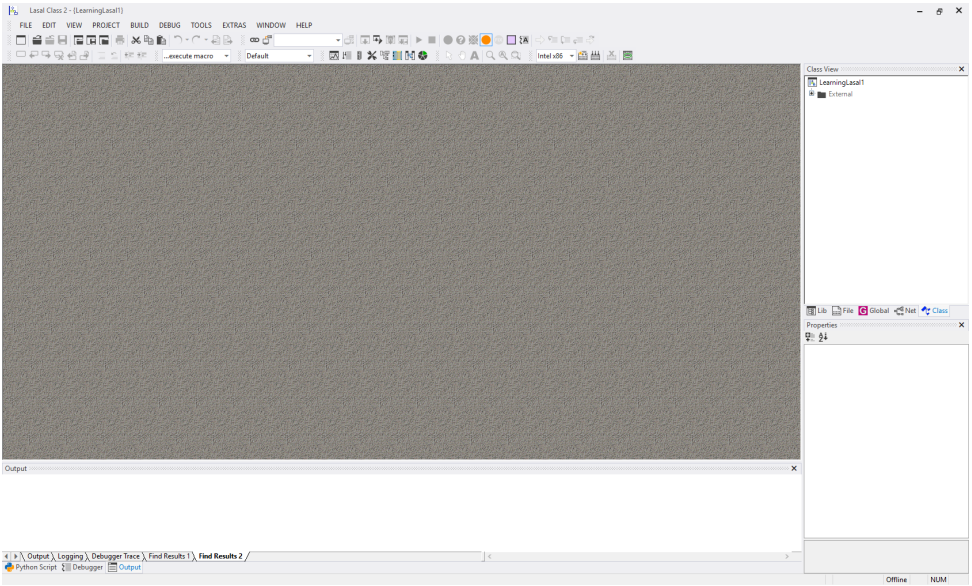
If Backup **Includes and Loader** is selected, the current loader and all includes are saved. This ensures that at a later time, the project can be loaded into a PLC with a new OS / loader without difficulty (**PLC = PC**).

If the directory entered already exists, a window appears and asks whether the existing directory should be overwritten (yes) or not (no).



Confirm with **OK**.

The screen should now appear as pictured below.

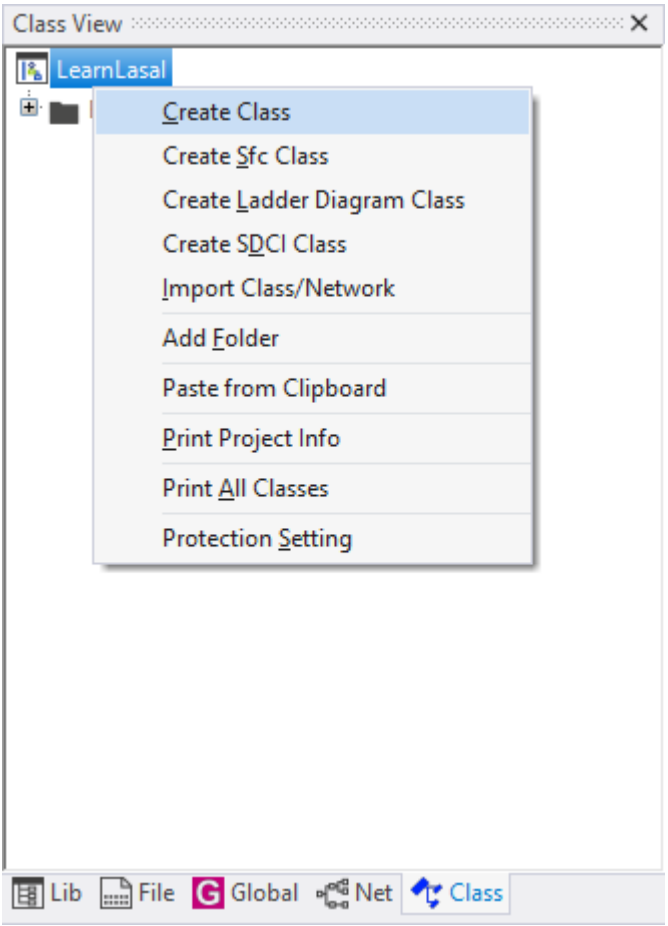


6.2.2 Creating a Class

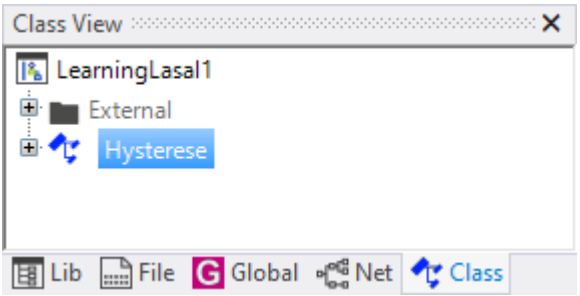
In the following Step, the creation of a new class in LASAL CLASS 2 is shown.

First, ensure that the Class View window is open. With a double click with the right mouse button on the project name (LearningLASAL) activate the context menu and select **Create Class**.

More information over the **Class View window** can be found under Classes.

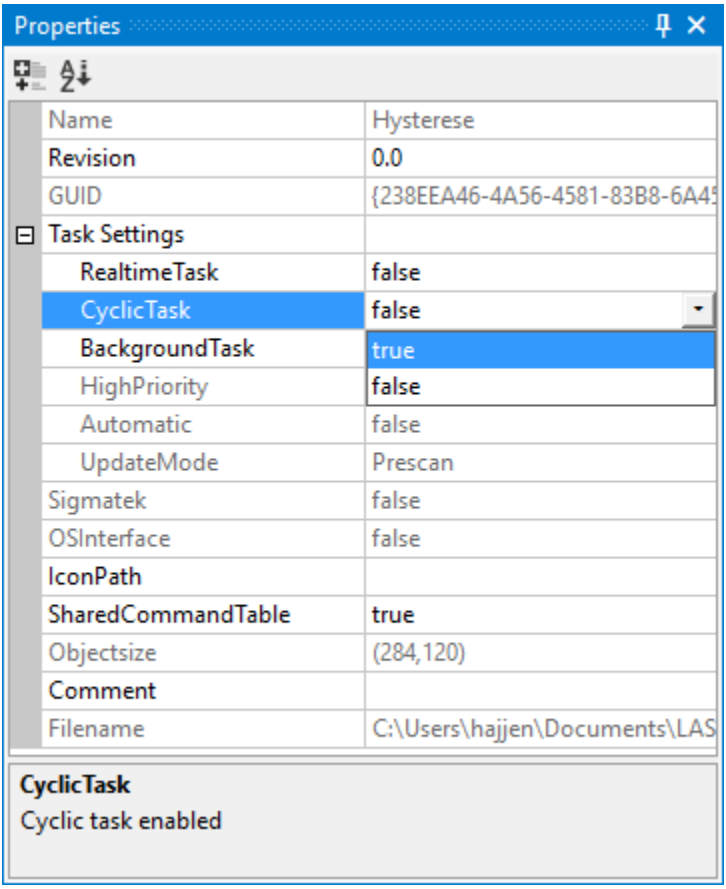


The opened field, enter the name of your class (**Hysteresis**).

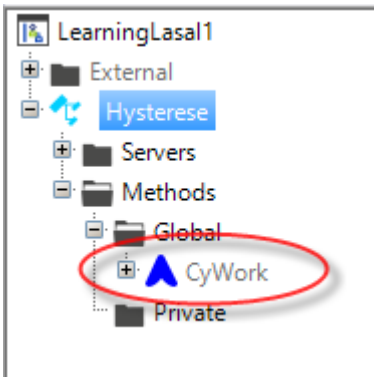

Class View
LearningLasal1
External
Hysterese
Lib
File
Global
Net
Class

6.2.3 Selecting a Method

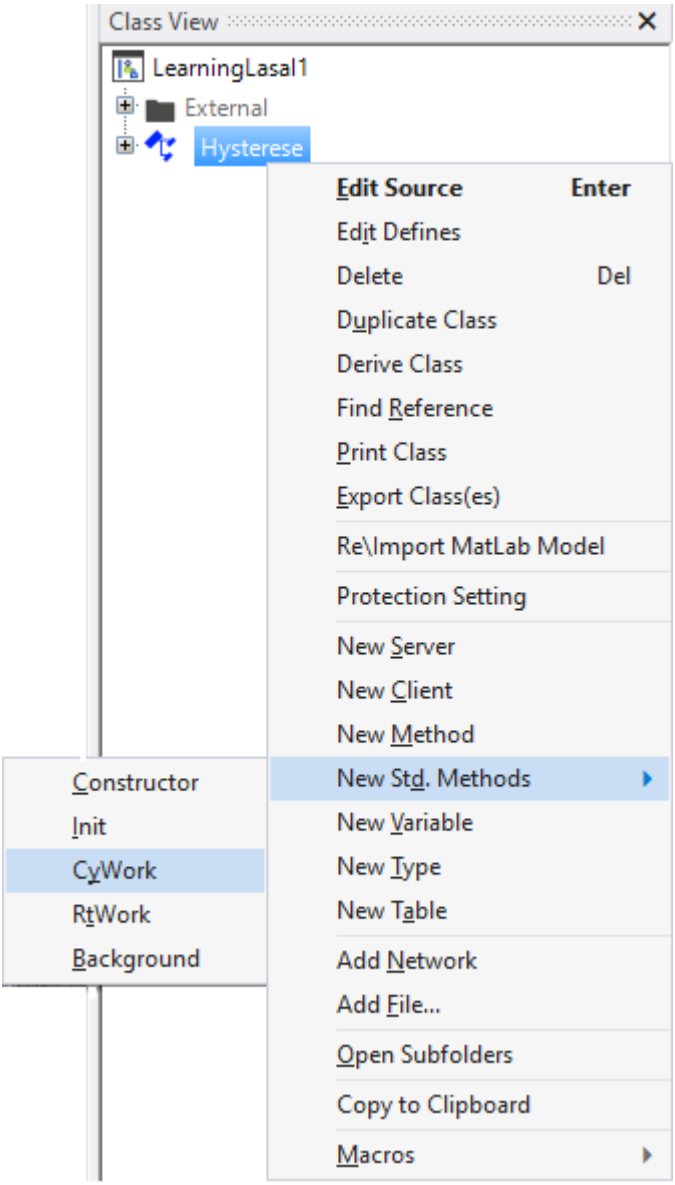
The following illustrations show how the **CyWork method** is set for Hysterese class. Set - after clicking the class Hysterese in the Class tree - **CyclicTask** to **True** in the Properties window, as illustrated on the picture.



Thus the **CyWork method** under **Methods, Global** is created automatically.



The **CyWork method** can also be created without setting the **CyclicTask** of the **Hysterese** class to **True** in the Properties window. Only select **New Std. Methods => CyWork** in the context menu of the class.

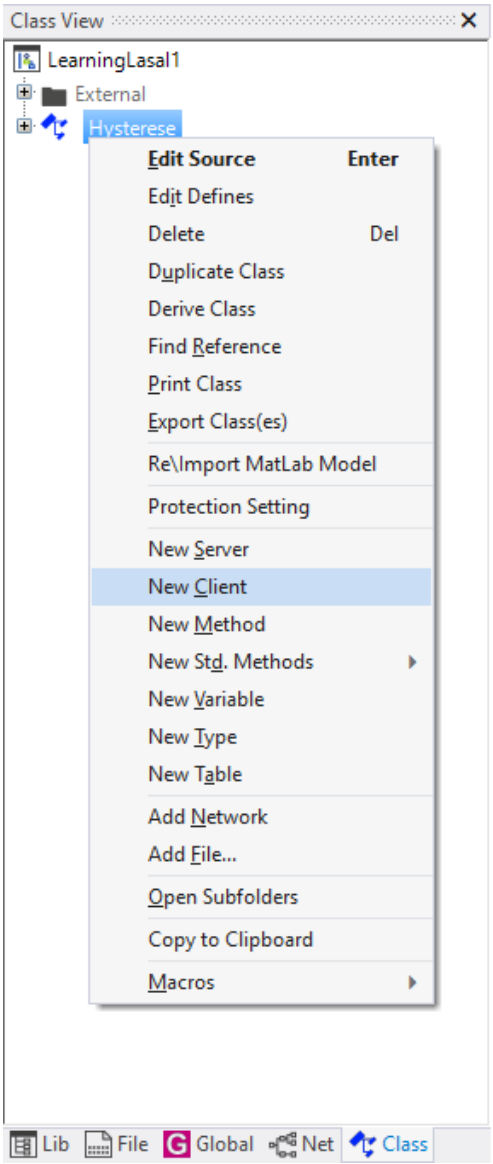

Class View
LearningLasal1
External
Hysterese
Edit Source Enter
Edit Defines
Delete Del
Duplicate Class
Derive Class
Find Reference
Print Class
Export Class(es)
Re\Import MatLab Model
Protection Setting
New Server
New Client
New Method
New Std. Methods
New Variable
New Type
New Table
Add Network
Add File...
Open Subfolders
Copy to Clipboard
Macros
Constructor
Init
CyWork
RtWork
Background



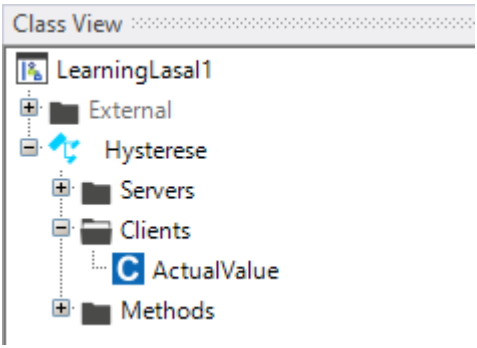
However, in this way you have to set the **CyclicTask** to **True** again in the Properties window of the Hysterese class to activate the method!

6.2.4 Creating a Client

In the Class View window, the required clients for the hysteresis class are created. Click on the **Hysteresis** class with the right mouse button to open the context menu and select **New Client**.



Another field is opened in which the corresponding client name (**ActualValue**) is entered.



Repeat this process for the remainder of the clients required for this class:

MaxValue

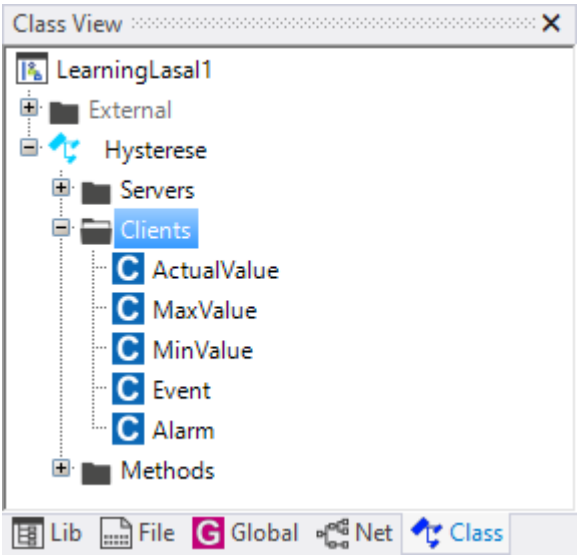
MinValue

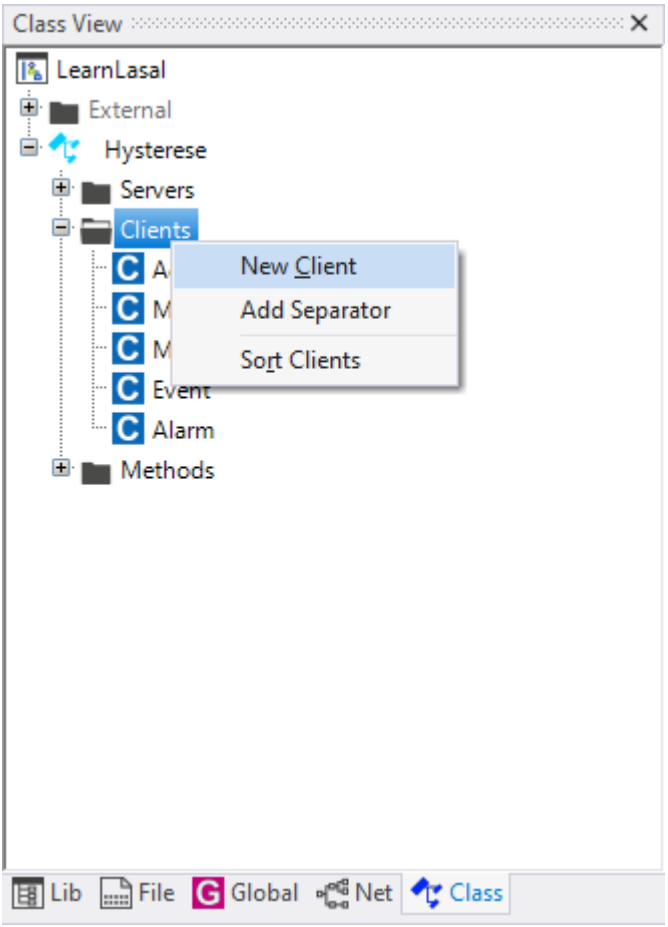
Event

Alarm



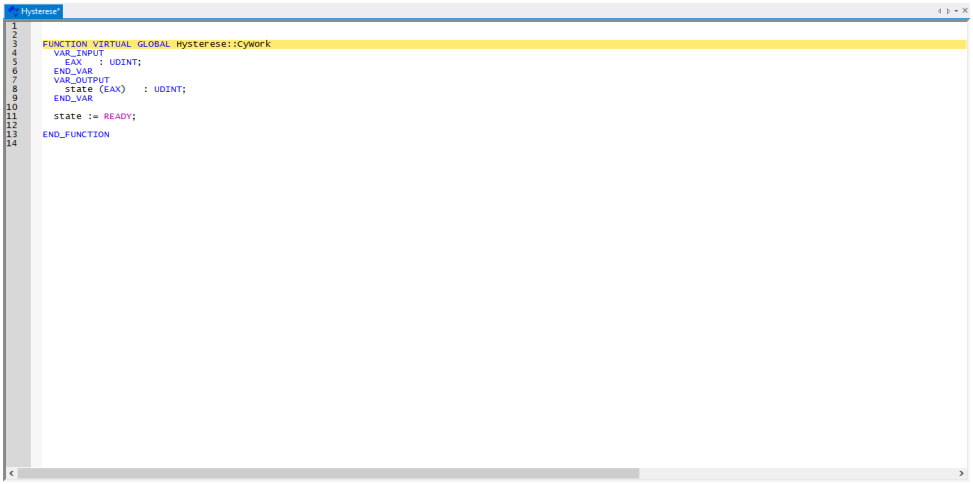
As soon as a client is assigned, additional clients can be created with a right click on the **Clients** folder and selecting **New Client** (right graphic). With the second menu item in the context menu (Sort Clients), the clients can be sorted alphabetically ascending.


Class View
LearningLasal1
External
Hysterese
Servers
Clients
ActualValue
MaxValue
MinValue
Event
Alarm
Methods
Lib
File
Global
Net
Class


Class View
LearnLasal
External
Hysterese
Servers
Clients
New Client
Add Separator
Sort Clients
Event
Alarm
Methods
Lib
File
Global
Net
Class

6.2.5 Program Code Editor

The program code editor is opened through a double click on **CyWork**.



The following code is inserted between the lines END_VAR and state:=READY; of the CyWork method in the program code editor.

```
FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
  VAR_INPUT
    EAX    : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)    : UDINT;
  END_VAR

  state := READY;

END_FUNCTION
```

Programmcode

Insert the following program code in the CyWork method of the Hysterese class.



To avoid typing errors it is recommended, the client names be placed in the editor using the **Drag & Drop** method.



With Ctrl + mouse wheel, the code editor view can be zoomed. This setting applies to all text views (not only for the actual view).



With the F1 function key, the integrated LASAL help – if available – for the selected password / class can be called.

```
FUNCTION VIRTUAL GLOBAL Hysterese::CyWork
  VAR_INPUT
    EAX    : UDINT;
  END_VAR
  VAR_OUTPUT
    state (EAX)    : UDINT;
  END_VAR

  // *** Read the actual value from the peripherie ***
  ActualValue := ActualValue.Read();
  MaxValue    := MaxValue.Read();
  MinValue    := MinValue.Read();


  //**** Control now the Alarm and Pumpcontrol ****
  if (MinValue >= MaxValue) then
    Alarm := 1;
  else
    Alarm := 0;
  end_if;

  if (ActualValue >= MaxValue) then
    Event := 1;
  elsif (ActualValue <= MinValue) then
    Event := 0;
  end_if;

  Event.Write(Event);
  Alarm.Write(Alarm);


  state := READY;

END_FUNCTION
```

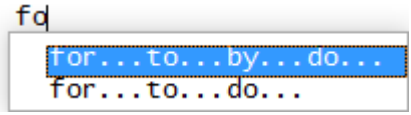


This is a good point to save the project. To accomplish this, click on the  symbol.

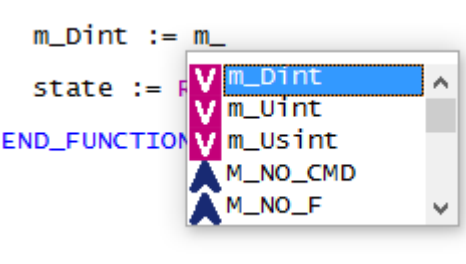
6.2.6 Automatic Code Extension (auto completion)

LASAL contains the automatic code extension function (enabled/disabled under Tools/Option). The affects:

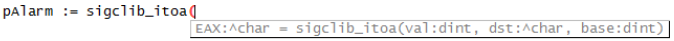
- **Key Words**



- **Constants, Variables, Methods**



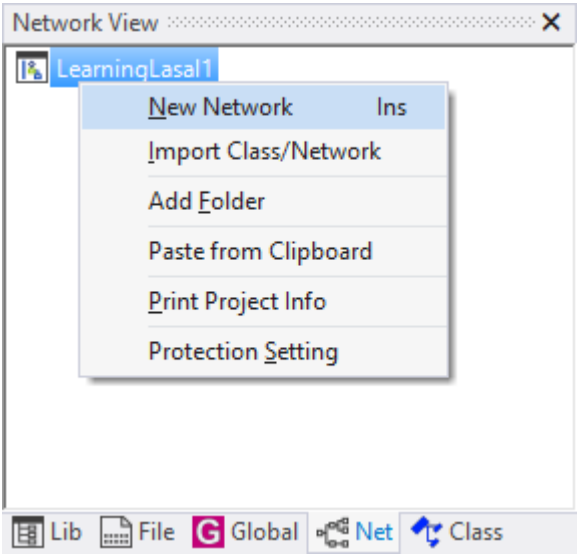
- **Function Parameters**



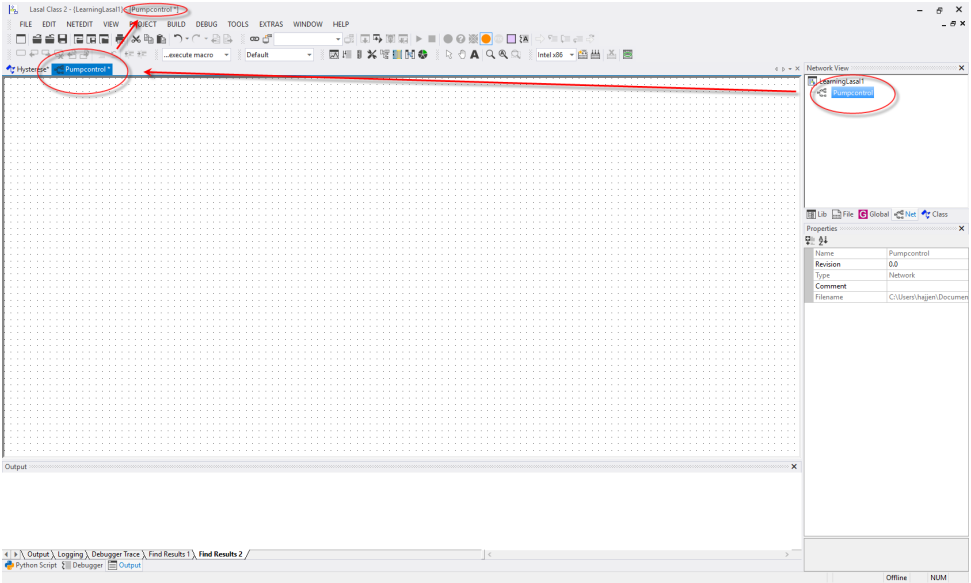
After typing in the first two letters, possible code sections are suggested, from which the programmer can selected the one required.

