

Rexroth IndraLogic XLC IndraMotion MLC 12VRS Generic Application Template

R911334191
Edition 01

Application Manual



Title	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Generic Application Template								
Type of Documentation	Application Manual								
Document Typecode	DOK-XLCMLC-TF*GAT**V12-AP01-EN-P								
Internal File Reference	RS-fb6e556e1586a8de0a6846a001dcf9bf-1-en-US-11								
Purpose of Documentation	This documentation provides a structured template to the IndraLogic PLC programmer. This template can be used to add and edit the PLC programming code. It includes the template, the wizard and example applications.								
Record of Revision	<table><tr><th>Edition</th><th>Release Date</th><th>Notes</th></tr><tr><td>Edition 01</td><td>05.2011</td><td>First edition for 12VRS</td></tr></table>			Edition	Release Date	Notes	Edition 01	05.2011	First edition for 12VRS
Edition	Release Date	Notes							
Edition 01	05.2011	First edition for 12VRS							
Copyright	© Bosch Rexroth AG 2011 Copying this document, giving it to others and the use or communication of the contents thereof without express authority, are forbidden. Offenders are liable for the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design (DIN 34-1).								
Validity	The specified data is for product description purposes only and may not be deemed to be guaranteed unless expressly confirmed in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.								
Published by	Bosch Rexroth AG Bgm.-Dr.-Nebel-Str. 2 ■ 97816 Lohr a. Main, Germany Phone +49 (0)93 52/ 40-0 ■ Fax +49 (0)93 52/ 40-48 85 http://www.boschrexroth.com/ System Development Automation Motion Logic Control, AK (KaWa/PiGe)								
Note	This document has been printed on chlorine-free bleached paper.								

Table of Contents

	Page
1 About this Documentation.....	9
1.1 Validity of the Documentation.....	9
1.2 Required and Supplementing Documentations.....	9
1.3 Information Representation.....	13
1.3.1 Safety Instructions.....	13
1.3.2 Icons.....	14
1.3.3 Descriptions.....	14
1.3.4 Abbreviations.....	14
2 Important Instructions on Use.....	17
2.1 Appropriate Use.....	17
2.1.1 Introduction.....	17
2.1.2 Areas of Use and Application.....	17
2.2 Inappropriate Use.....	18
3 Safety Instructions for Electric Drives and Controls.....	19
3.1 Definitions of Terms.....	19
3.2 General Information.....	20
3.2.1 Using the Safety Instructions and Passing Them on to Others.....	20
3.2.2 Requirements for Safe Use.....	20
3.2.3 Hazards by Improper Use.....	21
3.3 Instructions with Regard to Specific Dangers.....	23
3.3.1 Protection Against Contact With Electrical Parts and Housings.....	23
3.3.2 Protective Extra-Low Voltage as Protection Against Electric Shock	24
3.3.3 Protection Against Dangerous Movements.....	24
3.3.4 Protection Against Magnetic and Electromagnetic Fields During Operation and Mounting.....	26
3.3.5 Protection Against Contact With Hot Parts.....	26
3.3.6 Protection During Handling and Mounting.....	27
3.3.7 Battery Safety.....	27
3.3.8 Protection Against Pressurized Systems.....	28
3.4 Explanation of Signal Words and the Safety Alert Symbol.....	28
4 Introduction.....	31
5 Automating Applications.....	33
5.1 General Information.....	33
5.2 Initialization.....	33
5.3 Operation Modes.....	33
5.3.1 General Information.....	33
5.3.2 Automatic.....	34
5.3.3 Manual.....	34
5.4 Cyclical Processing.....	34
5.4.1 General Information.....	34

Table of Contents

	Page
5.4.2	Operation Mode Independent Logic..... 34
5.4.3	Generating and Displaying Diagnostic Messages..... 34
5.5	Task Management..... 34
5.5.1	General Information..... 34
5.5.2	Principles of Task Management..... 34
5.5.3	Recommended Task Division when Using IndraWorks/IndraLogic..... 35
	General Information..... 35
	Typical Tasks for the MotionTask..... 35
	Typical Tasks for the PlcTask..... 36
	Typical Tasks for the BackgroundTask..... 36
5.5.4	Measuring Cycle Times..... 36
	Monitoring Dialog for Task Configuration..... 36
	Cycle Time Dialog of the Control..... 37
	Task Execution Viewer of the Control..... 38
	Time Measurement in the PLC Project..... 38
5.5.5	Application-specific Watchdog..... 39
5.6	Error Management..... 40
5.6.1	General Information..... 40
5.6.2	Principles of Error Management..... 40
5.6.3	Error Detection..... 40
5.6.4	Error Reaction..... 41
5.6.5	Error Diagnostics..... 42
5.6.6	Error Acknowledgement / Restart after an Error..... 43
5.7	Visualization..... 43
5.7.1	General Information..... 43
5.7.2	Typical Contents of a Visualization..... 43
	Machine Status Display..... 43
	Controlling the Machine States..... 43
5.7.3	Providing Data Using the PLC..... 44
	General Information..... 44
	Symbol Configuration..... 44
	Communication Configuration..... 46
6	ML_TechInterface.library..... 47
6.1	Introduction and Overview..... 47
6.2	Initial TechInterface Configuration..... 53
6.3	AxisInterface Function Blocks..... 57
6.3.1	MB_AxisInitType01..... 57
6.3.2	MB_AxisInterfaceType01..... 59
6.4	Operation Modes..... 61
6.4.1	Overview..... 61
6.4.2	Drive Ready..... 63
6.4.3	Drive Halt..... 64
6.4.4	Drive Stop..... 64
6.4.5	Drive Homing..... 65
6.4.6	Absolute Positioning..... 65

Table of Contents

	Page
6.4.7	Relative Positioning..... 66
6.4.8	Additive Positioning..... 67
6.4.9	Velocity Control..... 67
6.4.10	Operation Mode "Phase Synchronization"..... 68
6.4.11	Operation Mode "Velocity Synchronization"..... 69
6.4.12	Operation Mode "Electronic Cam"..... 69
6.4.13	Operation Mode "Electronic MotionProfile"..... 70
6.4.14	Operation Mode "Electronic FlexProfile"..... 71
6.4.15	Operation Mode "Drive-controlled Interruption"..... 72
6.4.16	Operation Mode "External Function Block"..... 73
6.5	Global Variables..... 74
6.6	First Steps, AxisInterface..... 75
6.6.1	Overview..... 75
6.6.2	Program for the AxisInterface..... 75
6.6.3	AxisInterface Visualizations..... 76
	Overview..... 76
	System Overview Visualization..... 76
	Axis Overview Visualization..... 78
	Positioning Mode Visualization..... 80
	Velocity Mode Visualization..... 81
	Synchronization Mode Visualization..... 82
	Visualization, Synchronization Mode with Profiles..... 84
	IMC Interface Control and Status Visualization..... 86
	PLC Load Visualization..... 88
6.7	AxisInterface User Extensions..... 88
6.7.1	Overview..... 88
6.7.2	Extending the arAxisCtrl_gb[] Structure..... 90
6.7.3	Extending the arAxisStatus_gb[] Structure..... 91
6.7.4	Extending the Function Block..... 91
6.8	AxisInterface Structures..... 93
6.8.1	Overview..... 93
6.8.2	MB_AXIS_STATUS_TYPE01 Structure..... 94
	MB_AXIS_STATUS_TYPE01..... 94
	MB_AXIS_STATUS_TYPE01 Elements..... 95
6.8.3	MB_AXIS_CONTROL_TYPE01 Structure..... 100
	MB_AXIS_CONTROL_TYPE01..... 100
	MB_AXIS_CONTROL_TYPE01 Elements..... 101
6.9	IMC Interface Structures..... 115
6.9.1	Overview..... 115
6.9.2	MB_IMC_Status_Type01..... 116
6.9.3	MB_IMC_Control_Type01..... 117
6.10	Special Features of the IndraMotion MLD..... 117
6.10.1	Introduction..... 117
6.10.2	Parameterization..... 118
6.10.3	PLC Programming..... 119

Table of Contents

	Page
7 GAT^{compact}	121
7.1 Overview of GAT ^{compact}	121
7.2 First Steps - GAT ^{compact}	122
7.2.1 Preparation	122
7.2.2 Creating a Simple Motion Application	122
General Information	122
Topic	122
Operating sequence	123
7.3 Initialization	130
7.4 Operation Modes	133
7.5 Cyclical Processing	135
7.6 Task Handling	135
7.6.1 Task Configuration, GAT ^{compact}	135
7.6.2 Adding a MotionTask	135
7.6.3 Adding a BackgroundTask	137
7.7 Error Handling	138
7.7.1 Error Detection and Diagnostics	138
7.7.2 Error Reaction	140
7.7.3 Clearing Errors	142
7.8 Visualization	142
7.8.1 Internal Visualization for Assistance during Start-up	142
General Information	142
System_Overview	142
7.8.2 External Visualization with Separate HMI (WinStudio Visualization)	144
General Information	144
Dialog Structure	146
Extending the Visualization	147
Communication Connection	149
Multiple Language Capability	151
7.9 Interfaces to GAT ^{compact}	152
7.9.1 General Information	152
7.9.2 Data for the State Machine	152
7.9.3 Data for Error Handling	152
7.9.4 Axis Data	153
7.9.5 Control Data	153
7.10 How To: Typical User Activities	154
7.10.1 Access to Axis Data	154
7.10.2 Adding an Axis	155
7.10.3 Removing/renaming an Axis	156
7.10.4 Adding an Operation Mode	158
7.10.5 Processing in MotionTask	159
7.10.6 AxisInterface Extensions	159
7.11 GAT ^{compact} Brief Overview	161
7.11.1 Structure Overview	161
7.11.2 Overview of the Machine's Most Important Global Variables	161
7.11.3 Overview of the Most Important Global Variables of the Axes	162

Table of Contents

	Page
7.11.4 Overview of the Machine's Methods and Actions.....	162
7.11.5 Access to Axes within the Machine.....	162
8 GAT with Central State Machine.....	165
8.1 General Information.....	165
8.2 Overview.....	165
8.2.1 Basic Idea.....	165
8.2.2 Requirements for Use.....	166
8.3 First Steps - GAT with Central State Machine.....	167
8.3.1 Preparation.....	167
8.3.2 Creating a Simple Motion Application.....	167
General Information.....	167
Topic.....	167
Operating Sequence.....	168
8.4 Functional Principle.....	180
8.5 Initialization.....	180
8.6 Operation Modes.....	181
8.6.1 General Information.....	181
8.6.2 States of an Operation Mode.....	182
8.6.3 Setting the Target Operation Mode after Initialization.....	183
8.6.4 Changing the Operation Mode.....	183
8.6.5 User Activities when Exiting an Operation Mode.....	184
8.6.6 User Activities when Starting an Operation Mode.....	185
8.6.7 Bypassing the Central Operation Mode.....	185
8.7 Modules.....	186
8.7.1 General Information.....	186
8.7.2 Creation Rules.....	186
8.7.3 Module Interface.....	187
General Information.....	187
Control and Status Structure.....	187
Module axes.....	187
Module-specific I/O data.....	190
Module Instances.....	191
8.7.4 States in the Module.....	191
8.7.5 Module Coordination.....	192
8.8 Task Handling.....	192
8.8.1 Task Configuration, GAT.....	192
8.8.2 GATMotionSyncTask.....	192
8.8.3 GATBackgroundProgTask.....	193
8.9 Error Handling.....	194
8.9.1 Error Detection and Diagnostics.....	194
8.9.2 Error Reaction.....	195
8.9.3 Clearing Errors.....	197
8.10 Visualization.....	198
8.10.1 Internal Visualization for Commissioning Support.....	198
General Information.....	198

Table of Contents

	Page
AXIS_GAT.....	198
8.10.2 External Visualization with Separate HMI (WinStudio Visualization).....	199
General Information.....	199
Dialog Structure.....	201
Extending the Visualization.....	202
Communication Connection.....	204
Multiple Language Capability.....	206
8.11 Interfaces to GAT.....	207
8.11.1 General Information.....	207
8.11.2 Machine Data.....	207
8.11.3 Data for Error Handling.....	208
8.11.4 Module Data.....	208
8.11.5 Axis Data.....	209
8.11.6 Control Data.....	210
8.12 How To: Typical User Activities.....	210
8.12.1 Access to Axis Data	210
8.12.2 Adding an Axis.....	211
8.12.3 Removing/renaming an Axis	212
8.12.4 Adding an Operation Mode.....	214
8.12.5 Selecting the Desired Operation Mode after Clear Error.....	215
8.12.6 Superordinate Logic for Operation Mode Specification.....	216
8.12.7 Adding States and Transitions.....	216
8.12.8 Adding an New Module.....	217
8.12.9 Adding Another Instance of a Module.....	218
8.12.10 Deactivating a Module.....	219
8.12.11 Reusing Modules.....	220
8.12.12 Using Global and Address Variables in Modules.....	222
8.12.13 Processing in MotionTask.....	224
8.12.14 AxisInterface Extensions.....	224
8.12.15 Generating Your Own Program Framework Based on the GAT.....	225
8.12.16 PackML V3.0 Program Framework Based on GAT.....	226
8.12.17 Time Sensitive, Application-specific Error Reaction.....	228
8.12.18 Switching off GAT Error Reaction for F0 Errors.....	228
8.12.19 Updating from Version 10/11 to 12.....	228
8.13 GAT, Brief Overview.....	229
8.13.1 Structure Overview.....	229
8.13.2 Overview of the Machine's Most Important Global Variables.....	229
8.13.3 Overview of the Module's Most Important Global Variables.....	230
8.13.4 Overview of the Most Important Global Variables of the Axes.....	230
8.13.5 Overview of the Machine's Methods and Actions.....	230
8.13.6 Overview of the Machine's Methods and Actions.....	231
8.13.7 Access to Axes within the Module.....	232
8.13.8 Access to I/O Data within the Module.....	232
9 GAT with Decentral State Machine.....	233
9.1 General Information.....	233

Table of Contents

	Page
9.2 Overview.....	233
9.2.1 Basic Idea.....	233
9.2.2 Requirements for Use.....	235
9.3 First Steps - GAT with Decentral State Machine.....	235
9.3.1 Preparation.....	235
9.3.2 Creating a Simple Motion Application.....	235
General Information.....	235
Topic.....	235
Operating Sequence.....	236
9.4 Functional Principle.....	243
9.5 Initialization.....	244
9.6 Operation Modes.....	245
9.7 Units.....	245
9.7.1 General Information.....	245
9.7.2 Creation Rules.....	246
9.7.3 Unit Interface.....	247
General Information.....	247
Control and Status Structure.....	247
Unit Axes.....	247
Unit-specific I/O Data.....	250
Unit Instances.....	251
9.7.4 Operation Modes in the Unit.....	251
9.7.5 Sequential Programming.....	251
9.7.6 Unit Coordination.....	252
9.8 Task Management.....	253
9.8.1 Task Configuration, GAT.....	253
9.8.2 Adding a BackgroundTask.....	253
9.9 Error Handling.....	254
9.9.1 Error Detection and Diagnostics.....	254
9.9.2 Error Reaction.....	256
9.9.3 Clearing Errors.....	257
9.10 Product and Machine Data Handling.....	257
9.10.1 Introduction.....	257
9.10.2 Machine Data Handling.....	258
9.10.3 Product Data Handling.....	258
9.11 Visualization.....	258
9.11.1 Switching the Visualization.....	258
9.11.2 Internal Visualization for Commissioning Support.....	259
General Information.....	259
System_Overview.....	259
9.11.3 External Visualization with Separate HMI (WinStudio Visualization).....	261
General Information.....	261
Dialog Structure.....	263
Extending the Visualization.....	264
Extending the Maximum Number of Axes.....	266
Communication Connection.....	268

Table of Contents

	Page
Multiple Language Capability.....	270
9.12 Interfaces to GAT with Decentral State Machine.....	271
9.12.1 General Information.....	271
9.12.2 Machine Machine Handler Data (TE_MachineControl).....	271
9.12.3 Central Error Handling Data (TE_ErrorHandling).....	271
9.12.4 Unit Data.....	272
9.12.5 Axis Data.....	273
9.12.6 Control Data.....	274
9.13 How To: Typical User Activities.....	274
9.13.1 Access to Axis Data	274
9.13.2 Adding an Axis.....	275
9.13.3 Removing/renaming an Axis	276
9.13.4 Adding a Primary Operation Mode.....	276
9.13.5 Adding a New Unit.....	277
9.13.6 Adding Another Instance of a Unit.....	277
9.13.7 Deactivating Units.....	278
9.13.8 Reusing Units.....	279
9.13.9 Using Global and Address Variables in Units.....	280
9.13.10 AxisInterface Extensions.....	280
9.13.11 Generating Your Own Program Framework Based on the GAT.....	281
9.14 GAT - Decentral State Machine, Brief Overview.....	283
9.14.1 Structure Overview.....	283
9.14.2 Overview of the Machine's Most Important Global Variables and Error Handling.....	283
9.14.3 Overview of the Most Important Global Variables of the Units.....	284
9.14.4 Overview of the Most Important Global Variables of the Axes.....	284
9.14.5 Overview of the Methods and Actions of TE_MachineControl.....	284
9.14.6 Overview of the Methods and Actions of TE_ErrorHandling.....	285
9.14.7 Overview of the Methods and Actions of the Units.....	285
9.14.8 Access to Axes within the Unit.....	286
9.14.9 Access to I/O Data within the Unit.....	286
10 Service and Support.....	287
Index.....	289

1 About this Documentation

1.1 Validity of the Documentation

- Target group** This documentation is intended for users who can use these templates
- GAT compact
 - GAT (central state machine)
 - GAT (decentral state machine)
 - User-defined GAT
- to create clearly structured Logic applications quickly and easily.
This documentation supports the user during all phases
- Engineering
 - Commissioning
 - Operation

1.2 Required and Supplementing Documentations

Documentation titles with type designation codes and parts numbers

IndraWorks		MLC	XLC
/36/	Rexroth IndraWorks 12VRS Software Installation DOK-IWORKS-SOFTINS*V12-COxx-EN-P, R911334396 This documentation describes the IndraWorks installation.	X	X
/5/	Rexroth IndraWorks 12VRS Engineering DOK-IWORKS-ENGINEE*V12-APxx-EN-P, R911334388 This documentation describes the application of IndraWorks in which the Rexroth Engineering tools are integrated. It includes instructions on how to work with IndraWorks and how to operate the oscilloscope function.	X	X
/20/	Rexroth IndraMotion MLC 12VRS Functional Description DOK-MLC***-FUNC****V12-APxx-EN-P, R911333848 This documentation describes wizards, context menus, dialogs, control commissioning, device configuration and functionalities of the IndraMotion MLC.	X	
/20/	Rexroth IndraLogic XLC 12VRS Functional Description DOK-XLC***-FUNC****V12-APxx-EN-P, R911333878 This documentation describes wizards, context menus, dialogs, control commissioning, device configuration and functionalities of the IndraLogic XLC.		X
/7/	Rexroth IndraWorks 12VRS CamBuilder DOK-IWORKS-CAMBUIL*V12-APxx-EN-P, R911333842 This documentation describes the basic principles and operation of the CamBuilder, the cam editing tool.	X	X
/37/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Automation Interface DOK-XLCMLC-AUT*INT*V12-APxx-EN-P, R911334178 This documentation describes the script-based access to IndraWorks project data via the interface of the Automation Interface.	X	X

About this Documentation

/38/	Rexroth IndraWorks 12VRS FDT Container DOK-IWORKS-FDT*CON*V12-APxx-EN-P, R911334398 This documentation describes the IndraWorks FDT Container functionality. It includes the activation of the functionality in the project and working with DTMs.	X	X
/29/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Project Conversion DOK-XLCMLC-PROCONV*V12-APxx-EN-P, R911334187 This documentation described the project conversion of IndraLogic 04VRS and IndraMotion MLC04VRS on IndraWorks Version 12 with IndraLogic 2G. It especially focuses on changes in the field of Motion and PLC.	X	X
/28/	Rexroth IndraMotion MLC 12VRS Commissioning DOK-MLC***-STARTUP*V12-COxx-EN-P, R911333858 This documentation describes the steps required for commissioning and performing service on the IndraMotion MLC system. It includes checklists for tasks to be frequently performed and a detailed description of the steps.	X	

Motion		MLC	XLC
/23/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS PLCopen Libraries DOK-XLCMLC-FUNLIB**V12-LIxx-EN-P, R911334182 This documentation describes the function blocks, functions and data types of the RIL_CommonTypes, ML_Base and ML_PLCOpen libraries for the IndraLogic XLC/IndraMotion MLC. It also includes the error reactions of function blocks.	X	X
/27/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Generic Application Template DOK-XLCMLC-TF*GAT**V12-APxx-EN-P, R911334191 This documentation provides a structured template to the IndraLogic PLC programmer. This template can be used to add and edit the PLC programming code. It includes the template, the template wizard and example applications.	X	X
/31/	Rexroth IndraMotion MLC 12VRS RCL Programming Instruction DOK-MLC***-RCL*PRO*V12-APxx-EN-P, R911333852 This documentation provides information on the RobotControl. It is given most importance to the programming language RCL (RobotControl Language). The program structure, variables, functions, motion statements and the required system parameters are described.	X	
/21/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Parameters DOK-XLCMLC-PARAM***V12-RExx-EN-P, R911334176 This documentation describes the parameters of the XLC/MLC systems as well as the interaction between parameterization and programming. It includes the axis parameters, control parameters, kinematic parameters, touch probe parameters and programmable limit switch parameters.	X	X
/10/	Rexroth IndraDrive; Firmware for Drive Controllers MPH, MPB, MPD, MPC-07 DOK-INDRV*-MP*-07VRS**-FKxx-EN-P, R911328670		
/11/	Rexroth IndraDrive MPx-16 Functions DOK-INDRV*-MP*-16VRS**-APxx-EN-P, R911326767		

About this Documentation

Field buses		MLC	XLC
/39/	Rexroth IndraMotion MLC 11VRS PLCopen Field Bus DOK-IM*ML*-PLCFBUS*V11-APxx-EN-P, R911333896 This documentation describes the creation of field bus drives in an IndraWorks project, function blocks, functions and data types of the libraries RIL_CommonTypes.library (excerpt for field bus drives), RMB_PLCPopenFieldBus.library, RIL_Utility.library (excerpt for field bus drives). It also includes the error reactions of function blocks.	X	X
/4/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Field Buses DOK-XLCMLC-FB*****V12-APxx-EN-P, R911334394 This documentation describes the supported field buses and their diagnostic function blocks.	X	X

HMI		MLC	XLC
/8/	Rexroth IndraWorks 12VRS HMI DOK-IWORKS-HMI*****V12-APxx-EN-P, R911334392 This documentation describes the functions, configuration and operation of the user interfaces IndraWorks HMI Engineering and IndraWorks HMI Operation.	X	X
/6/	Rexroth IndraWorks 12VRS WinStudio DOK-IWORKS-WINSTUD*V12-APxx-EN-P, R911333844 This documentation describes the installation of the software, working with WinStudio and the creation and operation of applications.	X	X
/50/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS HMI Connection DOK-XLCMLC-HMI*****V12-APxx-EN-P, R911334184 This documentation describes the visualization systems supported by the IndraLogic XLC and IndraMotion MLC and their connection.	X	X

PLC		MLC	XLC
/3/	Rexroth IndraWorks 12VRS IndraLogic 2G Programming Instruction DOK-IWORKS-IL2GPRO*V12-APxx-EN-P, R911334390 This documentation describes the PLC programming tool IndraLogic 2G and its usage. It includes the basic usage, first steps, visualization, menu items and editors.	X	X
/33/	Rexroth IndraWorks 12VRS, Basic Libraries, IndraLogic 2G DOK-IL*2G*-BASLIB**V12-LIxx-EN-P, R911333835 This documentation describes the system-comprehensive PLC libraries.	X	X

Technology			
/30/	Rexroth IndraMotion MLC 12VRS Technology Libraries DOK-MLC***-TF*LIB**V12-LIxx-EN-P, R911333868 This documentation describes the function blocks, functions and data types of the libraries "ML_TechInterface.library", "ML_TechMotion.library", "RMB_TechCam.library" and "ML_TechBase.library". It also includes libraries for the winder functionality, register controller functionality and CrossCutter functionality.	X	
/60/	Rexroth IndraMotion MLC 12VRS RegisterControl (Library) DOK-MLC***-REGI*CO*V12-LIxx-EN-P, R911333856 This documentation describes the inputs and outputs of the individual function blocks and provides notes on their usage.	X	

About this Documentation

/62/	Rexroth IndraMotion MLC 12VRS RegisterControl (Application Manual) DOK-MLC***-REGI*CO*V12-APxx-EN-P, R911333854 This documentation describes the application of the integrated register control for a rotogravure printing machine. The components of the mark stream sensor, the HMI application and the error recovery options are described. This instruction provides information on how to operate the register control, react on errors and query diagnostics. This documentation is written for machine setters and machine operators.	X	
/49/	Rexroth IndraMotion MLC 12VRS Application of Winder Functions DOK-MLC***-TF*WIND*V12-APxx-EN-P, R911333870 This application-related system documentation describes the usage of the winder technology functions.	X	

Hardware		MLC	XLC
/1/	Rexroth IndraControl L45/L65 DOK-CONTRL-IC*L45*L65*-PRxx-EN-P, R911324661	X	X
/2/	Rexroth IndraControl L25 DOK-CONTRL-IC*L25*****-PRxx-EN-P, R911328474	X	X
/24/	Rexroth IndraControl Lxx 12VRS Function Modules DOK-CONTRL-FM*LXX**V12-APxx-EN-P, R911333830 This documentation describes all function modules of the Lxx controls including engineering and diagnostics.	X	X
/12/	Rexroth IndraDrive Drive Controllers MPx-02 to MPx-07 DOK-INDRV*-GEN-**VRS**-PAxx-EN-P, R911297317		
/13/	Rexroth IndraDrive MPx-02 to MPx-07 and HMV DOK-INDRV*-GEN-**VRS**-WAxx-EN-P, R911297319		
/35/	Rexroth IndraDrive Drive Controller Control Sections CSB01, CSH01, CDB01 DOK-INDRV*-CSH*****-PR08-EN-P, R911295012		

Diagnostics and Service		MLC	XLC
/26/	Rexroth IndraMotion MLC/XLC 11VRS Service Tool DOK-IM*ML*-IMST****V11-RExx-EN-P, R911331940	X	X
/22/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Diagnostics DOK-XLCMLC-DIAG****V12-RExx-EN-P, R911334180 This documentation includes all control parameters implemented in the control systems IndraLogic XLC and IndraMotion MLC.	X	X

System Overview		MLC	XLC
/48/	Rexroth IndraMotion for Printing 12VRS System Overview DOK-IM*PR*-SYSTEM**V11-PRxx-EN-P, R911333840 This documentation describes the product IndraMotion for Packaging. It introduces the control systems, drive systems and I/O systems as well as the commissioning and programming.	X	
/48/	Rexroth IndraMotion for Packaging 12VRS System Overview DOK-IM*PA*-SYSTEM**V12-PRxx-EN-P, R911333838 This documentation describes the product IndraMotion for Packaging. It introduces the control systems, drive systems and I/O systems as well as the commissioning and programming.	X	

About this Documentation

/9/	Rexroth IndraMotion MLC 12VRS System Overview DOK-MLC***-SYSTEM**V12-PRxx-EN-P, R911333860 This documentation provides an overview on the possible hardware/software components of the automation system IndraMotion MLC of the named version. It helps assembling a system.	X	
/9/	Rexroth IndraLogic XLC 12VRS System Overview DOK-XLC***-SYSTEM**V12-PRxx-EN-P, R911333880 This documentation provides an overview on the possible hardware/software components of the automation system IndraLogic XLC of the named version. It helps assembling a system.		X

First Steps		MLC	XLC
/25/	Rexroth IndraMotion MLC 12VRS First Steps DOK-MLC***-F*STEP**V12-COxx-EN-P, R911333846 This documentation describes the first steps of the IndraMotion MLC and the RobotControl. It includes the hardware and software prerequisites as well as the creation of a project.	X	
/25/	Rexroth IndraLogic XLC 12VRS First Steps DOK-XLC***-F*STEP**V12-COxx-EN-P, R911333876 This documentation describes the first steps of the IndraLogic XLC. It includes the hardware and software prerequisites as well as the creation of a project.		X

xx Respective edition
Fig. 1-1: XCL/MLC documentation overview

1.3 Information Representation

1.3.1 Safety Instructions

Structure for safety instructions for electric drives and controls:

- [3.1 Definition of Terms, page 19](#)
- [3.2 General Information, page 20](#)
 - [3.2.1 Using the Safety Instructions and Passing Them on to Others, page 20](#)
 - [3.2.2 Requirements for Safe Use, page 20](#)
 - [3.2.3 Hazards by Improper Use, page 21](#)
- [3.3 Hazard-related Instructions, page 23](#)
 - [3.3.1 Protection Against Contact with Electrical Parts and Housings, page 23](#)
 - [3.3.2 Protective Extra-Low Voltage as Protection Against Electric Shock, page 24](#)
 - [3.3.3 Protection Against Dangerous Movements, page 24](#)
 - [3.3.4 Protection Against Magnetic and Electromagnetic Fields During Operation and Mounting, page 26](#)
 - [3.3.5 Protections Against Contact with Hot Parts, page 26](#)
 - [3.3.6 Protection During Handling and Mounting, page 27](#)
 - [3.3.7 Battery Safety, page 27](#)
 - [3.3.8 Protection Against Pressurized Lines, page 28](#)
- [3.4 Explanation of Signal Words and the Safety Alert Symbol, page 28.](#)

About this Documentation

1.3.2 Icons

Representation

The representation of different symbols can be found under chapter [3.4 Explanation of Signal Words and the Safety Alert Symbol](#), page 28.

1.3.3 Descriptions

Description	Explanation
GAT	Generic Application Template
AxisInterface	The AxisInterface pools and extends PLCopen motion function blocks and provides an easy-to-operate interface for the drive functionality.
IMC Interface	The IMC Interface provides an easy-to-operate interface for the control functionality. It includes control signals and parameters for the diagnostics and administration of the control.
TechInterface	The TechInterface consists of these components: • AxisInterface and • IMC Interface
Function block	Provides application-specific functionalities
IndraMotion MLC	Compact Motion Logic systems with Motion, Robot and Logic Control functionalities
IndraLogic XLC	Compact PLC system with integrated Motion functionality
IndraWorks Engineering Framework	Project planning and commissioning tool of Bosch Rexroth
IndraDrive	Drive control unit
Wizard	Wizard guiding the user through several dialogs for an ergonomic data input

Fig. 1-2: Used descriptions

1.3.4 Abbreviations

Abbreviation	Meaning
IEC	International Electrotechnical Commission
SERCOS	SERCOS (SErial Realtime COmmunication System) interface is a worldwide standardized digital interface to communicate between controls and drives
XML	Extensible Markup Language (XML). It is a markup language to represent hierarchical structured data as text data
CSV	This file format describes the structure of a text file to save or exchange simply structured data. The file extension .csv is an abbreviation for "comma separated values".
IMC	IndraMotion Control
MLD-S	Motion Logic Drive (single drive)
MLD-M	Motion Logic Drive (multi drive)
i.e.	that is
e.g.	for example

About this Documentation

ms	millisecond
mm	millimeter
%	percent
rpm	speed

Fig. 1-3: Used abbreviations

2 Important Instructions on Use

2.1 Appropriate Use

2.1.1 Introduction

Rexroth products represent state-of-the-art developments and manufacturing. They are tested prior to delivery to ensure operating safety and reliability.

The products may only be used in the appropriate way. If they are not used in the appropriate way, situations resulting in property damage and personal injury can occur.



Bosch Rexroth as manufacturer is not liable for any damages resulting from inappropriate use. In such cases, the guarantee and the right to payment of damages resulting from inappropriate use are forfeited. The user alone carries all responsibility of the risks.

Before using Rexroth products, make sure that all the pre-requisites for an appropriate use of the products are satisfied:

- Personnel that in any way, shape or form uses our products must first read and understand the relevant safety instructions and be familiar with appropriate use.
- If the products take the form of hardware, then they must remain in their original state, in other words, no structural changes are permitted. It is not permitted to decompile software products or alter source codes.
- Do not mount damaged or faulty products or use them in operation.
- Ensure that the products have been installed in the manner described in the relevant documentation.

2.1.2 Areas of Use and Application

Rexroth IndraMotion MLC/
IndraLogic XLC

IndraMotion MLC/IndraLogic XLC and its function modules are intended for Motion/Logic applications.



The IndraMotion MLC/IndraLogic XLC and its function modules may only be used with the accessories and mounting parts listed in this documentation. Components that are not expressly mentioned must neither be attached nor connected. The same is valid for cables and lines.

The operation must only be carried out with the hardware component configurations and combinations that were expressly mentioned and with the software and firmware indicated and specified in the respective documentations and functional descriptions.

IndraMotion MLC/IndraLogic XLC and its function modules have been developed for use in single and multi-axis drive and control tasks.

For the application-specific use of the machine operating and visualization terminals, device models featuring different equipments and different interfaces are available.

Typical scopes of use where IndraMotion MLC/IndraLogic XLC and its function modules can be applied are:

- [Handling and mounting systems]
- [Packaging and food machines]

Important Instructions on Use

- [Printing and paper-processing machines]

Rexroth IndraMotion MLC/IndraLogic XLC and the function modules must only be operated under the mounting and installation conditions, the position, and the ambient conditions (temperature, type of protection, moisture, EMC, etc.) specified in the related documentations.

2.2 Inappropriate Use

The use of Rexroth IndraMotion MLC/IndraLogic XLC and its function modules in applications other than those specified or described in the documentation and the technical data is considered as "improper".

The Rexroth IndraMotion MLC/IndraLogic XLC and its function modules may not be used if it is

- exposed to operating conditions which do not correspond to the specified ambient conditions. Operation under water, extreme temperature fluctuations or extreme maximum temperatures etc. is prohibited.
- Furthermore, the IndraMotion MLC/IndraLogic XLC and its function modules must not be used in any applications not expressly approved by Bosch Rexroth. Therefore, read the statements given in the general safety instructions!

3 Safety Instructions for Electric Drives and Controls

3.1 Definitions of Terms

Application Documentation	Application documentation comprises the entire documentation used to inform the user of the product about the use and safety-relevant features for configuring, integrating, installing, mounting, commissioning, operating, maintaining, repairing and decommissioning the product. The following terms are also used for this kind of documentation: User Guide, Operation Manual, Commissioning Manual, Instruction Manual, Project Planning Manual, Application Manual, etc.
Component	A component is a combination of elements with a specified function, which are part of a piece of equipment, device or system. Components of the electric drive and control system are, for example, supply units, drive controllers, mains choke, mains filter, motors, cables, etc.
Control System	A control system comprises several interconnected control components placed on the market as a single functional unit.
Device	A device is a finished product with a defined function, intended for users and placed on the market as an individual piece of merchandise.
Electrical Equipment	Electrical equipment encompasses all devices used to generate, convert, transmit, distribute or apply electrical energy, such as electric motors, transformers, switching devices, cables, lines, power-consuming devices, circuit board assemblies, plug-in units, control cabinets, etc.
Electric Drive System	An electric drive system comprises all components from mains supply to motor shaft; this includes, for example, electric motor(s), motor encoder(s), supply units and drive controllers, as well as auxiliary and additional components, such as mains filter, mains choke and the corresponding lines and cables.
Installation	An installation consists of several devices or systems interconnected for a defined purpose and on a defined site which, however, are not intended to be placed on the market as a single functional unit.
Machine	A machine is the entirety of interconnected parts or units at least one of which is movable. Thus, a machine consists of the appropriate machine drive elements, as well as control and power circuits, which have been assembled for a specific application. A machine is, for example, intended for processing, treatment, movement or packaging of a material. The term "machine" also covers a combination of machines which are arranged and controlled in such a way that they function as a unified whole.
Manufacturer	The manufacturer is an individual or legal entity bearing responsibility for the design and manufacture of a product which is placed on the market in the individual's or legal entity's name. The manufacturer can use finished products, finished parts or finished elements, or contract out work to subcontractors. However, the manufacturer must always have overall control and possess the required authority to take responsibility for the product.
Product	Examples of a product: Device, component, part, system, software, firmware, among other things.
Project Planning Manual	A project planning manual is part of the application documentation used to support the sizing and planning of systems, machines or installations.
Qualified Persons	In terms of this application documentation, qualified persons are those persons who are familiar with the installation, mounting, commissioning and operation of the components of the electric drive and control system, as well as with the hazards this implies, and who possess the qualifications their work requires. To comply with these qualifications, it is necessary, among other things,

Safety Instructions for Electric Drives and Controls

- 1) to be trained, instructed or authorized to switch electric circuits and devices safely on and off, to ground them and to mark them
- 2) to be trained or instructed to maintain and use adequate safety equipment
- 3) to attend a course of instruction in first aid

User A user is a person installing, commissioning or using a product which has been placed on the market.

3.2 General Information

3.2.1 Using the Safety Instructions and Passing Them on to Others

Do not attempt to install and operate the components of the electric drive and control system without first reading all documentation provided with the product. Read and understand these safety instructions and all user documentation prior to working with these components. If you do not have the user documentation for the components, contact your responsible Bosch Rexroth sales partner. Ask for these documents to be sent immediately to the person or persons responsible for the safe operation of the components.

If the component is resold, rented and/or passed on to others in any other form, these safety instructions must be delivered with the component in the official language of the user's country.

Improper use of these components, failure to follow the safety instructions in this document or tampering with the product, including disabling of safety devices, could result in property damage, injury, electric shock or even death.

3.2.2 Requirements for Safe Use

Read the following instructions before initial commissioning of the components of the electric drive and control system in order to eliminate the risk of injury and/or property damage. You must follow these safety instructions.

- Bosch Rexroth is not liable for damages resulting from failure to observe the safety instructions.
- Read the operating, maintenance and safety instructions in your language before commissioning. If you find that you cannot completely understand the application documentation in the available language, please ask your supplier to clarify.
- Proper and correct transport, storage, mounting and installation, as well as care in operation and maintenance, are prerequisites for optimal and safe operation of the component.
- Only qualified persons may work with components of the electric drive and control system or within its proximity.
- Only use accessories and spare parts approved by Bosch Rexroth.
- Follow the safety regulations and requirements of the country in which the components of the electric drive and control system are operated.
- Only use the components of the electric drive and control system in the manner that is defined as appropriate. See chapter "Appropriate Use".
- The ambient and operating conditions given in the available application documentation must be observed.
- Applications for functional safety are only allowed if clearly and explicitly specified in the application documentation "Integrated Safety Technology". If this is not the case, they are excluded. Functional safety is a safety

Safety Instructions for Electric Drives and Controls

concept in which measures of risk reduction for personal safety depend on electrical, electronic or programmable control systems.

- The information given in the application documentation with regard to the use of the delivered components contains only examples of applications and suggestions.

The machine and installation manufacturers must

- make sure that the delivered components are suited for their individual application and check the information given in this application documentation with regard to the use of the components,
- make sure that their individual application complies with the applicable safety regulations and standards and carry out the required measures, modifications and complements.
- Commissioning of the delivered components is only allowed once it is sure that the machine or installation in which the components are installed complies with the national regulations, safety specifications and standards of the application.
- Operation is only allowed if the national EMC regulations for the application are met.
- The instructions for installation in accordance with EMC requirements can be found in the section on EMC in the respective application documentation.

The machine or installation manufacturer is responsible for compliance with the limit values as prescribed in the national regulations.

- The technical data, connection and installation conditions of the components are specified in the respective application documentations and must be followed at all times.

National regulations which the user must take into account

- European countries: In accordance with European EN standards
- United States of America (USA):
 - National Electrical Code (NEC)
 - National Electrical Manufacturers Association (NEMA), as well as local engineering regulations
 - Regulations of the National Fire Protection Association (NFPA)
- Canada: Canadian Standards Association (CSA)
- Other countries:
 - International Organization for Standardization (ISO)
 - International Electrotechnical Commission (IEC)

3.2.3 Hazards by Improper Use

- High electrical voltage and high working current! Danger to life or serious injury by electric shock!
- High electrical voltage by incorrect connection! Danger to life or injury by electric shock!
- Dangerous movements! Danger to life, serious injury or property damage by unintended motor movements!
- Health hazard for persons with heart pacemakers, metal implants and hearing aids in proximity to electric drive systems!
- Risk of burns by hot housing surfaces!

Safety Instructions for Electric Drives and Controls

- Risk of injury by improper handling! Injury by crushing, shearing, cutting, hitting!
- Risk of injury by improper handling of batteries!
- Risk of injury by improper handling of pressurized lines!

3.3 Instructions with Regard to Specific Dangers

3.3.1 Protection Against Contact With Electrical Parts and Housings



This section concerns components of the electric drive and control system with voltages of **more than 50 volts**.

Contact with parts conducting voltages above 50 volts can cause personal danger and electric shock. When operating components of the electric drive and control system, it is unavoidable that some parts of these components conduct dangerous voltage.

High electrical voltage! Danger to life, risk of injury by electric shock or serious injury!

- Only qualified persons are allowed to operate, maintain and/or repair the components of the electric drive and control system.
- Follow the general installation and safety regulations when working on power installations.
- Before switching on, the equipment grounding conductor must have been permanently connected to all electric components in accordance with the connection diagram.
- Even for brief measurements or tests, operation is only allowed if the equipment grounding conductor has been permanently connected to the points of the components provided for this purpose.
- Before accessing electrical parts with voltage potentials higher than 50 V, you must disconnect electric components from the mains or from the power supply unit. Secure the electric component from reconnection.
- With electric components, observe the following aspects:
Always wait **30 minutes** after switching off power to allow live capacitors to discharge before accessing an electric component. Measure the electrical voltage of live parts before beginning to work to make sure that the equipment is safe to touch.
- Install the covers and guards provided for this purpose before switching on.
- Never touch electrical connection points of the components while power is turned on.
- Do not remove or plug in connectors when the component has been powered.
- Under specific conditions, electric drive systems can be operated at mains protected by residual-current-operated circuit-breakers sensitive to universal current (RCDs/RCMs).
- Secure built-in devices from penetrating foreign objects and water, as well as from direct contact, by providing an external housing, for example a control cabinet.

High housing voltage and high leakage current! Danger to life, risk of injury by electric shock!

- Before switching on and before commissioning, ground or connect the components of the electric drive and control system to the equipment grounding conductor at the grounding points.

Safety Instructions for Electric Drives and Controls

- Connect the equipment grounding conductor of the components of the electric drive and control system permanently to the main power supply at all times. The leakage current is greater than 3.5 mA.
- Establish an equipment grounding connection with a minimum cross section according to the table below. With an outer conductor cross section smaller than 10 mm² (8 AWG), the alternative connection of two equipment grounding conductors is allowed, each having the same cross section as the outer conductors.

Cross section outer conductor	Minimum cross section equipment grounding conductor Leakage current ≥ 3.5 mA	
	1 equipment grounding conductor	2 equipment grounding conductors
1.5 mm ² (16 AWG)	10 mm ² (8 AWG)	2 × 1.5 mm ² (16 AWG)
2.5 mm ² (14 AWG)		2 × 2.5 mm ² (14 AWG)
4 mm ² (12 AWG)		2 × 4 mm ² (12 AWG)
6 mm ² (10 AWG)		2 × 6 mm ² (10 AWG)
10 mm ² (8 AWG)		-
16 mm ² (6 AWG)	16 mm ² (6 AWG)	-
25 mm ² (4 AWG)		-
35 mm ² (2 AWG)		-
50 mm ² (1/0 AWG)	25 mm ² (4 AWG)	-
70 mm ² (2/0 AWG)	35 mm ² (2 AWG)	-
...	...	...

Fig. 3-1: Minimum Cross Section of the Equipment Grounding Connection

3.3.2 Protective Extra-Low Voltage as Protection Against Electric Shock

Protective extra-low voltage is used to allow connecting devices with basic insulation to extra-low voltage circuits.

On components of an electric drive and control system provided by Bosch Rexroth, all connections and terminals with voltages between 5 and 50 volts are PELV ("Protective Extra-Low Voltage") systems. It is allowed to connect devices equipped with basic insulation (such as programming devices, PCs, notebooks, display units) to these connections.

Danger to life, risk of injury by electric shock! High electrical voltage by incorrect connection!

If extra-low voltage circuits of devices containing voltages and circuits of more than 50 volts (e.g., the mains connection) are connected to Bosch Rexroth products, the connected extra-low voltage circuits must comply with the requirements for PELV ("Protective Extra-Low Voltage").

3.3.3 Protection Against Dangerous Movements

Dangerous movements can be caused by faulty control of connected motors. Some common examples are:

Safety Instructions for Electric Drives and Controls

- Improper or wrong wiring or cable connection
- Operator errors
- Wrong input of parameters before commissioning
- Malfunction of sensors and encoders
- Defective components
- Software or firmware errors

These errors can occur immediately after equipment is switched on or even after an unspecified time of trouble-free operation.

The monitoring functions in the components of the electric drive and control system will normally be sufficient to avoid malfunction in the connected drives. Regarding personal safety, especially the danger of injury and/or property damage, this alone cannot be relied upon to ensure complete safety. Until the integrated monitoring functions become effective, it must be assumed in any case that faulty drive movements will occur. The extent of faulty drive movements depends upon the type of control and the state of operation.

Dangerous movements! Danger to life, risk of injury, serious injury or property damage!

A **risk assessment** must be prepared for the installation or machine, with its specific conditions, in which the components of the electric drive and control system are installed.

As a result of the risk assessment, the user must provide for monitoring functions and higher-level measures on the installation side for personal safety. The safety regulations applicable to the installation or machine must be taken into consideration. Unintended machine movements or other malfunctions are possible if safety devices are disabled, bypassed or not activated.

To avoid accidents, injury and/or property damage:

- Keep free and clear of the machine's range of motion and moving machine parts. Prevent personnel from accidentally entering the machine's range of motion by using, for example:
 - Safety fences
 - Safety guards
 - Protective coverings
 - Light barriers
- Make sure the safety fences and protective coverings are strong enough to resist maximum possible kinetic energy.
- Mount emergency stopping switches in the immediate reach of the operator. Before commissioning, verify that the emergency stopping equipment works. Do not operate the machine if the emergency stopping switch is not working.
- Prevent unintended start-up. Isolate the drive power connection by means of OFF switches/OFF buttons or use a safe starting lockout.
- Make sure that the drives are brought to safe standstill before accessing or entering the danger zone.
- Additionally secure vertical axes against falling or dropping after switching off the motor power by, for example,
 - mechanically securing the vertical axes,
 - adding an external braking/arrester/clamping mechanism or
 - ensuring sufficient counterbalancing of the vertical axes.

Safety Instructions for Electric Drives and Controls

- The standard equipment **motor holding brake** or an external holding brake controlled by the drive controller is **not sufficient to guarantee personal safety!**
- Disconnect electrical power to the components of the electric drive and control system using the master switch and secure them from reconnection ("lock out") for:
 - Maintenance and repair work
 - Cleaning of equipment
 - Long periods of discontinued equipment use
- Prevent the operation of high-frequency, remote control and radio equipment near components of the electric drive and control system and their supply leads. If the use of these devices cannot be avoided, check the machine or installation, at initial commissioning of the electric drive and control system, for possible malfunctions when operating such high-frequency, remote control and radio equipment in its possible positions of normal use. It might possibly be necessary to perform a special electromagnetic compatibility (EMC) test.

3.3.4 Protection Against Magnetic and Electromagnetic Fields During Operation and Mounting

Magnetic and electromagnetic fields generated by current-carrying conductors or permanent magnets of electric motors represent a serious danger to persons with heart pacemakers, metal implants and hearing aids.

Health hazard for persons with heart pacemakers, metal implants and hearing aids in proximity to electric components!

- Persons with heart pacemakers and metal implants are not allowed to enter the following areas:
 - Areas in which components of the electric drive and control systems are mounted, commissioned and operated.
 - Areas in which parts of motors with permanent magnets are stored, repaired or mounted.
- If it is necessary for somebody with a heart pacemaker to enter such an area, a doctor must be consulted prior to doing so. The noise immunity of implanted heart pacemakers differs so greatly that no general rules can be given.
- Those with metal implants or metal pieces, as well as with hearing aids, must consult a doctor before they enter the areas described above.