6.2.7 Object Network

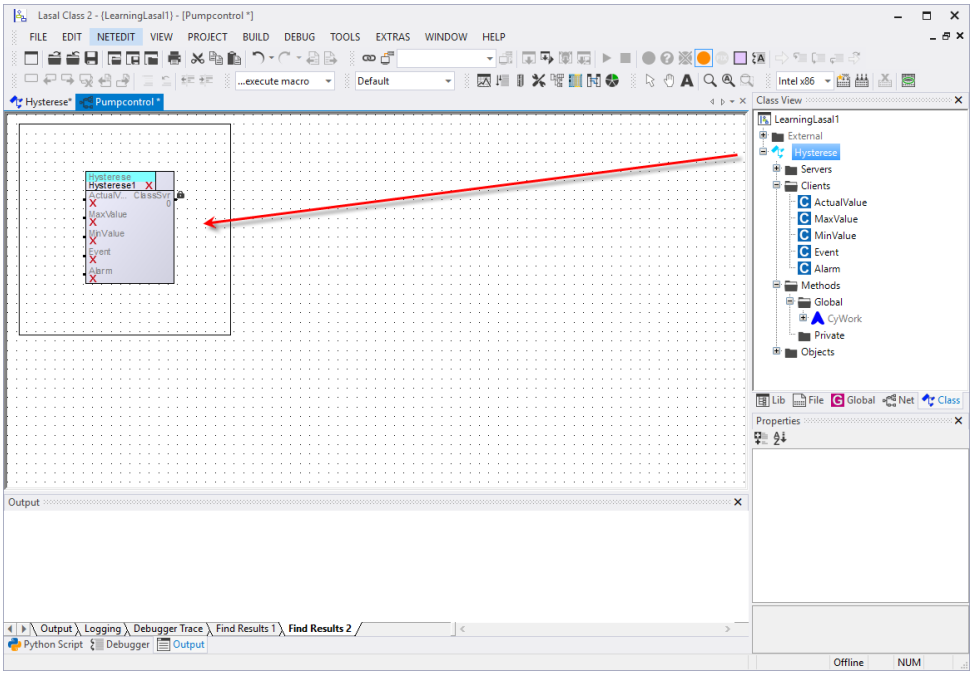
To create an object network, go to the Network View window, open the context menu with a right mouse click and select **New Network**.



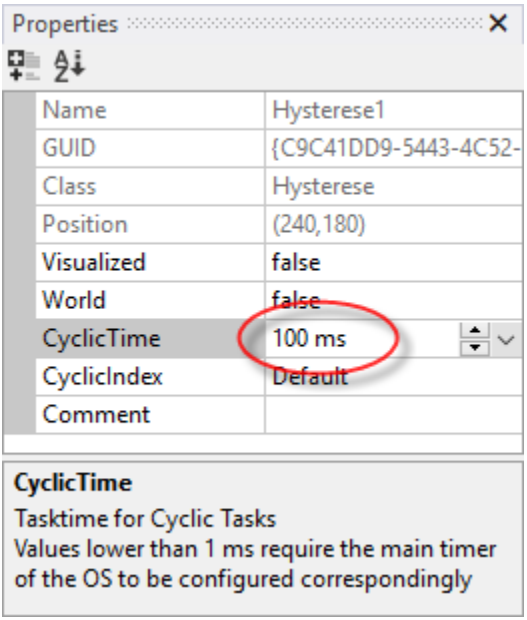
Enter the name (Pumpcontrol) of the network in the open field then double click on the network name **Pumpcontrol** to display the network.



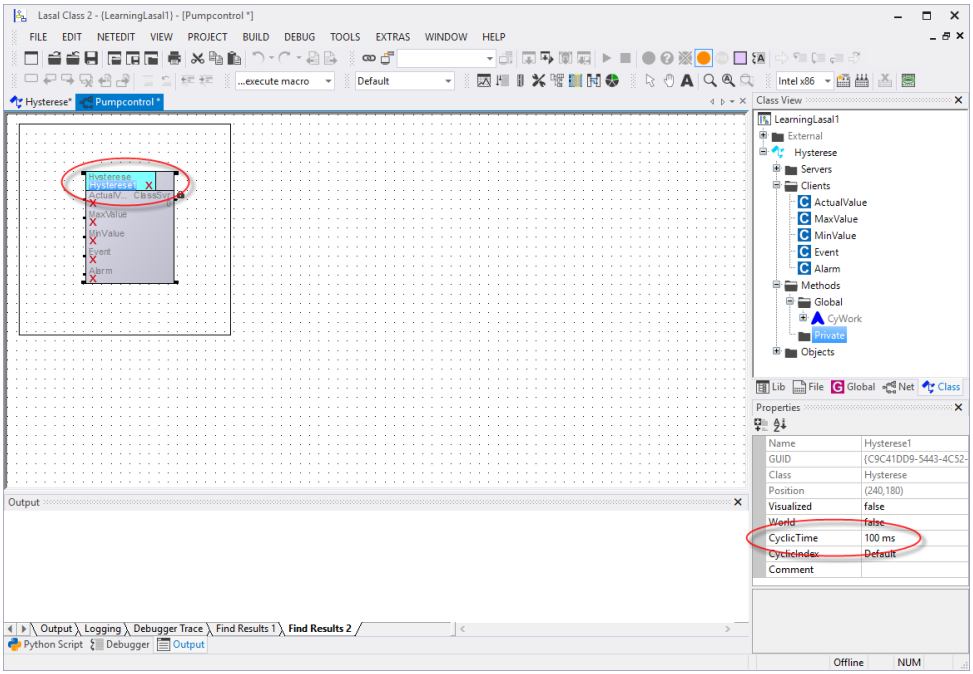
Go to the Class View window to place an object of the class, Hysterese in the Object network (**Pumpcontrol**). Mark the **Hysterese** class with the left mouse button and place it the network using the Drag & Drop method as shown below:



In the Properties window of the object, set the cyclic time to 100. To set the cyclic time, click on the object name (**Hysterese1**) in the object network or the **Class View window** of the project tree.

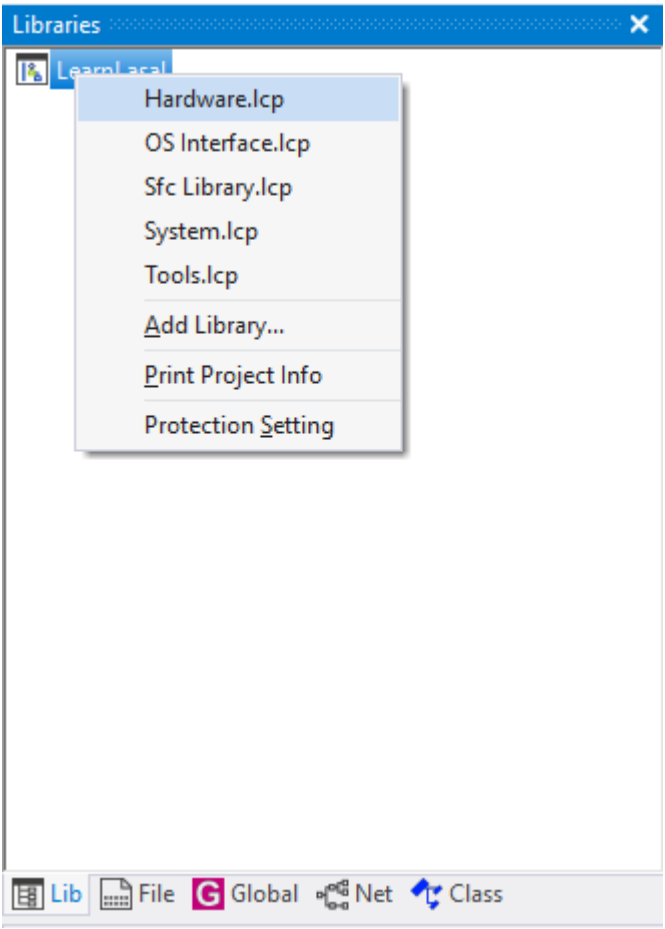


With the cyclic work assignment in the Properties window (**CyclicTime = 100**), the color of the object heading is changed as can be seen in the graphic below. This means, this object is accessed by the operating system every 100ms.

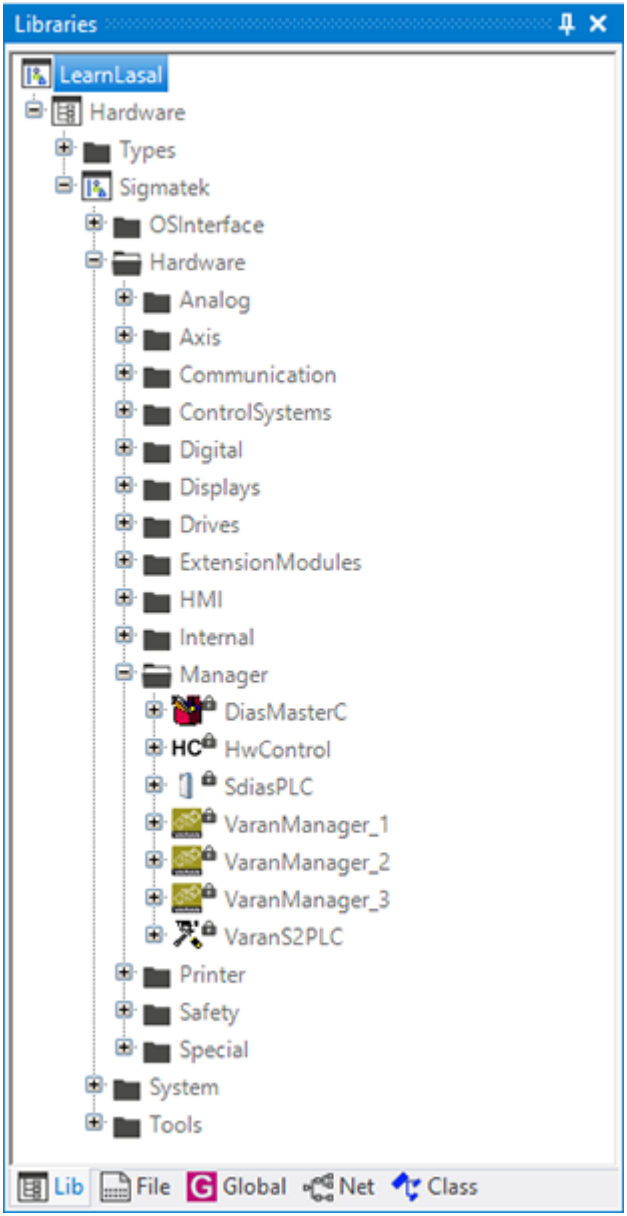

Lasal Class 2 - (LearningLasal1) - [Pumpcontrol *]
FILE EDIT NETEDIT VIEW PROJECT BUILD DEBUG TOOLS EXTRAS WINDOW HELP
Hysterese* Pumpcontrol *
Class View
LearningLasal1
External
Hysterese
Servers
Clients
ActualValue
MaxValue
MinValue
Event
Alarm
Methods
Global
CyWork
Private
Objects
Lib File Global Net Class
Properties
Name Hysterese1
GUID {C9C41DD9-5443-4C52-
Class Hysterese
Position (240,180)
Visualized false
World false
CyclicTime 100 ms
CyclicIndex Default
Comment
Output
Output Logging Debugger Trace Find Results 1 Find Results 2
Python Script Debugger Output
Offline NUM

6.2.8 Library

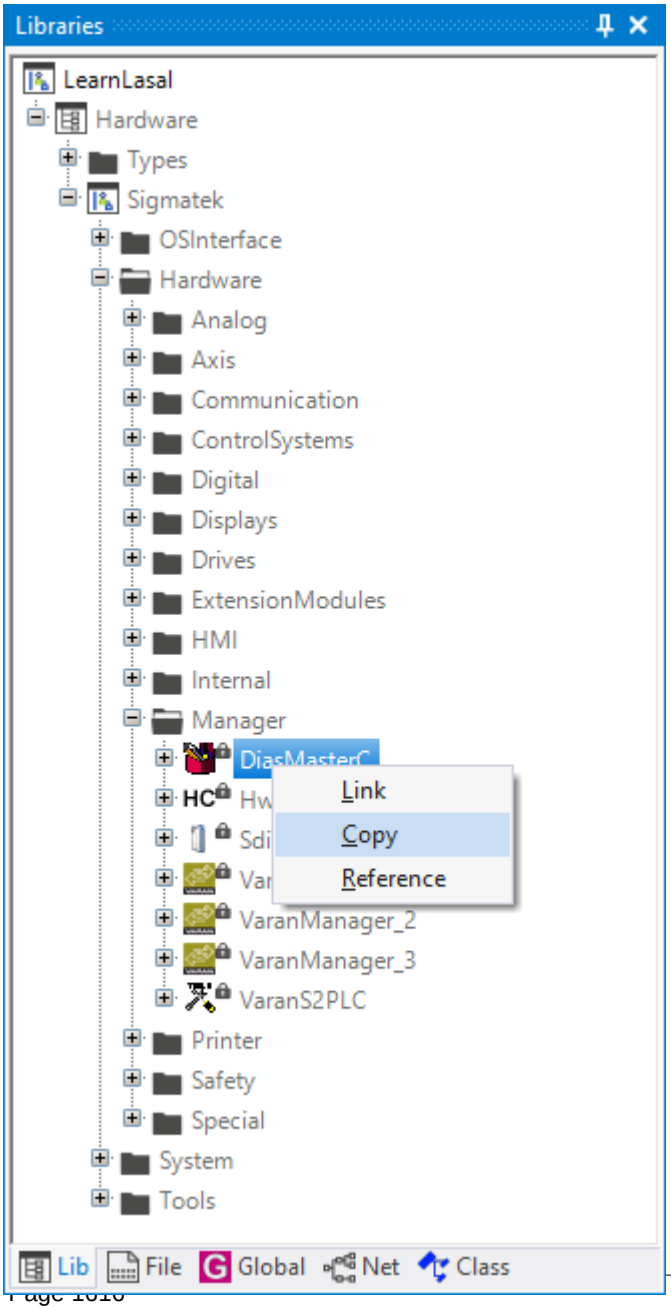
To use a class from the library, change to the Libraries window and open the context menu with a right click on Library and select **Hardware.lcp**.



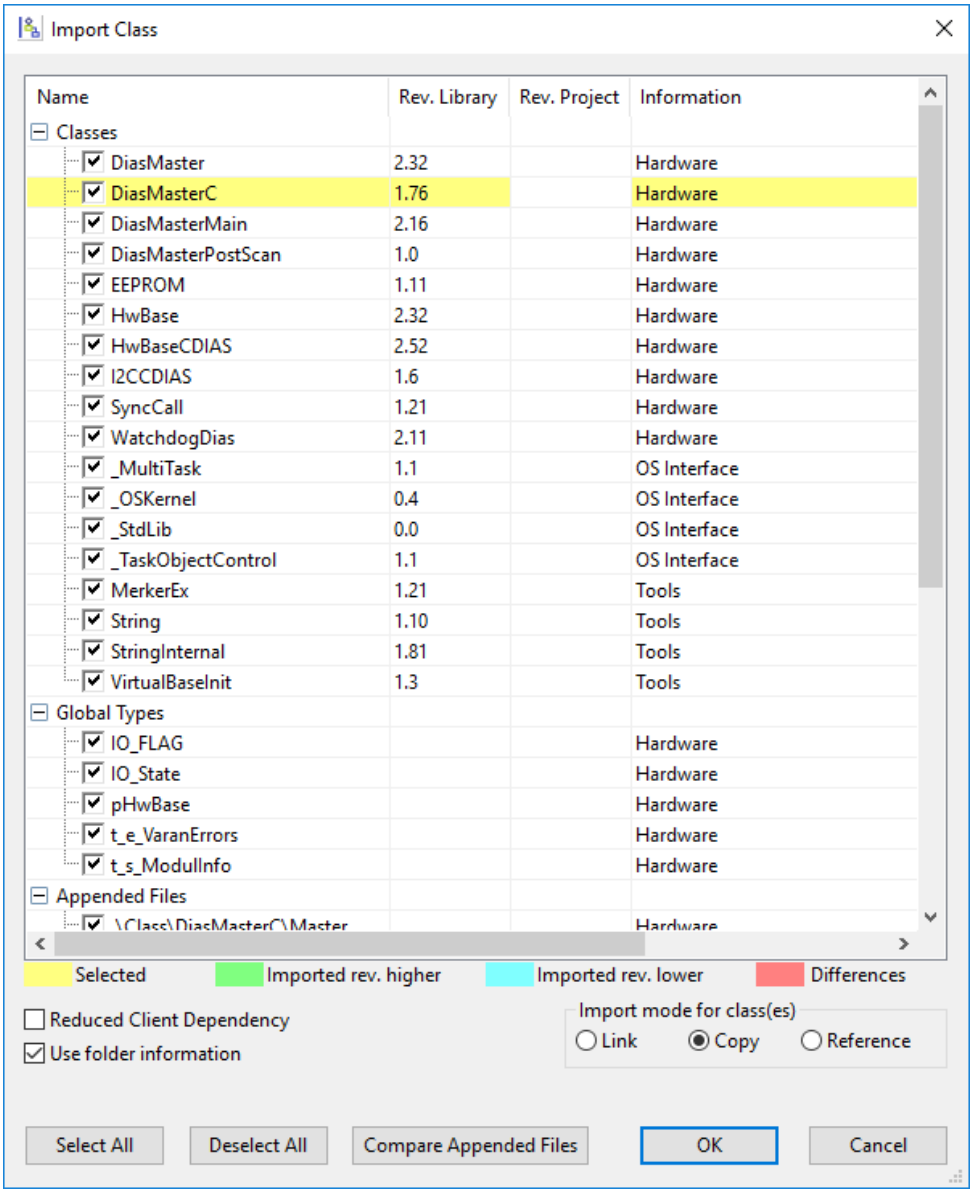
Click on the + symbol next to **Hardware** to display the list of available hardware classes.


Libraries
LearnLasal
Hardware
Types
Sigmatek
OSInterface
Hardware
Analog
Axis
Communication
ControlSystems
Digital
Displays
Drives
ExtensionModules
HMI
Internal
Manager
DiasMasterC
HwControl
SdiasPLC
VaranManager_1
VaranManager_2
VaranManager_3
VaranS2PLC
Printer
Safety
Special
System
Tools
Lib
File
Global
Net
Class

Scroll through the list until **DiasMasterC** is found and open the context menu with a right click and select **Copy**.


Libraries
LearnLasal
Hardware
Types
Sigmatek
OSInterface
Hardware
Analog
Axis
Communication
ControlSystems
Digital
Displays
Drives
ExtensionModules
HMI
Internal
Manager
Link
Copy
Reference
VaranManager_2
VaranManager_3
VaranS2PLC
Printer
Safety
Special
System
Tools
Lib
File
Global
Net
Class

Afterwards in the **Import Dialog (as copy) window** click on **OK**.


Import Class
Name
Rev. Library
Rev. Project
Information
Classes
DiasMaster 2.32 Hardware
DiasMasterC 1.76 Hardware
DiasMasterMain 2.16 Hardware
DiasMasterPostScan 1.0 Hardware
EEPROM 1.11 Hardware
HwBase 2.32 Hardware
HwBaseCDIAS 2.52 Hardware
I2CCDIAS 1.6 Hardware
SyncCall 1.21 Hardware
WatchdogDias 2.11 Hardware
_MultiTask 1.1 OS Interface
_OSKernel 0.4 OS Interface
_StdLib 0.0 OS Interface
_TaskObjectControl 1.1 OS Interface
MerkerEx 1.21 Tools
String 1.10 Tools
StringInternal 1.81 Tools
VirtualBaseInit 1.3 Tools
Global Types
IO_FLAG Hardware
IO_State Hardware
pHwBase Hardware
t_e_VaranErrors Hardware
t_s_ModulInfo Hardware
Appended Files
Selected
Imported rev. higher
Imported rev. lower
Differences
Reduced Client Dependency
Use folder information
Import mode for class(es)
Link
Copy
Reference
Select All
Deselect All
Compare Appended Files
OK
Cancel

6.2.9 Hardware Editor

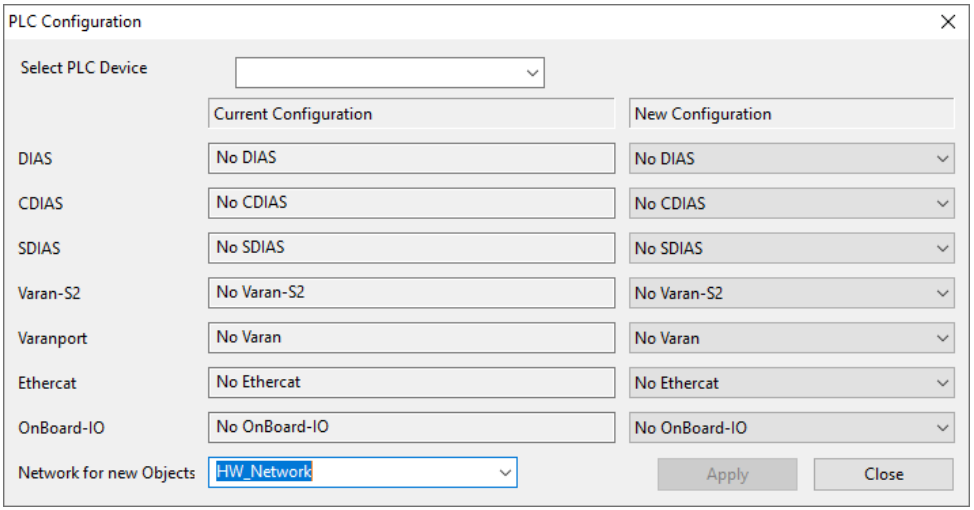
To import several hardware modules (in this example, **CTO 163** and **CAM 123**), the process must be repeated for each one. In LASAL however, there is a significantly faster way to configure hardware – the Hardware Editor. To achieve the same result with this tool, only a few mouse clicks are required. Proceed as follows:

First remove the **DiasMasterC** class and the primary **Manager** folder from your project. Next, go to the **Lib** register card and remove the complete **hardware library**.

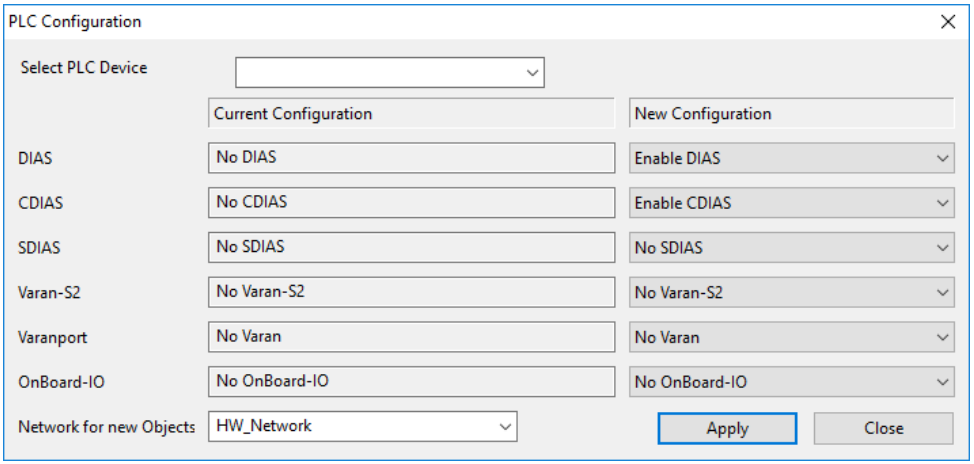
The hardware editor can now be started. To open the editor, click on the  icon in the toolbar. The hardware tree now appears:



With a right click and **PLC Configuration** or a double click on PLC the configuration dialog opens.

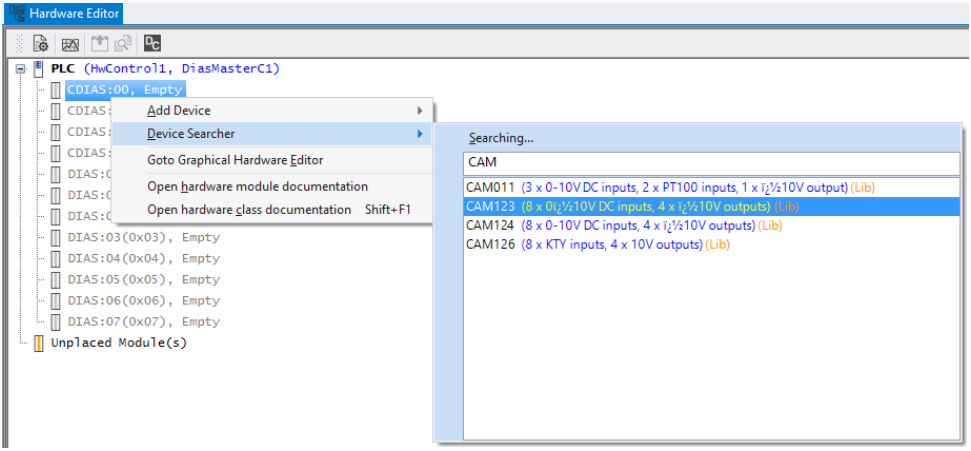


In the highlighted fields, the network in which new hardware objects are automatically placed can be entered. Here, switch HW_Network to Pumpcontrol. Instead of **No DIAS/No CDIAS**, select **Enable DIAS/Enable CDIAS**. The Apply button is can now be clicked. Click the Apply button or press the Enter key.

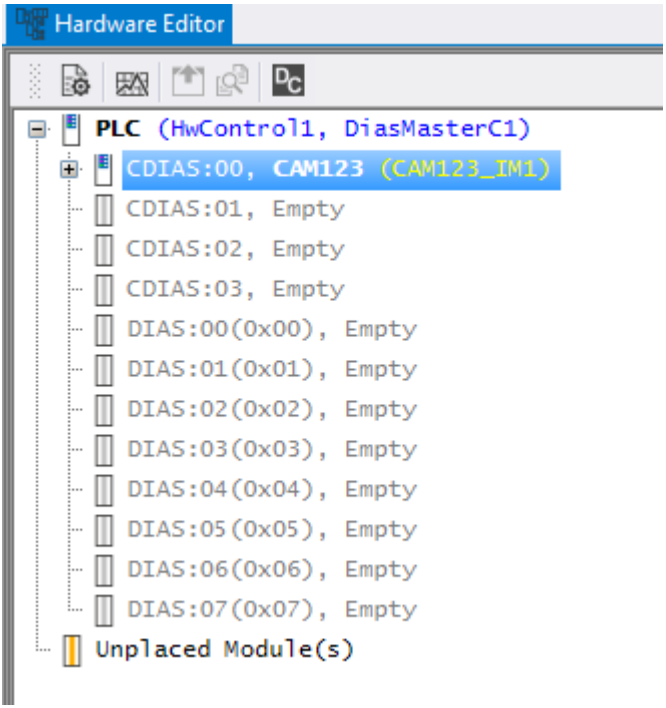


The hardware tree now appears with the configured **PLC**. Click on the CDIAS:0 interface with the right mouse button, select **Device Searcher**, click on empty field in the window

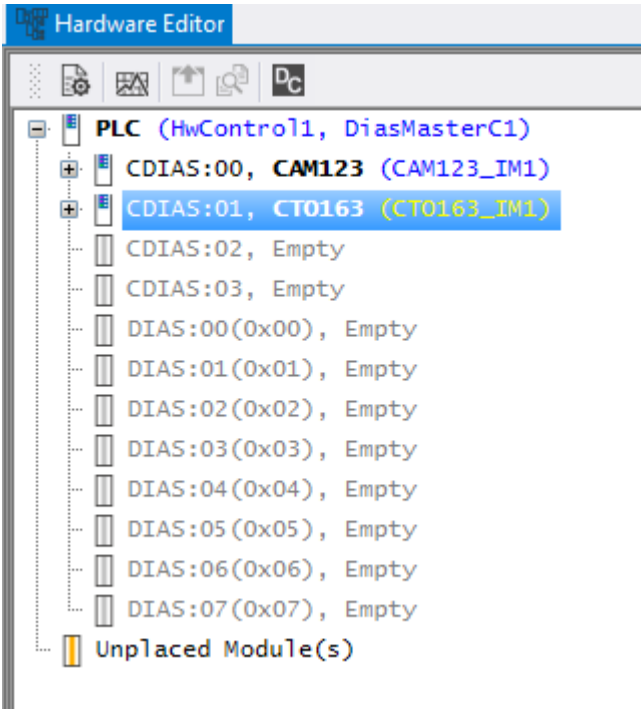
that opens and enter the first letter of the module – in this example, it is **CAM 123**. Next, select CAM 123 and press **Enter**.



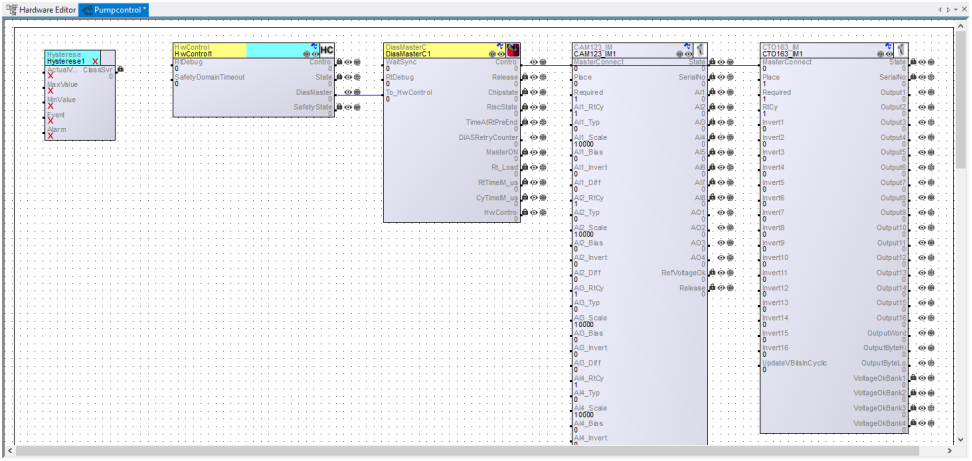
The added module is now visible in the hardware tree.



Repeat the last steps for the **CTO 163 module**. The hardware tree should now appear as follows:



Click on the network view for Pumpcontrol. As can be seen, it is more practical to configure the hardware using the hardware editor.

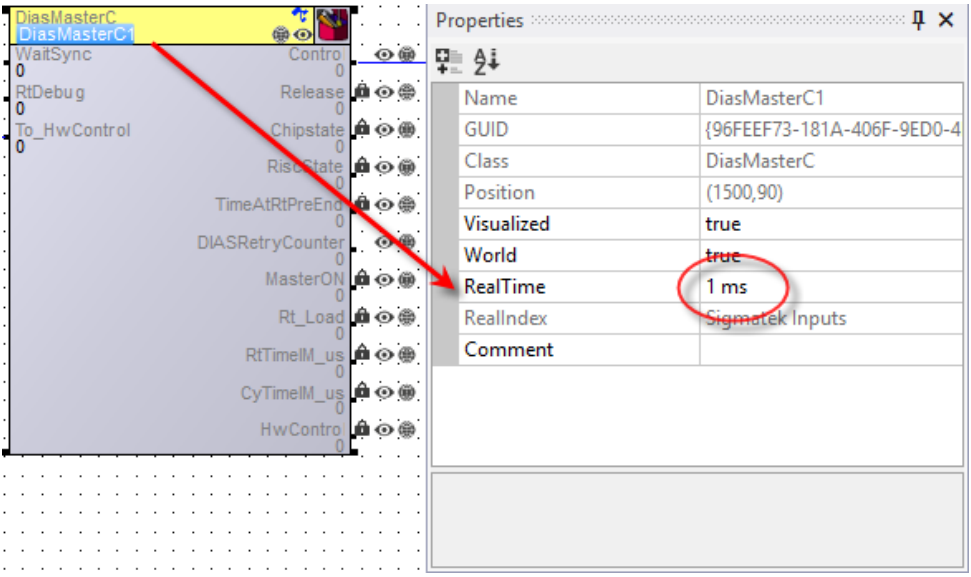


For a comprehensive description, see the chapter **Hardware Editor**.

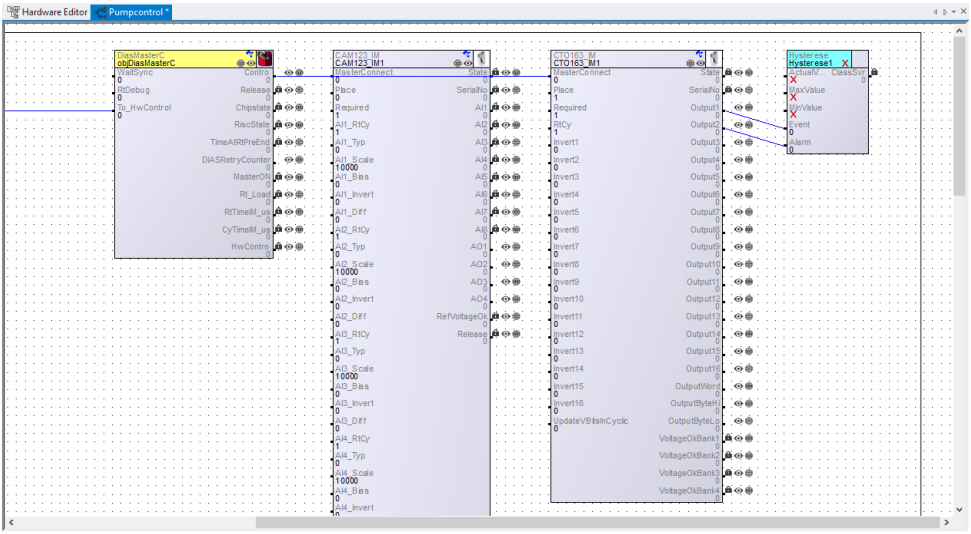
Now click on the DiasMasterC1 object. In the properties window, the object properties can be seen.

Define the **RealTime** value for the **DiasMasterC** in the **Properties window**.

The **RealTime setting** is either 2ms or (386 CPUs) 1ms (IPCs CIPCs, CCL911…) according to the CPU employed.



Rename the **DiasMasterC** object to **objDiasMasterC** now. Connect the objects in the network as shown in the graphic below.

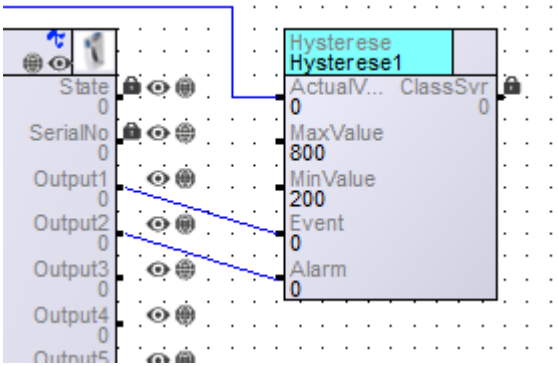


For each **client value** in the above illustration crossed out in **red**, an initial value must be entered.

With a double click on each of the **red crossed-out client values**, a white field appears in which the **initial value** is entered. Press enter to confirm the value and close the field.

A network object, in which at least one client in not initialized, is also marked with a **red X** in the heading (see Hysteresis in screen shot). If the last clients are also initialized and connected to a server, the red X disappears from the object heading.

Next, enter an initial value in the object (**objHysterese**) of the Hysterese class (**MaxValue = 800, MinValue = 200**).



As shown below, these values can also be defined in the **Properties** window.

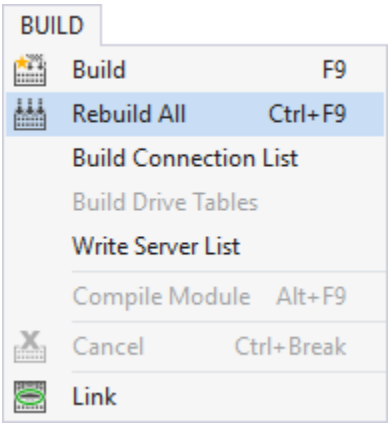


This is a good point to save the project again.

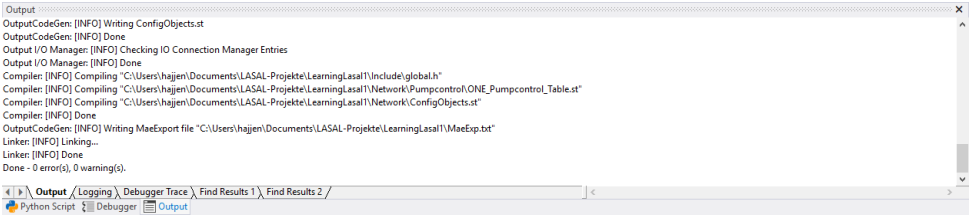
If the project has been saved, open **Build** in the menu bar and select **Rebuild All** from the context menu to download the changes and display any possible errors.



The **Rebuild All** instruction can also be executed using the shortcut Ctrl + F9.



The Compilation process can be observed in the Output window. After the compilation process, 0 errors and 0 warnings displayed in the output window.



To download the project to the control, the online settings must have been made. These settings can be seen under the heading Online Settings.

The first project is now successfully completed.

7 APPENDIX

7.1 File Types in LASAL Class 2

7.1.1 Project File *.lcp

The project file is a list of project component elements:

- Classes and other sources
- Networks
- Object files

Project specific settings (e.g. online connection, compiler settings, etc.) are also saved here.

The format for project files is XML.

7.1.2 Source Code Files

7.1.2.1 *.ST

The preferred programming language of LASAL is structured text (ST). The definition of this language is covered under the IEC 61131-standard. Object oriented language elements were added for LASAL. The source code from LASAL classes are usually stored in files with the *.ST extension. SIGMATEK programming languages (ST, IL) can be combined within an ST file. The functions in an ST file must be programmed in Structured Text when no other language is specified.

7.1.2.2 *.AWL

Instruction List (Sprachanweisungsliste) programming language can be used in files with the *.AWL extension. When no other is specified, functions in AWL files must be written using Instruction List. In this file, ST and IL can be combined.

7.1.2.3 *.IPR

Files with the *.IPR extension are used to store interpreter code. The Interpreter code can be used to define the run sequence of a machine control system. IPR is a LASAL specific program language.

7.1.2.4 *.H

The user can link any amount of header files in a project. This is done using the Add File... instruction in the context menu of a class. The setting is then made to define the header file as global or local to the class in which it was linked.

If the **Global property is true**, the Include flag can **no longer be set to true** (a warning appears):

OutputProject: [WARN] Class 'Class0' includes file "D:\Projekte\TEst Global-Include-Flags\Class\Class0\myHeader.h", but is not allowed with global flag enabled.

If the file was added to the file tree with **Add File**, **Global can only be set to true** as long as the file was **not locally included in another class (Global = false) and the Include flag was set to true**. Such an attempt is also canceled with a warning:

OutputCommand: [WARN] Set Global Flag Global flag true is not allowed with include is true.

If the file is **added to multiple classes** and the global flags are set differently, **Global always has priority**. As soon as **Global = true** in one of the classes, the file is included globally.

When the header file is linked to a class, it is also transferred with the class if it is copied as long as it is in the project directory.

7.1.2.5 *.C, *.CPP

LASAL offers the possibility to program functions in C. the support for C is comprised of a compiler and supplies the outputs in COFF formant. A list of the compatible compilers is available at SIGMATEK.

7.1.2.6 *.lcn

In this file, the object network information is stored (ON). The user changes this file in the network editor. The format for the network file is XML.

7.1.3 Project Administration *.py

Commands in the form of Python scripts can be relayed to LASAL CLASS 2 using PY files. Py files are designed for the creation of complex projects from individual projects.

7.1.4 Temporary Files

7.1.4.1 *.lcb

LCB files contain binary information so that ASCII files don't always have to be used, therefore accelerating work with LASAL 2. LCB files exist for classes, networks and the project.

7.1.4.2 *.lob

LASAL object files are created as the output files of a compiler process using an ST or IL file. They contain the code and debug information.

7.1.4.3 *.lba

Like *.lob, only for an ARM-CPU.

7.1.4.4 *.obj

Output file for a C (C++) compiler. LASAL converts OBJ files to LOB files. These files must correspond to the COFF format.

7.1.4.5 *.ldi

This file contains the global debug information for debugging a project.

7.1.4.6 *.bin

Files of this type are located in the Project Internal and contain temporary binary information.

7.1.5 Miscellaneous Files

7.1.5.1 *.vov

This file contains a Visual Object View – a graphic view created in LSE. The file can be edited (the screen is opened for editing in LSE when the file in the Files sub folder of the class is double clicked) or displayed (The LSE kernel opens and the screen is displayed when it is selected from the object context menu in the object network.

7.1.5.2 *.lda

*.lda files (previously *.xml) are Dias drive files that contain drive parameters. With a double click on an *lda file in the file tree (**Dias Drive files** folder), the corresponding list view opens in LASAL.

7.1.5.3 *.lce

*.lce files are **connection file editor files**. By double clicking on such a file in the file tree (**connection files folder**), the corresponding tab in the lower area (Connection Manager => Project Connection Files) opens and the corresponding connections are displayed.

7.1.5.4 *.lcf

*.lcf files are **internal connection file editor files** that are created when an ***.lce files** are loaded into the control.

7.1.5.5 *.lcc

*.lcc files are configuration files from the data analyzer.

7.1.5.6 *.lds

*.lds files are exported Scope files from the data analyzer. They can be imported and exported with the appropriate menu items.

7.1.5.7 *.ldc

*.ldc files are exported **channel files** from the **Data Analyzer**. They can be imported and exported with the appropriate menu items.

7.1.5.8 *.csv

*.csv files are comma separated value files, e.g. exported as such from the **Data Analyzer**.

7.1.5.9 *.txt

*.txt files are exported **visualized server files**.

7.1.5.10 *.etf

*.etf files are exported **EasyCase files**.

7.1.5.11 *.lcz

*.lcz files are **exported classes / networks**. (performed in the context menu of the class or network **Export Class(es)** or **Export Network(s)**).

7.1.5.12 *.luc

The **L**ASAL **U**nit **C**onfiguration file is the global, company-specific unit file. Here, the user-defined project units are stored. For further information, see the chapter Unit Configuration.

7.1.5.13 *.mme

*.mme files are server data exported from the MultiMaster.

7.1.5.14 *.mmc

*.mmc files are MultiMaster configuration data saved directly in the project (in the directory Network).

7.1.5.15 *.xml

Numerous *.xml files are available in LASAL. The stored **IO connection manager files**, or the **Matlab / Simulink files** that can be imported and exported with the appropriate menu items, are available in this format. *.xml is also the extension for old Drive files.

7.1.6 Git-instructions for LASAL Tools

Introduction
Describing the incompatibility
How to fix the problem

7.1.6.1 Introduction

A very common tool for development teams is the use of a VCS (Version Control System) Tool.

Link (EN): https://en.wikipedia.org/wiki/Version_control

Link (DE): https://de.wikipedia.org/wiki/Versionsverwaltung

The most common tools in this area are Git, Subversion, Azure DevOps and Perforce. You can find a good comparison here: https://en.wikipedia.org/wiki/Comparison_of_version-control_software

This chapter focuses on the **Git VCS** because a peculiarity of its internal architecture **can cause incompatibilities with Lasal tools**.

The roots of Git is the **Linux Kernel**, which shares the same developer and thus text files uses simple **LF (Line-Feed)** characters. On the other side, **Windows** uses the same line endings used on legacy text printers: a sequence of **CR (Carriage Return) and LF (Line-Feed)**.

The most important prerequisite for a Git server to correctly recognize changes in source files is the **use of text files with simple LF - endings**. This is required even if no projects are developed on UNIX / Linux systems.

Because of this, when a file is sent from a Windows computer to a Git server, files are spontaneously **converted from CR+LF to LF line endings**. The opposite occurs when a file is sent from this server to a Windows computer - on the latter **the LF 's are converted to CR+LF**.

It should be noted that **not every file may be converted blindly**. Performing such conversion of e.g. a **binary file** destroys its contents, **makes it unusable and probably unrecoverable**.

Therefore, files such as images, databases, MSWord documents, archives, executable files and many other types should not be converted in order to preserve the integrity of the file.

So when transferring files, Git needs to know the type of the file. Three methods have been established for this purpose:

- The project repository and working copy may **contain a list of file types and contents**. This is an optional file/configuration that is not present in a new repository
- The Git client installer adds a file that contains a list of known file extensions and the content type. It should be noted here that this list is not exhaustive. This list can be stored e.g. in C:\Program Files\Git\etc\gitattributes, but this file should not be edited!
- An automatic detection program that attempts to guess the contents of the file is run if the previous methods do not produce a satisfactory result. This is a conservative method that **prefers a false positive to a false negative**, since converting a binary file is really unacceptable

7.1.6.2 Describing the incompatibility

Incompatibilities occur with LasalClass2 projects that contain files with file extensions unknown to Git, and also cannot be recognized by the third method - described on the introduction page.

Link: https://www.sigmatek-automation.com/fileadmin/user_upload/downloads/gitattributes_eng.zip

In this case LASAL CLASS 2 receives an unconverted text file. Since LASAL CLASS 2 **V02.02.206** the tool detects this inconsistency and updates the file to the Windows format.

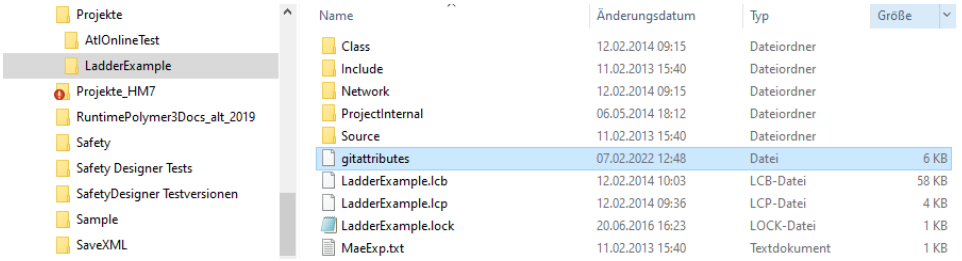
However, some inconveniences may still occur:

- Unnecessary rebuilds
- Files are recognized as modified without visible changes (files are different at the line-ends)
- Incorrect line alignment when debugging code (executed and displayed lines of code do not match)

7.1.6.3 How to fix the problem

For each Lasal Class 2 project, add a .gitattributes file with the file extensions used by SigmaTek.

We have provided a ready-made version of this file that can be downloaded from https://www.sigmatek-automation.com/fileadmin/user_upload/downloads/gitattributes_eng.zip. This file should be copied to the root directory of the project so that all developers have a copy of the file.



Alternative: Installing the file for a user account

It is possible to install this file for a user account, and all the user's projects will immediately benefit from this setting. For this option, copy the file to the user's .gitattributes root directory and run the following line at the command prompt:

```
c:\Program Files\Git\bin\git.exe" config core.attributesFile %HOMEDRIVE%%HOMEPATH%\.gitattributes
```



Tip

Note that while this option seems to be the best, it is unfortunately not optimal for a mixed team of developers. However, if your organization has an automated domain server, IT staff can automate this setting for each developer.

7.2 Number Systems

When creating a program, various number systems are used. The reason for using different number systems is partly due to the process by which the computer searches (logic 0 or 1) and also for the clarity and understandability of the program. The number systems relevant to the software design are the decimal, hexadecimal and binary systems.

7.2.1 Decimal Number System

The decimal number system is the widest and most commonly known system. In order to make the other number systems easier to understand, the basics of the decimal number system must be reviewed. The decimal number system has 10 different characters with which different number values are represented. These number characters begin with 0.

In the decimal number system, the numbers 0 to 9 can be represented. This scale, however, is not large enough for the required range of numbers. For this reason the values are cascaded. Since the number 5 can also be written as 05, after the number 9 is reached, the 0 is incremented to 1 and the value becomes 10. In higher cascades, when the character set is exhausted, the next cascade is incremented and the previous ones restart at 0.

E.g.: 00999 + 1 = 01000.

The Value of a cascade is raised by an exponent of 10 from right to left starting at 0.

Nominal value:	0 1 2 3 4 5 6 7 8 9
Basis:	10
Highest nominal value:	9
Local value:	$10^0 = 1$, $10^1 = 10$, $10^2 = 100$, and so on

	10^3	10^2	10^1	10^0			
Dec	1	2	3	7	=	**Dec**	1237

7.2.2 Binary Number System

Since the CPU and digital I/Os operate with logic conditions (0 and 1), the binary system is the actual number system used by the computer. The binary number system consists of 0 and 1. The calculation process is the same as with the decimal system, the difference being that only the characters 0 and 1 are utilized. The disadvantage of this system is that

larger numbers quickly become long and complex. The advantage, however, is that the individual bits in binary variables can be easily identified as High (1) or Low (0).

The Cascade is also carried out in the binary number system. The difference to the decimal system is that the cascade through the reduced number characters also has a reduced value.

Nominal value: 0 and 1
Basis: 2
Highest nominal value: 1
Local value: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, and so on

	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	1024	512	256	128	64	32	16	8	4	2	1	
Bin	1	0	0	1	1	0	1	0	1	0	1	= **Dec** 1237
	1024	+		128	+ 64	+	16	+	4	+	1	= 1237

7.2.3 Hexadecimal Number System

The hexadecimal number system is favored most for the creation of programs because it combines the properties of both previously mentioned number systems. Large binary numbers have the disadvantage of being complex. As a remedy the **hexadecimal** system was created. In this system, 4 bits of a binary number are replaced with a hexadecimal character. Since a 4 bit binary number represents 16 conditions and we only know 10 decimal characters, 6 letters were added in the hexadecimal system.

The hexadecimal number system consists of 16 characters. The known number characters were expanded with letter characters.

Nominal value: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Basis: 16
Highest nominal value: F
Local value: $16^0 = 1$, $16^1 = 16$, $16^2 = 256$, and so on

	16^2	16^1	16^0		
Hex	4	D	5	=	**Dec** 1237

Because the cascaded weighting is part of the binary system, it's easy convert back to the binary number system.

Each hexadecimal place contains 4 binary places:

	4	D	5	=	Dec	1237
Bin	0100	1101	0101			

7.3 The MathBSD Library

7.3.1 Properties

Mathematical functions are an essential part of the ST programming language. However, if the application is designed for high accuracy and/or speed or needs functions that are not available in the ST language, it may make sense to include the MathBSD library.

Using the MathBSD library offers:

- ...more accurate results,
- ...in most cases a significantly faster performance,
- ...a considerably extended range of functions compared to the ST standard.