3.3.5 Protection Against Contact With Hot Parts

Hot surfaces of components of the electric drive and control system. Risk of burns!

- Do not touch hot surfaces of, for example, braking resistors, heat sinks, supply units and drive controllers, motors, windings and laminated cores!
- According to the operating conditions, temperatures of the surfaces can be **higher than 60 °C (140 °F)** during or after operation.
- Before touching motors after having switched them off, let them cool down for a sufficient period of time. Cooling down can require **up to 140 minutes!** The time required for cooling down is approximately five times the thermal time constant specified in the technical data.

Safety Instructions for Electric Drives and Controls

- After switching chokes, supply units and drive controllers off, wait **15 minutes** to allow them to cool down before touching them.
- Wear safety gloves or do not work at hot surfaces.
- For certain applications, and in accordance with the respective safety regulations, the manufacturer of the machine or installation must take measures to avoid injuries caused by burns in the final application. These measures can be, for example: Warnings at the machine or installation, guards (shieldings or barriers) or safety instructions in the application documentation.

3.3.6 Protection During Handling and Mounting

Risk of injury by improper handling! Injury by crushing, shearing, cutting, hitting!

- Observe the relevant statutory regulations of accident prevention.
- Use suitable equipment for mounting and transport.
- Avoid jamming and crushing by appropriate measures.
- Always use suitable tools. Use special tools if specified.
- Use lifting equipment and tools in the correct manner.
- Use suitable protective equipment (hard hat, safety goggles, safety shoes, safety gloves, for example).
- Do not stand under hanging loads.
- Immediately clean up any spilled liquids from the floor due to the risk of falling!

3.3.7 Battery Safety

Batteries consist of active chemicals in a solid housing. Therefore, improper handling can cause injury or property damage.

Risk of injury by improper handling!

- Do not attempt to reactivate low batteries by heating or other methods (risk of explosion and cauterization).
- Do not attempt to recharge the batteries as this may cause leakage or explosion.
- Do not throw batteries into open flames.
- Do not dismantle batteries.
- When replacing the battery/batteries, do not damage the electrical parts installed in the devices.
- Only use the battery types specified for the product.



Environmental protection and disposal! The batteries contained in the product are considered dangerous goods during land, air, and sea transport (risk of explosion) in the sense of the legal regulations. Dispose of used batteries separately from other waste. Observe the national regulations of your country.

Safety Instructions for Electric Drives and Controls

3.3.8 Protection Against Pressurized Systems

According to the information given in the Project Planning Manuals, motors and components cooled with liquids and compressed air can be partially supplied with externally fed, pressurized media, such as compressed air, hydraulics oil, cooling liquids and cooling lubricants. Improper handling of the connected supply systems, supply lines or connections can cause injuries or property damage.

Risk of injury by improper handling of pressurized lines!

- Do not attempt to disconnect, open or cut pressurized lines (risk of explosion).
- Observe the respective manufacturer's operating instructions.
- Before dismantling lines, relieve pressure and empty medium.
- Use suitable protective equipment (safety goggles, safety shoes, safety gloves, for example).
- Immediately clean up any spilled liquids from the floor due to the risk of falling!



Environmental protection and disposal! The agents (e.g., fluids) used to operate the product might not be environmentally friendly. Dispose of agents harmful to the environment separately from other waste. Observe the national regulations of your country.

3.4 Explanation of Signal Words and the Safety Alert Symbol

The Safety Instructions in the available application documentation contain specific signal words (DANGER, WARNING, CAUTION or NOTICE) and, where required, a safety alert symbol (in accordance with ANSI Z535.6-2006).

The signal word is meant to draw the reader's attention to the safety instruction and identifies the hazard severity.

The safety alert symbol (a triangle with an exclamation point), which precedes the signal words DANGER, WARNING and CAUTION, is used to alert the reader to personal injury hazards.

⚠ DANGER

In case of non-compliance with this safety instruction, death or serious injury **will** occur.

⚠ WARNING

In case of non-compliance with this safety instruction, death or serious injury **could** occur.

⚠ CAUTION

In case of non-compliance with this safety instruction, minor or moderate injury **could** occur.

Safety Instructions for Electric Drives and Controls

NOTICE

In case of non-compliance with this safety instruction, property damage could occur.

4 Introduction

In addition to the functionalities that an automation solution offers for the automation of machines and systems, effective engineering is increasingly important. The key component is to map an application quickly using efficient program code.

Knowing which aspects of machine automation must be taken into consideration is the main requirement. In addition, coordinating the movement of the axes plays a significant role for some of the specific features in the automation of Motion applications.

For these reasons, the Bosch Rexroth **Generic Application Template (GAT)** provided helps users develop clearly arranged Logic applications quickly and easily. In addition, it takes the specific features of Motion applications into consideration.

This documentation describes how the GAT is used.

The section [Basic Procedures for Automating Applications, page 33](#), demonstrates the basic procedures for automating applications.

Three versions of the GAT are available, depending on the complexity of the application.

- [GATcompact, page 121](#), example program for simple applications with few axes
- [GAT, page 165](#), program framework for flexible, modular machine concepts with **central** operation mode SFC.
- [GAT, page 233](#), program framework for flexible, modular machine concepts with **decentral** operation mode SFC.

GAT and GAT^{compact} use a simplified form of access, based on PLCopen, for axes (AxisInterface) and the control (IndraMotion Control Interface). For a description of AxisInterface or IMC interface¹⁾ see [chapter 6 "ML_TechInterface.library" on page 47](#).

Both versions of the Generic Application Template are integral components of the IndraWorks Engineering Framework.



The GAT with centralized and decentralized operation mode SFC requires an IndraMotion MLC control (IndraLogic XLC is not supported).

GAT^{compact} and the AxisInterface example project support both IndraMotion MLC and IndraLogic XLC controls.

GAT Wizard

The GAT Wizard is a tool that is integrated into IndraWorks that provides the user with support when importing and editing a GAT based project.

The GAT Wizard is opened using the GAT button  in the IndraWorks Toolbar.



The GAT Wizard always refers to the active application in the IndraWorks project.

¹⁾ *IndraMotionControl Interface = IndraLogic library with an interface to the control, e.g. clear errors*

5 Automating Applications

5.1 General Information

Based on experience, the following tasks must be carried out when automating a machine:

- Initialization
- Operation mode management
- Cyclical processing
- Task handling / computing time allocation
- Diagnostics / error management
- Visualization

The following section describes the basic procedures for carrying out these tasks also with regard to Motion application specifics. With regard to the communication between the controls and the drives in this context, the SERCOS bus is particularly of interest. The IndraWorks Engineering framework also takes task and error management as well as visualization into consideration.

5.2 Initialization

When the machine starts to power up, most of the parameters and variables in the control and the associated drives specific to the system are set in order to prepare for operation. This procedure is called initialization.

There are two different types of initialization: in communication phase 2 (P2, parameterization mode) and communication phase 4 (BB, operating mode).

Communication phase 2 means that the control and drives are in parameterization mode. Many parameters that can be written in this mode can no longer be written in communication phase 4 (operating mode).

In communication phase 2 the following is carried out as follows:

- Usually the communication parameters are transferred from the master to the drive (for field bus and SERCOS devices).
- If necessary, the "Load drive parameters" and "Save drive parameters" ("File service") functions are performed.

Typically, the following activities are carried out during the initialization in order to prepare the machine for operation:

- Drive firmware is checked
- Existing hardware is scanned
- Axes are configured
- Control and drive parameters are checked and/or set
- PLC variables/FBs are set/reset
- At the end of the initialization, the torque is added to the axis
- System phase switching into operating mode (P4)

5.3 Operation Modes

5.3.1 General Information

Typical operation modes for each application are **Automatic** and **Manual**. There may be further operation modes (e.g. Cleaning).

Automating Applications

Normally an operation mode change occurs in Motion applications only when all axes are stopped.

5.3.2 Automatic

In the Automatic operation mode in Motion applications, the axes generally move in coordination with each other. In this way, often one or more slave axes follow a master axis with synchronous velocity or position. In addition, a slave axis can traverse a MotionProfile (cam) specific to the application. The axes move without operator influence.

In order to achieve better management or a better overview in complex applications, the Automatic operation mode is often subdivided into several states.

5.3.3 Manual

The Manual operation mode is used to set up or adjust the machine. Typically, each axis is moved using a certain velocity profile or incrementally (jogged). The axes can also be moved synchronously. Depending on the degree of complexity, the Manual operation mode can also be subdivided into several states.

5.4 Cyclical Processing

5.4.1 General Information

All automation tasks that are not dependent on the operation mode should be combined in their own program section. This section can then be processed, e.g. within a cyclical task.

The following section describes typical automation tasks for cyclical processing.

5.4.2 Operation Mode Independent Logic

Normally, the logic that does not depend on the operation mode is handled in a separate program section. In general, this section is processed cyclically. All tasks that do not depend on the operation mode should be programmed in this cyclical task (e.g. do not allow jogging during Automatic operation).

5.4.3 Generating and Displaying Diagnostic Messages

During everyday system use, users need current information on its status (e.g. current operation mode, temperature, hydraulic pressure, ...). To provide this necessary information, it must be collected at a central location and adapted into meaningful status messages. These are provided for display on the HMI, for example. For further information, please see [chapter 5.7 "Visualization" on page 43](#).

5.5 Task Management

5.5.1 General Information

One subtask in automation is the practical division of the application into tasks. At the same time, the computing time must be measured in order to evaluate the load on the system.

5.5.2 Principles of Task Management

The following principles always have to be observed:

- As many tasks as necessary, as few as possible
 - Multitasking increases system complexity
 - Task switching requires more computing time

Automating Applications

- PLC tasks are always to be performed with a watchdog (exception: free running tasks BackgroundTask). Otherwise, unexpected system behavior can result.
 - On: Watchdog time=Task cycle time and sensitivity=1
 - Normal: Watchdog time=(Task cycle time+50 to 100%) and sensitivity=2
 - Optional watchdog in the application program as an early warning: Time measurement functions can be used to implement a "warning watchdog" in the application program with 90% of the time of the system watchdog, for example. In this way, a warning can be generated and countermeasures can be taken.
- A system is under optimum load if:
 - the smallest control variant possible can be used for this task
 - all tasks of the control are executed on time
 - no PLC task watchdog (worst case) occurs in any of the operating states

5.5.3 Recommended Task Division when Using IndraWorks/IndraLogic

General Information

If the process takes longer than the set watchdog time and the watchdog is activated in IndraWorks/IndraLogic for this task, this task will be interrupted with error status (exception error). In this case, sensitivity is calculated in as follows:

- If the watchdog time is exceeded several times in a row, there are:
 - Exception errors in cycle 1 if sensitivity is 1 or 2
 - Exception errors in cycle 2 if sensitivity is 3
 - ...
 - Exception errors in cycle (n-1) if sensitivity is n
- If the watchdog time is exceeded once, an exception error occurs if the current cycle time is greater than (watchdog time * sensitivity).

For example: Current cycle time = 10 ms, sensitivity = 5 -> exception error if the task exceeds 50 ms once.

Based on this information, the following task and the associated time-related settings are recommended.

	Priority	Cycle time	Watchdog time	Watchdog sensitivity
MotionTask (optional with GAT ^{compact})	2	=SERCOS cycle time	T#2 to 10 ms	1..2
PlcTask	~5	T#10 to 50 ms	T#10 to 100 ms	2..3
BackgroundTask (optional with GAT ^{compact})	~6	T#40 to 100 ms	-	-

Typical Tasks for the MotionTask

In general, the MotionTask is reserved for certain tasks:

- Handling gantry axes and axes with special requirements
- Handling technology function blocks with special time requirements (winders, register control, SmartBelt, flying shears, ...)
- Fast error reactions
- Handling Fast I/O

Automating Applications

Typical Tasks for the PlcTask

The PlcTask is the central location for all standard tasks:

- Handling the axes, operation mode switching
- Machine sequences
- Standard I/O
- Error management
- Support for safety technology

Typical Tasks for the BackgroundTask

The optional BackgroundTask is used to reduce the load on the PlcTask. All tasks that are not time critical can be handled here, e.g.:

- Generating and displaying diagnostic messages
- Preparing data for visualization
- Complex calculations (e.g. cams) to prepare for the next product change

5.5.4 Measuring Cycle Times

Monitoring Dialog for Task Configuration

Taskkonfiguration[IndraMotionMlc1: Logic: Application]						
Properties		Monitor				
Task	Status	Cycle Count	Last Cycle Time (µs)	Average Cycle Time (µs)	Max. Cycle Time (µs)	Min. Cycle Time (µs)
 GATBackgroundProgTask	Gültig	7078	5	5	58	3
 GATMotionSyncTask	Gültig	141560	10	9	69	4
 GATProgTask	Gültig	28312	101	101	153	70
 VISU_TASK	Gültig	2832	71	70	419	3

Fig.5-1: Cycle times for PLC tasks

Cycle Time Dialog of the Control

Cycle Times - IndraMotionMlc1

IndraMotionMlc1

Cycle time settings

Desired MC cycle time (T_{cyc})

 us

SERCOS-Cycletime

 us

Sensitivity of the motion calculation

Motion-watchdog sensibility

Absolute error counter of the motion

Successive errors of the motion

Reset Error memory

Motion time display

	Minimum	Current	Maximum	
Actual MC cycle time (T _{cyc})		2000		us
Time for actual value acquisition	2	25	58	us
Pausing by motion-synchronous PLC tasks	0	29	88	us
Time for command value creation	0	7	18	us
Total time	2	62	125	us

Reset Minimum/Maximum

Fig. 5-2: Division of the cycle times for Motion Control and PLC

Automating Applications

Task Execution Viewer of the Control

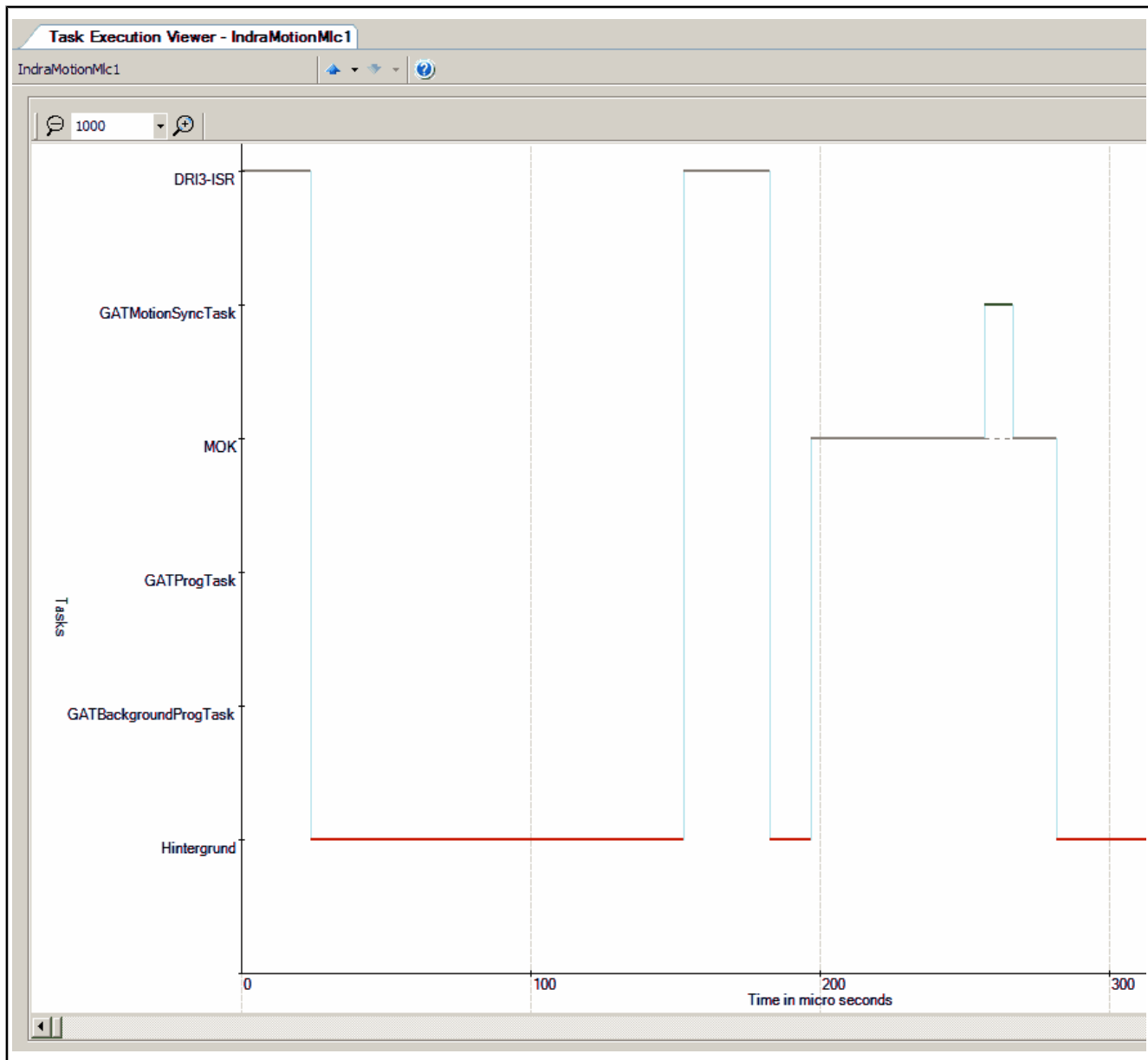


Fig. 5-3: Time sequence for the PLC and selected system tasks

Time Measurement in the PLC Project

In order to measure individual code sections, the control system provides high resolution time measurement functions. These are:

- IL_HighResTimeTick and
- IL_HighResTimeDiff

in the RIL_UTILITIES library.

PLC_load[IndraMotionMlc1: Logic: Application]

PLC task times (in uSec):

Minimum Time	Actual Time	MaximumTime	Clear values
36	52	70	

TE_AxisInterfaceType01Prog[IndraMotionMlc1: Logic: Application]

IndraMotionMlc1.Application.TE_AxisInterfaceType01Prog

Expression	Comment	Type	Value
ClearError	clear internal errors of the program	BOOL	FALSE

```

90
91 udiStartTime[16#1D367869] := IL_HighResTimeTick(TRUE); // latch starting time
92
93 IF (InitDone[TRUE] AND bInitReferencesDone[TRUE]) THEN
94   FOR uiLoop[16#0011] := 1 TO uiMAX_AXIS_INDEX DO
95     fbAxis[uiLoop[16#0011]]( ); // call FB, inputs are already set....
96   END_FOR
97 END_IF
98
99 udiStopTime[16#1D367891] := IL_HighResTimeTick(TRUE); // latch end time
100
101 // calculate actual runtime and store MIN and MAX
102 udiActCycleTime[16#00000028] := IL_HighResTimeDiff(udiStartTime[16#1D367869],udiStopTime[16#1D367891]);
103
104 stCyclesTime.udiActualTime[16#00000028] := udiActCycleTime[16#00000028];
105
106 IF (stCyclesTime.udiMaximumTime[16#00000046] < udiActCycleTime[16#00000028]) THEN
107   stCyclesTime.udiMaximumTime[16#00000046] := udiActCycleTime[16#00000028];
108 END_IF
109
110 IF (stCyclesTime.udiMinimumTime[16#00000024] > udiActCycleTime[16#00000028]) THEN
111   stCyclesTime.udiMinimumTime[16#00000024] := udiActCycleTime[16#00000028];
112 END_IF[RETURN]

```

Fig.5-4: Time measurement in the PLC project

Time measurement functions can be used to implement a "warning watchdog" in the application program with 90% of the time of the system watchdog, for example. In this way, an error/warning can be implemented that does not lead to a machine stop. In contrast, the system watchdog stops the PLC.

5.5.5 Application-specific Watchdog

If a watchdog is entered in the IndraLogic task configuration, the PLC program is interrupted as soon as this watchdog responds. In this way, no further reactions based on this event can occur in the application within the PLC program.

By implementing an application-specific watchdog, i.e. internal cycle time measurement, in case the cycle time is exceeded, a reaction can still take place within the PLC program. The functions used for this are described in [chapter "Time Measurement in the PLC Project" on page 38](#).

5.6 Error Management

5.6.1 General Information

Special requirements are made for industrial automation systems in the area of error management. It must be ensured that damages are minimized if errors occur. Furthermore, downtime is to be reduced to a minimum. This can be achieved with meaningful diagnostics and defined restart procedures.

The following section describes error management more specifically.

5.6.2 Principles of Error Management

Error management is subdivided into the following 4 subsections:

1. **Error detection, page 40**

All possible sources of error must be investigated. Examples of error sources include drives (AxisInterface), control (IMC interface), function block errors, peripheral errors (I/O, field bus), user-defined errors (material jam...), PLC errors...

2. **Error reaction, page 41**

In case of error, damage is to be limited to a minimum. This can normally be achieved with a defined shut down. A few components (such as drives) have error reactions in this case as well. In addition, a user-defined error reaction is often necessary as well (e.g. synchronized drive shut down so that the paper or film does not tear).

3. **Error diagnostics, page 42**

To reduce downtime, the cause for the error must be determined as quickly as possible. For distributed systems, the diagnostics should be provided at a central location. Furthermore, plain text should be used as much as possible for the messages. In some cases, the error displayed is an after effect that can only partially indicate the cause of the error. For this reason, a logbook is often very helpful as an additional diagnostic tool for finding the actual cause of the error and for a better analysis of the error scenario.

4. **Error acknowledgement / Restart after an error, page 43**

When the cause of the error is repaired, production mode must be reestablished as soon as possible. The error acknowledgement should only be made at a central location. Steps often also need to be taken at the application site in order to restore trouble-free production mode (e.g. resetting user-defined errors, starting the virtual master axis...)

5.6.3 Error Detection

Typically, when an application program is generated, the following sources of error are to be investigated and made available to the machine operator at a central location.

Error source	Query / Detection
Drive	- Without AxisInterface: AxisData [AxisRef.Axis-No].Error Bit - With AxisInterface: MB_AXIS_DIAGNOSIS.Error Bit
Control	- IndraLogic XLC/IndraMotion MLC control without IMC interface: ML_CONTROLDATA.C_0_0023.Bit26 - IndraMotion MLD control without IMC interface: P-0-0115.Bit13 - IndraLogic XLC/IndraMotion MLC and IndraMotion MLD control without IMC interface: MB_IMC_DIAGNOSIS.Error
Function block	Error output of the called FB
Periphery	Specific to field bus, e.g. library RIL_Profi-busDP_02.lib
Application-specific	User-defined

IMC interface IndraMotionControl Interface = IndraLogic library with an interface to the control, e.g. clear errors

5.6.4 Error Reaction

After an error occurs, an error reaction generally follows in order to prevent damage to the machine and material. Depending on the source of the error, the basic system will have already initiated an error reaction (see the following table). In addition, another user-specific error reaction can be implemented in the application program (e.g. defined shut down of a virtual master axis).

Error source	Reaction
Drive	- Reaction depending on degree of error severity: - F0 to F2: Best possible shut down -> in accordance with set parameterization - F6 to F7: Velocity command value, set to zero - F8: Torque enable - Only the axes affected by the error are shut down - The application program continues to run Caution If "30s NC Reaction" is set as the best possible shut down in the drive, the control system delays the real axis affected by the error by switching to AH.

Automating Applications

Error source	Reaction
Control	- IndraLogic XLC/IndraMotion MLC: All axes are shut down -> Fail safe shut down with deceleration ramp A-0-0224 (emergency deceleration) - IndraMotion MLD: Reaction can be parameterized - Application program does not continue to run in all cases: - SERCOS interruption: The application program continues to run - Cycle exceeded for Motion kernel: The application program does not continue to run
Function block	- Various, reaction in accordance with FB documentation - The application program continues to run
Periphery	- No reaction - The application program continues to run
Application-specific	- Reaction is user-specific - The application program continues to run

5.6.5 Error Diagnostics

To determine the cause of error, the components of the control system provide comprehensive diagnostic messages (see the following table). For this reason, the error messages for the drives and control also contain a degree of severity. In general, errors are to be indicated initially at the highest degree of severity. Diagnostic messages are to be summarized at a central location and made available to the machine operator in order to ensure a quick diagnosis.

Error source	Error diagnosis
Drive	- Without AxisInterface: A-0-0023 (only for XLC/MLC), S-0-0095, S-0-0390 - With AxisInterface: MB_AXIS_DIAGNOSIS.Message
Control	- IndraLogic XLC/IndraMotion MLC without IMC interface - C-0-0650, C-0-0627 - IndraMotion MLD without IMC interface - P-0-0115; P1351 - IndraMotion MLC & IndraMotion MLD with IMC interface - MB_IMC_DIAGNOSIS.Number - MB_IMC_DIAGNOSIS.Message
Function block	ErrorID, ErrorIdent at the FB output
Periphery	See field bus / Inline documentation
Application-specific	Application-specific

IMC interface IndraMotionControl Interface = IndraLogic library with an interface to the control, e.g. clear errors

5.6.6 Error Acknowledgement / Restart after an Error

After the cause of the error has been repaired, the error must be acknowledged at the source of the error. From the user's perspective, a **Clear central error** is required. For this reason, the error acknowledgement must be carried out from the central location to the causes of error in the application program. The following table shows how to clear the error at the respective subcomponents.

Error source	Error acknowledgement
Drive	- Without AxisInterface: Function block MC_Reset - With AxisInterface: MB_AXIS_ADMINISTRATION.ClearError
Control	- IndraLogic XLC/IndraMotion MLC without IMC interface - Start command C-0-1030 (using MB_Command) - IndraMotion MLD without IMC interface - MC_Reset function block with AxisRef = local axis - IndraMotion MLC & IndraMotion MLD with IMC interface - MB_IMC_ADMINISTRATION.ClearError
Function block	Falling edge at execute / Enable FB input
Periphery	See field bus / Inline documentation
Application-specific	Application-specific

IMC interface IndraMotionControl Interface = IndraLogic library with an interface to the control, e.g. clear errors

5.7 Visualization

5.7.1 General Information

A visualization should not just provide the data that are required for machine operation, but it should also provide support for error diagnostics.

5.7.2 Typical Contents of a Visualization

Machine Status Display

- Active operation mode
- Current diagnostics /diagnostic history (see [Error detection, page 40](#))
- Further information for support during commissioning, e.g. axis states (position, velocity, status, ...)
- Application-specific machine data

Controlling the Machine States

- Switching the operation mode (Automatic, Manual, ...)
- Acknowledging existing errors
- Switching the operation screens
- Operating application-specific machine functionalities

Automating Applications

5.7.3 Providing Data Using the PLC

General Information

The data to be visualized must be entered in the symbol file.

Symbol Configuration

The variables can be configured using the following access authorizations:

- No access
- Read
- Write
- Read and write

The symbol configuration can be carried out using a dialog in the "Symbol configuration" window, using "Pragma directives" in the variable declaration area of the POU or using variable lists.

A prerequisite for symbol configuration is that the PLC object "Symbol configuration" exists below the PLC application in the IndraWorks project tree. To do this, use drag and drop to move the object from the "PLC objects / General objects" library window and add it below the "Logic / Application" device node.

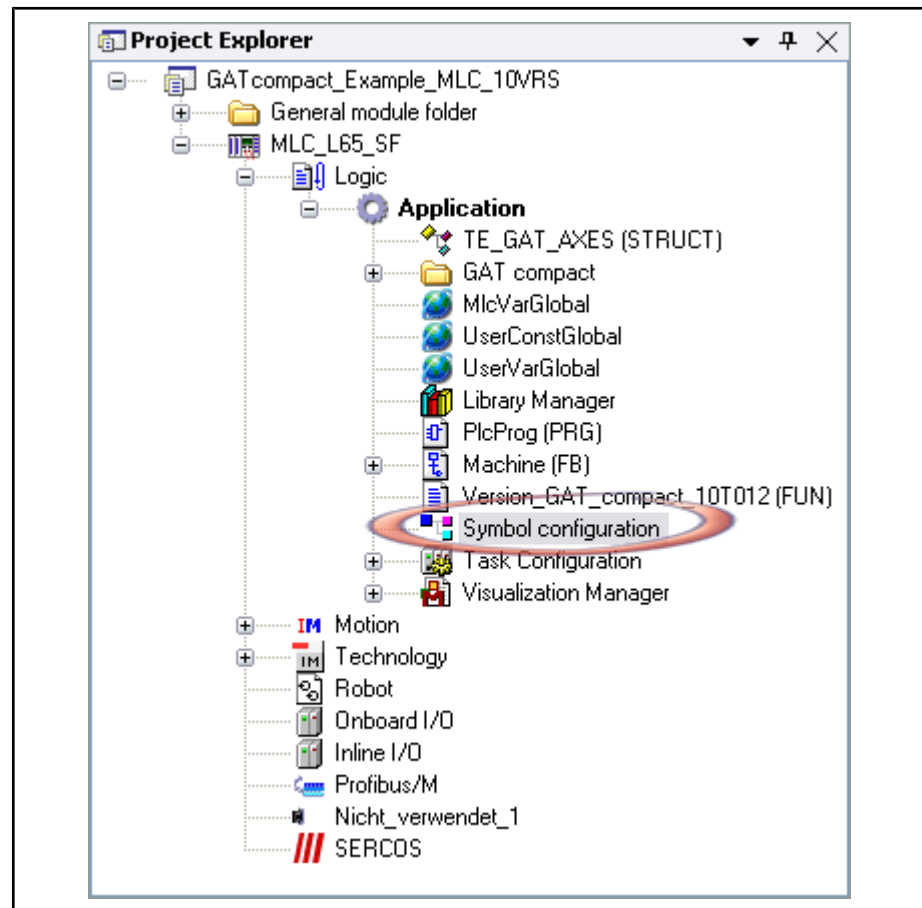


Fig.5-5: Symbol configuration in project tree

Symbol configuration using a dialog

The dialog for symbol configuration can be opened using the new "Symbol configuration" node in the IndraWorks project explorer. The symbol configuration window is divided into two areas:

Automating Applications

Available variables	List of all of variables in the PLC program with the respective maximum access authorization level. If no variables are listed, use the "Refresh" link to update the list. Note that it must be possible to compile the project without errors.
Selected variables	List of all of the variables configured for symbolic access with the respective configured access authorization. Select the "Display invalid variables" link to highlight all variables that no longer exist in the PLC project.

Select or delete variables by double-clicking on the element or by using the > and < buttons. This either selects or removes the selected element and all subordinate elements.

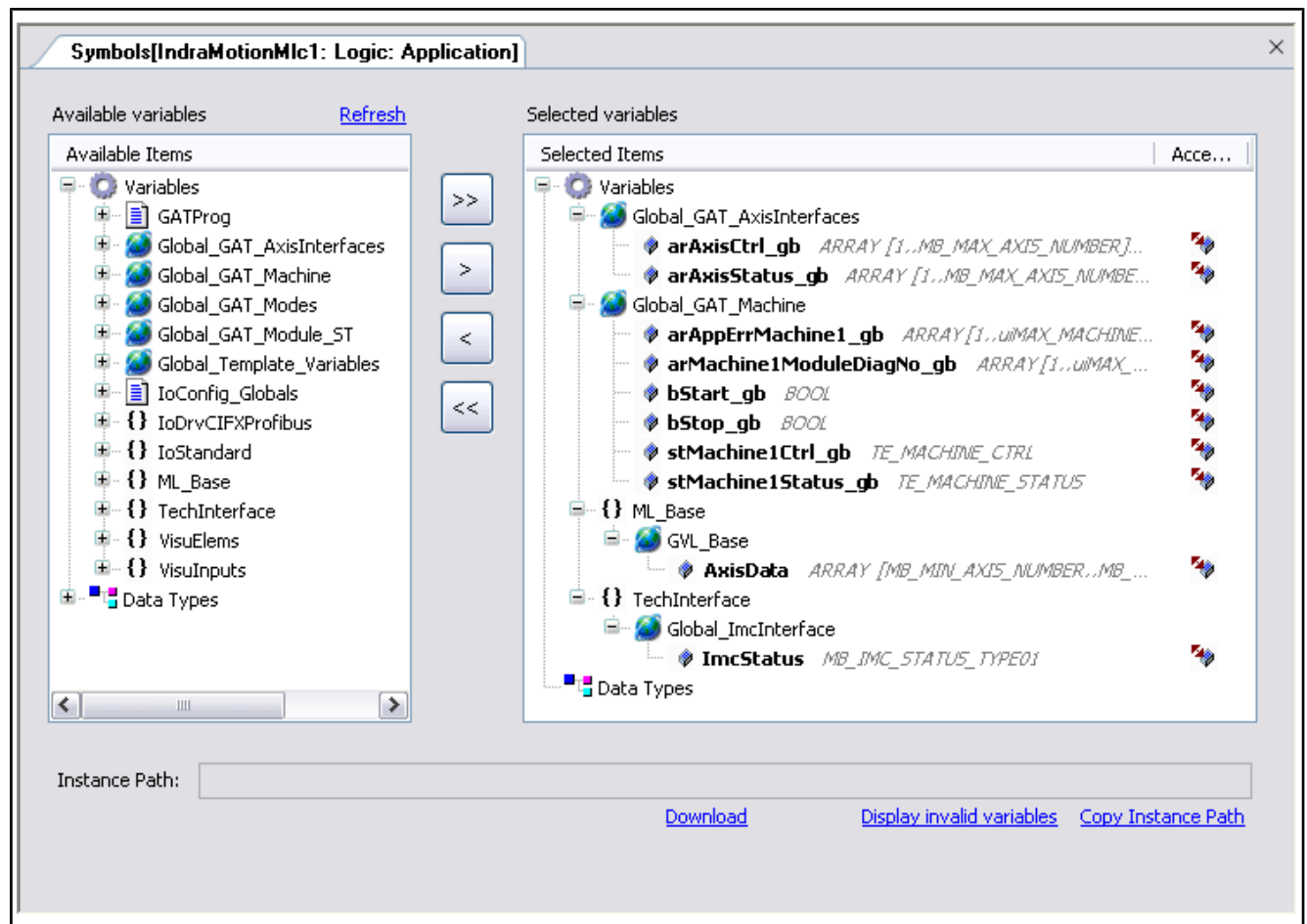


Fig. 5-6: Symbol configuration using a dialog

The access rights for the respective variable can be changed by clicking on the symbol in the column "Access rights" in the section "Selected variables".

Access rights are switched in the following order:

-  Read and write access
-  Write access
-  Read access



Symbol entries for variables from global variable lists that come from a library must be reinserted in the symbol configuration after the respective library version has been modified.

Symbol Configuration Using a Pragma

Automating Applications

Symbol configuration using "Pragma directives" is carried out directly in the variables' declaration area, e.g. in variable lists or in POU's.

The "{attribute 'symbol'}" attribute with the parameters for access authorization is available for pragma configuration. The possible configurations are shown in the following table.

Pragma directive	Description
{attribute 'symbol' := 'none'}	Access to the variables is blocked and they are not displayed in the browsing list of the BR-WS driver.
{attribute 'symbol' := 'read'}	Access to the variables is allowed only for read access and they are displayed in the browsing list of the BR-WS driver.
{attribute 'symbol' := 'write'}	Access to the variables is allowed only for write access and they are displayed in the browsing list of the BR-WS driver.
{attribute 'symbol'} {attribute 'symbol' := 'readwrite'}	Access to the variables is allowed for read and write access and they are displayed in the browsing list of the BR-WS driver.

BR-WS driver WinStudio communication driver from Bosch Rexroth

Depending on the position or the line in which the Pragma is written, it is valid for all variables in the declaration range or for just one variable.

- In order for a Pragma to affect just one variable, it must be written in the line just before the variable.
- If it is to affect all of the variables in the declaration area, it must appear in the first line of the declaration list or in the line just before the declaration area (e.g. just before **VAR GLOBAL...END_VAR**). In this case individual variables can still be assigned a different access authorization.

When the PLC application is downloaded, the symbol information is created in the IndraWorks "IndraLogic" project subdirectory as an XML file and is loaded into the control as a symbol application.

Communication Configuration

- The communication between the WinStudio application and the control is carried out using the BR-WS driver. This can be used to visualize the variables from the symbol file.
- To optimize communications performance, the variables should be distributed in various driver sheets that are updated at different intervals.

6 ML_TechInterface.library

6.1 Introduction and Overview

The TechInterface consists of the these components:

- AxisInterface and
- IMC interface (IMC = IndraMotion Control)

The **AxisInterface** pools and extends PLCopen motion function blocks and provides an easy-to-operate interface for the drive functionality. Less code and commands with a higher performance accelerate the development of the application programs. The AxisInterface contains control signals and parameters for the various operation modes of the master axis and slave axis as well as the setting options for the process values selected.

The **IMC interface** provides an easy-to-operate interface for the control functionality. The IMC interface includes control signals and parameters for the diagnostics and administration of the control.

The TechInterface is supported by both the IndraMotion MLC control system and the IndraMotion MLD drive system. The following table shows the supported target systems and libraries that contain the function blocks and data structures for the TechInterface:

System support	Drive firmware support	Library
IndraMotion MLC 10VRS or higher	MPx04VRS to MPx07VRS (MPx03VRS not supported)	ML_TechInterface.compiled-library
IndraMotion MLD drive system 07VRS or higher See documentation R911322053 "Rexroth IndraMotion AxisInterface for IndraLogic 1G"	MPx07VRS or higher	MX_AxisInterface.lib MX_ImcInterface.lib

Fig. 6-1: TechInterface libraries

TechInterface function blocks

Function block	Description
MB_AxisInitType01	Used to initialize the AxisInterface for a single axis. The function block must be called only once when the program is started or each time the phase is switched from the parameterization mode to the operating mode.
MB_AxisInterfaceType01	Used to configure the AxisInterface for a single axis. This function block must be called cyclically as long as the user is in the operating mode.
MB_GetAxisInterfaceIndex (function)	Calculates the matching index for use in the AxisInterface array from the transferred axis reference.
MB_ImcInterface (program)	Functionality for triggering and composing control status messages

Fig. 6-2: TechInterface

ML_TechInterface.library



The AxisInterface is provided as an IndraMotion programming template or as a standalone interface for the axis functionality.

If it is used with the GAT programming template (Generic Application Template) or GAT^{compact}, the user need not be concerned with instance calls of the FBs within the project. This functionality is completely integrated into the template - the user just needs to write a few lines of code.

But if the AxisInterface is used as a stand-alone functionality, the user must create the instances of both FBs for each axis. This section contains descriptions and examples on how to use the AxisInterface as a stand-alone functionality.

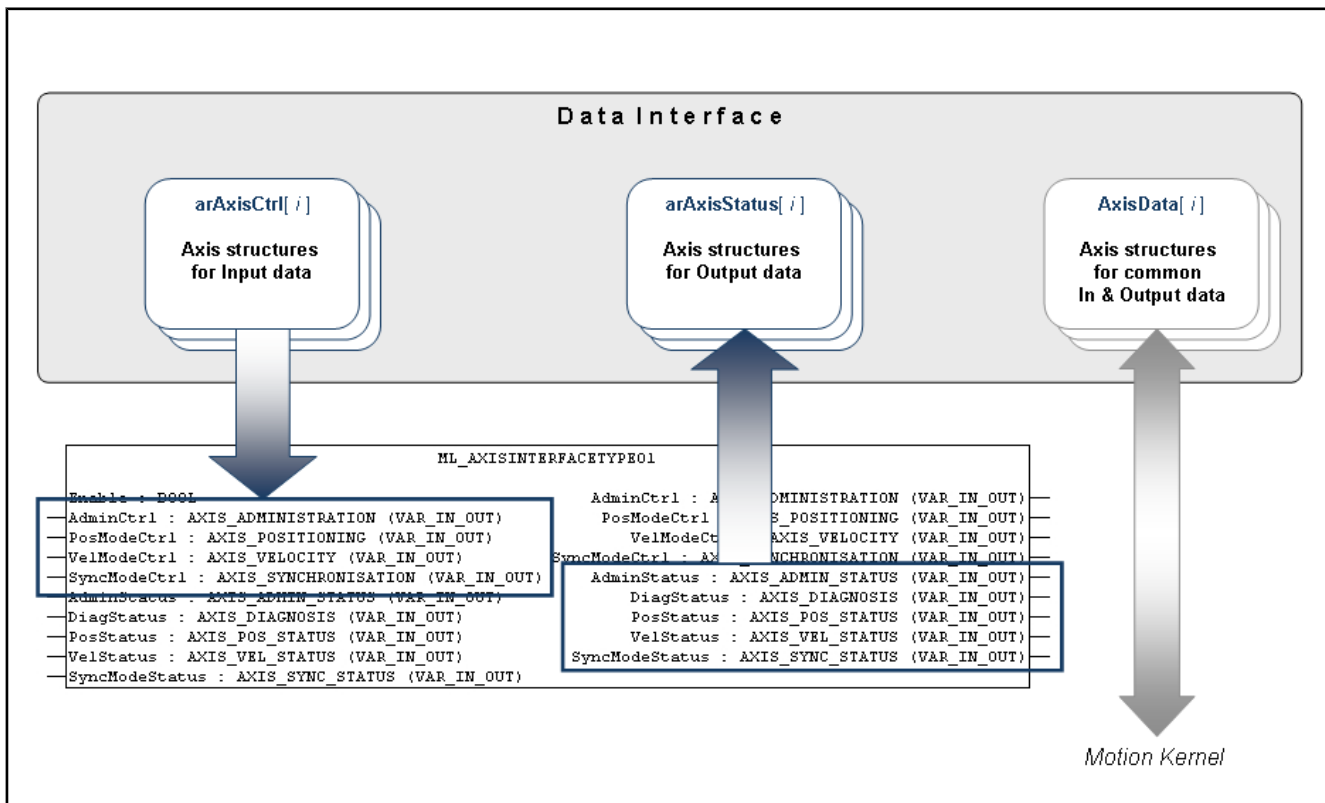


Fig.6-3: AxisInterface data structure of the interface

The following table shows the user interface for the AxisInterface data structure:

User interface	Type	Description
arAxisCtrl[]_gb	MB_AXIS_CONTROL_TYPE01	Control structure, including command values and variables to activate the operation modes
arAxisStatus[]_gb	MB_AXIS_STATUS_TYPE01	Status structure, including diagnostic information and acknowledgements for the operation modes
AxisData[]		Status structure and control structure, including actual values, status bits and access to cyclically configurable user data (processed by the MotionControl in the firmware).

User interface	Type	Description
ImcCtrl[]	MB_IMC_CONTROL_TYPE01	Control structure, including specification of the communication mode and diagnostic handling.
ImcStatus[]	MB_IMC_STATUS_TYPE01	Status structure, including status of the communication mode and diagnostic information.

Fig. 6-4: TechInterface – user interface data structures

The TYPE01 designation, which appears in the FB names and their structures, is planned for future extensions.



Between the various AxisInterface versions "01VRS" (MLC 03VRS), "02VRS" (MLC 04VRS) and the TechInterface "09VRS" (MLC09VRS) and "10VRS" (MLC10VRS) and newer versions, modifications were introduced, some of which require adaptations in the user program. The required adaptations are described in chapter 6.2 "Initial TechInterface Configuration" on page 53.

Ausdruck	Datentyp	Wert	Ausdruck	Datentyp	Wert
IndraMotionMlc1.Application.arAxisCtrl_gb[4]	TE_AXIS_CONTROL...		IndraMotionMlc1.Application.AxisData[4]	ML_AXISDA...	
Admin	TE_AXIS_ADMINIST...		wUserCmdDataBtA_q	WORD	0
Axis	AXIS_REF		wUserCmdDataBtB_q	WORD	0
DiagNbrRefreshTime	TIME	T#2s	wUserCmdDataBtC_q	WORD	0
StopDeceleration	REAL	99	wUserCmdDataBtD_q	WORD	0
WaitTimePowerRestart	UINT	0	dwUserCmdDataA_q	SM_TYPES	
ClearError	BOOL	FALSE	dwUserCmdDataB_q	SM_TYPES	
SetAbsRef	BOOL	FALSE	dwUserCmdDataC_q	SM_TYPES	
EnableCyclicScanning	BOOL	TRUE	dwUserCmdDataD_q	SM_TYPES	
UpdateEveryInput	BOOL	FALSE	dwUserCmdDataE_q	SM_TYPES	
DriveInterruptReaction	BOOL	TRUE	dwUserCmdDataF_q	SM_TYPES	
PowerOn	BOOL	TRUE	dwMotionProfileStatusBits_j	DWORD	0
_OpMode	MB_AXIS_MODE_UN...		dwFlexProfileEventControlBits_q	DWORD	0
en	MB_AXIS_MODE	ModeAH	FlexProfileSync	BOOL	FALSE
b	MB_AXIS_MODE_BITS		FlexProfileActive	BOOL	FALSE
RetriggerOpMode	BOOL	FALSE	FlexProfileWaitForSwitching	BOOL	FALSE
PostMode	TE_AXIS_POSITIONI...		FlexProfileExeSingleFinished	BOOL	FALSE
VelMode	TE_AXIS_VELOCITY		FlexProfileSet0_Checked	BOOL	FALSE
SyncMode	TE_AXIS_SYNCHRO...		FlexProfileSet0_Ok	BOOL	FALSE
SetupMode	TE_AXIS_SETUP_MO...		FlexProfileSet1_Checked	BOOL	FALSE
IndraMotionMlc1.Application.arAxisStatus_g...	TE_AXIS_STATUS_T...		FlexProfileSet1_Ok	BOOL	FALSE
PtrAxisData	POINTER TO ML_AX...	16#0A9233F8	FlexProfileSet2_Checked	BOOL	FALSE
Admin	TE_AXIS_ADMIN_ST...		FlexProfileSet2_Ok	BOOL	FALSE
Active	BOOL	TRUE	FlexProfileSet3_Checked	BOOL	FALSE
Name	STRING(40)	MyRealAxis3	FlexProfileSet3_Ok	BOOL	FALSE
CmdDone	BOOL	FALSE	FlexProfileEvtSet0_Checked	BOOL	FALSE
AxisType	MB_AXIS_CONFIG	RealAxisCfg	FlexProfileEvtSet0_Ok	BOOL	FALSE
AbsRefCmdStatus	MB_SERCOS_DATA...	MB_SERCOS_D...	FlexProfileEvtSet1_Checked	BOOL	FALSE
InternalFlags	DWORD	5	FlexProfileEvtSet1_Ok	BOOL	FALSE
_OpModeAck	MB_AXIS_MODE	ModeAH	FlexProfileEvtSet2_Checked	BOOL	FALSE
MODE_AH	BOOL	TRUE	FlexProfileEvtSet2_Ok	BOOL	FALSE
MODE_HOMING	BOOL	FALSE	FlexProfileEvtSet3_Checked	BOOL	FALSE
MODE_POS_ABS	BOOL	FALSE	FlexProfileEvtSet3_Ok	BOOL	FALSE
MODE_POS_REL	BOOL	FALSE	dwFlexProfileEventStatusBits_j	DWORD	0
MODE_POS_ADD	BOOL	FALSE	Axis_CamTab_0	BOOL	FALSE
MODE_VEL	BOOL	FALSE	Axis_CamTab_1	BOOL	FALSE
MODE_SYNC_PHASE	BOOL	FALSE	Axis_CamTab_2	BOOL	FALSE
MODE_SYNC_VEL	BOOL	FALSE	Axis_CamSwitching	BOOL	FALSE
MODE_SYNC_CAM	BOOL	FALSE	Axis_InVelocity	BOOL	TRUE
MODE_SYNC_PROFILE	BOOL	FALSE	Axis_Standstill	BOOL	FALSE
MODE_FLEX_PROFILE	BOOL	FALSE	Axis_InPosition	BOOL	FALSE
MODE_STOPPING	BOOL	FALSE	Axis_InSynchron	BOOL	FALSE
MODE_INTERRUPTED	BOOL	FALSE	Axis_Warning	BOOL	FALSE
MODE_EXTERNAL_FB	BOOL	FALSE	Axis_Error	BOOL	FALSE
Diag	TE_AXIS_DIAGNOSIS		Axis_Homed	BOOL	TRUE
PostMode	TE_AXIS_POS_STAT...		Axis_InTorque	BOOL	FALSE
VelMode	TE_AXIS_VEL_STAT...		Axis_OperationMode	BOOL	TRUE
SyncMode	TE_AXIS_SYNC_STA...		Axis_Inbb	BOOL	TRUE
SetupMode	TE_AXIS_SETUP_MO...		Axis_InAb	BOOL	TRUE
AxisData_HMI	TE_AXIS_DATA_STA...		Axis_Power	BOOL	TRUE
			Axis_CmdActive	BOOL	FALSE

Fig. 6-5: Overview of the AxisInterface data structures



Use the AxisNo of the AXIS_REF structure as an index for the array, e.g. arAxisCtrl_gb[MyVirtualAxis.AxisNo].Admin. etc.

EnableCyclicScanning

The internal handling of certain command values can be controlled by the arAxisCtrl_gb[.Admin.EnableCyclicScanning element. If EnableCyclicScanning is

ML_TechInterface.library

set to TRUE, some of the command values of the arAxisCtrl_gb[] structure are scanned cyclically and become effective immediately if a value changes.

Cyclic scanned	NOT Cyclic scanned
IndraMotionMlc1.Application.arAxisCtrl_gb[4] Admin Axis DiagHzRefreshTime StopDeceleration WaitTimePowerRestart ClearError SetAbolish EnableCyclicScanning UpdateEveryInput DriveInterruptReaction PowerOn OpMode en b MODE_AH MODE_HOMING MODE_POS_ABS MODE_POS_REL MODE_POS_ADD MODE_VEL MODE_SYNC_PHASE MODE_SYNC_VEL MODE_SYNC_CAM MODE_SYNC_PROFILE MODE_FLEX_PROFILE MODE_STOPPING MODE_INTERRUPTED MODE_EXTERNAL_FB RetriggerOpMode PosMode	TE_AXIS_CONTROL... TE_AXIS_ADMINST... AXIS_REF TIME T#2s REAL 99 UNIT 0 BOOL FALSE BOOL FALSE BOOL TRUE BOOL TRUE BOOL TRUE MB_AXIS_MODE_UN... ModeAt MB_AXIS_MODE_BITS MODE_AH MODE_HOMING MODE_POS_ABS MODE_POS_REL MODE_POS_ADD MODE_VEL MODE_SYNC_PHASE MODE_SYNC_VEL MODE_SYNC_CAM MODE_SYNC_PROFILE MODE_FLEX_PROFILE MODE_STOPPING MODE_INTERRUPTED MODE_EXTERNAL_FB RetriggerOpMode TE_AXIS_POSITIONI... Position REAL 0 Distance REAL 0 Velocity REAL 10 Deceleration REAL 10 MB_AXIS_ADJUST_D...

Fig. 6-6: Cyclically scanned elements of arAxisCtrl_gb[]

UpdateEveryInput

The data shown above, which are not cyclically scanned, are likewise accepted if a cyclically scanned element changes if the UpdateEveryInput input was set.

Example: the operation mode "relative positioning" was started with the values shown above, Velocity=10 and Acceleration=10 To start the next positioning procedure with a different acceleration, the input PosMode. Acceleration is changed from 10 to 20 and the next distance is written accordingly. The acceleration is also accepted with the change in the distance.



- During the activation of an operation mode (.Admin._Op-Mode), all input data is read regardless of the setting of the "EnableCyclicScanning" input.
- If EnableCyclicScanning = TRUE, all input data highlighted in green is read cyclically. This means that any change in values is read immediately after an operation mode was activated.
- In contrast to that, all input data highlighted in blue is not scanned cyclically. This means that the values are only read if an operation mode is activated.
- If UpdateEveryInput = TRUE, all input data highlighted in blue are also accepted if a cyclically scanned element belonging to this operation mode is modified.
- Data consistency is reached using EnableCyclicScanning (FALSE → write data → TRUE).

Various IndraLogic visualizations based on the five structure elements described in this section are available in the example project for commissioning purposes for details, see [chapter 6.6.3 "AxisInterface Visualizations" on page 76](#)).

What is new in version 02VRS (MLC04VRS)?

- Renaming:
All FBs were renamed from ML or MX to the comprehensive prefix MB in the IndraMotion MLC and IndraMotion MLD to ensure a system-inde-

pendent PLC programming. In all the data types, the prefix MB_ was supplemented.

- Diagnostic information:

A diagnostic number and a diagnostic text are now provided in the status structure `arAxisStatus[]`.Diag. Cyclic polling of the diagnostic number is controlled via `arAxisCtrl[]`.Admin.DiagNbrRefreshTime. If the diagnostic number changes, the diagnostic text is updated.

- FlexProfile operation mode:

The operation mode FlexProfile, newly introduced in MLC 04VRS, is supported by the AxisInterface. The profile-specific control variables for the operation modes "Electronic MotionProfile" and "Electronic FlexProfile" are now combined in the new structure `arAxisCtrl[]`.SyncMode.Profile.

- Pointer to AxisData:

The new structure element `arAxisStatus[]`.PtrAxisData permits access to the AxisData structure of the respective axis using pointers.

Example:

`arAxisStatus[i].PtrAxisData^.rActualPosition` has the same meaning as `AxisData[i].rActualPosition`.

As a result, the entire interface can be operated via the two structures `arAxisCtrl[]` and `arAxisStatus[]`. However, note that the pointer can be accessed only after the initialization was successfully completed (MB_AxisInitType01).

What is new in version 09VRS (MLC09VRS)?

- Merging AxisInterface and IMC interface into TechInterface:

In the context of the change to IndraLogic 2G, the libraries `ML_AxisInterface.lib` and `ML_ImcInterface.lib` were combined into one library, `ML_TechInterface.compiled-library`.

- Operation mode selection with UNION:

With IndraLogic 2G the operation mode selection `arAxisCtrl[MyRealAxis1.Axisno].Admin._OpMode` is implemented using a union data type "UNION" via an enumeration data type "ENUMERATION" and also in binary form via the new data type "BIT". In IndraLogic-1G a similar functionality was offered using "bitaccess". But this causes faulty assignments.

What is new in version 10VRS (MLC10VRS)?

- New input `arAxisCtrl_gb[]`.Admin.UpdateEveryInput: This input can be used to change even inputs that are not cyclically scanned during when an operation mode is active.

- New operation mode ModeStopping: the operation mode ModeStopping stops the PLCopen state machine in the "Stopping" state, i.e. a PLCopen function block called outside of the AxisInterface would be ended with the error "Command in state Stopping denied". The operation mode "ModeAH" now transfers the axis into the PLCopen state "StandStill", i.e. a PLCopen function block that is called outside of the AxisInterface is accepted and the AxisInterface reports "ModeExternalFB"

- New output `arAxisStatus_gb[]`.Admin.AbsRefCmdStatus: this output can be used to check the status of the command started using `arAxisCtrl_gb[]`.Admin.SetAbsRef.

- New output `ImcStatus`.PassiveMode: this output is set if the IMC interface is bypassed when phase switching is started, e.g. using IndraWorks. As soon as a switch is made back into the phase set at `ImcCtrl`.Admin._OpMode, the output is deleted again. Furthermore, the input `ImcCtrl`.Admin.RetriggerOpMode can be used when the output PassiveMode is set to switch back into the set target phase.

ML_TechInterface.library

**What is new in version 11VRS
(XLC11VRS/MLC11VRS)?**

- Support for the new axis type "controller axis": the axis type "controller axis", introduced in MLC10VRS, is detected as `arAxisStatus_gb[].Admin.AxisType = ControlAxisCfg (16#0080)` and supported as far as possible.
- User extensions that are easier to use with the object oriented language extensions of IndraLogic 2G:
Now the user can also extend the substructures of `arAxisCtrl_gb[]` and `arAxisStatus_gb[]` (see also [chapter 6.7 "AxisInterface User Extensions" on page 88](#))
- Monitoring of the state `Axis_Interrupted`: the `AxisInterface` monitors the Bit `AxisData[].Axis_Interrupted` and when `AxisStatus` is `TRUE`, it switches into "ModelInterrupted". In this way, the `AxisInterface` reports that the control has been taken over by the drive, e.g. by the safety technology in the drive. This monitoring can be deactivated by resetting the new input `arAxisCtrl_gb[].Admin.DriveInterruptReaction`. The monitoring is active by default.
The monitoring of `AxisInterrupted` is also available in previous versions (MLC10V08 or higher).
- Cyclical scanning of `arAxisCtrl_gb[].SyncMode.SyncDirection` (XLC/MLC11V04 or higher): When the "SyncMode.PhaseOffset" is adjusted, the set "SyncMode.SyncDirection" takes effect when there is interpolation in the drive. With machines with a material web, when the synchronization mode is activated, synchronization takes place in the positive direction. For subsequent adjustments of "SyncMode.PhaseOffset", it is to be adjusted to the shortest distance. This makes it possible to change "SyncMode.SyncDirection" when an operation mode is active.
- The `AxisInterface` now reads in the corresponding, respective drive or control parameters depending on the axis type and message type.
IndraDrive with interpolation in the drive:
 - Positive acknowledgements: drive parameters are read, e.g. A0012 Ready for operation
 - Error in the drive: drive parameters are read, e.g. F2026 No power
 - Error on the control: control parameters are read, e.g. F2229233 Invalid master axis
 IndraDrive with interpolation in the control.
 - Positive acknowledgements A-0-0020/A-0-0023 are read. e.g. "Synchronized Motion"
 - Error in the drive: drive parameters are read, e.g. F2026 No power
 - Error on the control: control parameters are read, e.g. F2229233 Invalid master axis
 SercosDrive and virtual axes:
 - The control parameters are read.

**What is new in version 12VRS
(XLC12VRS/MLC12VRS)?**

- New input `arAxisCtrl_gb[].Admin.PowerOn`: If this bit is set (default value is `TRUE`), the `MC_Power` is called internally in the `AxisInterface`. If the input is not set, the `MC_Power` must be served by the user program.
- Example Project `AxisInterface`: in addition to "GAT^{compact}", the Template wizard now offers a special `AxisInterface` example project.
- Composition of diagnostic number and diagnostic message: for 12VRS and higher, A-0-0020/A-0-0023 for IndraDrive drives are always composed from drive parameters S-0-0095/S-0-0390. The `AxisInterface` now

always reads control parameters A-0-0020/A-0-0023 by default. This behavior can be modified via `arAxisCtrl_gb[].Admin.DiagNbrRefreshTime`.

- Initialization of the IMC interface target mode: The target mode `ImcCtrl.Admin._OpMode` is now initialized depending on the parameterization instead of with zero:

IF the current mode (C-0-0452) >= startup target mode (C-0-0450) THEN
`ImcCtrl.Admin._OpMode := current mode (C-0-0452)`

OTHERWISE

`ImcCtrl.Admin._OpMode := startup target mode (C-0-0450)`

In this way, at least the startup target mode (C-0-0450) is achieved or the current mode is maintained.

This initialization is carried out in the library just once when the PLC program is started and can be overwritten at anytime thereafter.



Between the various AxisInterface versions "01VRS" (MLC 03VRS), "02VRS" (MLC 04VRS) and theTechInterface "09VRS" (MLC09VRS) and "10VRS" (MLC10VRS) and newer versions, modifications were introduced, some of which require adaptations in the user program. The required adaptations are described in [chapter 6.2 "Initial TechInterface Configuration" on page 53](#).

6.2 Initial TechInterface Configuration

Before the functionality of the TechInterface can be used it has to be initialized. The following steps activate the AxisInterface in the IndraLogic.



Many of the steps described in this section are already included in the program template "GAT or GAT^{compact}". These templates can be imported into the IndraLogic and used as a guide for new projects. When using the GAT or GAT^{compact} template, the user programs the AxisInterface handling completely, including possible extensions, such that it runs.

First steps in using the example project GAT^{compact}.

1. Create a new project and insert the corresponding control called "MyControl", for example. Enter the IP address of the control in the following. If real axes are connected to the control, insert them below the control.
2. Right click on the "Logic" icon below the control. In the menu that opens, select the "GAT wizard...".
3. In the wizard, select the "GAT compact" version and start the import with **Finish**. The GAT wizard can be called up at any time if the specified project structure is maintained in order to add or delete axes or to call up visualization examples. The GAT wizard already has two virtual axes inserted to get the user started. If real axes are present, add them using the GAT wizard.
4. To transfer the motion project into the control, click on "MyControl" and then start the transfer with "MyControl->Go online". Then use the "BB" icon to switch into operating mode.
5. Loading the PLC project can be started using **Debug ► Login to Application...**. Then start the PLC using the arrow icon.
6. Now the virtual axes should be able to be moved using the visualization examples included with delivery (System_Overview_All_Axis).

ML_TechInterface.library

First steps with the user's own program

1. Open the Library Manager in the project tree under "Logic->Application" and add the library **ML_TechInterface.compiled-library** to the current IndraLogic project.
2. Below the "Application" node, add two "global variable lists" and call them **"UserVarGlobal"** and **"UserConstGlobal"**, for example (in GAT compact). Then generate the following code:

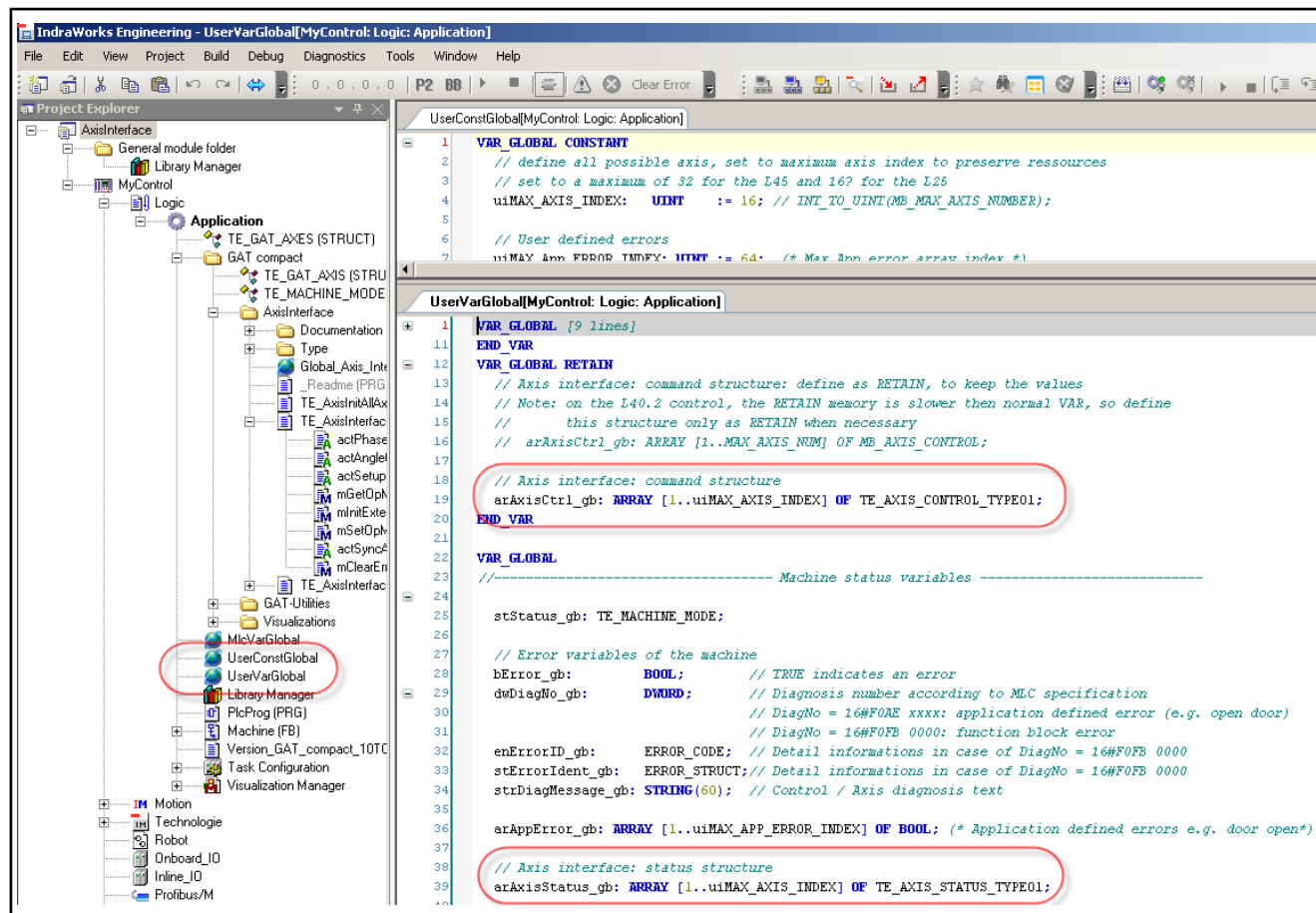


Fig.6-7: Global variables of the AxisInterface

3. Create a program (e.g. **TE_AxisInterfaceType01Prog** in GAT compact) that calls up the function block **MB_AxisInterfaceType01** for **MAX_AXIS_NUM** axes. This function block interprets the values in "arAxisCtrl_gb" and updates the values in "arAxisStatus_gb". It is also possible to call the function block in separate programs if it is required due to the project structure.
4. Click on the Task Configuration and specify the **TE_AxisInterfaceType01Prog** as a **PlcTask** to be called cyclically (e.g. Interval: T#20ms).
5. In the Task Configuration, call up the program "MB_ImcInterface" from the library **TECHINTERFACE** (e.g. Interval: T#20ms or in a free running task as well).
6. Create an initialization program that is called once at startup or at each phase switch from parameterization into operation mode and integrate this process into the previously created program (e.g. **TE_AxisInterfaceType01Prog** in GAT compact).
 - Use the **MB_AxisInitType01** FB for the basic initialization of **arAxisCtrl_gb** and **arAxisStatus_gb** for each axis used.

In GAT ^{compact}, depending on the parameterization, this is executed by the TE_AxisInitAllAxisType01 FB.

- The structure arAxisCtrl_gb[] can be defined as retain data. In this case, initialize all NOT RETAIN values with defined values (for defined values, see "STATE_FIRST_INIT" in TE_AxisInterface-Type01Prog).

If the structure is not defined as RETAIN, all the elements that differ from the default value must be initialized.

7. Use arAxisCtrl_gb[], arAxisStatus_gb[] and AxisData[] to program your application.

Converting AxisInterface projects



Before carrying out the following changes, create an IndraWorks project archive. If adaptations are made to example project program parts that were not intended to be modified, these may be lost.

The following adaptations to the AxisInterface are necessary to convert a project from the AxisInterface 01VRS (MLC 03VRS) to the AxisInterface 02VRS (MLC 04VRS).

1. Global variables:

Renaming the structures

- AXIS_CONTROL_TYPE01 in MB_AXIS_CONTROL_TYPE01 and
- AXIS_STATUS_TYPE01 in MB_AXIS_STATUS_TYPE01.

The constant MAX_AXIS_IF_CYCLES in the Global_Axis_Interface is no longer required.

2. Task configuration:

Change the call "ImcInterface()" to "MB_ImcInterface()".

3. The following structure elements were moved/renamed and must be adapted when they are used in the user program:

- | | |
|--|--|
| • arAxisStatus[i].Active | → arAxisStatus[i].Admin.Active |
| • arAxisCtrl[i].SyncMode.AdjCamShaftDist | → arAxisCtrl[i].SyncMode.AdjCamShaftDistance |
| • arAxisCtrl[i].Comment | → arAxisStatus[i].Admin.Name |
| • arAxisCtrl[i].SyncMode.InputRev | → arAxisCtrl[i].SyncMode.InputRevolution |
| • arAxisCtrl[i].SyncMode.OutputRev | → arAxisCtrl[i].SyncMode.OutputRevolution |
| • arAxisCtrl[i].SyncMode.CamShaftDist | → arAxisCtrl[i].SyncMode.CamShaftDistance |
| • arAxisCtrl[i].SyncMode.AdjCamShaftDist | → arAxisCtrl[i].SyncMode.AdjCamShaftDistance |

4. If the structures arAxisCtrl/arAxisStatus were extended, the modifications from MB_AXIS_CONTROL_TYPE01 or MB_AXIS_STATUS_TYPE01 must be incorporated into the user structures. All structures were renamed from AXIS_xxx to MB_AXIS_xxx.

The following adaptations to the AxisInterface are necessary to convert a project from the AxisInterface 02VRS (MLC 04VRS) to the TechInterface 09VRS (MLC 09VRS). It is assumed that the conversion of the IndraWorks project was already carried out.

ML_TechInterface.library

Due to the new BIT data type, access to _OpMode changes within the AxisCtrl structure. In the AxisStatus structure the accesses to _OpMode remain unchanged.

- Example for use as enumeration (enumeration data type):
 02VRS:
`arAxisCtrl[MyRealAxis1.Axisno].Admin._OpMode: = ModeAH`
 09VRS:
`arAxisCtrl[MyRealAxis1.Axisno].Admin._OpMode.en: = ModeAH`
- Example for using BIT:
 02VRS:
`arAxisCtrl[MyRealAxis1.Axisno].Admin.MODE_SYNC_PHASE: = TRUE;`
 09VRS:
`arAxisCtrl[MyRealAxis1.Axisno].Admin._OpMode.b.MODE_SYNC_PHASE: = TRUE;`
- 1. Procedure for use as enumeration (enumeration data type):
 - Search for "Admin._OpMode"
 - Replace with "Admin._OpMode.en" only in places where "arAxisCtrl[]" was found.
- 2. Procedure for use of BIT:
 - Search for "Admin.MODE_"
 - Replace with "Admin._OpMode.b.MODE_" only in places where "arAxisCtrl[]" was found.

No adaptations to the PLC project are required to convert a project from AxisInterface 09VRS (MLC 09VRS) to TechInterface 10VRS (MLC 10VRS). Only the IndraWorks project must be converted and the current TechInterface library must be integrated.

For MLC release 10V04 and higher, an AxisInterface example project will no longer be provided. Instead, the "GAT^{compact}" example project can be used as an introductory template. In the context of the conversion to "GAT^{compact}" many different parts of the example project were renamed:

- | | |
|--|--|
| • arAxisStatus[i] | → arAxisStatus_gb[i] |
| • arAxisCtrl[i] | → arAxisCtrl_gb[i] |
| • Global_Axis_Interface | → UserVarGlobal |
| • Global_Axis_Interface_Const | → UserConstGlobal |
| • MY_AxisInterface | → TE_AxisInterface |
| • ML_AxisInitAllAxisType01 | → TE_AxisInitAllAxisType01 |
| • AxisInterfaceType01Prog and InitProg | → combined into TE_AxisInterfaceType01Prog |
| • Structures MY_ | → TE_... |

6.3 AxisInterface Function Blocks

6.3.1 MB_AxisInitType01

Brief description The MB_AxisInitType01 FB is use to initialize the AxisInterface for a single axis. The function block must be called only once when the program is started or each time the phase is switched from the parameterization mode to the operating mode.

Assignment: Target system/library

Target system	Library
IndraMotion MLC 09VRS or higher	ML_TechInterface.compiled-library
IndraMotion MLD, MPx07 or higher, with additional package MA	MX_AxisInterface.lib

Fig.6-8: Reference table MB_AxisInitType01

Interface description

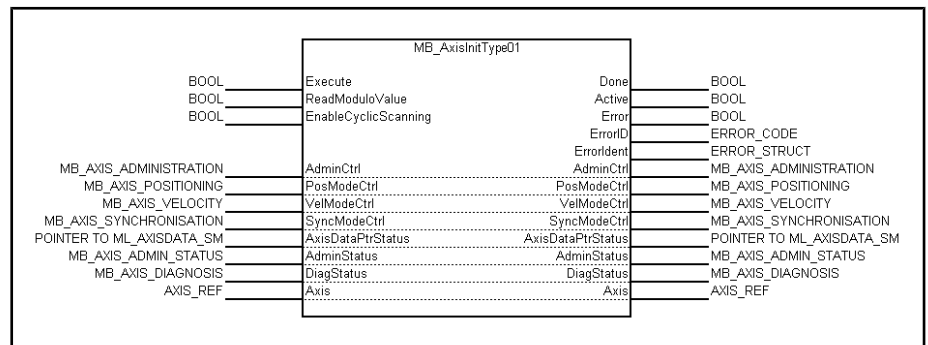


Fig.6-9: MB_AxisInitType01 function block

I/O type	Name	Data type	Comment
VAR_INPUT	Execute	BOOL	The initialization is started via a rising edge on "Execute".
	ReadModuloValue	BOOL	Depending on the parameter settings, the modulo value for rotary axes is written to the Adjustment structure of PosModeCtrl and SyncModeCtrl.AdjPhaseOffset. The modulo value is set to zero for all other set values. The default value is TRUE.
	EnableCyclicScanning	BOOL	Activates the cyclic testing and transfer of the modified inputs for the current operation mode - relevant for the values in arAxisCtrl_gb[] with the comment: cyclically tested AND transferred in case of change. The default value is TRUE.
VAR_OUTPUT	Done	BOOL	Set when the function block has completed processing.
	Active	BOOL	Set when the function block is active (not in idle mode).
	Error	BOOL	It indicates that an error has occurred in the function block instance.
	ErrorID	ERROR_CODE	Short note on the error cause
	ErrorIdent	ERROR_STRUCT	Detailed information on the error

ML_TechInterface.library

I/O type	Name	Data type	Comment
VAR_IN_OUT	Axis	AXIS_REF	Axis to be initialized in the AxisInterface
	AdminCtrl	MB_AXIS_ADMINISTRATION	Axis management Connect arAxisCtrl_gb[i].Admin
	PosModeCtrl	MB_AXIS_POSITIONING	Positioning operation modes Connect arAxisCtrl_gb[i].PosMode
	VelModeCtrl	MB_AXIS_VELOCITY	Velocity control operation mode Connect arAxisCtrl_gb[i].VelMode
	SyncModeCtrl	MB_AXIS_SYNCHRONISATION	Synchronization operation modes Connect arAxisCtrl_gb[i].SyncMode
	AxisDataPtrStatus	POINTER TO ML_AXISDATA_SM	Pointer to AxisData of this axis Connect arAxisStatus_gb[i].PtrAxisData
	AdminStatus	MB_AXIS_ADMIN_STATUS	Status management of the axis Connect arAxisStatus_gb[i].Admin
	DiagStatus	MB_AXIS_DIAGNOSIS	Diagnostic information on the axis Connect arAxisStatus_gb[i].Diag

Fig.6-10: I/O interface, MB_AxisInitType01



It is not possible to transfer the entire instance of the MB_Axis_Control_Type01 and MB_Axis_Status_Type01 structures (e.g. arAxisCtrl_gb[]/arAxisStatus_gb[]) to the function block using one input. This was done to allow special supplements to the existing code for the user application. More details under [chapter 6.7 "AxisInterface User Extensions"](#) on page 88.

Therefore, the required elements of the MB_Axis_Control_Type01 and MB_Axis_Status_Type01 are transferred as separate inputs.

The MB_AxisInitType01 FB initializes the following structure elements with default values:

Structure element	Default
AdminCtrl._OpMode	ModeAB
AdminCtrl.ClearError	FALSE
AdminCtrl.SetAbsRef	FALSE
AdminCtrl.EnableCyclicScanning	EnableCyclicScanning input
AxisDataPtrStatus	ADR(AxisData[Axis.AxisNo])
AdminCtrl.Axis.CntrlNo	LOCAL_CNTRL
AdminCtrl.Axis.AxisNo	Axis.AxisNo input
PosModeCtrl.Adjust.Enable	FALSE
VelModeCtrl.Adjust.Enable	FALSE
SyncModeCtrl.AdjCamShaftDistance.Enable	FALSE
SyncModeCtrl.AdjFineadjust.Enable	FALSE
SyncModeCtrl.AdjPhaseOffset.Enable	FALSE

ML_TechInterface.library

Structure element	Default
DiagStatus.Number	0
DiagStatus.Message	reading not activated
AdminStatus.Active	TRUE for enabled drives
AdminStatus.AxisType	Enumeration type AXIS_CONFIG
AdminStatus.Name	Axis name depending on the parameter settings

Fig. 6-11: Structure elements initialized by the function block

The structure elements listed above are only initialized if the **ReadModulo = TRUE** input and the following state exist:

- VelModeCtrl.Adjust.ModuloValue = 0
- SyncModeCtrl.AdjCamShaftDistance.ModuloValue = 0
- SyncModeCtrl.AdjFineadjust.ModuloValue = 0
- If the axis scaling is set to absolute format:
 - PosModeCtrl.Adjust.ModuloValue = 0
 - SyncModeCtrl.AdjPhaseOffset.ModuloValue = 0
- If the axis scaling is set to modulo format:
 - PosModeCtrl.Adjust.ModuloValue = Modulo value
 - SyncModeCtrl.AdjPhaseOffset.ModuloValue = Modulo value

Error handling The function block does not generate any special error codes. Only error codes of the internally used FBs to read the parameters are transferred.

6.3.2 MB_AxisInterfaceType01

Brief description The MB_AxisInterfaceType01 FB is used to configure the AxisInterface for a single axis. This function block must be called cyclically as long as the user is in the operating mode.

Assignment: Target system/library

Target system	Library
IndraMotion MLC 09VRS or higher	ML_TechInterface.compiled-library
IndraMotion MLD, MPx07 or higher, with additional package MA	MX_AxisInterface.lib

Fig. 6-12: Reference table, AxisInterfaceType01

ML_TechInterface.library

Interface description

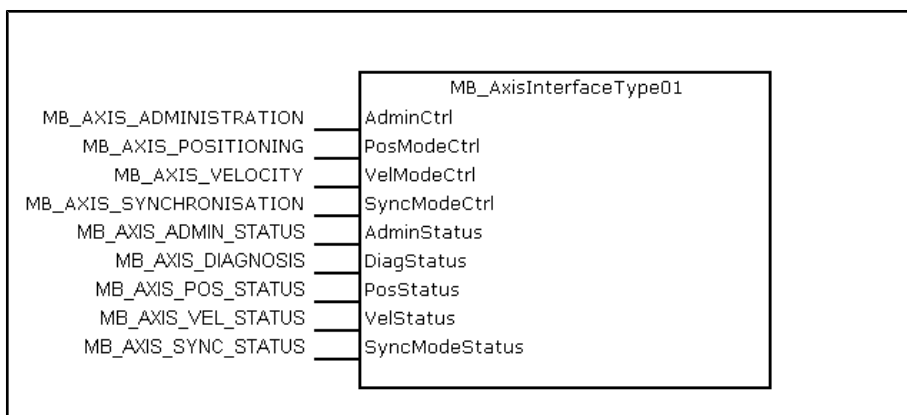


Fig. 6-13: MB_AxisInterfaceType01 function block

I/O type	Name	Data type	Comment
VAR_INPUT	AdminCtrl	REFERENCE TO MB_AXIS_ADMINISTRATION	Axis management, connect arAxisCtrl_gb[i].Admin
	PosModeCtrl	REFERENCE TO MB_AXIS_POSITIONING	Positioning operation modes, connect arAxisCtrl_gb[i].PosMode
	VelModeCtrl	REFERENCE TO MB_AXIS_VELOCITY	Velocity control operation mode, connect arAxisCtrl_gb[i].VelMode
	SyncModeCtrl	REFERENCE TO MB_AXIS_SYNCHRONISATION	Synchronization operation modes, connect arAxisCtrl_gb[i].SyncMode
	AdminStatus	REFERENCE TO MB_AXIS_ADMIN_STATUS	Status management of the axis, connect arAxisStatus_gb[i].Admin
	DiagStatus	REFERENCE TO MB_AXIS_DIAGNOSIS	Status diagnostics of the axis, connect arAxisStatus_gb[i].Diag
	PosStatus	REFERENCE TO MB_AXIS_POS_STATUS	Status of positioning operation modes, connect arAxisStatus_gb[i].PosMode
	VelStatus	REFERENCE TO MB_AXIS_VEL_STATUS	Status of velocity operation mode, connect arAxisStatus_gb[i].VelMode
	SyncModeStatus	REFERENCE TO MB_AXIS_SYNC_STATUS	Status of synchronization operation mode, connect arAxisStatus_gb[i].SyncMode

Fig. 6-14: Interface variables, MB_AxisInterfaceType01



It is not possible to transfer the entire instance of the MB_Axis_Control_Type01 and MB_Axis_Status_Type01 structures (e.g. arAxisCtrl_gb[]/arAxisStatus_gb[]) to the function block using one input. This was done to allow special supplements to the existing code for the user application. More details under [chapter 6.7 "AxisInterface User Extensions"](#) on page 88.

Therefore, the required elements of the MB_Axis_Control_Type01 and MB_Axis_Status_Type01 are transferred as separate inputs.

To optimize performance, the structures are connected with REFERENCE TO instead of as VAR_IN_OUT as VAR_INPUT. In this way, the inputs only have to be initialized once. In this way, if the function block is called cyclically, the structures do not have to be transferred.

The function block checks the inputs of arAxisCtrl_gb[] and internally generates the requested commands for the axis. The outputs of arAxisStatus_gb[] are updated depending on the result of this command. For example, setting arAxisCtrl_gb[].Admin._OpMode from ModeAb to ModePosAbs leads to the following sequence:

- Check the required states to activate a motion command such as "Axis in Ab"
- Activate the PLCopen function block MC_Power (when arAxisCtrl_gb.Admin.PowerOn = TRUE)
- Wait for the acknowledgement that the power of the axis has been added (AH/AF)
- Activate PLCopen function block MC_MoveAbsolute with the command values of PosModeCtrl
- Acknowledge arAxisStatus_gb[].Admin._OpModeAck to ModePosAbs (Bit MODE_POS_ABS)
- Scan the values of PosModeCtrl.Position and PosModeCtrl.Velocity and reactivate MC_MoveAbsolute in the case of modifications

Error handling The function block generates the following error messages in DiagStatus.ErrorIdent.Additional1/Additional2 for the table "F_RELATED_TABLE", 16#0170:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR, (16#0005)	16#0006	16#0000	Invalid function block state
DEVICE_ERROR, (16#0008)	16#1500	16#0000	Drive reports an error in AxisData[]
INPUT_INVALID_ERROR, (16#0001)	16#1501	16#0000	Invalid operation mode selected
RESOURCE_ERROR, (16#0003)	16#1502	16#0000	Drive is not in AB
INPUT_INVALID_ERROR, (16#0001)	16#1503	16#0000	Invalid AdminCtrl.Axis.AxisNo
RESOURCE_ERROR, (16#0003)	16#150F	16#0000	AdminCtrl.PowerOn = FALSE AND drive is not in AH/AF

Fig. 6-15: Error codes, MB_AxisInterfaceType01

6.4 Operation Modes

6.4.1 Overview

There are two methods to activate an operation mode:

ML_TechInterface.library

1. Selection of Enum values

Assigning a value of TYPE MB_AXIS_MODE to

arAxisCtrl_gb[].Admin._OpMode.en:

```
arAxisCtrl_gb[ ].Admin._OpMode.en := ModePosAbs;
```

- or -

```
arAxisCtrl_gb[ ].Admin._OpMode.en := ModeAB;
```

2. Using the bit access

First delete the "_OpMode" in the programming...

```
arAxisCtrl_gb[ ].Admin._OpMode.en := ModeAB;
```

Then set a bit using the bit access functionality.

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_POS_ABS := TRUE;
```



Setting several bits in arAxisCtrl_gb[].Admin._OpMode causes an error and the drive switches to AB.

Before an operation mode can be activated, values must be assigned to every attribute. All attributes are provided with default values. Some are provided with values not equal to zero while others are defined as 0. Due to the special demands, a value has to be assigned to them.



Only the attributes (e.g. position, velocity) that are used or that have default values that were changed must be declared before the current operation mode switch is executed.

The following example shows the correct programming sequence when changing the operation mode:

```

1 // The following code initializes position mode attributes
2 // to specific values
3
4 arAxisCtrl[MyRealAxis1.AxisNo].PosMode.Position := 250.0;
5 arAxisCtrl[MyRealAxis1.AxisNo].PosMode.Velocity := 500.0;
6 arAxisCtrl[MyRealAxis1.AxisNo].PosMode.Acceleration := 200.0;
7 arAxisCtrl[MyRealAxis1.AxisNo].PosMode.

```

maximum velocity (always positive) (not necessarily reached) [u/s]
- start moving with "ModePosAbs" or "ModePosRel" or "ModePosAdd"
(MC_MoveAbsolute, MC_MoveRelative, MC_MoveAdditive)
Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)

Positioniergeschwindigkeit, (Einheit: [u/s]) (positiv eingeben)
- Freigabe mit "ModePosAbs" oder "ModePosRel" oder "ModePosAdd"
(MC_MoveAbsolute, MC_MoveRelative, MC_MoveAdditive)
Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE)

```

15 // The following code commands the axis to switch to
16 // Absolute positioning mode
17 arAxisCtrl[MyRealAxis1.AxisNo].Admin._OpMode.en := ModePosAbs;
18
19 // The following code checks the status of the
20 // Absolute positioning mode
21 IF arAxisStatus[MyRealAxis1.AxisNo].Admin.MODE_POS_ABS THEN
22 // Customer specific code to handle status acknowledgement
23 // e.g. set a bit or output
24 ;
25 END_IF

```

Acceleration
Adjust
Deceleration
Distance
Position
Velocity

Fig.6-16: Example of an absolute positioning mode changeover



The status acknowledgement (`arAxisStatus_gb[] .Admin.`) for an operation mode is implemented as follows.

Example:

1. The status acknowledgement
`arAxisStatus_gb[] .Admin.MODE_POS_ABS`
Returns TRUE only if the command for switching the axis to the absolute positioning mode was executed.
2. The status acknowledgement
`arAxisStatus_gb[] .Admin.CmdDone`
is supported only for the Drive Halt operation mode, for homing the drive and for positioning operation modes.
If a positioning operation mode such as
`arAxisCtrl_gb[] .Admin._OpMode.b.MODE_POS_ABS`
is activated, the
`arAxisStatus_gb[] .Admin.MODE_POS_ABS`
output is set to TRUE if the drive switches to the positioning operation mode and starts to turn.
When the drive reached its target position,
`arAxisStatus_gb[] .Admin.CmdDone`
is set.

The input `arAxisCtrl_gb.Admin.PowerOn` controls the internal use of the `MC_Power` in the `AxisInterface`:

- `PowerOn = TRUE` (Default): the `AxisInterface` takes over the actuation of `MC_Power`. When an operation mode is activated, first `MC_Power` is called internally and only after a ready signal is given is the operation mode that is actually selected activated. In case of a switch off (`ModeAb`), `MC_Power` is removed after the drive has been halted.
- `PowerOn = FALSE`: the application program must be operated by `MC_Power`. If an operation mode is selected and `MC_Power` was not previously actuated, the `AxisInterface` outputs an error.

The following section describes the operation modes supported by the `AxisInterface Type01` and the attributes that can be assigned to them. In the following, it is assumed that the default value of the `PowerOn` input is TRUE.

6.4.2 Drive Ready

Activating this operation mode switches the drive to AB (Drive ready) and switches the torque off. The following command activates the operation mode:
`arAxisCtrl_gb[] .Admin._OpMode.en: = ModeAB;`



The `AxisInterface` uses the `MC_Power` and `MC_Stop` PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

ML_TechInterface.library

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl[]	Admin.StopDeceleration	REAL	100.0	Yes
	Admin.Axis	AXIS_REF		No

Fig. 6-17: "Drive ready" attributes

6.4.3 Drive Halt

Activating this operation mode switches the drive to AH (Drive Halt) and maintains the torque. The operation mode "ModeAH" now transfers the axis into the PLCopen state "StandStill", i.e. a PLCopen function block that is called outside of the AxisInterface is accepted and the AxisInterface reports "ModeExternalFB"

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode: = ModeAH;
```

or

```
arAxisCtrl_gb[ ].Admin.MODE_AH: = TRUE;
```



The AxisInterface uses the MC_Power and MC_Stop PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	Admin.StopDeceleration	REAL	100.0	Yes
	Admin.Axis	AXIS_REF		No
arAxisStatus_gb[]	Admin.MODE_AH	BOOL		Not applicable
	Admin.CmdDone	BOOL		Not applicable

Fig. 6-18: "Drive Halt" attributes

6.4.4 Drive Stop

Activating this operation mode switches the drive to AH (Drive Halt) and maintains the torque. The operation mode ModeStopping stops the PLCopen state machine in the "Stopping" state, i.e. a PLCopen function block called outside of the AxisInterface would be ended with the error "Command in state Stopping denied".

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode: = ModeStopping;
```

or

```
arAxisCtrl_gb[ ].Admin.MODE_STOPPING: = TRUE;
```



The AxisInterface uses the MC_Power and MC_Stop PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	Admin.StopDeceleration	REAL	100.0	Yes
	Admin.Axis	AXIS_REF		No
arAxisStatus_gb[]	Admin.MODE_STOPPING	BOOL		Not applicable
	Admin.CmdDone	BOOL		Not applicable

Fig. 6-19: Drive Stop attributes

6.4.5 Drive Homing

Activating this operation mode executes the command to home the drive. It controls axes with incremental encoders during the homing procedure.

The following command activates the operation mode:

```
arAxisCtrl[ ].Admin._OpMode.en: = ModeHoming;
```

or

```
arAxisCtrl[ ].Admin._OpMode.b.MODE_HOMING: = TRUE;
```

The following acknowledgement must follow the operation mode command. It returns TRUE when the drive reached its homing position.

```
arAxisStatus[ ].Admin.CmdDone
```



The AxisInterface uses the MC_Power and MC_Home PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl[]	Admin.Axis	AXIS_REF		No
arAxisStatus[]	Admin.MODE_HOMING	BOOL		Not applicable
	Admin.CmdDone	BOOL		Not applicable

Fig. 6-20: "Drive Homing" attributes

6.4.6 Absolute Positioning

Activating this operation mode causes an absolute motion to a previously specified target position.

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode.en: = ModePosAbs;
```

or

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_POS_ABS: = TRUE;
```

The following acknowledgement must follow the operation mode command. It returns TRUE, when the drive reached its target position.

```
arAxisStatus_gb[ ].Admin.CmdDone
```

ML_TechInterface.library



The AxisInterface uses the MC_Power and MC_MoveAbsolute PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	PosMode.Position	REAL	0.0	Yes
	PosMode.Velocity	REAL	10.0	Yes
	PosMode.Acceleration	REAL	10.0	No
	PosMode.Deceleration	REAL	10.0	No
	Admin.Axis	AXIS_REF		No
arAxisStatus_gb[]	Admin.MODE_POS_ABS	BOOL		Not applicable
	Admin.CmdDone	BOOL		Not applicable

Fig.6-21: "Absolute Positioning" attributes

6.4.7 Relative Positioning

Activating this operation mode causes a relative motion to a previously specified target position by adding PosMode.Distance to the current actual position.

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode.en := ModePosRel;
```

or

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_POS_REL := TRUE;
```

The following acknowledgement must follow the operation mode command. It returns TRUE, when the drive reached its target position.

```
arAxisStatus_gb[ ].Admin.CmdDone
```



The AxisInterface uses the MC_Power and MC_MoveRelative PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	PosMode.Distance	REAL	0.0	Yes
	PosMode.Velocity	REAL	10.0	Yes
	PosMode.Acceleration	REAL	10.0	No
	PosMode.Deceleration	REAL	10.0	No
	Admin.Axis	AXIS_REF		No
arAxisStatus_gb[]	Admin.MODE_POS_REL	BOOL		Not applicable
	Admin.CmdDone	BOOL		Not applicable

Fig.6-22: "Relative Positioning" attributes

6.4.8 Additive Positioning

Activating this operation mode leads to a relative motion to a previously specified target position by adding PosMode.Distance to the current target position.

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode.en: = ModePosAdd;
```

or

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_POS_ADD: = TRUE;
```

The following acknowledgement must follow the operation mode command. It returns TRUE, when the drive reached its target position.