A distinction must be made here between **ARM** and **Intel** processors. **ARM** processors only use **software** (libraries) to perform mathematical calculation functions. When including the MathBSD library, the performance increases noticeably almost everywhere. **Intel** processors, on the other hand, use both the **FPU** (**floating point co-processor**, i.e. **hardware**) and software libraries. Calculations in the hardware are generally faster than in the software; **in some cases**, however, the FPU delivers **incorrect results**. In particular, it affects **trigonometric functions** when they are called for values beyond PI. Therefore, especially with Intel processors, when **high accuracy** and **correct results** are required, the MathBSD **library should be included**. However, this may **slow down** the **calculations on an Intel processor**!



Tip

Since Loader version 02.02.239, the following ANSI C math functions can also be used:

- acosh
- acoshf
- asinh
- asinhf
- atan2
- atan2f
- atanh
- atanhf

- ceil
- ceilf
- cosh
- coshf
- exp2
- exp2f
- expm1
- expm1f
- floor
- floorf
- fmod
- fmodf
- hypot
- hypotf
- log1p
- log1pf
- pow
- remainder
- remainderf
- remquo
- remquof
- round
- roundf
- scalbn
- scalbnf
- sincos
- sincosf
- sinh
- sinhf
- tanh

- tanhf
- trunc
- truncf

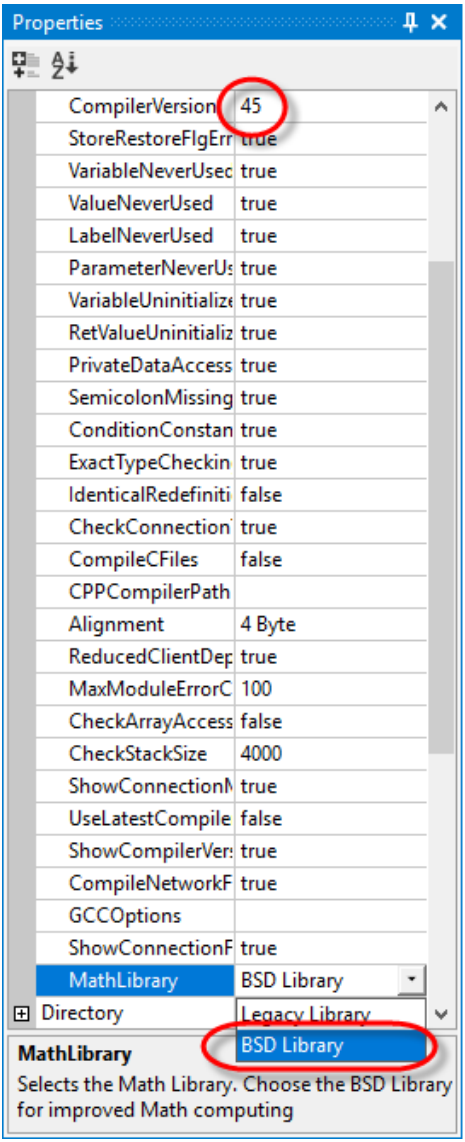
For more detailed information on the individual functions, see the following link: https://en.cppreference.com/w/c/numeric/math

7.3.2 Usage / Activation in LASAL Class 2

The library can be used in LASAL CLASS 2 **from version 02.02.177** and **from compiler version 45**. The extended functions of the MathBSD library can be used at any time using `#include "MathBSD.h".`

To automatically replace the ST standard functions with faster / more accurate ones from the MathBSD library, **set MathLibrary** to **BSD Library** in the **project properties** under **Compiler**. This is done automatically for new projects.

For existing projects, the **CompilerVersion** setting must be set to **45** or greater and the **MathLibrary** setting to **BSD Library**:


Properties
CompilerVersion 45
StoreRestoreFlgErr true
VariableNeverUsed true
ValueNeverUsed true
LabelNeverUsed true
ParameterNeverUs true
VariableUninitialize true
RetValueUninitializ true
PrivateDataAccess true
SemicolonMissing true
ConditionConstan true
ExactTypeCheckin true
IdenticalRedefiniti false
CheckConnection true
CompileCFiles false
CPPCompilerPath
Alignment 4 Byte
ReducedClientDep true
MaxModuleErrorC 100
CheckArrayAccess false
CheckStackSize 4000
ShowConnectionM true
UseLatestCompile false
ShowCompilerVer true
CompileNetworkF true
GCCOptions
ShowConnectionF true
MathLibrary BSD Library
Directory Legacy Library
BSD Library
MathLibrary
Selects the Math Library. Choose the BSD Library for improved Math computing

Caution!

Only from compiler version 45 there is the item MathLibrary under the project properties!

To prevent possible linker conflicts, the MathBSD functions are preceded by the bsd_ prefix.

7.3.3 Function Reference

7.3.3.1 ACOS

Calculates the arc cosine of the transferred value.

Name of the function in Structured Text: ACOS()

Name of the function in the MathBSD library: **bsd_acos()** for LREAL and **bsd_acosf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The value whose arc cosine is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The arc cosine of arg (a value between 0 and π in radians)
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 0.456789;
  dblY := ACOS(dblX);  // 1.0964140760883754

  // REAL arguments use single precision
  sngX := 0.456789;
  sngY := ACOS(sngX);  // 1.0964141
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/acos

7.3.3.2 ACOSH

Calculates the area cosine hyperbolicus of the given value.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_acosh()** for LREAL and **bsd_acoshf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The value whose area cosine hyperbolicus is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The area cosine hyperbolicus of arg (a value between 0 and ∞ in radians)
If failed (transfer parameter less than 1)		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 1.456789;
  dblY := bsd_acosh(dblX);  // 0.9227277912198553
  sngX := 1.456789;
  sngY := bsd_acoshf(sngX); // 0.9227278
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/acosh

7.3.3.3 ASIN

Calculates the arc sine of the transferred value.

Namo of the function in Structured Text: ASIN()

Name of the function in the MathBSD library: **bsd_asin()** for LREAL and **bsd_asinf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The value whose arc sine is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The arc sine of the transferred value (- 0.5 * π to 0.5 * π in radians)
If failed (transfer value is not between -1.0 and +1.0)		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

// LREAL arguments use double precision
  dblX := 0.456789;
  dblY := ASIN(dblX); // 0.4743822507065214

  // REAL arguments use single precision
  sngX := 0.456789;
  sngY := ASIN(sngX); // 0.4743823
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/asin

7.3.3.4 ASINH

Calculates the area sine hyperbolicus of the transferred value.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_asinh()** for LREAL and **bsd_asinhf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The value whose area sine hyperbolicus is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The area sine hyperbolicus of the transferred value
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
END_VAR

  dblX := 1.456789;
  dblY := bsd_asinh(dblX);  // 1.1705525649620534

  sngX := 1.456789;
  sngY := bsd_asinhf(sngX); // 1.1705526
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/asinh

7.3.3.5 ATAN

Calculates the arc tangent of the transferred value.

Namo of the function in Structured Text: ATAN()

Name of the function in the MathBSD library: **bsd_atan()** for LREAL and **bsd_atanf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The value whose arc tangent is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The arc tangent of the transferred value (- 0.5 * π to 0.5 * π in radians)
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := ATAN(dblX); // 0.9692283040505609

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := ATAN(sngX); // 0.9692283
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/atan

7.3.3.6 ATAN2

Calculates the arc tangent of the transferred division. Instead of a number like ATAN(), two values are passed here with which a division is performed. The background is that the correct quadrant in the coordinate system can be determined by the sign of the two numbers passed.

Namo of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_atan2()** for LREAL and **bsd_atan2f()** for REAL numbers.

Transfer parameters	Type	Description

argy	REAL or LREAL	The dividend used to calculate the division (argy / argx) for the arc tangent
argx	REAL or LREAL	The divisor used to calculate the division (argy / argx) for the arc tangent
Return parameters	**Type**	**Description**
If successful		
		The arc tangent of the result of the transferred division (-π to π in radians, since all four quadrants of the coordinate system can now be used)
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 1.456789;
  dblY := bsd_atan2(1.75, dblX);  // 0.8765790823018738

  sngX := 1.456789;
  sngY := bsd_atan2f(1.75, sngX); // 0.8765791
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/atan2

7.3.3.7 ATANH

Calculates the area tangent hyperbolicus of the transferred value.

Namo of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_atanh()** for LREAL and **bsd_atanhf()** for REAL numbers.

Transfer parameters	**Type**	**Description**

arg	REAL or LREAL	The value whose area tangent hyperbolicus is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The area tangent hyperbolicus of the transferred value
If failed (arg is not greater than -1 and not less than +1)		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 0.456789;
  dblY := bsd_atanh(dblX);  // 0.493246075590493

  sngX := 0.456789;
  sngY := bsd_atanhf(sngX); // 0.4932461
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/atanh

7.3.3.8 CEIL

Calculates the smallest integer, which is not smaller than the transferred value. Note: the set rounding mode is irrelevant!

Namo of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_ceil()** for LREAL and **bsd_ceilf()** for REAL numbers.

Transfer parameters	**Type**	**Description**
arg	REAL or LREAL	The value that - if decimal places exist - is to be rounded to the next higher integer
Return parameters	**Type**	**Description**

If successful		
		The transferred value itself (if it does not contain any decimal places), or the next higher integer
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
    VAR
        dblY : LREAL;
        sngY : REAL;
    END_VAR

    dblY := bsd_ceil(1.1);      // 2.0
    dblY := bsd_ceil(0.9);      // 1.0
    dblY := bsd_ceil(0.1);      // 1.0
    dblY := bsd_ceil(-0.1);     // 0.0
    dblY := bsd_ceil(-0.9);     // 0.0
    dblY := bsd_ceil(-1.1);     // -1.0

    sngY := bsd_ceilf(1.1);     // 2.0
    sngY := bsd_ceilf(0.9);     // 1.0
    sngY := bsd_ceilf(0.1);     // 1.0
    sngY := bsd_ceilf(-0.1);    // 0.0
    sngY := bsd_ceilf(-0.9);    // 0.0
    sngY := bsd_ceilf(-1.1);    // -1.0
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/ceil

7.3.3.9 COS

Calculates the cosine of the transferred value. Please note: The "Legacy Library" for x86 processors uses the FPU and is therefore much faster. Because the FPU uses a lower resolution for the ð value, the accuracy for |x| > 2 * ð is not acceptable.

Therefore, the bsd_cos() and bsd_cosf() also use the CPU in x86.

Name of the function in Structured Text: COS()

Name of the function in the MathBSD library: **bsd_cos()** for LREAL and **bsd_cosf()** for REAL numbers.

Transfer parameters	Type	Description

arg	REAL or LREAL	The value whose cosine is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The cosine of the transferred value (between -1.0 and +1.0)
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := COS(dblX); // 0.1137605156354449

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := COS(sngX); // 0.1137605
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/cos

7.3.3.10 EXP

Calculates the exponential value of the value transferred (e ^ arg).

Name of the function in Structured Text: EXP().

Name of the function in the MathBSD library: **bsd_exp()** for LREAL and **bsd_expf()** for REAL numbers.

Transfer parameters	**Type**	**Description**
arg	REAL or LREAL	The value whose exponential value is to be calculated
Return parameters	**Type**	**Description**
If successful		

		The exponential value of the value transferred
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := EXP(dblX); // 4.2921552669021947

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := EXP(sngX); // 4.2921553
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/exp

7.3.3.11 EXP2

Calculates the potency based on two with the exponent arg.

Namo of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_exp2()** for LREAL and **bsd_exp2f()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The exponent
Return parameters	**Type**	**Description**
If successful		
		The potency (2 ^ arg)
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 1.456789;
  dblY := bsd_exp2(dblX);  // 2.7449673705415689

  sngX := 1.456789;
  sngY := bsd_exp2f(sngX); // 2.7449675
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/exp2

7.3.3.12 EXPM1

Calculates the exponential value of the value transferred minus 1. This variant is more accurate than EXP(arg) - 1.0 if arg is very small (almost zero).

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_expm1()** for LREAL and **bsd_expm1f()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The exponent
Return parameters	**Type**	**Description**
If successful		
		The result of the calculation (e ^ arg) - 1.0
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
```

```
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 1.456789;
  dblY := bsd_expm1(dblX);  // 3.2921552669021947

  sngX := 1.456789;
  sngY := bsd_expm1f(sngX); // 3.2921553
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/expm1

7.3.3.13 FABS

Calculates the absolute value of the transferred value.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_fabs()** for LREAL and **bsd_fabsf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose absolute value is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The absolute value
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblY : LREAL;
    sngY : REAL;
  END_VAR

  dblY := bsd_fabs(1.1); // 1.1
  dblY := bsd_expm1(-1.1);        // 1.1

  sngY :=bsd_fabsf(1.1); // 1.1
```

```
  sngY :=bsd_fabsf(-1.1);// 1.1
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/fabs

7.3.3.14 FLOOR

Calculates the largest integer value, which is not greater than the transfer value. Note: the set rounding mode is irrelevant!

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_floor()** for LREAL and **bsd_floorf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The value that - if decimal places exist - is to be rounded to the next lower integer
Return parameters	**Type**	**Description**
If successful		
		The transferred value itself (if it does not contain any decimal places), or the next lower integer
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblY : LREAL;
    sngY : REAL;
  END_VAR

dblY := bsd_floor(1.1);      // 1.0
    dblY := bsd_floor(0.9);      // 0.0
    dblY := bsd_floor(0.1);      // 0.0
    dblY := bsd_floor(-0.1);     // -1.0
    dblY := bsd_floor(-0.9);     // -1.0
    dblY := bsd_floor(-1.1);     // -2.0

    sngY := bsd_floorf(1.1);     // 1.0
    sngY := bsd_floorf(0.9);     // 0.0
    sngY := bsd_floorf(0.1);     // 0.0
    sngY := bsd_floorf(-0.1);    // -1.0
```

```
    sngY := bsd_floorf(-0.9);    // -1.0
    sngY := bsd_floorf(-1.1);    // -2.0

END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/floor

7.3.3.15 FMOD

Calculates the residual floating point value of division x/y.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_fmod()** for LREAL and **bsd_fmodf()** for REAL numbers.

Transfer parameters	Type	Description
x	REAL or LREAL	The dividend
y	REAL or LREAL	The divisor
Return parameters	**Type**	**Description**
If successful		
		The residual floating point value. Exactly the value x - n*y is calculated, where n is the integer part of the division x/y. The return value has the same sign as x and is smaller than y.
If failed (e.g. because y = 0)		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 1.456789;
  dblY := bsd_fmod(dblX, 0.75);  // 0.7067889999999999

  sngX := 1.456789;
  sngY := bsd_fmodf(sngX, 0.75); // 0.706789
END_FUNCTION
```

For more information see following reference address:
https://en.cppreference.com/w/c/numeric/math/fmod

7.3.3.16 FREXP

Splits the floating point number arg into two parts: a normalized floating point number in the range (-1;-0.5],[0.5; 1) and an integer exponent of 2. The exponent (an integer) is stored in memory at the address pointed to by the pointer exp.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_frexp()** for LREAL and **bsd_frexpf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number to be split
exp		A memory address, pointing to an integer
Return parameters	**Type**	**Description**
If successful		
		arg is split into the form x*2^(*exp) and the normalized floating point value x is returned and the exponent is written to memory at the address specified in exp
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
    VAR
        dblX : LREAL;
        dblY : LREAL;
        sngX : REAL;
        sngY : REAL;
        exp1 : DINT;
        exp2 : DINT;
    END_VAR
    dblX := 1.456789;
    dblY := bsd_frexp(dblX, #exp1);  // 0.7283944999999999

    sngX := 1.456789;
    sngY := bsd_frexpf(sngX, #exp2); // 0.7283945
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/frexp

7.3.3.17 HYPOT

Calculates the hypotenuse of a right-angled triangle. Compared to SQRT(x*x + y*y) this method is slower, but it can handle overflow and underflow better (see code example).

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_hypot()** for LREAL and **bsd_hypoth()** for REAL numbers.

Transfer parameters	Type	Description
x	REAL or LREAL	A catheter of the right-angled triangle
y	REAL or LREAL	The other catheter of the right-angled triangle
Return parameters	**Type**	**Description**
If successful		
		The hypotenuse of the right-angled triangle
If failed		
		Undefined

Example

#include "MathBSD.h"

```
FUNCTION Test
    VAR
        dblA : LREAL;
        dblB : LREAL;
        dblC1 : LREAL;
        dblC2 : LREAL;
        fltA : REAL;
        fltB : REAL;
        fltC1 : REAL;
        fltC2 : REAL;
    END_VAR

    dblA := 2.0e+154;
    dblB := 3.0e+154;
    // Soft way to compute Hypotenuse
    dblC1 := SQRT(dblA*dblA + dblB*dblB);   // 1.#IN F
    // Slower but can handle overflows much better
    dblC2 := bsd_hypot(dblA, dblB);   // 3.60555127546399e+154
```

```
    fltA := 0.706789;
    fltB := 0.7283945;
    fltC1 := SQRT(fltA*fltA + fltB*fltB);   // 1.#IN F
    fltC2 := bsd_hypotf(fltA, fltB);        // 2.236068e+034
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/hypot

7.3.3.18 LOG

Calculates the decadic logarithm (based on 10) of the value transferred.

Name of the function in Structured Text: LOG().

Name of the function in the MathBSD library: **bsd_log10()** for LREAL and **bsd_log10f()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose decadic logarithm (based on 10) is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The logarithm of arg
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := LOG(dblX); // 0.1633966535053308

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := LOG(sngX); // 0.1633967
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/log10

7.3.3.19 LOG1P

Calculates the value of the natural logarithm (based on e, the Eulerian number) of the value transferred plus 1. This function returns more accurate results than log(1+arg).

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_log1p()** for LREAL and **bsd_log1pf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number used to calculate the value of the natural logarithm (based on e, the Eulerian number) of the value transferred plus 1
Return parameters	**Type**	**Description**
If successful		
		Value of the natural logarithm (based on e, the Eulerian number) of the value transferred plus 1
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 1.456789;
  dblY := bsd_log1p(dblX);  // 0.8988552127655168

  sngX := 1.456789;
  sngY := bsd_log1pf(sngX); // 0.8988552
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/log1p

7.3.3.20 LN

Calculates the natural logarithm (based on e, the Eulerian number) of the value transferred.

Name of the function in Structured Text: LN().

Name of the function in the MathBSD library: **bsd_log()** for LREAL and **bsd_logf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose natural logarithm (based on e, the Eulerian number) is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The natural logarithm (based on e, the Eulerian number) of arg
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := LN(dblX); // 0.3762346986064879

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := LN(sngX); // 0.3762347
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/log

7.3.3.21 MODF

Separates the transferred number into an integer value and a decimal point value. A pointer in memory is transferred for the integer value.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_modf()** for LREAL and **bsd_modff()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number to be split
iptr	Pointer to a floating point value in the memory	The pointer that defines the memory area in which the integer part (as floating point number!) is to be written
Return parameters	**Type**	**Description**
If successful		
		The decimal part of the number as a floating point value. Same type and same sign as arg. The integer part is written into the memory at the address iptr as a floating point number (!).
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
    VAR
        dblX : LREAL;
        dblY : LREAL;
        ipart1 : LREAL;
        sngX : REAL;
        sngY : REAL;
        ipart2 : REAL;
    END_VAR

    dblX := 1.456789;
    dblY := bsd_modf(dblX, #part1);  // 0.456789 / 1.0

    sngX := 1.456789;
    sngY := bsd_modff(sngX, #ipart2); // 0.456789 / 1.0
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/modf

7.3.3.22 POW

Squares the transferred number.

Internal operator in Structured Text: **.

Name of the function in the MathBSD library: **bsd_pow()** for LREAL and **bsd_powf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number to be squared
Return parameters	**Type**	**Description**
If successful		
		The square of arg
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := 2.75 ** dblX; // 4.3653094511945598

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := 2.75 ** sngX; // 4.3653097
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/pow

7.3.3.23 REMAINDER

Calculates the residual floating point value of division x/y.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_remainder()** for LREAL and **bsd_remainderf()** for REAL numbers.

Transfer parameters	Type	Description
x	REAL or LREAL	The dividend
y	REAL or LREAL	The divisor
Return parameters	**Type**	**Description**
If successful		
		The residual floating point value. Exactly the value x - n*y is calculated, where n is the closest integer to the x/y division. Unlike FMOD, the return value does not have the same sign as x - it becomes negative if FMOD(x,y) > y/2.
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := bsd_remainder(dblX,2.75); // -1.2932110000000001

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := bsd_remainder(sngX, 2.75); // -1.293211
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/remainder

7.3.3.24 REMQUO

Calculates the residual floating point value of division x/y in the same way as the function REMAINDER(x,y) does, with the difference that the sign and the last 3 bits (modulo 8 of the integer result of x/y) are stored in the memory area determined by the third transfer parameter. This can be helpful for periodic (circular) trigonometric functions to limit the range where the result is in the coordinate system (quadrant or octant).

For example, if sin(10000000000000000000000000000000000000000000000000000000) is required, the result without REMQUO() could be very inaccurate due to the divisions to be performed internally during the calculation. Modulo PI/4 leaves exactly the last phase, i.e. the sign and the quadrant in which the sine value is located. This makes it possible to achieve a very accurate result, as no division with a very large dividend and a very small divisor now appears in the internal calculations.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_remquo()** for LREAL and **bsd_remquof()** for REAL numbers.

Transfer parameters	Type	Description
x	REAL or LREAL	The dividend
y	REAL or LREAL	The divisor
quo	pointer to a 4-bit memory area	Here the sign and the last three bits of the integer result of x/y are stored
Return parameters	**Type**	**Description**
If successful		
		The residual floating point value. Exactly the value x - n*y is calculated, where n is the closest integer to the x/y division. Unlike FMOD, the return value does not have the same sign as x - it becomes negative if FMOD(x,y) > y/2. In addition, the sign and the last 3 bits of the integer result of x/y are stored in the memory area displayed by the transfer parameter quo.
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
    VAR
        dblX : LREAL;
```

```
        dblY : LREAL;
        quo1 : DINT
        sngX : REAL;
        sngY : REAL;
        quo2 : DINT;
    END_VAR

    dblX := 1.456789;
    dblY := bsd_remquo(dblX, 2.75, #quo1);  // -1.2932110000000001 / 1

    sngX := 1.456789;
    sngY := bsd_remquof(sngX, 2.75, #quo2); // -1.293211 / 1
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/remquo

7.3.3.25 ROUND

Rounds the transferred value to an integer. The following applies here: As of .5, it is rounded to the next higher integer in absolute value, under .5 to the next lower integer.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_round()** for LREAL and **bsd_roundf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number to be rounded
Return parameters	**Type**	**Description**
If successful		
		Arg rounded to an integer
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
    VAR
        dblY : LREAL;
        sngY : REAL;
    END_VAR

    dblY := bsd_round(1.1);     // 1.0
    dblY := bsd_round(0.9);     // 1.0
```

```
    dblY := bsd_round(0.1);     // 0.0
    dblY := bsd_round(-0.1);    // 0.0
    dblY := bsd_round(-0.9);    // -1.0
    dblY := bsd_round(-1.1);    // -1.0

    sngY := bsd_roundf(1.1);    // 1.0
    sngY := bsd_roundf(0.9);    // 1.0
    sngY := bsd_roundf(0.1);    // 0.0
    sngY := bsd_roundf(-0.1);   // 0.0
    sngY := bsd_roundf(-0.9);   // -1.0
    sngY := bsd_roundf(-1.1);   // -1.0
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/round

7.3.3.26 SCALBN

Returns the value arg * FLT_RADIX ^ exp.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_scalbn()** for LREAL and **bsd_scalbnf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number to be multiplied
exp	Integer	The exponent with which the constant FLT_RADIX is to be exponentiated
Return parameters	**Type**	**Description**
If successful		
		Arg * FLT_RADIX ^ exp
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR
```

```
  dblX := 1.456789;
  dblY := bsd_scalbn(dblX, 9);  // 745.87596799999994

  sngX := 1.456789;
  sngY := bsd_scalbnf(sngX, 9); // 745.8759766
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/scalbn

7.3.3.27 SIN

Calculates the sine of the value transferred.

Name of the function in Structured Text: SIN().