```
arAxisStatus_gb[ ].Admin.CmdDone
```



The AxisInterface uses the MC_Power and MC_MoveAdditive PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	PosMode.Distance	REAL	0.0	Yes
	PosMode.Velocity	REAL	10.0	Yes
	PosMode.Acceleration	REAL	10.0	No
	PosMode.Deceleration	REAL	10.0	No
	Admin.Axis	AXIS_REF		No
arAxisStatus_gb[]	Admin.MODE_POS_ADD	BOOL		Not applicable
	Admin.CmdDone	BOOL		Not applicable

Fig. 6-23: "Additive Positioning" attributes

6.4.9 Velocity Control

Activating this operation mode executes motions at a previously specified velocity until a halt command is issued.

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode.en: = ModeVel;
```

or

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_VEL: = TRUE;
```



The AxisInterface uses the MC_Power and MC_MoveVelocity PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

ML_TechInterface.library

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	VelMode.Velocity	REAL	10.0	Yes
	VelMode.Acceleration	REAL	10.0	Yes
	VelMode.Deceleration	REAL	10.0	Yes
	Admin.Axis	AXIS_REF		No
arAxisStatus_gb[]	Admin.MODE_VEL	BOOL		Not applicable

Fig.6-24: "Velocity Control" attributes

6.4.10 Operation Mode "Phase Synchronization"

When this operation mode is activated, the drive is operated using an electronic gear with fine adjustment. This functionality causes a phase synchronization between the master axis and the selected slave axis.

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode.en: = ModeSyncPhase;
```

or

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_SYNC_PHASE: = TRUE;
```



The AxisInterface uses the MC_Power, MB_GearInPos, MC_Gear-Out and MB_PhasingSlave PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	SyncMode.OutputRevolution	UINT	1	Yes
	SyncMode.InputRevolution	UINT	1	Yes
	SyncMode.Fineadjust	REAL	0.0	Yes
	SyncMode.SyncDirection	MC_SYNC_DIRECTION	SHORTEST_WAY	Yes
	SyncMode.StartMode	MC_START_MODE	ABSOLUTE	No
	SyncMode.Master	AXIS_REF		Yes
	Admin.Axis	AXIS_REF		No
	SyncMode.PhaseOffset	REAL	0.0	Yes
	SyncMode.PhaseOffsetVel	REAL	1.0	No
	SyncMode.PhaseOffsetAcc	REAL	100.0	No
arAxisStatus_gb[]	Admin.MODE_SYNC_PHASE	BOOL		Not applicable
	SyncMode.PhasingSlaveDone	BOOL		Not applicable

Fig.6-25: Attributes, operation mode "phase synchronization"

6.4.11 Operation Mode "Velocity Synchronization"

When this operation mode is activated, the drive is operated using an electronic gear with fine adjustment. This functionality causes a velocity synchronization between the master axis and the selected slave axis.

The following command activates the operation mode:

```
arAxisCtrl_gb[].Admin._OpMode.en: = ModeSyncVel;
```

or

```
arAxisCtrl_gb[].Admin._OpMode.b.MODE_SYNC_VEL: = TRUE;
```



The AxisInterface uses the MC_Power, MC_GearIn and MC_Gear-Out PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	SyncMode.OutputRevolution	UINT	1	Yes
	SyncMode.InputRevolution	UINT	1	Yes
	SyncMode.Fineadjust	REAL	0.0	Yes
	SyncMode.Master	AXIS_REF		Yes
	Admin.Axis	AXIS_REF		No
arAxisStatus_gb[]	Admin.MODE_SYNC_VEL	BOOL		Not applicable

Fig. 6-26: Attributes, operation mode "velocity synchronization"

6.4.12 Operation Mode "Electronic Cam"

Activating this operation mode causes the drive to follow the master using a cam profile that is stored in the drive. This operation mode is only supported by IndraDrive drives.

The following command activates the operation mode:

```
arAxisCtrl_gb[].Admin._OpMode.en := ModeSyncCam;
```

or

```
arAxisCtrl_gb[].Admin._OpMode.b.MODE_SYNC_CAM := TRUE;
```



The AxisInterface uses the MC_Power, MC_CamIn, MC_CamOut and MB_PhasingSlave PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

ML_TechInterface.library

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	SyncMode.OutputRevolution	UINT	1	Yes
	SyncMode.InputRevolution	UINT	1	Yes
	SyncMode.Fineadjust	REAL	0.0	Yes
	SyncMode.CamShaftDistance	REAL	360.0	Yes
	SyncMode.SyncDirection	MC_SYNC_DIRECTION	SHORTEST_WAY	Yes
	SyncMode.StartMode	MC_START_MODE	ABSOLUTE	No
	SyncMode.CamTableID	MC_CAM_ID	CAM_TABLE_1	Yes
	SyncMode.Master	AXIS_REF		Yes
	Admin.Axis	AXIS_REF		No
	SyncMode.PhaseOffset	REAL	0.0	Yes
	SyncMode.PhaseOffsetVel	REAL	1.0	No
	SyncMode.PhaseOffsetAcc	REAL	100.0	No
arAxisStatus_gb[]	SyncMode.PhasingSlaveDone	BOOL		
	SyncMode.CamActiveID	MC_CAM_ID		
	Admin.MODE_SYNC_CAM	BOOL		Not applicable

Fig.6-27: Attributes, operation mode "electronic cam"

6.4.13 Operation Mode "Electronic MotionProfile"

Activating this operation mode switches the electronic cam function on in the drive. The cam profiles are logically defined using the parameters for the electronic MotionProfile. A synchronous position command value for the slave axis is determined by the position of the master axis. This operation mode can only be used for virtual axes or with generic drives (profile is calculated in the control).

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode.en := ModeSyncProfile;
```

or

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_SYNC_PROFILE := TRUE;
```



The AxisInterface uses the MC_Power, MB_MotionProfile, MC_GearOut and MB_PhasingSlave PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	SyncMode.OutputRevolution	UINT	1	Yes
	SyncMode.InputRevolution	UINT	1	Yes
	SyncMode.Fineadjust	REAL	0.0	Yes
	SyncMode.Profile.SetSelection	UINT	0	Yes
	SyncMode.Profile.RelativePositioning	BOOL	FALSE	No
	SyncMode.SyncDirection	MC_SYNC_DIRECTION	SHORTEST_WAY	Yes
	SyncMode.StartMode	MC_START_MODE	ABSOLUTE	No
	SyncMode.Master	AXIS_REF		Yes
	Admin.Axis	AXIS_REF		No
	SyncMode.PhaseOffset	REAL	0.0	Yes
	SyncMode.PhaseOffsetVel	REAL	1.0	No
	SyncMode.PhaseOffsetAcc	REAL	100.0	No
arAxisStatus_gb[]	SyncMode.PhasingSlaveDone	BOOL		Not applicable
	SyncMode.ProfileActiveSet	UINT		Not applicable
	Admin.MODE_SYNC_CAM	BOOL		Not applicable

Fig. 6-28: Attributes, "electronic MotionProfile"

6.4.14 Operation Mode "Electronic FlexProfile"

Activating this operation mode switches the electronic cam function on in the drive. The cam profiles are logically defined using the parameters for the electronic FlexProfile. This operation mode can only be used for virtual axes or with generic drives (profile is calculated in the control). This operation mode is not available on the drive-based IndraMotion MLD system.

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode.en: = ModeFlexProfile;
```

or

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_FLEX_PROFILE: = TRUE;
```



The AxisInterface uses the MC_Power, ML_FlexProfile, MC_Gear-Out and MB_PhasingSlave PLCopen FBs internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

ML_TechInterface.library

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	SyncMode.OutputRevolution	UINT	1	Yes
	SyncMode.InputRevolution	UINT	1	Yes
	SyncMode.Fineadjust	REAL	0.0	Yes
	SyncMode.Profile.SetSelection	UINT	0	Yes
	SyncMode.Profile.UseSwitchingPosition	BOOL	FALSE	No
	SyncMode.Profile.SwitchingPosition	REAL	0.0	No
	SyncMode.Profile.ProfileEntry	MB_PRO- FILE_START_POINT	SLAVE_ORI- GIN_MAS- TER_ORIGIN	No
	SyncMode.Profile.MasterOffset	REAL	0.0	No
	SyncMode.Profile.SlaveOffset	REAL	0.0	No
	SyncMode.Profile.SyncType	MB_SYNC_TYPE	SYNC_RAM- PIN_SHORT- EST_WAY	No
	SyncMode.Profile.SyncVelocity	REAL	10.0	No
	SyncMode.Profile.SyncAcceleration	REAL	10.0	No
	SyncMode.Profile.ExecutionMode	MB_EXECUTION_MODE	EXECUTE_CY- CLIC	No
	SyncMode.Master	AXIS_REF		Yes
	Admin.Axis	AXIS_REF		No
	SyncMode.PhaseOffset	REAL	0.0	Yes
	SyncMode.PhaseOffsetVel	REAL	1.0	No
	SyncMode.PhaseOffsetAcc	REAL	100.0	No
arAxisStatus_gb[]	SyncMode.PhasingSlaveDone	BOOL		Not appli- cable
	SyncMode.ProfileActiveSet	UINT		Not appli- cable
	Admin.MODE_FLEX_PROFILE	BOOL		Not appli- cable

Fig.6-29: "Electronic FlexProfile" attributes

6.4.15 Operation Mode "Drive-controlled Interruption"

This operation mode is used to display a drive-controlled interruption.

The AxisInterface monitors the state of the drive with the bit AxisData[].Axis_Interrupted and when AxisStatus is TRUE, it switches into "ModeInterrupted". In this way, the AxisInterface reports that the control has been taken over by the drive, e.g. by the safety technology in the drive.

This monitoring can be deactivated by resetting the input arAxisCtrl_gb[].Admin.DriveInterruptReaction. The monitoring is active by default.

This operation mode cannot be specified. If "ModeInterrupted" is specified anyway, as in "ModeExternalFb" the drive will be switched to AH (MC_Stop is NOT activated).

The following table contains the attributes supported by this operation mode:

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	Admin.DriveInterruptReaction	BOOL	TRUE	Yes
arAxisStatus_gb[]	Admin.MODE_INTERRUPTED	BOOL		Not applicable

Fig. 6-30: Attributes, operation mode "drive-controlled interruption"

Applications for monitoring Axis_Interrupted and the input DriveInterruptReaction:

- Approaching with a drive with a brake and interpolation in the control: If the value in S-0-0206 "Drive on delay time" is large, the interpolator in the control starts specifying command values without the drive following. This can cause a command value jump at the end of the wait time. Monitoring during approaching is always activated, regardless of the input DriveInterruptReaction.
- Safety technology or drive-internal command (e.g. brake test): DriveInterruptReaction = TRUE. When Axis_Interrupted occurs in an operation mode, in arAxisStatus_gb[].Admin.MODE_INTERRUPTED is reported until a change in the selected operation mode is made. At a rising edge of Axis_Interrupted, velocity control is specified as zero velocity. If new operation mode is specified before the interruption ends, right after Axis_Interrupted ends the sequence continues with the new operation mode.
- Using the drive functionality applications (e.g. rapid stop, touch probe): monitoring by the AxisInterface is not desired here and is switched off with DriveInterruptReaction = FALSE.
- Temporary switch off: monitoring can also be switched off for specific periods of time. Example: when homing with a fixed stop, motion away from the fixed stop can only be made with DriveInterruptReaction = FALSE.

6.4.16 Operation Mode "External Function Block"

Activating this operation mode switches the drive to AH (MC_Stop is NOT active); then it waits for an external motion command, such as one from a technology function.

The following command activates the operation mode:

```
arAxisCtrl_gb[ ].Admin._OpMode.en: = ModeExternalFB;
```

or

```
arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_EXTERNAL_FB: = TRUE;
```



The AxisInterface uses the MC_Power and PLCopen function block internally to carry out the switchover.

The following table contains the attributes supported by this operation mode:

ML_TechInterface.library

Element	Name	Type	Default	Cyclically scanned
arAxisCtrl_gb[]	Admin.Axis	AXIS_REF		No
arAxisStatus_gb[]	Admin.MODE_EXTERNAL_FB	BOOL		Not applicable

Fig.6-31: Attributes, operation mode "external function block"

6.5 Global Variables

The library ML_TechInterface (with IndraMotion MLD MX_AxisInterface.lib and MX_ImcInterface.lib) contains two global variable lists:

1. Global_MB_AxisInterfaceVars.

This list defines constants for the bit access on the OpMode variables

- (arAxisCtrl_gb[].Admin._OpMode and
- arAxisStatus_gb[].Admin._OpModeAck)

2. Global_ImcInterface.

This list defines the control and status structures of the IMC interface.

- Control structure: ImcCtrl
- Status structure: ImcStatus

```

Global_MB_AxisInterfaceVars
1  VAR_GLOBAL CONSTANT
2      (* define masks for the sync operating modes *)
3      (* e.g. IF( (INT_TO_WORD(AdminStatus._OpMode.enAck) AND MASK_MODE_WITH_FINE_ADJ) <> 0 ) THEN *)
4      MASK_MODE_FINE_ADJUST      :WORD := 16#0BC0; (* SYNC_PHASE, SYNC_VEL, SYNC_CAM,
5                                                    SYNC_PROFILE, FLEX_PROFILE *)
6      MASK_MODE_PHASE_OFFSET     :WORD := 16#0B40; (* SYNC_PHASE, SYNC_CAM, SYNC_PROFILE,
7                                                    FLEX_PROFILE *)
8      MASK_MODE_CAM_SHAFT_DIST   :WORD := 16#0100; (* SYNC_CAM *)
9
10     (* define masks for the axis config types *)
11     (* e.g. IF( (INT_TO_WORD(AdminStatus.AxisType) AND MASK_AXIS_COMMAND_POSSIBLE) <> 0 ) THEN *)
12     MASK_AXIS_COMMAND_POSSIBLE :WORD := 16#0098; (* RealAxisCfg, VirtualAxisCfg, ControlAxisCfg *)
13

Global_ImcInterface
1  VAR_GLOBAL
2
3      ImcCtrl:  MB_IMC_CONTROL_TYPE01; (* Instance in order to command the control *)
4      ImcStatus: MB_IMC_STATUS_TYPE01; (* Instance for getting control status information *)
5
6  END_VAR
7
  
```

Fig.6-32: Global variables of TechInterface

6.6 First Steps, AxisInterface

6.6.1 Overview

The programming template "AxisInterface example project" is used as an example application for the AxisInterface. This example project can be generated at any time using the GAT wizard.

1. Create a new IndraWorks project.
2. Add the control IndraMotion MLC.
3. Use the icon at upper left in the menu bar to start the GAT wizard.
4. Select AxisInterface example project and click the **Finish** button to start importing the project.

6.6.2 Program for the AxisInterface

The program **TE_AxisInterfaceType01Prog** of the example project covers the following points:

- **Initializing the AxisInterface:**

When parameterization mode (Phase 2) is reached, the function block **TE_AxisInitAllAxisType01** is used to initialize the AxisInterface for all configured axes. When operation mode (Phase 4) is reached, initialization is executed again in order to detect changes in the parameterization.

- **Cyclic call of the AxisInterface:**

After initialization is complete, the AxisInterface is called cyclically. To do this, the "TE_AxisInterface" function block is called. It is derived from **MB_AxisInterfaceType01** in order to make user extensions possible.

- **User extensions for the AxisInterface:**

The "TE_AxisInterface" function block contains sample programmed user extensions. Users can add further elements to AxisInterface structures in order to cover their respective applications as well as possible.



The program **TE_AxisInterfaceType01Prog** is helpful for integrating the AxisInterface into an existing program.

For new projects, using the import function of the GAT wizard for creating the AxisInterface is strongly recommended.

The following procedure can be used to integrate the AxisInterface into an existing project:

1. Use the icon at upper left in the menu bar to start the GAT wizard.
2. In the "Import template code" dialog that appears, select "AxisInterface".
Now the AxisInterface example project is created.
3. The structures "arAxisCtrl_gb", "arAxisStatus_gb" and "AxisData" can now be used to approach the axes.
4. The structures "ImcCtrl" and "ImcStatus" can be used to operate the control.

In the program "PlcProg" the access to the AxisInterface is demonstrated. This code must be adapted accordingly for the user's own project.

ML_TechInterface.library

6.6.3 AxisInterface Visualizations

Overview

Visualization masks are delivered with the AxisInterface and IMC interface to provide a prefabricated and easy interface for setting and controlling the system axes and the control.

The following visualizations are contained in the GAT and GAT^{compact}:

Visualization	Description
System_Overview_All_Axis	Complete overview of all defined axes, including simple diagnostics, status and jog operating elements.
Axis_Overview	It indicates the current command and values for position and velocity, together with navigation for the position display, velocity display and synchronization display for the current axis. The axes can also be homed and switched off.
Position_mode	Monitoring the positioning mode
Velocity_mode	Monitoring the velocity mode
Synchronization_mode	Monitoring the synchronization mode
Profile_mode	Monitoring the synchronization mode with profiles
ImcInterface_Status_Control	MLC control and status interface of the operation modes, acknowledgements and system diagnostics.
PLC_load	It indicates the minimum, current and maximum PLC task cycle time for the AxisInterfaceType01Prog.

Fig. 6-33: Example project visualizations

The following global variables are used to control and access system information within the visualizations:

- arAxisCtrl_gb[]
- arAxisStatus_gb[]
- AxisData[] (MLC standard)
- VisuAxisNo

System Overview Visualization

The System_Overview_All_Axis visualization allows the user to quickly connect each axis that is configured in the project. The default axes shown (MyVirtualAxis1 and MyVirtualAxis2), are part of the GAT^{compact} example project. Additional axes can be added for display in the offline mode.

This display provides an overall system status, permits to clear errors allows to set emergency stops. Individual axis operating elements provide the axis name, diagnostics, the current position and velocity together with the operation mode status displays. With the SetupMode section, the user can jog and position individual axes as well as start and stop virtual axes.

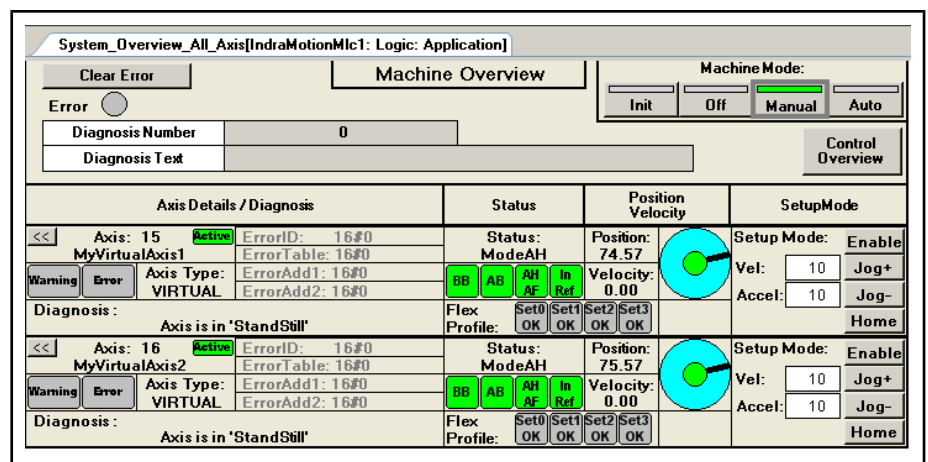


Fig. 6-34: System_Overview_All_Axis display

Adding an axis to the system overview

An axis is added to the system overview display as it is carried out when adding a new visualization: by selecting the respective element and specifying the axis name and axis number. The following steps describe the procedure briefly:

1. Open the corresponding IndraWorks project and select the tab **Logic->Application->Visualizations->SystemOverview**.
2. With IndraLogic in offline mode, double-click the System_Overview_All_Axis visualization.
3. In the Toolbox window, select the button **Frame**  and, under the last line in the axis table, generate an outline with the correct line height and width.

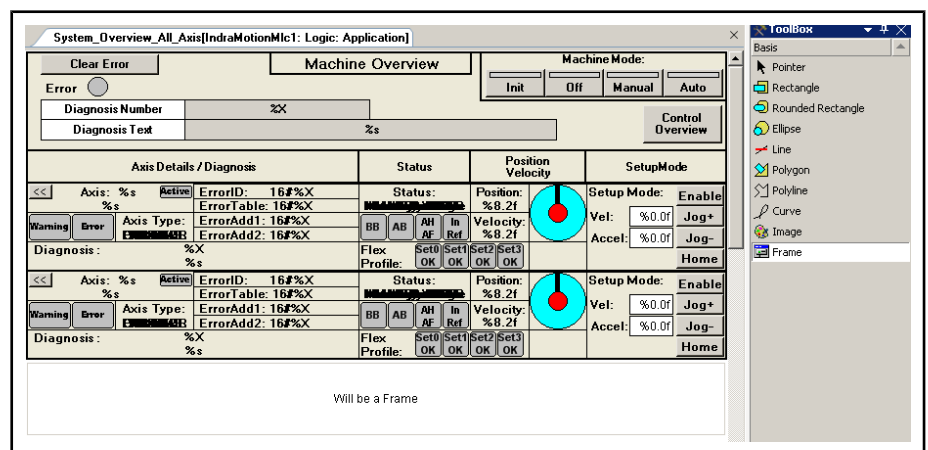


Fig. 6-35: Frame

4. Right-click on the frame and activate Frame selection. Select the **System_Overview** element from the visualization selection window. An entirely new system overview "Axis Line" appears.
5. After clicking in the new visualization, enter the name of the defined axis in the Properties window as the value for **m_Input_AxisName**.
6. The new AxisInterface for the system overview can now be resized and positioned.
7. Recompile the IndraLogic project and go online.



The steps above must be repeated for all axes added to the project.

ML_TechInterface.library

Navigation system overview

An overview display of the individual axes can be opened by clicking on the "Axis Name" button located in the "Details" column of the axis table. From the axis overview display, the user can navigate to the operation mode displays for position, velocity and synchronization.

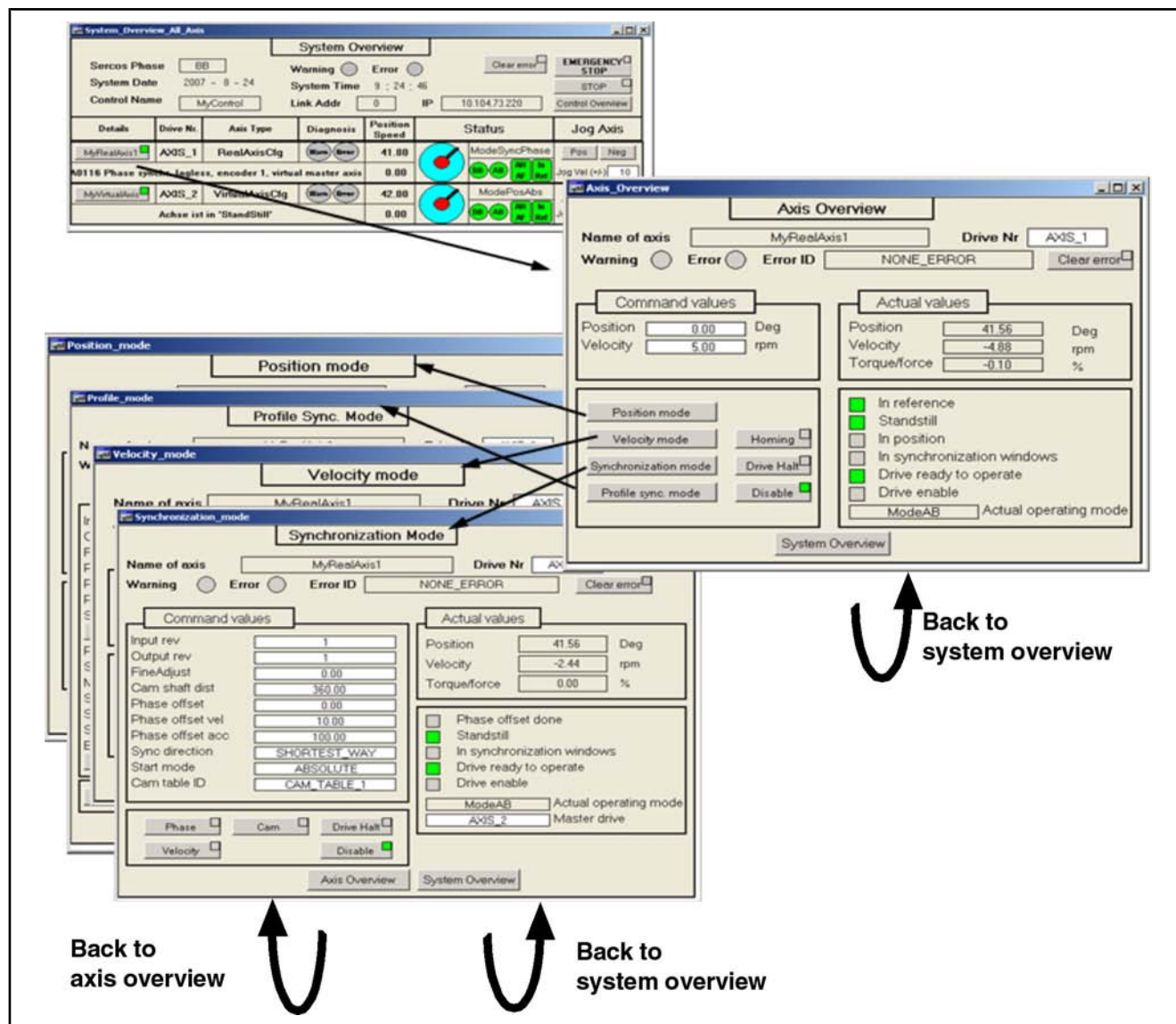


Fig.6-36: Navigation system overview

Axis Overview Visualization

The Axis Overview visualization provides a simple interface for the current command and actual values of an axis. Command values for the positioning mode and the velocity mode are provided. The user can switch back and forth between the axes in a project by clicking on the **Drive No.** field and entering the respective axis number.

The interface is titled "Axis Overview" and contains the following elements:

- Name of axis:** MyRealAxis1
- Drive Nr:** AXIS_1
- Warning:** ☐ **Error:** ☐ **Error ID:** 0 **Clear error:**
- Command values:**
 - Position: 0.00 Deg
 - Velocity: 5.00 rpm
- Actual values:**
 - Position: 41.96 Deg
 - Velocity: -0.09 rpm
 - Torque/force: 0.00 %
- Operation modes:**
 - Position mode:
 - Velocity mode:
 - Synchronization mode:
 - Profile sync. mode:
 - Homing:
 - Drive Halt:
 - Disable:
- Status LEDs:**
 - ☒ In reference
 - ☒ Standstill
 - ☐ In position
 - ☐ In synchronization windows
 - ☒ Drive ready to operate
 - ☐ Drive enable
- Actual operating mode:** 0
- System Overview:**

Fig.6-37: Axis Overview Visualization

The buttons for operation modes Positioning Mode, Velocity Control, Synchronization Mode and Synchronization mode with Profiles allows the user to switch to the respective operation mode in which special values can be entered for the individual operation mode. The current axis can be homed, switched after Drive Halt while maintaining the torque (AH) or switched off (Ab).

The following table contains the (general) AxisInterface structure elements used as part of the visualization:

Visualization element	AxisInterface structure element
Drive no.	VisuAxisNo
Command: Position	arAxisCtrl_gb[].PosMode.Position
Command: Velocity	arAxisCtrl_gb[].PosMode.Velocity
Homing button	arAxisCtrl_gb[].Admin._OpMode: = ModeHoming
Drive Halt button	arAxisCtrl_gb[].Admin._OpMode: = ModeAh
Switch off button	arAxisCtrl_gb[].Admin._OpMode: = ModeAb
Current: Position	AxisData[].rActualPosition_i
Current: Velocity	AxisData[].rActualVelocity_i
Current: Torque/force	AxisData[].rActualTorqueForce_i
LED: Homed	AxisData[].Axis_Homed
LED: At standstill	AxisData[].Axis_Standstill
LED: In-position	AxisData[].Axis_InPosition
LED: In-synchronization window	AxisData[].Axis_InSynchron
LED: Drive ready for operation	AxisData[].Axis_InAb

ML_TechInterface.library

Visualization element	AxisInterface structure element
LED: Drive enabled	AxisData[].Axis_Power
Current operation mode	arAxisStatus_gb[].Admin._OpModeAck

Fig.6-38: Axis overview structure elements



The arAxisCtrl_gb[] and arAxisStatus_gb[] structures are part of the ML_TechInterface library. The AxisData[] structure is part of the ML_Base library.

Positioning Mode Visualization

The positioning mode visualization is used to enter command values for

- Position
- Distance
- Velocity
- Acceleration
- Deceleration

The user can select the positioning type (absolute, relative or additive).

The interface, titled 'Axis_Overview', displays the following information:

- Position mode:** A dropdown menu.
- Name of axis:** MyRealAxis1
- Drive Nr:** AXIS_1
- Warning:** ☐ (selected)
- Error:** ☐
- Error ID:** 0
- Clear error:** Button
- Command values:**
 - Position: 42.00
 - Distance: 0.00
 - Velocity: 200.00
 - Acceleration: 10.00
 - Deceleration: 10.00
- Actual values:**
 - Position: 41.98
 - Velocity: -0.31
 - Torque/force: 1.20
- Positioning mode selection:**
 - Absolute: ☒
 - Relative: ☐
 - Additive: ☐
 - Drive Halt: ☐
 - Disable: ☐
- Status indicators:**
 - Standstill: ☒
 - In position: ☒
 - Drive ready to operate: ☒
 - Drive enable: ☒
- Actual operating mode:** 4
- Navigation:** Axis Overview (selected), System Overview

Fig.6-39: Positioning mode visualization

For more information, refer to the following positioning types:

For absolute, see [chapter 6.4.6 "Absolute Positioning" on page 65](#).

For relative, see [chapter 6.4.7 "Relative Positioning" on page 66](#).

For additive, see [chapter 6.4.7 "Relative Positioning" on page 66](#).

The following table contains the (general) AxisInterface structure elements used as part of the visualization:

Visualization element	AxisInterface structure element
Drive no.	VisuAxisNo
Command: Position	arAxisCtrl_gb[].PosMode.Position
Command: Distance	arAxisCtrl_gb[].PosMode.Distance
Command: Velocity	arAxisCtrl_gb[].PosMode.Velocity
Command: Acceleration	arAxisCtrl_gb[].PosMode.Acceleration
Command: Deceleration	arAxisCtrl_gb[].PosMode.Deceleration
Drive Halt button	arAxisCtrl_gb[].Admin._OpMode: = ModeAh
Switch off button	arAxisCtrl_gb[].Admin._OpMode: = ModeAb
Current: Position	AxisData[].rActualPosition_i
Current: Velocity	AxisData[].rActualVelocity_i
Current: Torque/force	AxisData[].rActualTorqueForce_i
LED: At standstill	AxisData[].Axis_Standstill
LED: In-position	AxisData[].Axis_InPosition
LED: Drive ready for operation	AxisData[].Axis_InAb
LED: Drive enabled	AxisData[].Axis_Power
Current operation mode	arAxisStatus_gb[].Admin._OpModeAck

Fig. 6-40: Positioning mode structure elements

Velocity Mode Visualization

The velocity mode visualization is used to enter command values for

- Velocity
- Acceleration
- Deceleration

ML_TechInterface.library

The interface is titled "Axis_Overview" and "Velocity mode". It displays the following information:

- Name of axis:** MyRealAxis1
- Drive Nr:** AXIS_1
- Warning:** (Warning icon) (Error icon)
- Error ID:** 0
- Clear error:** (button)
- Command values:**
 - Velocity: 5.00
 - Acceleration: 10.00
 - Deceleration: 10.00
- Actual values:**
 - Position: 196.13 Deg
 - Velocity: 4.15 rpm
 - Torque/force: 3.70 %
- Control buttons:** Velocity (LED), Drive Halt (button), Disable (button)
- Status LEDs:**
 - In velocity (LED)
 - Standstill (LED)
 - Drive ready to operate (LED)
 - Drive enable (LED)
- Actual operating mode:** 32
- Navigation buttons:** Axis Overview, System Overview

Fig.6-41: Velocity mode visualization

The following table contains the (general) AxisInterface structure elements used as part of the visualization:

Visualization element	AxisInterface structure element
Drive no.	VisuAxisNo
Command: Velocity	arAxisCtrl_gb[].VelMode.Velocity
Command: Acceleration	arAxisCtrl_gb[].VelMode.Acceleration
Command: Deceleration	arAxisCtrl_gb[].VelMode.Deceleration
Drive Halt button	arAxisCtrl_gb[].Admin._OpMode: = ModeAh
Switch off button	arAxisCtrl_gb[].Admin._OpMode: = ModeAb
Current: Position	AxisData[].rActualPosition_i
Current: Velocity	AxisData[].rActualVelocity_i
Current: Torque/force	AxisData[].rActualTorqueForce_i
LED: In velocity	AxisData[].Axis_InVelocity
LED: At standstill	AxisData[].Axis_Standstill
LED: In-position	AxisData[].Axis_InPosition
LED: Drive ready for operation	AxisData[].Axis_InAb
LED: Drive enabled	AxisData[].Axis_Power
Current operation mode	arAxisStatus_gb[].Admin._OpModeAck

Fig.6-42: Velocity mode structure elements

Synchronization Mode Visualization

The synchronization mode visualization is used to control the functions of the electronic gear of the drive. A requirement is that fine adjustment and phase offset elements are used. The **Drive No.** field defines the slave axis and the **Master drive** field defines the master axis.

The user can select the type of synchronization mode (phase synchronization or velocity synchronization, cam or MotionProfile).

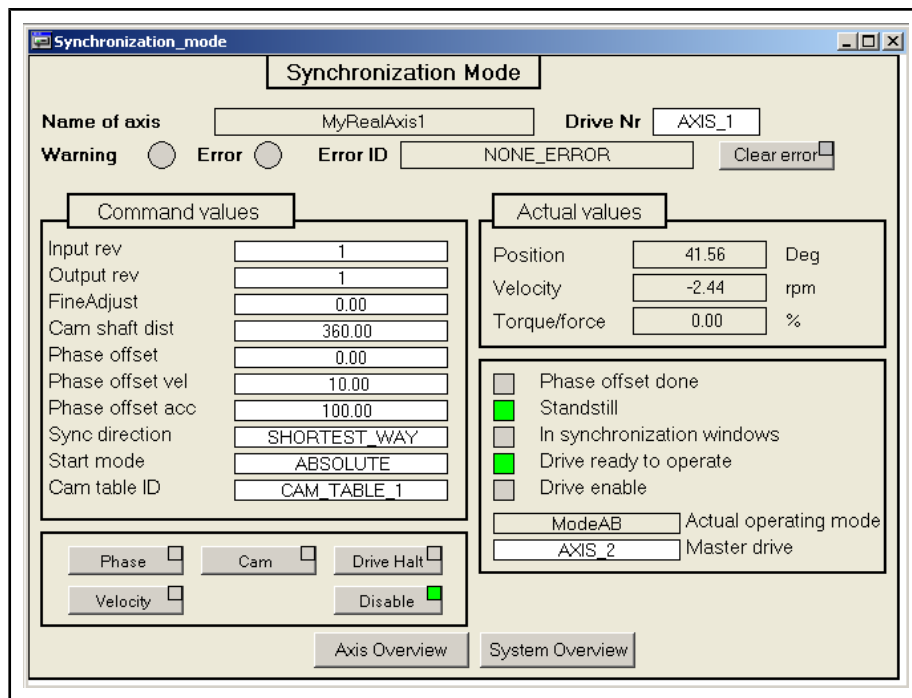


Fig.6-43: Synchronization mode visualization

For more information, refer to the following types of synchronization modes:

For phase synchronization, see [chapter 6.4.10 "Operation Mode Phase Synchronization"](#) on page 68.

For velocity synchronization, see [chapter 6.4.11 "Operation Mode Velocity Synchronization"](#) on page 69.

For cams, see [chapter 6.4.12 "Operation Mode Electronic Cam"](#) on page 69.

The following table contains the (general) AxisInterface structure elements used as part of the visualization:

Visualization element	AxisInterface structure element
Drive no.	VisuAxisNo
Command: Input revolutions	arAxisCtrl_gb[].SyncMode.InputRevolution
Command: Output revolutions	arAxisCtrl_gb[].SyncMode.InputRevolution
Command: Fine adjustment	arAxisCtrl_gb[].SyncMode.Fineadjust
Command: Cam shaft stroke	arAxisCtrl_gb[].SyncMode.CamShaftDistance
Command: Phase offset	arAxisCtrl_gb[].SyncMode.PhaseOffset
Command: Phase offset velocity	arAxisCtrl_gb[].SyncMode.PhaseOffsetVel
Command: Phase offset acceleration	arAxisCtrl_gb[].SyncMode.PhaseOffsetAcc
Command: Synchronization direction	arAxisCtrl_gb[].SyncMode.SyncDirection
Command: Synchronization startup mode	arAxisCtrl_gb[].SyncMode.StartMode
Command: Cam ID	arAxisCtrl_gb[].SyncMode.CamTableID
Phase button	arAxisCtrl_gb[].Admin._OpMode: = ModeSyncPhase

ML_TechInterface.library

Visualization element	AxisInterface structure element
Velocity button	arAxisCtrl_gb[].Admin._OpMode: = ModeSyncVelocity
Cam button	arAxisCtrl_gb[].Admin._OpMode: = ModeSyncCam
Drive Halt button	arAxisCtrl_gb[].Admin._OpMode: = ModeAh
Switch off button	arAxisCtrl_gb[].Admin._OpMode: = ModeAb
Current: Position	AxisData[].rActualPosition_i
Current: Velocity	AxisData[].rActualVelocity_i
Current: Torque/force	AxisData[].rActualTorqueForce_i
LED: Phase offset executed	arAxisStatus_gb[].SyncMode.PhasingSlaveDone
LED: At standstill	AxisData[].Axis_Standstill
LED: In-synchronization window	AxisData[].Axis_InSynchron
LED: Drive ready for operation	AxisData[].Axis_InAb
LED: Drive enabled	AxisData[].Axis_Power
Current operation mode	arAxisStatus_gb[].Admin._OpModeAck
Master axis drive	arAxisCtrl_gb[].SyncMode.Master.AxisNo

Fig. 6-44: Synchronization operation mode structure elements

Visualization, Synchronization Mode with Profiles

The visualization "synchronization mode with profiles" is used to control the functionalities of the electronic gear of the drive with profiles. A requirement is that fine adjustment and phase offset elements are used. The **Drive No.** field defines the slave axis and the **Master drive** field defines the master axis.

The user can select the type of synchronization mode with profiles to be used (FlexProfile or MotionProfile).

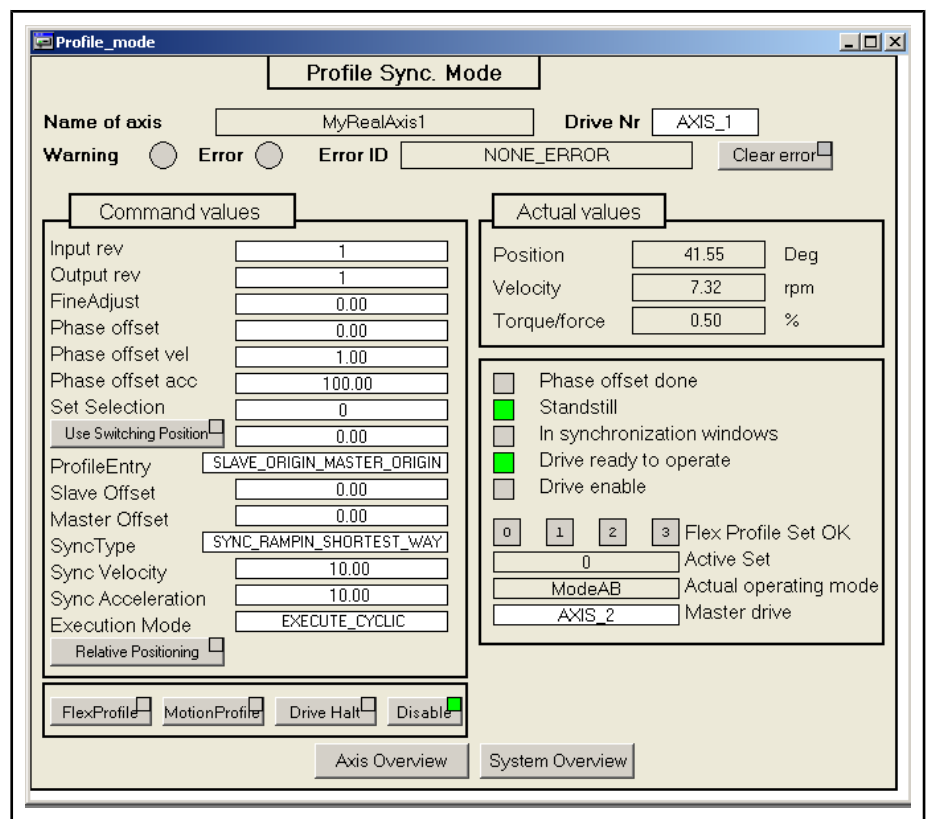


Fig. 6-45: Visualization - synchronization mode with profiles

For more information, refer to the following types of synchronization modes:

For MotionProfile, see [chapter 6.4.13 "Operation Mode Electronic MotionProfile" on page 70](#).

For FlexProfile, see [chapter 6.4.14 "Operation Mode Electronic FlexProfile" on page 71](#).

The following table contains the (general) AxisInterface structure elements used as part of the visualization:

Visualization element	AxisInterface structure element
Drive no.	VisuAxisNo
Command: Input revolutions	arAxisCtrl_gb[].SyncMode.InputRevolution
Command: Output revolutions	arAxisCtrl_gb[].SyncMode.InputRevolution
Command: Fine adjustment	arAxisCtrl_gb[].SyncMode.Fineadjust
Command: Phase offset	arAxisCtrl_gb[].SyncMode.PhaseOffset
Command: Phase offset velocity	arAxisCtrl_gb[].SyncMode.PhaseOffsetVel
Command: Phase offset acceleration	arAxisCtrl_gb[].SyncMode.PhaseOffsetAcc
Command: Profile set selection	arAxisCtrl_gb[].SyncMode.Profile.SetSelection
Relative Positioning button	arAxisCtrl_gb[].SyncMode.Profile.RelativePositioning
MotionProfile button	arAxisCtrl_gb[].Admin._OpMode: = ModeSyncProfile
FlexProfile button	arAxisCtrl_gb[].Admin._OpMode: = ModeFlexProfile
Drive Halt button	arAxisCtrl_gb[].Admin._OpMode: = ModeAh

ML_TechInterface.library

Visualization element	AxisInterface structure element
Switch off button	arAxisCtrl_gb[].Admin._OpMode: = ModeAb
Current: Position	AxisData[].rActualPosition_i
Current: Velocity	AxisData[].rActualVelocity_i
Current: Torque/force	AxisData[].rActualTorqueForce_i
LED: Phase offset executed	arAxisStatus_gb[].SyncMode.PhasingSlaveDone
LED: At standstill	AxisData[].Axis_Standstill
LED: In-synchronization window	AxisData[].Axis_InSynchron
LED: Drive ready for operation	AxisData[].Axis_InAb
LED: Drive enabled	AxisData[].Axis_Power
Current operation mode	arAxisStatus_gb[].Admin._OpModeAck
Master axis drive	arAxisCtrl_gb[].SyncMode.Master.AxisNo

Fig. 6-46: Synchronization operation mode with profile and structure elements

IMC Interface Control and Status Visualization

The ImcInterface_Status_Control visualization permits the access on control functionalities such as switching between parameter mode and operating mode, clearing errors and restarting the phase switching. The visualization also provides the control status of the various SERCOS operation modes and error diagnostics.

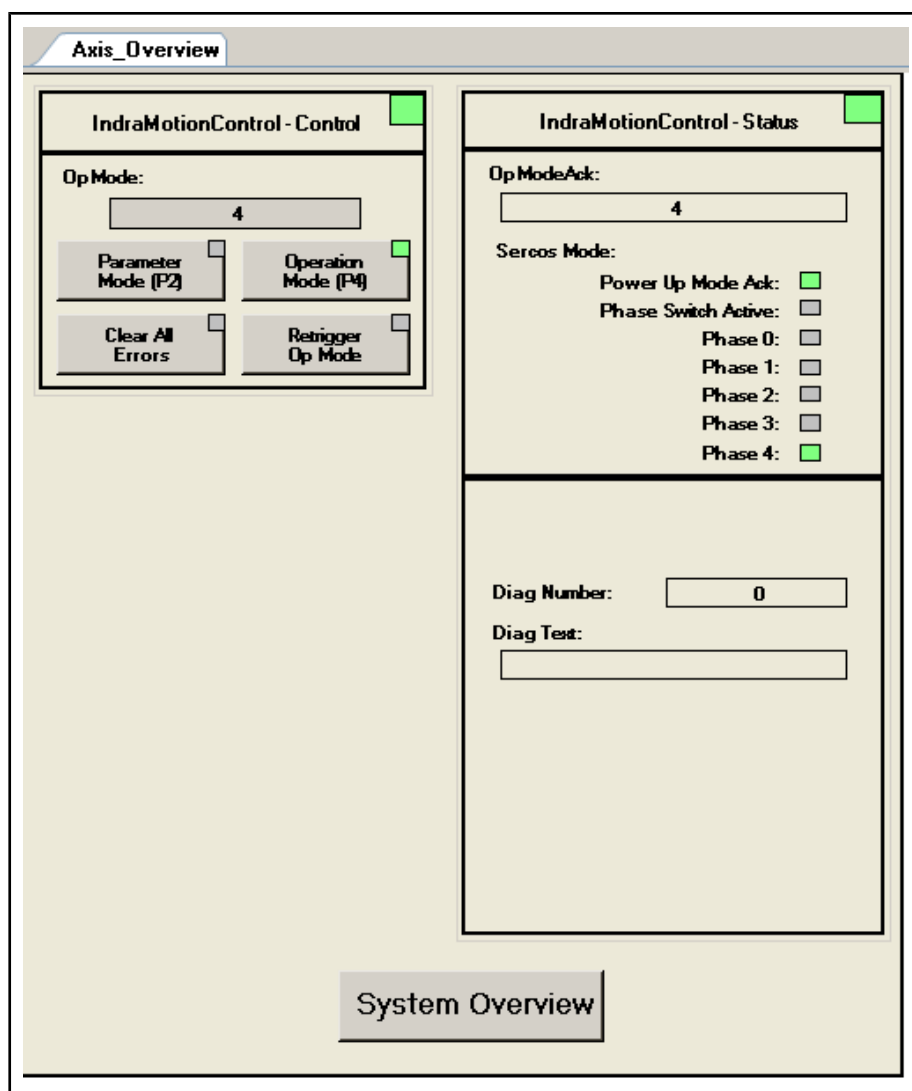


Fig. 6-47: ImcInterface status and control visualization

The following table contains the (general) AxisInterface structure elements used as part of the visualization:

Visualization element	AxisInterface structure element
IndraMotion Control - control	
Phase	ImcCtrl.Admin._OpMode
Parameterization mode (P2)	ImcCtrl.Admin._OpMode: = IMC_MODE_CTRL_P2
Operating mode (P4)	ImcCtrl.Admin._OpMode: = IMC_MODE_CTRL_BB
Clear all errors	ImcCtrl.Admin.ClearError
Restart phase switching	ImcCtrl.Admin.RetriggerOpMode
IndraMotion Control - status	
Phase acknowledgement	ImcStatus.Admin._OpModeAck
LED: Switch-on mode acknowledgement	ImcStatus.Admin.PwrUpModeAck
LED: Phase switching active	ImcStatus.Admin.PhaseSwitchActive

ML_TechInterface.library

Visualization element	AxisInterface structure element
LED: Phase 0	ImcStatus.Admin.ModeStatus_P0
LED: Phase 1	ImcStatus.Admin.ModeStatus_P1
LED: Phase 2	ImcStatus.Admin.ModeStatus_P2
LED: Phase 3	ImcStatus.Admin.ModeStatus_P3
LED: Phase 4	ImcStatus.Admin.ModeStatus_BB
LED: Clear error acknowledgement	ImcStatus.Diag.ClearErrorAck
LED: Error	ImcStatus.Diag.Error
LED: Warning	ImcStatus.Diag.Warning
ErrorID	ImcStatus.Diag.ErrorID
Error identification: Table	ImcStatus.Diag.ErrorIdent.Table
Error identification: Additional 1	ImcStatus.Diag.ErrorIdent.Additional1
Error identification: Additional2	ImcStatus.Diag.ErrorIdent.Additional2

Fig. 6-48: IMC interface structure elements

PLC Load Visualization

The PLC_load visualization is used to monitor the PLC load. The minimum, current and maximum PLC cycle times for cyclic calls of the AxisInterface function block in the TE_AxisInterfaceType01Prog are displayed.

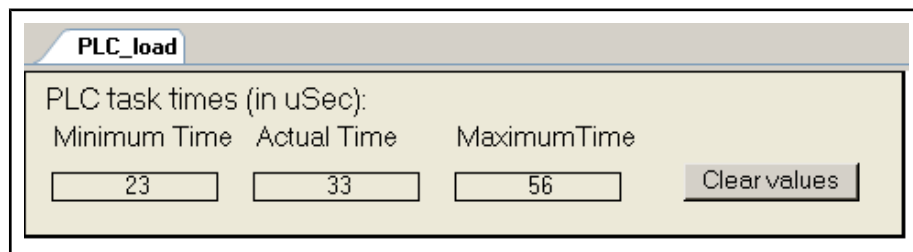


Fig. 6-49: PLC load visualization

6.7 AxisInterface User Extensions

6.7.1 Overview

The **arAxisCtrl_gb[]** and **arAxisStatus_gb[]** structures can be extended by the user to adapt the AxisInterface to special applications.



The **arAxisCtrl_gb[]** and **arAxisStatus_gb[]** structures of the user interface are

MB_Axis_Control_Type01 and MB_Axis_Status_Type01 data structures (see [chapter 6.8.1 "Overview" on page 93](#)).

Since the AxisInterface data structure is integrated in the ML_TechInterface library and is thus not accessible by the user, a new data structure that corresponds to the existing one must be created within the current project so that functionalities can be added.

This section shows how the AxisInterface is extended by adding the **MB_Phasing** functionality. The MB_Phasing function block allows to add an offset to a

master axis. The master axis is connected as input to the **MC_CamIn** or the **MC_GearInPos** synchronization function block.

The **MB_PhasingSlave** functionality is integrated into the AxisInterface using the

`arAxisCtrl_gb[ ].SyncMode.PhaseOffset`, `(.PhaseOffsetVel)`, `(.PhaseOffsetAcc)` elements and the

`arAxisStatus_gb[ ].SyncMode.PhasingSlaveDone` output.

The following new inputs and outputs are defined:

- `arAxisCtrl_gb[ ].SyncMode.PhaseShift`
- `arAxisCtrl_gb[ ].SyncMode.PhaseShiftVel`
- `arAxisCtrl_gb[ ].SyncMode.PhaseShiftAcc`
- `arAxisStatus_gb[ ].SyncMode.PhasingMasterDone`



The `arAxisCtrl_gb[ ].SyncMode.PhaseShift` input is scanned cyclically and activated immediately when there is a change.

The following user-specific extensions are implemented in the GAT and GAT compact.

- Jog functionality and homing as "SetupMode"
- Parameter A-0-2730 "phase offset start of table" as "SyncMode.AngleOffset"
- Parameter A-0-2790 "synchronization velocity" as "SyncMode.SyncVelocity"
- Parameter A-0-2791 "synchronization acceleration" as "SyncMode.SyncAcceleration"
- Function block `MB_Phasing()` as `SyncMode.PhaseShift`, `-.PhaseShiftVel` and `-.PhaseShiftAcc`

In the following it is assumed that the structure prepared in the GAT or GAT compact will be used. The following describes the required changes in the POU's specified there.

Notes regarding the implementation of user extensions

The user extensions were implemented using the object-oriented extensions from IndraLogic 2G. Please note the following particularities.

- The function block (FB) `TE_AxisInterface` is derived from Basis FB `MB_AxisInterfaceType01`. The keyword "SUPER" can be used to call the Basis FB or methods/actions related to the Basis FB. For example, at various places, the method `mSetError` of the FB `MB_AxisInterfaceType01` is called using `SUPER^.mSetError(...)`; to enter errors in the diagnostic system.
- Within the FB `TE_AxisInterface`, the data of `arAxisCtrl_gb` can be accessed using the inputs `AdminCtrlExt`, `PosModeCtrlExt`, etc. and the data of `arAxisStatus_gb` can be accessed using `AdminStatusExt`, `DiagStatusExt`, etc.

The inputs `AdminCtrl` (without Ext), etc. belong to the Basis FB and are not to be used.

- The inputs of the FB `TE_AxisInterface` are defined as "REFERENCE TO". In the method `mlnitExtension`, the reference are initialized once and do not have to be transferred again during a cyclical call of the FB.
- If the basis structures `TE_AXIS_CONTROL_TYPE01` and `TE_AXIS_STATUS_TYPE01` are to be extended with additional substructures, the fol-

ML_TechInterface.library

Following additional steps must be added to the procedure listed below (see SetupMode and SetupModeAck as examples):

- Enter new elements in TE_AXIS_CONTROL_TYPE01 and TE_AXIS_STATUS_TYPE01.
- Add the additional inputs at the FB TE_AxisInterface as REFERENCE TO.
- Initialize the references in the method mInitExtension of the FB TE_AxisInterface.

6.7.2 Extending the arAxisCtrl_gb[] Structure

To add the additional functionality of the MB_Axis_Control_Type01 structure, the user must create a new structure, derive the substructures from the existing structure and then add the new elements. In the GAT/GAT^{compact}, a structure TE_AXIS_CONTROL_TYPE01 and the substructures TE_AXIS_ADMINISTRATION, etc. are already prepared.

Proceed according to the following steps to extend the functionality of the TE_AXIS_SYNCHRONISATION structure:

1. Open the respective project in IndraWorks.
2. Open the TE_AXIS_SYNCHRONISATION structure.

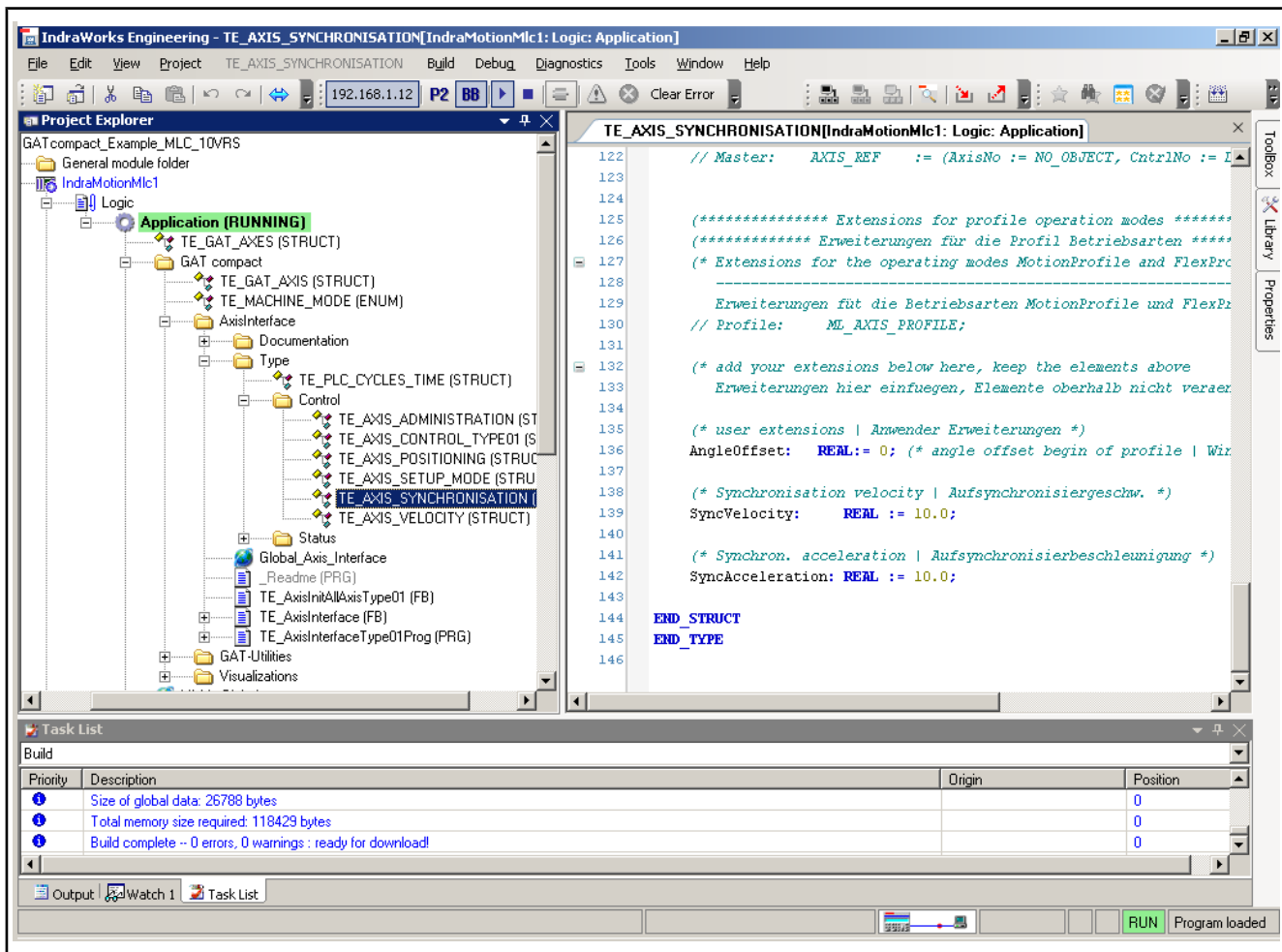


Fig.6-50: Opening the TE_AXIS_SYNCHRONISATION structure

3. Add the three following new element allocations

- PhaseShift

- PhaseShiftVel
- PhaseShiftAcc

in "TE_AXIS_SYNCHRONISATION" below the comment as REAL type.

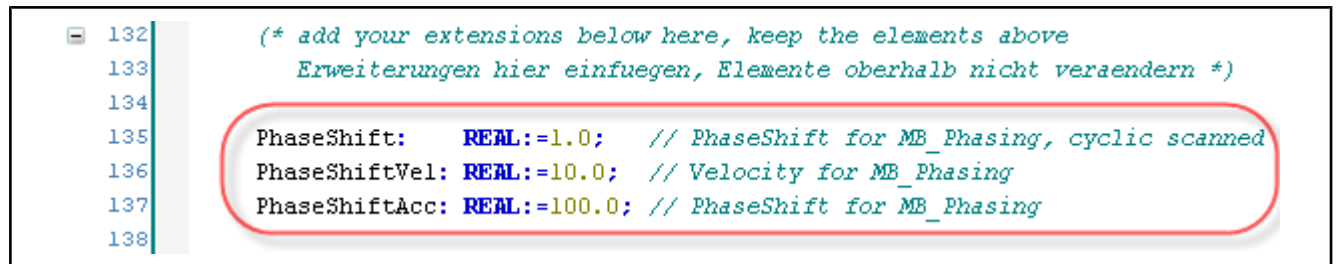


Fig.6-51: New elements added to TE_AXIS_SYNCHRONISATION

6.7.3 Extending the arAxisStatus_gb[] Structure

Extend the **arAxisStatus_gb[]** structure using the steps in "Extending the arAxisCtrl_gb[] structure". The following steps describe the procedure briefly:

1. Open the object TE_AXIS_SYNC_STATUS.
2. Add the following new element allocations **PhasingMasterDone** as a BOOL type to the TE_AXIS__SYNC_STATUS below the comment.

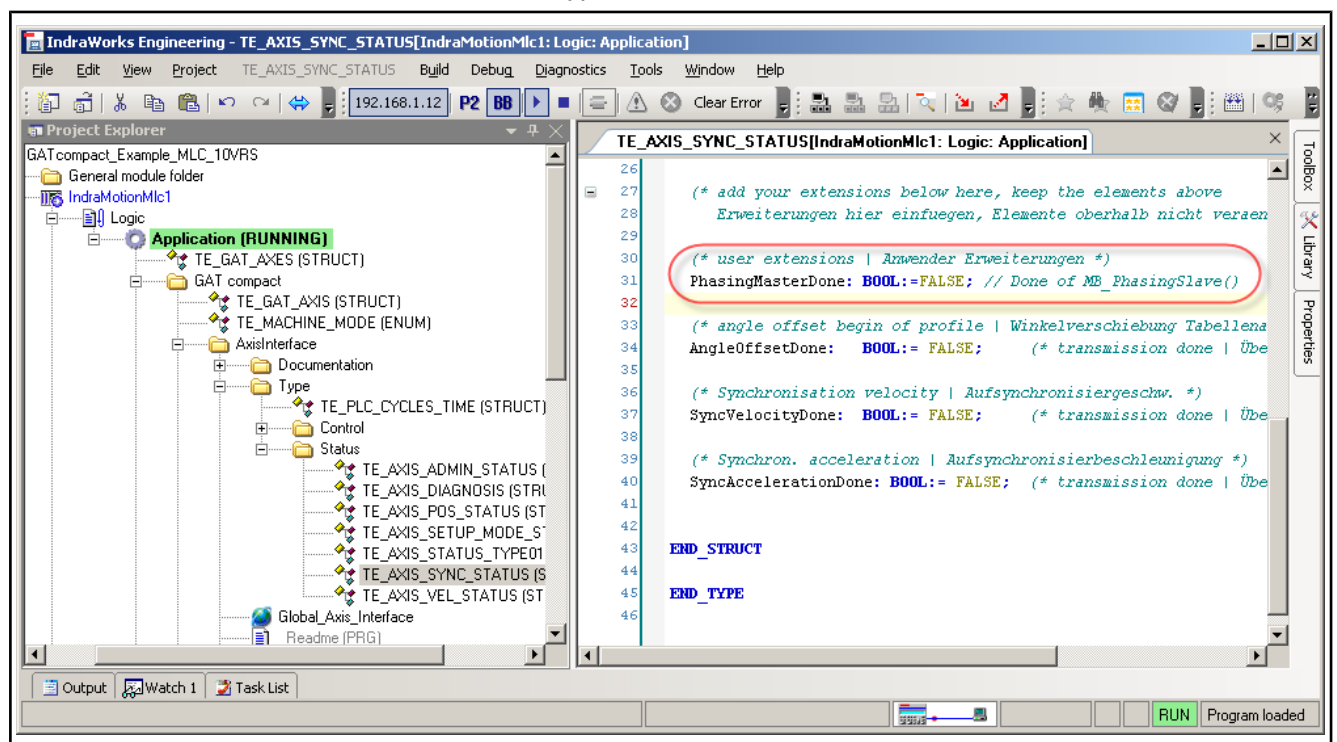


Fig.6-52: New element added to TE_AXIS_SYNC_STATUS

6.7.4 Extending the Function Block

The last step in the user extension sequence is to extend the function block so that the new elements can be used.

1. In the project tree, right-click to select **TE_AxisInterface** and **Add -> Action**.
2. Name the new action **actPhaseShift**.
3. Declare the following variables in the FB TE_AxisInterface.

ML_TechInterface.library

- fbAxisPhasing: MB_Phasing;
- bExecPhasing: BOOL := FALSE;
- rOldPhaseShift: REAL := 0.01; // default old value not equal 0 to enforce a transmission

4. The action could be programmed in this way.