Name of the function in the MathBSD library: **bsd_sin()** for LREAL and **bsd_sinf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose sine is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The sine of arg
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := SIN(dblX); // 0.9935082008128356

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := SIN(sngX); // 0.9935082
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/sin

7.3.3.28 SINCOS

Calculates the sine and the cosine of the transferred value in parallel. Some applications require both values (sine and cosine) of the same number. Im Vergleich zu zwei aufeinanderfolgenden Aufrufen (SIN() und COS()) bringt hier SINCOS einen Performancegewinn.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_sincos()** for LREAL and **bsd_sincos()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose sine and cosine are to be calculated
sin	Pointer to a floating point value	Pointer pointing to the memory area where the calculated sine value is stored
cos	Pointer to a floating point value	Pointer pointing to the memory area where the calculated cosine value is stored
Return parameters	**Type**	**Description**
If successful		
		void
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
    VAR
        dblX : LREAL;
        dblS : LREAL;
        dblC : LREAL;
        sngX : REAL;
        sngS : REAL;
        sngC : REAL;
    END_VAR

    dblX := 1.456789;
    bsd_sincos(dblX, #dblS, #dblC);  // dblS: 0.9935082008128356
```

```
                                    // dblC: 0.1137605156354449
    sngX := 1.456789;
    bsd_sincosf(sngX, #sngS, #sngC); // sngS: 0.9935082
                                    // sngC: 0.1137605
END_FUNCTION
```

For more information see following reference address:

https://linux.die.net/man/3/sincos

7.3.3.29 SINH

Calculates the hyperbolic sine of the transferred value.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_sinh()** for LREAL and **bsd_sinhf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose sine hyperbolicus is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The hyperbolic sine of arg
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 1.456789;
  dblY := bsd_sinh(dblX);  // 2.0295860414866556

  sngX := 1.456789;
  sngY := bsd_sinhf(sngX); // 2.0295861
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/sinh

7.3.3.30 SQRT

Calculates the square root of the value transferred.

Name of the function in Structured Text: SQRT().

Name of the function in the MathBSD library: **bsd_sqrt()** for LREAL and **bsd_sqrtf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose square root is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The square root of arg
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := SQRT(dblX); // 1.2069751447316552

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := SQRT(sngX); // 1.2069751
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/sqrt

7.3.3.31 TAN

Calculates the tangent of the value transferred.

Name of the function in Structured Text: TAN().

Name of the function in the MathBSD library: **bsd_tan()** for LREAL and **bsd_tanf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose tangent is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The tangent of arg
If failed		
		Undefined

Example

```
FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  // LREAL arguments use double precision
  dblX := 1.456789;
  dblY := TAN(dblX); // 8.7333306750877924

  // REAL arguments use single precision
  sngX := 1.456789;
  sngY := TAN(sngX); // 8.7333317
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/tan

7.3.3.32 TANH

Calculates the tangent hyperbolicus of the transferred value.

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_tanh()** for LREAL and **bsd_tanhf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose tangent hyperbolicus is to be calculated
Return parameters	**Type**	**Description**
If successful		
		The tangent hyperbolicus of arg
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
  VAR
    dblX : LREAL;
    dblY : LREAL;
    sngX : REAL;
    sngY : REAL;
  END_VAR

  dblX := 1.456789;
  dblY := bsd_tanh(dblX);  // 0.8970271577497947

  sngX := 1.456789;
  sngY := bsd_tanhf(sngX); // 0.8970271
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/tanh

7.3.3.33 TRUNC

Cuts off the decimal places of the value passed (so an integer remains).

Name of the function in Structured Text: not available

Name of the function in the MathBSD library: **bsd_trunc()** for LREAL and **bsd_truncf()** for REAL numbers.

Transfer parameters	Type	Description
arg	REAL or LREAL	The number whose pre-decimal places are to be calculated
Return parameters	**Type**	**Description**
If successful		
		The integer part of arg
If failed		
		Undefined

Example

```
#include "MathBSD.h"

FUNCTION Test
    VAR
        dblY : LREAL;
        sngY : REAL;
    END_VAR

    dblY := bsd_trunc(1.1);     // 1.0
    dblY := bsd_trunc(0.9);     // 0.0
    dblY := bsd_trunc(0.1);     // 0.0
    dblY := bsd_trunc(-0.1);    // 0.0
    dblY := bsd_trunc(-0.9);    // 0.0
    dblY := bsd_trunc(-1.1);    // -1.0

    sngY := bsd_truncf(1.1);    // 1.0
    sngY := bsd_truncf(0.9);    // 0.0
    sngY := bsd_truncf(0.1);    // 0.0
    sngY := bsd_truncf(-0.1);   // 0.0
    sngY := bsd_truncf(-0.9);   // 0.0
    sngY := bsd_truncf(-1.1);   // -1.0
END_FUNCTION
```

For more information see following reference address:

https://en.cppreference.com/w/c/numeric/math/trunc

7.3.4 C Programming Language Interface

7.3.4.1 Application

<MathBSD.h> should be included. The functions are the same as in the <math.h> library, but have a "bsd_" prefix.

7.3.4.2 Example with LREAL

Task: the sine of the angle sum is to be calculated:

```
sin(a + b)
```

Sine of the angle sum in Structured Text:

```
#include "MathBSD.h"

FUNCTION GLOBAL __CDECL LslSinSum
  VAR_INPUT
    a: LREAL;
    b LREAL;
  END_VAR
  VAR_OUTPUT
    y : LREAL;
  END_VAR
  VAR
    sin_a : LREAL;
    sin_b : LREAL;
    cos_a : LREAL;
    cos_b : LREAL;
  END_VAR

  bsd_sincos(x:=a, sine:=#sin_a, cosine:=#cos_a);
  bsd_sincos(x:=b, sine:=#sin_b, cosine:=#cos_b);

  y := sin_a * cos_b + cos_a * sin_b;

END_FUNCTION
```

Sine of the angle sum in the programming language C:

```
#include <MathBsd.h>

/*!
Returns the sin of the sum of two angle, using the classical formula:

    sin(A+B) = sin(A).cos(B) + cos(A).sin(B)
*/
extern "C" double SinSum(double a, double b)
{
  double sin_a, sin_b, cos_a, cos_b;

  // Use speed optimized functions
  bsd_sincos(a, &sin_a, &cos_a);
  bsd_sincos(b, &sin_b, &cos_b);
```

```
  // sin(A+B) = sin(A).cos(B) + cos(A).sin(B)
  return sin_a * cos_b + cos_a * sin_b;
}
```

How can the C function be accessed from Structured Text?

The SinSum function was created in C code. If you want to access this function later from Structured Text, it must be declared as external "C" in C. In Structured Text you need the following declaration:

```
// Declare the C functions for our ST program
FUNCTION __CDECL GLOBAL SinSum
  VAR_INPUT
    a: LREAL;
    b LREAL;
  END_VAR
  VAR_OUTPUT
    y : LREAL;
  END_VAR;
```

7.3.4.3 Example with REAL



Tip

You can read more about the interrelationships between the C programming language and ST code especially for Intel x86 CPUs in the chapter Compatibility between LASAL CLASS 2 and CDECL functions with REAL return values

The programming languages C and LASAL CLASS 2 handle REAL values in different ways and are therefore not compatible with each other.

The header file MathBSD.h provides a data type LslFloatC_t for the C programs. With the macros L2F_() and F2L_() you can easily convert the values back and forth.

Sine of the double angle in Structured Text:

```
#include "MathBSD.h"

FUNCTION __CDECL GLOBAL LslSin2x
  VAR_INPUT
    x : REAL;
  END_VAR
  VAR_OUTPUT
    y : REAL;
  END_VAR
  VAR
    sin_x : REAL;
    cos_x : REAL;
  END_VAR

  sin_x := SIN(x);
  cos_x := COS(x);
```

```
  y := 2 * sin_x * cos_x;

END_FUNCTION
```

Sine of the double angle in C

```
/*!
This is function computes the sin of the double-x, using the classical formula:

    sin(2.X) = 2.sin(X).cos(x)

This example demonstrates how to interface the float family of functions, since Lasal2 uses a d
MathBSD.h header offers a *LslFloatC_t* type and the *L2F_()* and *F2L_()* macros to perform th
*/

extern "C" LslFloatC_t Sin2x(float x)
{
  // 'LslFloatC_t' values can be easily converted to 'float' using the
  // L2F_() macro
  float sin_x = L2F_(bsd_sinf(x));
  float cos_x = L2F_(bsd_cosf(x));

  // A 'float value can be converted to 'LslFloatC_t' using the F2L_()
  // macro
  return F2L_(2 * sin_x * cos_x);
}
```

Again, the declaration must be used to access the C function from LASAL CLASS 2:

```
// Declare the C functions for our ST program
FUNCTION __CDECL GLOBAL Sin2x
  VAR_INPUT
    x : REAL;
  END_VAR
  VAR_OUTPUT
    y : REAL;
  END_VAR;
```

7.4 Compatibility between LASAL CLASS 2 and CDECL functions with REAL return values

With **Intel x86 processors** and **CDECL** functions with **REAL return values**, **complex possibilities** arise with regard to parameter passing or result return. Depending on whether the caller or called function is written in **C or Stuctured Text** (ST), and depending on whether **the EAX register is specified or not** as the passing interface, incompatibilities, and thus unexpected results, may occur in certain compiler versions. In this chapter all possible combinations are described depending on the respective compiler version.

There are now two possibilities of the **function declaration** of a CDECL function:

- **Option 1:** Backward compatibility is guaranteed (so-called "**legacy mode**") - this variant should be preferred above all with libraries, which are to function also in old compiler versions.



"Legacy mode":

```
FUNCTION __CDECL MyTestFunction
    VAR_OUTPUT
        RetVal (EAX) : REAL; // uses backward compatible ABI
    END_VAR
    ...
END_FUNCTION
```

- **Option 2:** C-Convention. Here, the procedure is strictly in accordance with the specifications of the **C-ABI Convention**:



"C ABI Konvention":

```
FUNCTION __CDECL MyTestFunction
    VAR_OUTPUT
        RetVal : REAL; // This use the default (FPU stack)
    END_VAR
    ... // Custom code
END_FUNCTION
```

- If a C function is to be used in ST, it needs a **forward declaration** in the latter so that the ST compiler recognizes the C function as such, such as:



"Forward declaration":

```
FUNCTION __CDECL MyCFunction
    VAR_OUTPUT
        RetVal : REAL; // The only way for a C Function returning REAL
                       // is demonstrated here
    END_VAR;
```

The following table now shows the **compatibility** between LASAL CLASS 2 and CDECL functions with REAL return values in the different compiler versions on Intel x86 CPUs.



Attention

When interpreting the table, it should be noted that there are always two parties involved: the calling function, and the called function. Both must use the same convention for a compatible interface. Since the ST compiler allows the specification of registers, there are two possibilities each: **with and without EAX declaration**.

Compiler version	Calling function	Called function	compatible?	Note
all versions	C	C	yes	Default, the FPU stack is used
all versions	C	ST (with EAX)	**no**	The ST code returns the result in the EAX register - but **this is against the C convention**, because the C programming language expects the result in the FPU stack
C66 and earlier				**The ST compiler bypasses the C convention: everything is done in the EAX register**
	C	ST (without EAX)	no	The C code expects the result in the FPU stack, but the EAX register is used implicitly in the ST up to compiler version 66
	ST (with EAX in the forward declaration)	C	no	The C code returns the result via the FPU stack, but ST expects it in the EAX register
	ST (without EAX in the forward declaration)	C	no	Like the previous line, because up to compiler version 66 the EAX register is implicitly used.
	ST	ST (with EAX)	yes	This variant is not C-compliant, but still works, because both functions (caller and called) are in the ST area
	ST	ST (without EAX)	yes	This variant works for the same reason as the previous one, because the EAX register is used implicitly up to and including compiler version 66!
C66-C71				**Here the ST compiler always expects the result in the FPU stack, but the called ST functions cannot yet return via the FPU stack in these compiler versions. This behavior has been corrected as of compiler version 72**
	C	ST (without EAX)	no	ST functions still return the result in the EAX register

	ST (with EAX in the forward declaration)	C	yes	The specification EAX in the forward declaration has no effect (see the example at the beginning for backward compatibility). On both sides the FPU stack is used
	ST (without EAX in the forward declaration)	C	yes	As in the previous line, since here implicitly the FPU stack is always used
	ST	ST (with EAX)	no	Specifying EAX on return has no effect, the caller function expects the result in the FPU stack
	ST	ST (without EAX)	no	As in the previous line, because the EAX register is implicitly used for the return
C72 and later				**Now, C-area always follows the C-convention; in ST-area, however, the declaration determines whether the transfer should happen in the EAX-register, or in the FPU-stack**
	C	ST (without EAX)	yes	Works according to C convention (recommended!)
	ST (with EAX in the forward declaration)	C	**no**	**Does not work, because according to C convention the transfer is always expected via the FPU stack!**
	ST (without EAX in the forward declaration)	C	yes	Works. Please note that the forward declaration for pure C functions must follow the model in the example above (**an EAX register specification is not allowed**!)
	ST	ST (with EAX)	yes	Backward compatible mode. Primarily to be used with libraries that must work with all compiler versions
	ST	ST (without EAX)	yes	The C convention is applied

7.5 Keyboard Shortcuts

In LASAL CLASS 2 keyboard shortcuts are available for most of the function. The table below lists them.

Ctrl+Alt	F2	Delete all bookmarks
	F2	Goto next bookmark
Shift	F2	Goto previous bookmark
Ctrl	F2	Set / delete a bookmark
	F9	Compile changes
Ctrl	Cancel	Abort compiler process
Alt	F9	Compile module
Ctrl	F9	Compile project
Alt	F6	Go online
	F6	Download
	F8	Reset
	F7	Start/Restart
Shift	F4	Delete all breakpoints
Ctrl+Alt	F4	Add a data breakpoint
	F4	Set/delete a breakpoint
	F5	Single step
Shift	F5	Procedure-step
Alt	F5	Execute until return
Ctrl+Shift	C	Set program code under Commentary
Ctrl+Shift	X	Set program code under Commentary back again

Ctrl+Shift	D	Show assembler code
Ctrl	C	Copy
Ctrl	Insert	Copy
Ctrl	X	Cut
Shift	Delete	Cut
Ctrl	V	Paste
Shift	Insert	Paste
Ctrl	A	Select all
Ctrl	F	Find
	F3	Find again
Ctrl+Shift	F	Find in files
Ctrl	R	Replace
Ctrl+Shift	R	Replace in files
Ctrl+Shift	T	Find in tree view
Ctrl	G	Goto line
Ctrl	Z	Undo the last change
Ctrl	Y	Restore the last change removed with Undo (Redo)
	F11	Go To definition
Shift	F11	Jump to cursor position in the code according to "Go To Definition" and/or F11
	F12	Jump to the next error in the program code
Shift	F12	Jump to the previous error in the program code
Ctrl	F12	Jump to the next error in the program code (without warning)
Ctrl + Shift	F12	Jump to the previous error in the program code (without warning)

Ctrl+Shift	**N**	Add a new file
Ctrl	**N**	Add a new project
Ctrl+Shift	**O**	Opens an (existing) file
Ctrl	**O**	Opens an (existing) project
Ctrl	**S**	Save file
Ctrl+Shift	**S**	Save project
Ctrl	**Cursor Up** **Cursor Dn**	Scroll without moving the cursor but so that the cursor remains in the visible area ("drag pointer").
Ctrl	**PageDownPageUp**	Jump to then next/previous function in the program code
Ctrl	**F4**	Close the marked pane in the work area
Ctrl	**Tab**	Jump to next pane
Ctrl	**F6**	Jump to next pane
Ctrl+Shift	**Tab**	Jump to previous pane
Ctrl+Shift	**F6**	Jump to previous pane
Ctrl	**D**	Set focus to the Properties window
Ctrl	**T**	Set focus to the tree
Ctrl	**E**	Set focus to the Editor
Alt	**O**	Open the Options dialog
Ctrl	**Mouse wheel**	Zoom the code editor

Ctrl+Alt	I	Import EasyCase Class
Ctrl+Alt	E	Export EasyCase Class
Shift	Mouse wheel	Horizontal scroll in network
Ctrl	Mouse wheel	Zoom in network

7.6 Loader Trace Messages

This chapter explains all messages output by the Loader.

Message	Description
Loading project...	The loader has been started (only LASAL 2).
Project successfully loaded	The loader has been successfully loaded (only LASAL 2).
No more Array for LOGIN Data (COMLINK_CAN_COMCHS)	The maximum amount of the Comlink channels has been reached (configurable via the global #define COMLINK_CAN_COMCHS, default value => 5). This #define must have the same value on all stations in a Comlink network.
No more Memory for LOGIN Data (Online --> COMDATA)	No memory could be reserved for a Comlink channel (see the previous message).
LOGIN: no more CAN comlink channels available	For the login no CAN Comlink channels are available anymore (only LASAL CLASS).
sizeof maxStatCOMcell (comlink.st), sizeof maxDynCOMcell (comlink.st)	The maximum number of static/dynamic update segments of the Comlink server has been reached. Update segments are collected usually and send to the client as packet.
Comlink/TCP client error: TCP_USER interface not available	On the Comlink client the- desired TCP interface is not available.
Comlink/TCP client error: MT interface not available	On the Comlink client the LSL_Multitask interface is not supported.

Comlink/TCP client error: wrong OS-Version (must be > 5.42)	On the Comlink client the LSL_Multitask interface is not supported, because a wrong operating system version runs.
Comlink/TCP client error: unknown error code	A not specified error occurred at initialization of the Comlink clients (Mutex could not be created).
TCP-Server for command data not started	The TCP port for the command channel of the Comlink server could not be set into receipt mode. Possibly the port is already allocated.
TCP-Server for refresh data not started	The TCP port for the data channel of the Comlink server could not be set into receipt mode. Possibly the port is already allocated.
Comlink/TCP client error: missing client code (COMLINK_TCP_CLIENT not defined)	The #define COMLINK_TCP_CLIENT in the Loader.h (LASAL CLASS), respectively, UserDef.h (LASAL 2) is commented or missing.
Name specifies channel with wrong type, expectig object channel!	The defined client is no object channel. Through setting a breakpoint on this line in the loader, the name of the client can be found out via the ppScan variable.
Missing Loader: You are using a dummy loader instead of the real loader!	There is no loader available on the control. Via LASAL 2 the project can be send to the control, also without the loader (only LASAL 2).
Covert of RAM not possible --> no Memory	At conversion of the S-RAM data an error occurred (too low memory).
SAVE OF RamFile not possible (Disk full?)	At saving the RamFiles an error occurred (FileOpen failed).
Error in SRAM, see c:\sysmsg\event19.log	The SRAM is erroneous (FileOpen failed, write error, read error, too much or 0 entries, check sum not accepted, RamEx erroneous).
Invalid format string	An invalid format string has been assigned to the _Format function (only LASAL 2).
Use the RAM/RAMex – class to store retentive data of this platform!	The call of SetRetentiveFlag is only permitted for IPC platforms. All other platforms have to use instances of RAM or RAMex classes.

CLASS RAM must have a minor version of minimum 1 --> Please import new CLASS RAM	The C-IPC platform needs a version number of > 0.1 for the RAM class.
CLASS RAMex must have a minor version of minimum 2 --> Please import new CLASS RAM	The C-IPC platform needs a version number of > 0.2 for the RAMex class.
Invalid object number	The GetObjectNo method of the loader class ObjectReader delivered an invalid object number (>Object number). A „Rebuild All" should help (only LASAL 2).
Configuration table not found	The *SetTable* method of the loader class ObjectReader has been called with a NULL pointer in argument (only LASAL 2).
Configuration table has a wrong ID	The identification word (Base + 4) of the table is invalid (Call of the *SetTable* method of the loader class ObjectReader) (only LASAL 2).
Do not create an object twice!	The *NewObj* method of the loader class ObjectReader has been given an object name, which already exists.
No object created in this project	No objects are existent in the project. It has not to be an error, there can be sources without classes and objects (only LASAL 2).
Reentrance in not supported!	A running interpreter program is recalled. There are no protection mechanisms to control this state.
Make sure the <RamEx> class is part of your project!	The RamEx class from the memory library is necessary, in order to save interpreter programs and breakpoint lists.
Internal loader error: class ProgIp	The loader requires the ProgIp class (only LASAL 2).
Could not load interpreter programs	The check of the interpreter code fails (only LASAL 2).
RAMex block maximum 64 kByte	The *Alloc* method of the RAMex class can reserve a maximum of 64 kByte per call (only LASAL 2).
This interface is not available with the installed version oft he OS	An error occurred at installing the global interface pointer. The required interface is not existent on the control. Eventually the operating system version is too old (only LASAL 2).
ERROR: OBJECT-CONTROL Interface not available with	The LASALTASK interface is not existent (only LASAL 2).

this veersion / platform!	
OS-version too old. Doubleclick for more information!	The operating system version is too old (<5.0) for supplying the RAM class (only Lasal Class).
Alloc Problem	The given index exceeds the size (CstoreTabs class) (only LASAL CLASS).
OSI – classes too old?	The OSI classes are too old, a new import of the OSI classes (Graphic, System,…) can resolve the problem.
Cannot enter connection to ‚objectname‘	A client connection to this object cannot be created. Possible causes: Object has been deleted.
Ext. Connection from object ‚objectname‘ at client.no 00000002 failed	A connection of this object in an external network could not be created. The number indicates the client, who wanted to make the connection. Only connections to external networks are relevant.

7.7 Compiler Messages

This chapter explains all messages output by the Compiler.

7.7.1 E 0001: Value missing instead of '%s'

This error occurs when an attempt is made to assign a specific Enum-entry to a non-constant value. If in the example above, the DintVar was replaced with a constant, the lines of code are complied without error.

Example

```
TYPE
  enumType:
  (
    Item1 := DintVar
  );
END_TYPE

Compiler: [ERROR] E 0001
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(426)Value missing instead
of 'DintVar'
```

7.7.2 E 0002: Missing '%s' instead of '%s'

This error occurs when blocks or instructions are not ended correctly. In this example, the "end_if" instruction is missing; therefore an error is generated. A warning with the same number is generated if, for example, a semicolon is missing from the end of an instruction and the corresponding compiler settings are made.

Example

```
FUNCTION Class0::E0002
  if (DintVar > 1) then
    DintVar := DintVar SHL 3;
    DintVar += 2;
END_FUNCTION

Compiler: [ERROR] E 0002
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(434)Missing 'END_IF'
instead of 'END_FUNCTION'
```

7.7.3 E 0003: Multiple declaration of '%s'

This error is triggered when multiple elements have the same name (see example: 2 local variables with the name "LokVar"). The error message is not only triggered by local variable but also generated by double names for TYPE-, VAR_GLOBAL instructions.

Example

```
FUNCTION Class0::E0003
VAR                 //can also be e.g. TYPE, VAR_GLOBAL etc.
  LokVar   : DINT;
  LokVar   : BOOL;
END_VAR
END_FUNCTION
```

```
Compiler: [ERROR] E 0003
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(440)Multiple declaration
of 'LokVar'
```

7.7.4 E 0004: Undefined type '%s'

This error occurs when a variable is declared with an unknown type. The type is either not available in the project or not accessible in the respective context (e.g.: class files) because it was, for example, defined in a header file not connected to a class.

Example

```
FUNCTION Class0::E0004
VAR
  LokVar1 : NOT_EXISTING_TYPE;
END_VAR
END_FUNCTION
```

```
Compiler: [ERROR] E 0004
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(448)Undefined type
'NOT_EXISTING_TYPE'
```

7.7.5 E 0005: '%s' is no integer value

This error message is given when, in a specific context, a whole number value is expected as shown in the example above. However, a real value was given. The same error message is also generated when the real value above is replaced with a string (e.g.:"MyString").

Example

```
FUNCTION Class0::E0005
VAR
  LokArrVar : ARRAY [0 .. 1.34] OF DINT;
END_VAR
END_FUNCTION
```

```
Compiler: [ERROR] E 0005
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(456)'1.34' is no integer
value
```

7.7.6 E 0006: '%s' is no constant value

This error message is occurrs when the compiler expects a literal constant numeric value and another kind of token is found, which is described in the message.

7.7.7 E 0007: Value '%s' out of range

This error occurs when a fixed value range is exceeded. In the example, this error is generated twice. Once for exceeding the array limit (element 23 is accessed although the elements range from 0 to 3) and again for exceeding the bit limits (a BDINT type has only 32 bits, however the 34th bit was called).

Example

```
FUNCTION Class0::E0007
VAR
  LokArr  : ARRAY [0..3] OF DINT;
END_VAR
  ArrVar[23] := 13;
  BdintVar.34 := true;
END_FUNCTION
```

```
Compiler: [ERROR] E 0007
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(469)Value '23' out of ran-
ge
Compiler: [ERROR] E 0007
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(470)Value '34' out of ran-
ge
```

7.7.8 E 0008: Expected constant instead of '%s'

This error occurs when a pre-processor instruction with "#if" or "#elif" are processed using non-constant values. In the example above, the "#if" query contains a local variable instead of a constant.

Pre-processor instructions affect the compiler. At the time of the compilation however, the value of a variable is not yet known.

Example:

```
FUNCTION Class0::E0008
VAR
  LokVar  : DINT;
END_VAR
  #if LokVar = 1
  #endif
END_FUNCTION
```

```
Compiler: [ERROR] E 0008
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(495)Expected constant
instead of 'LokVar'
```

7.7.9 E 0009: '%s' has wrong type

The specified element does not match the expected type. The data type for the expression, variable or parameter must be consistent.

7.7.10 E 0010: Incorrect string '%s'

This error is triggered when the '$' character occurs within a string, the character following is not a letter or digit from the ASCII character set. The following combinations with the '$' character are used as control characters within a string.

The following combinations with the character '$' can be used as control characters within a string.

- L or l($L or $l = LineFeed)
- N or n($N or $n = NewLine)
- P or p($P or $p = FormFeed)
- R or r($R or $r = CarriageReturn)
- T or t($T or $t = Tabulator)

Example

```
FUNCTION Class0::E0010
VAR
  pcName: ^CHAR;
END_VAR
  pcName := "abc$-def";
END_FUNCTION
```

```
Compiler: [ERROR] E 0010
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(512)Incorrect string
'abc$'
```

7.7.11 E 0011: Incorrect value '%s'

This error occurs when a number base is assigned that is not supported by the compiler. The compiler supports the binary (2#), octal (8#) and hexadecimal (16#) number systems.

Example

```
FUNCTION Class0::E0011
  DintVar := 13#123;
END_FUNCTION
```

```
Compiler: [ERROR] E 0011
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(485)Incorrect value '13'
```

7.7.12 E 0012: Different types: using '%s' instead of '%s'

This error occurs when 2 incompatible elements are assigned to each other. In this case, the compiler setting "ExactTypeChecking" plays a significant role. If this compiler setting is set to **false**, no error would have been generated in the example above since both types differ only by the sign.

When a REAL value is assigned to a DINT, the compiler automatically returns an error since the format of both types is not compatible. Assigning values to type with an insufficient data length (e.g. DINT to INT) also generates an error.

Note:
If a directly assigned string (e.g. "MyString") is assigned by an instruction, this corresponds the type "^USINT".

Example

```
FUNCTION Class0::E0012
  DintVar := UdintVar;
END_FUNCTION
```

```
Compiler: [ERROR] E 0012
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(491)Different types: using
'UDINT' instead of 'DINT'
```

7.7.13 E_0013: Expected '%s' init values instead of '%s'

The declared size of an array and the total number of elements declared does not match. You should declare the exact number of elements as the size of the declared array.

7.7.14 E_0014: Unknown error

The specified element is not part of the structure. Check for misspelled names or deleted structure members.

7.7.15 E 0015: Error reading file '%s'

This error occurs when non-existing files are assigned and/or files that cannot be found in the specified location. In LASAL 2, this error message is triggered only when files are connected directly in the source code files.

If a class-specific file is not found, LASAL 2 indicates this when opening the project and then can be assigned to a new location.

Example

```
FUNCTION Class0::E0015
  #include ".\MyHeader.h"
END_FUNCTION
```

```
Compiler: [ERROR] E 0015
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(503)Error reading file
'.\MYHEADER.H'
```

7.7.16 E 0016: Not expected input '%s'

This error message indicates a syntax problem in an instruction. In the example above, the constant is replaced with the expression "*16". The character '*' is invalid as is the semicolon in the 2nd instruction.

Example

```
FUNCTION Class0::E0016
  #define E0016KONST *16
  DintVar := E0016KONST;
  DintVar := ;
END_FUNCTION
```

```
Compiler: [ERROR] E 0016
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(510)Not expected input '*'
Compiler: [ERROR] E 0016
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(511)Not expected input ';'
```

7.7.17 E 0017: Syntax error: '%s'

This error occurs with syntax errors in AWL functions. In AWL, the syntax is dependent on each instruction on the left side being predefined.

In the above example, the first error, "NotExistingName", is generated due to the unknown variable/constant. The second error is generated because of the unnecessary argument (", 3000").

Example

```
FUNCTION AWL Class0::E0017
  L.EAX   NotExistingName
  L.EAX   DintVar, 3000
END_FUNCTION
```

```
Compiler: [ERROR] E 0054
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(517)Syntax error
Compiler: [ERROR] E 0054
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(518)Syntax error
```

7.7.18 E 0018: Expected value instead of '%s'

This error occurs when a numeric value in a specific context is expected but is not received. In the example above, this error is triggered in a case instruction since the individual cases must be numeric values.

Example

```
FUNCTION Class0::E0018
  case DintVar of
  "hallo":
  end_case;
END_FUNCTION
```

```
Compiler: [ERROR] E 0018
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(525)Expected value instead
of ':'
```

7.7.19 E 0019: Expected identifier instead of '%s'

This error occurs when the header of a method or function contains unexpected symbols (constants, variables, digits letters).

Example

```
FUNCTION VIRTUAL 123 GLOBAL Class0::E0019
END_FUNCTION
```

```
Compiler: [ERROR] E 0019
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1235)Expected identifier
instead of 'VIRTUAL'
```

7.7.20 E 0020: UNUSED

This error message is currently not used.

7.7.21 E 0021: Use only one return value with a register

A register can hold only a single information at a time. More than one value was associated with the register.

7.7.22 E 0022: Invalid register '%s'

This error is generated when a method declared with "__cdecl" returns a value whose data length is greater than 5 bytes. C functions expect a return value from the EAX register, therefore the return value is limited to 4 bytes.

Example

```
TYPE
  T5Byte: STRUCT
    item1: DINT;
    item2: BYTE;
  END_STRUCT;
END_TYPE

FUNCTION __CDECL Class0::E0022
VAR_OUTPUT
  OutParam: T5Byte;
END_VAR
END_FUNCTION

Compiler: [ERROR] E 0022
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(582)Invalid register 'EAX'
```

7.7.23 E 0023: UNUSED

This error message is currently not used.

7.7.24 E 0024: Register '%s' already used as parameter

For transfer and return values, the register to which values are sent can be defined. The transfer value must, however, have its own register. An error is generated when two or more parameters use the same register.

Example

```
FUNCTION Class0::E0024
VAR_INPUT
  in1 (EAX)   : DINT;
  in2 (EAX)   : DINT;
END_VAR
  DintVar := in1;
  UdintVar := in2$UDINT;
END_FUNCTION
```

```
Compiler: [ERROR] E 0024
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(548)Register 'EAX' already
used as parameter
```

7.7.25 E 0025: Expected 0 or 1, not '%s'

This error occurs when flags from the flags register of the processor are queried (e.g. CarryFlag, see example above) when the query is at a value unlike 0 or 1.

Example

```
FUNCTION Class0::E0025
  IF CF = 2 THEN
  END_IF;
END_FUNCTION
```

```
Compiler: [ERROR] E 0025
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(596)Expected 0 or 1, not
'2'
```

7.7.26 E 0026: Undefined symbol '%s'

This error occurs when a name in the corresponding context is not recognized (e.g. Class files).

Example

```
FUNCTION Class0::E0026
  DintVar := UndefinedSymbol;
END_FUNCTION
```

```
Compiler: [ERROR] E 0026
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(560)Undefined symbol 'Un-
definedSymbol'
```

7.7.27 E 0027: STRUCT '%s' has no element '%s'

This error occurs when a non-existent structure element is accessed. In the above example, the type "StructType" has no "MyItem" element.

Example

```
FUNCTION Class0::E0027
  DintVar := StructVar.MyItem;
END_FUNCTION
```

```
Compiler: [ERROR] E 0027
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(566)STRUCT 'StructType'
has no element 'MyItem'
```

7.7.28 E 0028: Bitfield '%s' has no element '%s'

This error is generated when a non-defined bit name is accessed. In the above example, "BdintType" has no bit named "MyItem".

Example

```
FUNCTION Class0::E0028
  DintVar := BdintVar.MyItem;
END_FUNCTION
```

```
Compiler: [ERROR] E 0028
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(569)Bitfield 'BdintType'
has no element 'MyItem'
```

7.7.29 E 0029: Class '%s' has no element '%s'

This error occurs when a non-existing class element is accessed. As in the above example, the class must be specified directly otherwise the error message "Undefined symbol" is returned.

The error E0029 is generated when a class method is implemented in the source code file in which the class declaration is not defined (this means that it's not visible in the tree).

Example

```
FUNCTION Class0::E0029
  Class0::Server3::Read();
END_FUNCTION
```

```
Compiler: [ERROR] E 0029
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(575)Class 'Class0' has no
element 'Server3'
```

7.7.30 E 0030: Misplaced EXIT

This error occurs when an Exit instruction is placed outside of a loop (for, while, repeat).

Example

```
FUNCTION Class0::E0030
  Exit;
END_FUNCTION
```

```
Compiler: [ERROR] E 0030
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(581)Misplaced EXIT
```

7.7.31 E 0031: Constant not allowed in multiple assignment

This error occurs when a constant is used in a multiple assignment. A constant can naturally be placed at the end of an instruction but never at the beginning or middle as in the example above.

Example

```
FUNCTION Class0::E0031
  DintVar := KONSTVALUE := UdintVar;
END_FUNCTION

Compiler: [ERROR] E 0031
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(587)Constant not allowed
in multiple assignment
```

7.7.32 E 0032: Calculation not allowed in multiple assignment

This error is generated when a calculation is made within a multiple assignment. A calculation can take place at the end of an multiple assignment but not at the beginning or middle as in the example above.

Example

```
FUNCTION Class0::E0032
  DintVar := UdintVar$DINT * 2 := KONSTVALUE;
END_FUNCTION

Compiler: [ERROR] E 0032
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(593)Calculation not allo-
wed in multiple assignment
```

7.7.33 E 0033: Can not evaluate the address of a bit

This error occurs when an attempt is made to determine the address of individual bits. This of course is not possible. However, the address of the variable (BdintVar in the above example) can be determined and using a pointer loaded with the address, the bit can be accessed.

Example

```
FUNCTION Class0::E0033
  DintVar := #BdintVar.1;
END_FUNCTION

Compiler: [ERROR] E 0033
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(599)Can not evaluate the
address of a bit
```

7.7.34 E 0034: Wrong data type for index

This error occurs when no whole number value is used as an array index, rather a REAL number as in the example above. The same error message is generated when REAL value is replaced with a string (e.g. "MyString").

Example

```
FUNCTION Class0::E0034
  ArrVar[RealVar] := 33;
END_FUNCTION

Compiler: [ERROR] E 0034
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(605)Wrong data type for
index
```

7.7.35 E 0035: Only '=' and '<>' are possible

This error occurs when structures are compared in which "like" or unlike" is not queried. When two structures are compared, the result can state whether the content of the structures are like or the content of one is larger than the other (see example).

Example

```
FUNCTION Class0::E0035
  if StructVar >= StructVar2 then
  end_if;
END_FUNCTION

Compiler: [ERROR] E 0035
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(611)Only '=' and '<>' are
possible
```

7.7.36 E 0036: UNUSED

This error message is currently not used.

7.7.37 E 0037: UNUSED

This error message is currently not used.

7.7.38 E 0038: Duplicate case '%s'

This error is generated when a case instruction tests the same step twice. An error also occurs when constants are used.

Example

```
FUNCTION Class0::E0038
  case DintVar of
  0:
  0:
  end_case;
END_FUNCTION
```

```
Compiler: [ERROR] E 0038
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(632)Duplicate case '0'
```

7.7.39 E 0039: Variable is not a pointer

This error occurs when a variable is used as pointer although it was not defined as one. In the example above, DintVar is a normal variable of type DINT.

Example

```
FUNCTION Class0::E0039
  DintVar$ClassX::GlobVirtMeth();
END_FUNCTION
```

```
Compiler: [ERROR] E 0039
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(639)Variable is not a
pointer
```

7.7.40 E 0040: Undefined function '%s'

This error occurs when a function is called that is unknown in the corresponding context (e.g. class files).

Example

```
FUNCTION Class0::E0040
  UnknownMethod(nil, 0, 0);
END_FUNCTION

Compiler: [ERROR] E 0040
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(646)Undefined function
'UnknownMethod'
```

7.7.41 E 0041: Missing parameter '%s'

This error is generated when too few parameters are given while calling a function. In the above example, the function expects three parameters however only two were given.

Note:
If a function with n parameters is called with only n-1 parameters and no error is returned, the last parameter has a default value.

Example

```
FUNCTION Class0::E0041
  _memset(pDintVar, 0);
END_FUNCTION

Compiler: [ERROR] E 0041
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(652)Missing parameter
'cntr'
```

7.7.42 E 0042: Extra parameter '%s'

This error occurs when too many parameters are given when calling a function. In the example above, the function expects three parameters, however, four were given.

Example

```
FUNCTION Class0::E0042
  _memset(pDintVar, 0, 30, 0);
END_FUNCTION
```

```
Compiler: [ERROR] E 0042
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(658)Extra parameter '0'
```

7.7.43 E 0043: UNUSED

This error message is currently not used.

7.7.44 E 0044: Function '%s' has no output parameter

This error is generated when the return value is assigned to a variable although the function has no return value.

Example

```
FUNCTION Class0::E0044
  DintVar := ToDerive.GlobVirtMeth();
END_FUNCTION
```

```
Compiler: [ERROR] E 0044
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(664)Function 'GlobVirt-
Meth' has no output parameter
```

7.7.45 E 0045: Class '%s' is not defined

This error is generated when an unknown name is used where a class is expected. In the above example, the class file of "Class0" is accessed.

Example

```
FUNCTION ClassXXX::E0045
END_FUNCTION

Compiler: [ERROR] E 0045
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(669)Class 'ClassXXX' is
not defined
```

7.7.46 E 0046: Only constant expressions allowed

This error message is triggered when tables don't contain all constant expressions. In the example above, a variable is included.

Example

```
FUNCTION_TAB Class0::E0046
  0, 0, 0, DintVar
END_FUNCTION

Compiler: [ERROR] E 0046
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(679)Only constant expres-
sions allowed
```

7.7.47 E 0047: STRUCT '%s' without component

This error occurs when a structure is defined without subelements.

Example

```
TYPE
  NoCompStruct  : STRUCT
  END_STRUCT;
END_TYPE

Compiler: [ERROR] E 0047
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(684)STRUCT 'NoCompStruct'
without component
```

7.7.48 E 0048: Bit is not allowed

This error is generated when bits are used in AWL instructions in which no bits are allowed. In the example above, individual bits cannot be shifted to the left (shift instructions function with 1, 2 or 4 byte data lengths).

Example

```
FUNCTION AWL Class0::E0048
  SHL.B   BdintVar.1, 2
END_FUNCTION
```

```
Compiler: [ERROR] E 0048
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(692)Bit is not allowed
```

7.7.49 E 0049: Expected value is 0 or 1

This error occurs when the result of a comparison must be 0 or 1; the query of a bit in the example above is 2.

Example

```
FUNCTION AWL Class0::E0049
  CMP.B   BdintVar.1, 2
END_FUNCTION
```

```
Compiler: [ERROR] E 0049
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(698)Expected value is 0 or
1
```

7.7.50 E 0050: Instruction is not allowed

This error occurs when AWL instructions are used which are no longer supported in LASAL.

Example

```
FUNCTION AWL Class0::E0050
  TMR  DintVar
END_FUNCTION

Compiler: [ERROR] E 0050
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(704)Instruction is not
allowed
```

7.7.51 E 0051: Unknown command '%s'

This error is generated when unknown AWL instructions are used. In the above example, the correct syntax should read "S.EAX DintVar".

Example

```
FUNCTION AWL Class0::E0051
  SAVE.EAX  DintVar
END_FUNCTION

Compiler: [ERROR] E 0051
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(710)Unknown command 'SAVE'
```

7.7.52 E 0052: Invalid register

This error occurs when a register is used in an AWL instruction where a different register is expected. In the example above, the "CX" is expected.

Example

```
FUNCTION AWL Class0::E0052
  ROL.AX  DX
END_FUNCTION

Compiler: [ERROR] E 0052
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(716)Invalid register
```

7.7.53 E 0053: Address missing

This error is generated with an incomplete AWL instruction in which the value is not defined.

Example

```
FUNCTION AWL Class0::E0053
  INC.D
END_FUNCTION
```

```
Compiler: [ERROR] E 0053
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(722)Address missing
```

7.7.54 E 0054: Syntax error

This error occurs with an AWL instruction in which at least one element is unknown. In the example above, the instruction should increment an unknown variable, which leads to the corresponding error. Solutions for syntax errors in AWL instructions can be found in the help index for the respective error. Every AWL error is described with an example in the help index.

Example

```
FUNCTION AWL Class0::E0054
  INC.D  UnknownVariable
END_FUNCTION
```

```
Compiler: [ERROR] E 0054
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(779)Syntax error
```

7.7.55 E 0055: Invalid address %s %s

This error occurs when the value used in conjunction with the respective AWL instruction is not valid. In the example above, an attempt is made to use the clear instruction in conjunction with indirect addressing. The clear instruction is, however, can only be used for registers (e.g. CLR EDX).

Example

```
FUNCTION AWL Class0::E0055
  CLR    (EDX)
END_FUNCTION

Compiler: [ERROR] E 0055
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(785)Invalid address
```

7.7.56 E 0056: Not the same data length

This error is generated when the data length specified by the instruction (Byte in the example above) does not agree with the specified value (in the example above, a variable of type DINT).

Example

```
FUNCTION AWL Class0::E0056
  L.B    DintVar, 0
END_FUNCTION

Compiler: [ERROR] E 0056
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(791)Not the same data
length
```

7.7.57 E 0057: Reserved system call

SYS instructions are no longer supported in LASAL and may not be used.

Example

```
FUNCTION AWL Class0::E0057
  SYS    90     //gilt auch für Structured Text
END_FUNCTION

Compiler: [ERROR] E 0057
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(797)Reserved system call
```

7.7.58 E 0058: N, P and R require BIT-address and bit processor available

This error occurs when instruction forms for AWL methods are used where a bit processor is required.

Example

```
FUNCTION AWL Class0::E0058
  AND.B P BdintVar.1, 1
END_FUNCTION
```

```
Compiler: [ERROR] E 0058
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(803)N, P and R require
BIT-address and bit processor available
```

7.7.59 E 0059: UNUSED

This error message is currently not used.

7.7.60 E 0060: Use SI or DI, not BX

The AWL instruction you are using cannot use the BX register. Please use SI or DI instead.

7.7.61 E 0061: Invalid register. Use SI, DI, BX or BP

The AWL instruction you are using cannot be used with the given register, Please use SI, DI, BX or BP.

7.7.62 E 0062: Segment register not allowed

This error occurs when segment registers are used although the respective AWL instruction doesn't allow it.

Example

```
FUNCTION AWL Class0::E0062
  IMUL  ES
END_FUNCTION

Compiler: [ERROR] E 0062
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(817)Segment register not
allowed
```

7.7.63 E 0063: Missing ','

This instruction is generated when a comma between to AWL operands is forgotten.

Example

```
FUNCTION AWL Class0::E0063
  SHRD.EAX  EBX CX
END_FUNCTION

Compiler: [ERROR] E 0063
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(823)Missing ','
```

7.7.64 E 0064: Missing constant

This error is generated when a constant value is expected in an AWL instruction but a register or variable is given instead. In the example above, EBX must be replaced with a constant expression.

Example

```
FUNCTION AWL Class0::E0064
  IMUL.EAX  DintVar,  EBX
END_FUNCTION

Compiler: [ERROR] E 0064
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(829)Missing constant
```

7.7.65 E 0065: Register is expected

This error occurs when a register is expected in an AWL instruction but a constant value or variable is give instead. In the example above, "3600" must be replaced with "ECX".

Example

```
FUNCTION AWL Class0::E0065
  idiv 3600
END_FUNCTION

Compiler: [ERROR] E 0065
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(835)Register is expected
```

7.7.66 E 0066: Division by 0

This error message is generated when a division by 0 occurs calculation with only constant values. Here, it is important that this error only occurs when all values in the calculation are constants.

Example

```
FUNCTION Class0::E0066
  #define KONST1 1
  #define KONST2 0
  IF 1 = KONST1/KONST2 THEN
  END_IF;
END_FUNCTION

Compiler: [ERROR] E 0066
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(837)Division by 0
```

7.7.67 E 0067: Label '%s' already exists

This error occurs when a label is used twice in a function (Structured Text of AWL).

Example

```
FUNCTION Class0::E0067
LABEL1:
LABEL1:
END_FUNCTION
```

```
Compiler: [ERROR] E 0067
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(849)Label 'LABEL1' already
exists
```

7.7.68 E 0068: NOT 0/1 only inverts the least significant bit. Use NOT (FALSE/TRUE) instead.

This warning is generated when the instruction "NOT 0" or "NOT 1" is assigned. With this technique, only the lowest bit is inverted. This means that 0 is changed to 1 and not 16#FFFFFFFF.

Example

```
FUNCTION Class0::E0068
  DintVar := DintVar AND NOT 1;
END_FUNCTION
```

```
Compiler: [WARN] E 0068
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(855)NOT 0/1 only inverts
the least significant bit. Use NOT (FALSE/TRUE) instead.
```

7.7.69 W 0069: Condition is always %s

This warning is given when a condition is always **true**. In the example above, the If-query has only constant expressions so the condition is always **true**. The warning is only output when "ConditionConstant" is set in the compiler settings.

Example

```
FUNCTION Class0::W0069
  #define KONSTVAL1 1
  #define KONSTVAL2 2
  if (KONSTVAL1 < KONSTVAL2) then
  end_if;
END_FUNCTION

Compiler: [WARN] W 0069
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(863)Condition is always
TRUE
```

7.7.70 W 0070: Possibly mixing of '%s' and '%s'

This warning is generated when an If-instruction is mixed with "AND" or '&', or "OR" wit '|'. When several conditions are to be linked in an If-instruction logically, the key characters '&' and '|' should be used.

Example

```
FUNCTION Class0::W0070
  if (DintVar = 0) AND (UdintVar = 1) then
  end_if;
END_FUNCTION

Compiler: [WARN] W 0070
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(870)Possibly mixing of '&'
and 'AND'
```

7.7.71 W 0071: Variable '%s' is also used as type name: '%s'

This warning occurs when the variable has the same name as a type and it does not appear clearly in the context of the instruction whether the type or the variable is meant.

Example

```
TYPE
  DintVar : USINT;
END_TYPE
FUNCTION Class0::W0071
  UdintVar := sizeof(DintVar);
END_FUNCTION
```

```
Compiler: [WARN] W 0071
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(880)Variable 'DintVar' is
also used as type name: '::DintVar'
```

7.7.72 W 0072: '%s' declared but never used

This warning is given when a local variable is not used in the function. This warning is only generated when the compiler setting "VariableNeverUsed" is set.

Example

```
FUNCTION Class0::W0072
VAR
  LokVar  : DINT;
END_VAR
END_FUNCTION
```

```
Compiler: [WARN] W 0072
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(887)'LokVar' declared but
never used
```

7.7.73 W 0073: Parameter '%s' is never used

This warning occurs when the transfer parameter is not used in the function. This warning is only generated when the compiler setting "ParameterNeverUsed" is set.

Example

```
FUNCTION Class0::W0073
VAR_INPUT
  in1 : DINT;
END_VAR
  DintVar := in1;
END_FUNCTION
```

```
Compiler: [WARN] W 0073
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(894)Parameter 'in1' is
never used
```

7.7.74 W 0074: '%s' assigned a value which is never used

This warning is given when a local variable is not used in the function. As in the example above one assignment to the local variable is not sufficient to eliminate the warning. This warning is only generated when the compiler setting "ValueNeverUsed" is set.

Example

```
FUNCTION Class0::W0074
VAR
  LokVar : DINT;
END_VAR
  LokVar := 1000;
END_FUNCTION
```

```
Compiler: [WARN] W 0074
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(906)'LokVar' assigned a
value which is never used
```

7.7.75 W 0075: Possible use of '%s' before definition

This warning is generated when a local variable is used before it was assigned a defined value or initialized. This warning is only generated when the compiler setting "VariableUninitialized" is set.

Example

```
FUNCTION Class0::W0075
VAR
  LokVar : DINT;
END_VAR
  DintVar := LokVar;
END_FUNCTION
```

```
Compiler: [WARN] W 0075
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(918)Possible use of 'Lok-
Var' before definition
```

7.7.76 W 0076: Return value '%s' is not defined

This warning is triggered when the return value is not set anywhere in the function. This warning is only generated when the compiler setting "RetValueUninitialized" is set.

Example

```
FUNCTION Class0::W0076
VAR_OUTPUT
  out1 : DINT;
END_VAR
END_FUNCTION
```

```
Compiler: [WARN] W 0076
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(927)Return value 'out1' is
not defined
```

7.7.77 W 0077: Define directive needs an identifier

This warning is generated by a "#define" instruction without the corresponding identifier (name).

Example

```
FUNCTION Class0::W0077
  #define
END_FUNCTION
```

```
Compiler: [WARN] W 0077
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(936)Define directive needs
an identifier
```

7.7.78 W 0078: Redefinition of '%s'

This warning is generated when a constant with a specified name is defined differently several times (see example above). When both definitions of the same value are assigned, no warning is given.

Example

```
FUNCTION Class0::W0078
  #define   W78KONST1 10
  #define   W78KONST1 20
END_FUNCTION
```

```
Compiler: [WARN] W 0078
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(943)Redefinition of
'W78KONST1'
```

7.7.79 W 0079: Bad preprocessor directive syntax

This warning occurs when the wrong syntax is used for preprocessor instructions. In the below example the instruction should read:

#pragma warning (disable:78);

Example

```
FUNCTION Class0::W0079
  #pragma warning (disable);
END_FUNCTION
```

```
Compiler: [WARN] W 0079
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(949)Bad preprocessor di-
rective syntax
```

7.7.80 W 0080: Misplaced %s directive

This warning is generated when too little or too many terminations for preprocessor instructions (#if, #ifdef, etc.) are available in the source code file. In the above example, a preprocess instruction is terminated with "#endif" although there is no instruction in this location.