```

actPhaseShift[IndraMotionMlc1: Logic: Application: TE_AxisInterface]
1  //*****
2  // Implements a PhasingMaster with the function block MB_Phasing
3  //*****
4  //***** handle phase offset when sync-mode is active *****
5  IF ( (INT_TO_WORD(AdminStatusExt._OpModeAck) AND MASK_MODE_PHASE_OFFSET) <> 0 ) THEN
6      IF ( rOldPhaseShift <> SyncModeCtrlExt.PhaseShift) THEN
7          // start transmission
8          SyncModeStatusExt.PhasingMasterDone := FALSE;
9          bExecPhasing := TRUE;
10         END IF
11         IF fbAxisPhasing.Done THEN
12             // finished GOOD transmission
13             SyncModeStatusExt.PhasingMasterDone := TRUE;
14             bExecPhasing := FALSE;
15             // remember old value
16             rOldPhaseShift := SyncModeCtrlExt.PhaseShift;
17         ELSEIF (fbAxisPhasing.Error = TRUE) THEN
18             // finished BAD transmission
19             bExecPhasing := FALSE;
20             SUPER^.mSetError( Error      := TRUE,
21                             ErrorID    := fbAxisPhasing.ErrorID,
22                             ErrorIdent := fbAxisPhasing.ErrorIdent);
23         END IF
24         // keep the FB always active to generate PhasingMasterDone
25         fbAxisPhasing(
26             Execute      := bExecPhasing,
27             PhaseShift   := SyncModeCtrlExt.PhaseShift,
28             Velocity     := SyncModeCtrlExt.PhaseShiftVel,
29             Acceleration := SyncModeCtrlExt.PhaseShiftAcc,
30             Axis         := AdminCtrlExt.Axis );
31     ELSE
32         SyncModeStatusExt.PhasingMasterDone := FALSE; // Reset, when no Sync-Mode active
33     END_IF // end of 'IF( (INT_TO_WORD(AdminStatusExt._OpMode.enAck) AND MASK_MODE_PHASE_OFFSET) <> 0 ) THEN'

```

Fig.6-53: Extending the user program

5. IN the FB TE_AxisInterface call up the new action actPhaseShift();.
6. Compile the project again and check it for programming errors
7. Load the project in the control.

The new input and output elements are now part of the AxisInterface structure and can be viewed using the UserVarGlobal variables.

◆ FineAdjust	REAL	0
◆ CamShaftDistance	REAL	360
◆ PhaseOffset	REAL	0
◆ PhaseOffsetVel	REAL	1
◆ PhaseOffsetAcc	REAL	100
◆ SyncDirection	MC_SYNC_DIRECTION	MC_SYNC_DIRECTI...
◆ StartMode	MC_START_MODE	MC_START_MODE.A...
◆ CamTableID	MC_CAM_ID	MC_CAM_ID.CAM_T...
⊕ ◆ Master	AXIS_REF	
⊕ ◆ Profile	ML_AXIS_PROFILE	
⊕ ◆ AdjFineAdjust	MB_AXIS_ADJUST_DATA	
⊕ ◆ AdjCamShaftDistance	MB_AXIS_ADJUST_DATA	
⊕ ◆ AdjPhaseOffset	MB_AXIS_ADJUST_DATA	
◆ AngleOffset	REAL	0
◆ PhaseShift	REAL	0
◆ PhaseShiftVel	REAL	10
◆ PhaseShiftAcc	REAL	100
◆ SyncVelocity	REAL	10
◆ SyncAcceleration	REAL	10
IndraMotionMlc1.Application.UserVarGlobal.arAxisStatus_gb[1]	TE_AXIS_STATUS_TYPE01	
⊕ ◆ Admin	TE_AXIS_ADMIN_STATUS	
⊕ ◆ Diag	TE_AXIS_DIAGNOSIS	
⊕ ◆ PosMode	TE_AXIS_POS_STATUS	
⊕ ◆ VelMode	TE_AXIS_VEL_STATUS	
⊖ ◆ SyncMode	TE_AXIS_SYNC_STATUS	
◆ CamActiveID	MC_CAM_ID	16#0000
◆ ProfileActiveSet	UINT	16#0000
◆ PhasingSlaveDone	BOOL	FALSE
⊕ ◆ AdjFineAdjust	MB_AXIS_ADJUST_STATUS	
⊕ ◆ AdjCamShaftDistance	MB_AXIS_ADJUST_STATUS	
⊕ ◆ AdjPhaseOffset	MB_AXIS_ADJUST_STATUS	
◆ AngleOffsetDone	BOOL	TRUE
◆ SyncVelocityDone	BOOL	TRUE
◆ SyncAccelerationDone	BOOL	TRUE
◆ PhasingMasterDone	BOOL	FALSE
⊕ ◆ PtrAxisData	POINTER TO ML_AXISDAT...	16#0A6B1B94
⊕ ◆ SetupMode	TE_AXIS_SETUP_MODE_S...	

Fig.6-54: arAxisCtrl_gb and arAxisStatus_gb structures with user extensions

6.8 AxisInterface Structures

6.8.1 Overview

The AxisInterface provides a data interface for easy axis control. The data structures for the AxisInterface are described in the following section.



The AxisInterface in the programming template GAT (Generic Application Template) or GAT^{compact} is prepared such that the user can extend it and a few extensions are already built in. In order to enable these extensions, it is necessary to define the user's own structures in the user project that extend the library structures. The extended structures are identified by the prefix "TE_". If a structure in the GAT or GAT^{compact} is called **TE_AXIS_ADMIN_STATUS**, for example, it is an extended structure of **MB_AXIS_ADMIN_STATUS** and information regarding the structure can be found under the name **MB_AXIS_ADMIN_STATUS**.

The following table shows the user interface for the AxisInterface data structure:

ML_TechInterface.library

User interface	Type	Description
arAxisCtrl_gb[]	MB_AXIS_CONTROL_TYPE01 (GAT:TE_AX-IS_CONTROL_TYPE01)	Control structure, including command values and variables to activate the operation modes
arAxisStatus_gb[]	MB_AXIS_STATUS_TYPE01 (GAT:TE_AX-IS_CONTROL_TYPE01)	Status structure, including diagnostic information and acknowledgements for the operation modes
AxisData[]		Status structure and control structure, including actual values, status bits and access to cyclically configurable user data (processed by the Motion-Control in the firmware).

Fig.6-55: AxisInterface - User interface data structures

6.8.2 MB_AXIS_STATUS_TYPE01 Structure

MB_AXIS_STATUS_TYPE01

Brief description The MB_AXIS_STATUS_TYPE01 data structure is accessed using the **arAxis-Status_gb[]** instance. It contains the following elements:

```

TYPE MB_AXIS_STATUS_TYPE01 :
STRUCT
  (***** Outputs of the following axis - from motion control to logic control *****)
  Admin:    MB_AXIS_ADMIN_STATUS; (* status of the axis | Achs-Status *)
  Diag:     MB_AXIS_DIAGNOSIS; (* axis diagnosis information, error, warning....
    Diagnose Informationen der Achse, Fehler, Warnung.... *)
  PosMode:  MB_AXIS_POS_STATUS; (* Status operating mode positioning (absolute, rel. and add.)
    Statusmeldungen Positionierbetrieb (absolut, relativ und additiv) *)
  VelMode:  MB_AXIS_VEL_STATUS; (* Status operating mode velocity control
    Statusmeldungen Geschwindigkeitsregelung (Freilauf) *)
  SyncMode: MB_AXIS_SYNC_STATUS; (* Status operating mode synchronisation (phase, vel, and cam)
    Statusmeldungen Synchronbetriebsarten (Winkelsync., Geschwindigkeitssync. und Kurvensch.)*
  PtrAxisData: POINTER TO ML_AXISDATA_SM;
  (* Pointer to AxisData of this axis | Zeiger auf AxisData dieser Achse
    - Type .PtrAxisData^. for the intellisense menu | Eingabe .PtrAxisData^. fuer Auswahlmenue*)

  (* add your extensions below here, keep the elements above
    Erweiterungen hier einfuegen, Elemente oberhalb nicht veraendern *)

  (* user extensions | Anwender Erweiterungen *)

END_STRUCT
END_TYPE

```

Fig.6-56: MB_AXIS_STATUS_TYPE01 structure

Library ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD

Dependence MX_ImcInterface.lib for IndraMotion MLD

System IndraMotion MLC/IndraMoton MLD

MB_AXIS_STATUS_TYPE01 Elements

MB_AXIS_ADMIN_STATUS Structure

MB_AXIS_ADJUST_STATUS

Brief description

The MB_AXIS_ADMIN_STATUS structure is a sub-structure of [MB_AXIS_STATUS_TYPE01](#) and contains output status.

The following illustration shows the current MB_AXIS_ADMIN_STATUS structure:

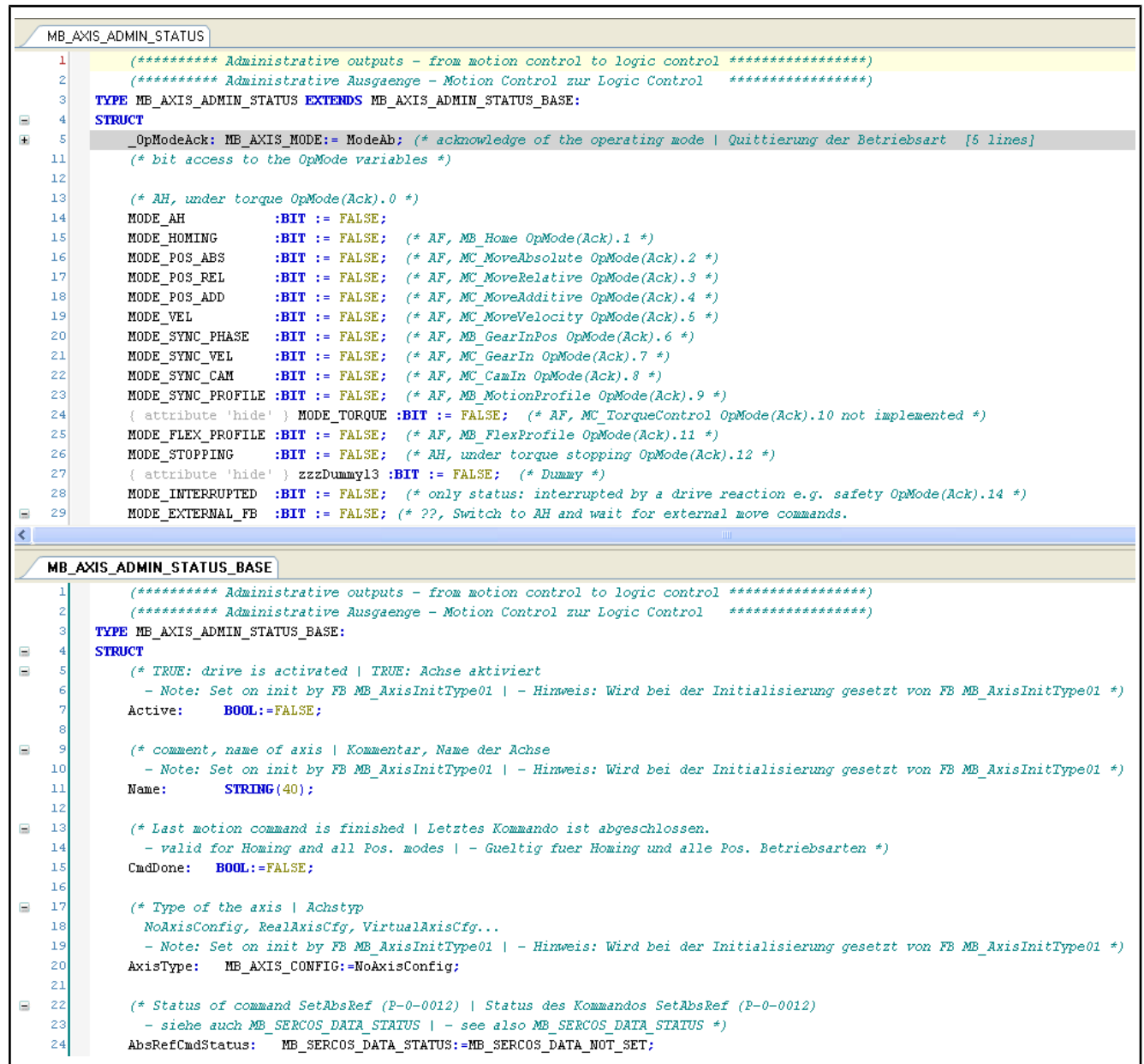


Fig. 6-57: MB_AXIS_ADMIN_STATUS Structure

Library ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD

Dependence MX_ImcInterface.lib for IndraMotion MLD

System IndraMotion MLC/IndraMotion MLD

ML_TechInterface.library

MB_AXIS_ADMIN_STATUS Elements**MB_AXIS_MODE**

Brief description This MB_AXIS_MODE is an enumeration used by the **_OpModeAck** element contained in the **MB_AXIS_ADJUST_STATUS** structure.

The following figure shows the current MB_Axis_Mode enumeration type:

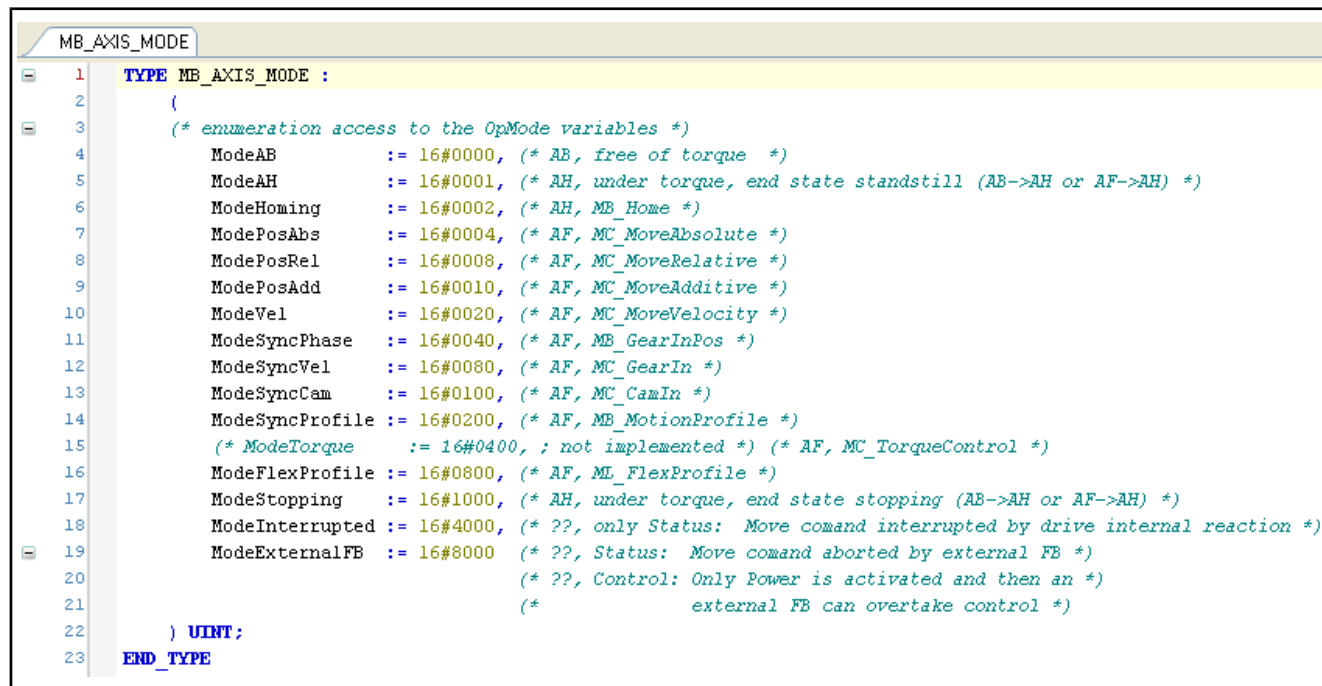


Fig.6-58: MB_Axis_Mode

For details, see chapter "MB_AXIS_MODE" on page 96.

MB_AXIS_CONFIG

Brief description The MB_AXIS_CONFIG is an enumeration type that is used by the **MB_AXIS_ADJUST_STATUS** structure.

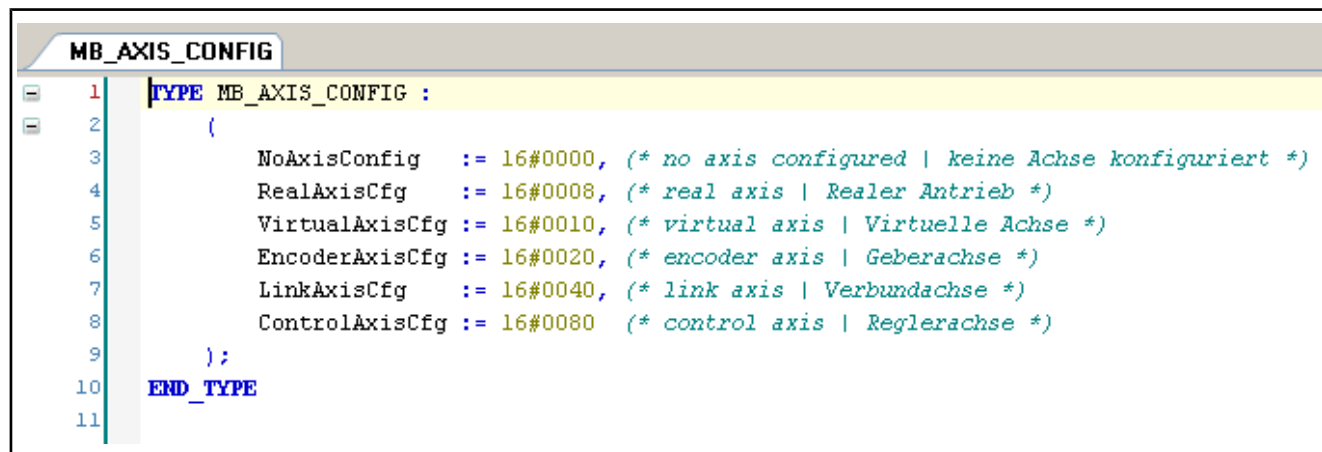


Fig.6-59: MB_AXIS_CONFIG enumeration type

Library ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD

Dependence MX_ImcInterface.lib for IndraMotion MLD

System IndraMotion MLC/IndraMotion MLD

MB_AXIS_DIAGNOSIS structure

MB_AXIS_DIAGNOSIS

Brief description The MB_AXIS_DIAGNOSIS structure is a sub-structure of [MB_AXIS_STATUS_TYPE01](#) and contains warning and error outputs. The following illustration shows the current MB_AXIS_DIAGNOSIS structure:

```

TYPE MB_AXIS_DIAGNOSIS :
STRUCT
  (***** Diagnosis outputs - from motion control to logic control *****)
  (***** Diagnose-Ausgaenge - Motion Control zur Logic Control *****)

  Warning:      BOOL := FALSE; (* Warning, copy of | Warnung, Kopie von
    AxisData[].Axis_Warning *)

  Error:        BOOL := FALSE;
  (* Error -> AxisData[].Axis_Error OR Error from an FB in the Axis-Interface
    Fehler -> AxisData[].Axis_Error ODER Fehler von einem FB im Axis-Interface *)

  ErrorID:      ERROR_CODE; (* Error type (internal used FB) | Fehlertyp (intern genutzter FB)
    - only valid when Error = TRUE | - gueltig wenn Error = TRUE *)

  ErrorIdent:   ERROR_STRUCT; (* Error id. (internal used FB) | Fehler-Id. (intern genutzter FB)
    - only valid when Error = TRUE | - gueltig wenn Error = TRUE *)

  Number:      DWORD; (* diagnosis number of the axis | Diagnosenummer der Achse
    - only valid in operating mode | nur gueltig im Betriebsmodus
    - refresh rate controlled with | Auffrischrate gesteuert mit
      arAxisCtrl[].Admin.DiagNbrRefreshTime *)

  Message:     STRING(60); (* diagnosis message of the axis | Diagnosemeldung der Achse
    - only valid in operating mode | nur gueltig im Betriebsmodus
    - refreshed when "Number" changes | Aufgefrischt bei Aenderung von "Number" *)

END_STRUCT
END_TYPE

```

Fig.6-60: MB_AXIS_DIAGNOSIS structure

Diagnostic displays

arAxisCtrl_gb[iColumn].Admin.DiagNbrRefreshTime is used to control the refresh rate of the element "Number". As soon as element "Number" changes, the text "Message" is read again. arAxisCtrl_gb[].Admin.DiagNbrRefreshTime can also be used to switch off reading of diagnostic information and to select the data source of the diagnostic information.

- DiagNbrRefreshTime = T#0ms: Composition of diagnostic number and diagnostic message switched off.
- DiagNbrRefreshTime between T#1ms and T#199ms (default T#100ms): The control parameters A-0-0023/A-0-0020 are always read and outputs as "Number" or Text "Message".
- DiagNbrRefreshTime greater than or equal to T#200ms: The AxisInterface reads in the corresponding, respective drive or control parameters depending on the axis type and message type.

IndraDrive with interpolation in the drive:

- Positive acknowledgements: drive parameters are read, e.g. A0012 Ready for operation

ML_TechInterface.library

- Error in the drive: drive parameters are read, e.g. F2026 No power
- Error on the control: control parameters are read, e.g. F2229233 Invalid master axis

IndraDrive with interpolation in the control.

- Positive acknowledgements A-0-0020/A-0-0023 are read. e.g. "Synchronized Motion"
- Error in the drive: drive parameters are read, e.g. F2026 No power
- Error on the control: control parameters are read, e.g. F2229233 Invalid master axis

SercosDrive and virtual axes:

- The control parameters are read.

Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	ML_ImcInterface.lib for IndraMotion MLC/MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD
	MB_AXIS_POS_STATUS structure
	MB_AXIS_POS_STATUS
Brief description	The MB_AXIS_POS_STATUS structure is a substructure of MB_AXIS_STATUS_TYPE01 and contains output status. The following illustration shows the current MB_AXIS_POS_STATUS structure:

```

TYPE MB_AXIS_POS_STATUS :
STRUCT
  (***** Outputs of the following axis - from motion control to logic control *****)
  (***** Achs-Ausgaenge - Motion Control zur Logic Control *****)

  Adjust: MB_AXIS_ADJUST_STATUS; (* -> outputs of the adjusting functionality for the position
    -> Ausgaenge fuer Tipfbetrieb der Position *)

END_STRUCT
END_TYPE

```

Fig.6-61: MB_AXIS_POS_STATUS structure

Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD
	MB_AXIS_ADJUST_STATUS
Brief description	The MB_AXIS_ADJUST_STATUS structure is a substructure of MB_AXIS_POS_STATUS and contains output status. The following illustration shows the current MB_AXIS_POS_STATUS structure:

```

TYPE MB_AXIS_ADJUST_STATUS :
STRUCT
  {***** Outputs of the following axis - motion control to logic control *****}
  {***** Achs-Ausgaenge - Motion Control zur Logic Control *****}

  ModeAck:    BOOL := FALSE; (* Ackn. enable adjust mode | Quittierung Freigabe Tippbetrieb *)

  HighLimitAck: BOOL := FALSE; (* max. value reached | max. Wert erreicht *)

  LowLimitAck:  BOOL := FALSE; (* min. value reached | min. Wert erreicht *)

END_STRUCT
END_TYPE

```

Fig.6-62: MB_AXIS_ADJUST_STATUS structure



The function blocks MB_ContinuousAdjustType01 and MB_IncrementalAdjustType01 from the library ML_TechBase are used internally to "Adjust".

Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD
	MB_AXIS_VEL_STATUS structure
	MB_AXIS_VEL_STATUS
Brief description	The MB_AXIS_VEL_STATUS structure is a substructure of MB_AXIS_STATUS_TYPE01 and contains output status. The following illustration shows the current MB_AXIS_VEL_STATUS structure:

```

TYPE MB_AXIS_VEL_STATUS :
STRUCT
  {***** Outputs of the following axis - from motion control to logic control *****}
  {***** Achs-Ausgaenge - Motion Control zur Logic Control *****}

  Adjust: MB_AXIS_ADJUST_STATUS; (* -> outputs of the adjusting functionality for the velocity
    -> Ausgaenge fuer Tippbetrieb der Geschwindigkeit *)

END_STRUCT
END_TYPE

```

Fig.6-63: MB_AXIS_VEL_STATUS structure

Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD

ML_TechInterface.library

MB_AXIS_SYNC_STATUS structure**MB_AXIS_SYNC_STATUS**

Brief description The MB_AXIS_SYNC_STATUS structure is a substructure of [MB_AXIS_STATUS_TYPE01](#) and contains output status.

The following illustration shows the current MB_AXIS_SYNC_STATUS structure:

```

TYPE MB_AXIS_SYNC_STATUS :
STRUCT
  (***** Outputs of the following axis - from motion control to logic control *****)
  (***** Achs-Ausgaenge - Motion Control zur Logic Control *****)

  CamActiveID:      MC_CAM_ID; (* - ID of the active CamTable | - Nr. aktive Stuetzpunkt-Tab.*)

  ProfileActiveSet: UINT:= 0;  (* - Number of the active motion profile set 0 or 1
    - Nummer des aktiven Satzes 0 oder 1 des Bewegungsprofils *)

  PhasingSlaveDone: BOOL := FALSE; (* - Change of PhaseOffset is finished
    - Verstellung des PhaseOffset ist abgeschlossen *)

  (***** status for adjusting of synchronisation mode values *****)
  (***** Statusmeldungen, Tippen im Synchronbetrieb *****)

  AdjFineAdjust:      MB_AXIS_ADJUST_STATUS; (* adjusting | vertippen
    arAxisCntrl[].SyncMode.FineAdjust *)

  AdjCamShaftDistance: MB_AXIS_ADJUST_STATUS; (* adjusting | vertippen
    arAxisCntrl[].SyncMode.CamShaftDistance *)

  AdjPhaseOffset:      MB_AXIS_ADJUST_STATUS; (* adjusting | vertippen
    arAxisCntrl[].SyncMode.PhaseOffset *)

END_STRUCT
END_TYPE

```

Fig.6-64: MB_AXIS_SYNC_STATUS structure

Library ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD

Dependence MX_ImcInterface.lib for IndraMotion MLD

System IndraMotion MLC/IndraMotion MLD

6.8.3 MB_AXIS_CONTROL_TYPE01 Structure

MB_AXIS_CONTROL_TYPE01

Brief description The MB_AXIS_CONTROL_TYPE01 data structure is accessed via the **arAxisCtr_gbl[]** instance and contains the following elements:

```

TYPE MB_AXIS_CONTROL_TYPE01 :
STRUCT
  Admin:      MB_AXIS_ADMINISTRATION; (* Administration of the axis | Verwaltung der Achse *)
  PosMode:    MB_AXIS_POSITIONING; (* operating mode positioning (absolute, rel. and additive)
    Positionierbetrieb (absolut, relativ und additiv) *)
  VelMode:    MB_AXIS_VELOCITY; (* operating mode velocity control | Geschwindigkeitsregelung *)
  SyncMode:   MB_AXIS_SYNCHRONISATION; (* operating mode synchronisation (phase, vel, and cam)
    Synchronbetriebsarten (Winkelsync., Geschwindigkeitssync. und Kurvenscheibe) *)
  (* add your extensions below here, keep the elements above
    Erweiterungen hier einfügen, Elemente oberhalb nicht veraendern *)
  (* user extensions | Anwender Erweiterungen *)

END_STRUCT
END_TYPE

```

Fig. 6-65: MB_AXIS_CONTROL_TYPE01 structure

Library ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib
for IndraMotion MLD

Dependence MX_ImcInterface.lib for IndraMotion MLD

System IndraMotion IndraMotion MLC/IndraMotion MLD

MB_AXIS_CONTROL_TYPE01 Elements

MB_AXIS_ADMINISTRATION structure

MB_AXIS_ADMINISTRATION

Brief description The MB_AXIS_ADMINISTRATION structure is a substructure of [MB_AXIS_CONTROL_TYPE01](#) and contains control variables and command values. The following illustration shows the current MB_AXIS_ADMINISTRATION structure:

ML_TechInterface.library

```

MB_AXIS_ADMINISTRATION
1  TYPE MB_AXIS_ADMINISTRATION :
2  STRUCT
3      (* selection of the operating mode | Betriebsarten-Anwahl
4          - level controlled
5          - set the corresponding variables for your operating mode before activating
6          - set ..._OpMode.en.en := ModeAB, ModeAH, ModePosAbs .... to activate an operating mode
7          - OR use bit variables ..._OpMode.en.b.MODE_AB, ..._OpMode.en.b.MODE_AH, _OpMode.b.MODE_POS_ABS. [1 lines]
8          -----
9          - Pegel-sensitiv
10         - vor Anwahl der Betriebsart die Eingangsdaten fuer die Betriebsart beschreiben
11         - Betriebsarten-Anwahl: ..._OpMode.en.en := ModeAB, ModeAH, ModePosAbs .... (ENUMs)
12         - oder ..._OpMode.en.b.MODE_AB, ..._OpMode.en.b.MODE_AH, OpMode.b.MODE_POS_ABS. (BIT-Variablen) [1 lines]
13         -----
14         _OpMode: MB_AXIS_MODE_UNION := ( en := ModeAb);
15     [1 lines]
16     Axis:  AXIS_REF := (AxisNo := NO_OBJECT, CntrlNo := LOCAL_CNTRL);
17
18     (* Refresh time for arAxisStatus[.].Diag.Number | Auffrischrate arAxisStatus[.].Diag.Number
19     - switched off, if smaller T#200ms | - ausgeschaltet wenn kleiner T#200ms *)
20     DiagMbrRefreshTime: TIME:=T#2s;
21
22     (* Deceleration when DRIVE_HALT is activated - always positive, [6 lines]
23     StopDeceleration: REAL:=99.0;
24
25     (* WaitTime restart of the MC Power in [mSec] | Wartezeit Neustart des MC_Power in [mSec] [1 lines]
26     WaitTimePowerRestart: UINT:=0;
27
28     (* clears axis errors with MC_Reset() | loescht Achsfehler mit MC_Reset() [1 lines]
29     ClearError:  BOOL:=FALSE;
30
31     (* Set absolute reference by command P-0-0012 | Absolutmass setzen mit Kommando P-0-0012 [1 lines]
32     SetAbsRef:  BOOL:=FALSE;
33
34     (* Enable the cyclic check and transmission of changed inputs for the actual operating mode [6 lines]
35     EnableCyclicScanning:  BOOL:=TRUE;
36
37     (* Enable the update on inputs, that are not cyclic checked, without leaving the active opmode [7 lines]
38     UpdateEveryInput:  BOOL:=FALSE;
39
40     (* TRUE: When the drive is in ModeInterrupted (e.g. drive internal command) the axis will be stopped
41     -----
42     TRUE: Im Falle von ModeInterrupted (z.B. internes Antriebskommando) wird die Achse angehalten *)
43     DriveInterruptReaction:  BOOL:=TRUE;
44
45     (* TRUE: MC_Power is internally activated when an _OpMode is selected
46     FALSE: the user application has to handle the MC_Power before an _OpMode is selected
47     -----
48     TRUE: der MC_Power wird intern aktiviert wenn _OpMode angewählt wird
49     FALSE: die Anwenderapplikation muß den MC_Power freigeben, bevor eine Betriebsart angewählt wird *)
50     PowerOn:  BOOL:=TRUE;
51
52  END_STRUCT
53  END_TYPE

```

Fig.6-66: MB_AXIS_ADMINISTRATION structure

Operation mode selection Operation modes can be selected using one of the following methods:

- **Selection of Enum values**

Assigning a value of type MB_AXIS_MODE to
arAxisCtrl_gb[.].Admin._OpMode

e.g. arAxisCtrl_gb[.].Admin._OpMode.en := ModePosAbs; or

arAxisCtrl_gb[.].Admin._OpMode.en := ModeAB;

- **Selection via bit access**

First delete "_OpMode" using `arAxisCtrl_gb[ ].Admin._OpMode := ModeAB;`, then use the bit access function to set only one bit.

e.g. `arAxisCtrl_gb[ ].Admin._OpMode.b.MODE_POS_ABS := TRUE;`



Setting several bits in `arAxisCtrl_gb[ ].Admin._OpMode` can cause an error that switches the drive to AB.

Diagnostic displays	The "DiagNbrRefreshTime" element is used to control the refresh rate of the <code>arAxisStatus_gb[].Diag.Number</code> . As soon as <code>arAxisStatus_gb[].Diag.Number</code> changes, <code>arAxisStatus_gb[].Diag.Message</code> is read again.
Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD
	MB_AXIS_ADMINISTRATION Elements
	MB_Axis_Mode
Brief description	This <code>MB_AXIS_Mode_UNION</code> is a union data type (UNION) that is used by the <code>_OpMode</code> element contained in the <code>MB_AXIS_ADMINISTRATION</code> structure. The UNION contains the enumeration type <code>MB_AXIS_MODE</code> and the bit structure <code>MB_AXIS_MODE_BITS</code> . The control of <code>_OpMode</code> is possible using both data types. Both data types are located at the same memory location. The following figure shows the current <code>MB_AXIS_MODE</code> data type:

ML_TechInterface.library

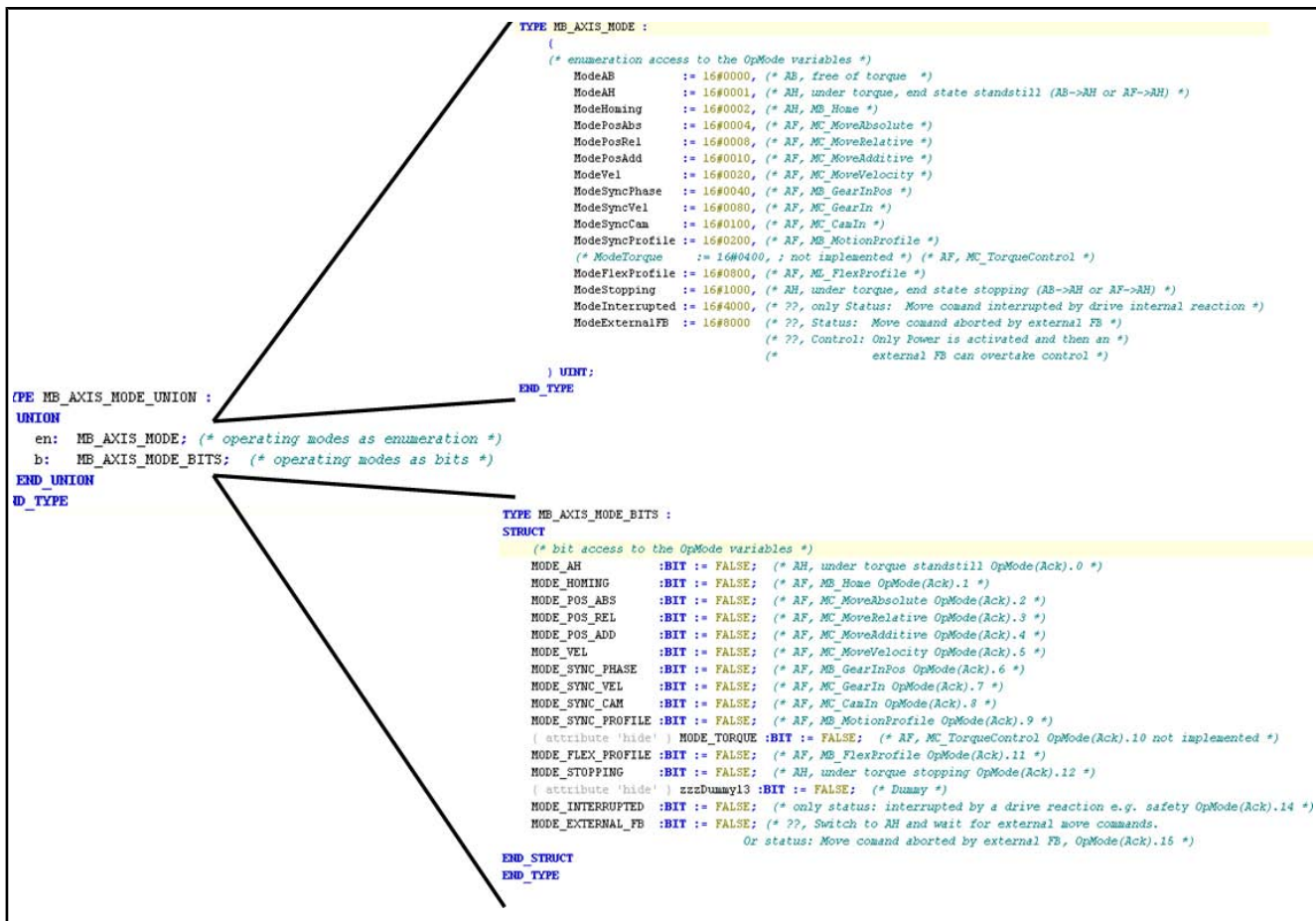


Fig.6-67: MB_AXIS_MODE data type

Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD
Axis_Ref	
Brief description	This Axis_Ref structure is part of the "RIL_CommonTypes" library and contains information on the respective axis. These elements call instances of the enumeration types CONTROLS and OBJECTS.

The following illustration shows the current structure:

```

(* Structure of PLCOpen element AXIS_REF. The AXIS_REF is a structure that contains information *)
(* on the corresponding axis. *)
(* It is used as a VAR_IN_OUT in all Motion Control Function Blocks. *)
TYPE AXIS_REF :
STRUCT
  CntrlNo:    CONTROLS := LOCAL_CNTRL;      (* control reference number *)
  AxisNo:    OBJECTS  := NO_OBJECT          (* object reference number *)
END_STRUCT
END_TYPE
  
```

Fig.6-68: Axis_Ref structure

MB_AXIS_POSITIONING structure

MB_AXIS_ADJUST_DATA

Brief description The structure MB_AXIS_ADJUST_DATA is a substructure of [MB_AXIS_POSITIONING](#) and contains binary setting commands and command values. The following illustration shows the current MB_AXIS_ADJUST_DATA structure:

```

TYPE MB_AXIS_ADJUST_DATA :
STRUCT
  {***** - binary adjust commands *****}
  {***** - Binaere Verstell-Eingaenge *****}

  Enable:          BOOL; (* Enable adjust mode | Freigabe Tippbetrieb
    - level controlled | - Pegel-sensitiv
    - set before activating: | - vor Freigabe folgende Groessen beschreiben:
      ModuloValue, HighLimit, LowLimit, StepWidth, LowStepWidth,
      StepsPerSecond, LowStepsPerSec *)

  Inc:             BOOL; (* Adjust + | Tippen +
    - level controlled when | - Pegel-sensitiv, wenn
      SelIncAdjust = FALSE (ML_ContinuousAdjustType01)
    - edge controlled when | - Flanken-sensitiv, wenn
      SelIncAdjust = TRUE (ML_IncrementalAdjustType01)
    - active, when Enable is TRUE | - aktiv, wenn Enable = TRUE *)

  Dec:             BOOL; (* Adjust - | Tippen -
    - level controlled when | - Pegel-sensitiv, wenn
      SelIncAdjust = FALSE (ML_ContinuousAdjustType01)
    - edge controlled when | - Flanken-sensitiv, wenn
      SelIncAdjust = TRUE (ML_IncrementalAdjustType01)
    - active, when Enable is TRUE | - aktiv, wenn Enable = TRUE *)

  SelIncAdjust:    BOOL; (* Incremental adjusting | Inkrementelle Verstellung
    - SelIncAdjust =
      TRUE: one step width is adjusted on every rising edge of Inc or Dec (edge triggered)
      FALSE: the adjusting is executed as long Inc or Dec are TRUE (level triggered)
    -----
    - SelIncAdjust =
      TRUE: Verstellung um 1 Schrittweite mit jeder steigenden Flanke an Inc oder Dec
      FALSE: kontinuierliche Verstellung, solange Inc oder Dec = TRUE *)

```

Fig.6-69: MB_AXIS_ADJUST_DATA structure (part 1 of 2)

ML_TechInterface.library

```

SellLowLevel:    BOOL; (* Reduced speed of changing | Reduzierte Verstellgeschwindigkeit
- SellLowLevel = TRUE means:
  step width = StepWidth * LowStepWidth% / 100% and
  speed of changing = StepsPerSecond * LowStepsPerSec% / 100%
-----
- SellLowLevel = TRUE bedeutet:
  Schrittweite = StepWidth * LowStepWidth% / 100% und
  Verstellgeschwindigkeit = StepsPerSecond * LowStepsPerSec% / 100% *)

(***** - setpoints of adjust | - Verstell-Variablen *****)

AdjModuloValue:  REAL:=0.0; (* ModuloValue for the adjusting | Modulo Wert fuer das Vertippen
- choose zero for absolute scaling | - bei Absolut-Wichtung auf 0 setzen
- read from Parameter by ML_AxisInitType01() *)

HighLimit:       REAL; (* Upper limit of the adj. value | Max. Wert vertippte Zielgroesse *)

LowLimit:        REAL; (* Lower limit of the adj. value | Min. Wert vertippte Zielgroesse *)

StepWidth:       REAL; (* StepWidth | Schrittweite
- s.a.: used to calculate the changing vel. | - s.o.: Berechnung Verstellgeschwindigkeit *)

LowStepWidth:    INT; (* Reduced StepWidth in percent | Reduzierte Schrittweite in %
- see above: used to calculate the changing velocity
  SellLowLevel = TRUE: step width = StepWidth * LowStepWidth% / 100%
-----
- siehe oben: Berechnung der Verstellgeschwindigkeit
  SellLowLevel = TRUE: Schrittweite = StepWidth * LowStepWidth% / 100% *)

StepsPerSecond:  REAL; (* Adjust steps per second | Verstell-Schritte pro Sekunde
- s.a.: used to calculate the changing vel. | - s.o.: Berechnung Verstellgeschwindigkeit *)

LowStepsPerSec:  INT;  (* Reduced changing speed in % | Reduzierte Verstellgeschw. in %
- see above: used to calculate the speed of changing
  when SellLowLevel = TRUE: speed of changing = StepsPerSecond * LowStepsPerSec% / 100%
-----
- siehe oben: Berechnung der Verstellgeschwindigkeit
  wenn SellLowLevel = TRUE: Verstellgeschw. = StepsPerSecond * LowStepsPerSec% / 100% *)

END_STRUCT
END_TYPE

```

Fig.6-70: MB_AXIS_ADJUST_DATA structure (part 2 of 2)



The function blocks MB_ContinuousAdjustType01 and MB_IncrementalAdjustType01 from the library ML_TechBase are used internally to "Adjust".

Library	ML_AxisInterface.lib for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD
MB_AXIS_POSITIONING	
Brief description	The MB_AXIS_POSITIONING structure is a substructure of MB_AXIS_CONTROL_TYPE01 , containing command values for positioning mode. The following illustration shows the current MB_AXIS_POSITIONING structure:

```

TYPE MB_AXIS_POSITIONING :
STRUCT
  (***** - setpoints of positioning mode *****)
  (***** - Positions-Sollwerte *****)

  Position:          REAL;
  (* position for an absolute move (in technical unit[u]) (negative or positive)
  - start moving with "ModePosAbs" (MC_MoveAbsolute)
  Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
  -----
  Zielposition, absolut (Einheit: [u]) (negativ oder positiv)
  - Freigabe mit "ModePosAbs" (MC_MoveAbsolute)
  Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

  Distance:          REAL;
  (* distance for a relative or additive move (in technical unit[u]) (negative or positive)
  - start moving with "ModePosRel" or "ModePosAdd" (MC_MoveRelative, MC_MoveAdditive)
  Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
  -----
  Relativer / additiver Sollwert (Einheit: [u]) (negativ oder positiv)
  - Freigabe mit "ModePosRel" oder "ModePosAdd" (MC_MoveRelative, MC_MoveAdditive)
  Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

  Velocity:          REAL:=10.0;
  (* maximum velocity (always positive) (not necessarily reached) [u/s]
  - start moving with "ModePosAbs" or "ModePosRel" or "ModePosAdd"
    (MC_MoveAbsolute, MC_MoveRelative, MC_MoveAdditive)
  Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
  -----
  Positioniergeschwindigkeit, (Einheit: [u/s]) (positiv eingeben)
  - Freigabe mit "ModePosAbs" oder "ModePosRel" oder "ModePosAdd"
    (MC_MoveAbsolute, MC_MoveRelative, MC_MoveAdditive)
  Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

  Acceleration:      REAL:=10.0;
  (* Value of the acceleration (always positive) (increasing energy of the motor) [u/s2]
  - start moving with "ModePosAbs" or "ModePosRel" or "ModePosAdd"
    (MC_MoveAbsolute, MC_MoveRelative, MC_MoveAdditive)
  -----
  Beschleunigung (Einheit: [u/s2]) (nur positive Werte)
  - Freigabe mit "ModePosAbs" oder "ModePosRel" oder "ModePosAdd"
    (MC_MoveAbsolute, MC_MoveRelative, MC_MoveAdditive) *)

  Deceleration:      REAL:=10.0;
  (* Value of the deceleration (always positive) (decreasing energy of the motor) [u/s2]
  - start moving with ModePosAbs or ModePosRel or ModePosAdd
    (MC_MoveAbsolute, MC_MoveRelative, MC_MoveAdditive)
  -----
  Verzögerung, (Einheit: [u/s2]) (nur positive Werte)
  - Freigabe mit "ModePosAbs" oder "ModePosRel" oder "ModePosAdd"
    (MC_MoveAbsolute, MC_MoveRelative, MC_MoveAdditive) *)

  Adjust:            MB_AXIS_ADJUST_DATA;
  (* -> adjusting of the element Position
  - only possible, when absolute positioning mode is active
  -----
  -> Manuelle Verstellung (Vertippen) der Position
  - wirksam bei freigegebenem Positionierbetrieb (absolut) *)

END_STRUCT
END_TYPE

```

Fig.6-71: MB_AXIS_POSITIONING structure

ML_TechInterface.library

Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD
	MB_AXIS_VELOCITY structure
	MB_AXIS_VELOCITY
Brief description	The MB_AXIS_VELOCITY structure is a substructure of MB_AXIS_CONTROL_TYPE01 and contains command values for velocity mode. The following illustration shows the current MB_AXIS_VELOCITY structure:

MB_AXIS_VELOCITY	
1	(***** - setpoints of velocity mode *****)
2	(***** - Sollwerte Geschwindigkeitsregelung (Freilauf) *****)
3	TYPE MB_AXIS_VELOCITY :
4	STRUCT
5	
6	(* Value of the maximum velocity (Unit: e.g. in [rpm] depending on parametrization)
7	- start moving with "ModeVel" (MC_MoveVelocity)
8	Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
9	-----
10	Geschwindigkeits-Sollwert, (Einheit: z.B. in [U/min] je nach Parametrierung)
11	- Freigabe mit "ModeVel" (MC_MoveVelocity)
12	Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)
13	Velocity: REAL:=0.0;
14	
15	(* Value of the acceleration (Unit: e.g. in [rad/s ²] depending on
16	parametrization) (always positive)
17	- start moving with "ModeVel" (MC_MoveVelocity)
18	Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
19	-----
20	Beschleunigung, (Einheit: z.B. in [rad/s ²] je nach Parametrierung) (immer positiv)
21	- Freigabe mit "ModeVel" (MC_MoveVelocity)
22	Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)
23	Acceleration: REAL:=10.0;
24	
25	(* Value of the deceleration (Unit: e.g. in [rad/s ²] depending on
26	parametrization) (always positive)
27	- start moving with "ModeVel" (MC_MoveVelocity)
28	Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
29	-----
30	Verzögerung, (Einheit: z.B. in [rad/s ²] je nach Parametrierung) (immer positiv)
31	- Freigabe mit "ModeVel" (MC_MoveVelocity)
32	Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)
33	Deceleration: REAL:=10.0;
34	
35	(* -> adjusting of the element Velocity
36	-> Manuelle Verstellung (Vertippen) der Geschwindigkeit *)
37	Adjust: MB_AXIS_ADJUST_DATA;
38	
39	END_STRUCT
40	END_TYPE

Fig. 6-72: MB_AXIS_VELOCITY structure

Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion IndraMotion MLC/IndraMotion MLD

ML_TechInterface.library

MB_AXIS_VELOCITY Elements

MB_AXIS_ADJUST_DATA

Brief description The structure MB_AXIS_ADJUST_DATA is a substructure of [MB_AXIS_POSITIONING](#) and contains binary setting commands and command values (see also [chapter "MB_AXIS_ADJUST_DATA" on page 105](#)).

MB_AXIS_SYNCHRONISATION Structure

MB_AXIS_SYNCHRONISATION

Brief description The MB_Axis_Synchronisation structure is a substructure of [MB_AXIS_CONTROL_TYPE01](#) and contains set points and command values and settings for synchronisation mode. The following figure shows the current MB_AXIS_SYNCHRONISATION structure:

```

TYPE MB_AXIS_SYNCHRONISATION :
STRUCT
  (***** - setpoint of synchronous motion *****)

  InputRevolution: UINT:=1;
  (* Electronic gear input revolution
   - start moving with "ModeSyncPhase", "ModeSyncVel", "ModeSyncCam" or "ModeSyncProfile"
   - RatioNumerator for MC_GearIn, MB_GearInPos, MC_CamIn, MB_MotionProfile
   Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
   -----
   Elektronisches Getriebe, Eingangsumdrehungen
   - Freigabe mit "ModeSyncPhase", "ModeSyncVel", "ModeSyncCam" oder "ModeSyncProfile"
   - RatioNumerator fuer MC_GearIn, MB_GearInPos, MC_CamIn, MB_MotionProfile
   Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

  OutputRevolution: UINT:=1;
  (* Electronic gear output revolution
   - start moving with "ModeSyncPhase", "ModeSyncVel", "ModeSyncCam" or "ModeSyncProfile"
   - RatioDenominator for MC_GearIn, MB_GearInPos, MC_CamIn, MB_MotionProfile
   Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
   -----
   Elektronisches Getriebe, Ausgangsumdrehungen
   - Freigabe mit "ModeSyncPhase", "ModeSyncVel", "ModeSyncCam" oder "ModeSyncProfile"
   - RatioDenominator fuer MC_GearIn, MB_GearInPos, MC_CamIn
   Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

  FineAdjust: REAL:=0.0;
  (* Master fine adjust
   - start moving with "ModeSyncPhase", "ModeSyncVel", "ModeSyncCam" or "ModeSyncProfile"
   - MasterFineadjust for MC_GearIn, MB_GearInPos, MC_CamIn, MB_MotionProfile
   Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
   -----
   Getriebe-Feinabgleich
   - Freigabe mit "ModeSyncPhase", "ModeSyncVel", "ModeSyncCam" oder "ModeSyncProfile"
   - MasterFineadjust fuer MC_GearIn, MB_GearInPos, MC_CamIn
   Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

  CamShaftDistance: REAL:=360.0;
  (* Cam Shaft Distance
   - start moving with "ModeSyncCam"
   - CamShaftDist for MC_CamIn
   Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
   -----
   Kurvenscheibe, Hub
   - Freigabe mit "ModeSyncCam"
   - CamShaftDist fuer MC_CamIn
   Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)
  
```

Fig.6-73: MB_AXIS_SYNCHRONISATION structure (part 1 of 3)

ML_TechInterface.library

```

PhaseOffset:    REAL:=0.0; (* Phase offset between master and slave axis
- start moving with "ModeSyncPhase", "ModeSyncCam" or "ModeSyncProfile"
- PhaseShift for MB_PhasingSlave
Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
-----
Winkelversatz (Lagesollwert additiv)
- Freigabe mit "ModeSyncPhase", "ModeSyncCam" oder "ModeSyncProfile"
- PhaseShift fuer MB_PhasingSlave
Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

PhaseOffsetVel: REAL:=1.0;
(* Velocity for the phase offset
- start moving with "ModeSyncPhase", "ModeSyncCam" or "ModeSyncProfile"
- Velocity for MB_PhasingSlave
-----
Aufsynchronisier-Geschwindigkeit
- Freigabe mit "ModeSyncPhase", "ModeSyncCam" oder "ModeSyncProfile"
- Velocity fuer MB_PhasingSlave *)

PhaseOffsetAcc: REAL:=100.0;
(* Acceleration for the phase offset
- start moving with "ModeSyncPhase", "ModeSyncCam" or "ModeSyncProfile"
- PhaseShift for MB_PhasingSlave
-----
Aufsynchronisier-Beschleunigung
- Freigabe mit "ModeSyncPhase", "ModeSyncCam" oder "ModeSyncProfile"
- PhaseShift fuer MB_PhasingSlave *)

SyncDirection: MC_SYNC_DIRECTION:= SHORTEST_WAY;
(* Synchronisation direction
- values 'SHORTEST_WAY'; 'CATCH_UP' (always forward); 'SLOW_DOWN' (see RIL_CommonTypes.lib)
- start moving with "ModeSyncPhase", "ModeSyncCam" or "ModeSyncProfile"
- SyncMode for MB_GearInPos, MC_CamIn, MB_MotionProfile
-----
Aufsynchronisier-Richtung
- Werte 'SHORTEST_WAY'; 'CATCH_UP' (positiv); 'SLOW_DOWN' (siehe RIL_CommonTypes.lib)
- Freigabe mit "ModeSyncPhase", "ModeSyncCam" oder "ModeSyncProfile"
- SyncMode fuer MB_GearInPos, MC_CamIn, MB_MotionProfile *)

StartMode:      MC_START_MODE:= ABSOLUTE;
(* Start mode of the synchronisation
- values 'ABSOLUTE'; 'RELATIVE' (see RIL_CommonTypes.lib)
- start moving with "ModeSyncPhase", "ModeSyncCam" or "ModeSyncProfile"
- StartMode for MB_GearInPos, MC_CamIn, MB_MotionProfile
-----
Aufsynchronisier-Mode
- Werte: 'ABSOLUTE'; 'RELATIVE' (siehe RIL_CommonTypes.lib)
- Freigabe mit "ModeSyncPhase", "ModeSyncCam" oder "ModeSyncProfile"
- StartMode fuer MB_GearInPos, MC_CamIn, MB_MotionProfile *)

```

Fig.6-74: MB_AXIS_SYNCHRONISATION structure (part 2 of 3)

```

CamTableID:    MC_CAM_ID:= CAM_TABLE_1;
(* Number of the selected cam table
- values 'CAM_TABLE_1'; 'CAM_TABLE_2' ... 'CAM_TABLE_8' (see RIL_CommonTypes.lib)
- start moving with "ModeSyncCam"
- CamTableID for MC_CamIn
Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
-----

Nummer Stuetzpunkt-Tabelle
- Werte: 'CAM_TABLE_1'; 'CAM_TABLE_2' ... 'CAM_TABLE_8' (siehe RIL_CommonTypes.lib)
- Freigabe mit "ModeSyncCam"
- CamTableID fuer MC_CamIn
Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

Master:    AXIS_REF    := (AxisNo := NO_OBJECT, CntrlNo := LOCAL_CNTRL);
(* -> AXIS_REF of the master axis
- "ModeSyncPhase", "ModeSyncVel", "ModeSyncCam", "ModeSyncProfile", "ModeFlexProfile"
- Master for MC_GearIn, MB_GearInPos, MC_CamIn, MB_MotionProfile, ML_FlexProfile
Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
-----
-> AXIS_REF der Leitachse
- "ModeSyncPhase", "ModeSyncVel", "ModeSyncCam", "ModeSyncProfile", "ModeFlexProfile"
- Master fuer MC_GearIn, MB_GearInPos, MC_CamIn, MB_MotionProfile, ML_FlexProfile
Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

(***** Extensions for profile operation modes *****)
(***** Erweiterungen für die Profil Betriebsarten *****)
Profile:    ML_AXIS_PROFILE;
(* Extensions for the operating modes MotionProfile and FlexProfile
-----
Erweiterungen für die Betriebsarten MotionProfile und FlexProfile *)

(***** adjusting of synchronisation mode values *****)
(***** Vertippen im Synchronbetrieb *****)

AdjFineAdjust:    MB_AXIS_ADJUST_DATA; (* adjusting | vertippen
arAxisCntrl[].SyncMode.FineAdjust *)

AdjCamShaftDistance:    MB_AXIS_ADJUST_DATA; (* adjusting | vertippen
arAxisCntrl[].SyncMode.CamShaftDistance *)

AdjPhaseOffset:    MB_AXIS_ADJUST_DATA; (* adjusting | vertippen
arAxisCntrl[].SyncMode.PhaseOffset *)

END_STRUCT
END_TYPE

```

Fig.6-75: MB_AXIS_SYNCHRONISATION structure (part 3 of 3)

Library	ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib for IndraMotion MLD
Dependence	MX_ImcInterface.lib for IndraMotion MLD
System	IndraMotion MLC/IndraMotion MLD
	MB_AXIS_SYNCHRONISATION Elements
	ML_AXIS_PROFILE
Brief description	The MB_AXIS_PROFILE structure is a substructure of chapter "MB_AXIS_SYNCHRONISATION" on page 110 and contains command values and

ML_TechInterface.library

settings for synchronisation mode. The following figure shows the current ML_AXIS_PROFILE structure:

```

TYPE ML_AXIS_PROFILE :
STRUCT
  (* Extensions for the operating modes MotionProfile and FlexProfile
  -----
  Erweiterungen für die Betriebsarten MotionProfile und FlexProfile *)

  SetSelection:  UINT:= 0;
  (* Number of the selected motion profile set 0 or 1
  - SetSelection for MB_MotionProfile, ML_FlexProfile
  Note: cyclic checked AND transmitted on change (EnableCyclicScanning = TRUE)
  -----
  Auswahl des Satzes 0 oder 1 des Bewegungsprofils
  - SetSelection fuer MB_MotionProfile, ML_FlexProfile
  Hinweis: wird permanent zyklisch verarbeitet (EnableCyclicScanning = TRUE) *)

  RelativePositioning:  BOOL:= FALSE;
  (* position data processing
  - FALSE: absolute, TRUE: relative
  - RelativePositioning for MB_MotionProfile
  -----
  Verarbeitung der Positionierdaten
  - FALSE: absolut, TRUE: relativ
  - RelativePositioning fuer MB_MotionProfile *)

  (* All following: Extensions for the operating mode FlexProfile
  - all inputs have the same name as the inputs of ML_FlexProfile
  -----
  Erweiterungen für die Betriebsart FlexProfile
  - alle Eingängen haben die gleichen Namen wie die Eingänge von ML_FlexProfile *)

  UseSwitchingPosition:  BOOL:=FALSE;
  (* TRUE: the switching to a different profile is done, when the master axis reaches
  the position of the input "SwitchingPosition".
  -----
  TRUE: Umschaltung auf ein anderes Profil erst wenn die Leitachse die am Eingang
  "SwitchingPosition" angelegte Position erreicht. *)

  SwitchingPosition:  REAL:=0.0;
  (* SwitchingPosition, where the switching to a different profile should be done.
  -----
  Umschaltposition bzw. Anfangsposition, an der die Umschaltung auf ein anderes
  Profil bzw. der Profilstart erfolgen soll. *)

```

Fig.6-76: ML_AXIS_PROFILE, part 1

```

(* Describes the interpretation of the profile beginning in case of a profile
   start or change of a profile
   - Values: 'SLAVE_ORIGIN_MASTER_ORIGIN'; 'SLAVE_CURRENT_MASTER_CURRENT',
             'SLAVE_ORIGIN_MASTER_CURRENT', 'SLAVE_CURRENT_MASTER_ORIGIN' (see ML_PLCOpen.lib)
   -----
   Beschreibt wie bei einem Profilstart oder Profilwechsel der Ursprung
   des Profils interpretiert werden soll
   - Werte: 'SLAVE_ORIGIN_MASTER_ORIGIN'; 'SLAVE_CURRENT_MASTER_CURRENT',
            'SLAVE_ORIGIN_MASTER_CURRENT', 'SLAVE_CURRENT_MASTER_ORIGIN' (siehe ML_PLCOpen.lib) *)
ProfileEntry:      MB_PROFILE_START_POINT := SLAVE_ORIGIN_MASTER_ORIGIN;

(* Offset of the master axis | Offset der Leitachse *)
MasterOffset: REAL := 0.0;

(* Offset of the following axis | Offset der Folgeachse *)
SlaveOffset: REAL := 0.0;

(* Synchronisation type
   - Values: 'SYNC_RAMPIN_SHORTEST_WAY'; 'SYNC_RAMPIN_CATCH_UP',
             'SYNC_RAMPIN_SLOW_DOWN', 'SYNC_DIRECT' (see ML_PLCOpen.lib)
   -----
   Aufsynchronisier-Art
   - Werte: 'SYNC_RAMPIN_SHORTEST_WAY'; 'SYNC_RAMPIN_CATCH_UP',
            'SYNC_RAMPIN_SLOW_DOWN', 'SYNC_DIRECT' (siehe ML_PLCOpen.lib) *)
SyncType:      MB_SYNC_TYPE := SYNC_RAMPIN_SHORTEST_WAY;

(* Execution type
   - Values: 'EXECUTE_CYCLIC'; 'EXECUTE_SINGLE' (see ML_PLCOpen.lib)
   -----
   Abarbeitungstyp
   - Werte: 'EXECUTE_CYCLIC'; 'EXECUTE_SINGLE' (siehe ML_PLCOpen.lib) *)
ExecutionMode: MB_EXECUTION_MODE := EXECUTE_CYCLIC;

END_STRUCT
END_TYPE

```

Fig.6-77: ML_AXIS_PROFILE, part 2

Library ML_TechInterface.compiled-library for IndraMotion MLC/MX_AxisInterface.lib
for IndraMotion MLD

Dependence MX_ImcInterface.lib for IndraMotion MLD

System IndraMotion MLC/IndraMotion MLD

6.9 IMC Interface Structures

6.9.1 Overview

The ML_ImcInterface libraries for IndraMotion MLC controls and the MX_ImcInterface for IndraMotion MLD-M drives provide an easy to use interface for every system. The interface is called an IMC interface - (I)ndra (M)otion (C)ontrol interface. The IMC interface provides the following functions when it is used as an interface for the IndraMotion MLC or IndraMotion MLD-D system:

- Selects MLC (IndraMotion MLD-M) operation mode, P0 (Phase 0), P2 (Parameter Mode), BB (Operation Mode).

ML_TechInterface.library

- Executes the Clear All Errors command.
- Executes RetriggerOpMode to allow users to re-execute the selected operation mode. The user can reset the control for the IMC interface operation mode if the SERCOS ring is disconnected.
- In case of phase switching via an external instance (e.g. IndraWorks), the IMC interface sets the output "PassiveMode" and the selected mode is not actuated. Use "RetriggerOpMode" or switch back into the selected mode to exit the "PassiveMode" again.



The library names used in this chapter depend on the respective system.

IndraMotion MLC control systems use the library ML_TechInterface.compiled-library.

IndraMotion MLD-M drive systems use the library MX_ImcInterface.lib.

The following table lists the user's interface to the control interface data structure:

User interface	Type	Description
ImcCtrl[]	MB_Imc_Control_Type01	Control structure that provides the inputs for operation mode, clearing errors and "retriggering the operation mode".
ImcStatus[]	MB_Imc_Status_Type01	Status structure that includes operation mode acknowledgements and diagnostic information.

Fig.6-78: Control interface - user interface data structures

In order to use the IMC interface, only the program "MB_ImcInterface" must be called cyclically, e.g. in the PlcTask. When using the GAT or. GAT ^{compact} the program call is already integrated.

6.9.2 MB_IMC_Status_Type01

Brief description The MB_IMC_Status_Type01 data structure is accessed using the instance **ImcStatus[]**. It contains the following elements:

```

TYPE MB_IMC_STATUS_TYPE01 :
  STRUCT
    InOperation: BOOL := FALSE;  (* TRUE: ImcInterface is operating *)

    Admin: MB_IMC_ADMIN_STATUS; (* -> Status of the administration of the control *)

    Diag: MB_IMC_DIAGNOSIS;      (* -> Control diagnosis information, error, warning..*)

    PassiveMode: BOOL := FALSE;  (* TRUE: External source controlling operation mode *)

  END_STRUCT
END_TYPE

```

Fig.6-79: MB_IMC_Status_Type01 structure

Library ML_TechInterface.compiled-library for IndraMotion MLC/MX_ImcInterface.lib for IndraMotion MLD-M

Dependence MX_AxisInterface.lib for IndraMotion MLD-M

System IndraMotion MLC/IndraMotion MLD-M

6.9.3 MB_IMC_Control_Type01

Brief description The MB_IMC_Control_Type01 data structure is accessed using the instance `ImcCtrl[]`. It contains the following elements:

```

TYPE MB_IMC_CONTROL_TYPE01 :
  STRUCT

    Admin:      MB_IMC_ADMINISTRATION;  (* -> Administration of the control *)

    UserDiag: MB_IMC_USER_DIAGNOSTIC;  (* -> User controlled Diagnostics *)