Example

```
FUNCTION Class0::W0080
#endif
END_FUNCTION
```

```
Compiler: [WARN] W 0080
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(956)Misplaced ENDIF direc-
tive
```

7.7.81 E 0081: Unknown preprocessor directive

The preprocessor directive you typed is not expected and cannot be recognized. Check the Programming Help to see all supported preprocessor directives.

7.7.82 E 0082: Incorrect argument for SEGMENT or OFFSET

This error is generated with the use of the "Seg()" and "Off()" when the argument is not a pointer. Both "Seg()" and "Off()" instructions have no meaning in LASAL since indirect addressing are no longer processed with segment and offset (FlatMemory).

Example

```
FUNCTION Class0::E0082
  UdintVar := off(this);
  UdintVar := seg(this);
END_FUNCTION
```

```
Compiler: [ERROR] E 0082
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(967)Incorrect argument for
SEGMENT or OFFSET
Compiler: [ERROR] E 0082
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(968)Incorrect argument for
SEGMENT or OFFSET
```

7.7.83 E 0083: Extra parameter '%s' for macro '%s'

This error occurs when a macro is assigned one or more parameters too many. In the example below, the macro requires three parameters, yet four were assigned.

Example

```
FUNCTION Class0::E0083
  #define MyMemSet(a,b,c) _memset(a,b,c)
  MyMemSet(pTmp, 16#FF, 20, 0);
END_FUNCTION
```

```
Compiler: [ERROR] E 0083
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(975)Extra parameter '0'
for macro 'MYMEMSET'
```

7.7.84 E 0084: Missing parameter for macro '%s'

This error occurs when a macro is assigned one or more parameters to little. In the example below, the macro requires three parameters, yet only two were assigned.

Example

```
FUNCTION Class0::E0084
  #define MyMemSet(a,b,c) _memset(a,b,c)
  MyMemSet(pTmp, 16#FF);
END_FUNCTION
```

```
Compiler: [ERROR] E 0084
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(982)Missing parameter for
macro 'MYMEMSET'
```

7.7.85 E 0085: Different redefinition of function '%s' because of '%s'

This error occurs when a function header in the implementation section is different from that in declaration section. This happens when the method was placed in the tree in implementation then a parameter was added but no "Update Method" was executed, or the changes were not made manually. In the example below, the method in the tree has no parameters, in the implementation section, however, the variable "in1" was added.

Example

```
FUNCTION Class0::E0085
VAR_INPUT
  in1 : DINT;
END_VAR
END_FUNCTION
```

```
Compiler: [ERROR] E 0085
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(992)Different redefinition
of function 'E0085' because of 'in1'
```

7.7.86 E 0086: UNUSED

This error message is currently not used.

7.7.87 E 0087: Function '%s' already has a body

This error is generated when a function is implemented in a source code file two or more times.

Example

```
FUNCTION Class0::E0087
END_FUNCTION

FUNCTION Class0::E0087
END_FUNCTION
```

```
Compiler: [ERROR] E 0087
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1000)Function
'_E0087@Class0' already has a body
```

7.7.88 E 0088: Linker: Unresolved external '%s' in "%s"

This error is generated when a forward declaration is not found in any Lob files when linking. In the example above, the "ForwardDeclaredMeth" function in the class file "Class0" is forward declared. This means it is assumed that a function with this name is in another source code file of the project. If this is not the case the error is triggered.

Example

```
FUNCTION ForwardDeclaredMeth;
FUNCTION Class0::LinkerUnresolvedExternal
  ForwardDeclaredMeth();
END_FUNCTION
```

```
Linker: [ERROR] Unresolved external '_ForwardDeclaredMeth' in
"D:\Temp\CompilerMessages\Class\Class0\Class0.lob"
```

7.7.89 E 0089: UNUSED

This error message is currently not used.

7.7.90 E 0090: Different redefinition of variable '%s'

This error occurs when a declaration with VAR_GLOBAL and VAR_EXTERNAL overlap, although the affected variable is declared differently. In the example below, the affected variable is declared as UDINT in VAR_GLOBAL and in VAR_EXTERNAL as DINT.

Note:
To illustrate, the VAR_GLOBAL and VAR_EXTERNAL blocks in the example below are found in the source code file. Usually both blocks are found in different files (e.g. VAR_GLOBAL in a class file, VAR_EXTERNAL in a global Header file).

Example

```
VAR_GLOBAL
  GlobVarErr0090  : UDINT;
END_VAR
VAR_EXTERNAL
  GlobVarErr0090  : DINT;
END_VAR
FUNCTION Class0::E0090
END_FUNCTION

Compiler: [ERROR] E 0090
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1010)Different redefiniti-
on of variable 'GlobVarErr0090'
```

7.7.91 E 0091: File '%s' could not be generated

This error is generated when a file cannot to be created. A possible cause for this could be that either the lob file or the whole project is write protected.

Example

```
Compiler: [ERROR] E 0091 ""(0)File
'D:\Temp\CompilerMessages\Class\Class0\Class0.lob' could not be generated
```

7.7.92 E 0092: Error reading file '%s'

The specified file name could not be found during a read operation. This may be caused by the operating system of the PLC.

7.7.93 E 0093: '%s' is not a Sigmatek OBJ file

The object file could not be found or failed to convert. When compiling C files, unsupported C functions can cause this problem. It can also be caused by the PLC operating system.

7.7.94 E 0094: '%s' is not a Sigmatek DEF file

This error message is currently not used. It can also be caused by the PLC operating system.

7.7.95 E 0095: Wrong version of '%s'

This error message is currently not used in LASAL CLASS 2. It can also be caused by the PLC operating system.

7.7.96 E 0096: UNUSED

This error message is currently not used.

7.7.97 E 0097: '%s' is not a channel

The specified element is not a channel but was used as one. A Client or Server Channel was expected.

7.7.98 E 0098: Label '%s' is not defined

This error occurs when a label in an instruction does not exist.

Example

```
FUNCTION Class0::E0098
  Goto UNDEFINED_LABEL;
END_FUNCTION
```

```
Compiler: [ERROR] E 0098
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1040)Label 'UNDEFI-
NED_LABEL' is not defined
```

7.7.99 E 0099: Redefinition of function '%s' in "%s"(%d) and in "%s"(%d)

This error occurs when a global function is implemented in two different source code files.

Example

```
Linker: [ERROR] Redefinition of function '_RedefFunction' in
"D:\Temp\CompilerMessages\Class\Class0\ERR0099.lob"(1) and in
"D:\Temp\CompilerMessages\Class\Class0\Class0.lob"(1216)
```

7.7.100 E 0100: Register as parameter is not allowed for __cdecl functions

This error message occurs when a register was assigned for a parameter, however, the "__CDECL" function was also declared. This is not allowed!

Example

```
FUNCTION __CDECL Class0::E0100
VAR_INPUT
  in1 (EBX)   : DINT;
END_VAR
END_FUNCTION

Compiler: [ERROR] E 0100
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1048)Register as parameter
is not allowed for __cdecl functions
```

7.7.101 E 0101: Only one return value is allowed for __cdecl functions

This error is generated when multiple return parameters were defined for a function and the "__CDECL" function was also declared. This is not allowed!

Example

```
FUNCTION __CDECL Class0::E0101
VAR_OUTPUT
  out1        : DINT;
  out2   : DINT;
END_VAR
  out1 := READY;
END_FUNCTION
```

```
Compiler: [ERROR] E 0101
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1065)Only one return value
is allowed for __cdecl functions
```

7.7.102 E 0102: '%s' is not for Intel 386 or later processors

A binary image for a different machine type was found while decoding a compiled C file. Only i386 binaries can be used. It may be caused by the PLC OS.

7.7.103 E 0103: '%s' contains not supported type of relocation '%s'

A C object file uses an addressing mode that is not supported by the PLC and could not be translated. It may be caused by the PLC OS.

7.7.104 E 0104: Not supported Symbol Table entry in '%s'

Failed to decode the symbol table for the current C object file. It may be caused by the PLC OS.

7.7.105 E 0105: Could not execute compiler '%s'

This error occurs with the compilation of C files when a compiler is available in the directory but is not executable.

7.7.106 E 0106: Variable '%s' in "%s" and variable '%s' in "%s" use same address

This error is generated when a global variable is assigned to a fixed memory location and overlaps with another global variable. In the example below, the "SameAddressVar" variable overlaps with the operating system variable "ops" (ops is defined in the loader).

Example

```
VAR_GLOBAL
  SameAddressVar       AT % D 00003       : DINT;
END_VAR
```

```
Linker: [ERROR] Variable 'OPS' in
"d:\LASAL2\Source\Runtime\Library\Loader\Source\Loader.lob" and variable '
SameAddressVar' in "D:\Temp\CompilerMessages\Class\Class0\Class0.lob" use
same address
```

7.7.107 E 0107: Missing function 'main'

This error occurs when the „main" function of the loader is not found in any Lob files. This can happen when the "Loader.lob" linked to the project file is mistakenly deleted.

Example

```
Linker: [ERROR] Missing function 'main'
```

7.7.108 E 0108: Standard methods are already defined

The standard methods of a class are already defined. You are attempting to define a second time. It may be caused by the PLC OS.

7.7.109 E 0109: Using pointer to undef type '%s'

You are using a pointer for an unknown element. This problem is common to forward declared elements. Please use `#pragma using` to include the class that contains the definition of this element.

7.7.110 E 0110: Incorrect use of channel '%s'

This error occurs when an attempt is made to declare variables or clients as servers. Structures with the prefix "CltCh", "CltChCmd", "SvrCh" and "SvrChCmd" are reserved.

Example

```
FUNCTION Class0::E0110
VAR
  Tmp : CltCh_DINT;
END_VAR
END_FUNCTION
```

```
Compiler: [ERROR] E 0110
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1095)Incorrect use of
channel 'CltCh_DINT'
```

7.7.111 E 0111: Too many errors

This error occurs when the maximum number of errors is greater than 100 or the error limit defined in the compiler settings per module is exceeded ("MaxModuleErrorCount").

Example

```
Compiler: [ERROR] E 0111
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1075)Too many errors
```

7.7.112 E 0112: '%s' is not a LIB file

Not used in LASAL CLASS 2. The error is caused by the PLC OS.

7.7.113 E 0113: Cannot use '%s' in LIB '%s'

Not used in LASAL CLASS 2. The error is caused by the PLC OS.

7.7.114 E 0114: Base Class '%s' undefined

This error is generated when an unknown base class is specified when declaring a class. This can only happen in LASAL 2 when user classes are created in source code files directly and not through the LASAL 2 graphical user interface. For real LASAL 2 classes, the availability of base classes is tested while the project is loading.

Example

```
MyNewClass : CLASS
: MyBaseClass
END_CLASS;
```

```
Compiler: [ERROR] E 0114
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1106)Base class 'MyBaseC-
lass' undefined
```

7.7.115 E 0115: '%s' is not a base class for '%s'

Not used in LASAL CLASS 2. The error is caused by the PLC OS.

7.7.116 E 0116: Redefinition of variable '%s' in '%s' and '%s'

This error occurs when a global variable is declared in two different source code files (both with "VAR_GLOBAL"). The declaration with "VAR_GLOBAL" can only be used in one location. If the variable is required in another source code file, it must be declared as "VAR_EXTERNAL".

Example

```
Linker: [ERROR] Redefinition of variable 'RedefVar' in
"D:\Temp\CompilerMessages\Class\Class0\Class0.lob" and
"D:\Temp\CompilerMessages\Class\Class0\ERR0116.lob"
```

7.7.117 E 0117: Size of channel data element must be 4 Bytes! %s

This error is generated when a server or client is assigned a type whose data length is not 4 bytes.

Example

```
Compiler: [ERROR] E 0117
"D:\Temp\CompilerMessages\Include\CHANNELS.H"(41)Size of channel data ele-
ment must be 4 Bytes!
```

7.7.118 E 0118: Private Function '%s' of class '%s' cannot be called here.

This error occurs when a private method is called via an object client. Only the call of global functions is allowed via object clients.

Example

```
FUNCTION Class0::E0118
  ObjClient.PrivMeth();
END_FUNCTION
```

```
Compiler: [ERROR] E 0118
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1115)Private Function
'PrivMeth' of class 'Derive' cannot be called here!
```

7.7.119 E 0119: Function '%s' is declared as a table. Tables cannot be called.

This error occurs at call of a function, whereby the function is a table. That's not allowed, as a table includes no executable code.

Example

```
FUNCTION Class0::E0119
  Table01();
END_FUNCTION

FUNCTION TAB Class0::Table01
01,02,03
END_FUNCTION
```

```
Compiler: [ERROR] E 0119
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1121)Function 'Table01' is
declared as a table. Tables cannot be called.
```

7.7.120 E 0120: Different redefinition of function '%s'

This message is issued by the Linker and indicates that a variable with the same name is defined in more than one module. Please note that global symbols needs to have unique names in a program and the error indicates that a name was used that already exists in another module or even a library.

7.7.121 E 0121: Different redefinition of variable '%s'

Not used in LASAL CLASS 2. The error is caused by the PLC OS.

7.7.122 E 0122: Linker is set for case sensitive linking: '%s'

This error is issued by the linker and indicates that a function name conflicts with another function name and the only difference is the character case. Note that ST is a case insensitive programming language.

7.7.123 W 0123: Function '%s' is declared as VIRTUAL in base class '%s'

This warning is triggered when a global method in the base class is declared as "Virtual" and the overwritten method in the derived class is not. This can be corrected by deleting the overwritten method in the derived class (save the implemented methods first!). The method can then be overwritten new. The properties in the base method are then assumed.

Example

```
Compiler: [WARN] W 0123
"D:\Temp\CompilerMessages\Class\Derive\Derive.st"(73)Function 'GlobVirtMeth'
is declared as VIRTUAL in base class 'Base2'
```

7.7.124 E 0124: Loader error '%s'

This error is used to forward error messages coming from the loader library. Please refer to the specific message and Loader documentation.

7.7.125 E 0125: RT-DataBase error: '%s'

Not used in LASAL CLASS 2. The error is caused by the PLC OS.

7.7.126 I 0126: Trace '%s': '%s'

This message code forward a trace message from the operating system.

7.7.127 E 0127: No memory left for variable '%s'

This error is generated when a local variable of a function has reserved too much memory. In the example below, 65533 bytes (USINT is from 0 to 65532) for a local array was reserved, the limit is however 65532 for the sum of all local variables.

Example

```
FUNCTION Class0::E0127
VAR
  LokArr  : ARRAY [0..65532] OF USINT;
END_VAR
END_FUNCTION
```

```
Compiler: [ERROR] E 0127
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1150)No memory left for
variable 'LokArr'
```

7.7.128 E 0128: File '%s' was compiled with a different alignment setting (%d)

This error occurs at existence of a linked class in the project, whose source project has been compiled with other alignment settings as the current project.

Example

```
Linker: [ERROR] File
'd:\Lasal2\Source\Runtime\Library\Memory\Class\Ram\Ram.lob' was compiled
with a different alignment setting (4)
```

7.7.129 W 0129: Missing '%s' instead of '%s'

Missing a **#pragma pop** to match a previous defined **#pragma pack**.

7.7.130 W 0130: '%s' '%s' '%s'

This error code is used to issue the message resulting from a **#pragma message** directive.

7.7.131 E 0131: %s

Under this error number, complete error messages and warnings generated during compilation of C files are given.

7.7.132 E 0132: The size of class '%s' is greater than 64 KByte

This error is generated when the memory requirement of a class exceeds 64 Kbytes. The memory requirement of a class includes variables, clients and servers. A client or server requires 12 bytes each.

Example

```
Compiler: [ERROR] E 0132
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(375)The size of class
'Class0' is greater than 64 kByte
```

7.7.133 E 0133: The file '%s' does not exist

Not used in LASAL CLASS 2. The error is caused by the PLC OS.

7.7.134 W 0134: Label '%s' is declared but never used

This warning is given when labels are defined but not used.

Caution!

Unknown constants with a case instruction are also interpreted as unknown labels (see example above). This warning is only given when the "LabelNeverUsed" property is enabled in the compiler settings.

Example

```
FUNCTION Class0::E0134
HELLO:

  case DintVar of
  START_CONDITION:
  STOP_CONDITION:
  end_case;

END_FUNCTION
```

```
Compiler:                    [WARN]                    W                    0134
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1160)Label 'HELLO' is dec-
lared but never used
Compiler:                    [WARN]                    W                    0134
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1163)Label
'START_CONDITION' is declared but never used
Compiler: [WARN] W 0134
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1164)Label
'STOP_CONDITION' is declared but never used
```

7.7.135 E 0135: The interpreter program is too long. The maximum code size is 64 Kbytes.

This error occurs when an interpreter program exceeds the code size of 64 Kbytes. To correct this, the interpreter functions should be divided over several interpreter files.

Example

```
Compiler: [ERROR] E 0135 ""(0)The interpreterprogram is too long. The maxi-
mum codesize is 64 kBytes.
```

7.7.136 E 0136: '%s' not found!

Not used in LASAL CLASS 2. The error is caused by the PLC OS.

7.7.137 W 0137: (Private) data element '%s' of class '%s' cannot be accessed here.

This warning is given when a private variable is accessed over an object client of a class. This is possible, however, it breaks through the capsuling of object-oriented programming and is therefore not recommended. The communication with other classes should be established over global methods or the data exchange between clients and servers only. This warning is triggered only when the "**PrivateDataAccess**" property in the compiler settings is enabled.

Example

```
FUNCTION Class0::E0137
  ObjClient.PrivVar := 1;
END_FUNCTION

Compiler: [WARN] W 0137
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1172)(Private) data ele-
ment 'PrivVar' of class 'Derive' cannot be accessed here.
```

7.7.138 W 0138: Class::Channel.Method results in a virtual call. Use Class::Channel::Method to call the method directly

The syntax using the [**.**] produces a virtual call. To call the method directly please use the [**::**] operator.

7.7.139 W 0139: #Objch/ObjPtr.Channel returns the address of the data element.Use #Obj.Ch.pMeth for Servers or #Obj.Ch.pCh for Clients target the Channel's address

A data structure is allocated for each Server and client. Using the operator [#] returns the address of that control structure. The message describes the correct form to obtain the address of the server or client target function.

7.7.140 W 0140: '%s' has been compiled with an other version of LASAL

Object file was created by a different compiler. Some changes on the object format could cause incompatibilities. Check Compiler changes documentation for more details.

7.7.141 W 0141: '%s'

This warning code is used to format warning messages from the C compiler to the output window. Check C compiler documentation to handle the the message.

7.7.142 W 0142: skipping #define '%s' because of endless recursion.

This warning is given when a preprocessor instruction cannot be completed because of an endless loop (the constants defined were used in the definition themselves). The constant "MYKONST" in the example above is ignored. If the constant were used in another section of code, an error would be generated in LASAL 2 (e.g. "Undefined symbol 'MYKONST'").

Example

```
FUNCTION Class0::E0142
  #define MYKONST MYKONST+1
END_FUNCTION
```

```
Compiler: [WARN] W 0142
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1190)Skipping #define 'MY-
KONST' because of endless recursion.
```

7.7.143 E 0143: Name collision with define '%s' - not replaced.

This error occurs when a '#define' has the same name as a class element (Client, Server, Methode, Variable, etc.). If the '#define' is already known at compilation of the class declaration, so this error will be declared. That's the case when the '#define' is declared in the Define-block of the class or in a global header file.

Example

```
Compiler: [WARN] W 0142
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(58)Name collision with
define 'Variable2' - not replaced.
```

7.7.144 E 0144: Offset of element '%s' exceeds 64 kB limit within struct '%s'

This error occurs when subelements of a structure are so great that they cannot be addressed anymore within the structure with a 2 byte offset. In the example

above the subelements "Array40k" and "Array30k" already need 70.000 bytes, so that "Array20k" is no longer addressable with 2 byte (65536 possibilities). Shifting the "Array40k" element to the last position of the structure would put things right in this case.

Example

```
TYPE
  BigStruct : STRUCT
        Array40k : ARRAY [0..40000] OF USINT;
        Array30k : ARRAY [0..30000] OF USINT;
        Array20k : ARRAY [0..20000] OF USINT;
  END_STRUCT;
END_TYPE

Compiler: [ERROR] E 0144
"D:\Temp\CompilerMessages\Class\Class0\Class0.st"(1193)Offset of element 'Array20k' exceeds 64
```

7.7.145 W 0145: Identical redefinition of '%s'

This warning is triggered when 2 #defines with the same name have the same value.

Example

```
#define MaxTemp 20        //possibly included through Headerfile1.h
#define MaxTemp 20        //possibly included through Headerfile2.h
…

Compiler: [WARN] W 0145 "D:\LasalTraining\Intro\Class\Pump\Pump.st"(122) Redefinition of 'MAXTE
```

7.7.146 W 0146: Multiple comparisons within one instruction. This leads to a boolean evaluation

This warning is triggered, when several comparisons exist in one instruction. This leads to a booloean evaluation.

Example

```
FUNCTION Class0::Method
VAR
        my_var : DINT;
```

```
END_VAR

  my_var := 2;
  IF 1 < my_var < 5 THEN
    my_var := 10;
  END_IF;

END_FUNCTION

Compiler: [WARN] W 0146 "D:\CompilerMessag-es\Class\Class0\Class0.st"(113) Multiple comparis
```

7.7.147 W 0147: '%s' is not supported by this architecture

This message is will be generated by the ARM compiler, when trying to use a feature specific do the X86 architecture.

7.7.148 W 0148: Unaligned access detected. REAL / LREAL elements have to start on 4 Byte-aligned addresses

This error is generated by the ARM compiler when a REAL/LREAL member is accessed without a four byte alignment. The ARM architecture imposes this alignment for those data-types.

7.7.149 W 0149: Using X86 register in ARM code

An Intel register name was specified for a ARM program, compiler will proceed with an implicit conversion to continue.

7.7.150 W 0150: Missing '%s' instead of '%s'

This warning is triggered when the semicolon at the end of a line is missing.

If the error warning is double clicked, a dialog opens in which whether the missing semicolon should be added for the displayed line only or the throughout the program can be specified.

Example

```
MyVar1 := 1
State := READY;
…

Compiler: [WARN] W 0150 "D:\Projekte\Test_SFC_NEU\Class\Class0\Class0.st"(108) Missing ';' i
```

7.7.151 W 0151: Class ‘%s’ was assigned to variable

This warning is triggered when a class was assigned a variable.

Example

```
MyVar := Class0;
...

Compiler: [WARN] W 0151 "D:\Projekte\Test_SFC_NEU\Class\Class0\Class0.st"(107) Class 'Class0' w
```

7.7.152 W 0152: Datatype ‘%s’ was assigned to variable

This warning is triggered when a class was assigned a data type.

Example

```
MyVar := MyDINT; (MyDINT is a class type of DINT)
...

Compiler: [WARN] W 0152 "D:\Projekte\Test_SFC_NEU\Class\Class0\Class0.st"(108) Datatype 'MyDINT
```

7.7.153 W 0153: Virtual function ‘%s’ is not declared as VIRTUAL in base class ‘%s’

This warning is triggered when a virtual method was not declared as VIRTUAL in the base class.

Example

```
Compiler: [WARN] W 0153 "D:\Projekte\GHE-Leerprojekt\Class\Class1\Class1.st"(57) Virtual functi
```

7.7.154 E 0154: Multiple declaration of enum ‘%s’. The enum has to be specified with the typescope

This error is triggered when an Enum item was declared multiple times. It must be specified with a type scope.

Example

```
Compiler: [ERROR] E 0154 "D:\Projekte\GHE-Leerprojekt\Class\Class1\Class1.st"(109) Multiple dec
```

7.7.155 E 0155: Multiple declaration of type ‘%s’. The type has to be specified with the classscope

This error is triggered when a type was declared multiple times. It must be specified with a class scope.

Example

```
Compiler: [ERROR] E 0155 "D:\Projekte\GHE-Leerprojekt\Class\Class1\Class1.st"(109) Multiple
```

7.7.156 W 0156: Stacksize ('%u' byte) of function ‘%s’ exceeds the warning level. Check the size of local variables. The value can be set in the compileroptions

This warning is triggered when the predefined size of the stack for the function was exceeded. In the compiler properties however, the stack size can be overwritten per function (the default value is currently 4000 bytes).

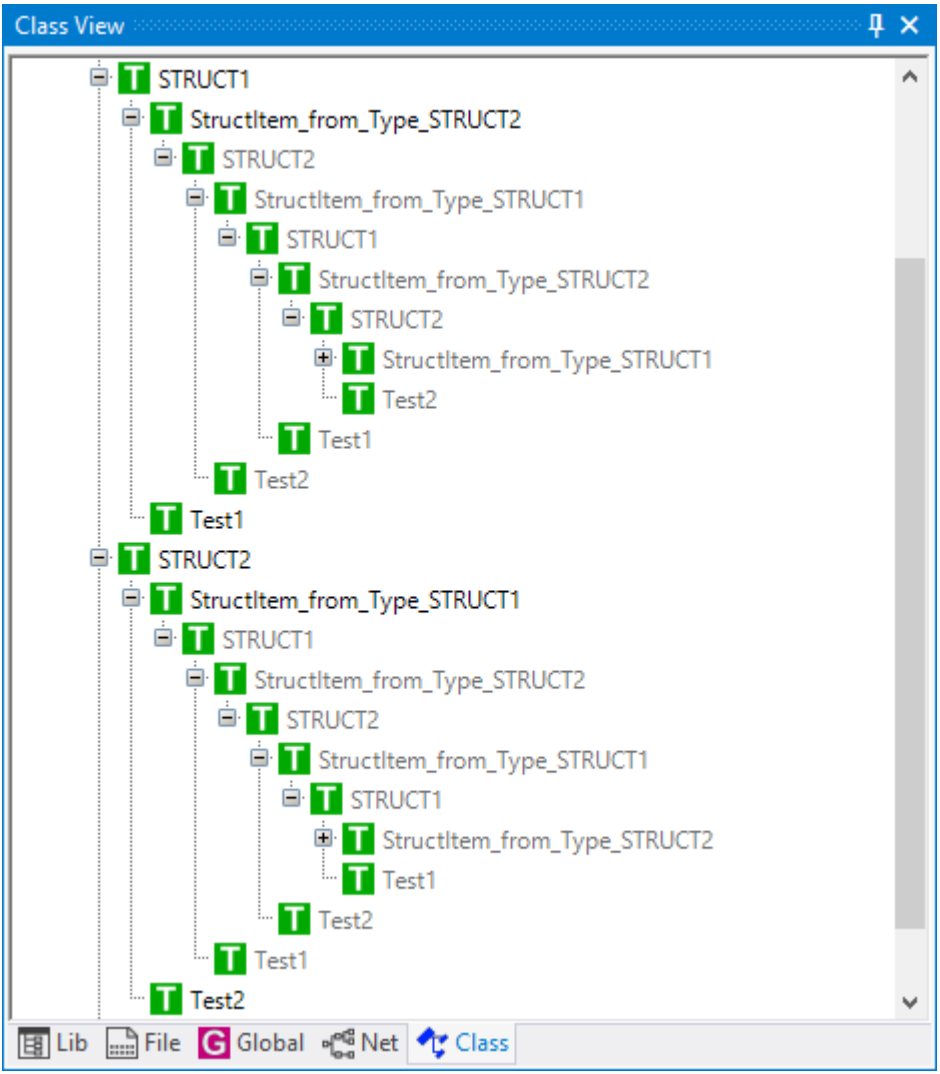
Example

```
Compiler: [WARN] W 0156 "D:\Projekte\Test W0156\Class\Class0\Class0.st"(110) Stacksize(12000
```

7.7.157 E 0157: Circular dependency in type ‘%s’ from ‘%s’

This error occurs when types are interdependent. The image below shows that when two structures are contained within one another, this leads to infinite nesting.

Example


Class View
STRUCT1
StructItem_from_Type_STRUCT2
STRUCT2
StructItem_from_Type_STRUCT1
STRUCT1
StructItem_from_Type_STRUCT2
STRUCT2
StructItem_from_Type_STRUCT1
Test2
Test1
Test2
Test1
STRUCT2
StructItem_from_Type_STRUCT1
STRUCT1
StructItem_from_Type_STRUCT2
STRUCT2
StructItem_from_Type_STRUCT1
STRUCT1
StructItem_from_Type_STRUCT2
Test1
Test2
Test1
Test2
Lib
File
Global
Net
Class

```
Compiler:       [ERROR]       E       0157       "D:\Projekte\GHE-
Leerprojekt\Class\Class0\Class0.st"(43) Circular dependency in type
'STRUCT2' from 'STRUCT1'
```

7.7.158 W 0158: Compiler-Bug detected, please update to compilerversion ‘%s’ or higher to fix the bug

This warning is triggered when a known compiler error was detected, which was already corrected in the new compiler version. Use the compiler version indicated in the error message or a newer one.

Example

```
Compiler: [ERROR] E 0155 "D:\Projekte\GHE-Leerprojekt\Class\Class1\Class1.st"(109) Compiler-Bu
```

7.7.159 E 0159: Negative Array limit ‘%s’

This error is triggered when in the declaration of an array, one or more negative numbers were used for the array limits. Since such valued cannot be directly entered, this situation can only occur in the limits declaration with Defines or expressions (ex. mGrenzwert -1). In this case, the user must ensure that the expression is never less than 0.

Example

```
Compiler: [ERROR] E 0159 "D:\Projekte\GHE-Leerprojekt\Class\Class0\Class0.st"(32) Negative arra
```

7.7.160 W 0160: Array of Struct with misaligned LREAL element

This warning is triggered in arrays of type “Struct” when the structure contains at least one LREAL element aligned to 4 bytes and the last element of the structure is not 4 bytes in size.

Example

```
Compiler: [ERROR] E 0155 "D:\Projekte\GHE-Leerprojekt\Class\Class1\Class1.st"(115) Array of St
```

7.7.161 W 0161: Possible syntax error in FOR-LOOP. Check for ">,>=,<,<=,<>,=" in the condition aera (to ...)

This warning occurs when a FOR-loop in the condition section (to ...) has a one of the characters '<', '=', or '>'.

Example

```
Compiler: [ERROR] E 0155 "D:\Projekte\GHE-Leerprojekt\Class\Class1\Class1.st"(109) Possible
```

7.7.162 E 0162: Table for object network '%s' not found

This error is generated when the Lob file of the corresponding object network is not available during linking.

Example

```
Linker: [ERROR] Table for object network 'Network0' not found
```

7.7.163 E 0163: Classtable '%s' not found

This error is generated when the Lob file of the corresponding class is not available during linking.

Example

```
Linker: [WARN] Classtable 'CLASS0::@CT_' not found
```

7.7.164 W 0164: C++ static constructors are not supported

C++ objects that have constructors and are instantiated as static objects create static constructors. The PLC is not able to call them properly, so they remain uninitialized.

Example

```
Linker: [WARN] C++ static constructors are not supported
```

7.7.165 W 0165: Name collision with define ‘%s’ replaces bitfield

Name collision with define ‚%s‘ replaces bitfield.

Example

```
TYPE
  BitStuff : BSINT
  [
    1 BAT_EMPTY,
    2 Bit2,
    3 Bit3,
    4 Bit4,
    5 Bit5,
    6 Bit6,
    7 Bit7,
    8 Bit8,
  ];
END_TYPE

#define BAT_EMPTY 5

FUNCTION Class0::Method
VAR
        my_var : BitStuff;
END_VAR

  my_var.BAT_EMPTY := 1;

END_FUNCTION
```

```
Compiler: [WARN] W 0165 "D:\CompilerMessages\Class\Class0\Class0.st"(136) Name collision with d
```

Please be aware, that a processor macro always has a higher priority than other definitions/elements! In the example above the line

```
my_var.BAT_EMPTY := 1;
```

is replaced with

```
my_var.5 := 1;
```

(the #define has a higher priority than the bit name). This is a valid allocation, but to bit 5 and not to bit 1, like probably expected.

7.7.166 E 0166: Incompatible types: Expression '%s' cannot be converted to '%s'

This error occurs, if the result of an expression is assigned to an incompatible type. Be aware, that the compiler setting "ExactTypeChecking" plays an important role here. If the mentioned compiler setting is set to "false", no error is generated in the below example, because the difference between the two types is only the sign.

An allocation of a result of the type REAL to a DINT variable always generates an error independently of the compiler setting, as the format of the two types is not compatible. Also allocations to a smaller data width (e.g. DINT to INT) always generate an error.

Note:
If a directly selected string (e.g. "MyString") is given in an allocation, this always corresponds to the type "^USINT".

Example

```
FUNCTION Class0::E0166
  DintVar := TO_UDINT(OS_GETMICROSEC() – UdintVar);
END_FUNCTION

Compiler: [ERROR] E 0166 "D:\Projekte\Test DEVSW-214\ ClassDEVSW_214.st"(138) Incompatible ty
```

7.7.167 W 0167: C file contains unresolved addresses: The program may fail to run

This message appears as a warning in compiler version 45 as above - after that the program execution will most likely be terminated. The reason for this message is that the C compiler generates unrecognizable addresses and the converter (which converts the C compilat to the *.lob file) fails. An additional message suggests the use of the newer compiler versions (from 46), since more C compiler constructs are recognized from this version on.

This error is an internal compiler error and should usually not appear at all. It was implemented as protection for unexpected cases.

Example

```
Compiler: [WARN] W 0167 "D:\Issues\DEVSW-539\Source\test.c"(0) C file contains unresolved ad
```

7.7.168 W 0168: Debugging of Variables on this C file may not work as expected

This warning only appears in compiler versions up to and including 45. Reason: the order of variables in memory does not match that in the debugger.

An additional message suggests the use of the newer compiler versions (from 46) as a solution.

Example

```
Compiler: [WARN] W 0168 "test.c"(0) Debugging of Variables on this C file may not work as expe
```

7.7.169 W 0169: Compiler optimization on this file may cause critical errors

This warning only appears in compiler versions up to and including 45. Reason: the order of the variables in memory does not match the order expected in the C-optimized code. Invalid allocations could stop the execution of the program.

An additional message suggests the use of the newer compiler versions (from 46) as a solution.

Example

```
Compiler: [WARN] W 0169 "test.c"(0) Compiler optimization on this file may cause critical erro
```

7.7.170 W 0170: '%s' cast of a '%s' type channel can lead to unexpected results

Since there were only channels of type DINT in older LASAL CLASS 2 versions, the conversion to DINT - as added under "Example" - was often used. This is not allowed for channels of type REAL, since this can invalidate data. The solution is to omit DINT conversion for REAL type channels.

Example

```
// Legacy Declaration: DINT:=Write(input:=DINT)
MyChannel.Write(input:=myRealVar$DINT);
Compiler: [WARN] W 0170 "Class/test/test.st"(215) 'DINT cast of a 'REAL' type channel can lead
```

If MyChannel is of the type REAL, the solution will look like this:

```
MyChannel.Write(input:=myRealVar);
```

7.7.171 W 0171: Accuracy of the constant value will get lost when applied to a single precision expression

If the compiler encounters a constant that is too precise for REAL (single precision) and assigned to such a variable, the following warning is issued:

```
Compiler: [WARN] W 0171 "test.st"(443) Accuracy of the constant value will get lost when appli
```

So that this warning is not output continuously when compiling, the check is only **limited to occurrences in the code**, it is also **not output for defines**.

Example

```
#define MY_PI 3.141592653589793
// This causes W 0171 because CONST has too much precision
// for the REAL variable
realVar := 3.141592653589793;
// No W 0171, since CONST came from a #define
realVar := MY_PI;permissive
```

The following very tolerant rules apply:

The warning is **only issued if**...

... the constant is **not defined by #define**,

... it contains **at least 11 significant digits** (leading zeros are not taken into account),

... it is **outside the range -5.19e+42 < x < 5.19e+42**

7.7.172 W 0172: Cannot resolve 'this' pointer for method 'NotMyBase-Class::MyVirtual'. Non-static methods will crash your program

If in a derived class a function of the same name **of another base class** is called by mistake - e.g. by a copy-paste error - the application may crash, because the THIS pointer may point to nothing. However, since such a call can make sense under certain circumstances, a warning is output instead of an error:

```
Compiler: [WARN] W 0172 "Class0.st"(22) Cannot resolve 'this' pointer for method 'NotMyBaseC
```

It is possible to switch off this warning. Either set ThisCallScope to false in the Compiler group of the project properties or enter #pragma warning(disable:172).

Example

```
FUNCTION __CDECL VIRTUAL GLOBAL MyMyIO1::IF_Start
        VAR_INPUT
                pio      : ^_IO;
                firsttime        : BOOL;
        END_VAR
  _ScrollArea2::IF_Start(pio, firsttime);  // ScrollArea2 is not the base class here, it is
END_FUNCTION
```

7.7.173 W 0173: Mixing '%s' with a REAL/LREAL constant is known to cause portability issues. It is recommended to use explicit conversion to the intended type

If a float constant with a UDINT variable is used in an expression with interchangeable (commutative) partial expressions (see example code below) and the result is assigned to a REAL/LREAL variable, the compiler may return incorrect results. In such a case the following warning will be sent:

```
Compiler: [WARN] W 0173 "Test.st"(164) Mixing UDINT with a REAL/LREAL constant is known to cau
```

For a correct result, it is necessary to cast the UDINT variable to LREAL (see last code line in the example below).

It is possible to **switch off** this **warning**. To do this, enter **#pragma warning(disable:173)**.

Example

```
// OK:
lr01 := 1.89 + ui01;
// Weird expression resolution:
lr01 := ui01 + 1.89;
// Workaround to correct result:
lr01 := to_lreal(ui01) + 1.89;
```

7.7.174 W 0174: Subexpression '%s' may cause overflow for huge values and produce a negative value.

If an unsigned expression result is assigned to a signed variable, a compiler evaluation takes place to ensure the data has a chance to overflow into the sign bit of the target. In such situation the binary value “wraps around” and produces a negative value, an undesired cause of bugs.

```
Compiler: [WARN] W 0174 "Test.st"(170) Subexpression 'DINT := (UINT * UINT) / UINT' may cause
```

To dismiss the warning one should use an explicit cast.

To completely shut the warning off, use the **#pragma warning(disable:174)** on top of your source file.

Example

```
// W0174 Subexpression 'DINT := (UINT * UINT) / UINT' may cause overflow for huge values a
di0 := ui0 * ui1 / ui2;
// The programmer uses a cast to tell the compiler that the conversion is intentional and
di0 := (ui0 * ui1 / ui2)$DINT;
```

7.7.175 E 0175: 16-byte alignment required

If you want to use optimizations like SSE with **new Intel processors (CP731)**, the following value must be entered in the **project properties** in the **subgroup Compiler** at the property "**GCCOptions**":

```
-O3 -march=core2 -mfpmath=sse,387
```

Now it can happen that **with SSE optimization switched on in C programs with complex mat**
The **compiler from version 72 onwards** can handle this problem. The following error message c

```
Compiler: [ERROR] E 0175 ... 16-byte (or more) data alignment was required for this C file.
```

So to fix this error a compiler version >= 72 must be used.

7.8 Online Status and Error Messages

The display of status and error messages occurs in the status test of the LASAL CLASS software. If the module is a CPU with a status display, the status or error number is also shown here. A POINTER or CHKSUM error message is also shown on the terminal screen.

Alternative to the display of "Er" and the code from 0 – 255, the following combination can also occur.

Pr Ln	currently linked
Er Ln	Linking error
Er Ld	Loading project
Er OP	Opsys programmed
Er AL	Cause of the error/status is unknown.
Er Lo	Loader error. Unfortunately there is no additional information for the cause of this error. E.g. the application has a problem (unknown Object, unknown connection, etc.)

Number	Report	Meaning	Reason/Remedy
00	**RUN RAM**	The user program is executing at the RAM at this moment. The display is not affected.	
01	**RUN ROM**	The user program, which is located inside the memory module, has been loaded into the RAM and is executing at the moment. The display is not affected.	
02	**RUNTIME**	Total duration of all cyclic objects exceeds maximum time; time can be configured by 2 system variables: • Runtime: Remaining rest time • SWRuntime: Pre-choice value for runtime counter	
03	**POINTER**	Incorrect pointers were located before the user program was executed.	Possible reasons: • Memory module is missing, not programmed or damaged. • Program in user program memory (RAM) is not executable. • Battery buffer broke down.

			• Software mistake, which overwrites the user program. Remedy: • New programming of the memory module, should it happen again, change module. • Change buffer-battery. • Get rid of program mistake.
04	**CHKSUM**	Wrong checksum was ascertained before execution of user program.	Reason/Remedy: see POINTER
05	**WATCHDOG**	Program interruption by watchdog-logic.	Possible reasons: • Interrupts of user program blocked for a longer time (command STI forgotten). • Defective programming of a hardware interrupt. • Wrong use of commands INB, OUTB, INW, OUTW • Processor damaged. Remedy: • Get rid of program mistake. • Change central unit.
06	**GENERAL ERROR**	General error	
07	**PROM DEFECT**	While programming the memory module a mistake appeared.	Possible reasons: • Memory module damaged. • User program too big. • Memory module missing. Remedy: • Change memory module.
08	**RESET**	CPU received command RESET and is waiting for further commands. The user program is not treated.	
09	**WD DEFECT**	The watchdog-logic is defective. The CPU checks the functions of the watchdog after the start. If there appears a mistake during the check the CPU is put into an intentionally,	Remedy: • Change CPU

		endless loop and does not accept commands anymore.	
10	**STOP**		
11	**PROG BUSYS**		
12	**PROGRAM LENGTH**		
13	**PROG END**	The programming of a memory module has been ended successfully.	
14	**PROG MEMO**	The CPU is programming the memory module at the moment.	
15	**STOP BRKPT**	The CPU was stopped by a breakpoint inside the program.	
16	**CPU STOP**	The CPU was stopped by the PG-software (F6 HALT at the status test).	
17	**INT ERROR**	The CPU executed a wrong interrupt and has stopped the user program, or discovered an unknown command during the execution of the program.	Possible reasons: • An operating system command was used, which does not exist. • Stack error (Odd number of PUSH and POP commands). • User program interruption through software error. Remedy: • Get rid of program error.
18	**SINGLE STEP**	CPU is in SINGLE STEP mode and waits for further commands.	
19	**READY**	A module respectively project is sent to the CPU, which is now ready to execute the program.	
20	**LOAD**	The program execution is stopped; the CPU is receiving a module or project at the moment.	
21	**UNZUL. MODUL**	The CPU received a module, which does not belong to the project.	
22	**MEMORY FULL**	The memory of the operating system (heap) is too small. No more memory could be reserved by a call to an internal function or an interface function from the application.	

23	**NOT LINKED**	At start of the CPU, a module was found to be missing in the project or a module does not belong to the project.	
24	**DIV BY 0**	An error encountered during a division.	Possible reasons: • Division to 0. • Result of division does not fit into register. Remedy: • Get rid of program error.
25	**DIAS ERROR**	An error encountered during access on a DIAS module.	Possible reasons: • Access on a non-existing DIAS module. • DIAS bus error. Remedy: • Check DIAS bus. • Check terminating resistors.
26	**WAIT**	CPU is busy. Remaining in this status over a long period of time indicates that the operating system is no longer operating correctly.	
27	**OP PROG**	Operating system is being programmed new.	
28	**OP INSTALLED**	Operating system is installed new.	
29	**OS TOO LONG**	Operating system cannot be transferred; memory too low.	
30	**NO OPERATING SYSTEM**	Boot loader message. No operating system found in RAM.	
31	**SEARCH FOR OS**	Boot loader searches for operating system in RAM.	
32	**NO DEVICE**		
33	**UNUSED CODE**		
34	**MEM ERROR**	The installed operating system is not adequate to the hardware configuration.	
35	**MAX IO**		

36	**MODULE LOAD ERROR**	LASAL module or project couldn't be loaded.	
37	**GENERAL OS ERROR**	General error while loading operating system.	
38	**APPLMEM ERROR**	Error at dynamic application memory administration (user heap).	
39	**OFFLINE**		
40	**APPL LOAD**		
41	**APPL SAVE**		
42	**ETHERCAT MANAGER ERROR**	In the EtherCAT manager, an error was detected.	
43	**ETHERCAT ERROR**		
44	**VARAN MANAGER ERROR**	An error number was entered in the VARAN Manager and the program has been stopped.	Possible causes: - the real network does not match the project Solution: - read log file
45	**VARAN ERROR**	A required VARAN client was removed or a communication error with a VARAN client has occurred. The program was stopped.	Possible causes: - damaged wiring - decentralized modules missing energy supply Solution: - read log file - analyze error tree
46	**APPL-LOAD-ERROR**	Error while loading application.	
47	**APPL-SAVE-ERROR**	Error while saving application.	
50	**ACCESS-EXCEPTION-ERROR**	Read-write access to prohibited memory area, e.g. writing on NULL pointer.	
51	**BOUND EXCEEDED**	Exception error at memory area infringement.	

52	**PRIVILEDGED INSTRUCTION**	Prohibited command for current CPU level, e.g. setting of segment register.	
53	**FLOATING POINT ERROR**	Error during a floating-point operation.	
60	**DIAS-RISC-ERROR**	Error from intelligent DIAS master.	
64	**INTERNAL ERROR**	Internal error, all Applications stopped	Restart, Message to SIGMATEK
65	**FILE ERROR**	Error during file operation.	
66	**DEBUG ASSERTION FAILED**	Internal error	Restart, Message to SIGMATEK
67	**REALTIME RUNTIME**	Total time of all Realtime objects exceeds maximum time; time can not be configured: • 2ms at 386er CPUs • 1ms all other CPUs	From version 1.1.7
68	**BACKGROUND RUNTIME**	Total time of all background objects exceeds maximum time, time can be configured with 2 system variables: • BTRuntime: remaining time • SWBTRuntime: Preselection value for runtime counter	
69	**PROG ERROR**		
70	**C-DIAS ERROR**	A C-DIAS module is required but is not available.	Cause: • The reason for this error is documented in the log file Solution: • Depends on the cause
71	**UNALIGNED ACCESS**	A memory access via an n-byte variable whose address cannot be divided by n is a non-aligned memory access. Such memory access is not supported on some platforms in certain memory areas and leads to an 'Unaligned Access' error. On an Edge2 CPU, only an aligned memory access is supported in the SRAM memory (zero voltage secure data)..	Possible causes: • On an Edge2 CPU an unaligned memory access is performed in the SRAM memory (zero voltage secure data). Solution: • Change the address of the variable or the pointer to the memory area so that it is aligned. If this is not possible, then carry out the access

			in smaller units (e.g. 1 byte accesses).
72	**S-DIAS ERROR**	A connection error with a S-DIAS module has occurred.	Possible causes: • real network does not match the project • S-DIAS client is defective Solution: • analyze logfile
75	**SRAM ERROR**	An error occurred while initializing, reading or writing SRAM data.	Possible causes: • SRAM configured incorrectly • SD card formatted incorrectly • SD card removed • battery for the supply of the internal program memory is empty Solution: • evaluate log file (Event00.log) • check configuration • change battery for the supply of the internal program memory • format SD card as EDGE medium with LASAL CLASS 2 • check SD card
85	**ERROR DRIVE C**	C:\-drive not found / mounted	Check SD-card or C: partition
86	**SALAMANDER EXIT**	Serious OS error	
87	**SALAMANDER RES2**	Currently not used	
88	**SALAMANDER RES3**	Currently not used	
89	**SALAMANDER RES4**	Currently not used	
90	**OS ERROR BKG**	Salamander OS error, the background task must be exited	
91	**OS ACCESS**	Access exception in an OS function	

92	**OS DOMAIN SW**	From the realtime task, a function of an unspecified duration was called (e.g.: File Save / Load etc.)	No such functions can be called from the RT task
93	**SALAMANDER RES5**	Currently not used	
94	**SALAMANDER RES6**	Currently not used	
95	**USER DEFINED 0**	Free useable code.	
96	**USER DEFINED 1**	Free useable code.	
97	**USER DEFINED 2**	Free useable code.	
98	**USER DEFINED 3**	Free useable code.	
99	**USER DEFINED 4**	Free useable code.	
100	**C_INIT**	Start of initialization, configuration is processed.	
101	**C_RUNRAM**	LASAL project has been started successfully from RAM.	
102	**C_RUNROM**	LASAL project has been started successfully from ROM.	
103	**C_RUNTIME**		
104	**C_READY**	Everything is okay.	
105	**C_OK**	Everything is okay.	
106	**C_UNKNOWN_CID**	Unknown class from a stand-alone or embedded object; or unknown basis class.	
107	**C_UNKNOWN_CONSTR**	Operating system class cannot be created, probably wrong operating system.	
108	**C_UNKNOWN_OBJECT**	Hint to an unknown object in an interpreter program. Creation of more than one DCC080 object.	

109	**C_UNKNOWN_CHNL**	Number of HW module bigger than 60.	
110	**C_WRONG_CONNECT**	No connection to required channels.	
111	**C_WRONG_ATTR**	Wrong server attributes.	
112	**C_SYNTAX_ERROR**	No specific error, recompile all part projects, transfer all.	
113	**C_NO_FILE_OPEN**	Tried to open an unknown table.	
114	**C_OUTOF_NEAR**	Memory assignment failed.	
115	**C_OUTOF_FAR**	Memory assignment failed.	
116	**C_INCOMAPTIBLE**	Object with the same name, but another class already exists.	
117	**C_COMPATIBLE**	Object with the same name and class already exists, but has to be updated.	
208 - 223	**LASALCLASS00 - LASALCLASS0F**	Error code reserved for LASAL CLASS.	
224	**LINKING**	Application will be linked.	
225	**LINKING ERROR**	Error at linking, message in LASAL status window.	
226	**LINKING DONE**	Linking finished	
230	**OP BURN**	Operating system will be stored in flash memory.	
231	**OP BURN FAIL**	Error while burning of operating system.	
232	**OP INSTALL**	Operating system will be installed.	
240	**USVWAIT**	Supply has been turned off, UPS is active.	
241	**REBOOT**	Restart of operating system.	
242	**LSLSAVE**		

243	**LSLLOAD**		
253	**PRE RUN**	Application will be started.	
254	**PRE RESET**	Application will be finished.	
255	**CONNECTION BREAK**		