  END_STRUCT
END_TYPE
  
```

Fig. 6-80: MB_IMC_Control_Type01 structure

Library ML_TechInterface.compiled-library for IndraMotion MLC/MX_ImcInterface.lib for IndraMotion MLD-M

Dependence MX_AxisInterface.lib for IndraMotion MLD-M

System IndraMotion MLC/IndraMotion MLD-M

6.10 Special Features of the IndraMotion MLD

6.10.1 Introduction

This chapter describes the special features that must be noted when using the AxisInterface in the IndraMotion MLD system.

The AxisInterface cannot be used in all firmware variants:

- MPC: application as IndraMotion MLD-M with up to 8 real axes and one virtual axis
- MPH: application as IndraMotion MLD-S with one real and one virtual axis
- MPB: this variant is not supported because the AxisData structure required for the AxisInterface is not possible with MPB.
- MPD: this variant is not supported because a drive-integrated PLC is not possible.

To use the AxisInterface, the additive function package "IndraMotion MLD with technology functions" must be enabled. The function package "extended technology function blocks" is not required.



This documentation describes the integration of the AxisInterface when using IndraLogic 2G and GAT^{compact} as an example application. The integration of the AxisInterface when using IndraLogic 1G as used with IndraMotion MLD is described in the functional description "Rexroth IndraMotion MLC04/MPx06 AxisInterface".

Differences from IndraMotion MLC

The AxisInterface and IMC interface are provided for the IndraMotion MLD in two separate libraries:

- Control interface: ML_ImcInterface.lib
- AxisInterface: ML_AxisInterface.lib

ML_TechInterface.library

The AxisInterface library is implemented as a firmware library with the IndraMotion MLC. For this reason, the library that corresponds with the firmware release must be used. Otherwise, during the PLC program download this error message appears: "The following functions could not be connected: Version_MB_AxisInterface_07Vxx".

The following functionalities are not implemented in the IndraMotion MLC execution of the AxisInterface and control interface:

- AxisInterface: arAxisCtrl_gb[].Admin.UpdateEveryInput input
- AxisInterface: ModeStopping operation mode
- AxisInterface: arAxisStatus_gb[].Admin.AbsRefCmdStatus output
- IMC Interface: ImcStatus.PassiveMode output

6.10.2 Parameterization

In order to use the AxisInterface on an IndraMotion MLC, the following parameterization must be implemented:

- The PLC controls the drive permanently: P-0-1367 "PLC configuration", bit 4 set
- AxisData structure support: P-0-1367 "PLC configuration", bit 6 set

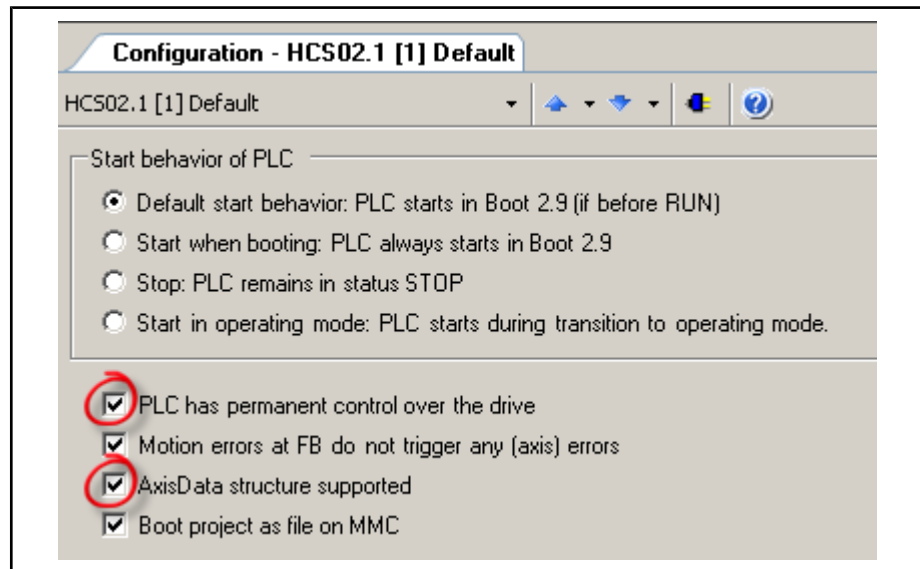


Fig.6-81: IndraMotion MLC parameterization for AxisInterface

An MMC (multi-media card) must be present for the boot project.

In order to use the master axis generator as a virtual axis for the AxisInterface, the following settings must be made.

- Master axis generator activation: P-0-0917 "Control word of the master axis generator", bit 0 set
- Master axis composition: P-0-0916, "Master axis format convert signal selection", P-0-758 entered
- Here, as an example, are the settings for a rotary virtual axis that can rotate in both directions:
 - Rotary module scaling: P-0-0756 "Virtual master axis, scaling type", Bit 2/1/0 = 010, bit 7 set
 - Shortest distance: P-0-0769 "Virtual master axis, command value mode", bit 1/0 = 10

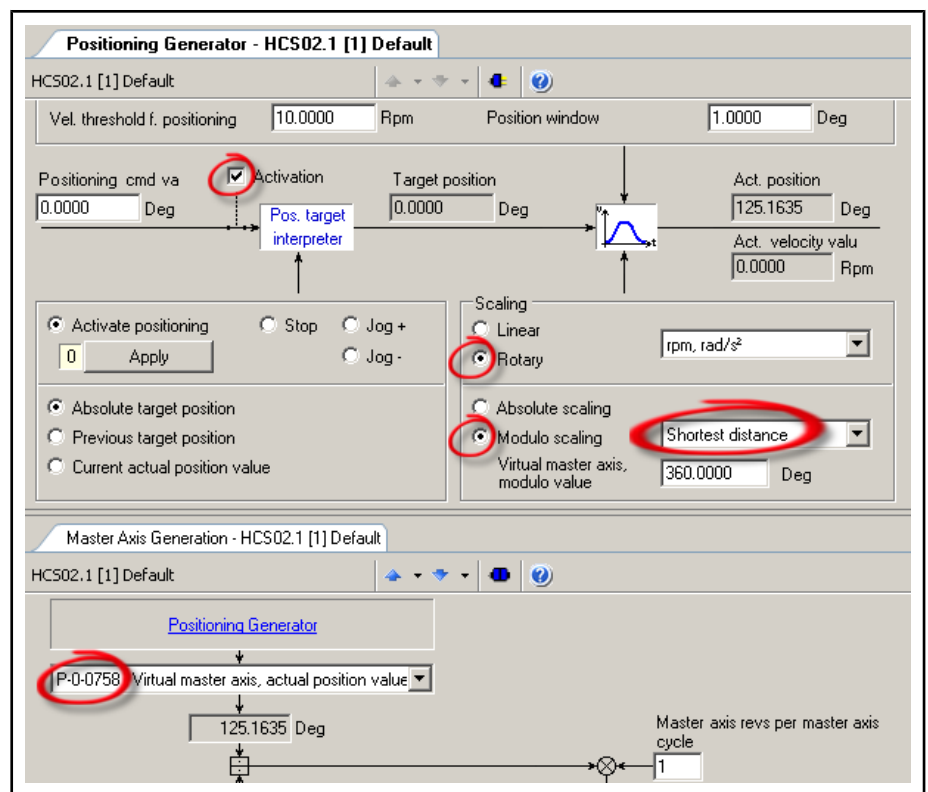


Fig. 6-82: IndraMotion MLD parameterization of master axis generator as virtual axis



If the master axis generator is not parameterized as described above, the example project must be adapted (function block ML_AxisInitAllAxisType01 or TE_AxisInitAllAxisType01). Then the function block MB_AxisInitType01() may not be called for the virtual axis.

6.10.3 PLC Programming

The AxisInterface structures are defined for 11 axes:

- 1 to 8: real axes
 - 9 and 10: placeholders and cannot be used in the IndraMotion MLD system mode.
 - 11: virtual axis (=master axis generator) is addressed via the VMA_INT_IDX Define.
- Notes regarding the virtual axis**
- Access to the structures in the AxisInterface with the VMA_INT_IDX (Index 11) Define.
 - The function MB_GetAxisInterfaceIndex() can be used to determine the index on the IndraMotion MLC and IndraMotion MLD regardless of system.
 - The AxisInterface simulates a part of the information provided in the AxisData structure for real axes for the virtual axis. A pointer in the AxisStatus structure can be used to access this information.

Note: This pointer is initialized from the AxisInterface Init function block. The pointer and the information it points to can only be accessed after the initialization is successfully executed.

ML_TechInterface.library

Information available for the virtual axis:

- arAxisStatus_gb[VMA_INT_IDX].PtrAxisData^.rActualPosition_i;
- arAxisStatus_gb[VMA_INT_IDX].PtrAxisData^.rActualVelocity_i;
- arAxisStatus_gb[VMA_INT_IDX].PtrAxisData^.Axis_CmdValueReached;
- arAxisStatus_gb[VMA_INT_IDX].PtrAxisData^.Axis_Standstill;

In operation mode, the following information is always set to TRUE:

- arAxisStatus_gb[VMA_INT_IDX].PtrAxisData^.Axis_Inbb;
- arAxisStatus_gb[VMA_INT_IDX].PtrAxisData^.Axis_InAb;
- arAxisStatus_gb[VMA_INT_IDX].PtrAxisData^.Axis_Power;
- arAxisStatus_gb[VMA_INT_IDX].PtrAxisData^.Axis_Homed;



If the AxisInterface is to be used on an IndraMotion MLD-S (MPH firmware), the example project must be adapted (function block ML_AxisInitAllAxisType01 or TE_AxisInitAllAxisType01). Reading of parameters not present with MPH is not allowed. The function block MB_AxisInitType01() may only be called for the real axis 1 and for the virtual axis.

7 GATcompact

7.1 Overview of GATcompact

General information The ^{compact} generic application template is an example project based on IEC 61131-3. It is made available by the IndraWorks engineering framework for IndraLogic XLC and IndraMotion MLC and a wizard can be used for loading and start up. It also supports the specific features of Motion applications such as the handling of axes.

Basic idea Using GAT^{compact} is based on the idea that a simple machine must have operation modes "Auto" and "Manual". In addition, axis handling is demonstrated based on examples.

The following figure shows a schematic of the GAT^{compact} architecture.

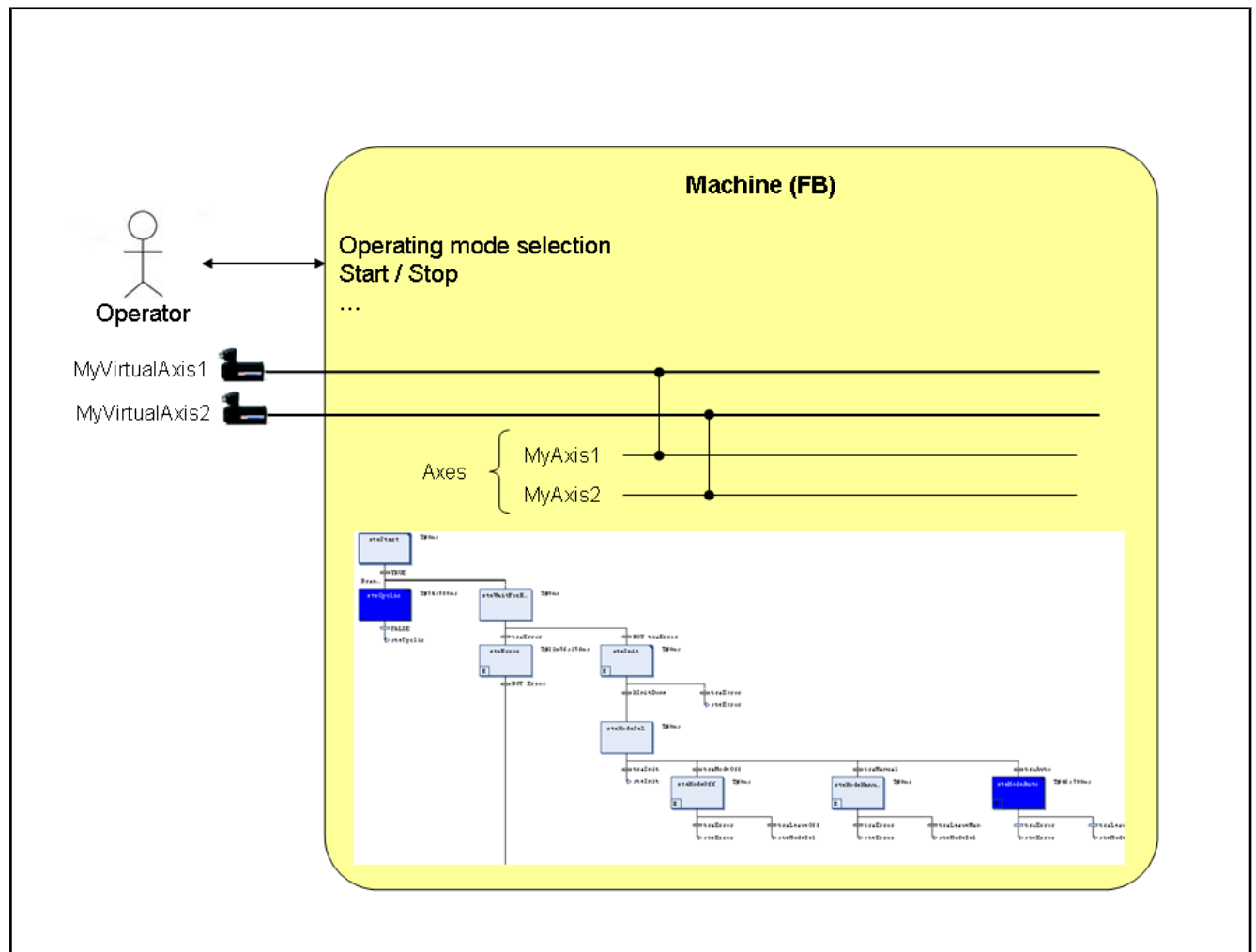


Fig.7-1: Basis for GAT^{compact}

Requirements for use The user must know which activities the machine to be implemented is to perform during Automatic mode and which during Manual mode.

GATcompact

7.2 First Steps - GAT^{compact}

7.2.1 Preparation

GAT^{compact} is integrated into the IndraWorks Suite installation package and is automatically available upon installation.

GAT^{compact} can be used if a project was created with an IndraLogic XLC-/IndraMotion MLC control.

The GAT wizard, which can be used to load the GAT^{compact} is called using the

GAT button  in the IndraWorks toolbar.

7.2.2 Creating a Simple Motion Application

General Information

This section describes how to create an application based on GAT^{compact}.

Topic

In the Automatic operation of a machine, a "master" axis is to move continuously at 100 rev./min. A second "slave" axis is to operate phase synchronously to the first axis. The following figure provides a schematic representation of the tasks.

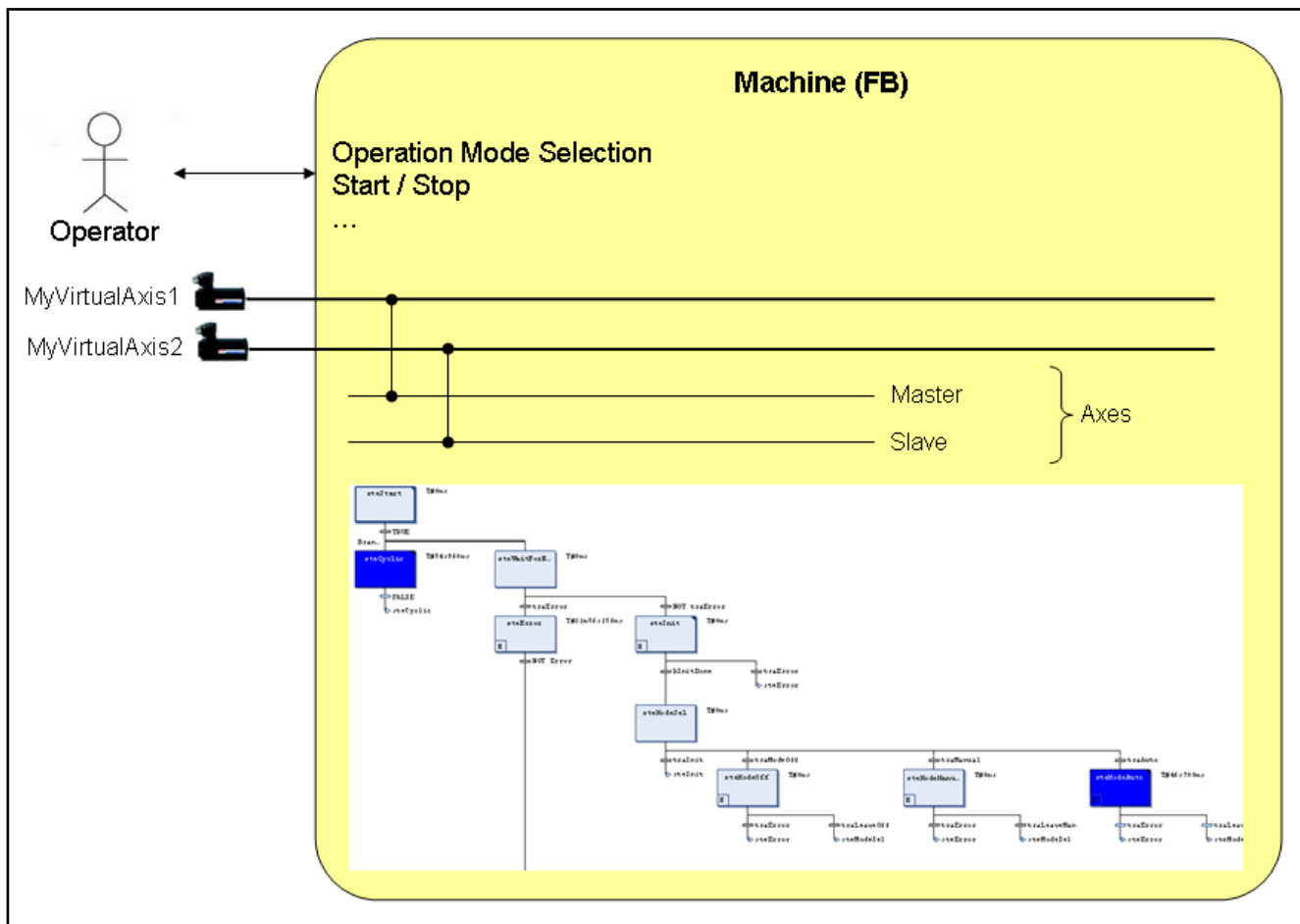


Fig.7-2: Schematic assignment of tasks

Operating sequence

As described earlier, the GAT wizard starts. In the following dialog, select "GAT compact" and then click the **Finish** button.

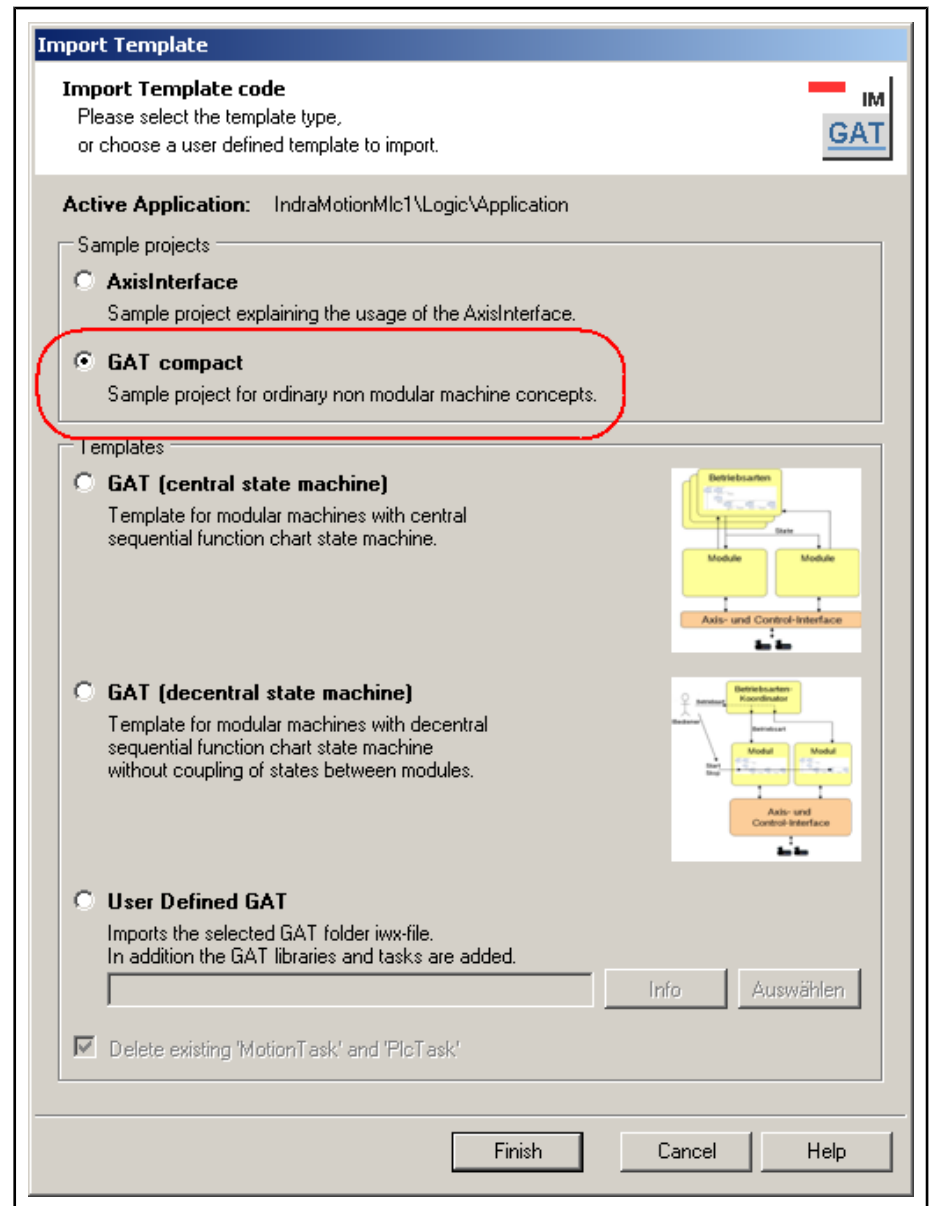


Fig.7-3: GAT wizard start up dialog

By selecting "GAT compact", the sample project is imported, in which two global virtual axes (MyVirtualAxis1, MyVirtualAxis2) have already been configured in addition to a PLC program template.

After the sample project has been successfully imported, the guided configuration of the GAT wizard appears (following figure). The guided configuration gives a short overview of the GAT^{compact} functionalities and can be exited at any time by pressing the <F2> button.

GATcompact

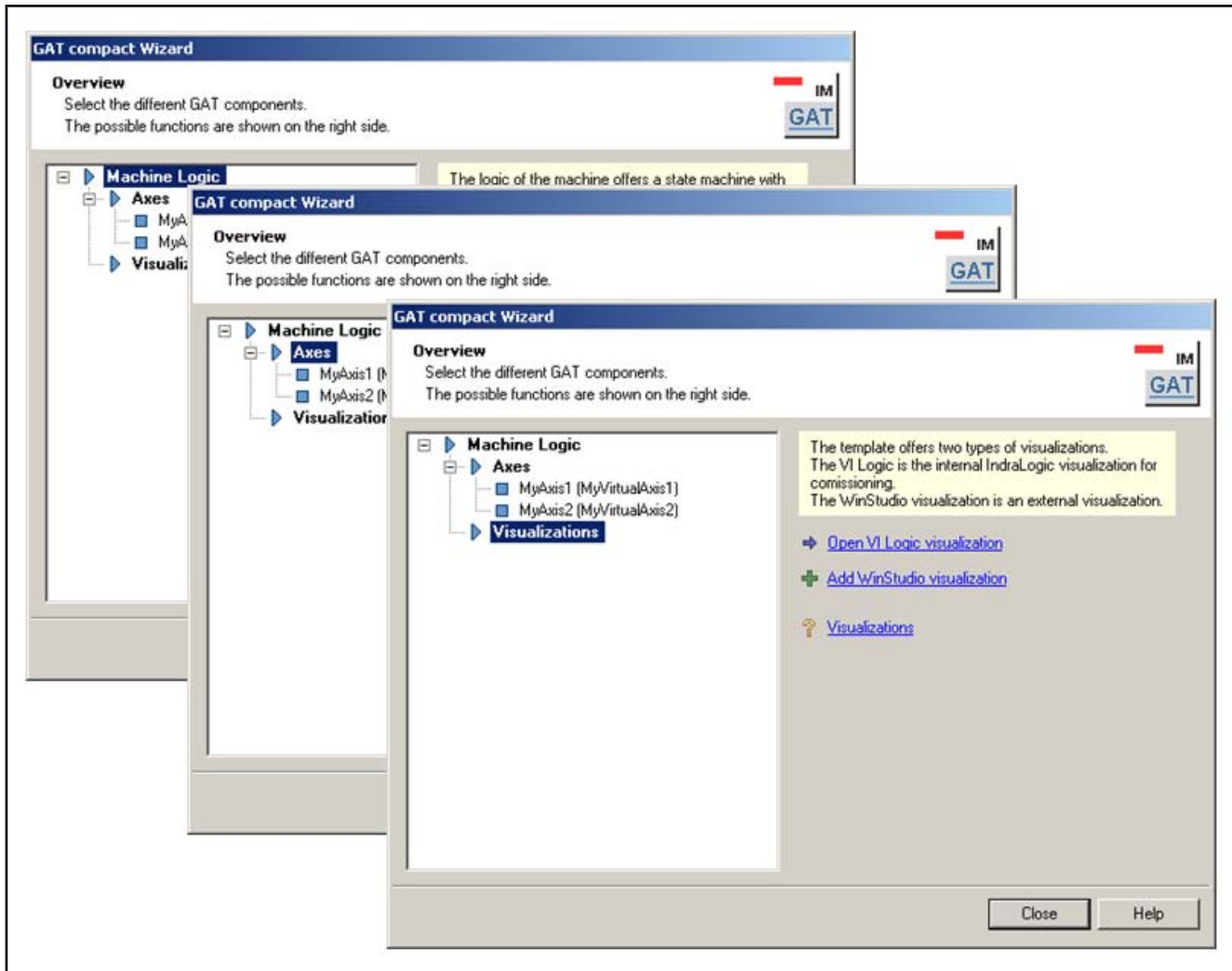


Fig.7-4: GAT wizard guided configuration

In GAT^{compact}, as an example, two virtual axes have been declared and implemented. The axes "MyAxis1" (global: MyVirtualAxis1) and "MyAxis2" (global: MyVirtualAxis2) are defined in the "TE_GAT_AXES" structure. The "Axes" variable of this type is declared in the Function_Block "Machine" and the assignment of global XLC/MLC axes to GAT^{compact} axes is carried out in the "FB_init" method in the FB "Machine" (see the following figure).

When using the GAT wizard, however, the user does not have to carry out these tasks. When adding, deleting or modifying GAT^{compact} axes, the GAT wizard carries out all of the code changes that affect the declaration of an axis in the GAT^{compact} axis container.

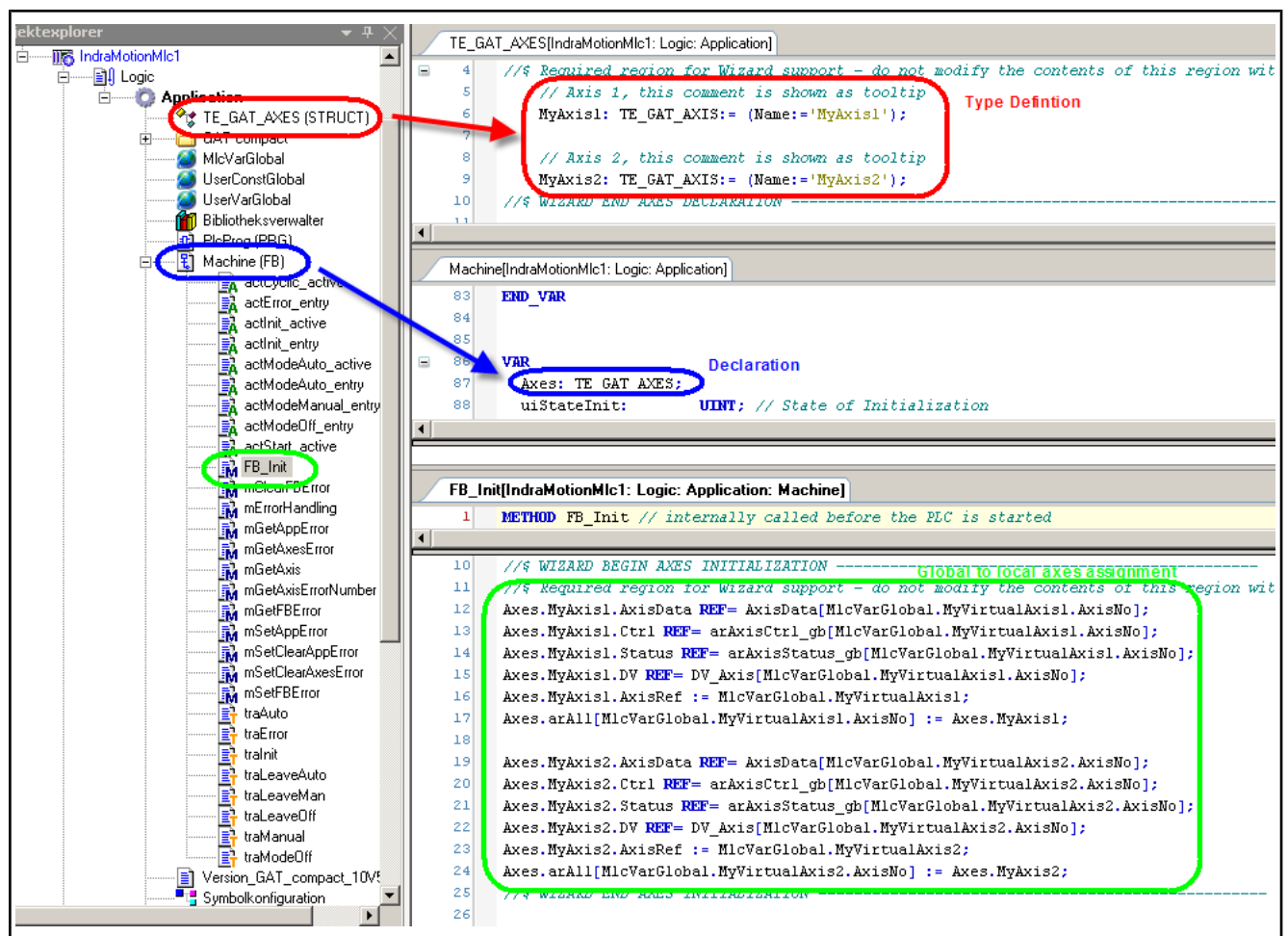


Fig.7-5: Declaring the axis data

The GATcompact axes are implemented in the following actions: "actModeAuto_entry" (process that takes place once when the "AUTO" operation mode is activated) and "actModeAuto_activ" (cyclical processing while the "AUTO" operation mode is active). The implementation of the actions is shown in the following figure.

GATcompact

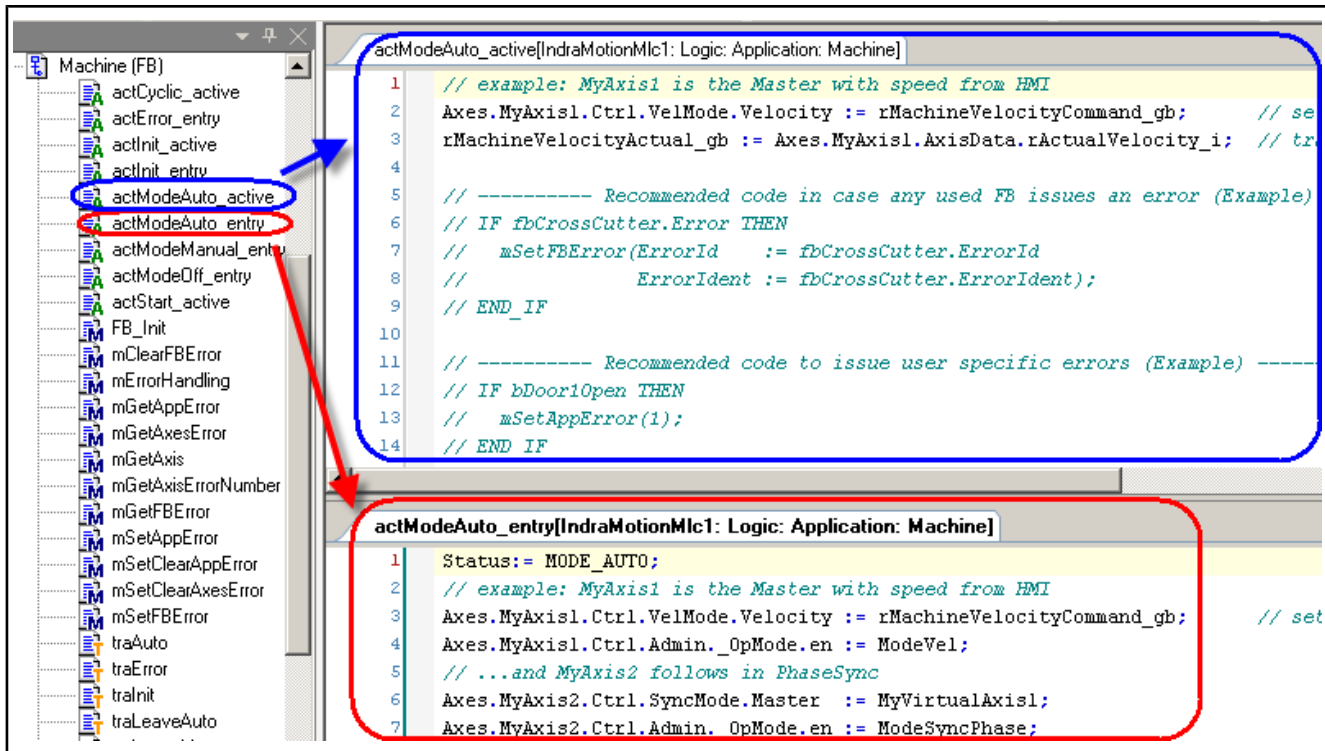


Fig.7-6: Controlling the axes

An analysis of the code for the "actModeAuto_entry" action demonstrates that "MyAxis1" is commanded in the velocity mode and "MyAxis2" should follow the first axis phase synchronously. This content corresponds exactly with the topic. According to the topic, however, the names of the GATcompact axes should be "Master" and "Slave". To do this, the GATcompact wizard opens and the first axis below the "Axes" node is highlighted. The axis name in parentheses is the name of the assigned global IndraMotion MLC-/IndraLogic XLC axis. The name of the GATcompact axis must be changed to "Master" (see the following figure).

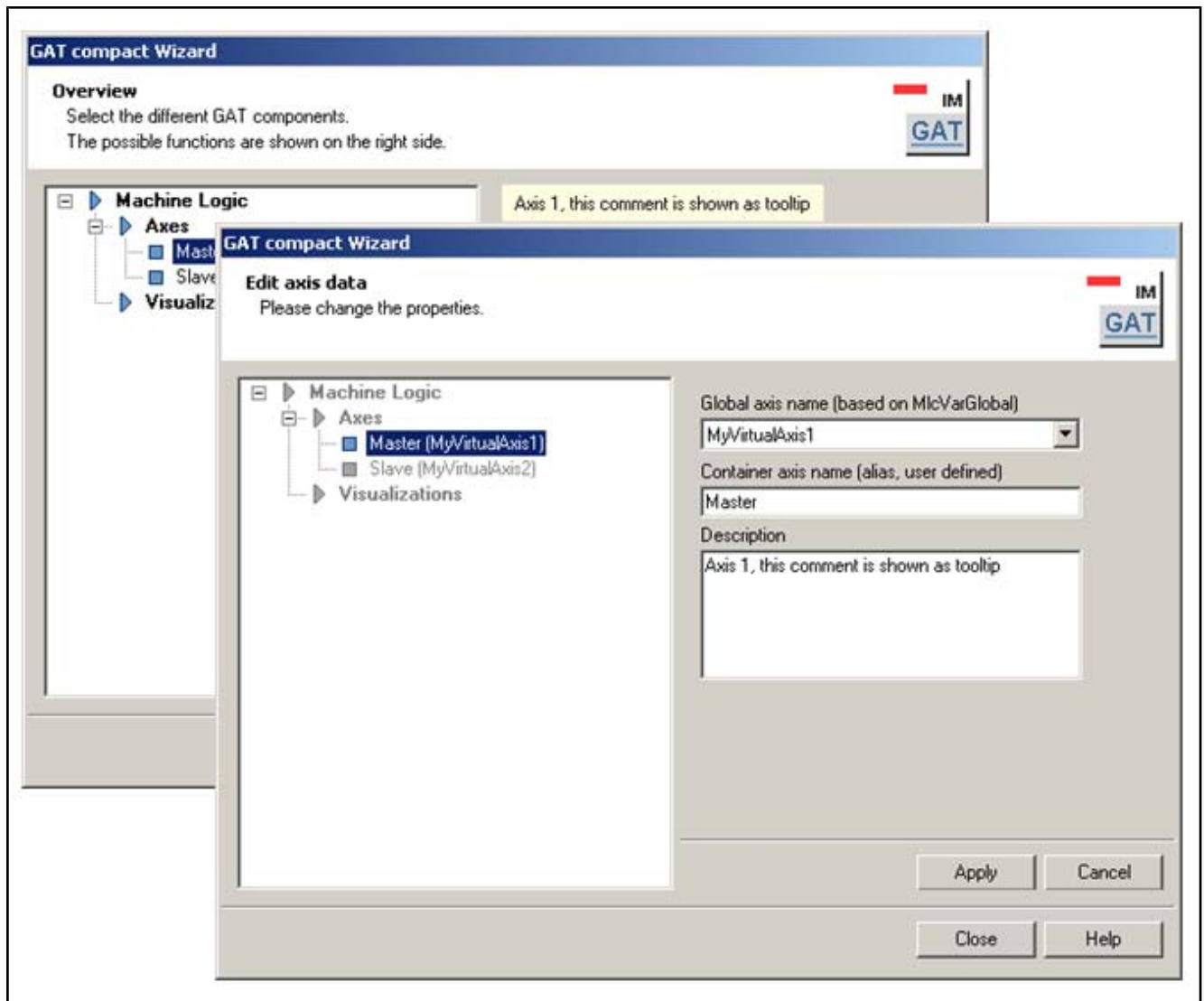


Fig.7-7: Changing the name of the axis with the GAT wizard

The name of the second GAT^{compact} axis is to be changed from "MyAxis2" to "Slave" in the same way. After the changes, the GAT^{compact} axis container now contains "Master" and "Slave" axes, displayed in the GAT^{compact} wizard as shown in the following figure.

GATcompact

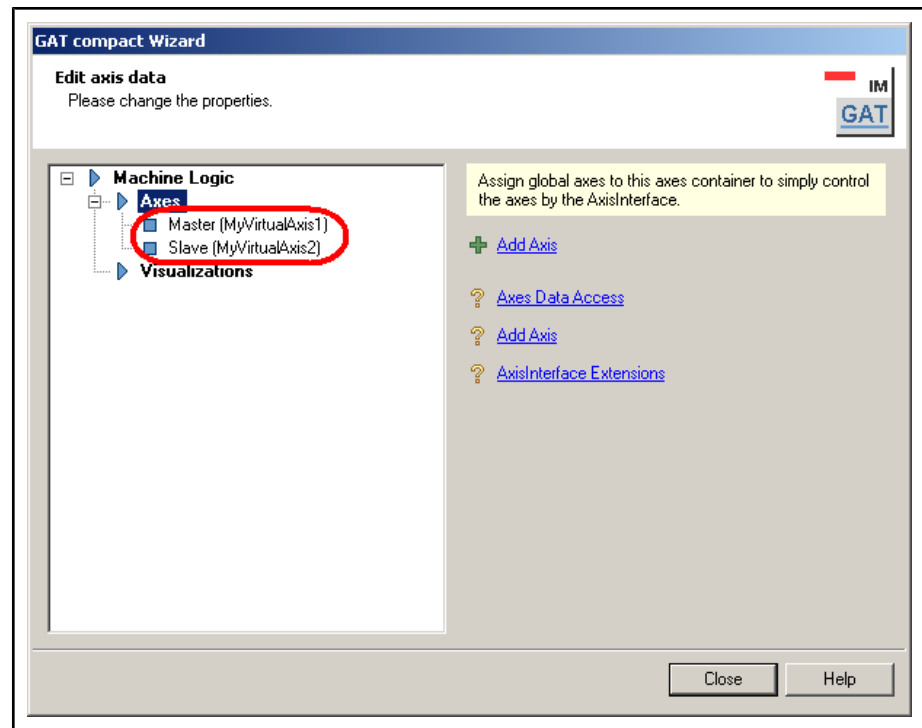


Fig. 7-8: Axis overview after adjustments

However, with the GAT^{compact} wizard, only the code changes with respect to the declaration of the GAT^{compact} axes are made. The implementation in the "actAutoMode_entry" and "actAutoMode_active" actions must be executed manually. This can be done with "Find and Replace" or the action is supported by the IndraLogic Intellisense functionality if the axis names are to be immediately changed into the actions. To do this, delete the point following the "Axes" identifier and insert it again. Following this action, all of the elements of the "Axes" structure will be displayed. In this way, when the "Slave" axis is highlighted and <Enter> is pressed, the "MyVirtualAxis2" identifier is replaced by "Slave" (see the following figure).

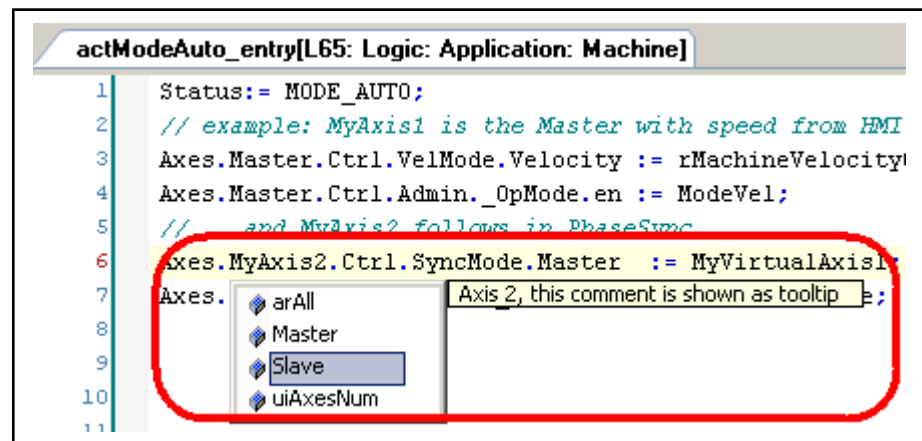


Fig. 7-9: Intellisense function

As a result, the actions in which the GAT^{compact} axes are implemented appear as follows:

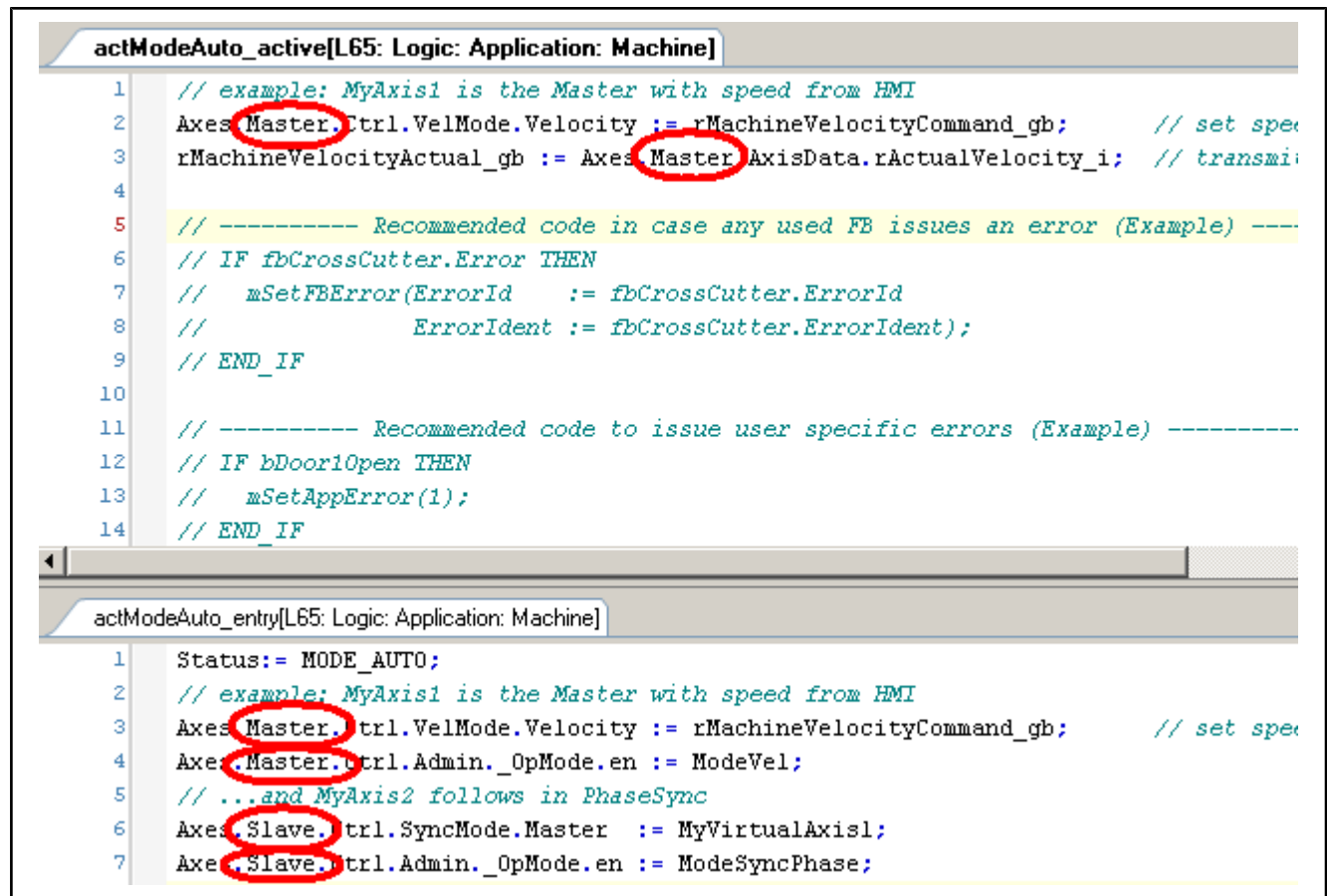


Fig.7-10: Implementation of the axis control for the example

Remark:

Changing the visualization is not necessary because the axis visualizations reference the global IndraLogic XLC-/IndraMotion MLC axes.

The implementation is complete.

If all of the changes are carried out correctly, the project can be compiled without any errors.

After the configuration (1) and the PLC project (2) have been downloaded, the PLC can be started (3) (see the following figure).



Fig.7-11: Downloading (configuration/PLC project) and starting the PLC

When the pre-defined internal visualization "System_Overview_All_Axis" is used, after the initialization is completed, press the **Auto** button to command the application into the "Auto" operation mode.

GATcompact

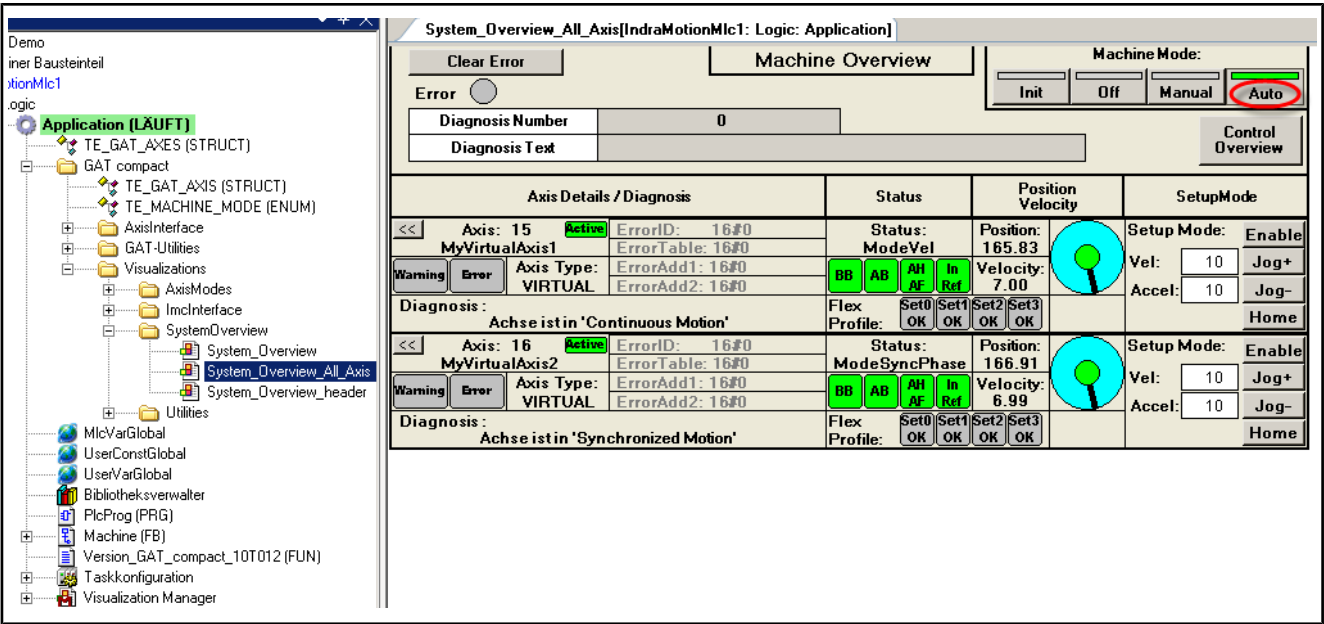


Fig.7-12: IndraLogic: Commissioning the visualization

In the "Auto" operation mode, the global physical axis "MyVirtualAxis1" ("Master" axis in the axis container of GATcompact) moves continuously at 100 rev./min. The global axis "MyVirtualAxis2" ("SlaveAxis" axis in the axis container of GATcompact) follows this axis phase synchronously.

This meets the requirements of the topic.

7.3 Initialization

Initialization is completed at the start. After a successful initialization, the machine is in operating mode.

At the beginning of the initialization, the system is commanded according to cold start (parameterization mode/P2). In the second step of the initialization, the system is commanded according to warm start (BB).

If the machine is already in operating mode, by selecting the initialization the machine can be commanded according to cold start. This is maintained until an operation mode of the operating mode is selected.

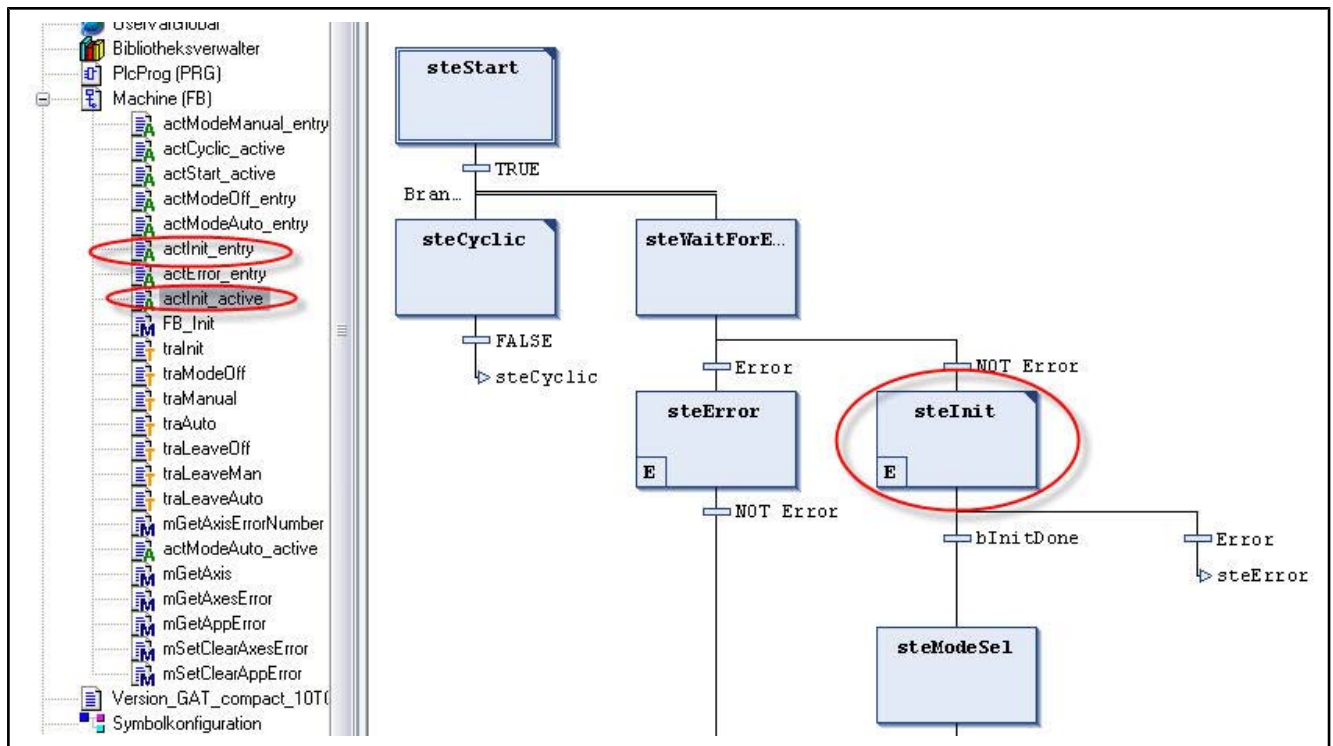


Fig. 7-13: Init step in the machine SFC

As long as an error has not occurred, the initialization is carried out in the "steInit" step. At the beginning of this step, the input action "steInit_entry" is processed once.

In case an error occurs during initialization, a switch is made into error status. After the error has been cleared, the initialization continues at the position at which it was interrupted.

When the bit "bInitDone" is set, the process exits the initialization and moves on to the next step.

GATcompact

```

actInit_active[IndraMotionMlc1: Logic: Application: Machine]
1  CASE uiStateInit OF
2      uiSTATE_INIT_PREPARE_COLDSTART:
3          bInitDone := FALSE; // Init is not done
4          bAllAxesOff := TRUE;
5
6          // switch all axes off to ModeAb
7  FOR uiLoopIndex := 1 TO uiMAX_AXIS_INDEX DO
8      // for all active axes, but only when added by the wizard (<>NO_OBJECT)
9      IF Axes.arAll[uiLoopIndex].Status.Admin.Active AND
10         Axes.arAll[uiLoopIndex].AxisRef.AxisNo <> NO_OBJECT AND
11         Axes.arAll[uiLoopIndex].Status.Admin._OpModeAck <> ModeAb
12      THEN
13          bAllAxesOff := FALSE; // at least one is under Power....
14      END_IF
15  END_FOR
16
17  IF (bAllAxesOff = TRUE)
18  THEN
19      uiStateInit := uiSTATE_INIT_SWITCH_TO_COLDSTART;
20  END_IF
21
22
23  uiSTATE_INIT_SWITCH_TO_COLDSTART:
24      ImcCtrl.Admin._OpMode := IMC_MODE_CTRL_P2; //Switch your MLC to P2 (Parameter mode)
25
26      IF (ImcStatus.Admin._OpModeAck = IMC_MODE_STATUS_P2)
27      THEN
28          uiStateInit := uiSTATE_INIT_COLDSTART;
29      END_IF
30
31
32  uiSTATE_INIT_COLDSTART:
33      IF (Ctrl <> MODE_INIT)
34      THEN
35          // *****
36          // Now we reached P2 (parameter mode)
37          // Add your code for writing the parameters (in P2) here
38          // *****
39          uiStateInit := uiSTATE_INIT_WARMSTART;
40      END_IF
41
42  uiSTATE_INIT_WARMSTART:
43      ImcCtrl.Admin._OpMode := IMC_MODE_CTRL_BB; //Switch your MLC to BB (Operation mode)
44
45      IF (ImcStatus.Admin._OpModeAck = IMC_MODE_STATUS_BB) AND
46          (TE_AxisInterfaceType01Prog.InitDone = TRUE)
47      THEN
48          // *****
49          // Now we reached BB (operating mode)
50          // Add your code for writing the parameters (in BB) here
51          // *****

```

Fig.7-14: Implementation of the initialization in the "actInit_active" action
The following occurs in the "actInit_active" step:

1. At the beginning, all active axes are checked to see if they are ready and free from torque (Ab) (lines 7 to 15). Otherwise, the machine cannot be commanded according to cold start (P2).
2. Then the machine is commanded into the cold start (P2) state (lines 23 to 29).
3. When the machine reaches the cold start state, the state is maintained until an operation mode of the operating mode (e.g. ModeOff, ModeAuto, ModeManual) is commanded. This is the position at which the parameters that can only be modified in P2 can be written in the control and the drives (lines 32 to 40).
4. Then the machine is commanded into the warm start state (line 39). This is the position at which the parameters that can only be modified in P4 can be written in the control and the drives (lines 42 to 51).
5. When the initialization is finished, the "bInitDone" bit is set to "TRUE" and the process exits the initialization. Now the machine is ready for operation in the "Manual" or "Auto" operation mode.

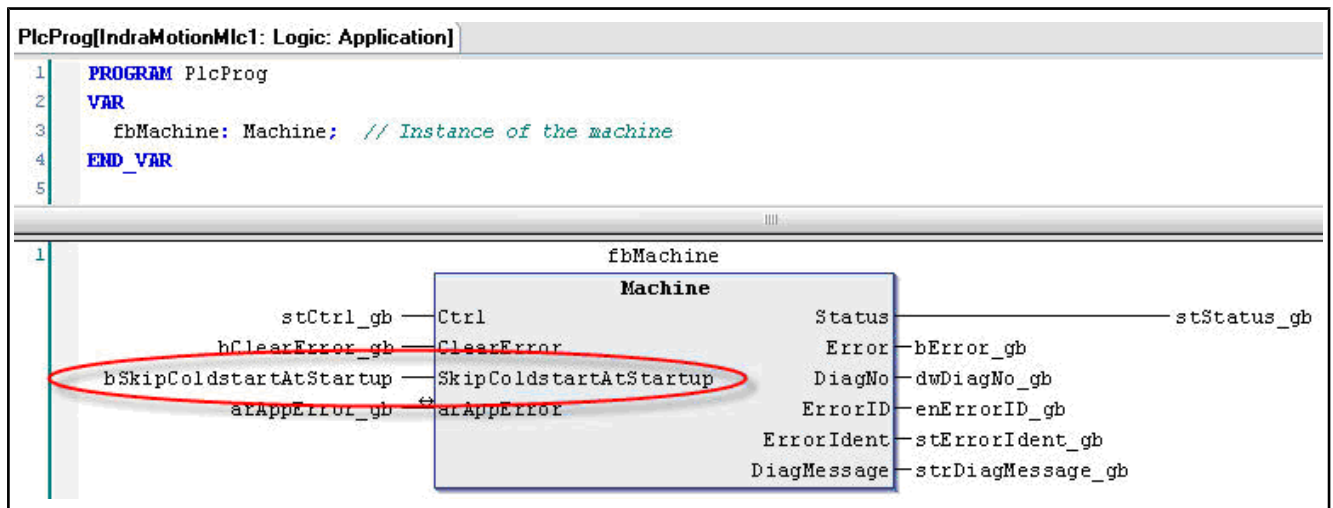


Fig. 7-15: "SkipColdStartAtStartup" input at the Machine FB

If the initialization should not be completed the first time the PLC is started, this can be set using the "bSkipColdstartAtStartup" bit. Then the process will jump into the warm start state; cold start will not be processed. This eliminates the switch into P2 and then back to BB. This setting saves time during when powering up. However, it should be ensured that the actions processed in the cold start state (e.g. writing parameters) are not required.

7.4 Operation Modes

After a successful initialization (see [Initialization, page 130](#)), the sequence branches into one of the 3 operation modes (see the following figure).

- ModeOff (all drives free of torque)
- ModeManual (machine setup, axes can be jogged)
- ModeAuto (cyclical Automatic operation)

GATcompact

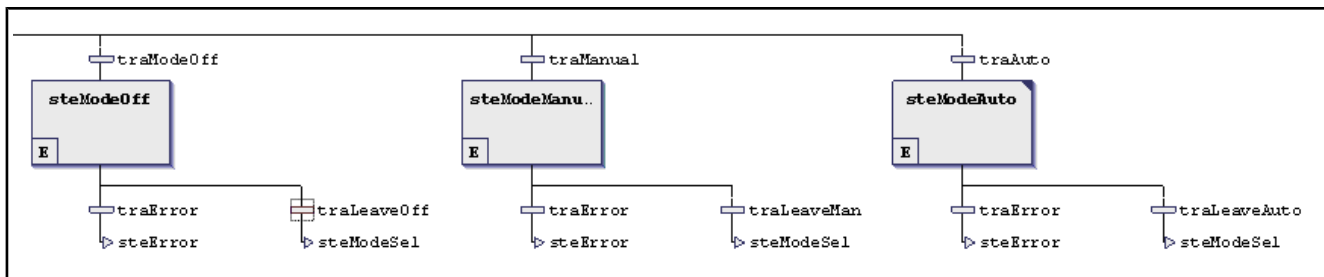


Fig.7-16: Operation Modes

In GATcompact, "ModeManual" is preset as the target operation mode after the initial machine start (see the following figure).

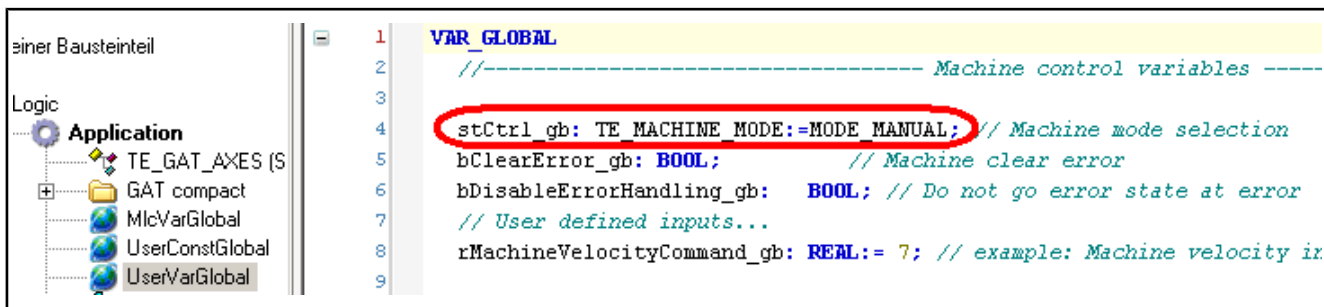


Fig.7-17: Setting the target operation mode after the initial machine start

Entry-/active action

When an operation mode is activated, a special "entry" action is carried out just once. The following table provides an overview of this action, shows how each operation mode is activated and what is done in the operation mode.

	Condition for activation	Action (executed once at the beginning)	Action executed cyclically
ModeOff (drives free of torque)	Ctrl=MODE_OFF query is carried out in traModeOff and traLeaveOff	actModeOff_entry - Sets the status to "MODE_OFF" - Sets all active axes to "ModeAb" (axes are free of torque)	No predefined cyclical action
ModeManual (machine setup, axis jogging)	Ctrl=MODE_MANUAL query is carried out in traManual and traLeaveManual	actModeManual_entry - Sets the status to "MODE_MANUAL" - Sets all active axes to "ModeAH" mode (axis loaded with torque, v=0)	No predefined cyclical action Typical activities: Machine setup and adjustment, e.g. jogging selected axes
ModeAuto (cyclical Automatic operation)	Ctrl=MODE_AUTO query is carried out in traAuto and traLeaveAuto	actModeAuto_entry - Sets the status to "MODE_AUTO" - Commands "MyVirtualAxis1" in the example into the axis velocity mode and synchronizes "MyVirtualAxis2" without phase offset	actModeAuto_active e.g. processing of process controllers, monitoring the production process (material available? synchronous coupling/decoupling)

An ongoing scan for errors is executed simultaneously with processing in an active operation mode. In case an error occurs, the process branches into error handling (see [chapter 8.9.1 "Error Detection and Diagnostics" on page 194](#)).

To add operation modes, see [chapter 7.10.4 "Adding an Operation Mode" on page 158](#).

7.5 Cyclical Processing

In GATcompact the cyclical program section is currently empty. For general information, see [chapter 5.4 "Cyclical Processing" on page 34](#).

7.6 Task Handling

7.6.1 Task Configuration, GATcompact

The sample application GATcompact is purposely limited to a PLC task (PlcTask). The following section shows the default setting for the PlcTask in GATcompact.

	Priority	Cycle time	Watchdog time	Watchdog sensitivity
PlcTask	5	T#10ms	T#20ms	3

If necessary, a quick MotionTask and a lower priority BackgroundTask can be supplemented in GATcompact.

7.6.2 Adding a MotionTask

See also [chapter "Typical Tasks for the MotionTask" on page 35](#).

	Priority	Cycle time	Watchdog time	Watchdog sensitivity
MotionTask	2	=SERCOS cycle time	T#2 to 10 ms	1..2

To add a MOTION synchronous task, the following steps are required:

- Task configuration, adding the task "MotionTask", for settings see [fig. 7-18 "Configuration of the MotionTask" on page 136](#).
- Add POU MotionProg and call MotionProg in the configuration of the MotionTask
- Add the method mMotionCyclic at the FB "Machine". Call it in MotionProg. With the method mMotionCyclic, all variables (e.g. central axis entry point Axes...) are available in the method.

GATcompact

The screenshot displays the configuration interface for the MotionTask and MotionProg. The top section is titled "MotionTask[IndraMotionMlc1: Logic: Application: Task Configuration]". It includes a "Configuration" tab with the following settings:

- Priority (1..20): 2
- Type: External (dropdown menu)
- External event: EVENT_OP_MODE_MOTION_CYCLIC
- Watchdog:
 - ☒ Enable
 - Time (e.g. t#200ms): T#10ms
 - Sensitivity: 1
- POUs:
 - Buttons: Add POU, Remove POU, Open POU, Input Assistant...
 - Table:

POU	Comment
MotionProg	

The bottom section is titled "MotionProg[IndraMotionMlc1: Logic: Application]". It shows a ladder logic program with the following code:

```

1  PROGRAM MotionProg
2  VAR
3  END_VAR
4
1  // call method of the Machine-FB
2  PlcProg.fbMachine.mMotionCyclic();
  
```

Fig. 7-18: Configuration of the MotionTask

In order to access axes consistently from the MotionTask, the AxisInterface must also be called in the MotionTask. Because processing all of the axes in the MotionTask would cause an unnecessary load on the MotionTask, the following procedure is recommended.

The following steps are required:

- Extend the structure TE_GAT_AXIS (in the "GATcompact" folder) to include the element MotionSync (type BOOL)
- In the Init section (e.g. actInit_entry), set the new element MotionSync for the selected axes to TRUE.
Example: ST Axes.MyVirtualAxis1.MotionSync := TRUE;
- TE_AxisInterfaceType01Prog():
 - Supplement the program with the flag query MotionSync = FALSE.

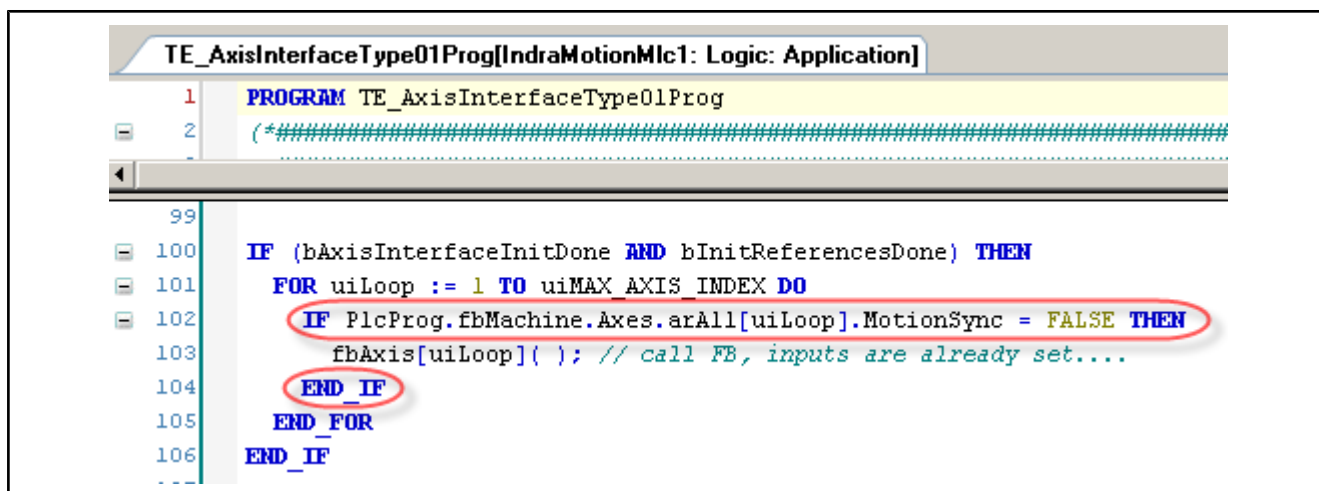


Fig. 7-19: Flag query MotionSync = FALSE

- Add action actMotionTask. Copy the call loop for the function block fbAxis() (see [fig. 7-19 "Flag query MotionSync = FALSE" on page 137](#)). Create new loop variable uiLoopMotionSync (do not use uiLoop!); change flag query MotionSync to TRUE.

Result:

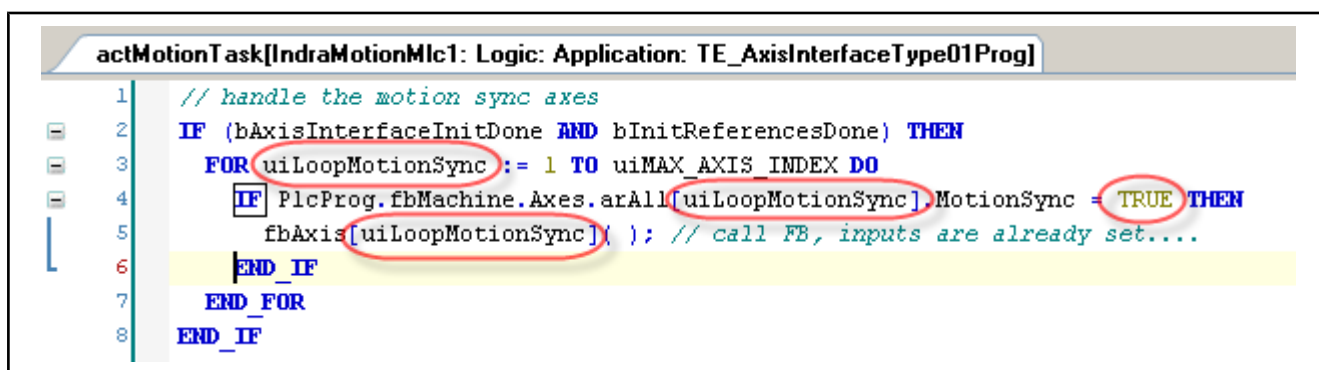


Fig. 7-20: "actMotionTask" action

- Add the call of the action actMotionTask in the MotionProg (TE_AxisInterfaceType01Prog.actMotionTask());

7.6.3 Adding a BackgroundTask

If the runtime for the PlcTask is too long, the load on the PlcTask can be reduced by moving tasks that are not time critical to a background task. Please note, however, that a greater load is placed on the entire system due to the additional task Switch PlcTask <> BackgroundTask (see also [chapter "Typical Tasks for the BackgroundTask" on page 36](#)).

	Priority	Cycle time	Watchdog time	Watchdog sensitivity
BackgroundTask	~6	T#40ms	-	-

In order to add a BackgroundTask, the following steps are required:

- Task configuration, adding the task "BackgroundTask", for settings see [fig. 7-21 "Configuration of the BackgroundTask" on page 138](#).
- Add POU BackgroundProg and call BackgroundProg in the configuration of the BackgroundTask

GATcompact

- Add the method mBackground at the FB "Machine". Call it in Background-Prog. With the method mBackground, all of the variables in the method are available.

The screenshot shows the configuration of a background task and its associated program. The top section, titled 'BackgroundTask[IndraMotionMlc1: Logic: Application: Task Configuration]', contains the following settings:

- Configuration** tab is selected.
- Priority (1..20):** 6
- Type:** Cyclic (dropdown menu)
- Interval (e.g. t#200ms):** t#40ms
- Watchdog:**
 - ☐ Enable
 - Time (e.g. t#200ms):** (empty field)
 - Sensitivity:** 1
- POUs:**
 - Buttons: Add POU, Remove POU, Open POU, Input Assistant...
 - Table:

POU	Comment
BackgroundProg	

The bottom section, titled 'BackgroundProg[IndraMotionMlc1: Logic: Application]', shows the program code:

```

1  PROGRAM BackgroundProg
2  VAR
3  END_VAR

1  // call method of the Machine-FB
2  PlcProg.fbMachine.mBackground();
  
```

Fig. 7-21: Configuration of the BackgroundTask

7.7 Error Handling

7.7.1 Error Detection and Diagnostics

The program template GAT^{compact} contains predefined functionalities for detecting, handling and clearing errors at a **central location**. Errors from various sources (e.g. control errors, application errors, FB errors, ...) are automatically gathered together with the cross-machine FB "Machine" in the action "mErrorHandling" and are made available using the diagnostic outputs of the FB "Machine".

If several errors occur simultaneously, the most severe is displayed. If several errors with the same degree of severity have occurred simultaneously, the most recently detected error is output.

The FB "Machine" contains the following described outputs for diagnostic purposes:

Element	Type	Description
Error	BOOL	TRUE indicates an error
DiagNo	DWORD	Number according to the IndraLogic XLC-/IndraMotion MLC documentation with the following extensions: • 16#F0AE xxxx: Application error (e.g. safety door is open) (xxxx = can be defined by the user) • 16#F0FB 0000: FB error (e.g. error at a TechFB) -> for details, see ErrorID / ErrorIdent • 16#F0AA 0000: GAT internal error -> for details, see ErrorID / ErrorIdent
ErrorID	ERROR_CODE	Detailed diagnostics in case of FB + GAT errors (DiagNo = 16#F0FB/AA 0000)
ErrorIdent	ERROR_STRUCT	Detailed diagnostics in case of FB + GAT errors (DiagNo = 16#F0FB/AA 0000)
DiagMessage	STRING(60)	It contains the diagnostic text for drive and control errors

The **DiagNo** output is only for use on an HMI. The error texts are assigned the numerical information in any desired language there.

DiagMessage, **ErrorID** and **ErrorIdent** provide additional information that is primarily displayed during commissioning by the programmer, e.g. using an IndraLogic visualization.

Note regarding application-specific errors

By calling the method `mSetAppError(Number)`, application errors (e.g. safety door is open) can be signaled. The "number" transferred at the method input corresponds with the application error number.

If an application number was signaled, the central diagnostic system in the "Machine" FB performs cyclical monitoring of the user errors that have been signaled and generates the application error `DiagNo = Error 16#F0AE xxxx` (xxxx = application error number).

In the global variable list "UserConstGlobal", the application errors can be assigned to diagnostic texts, which are then displayed in the VI logic visualization.

Numbers in the range 1 to `uiMAX_APP_ERROR_INDEX` are valid for application errors in the machine. The constant "`uiMAX_APP_ERROR_INDEX`" has the value 64 by default. If necessary, this constant can be adjusted in the global variable list "UserConstGlobal" below the "Application" folder.

Code sample for signaling an application error:

```
// ----- Recommended code to issue user specific errors (Example) -----
IF bDoor1Open THEN
    mSetAppError(1); // Issue application error #1
END_IF
```

Fig. 7-22: Code sample for signalling an application error

Note regarding severe errors

The control diagnostic system defines 10 degrees of "severity" (see IndraLogic XLC-/IndraMotion MLC diagnostic system). The degree of severity affects the error reaction (see [chapter 5.6.4 "Error Reaction" on page 41](#)).

- `DiagNo = 16#F0xx xxxx` is the lowest degree of severity (e.g. application errors, FB errors)
- `DiagNo = 16#F9xx xxxx` is the highest degree of severity (fatal system errors)

GATcompact

The error with the highest degree of severity is displayed if several errors occur simultaneously.

Note regarding function block errors

If errors occur in called function blocks (e.g. technology FBs), calling the method "mSetFBError(ErrorID, ErrorIdent)" is recommended along with transferring the "ErrorID" and "ErrorIdent" outputs of the FB that caused the error to the method. This ensures that the error is forwarded to the diagnostic system of the "Machine" FB and the "ErrorID" and "ErrorIdent" are made available to the diagnostic outputs of the "Machine" FB.

Code sample for signaling function block error:

```
// ----- Recommended code in case a used FB issues an error (Example) -----
IF fbCrossCutter.Error THEN
    mSetFBError(ErrorID := fbCrossCutter.ErrorId, ErrorIdent := fbCrossCutter.ErrorIdent);
END_IF
```

Fig.7-23: Code sample for signalling a function block error

7.7.2 Error Reaction

Depending on the system, the control system automatically executes an error reaction only for axis and control errors starting at a severity degree of 2 -> see [chapter 5.6.4 "Error Reaction" on page 41](#). For all other errors and lower degrees of severity (e.g. application errors, FB errors...) an automatic error reaction is not executed.

If GAT^{compact} error handling is activated ("Machine" FB input "DisableErrorHandling" = FALSE), in case of an error, the selected operation mode is always interrupted and the error reaction active. Additional application-specific error reactions can be programmed in the action "actError_entry" of the module.

When error handling is switched off ("DisableErrorHandling" = TRUE), the selected operation mode remains active even if the error output is set. The application-specific error reaction "actError" is not processed in this case.

For standard cases, the deceleration of all axes is programmed with "ModeAH" as an example in the following section (ModeAH = deceleration to full stop and torque remains on).

If the GAT^{compact} error handling is switched off (machine input "DisableErrorHandling" = TRUE), the selected operation mode is not interrupted (the application-specific error reaction is not started). The diagnostic outputs, however, are set in this case as well and can be deleted using "ClearError".

Standard case for the error reaction in the module action "actError_entry" (**Decelerate all axes**):

```
// ----- Error reaction example 1 -----
// Decelerate all module axes to standstill and hold power (e.g. hanging axes)
// --> switch all axes to ModeAH
// alternative: set to ModeAb to switch the axes OFF to Ab
// -----
FOR uiLoopIndex := 1 TO uiMAX_AXIS_INDEX DO
    // for all active axes, but only when added by the wizard (<>NO_OBJECT)
    IF Axes.arAll[uiLoopIndex].Status.Admin.Active AND
        Axes.arAll[uiLoopIndex].AxisRef.AxisNo <> NO_OBJECT THEN
        Axes.arAll[uiLoopIndex].Ctrl.Admin._OpMode.en := ModeAH; // or set ModeAb
    END_IF
END_FOR
```

Fig.7-24: Standard case for the error reaction in the module action "actError_entry"

If **"axes are suspended"**, torque may not be enabled in order to prevent falling axes. In these applications, switching all axes into "ModeAH" is recommended.

If torque enable is desired when an error occurs, in the example above, "ModeAH" must be replaced by "ModeAb" (ModeAb = Decelerate up to full stop and subsequent torque enable).

Often it is required that axes be decelerated **synchronously** (to prevent a material web from tearing, for example). In these applications, it is recommended that all axes remain connected in synchronous movement, and that only the master is decelerated. This use case is shown in the following example:

Example for **synchronous deceleration** of the axes in the module action actError_entry (only axes that are not synchronous are decelerated using "ModeAb").

```
// ----- Error reaction example 2 -----
// Keep all axes in synchronization with the master axis
// and decelerate only the master axis
// -----
FOR uiLoopIndex := 1 TO uiMAX_AXIS_INDEX DO
    // for all active axes, but only when added by the wizard (<>NO_OBJECT)
    IF Axes.arAll[uiLoopIndex].Status.Admin.Active AND
        Axes.arAll[uiLoopIndex].AxisRef.AxisNo <> NO_OBJECT THEN
        IF (Axes.arAll[uiLoopIndex].AxisData.SynchronizedMotion) THEN
            ; // Axis is in synchronized motion --> no reaction as the master axis
            // decelerates to standstill and the slave axis follows in synchronization
        ELSE
            Axes.arAll[uiLoopIndex].Ctrl.Admin._OpMode.en := ModeAb;
            ; // Axis is in synchronized motion --> do
        END_IF
    END_IF
END_FOR
```

Fig.7-25: Example for the synchronous deceleration of the axes in the module action "actError_entry"

For synchronous deceleration, the module bus reaction for the real axes must also be switched off. Furthermore, in the "best possible deceleration" of the real axes, "30s NC reaction" must be set.

GATcompact



In the setting "Best possible deceleration = 30s NC reaction" for a real axis, in the IndraLogic-XLC/IndraMotion MLC control system, in case of an F2 error, the axis affected by the error is decelerated with the emergency stop ramp A-0-0224. All other trouble-free axes are not affected.

7.7.3 Clearing Errors

The central clearing of error is automatically executed with a positive edge at the "ClearError" input of the "Machine" FB. The FB forwards the command to all subordinate components (e.g. ImcInterface¹⁾, AxisInterface ...) in order to bring the entire system into a trouble-free state. The active, selected operation mode continues processing after the error is cleared the process exited the mode due to the error reaction.

7.8 Visualization

7.8.1 Internal Visualization for Assistance during Start-up

General Information

GATcompact contains predefined IndraLogic visualization objects that offer assistance during commissioning as well as extended diagnostic possibilities.

System_Overview

The visualization element "System_Overview" can be instantiated for individual axes.

This is implemented as an example in the "System_overview_All_Axis" visualization for the "MyVirtualAxis1" and "MyVirtualAxis2" axes.

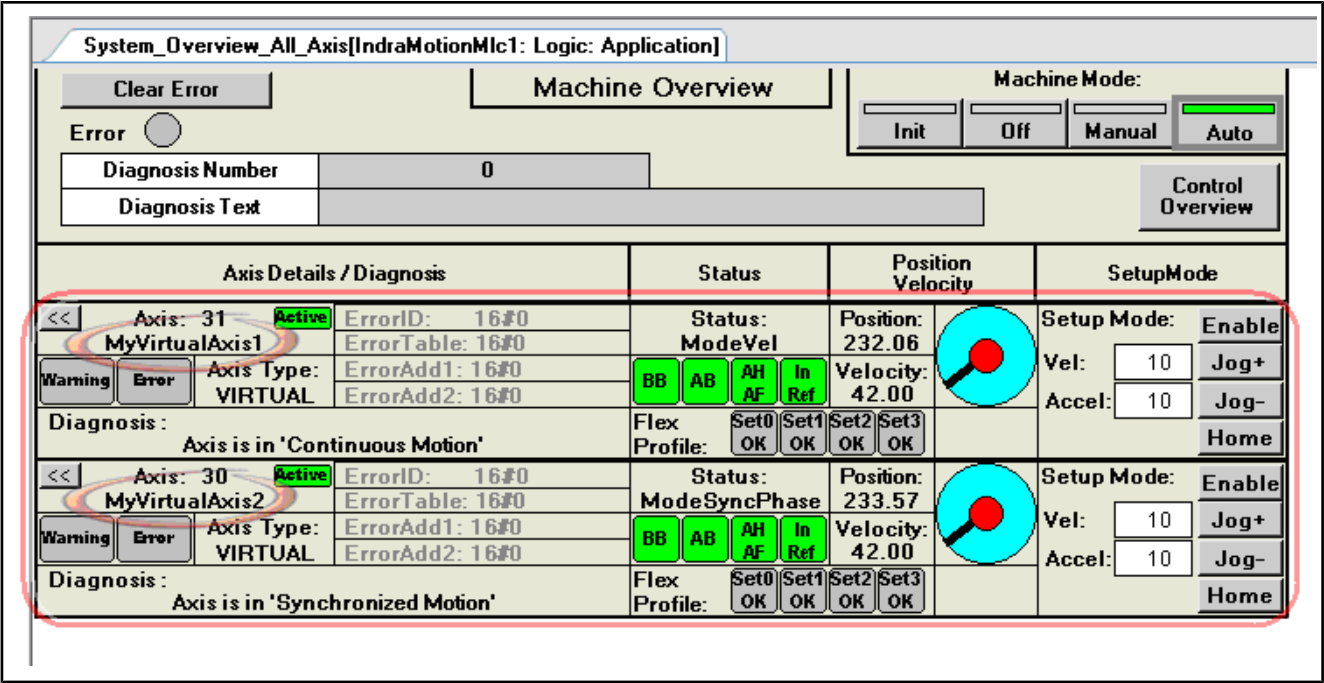


Fig. 7-26: Internal visualization for assistance during start up

¹⁾ IndraMotionControl Interface = IndraLogic library with an interface to the control, e.g. clear errors

The "Frame" object can be used to create an instance of the visualization object System_Overview in the visualization.

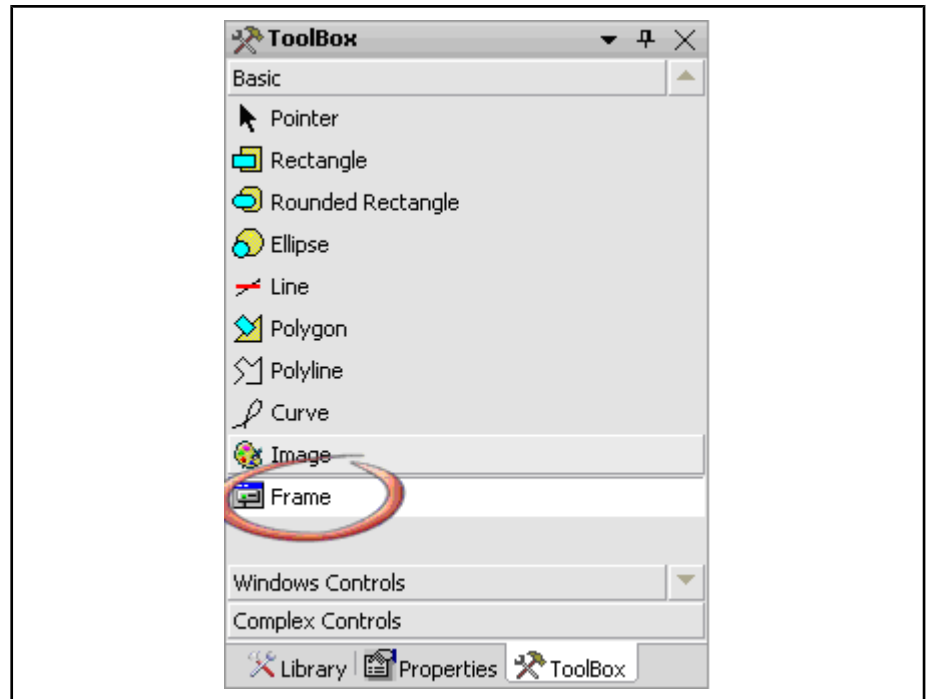


Fig.7-27: Creating an instance of the visualization object "System_Overview"

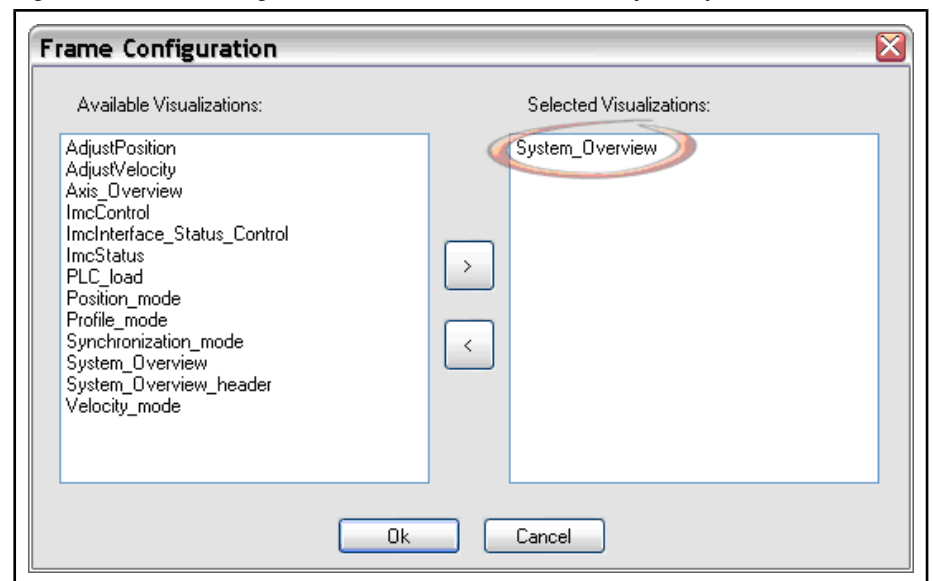


Fig.7-28: Creating an instance of the visualization object "System_Overview"

Now the corresponding axis reference must be entered under Properties - References.

GATcompact

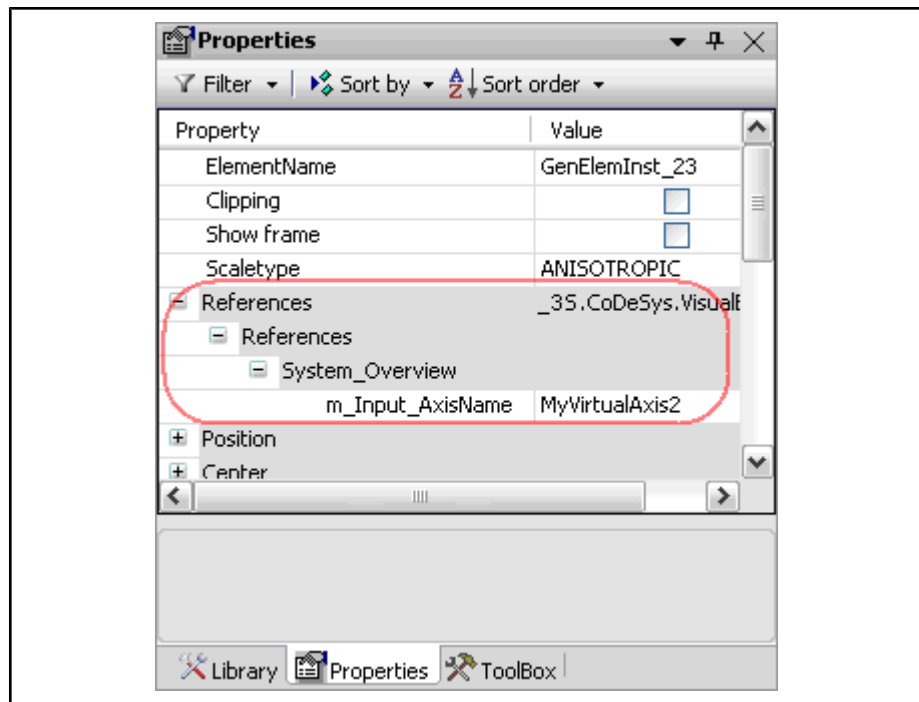


Fig. 7-29: Creating an instance of the visualization object "System_Overview"

7.8.2 External Visualization with Separate HMI (WinStudio Visualization)

General Information

A predefined WinStudio visualization is available for GAT^{compact}. A wizard can be used to add this visualization into the project.



To edit the WinStudio visualization, at least the following development environment license (or higher) is required:

WinStudio development license, max. 1,500 variables (Mat.No. R911323596)

For execution on a VEP, at least the following runtime license is required:

WinStudio runtime license for devices using Windows CE as an operating system, max. 1,500 variables (Mat. No. R911323620)

Adding the WinStudio visualization



In order to add the WinStudio visualization to the project, a corresponding visualization device must be present in the project.

Use "drag and drop" to add a VEP from the device library into the project tree and configure it as follows:

- Device type: VEPxx (CE)
- Application type: WinStudio application

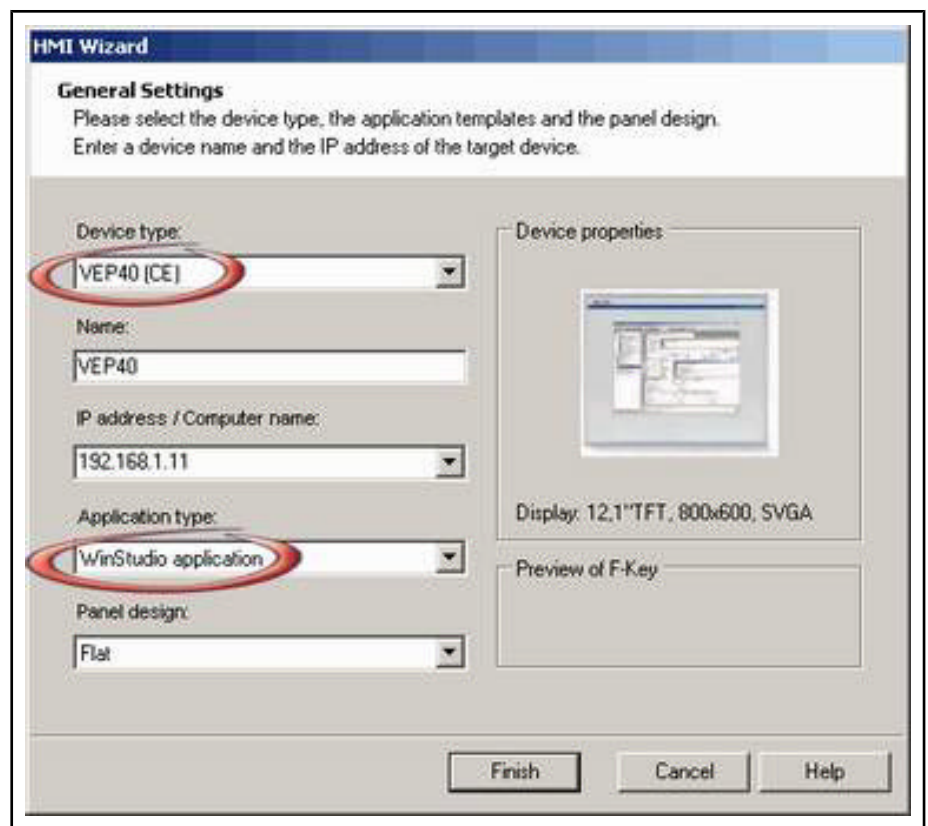


Fig. 7-30: Adding a visualization device

If a corresponding visualization device is present, the WinStudio visualization is added using the following link and the wizard.

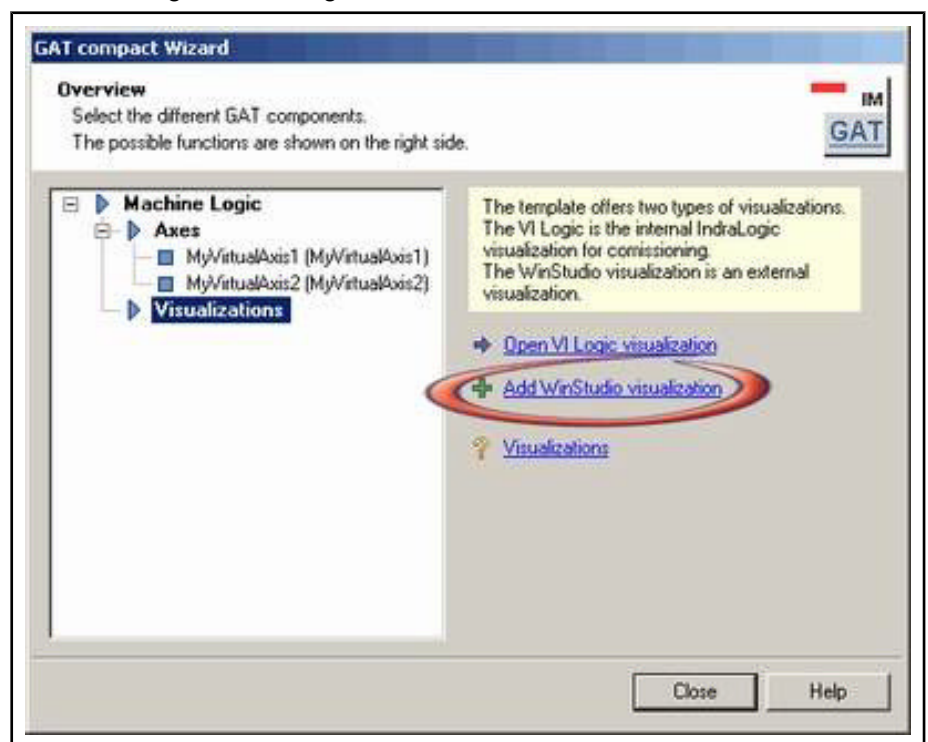


Fig. 7-31: Adding a WinStudio visualization

GATcompact

All of the project's visualization devices that correspond to the given configuration are included in the following wizard dialog.

After the corresponding visualization device is selected, click **Apply** to add the WinStudio visualization.

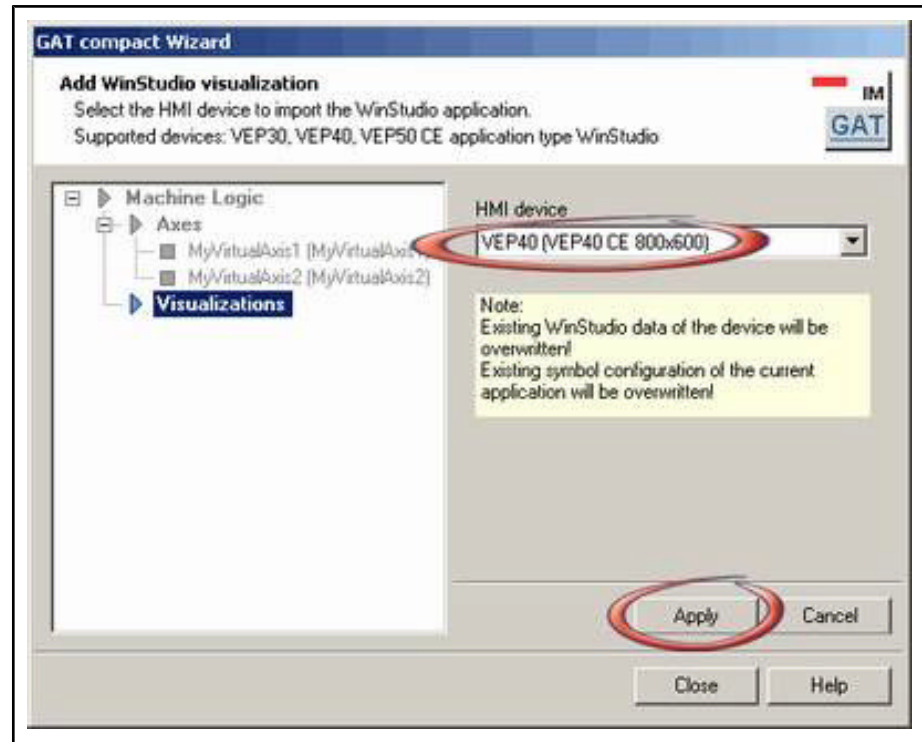


Fig.7-32: Selecting a visualization device and adding a WinStudio visualization



Existing visualization data from the visualization device along with the symbol configuration of the control are overwritten when the WinStudio visualization is added!

After the visualization is added, its data can be edited.

The control address is configured in the BR-WS driver properties dialog.

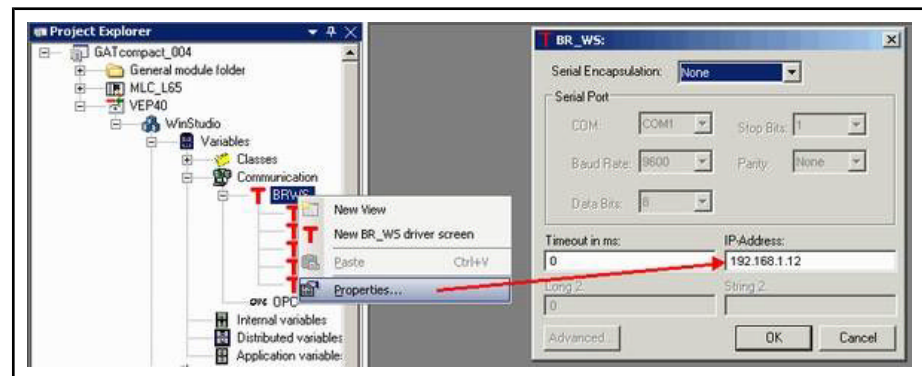


Fig.7-33: Setting the control's IP address

Now use  "Start application" in the WinStudio node context menu to start the emulation of the WinStudio visualization.

Dialog Structure

The operation screens are divided into 3 subsections:

- Header with status information
- Main screen with menu-related displays and function keys
- Footer with keys for switching from one user map to another

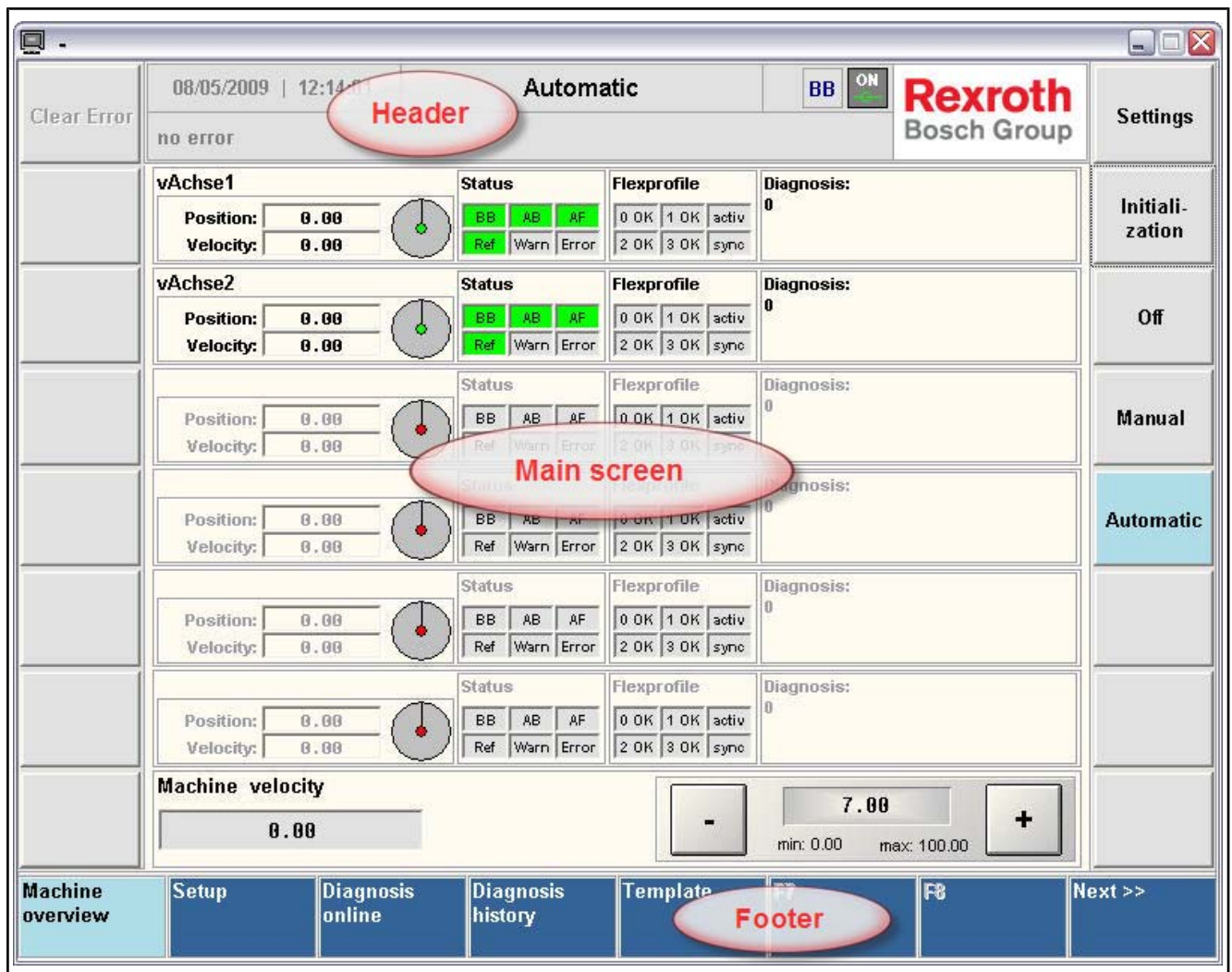


Fig. 7-34: WinStudio visualization

The **header** contains:

- Operation mode display
- Global function keys for clearing errors and opening the "Settings" dialog (for switching languages, setting the decimal place for numerical displays)
- Display of current machine diagnostics
- Display of current SERCOS phase (including active phase switching)
- Display of communication status

The **main screen** contains displays and function keys related to user maps (e.g. axis information, operation mode switching, ...)

The **footer** contains the keys for switching from one user map to another.

Extending the Visualization

Adding an operation screen

To extend the visualization, new user maps can be created by copying the existing "template" screens and inserting them with new names.

GATcompact

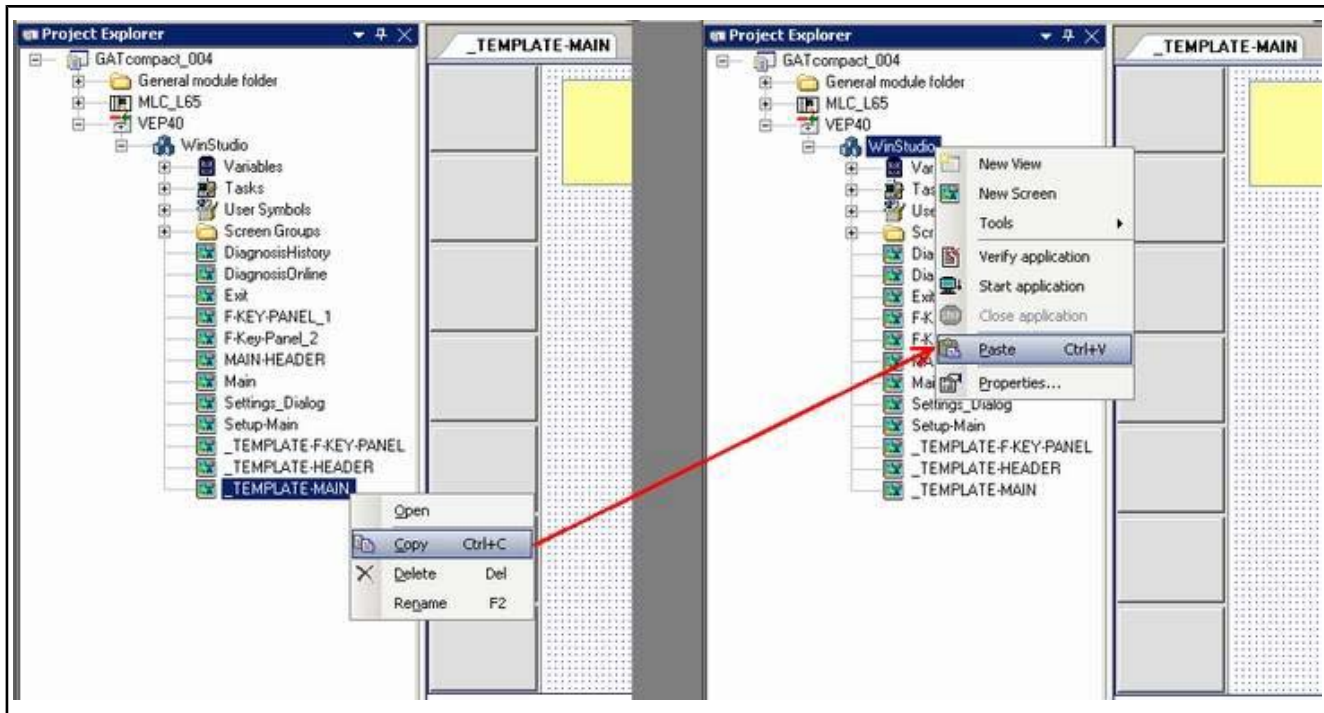


Fig. 7-35: Duplicating an operation screen

The elements for displaying the axis data are grouped individual objects that can be duplicated by "copying" and "pasting". To reconfigure the inserted object for a different axis the Replace function must now be used for the respective object to adapt the axis array index (see the following figures).



Fig. 7-36: Reconfiguring the inserted object

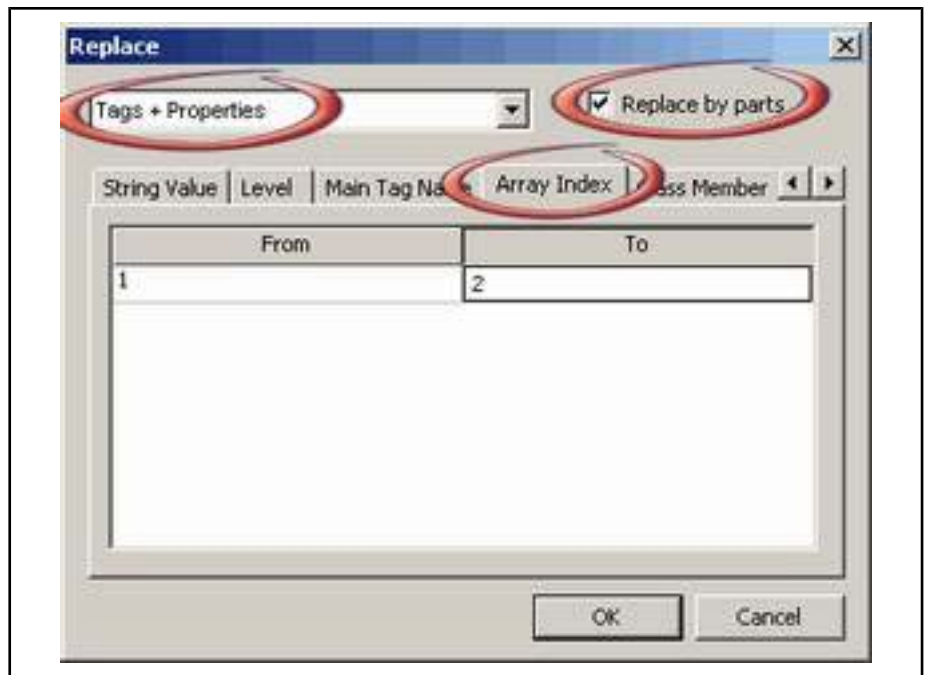


Fig.7-37: Reconfiguring the inserted object

Extending the footer

The keys used to switch from one user map to another are stored as graphic objects that can be parameterized (linked symbol). They have the following transfer parameters:

- Caption -> Key label
- Disable -> Input for blocking key functionality (grayed out label)
- ScreenName -> Screen name for the operation screens to be opened (or screen group)
- ScreenNumber -> Number to be assigned to the screen (to highlight the respective key)

Communication Connection

The communication between visualization and control is executed symbolically using the BR-WS driver. To do this, generate the symbol entries for the PLC variables to be visualized (see [chapter "Communication Configuration" on page 46](#)).

The required symbol configuration for the predefined WinStudio visualization contains the following entries:

GATcompact

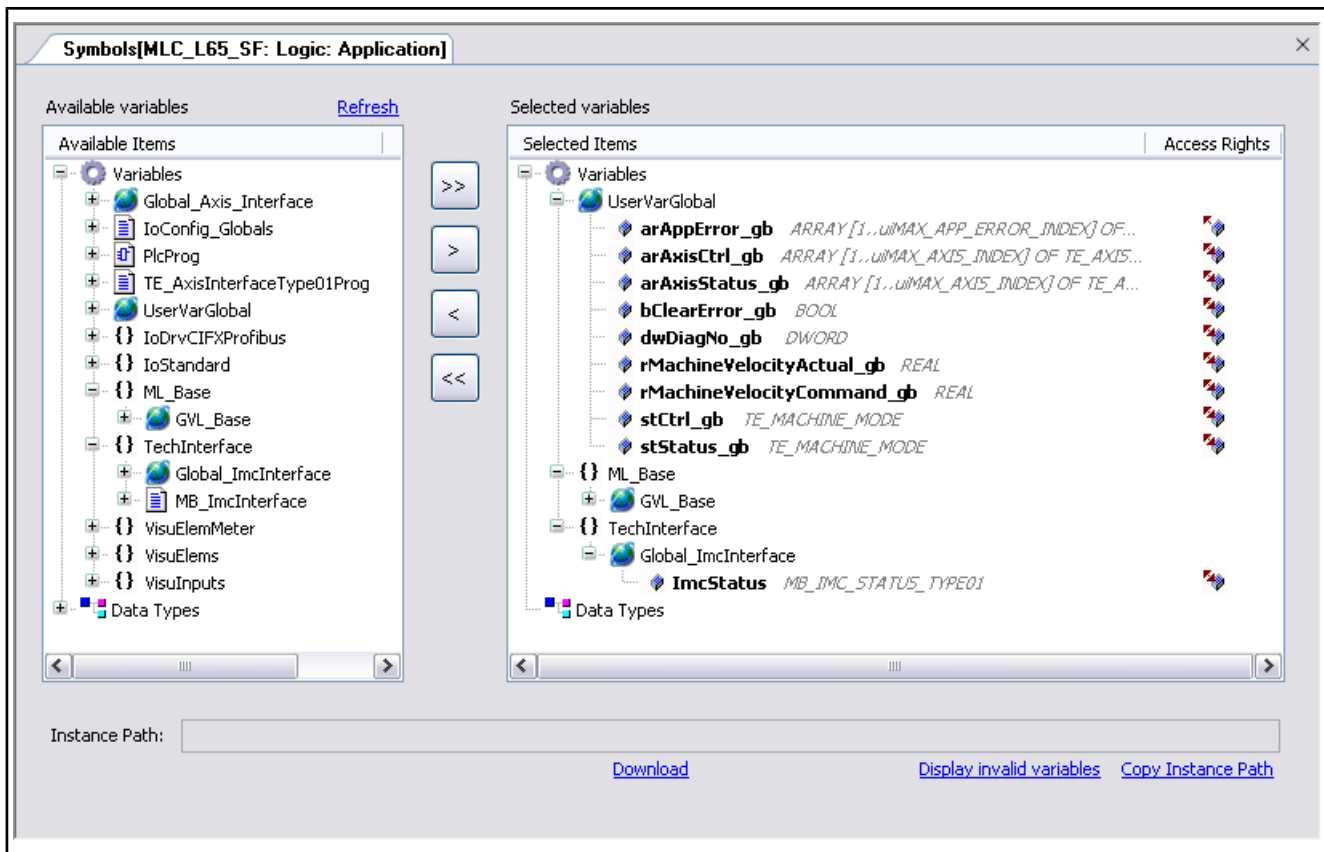


Fig.7-38: Symbol configuration of the predefined WinStudio visualization



Symbol entries for variables from global variable lists that come from a library must be reinserted in the symbol configuration after the respective library version has been modified.

Setting the control's IP address:

The control address is configured in the BR-WS driver properties dialog:

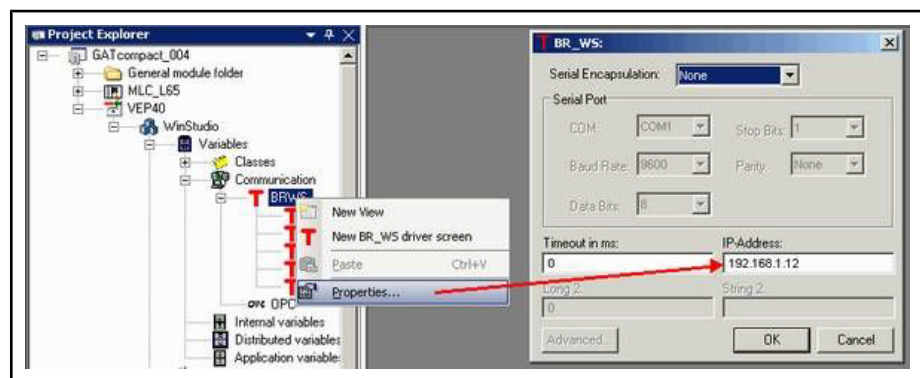


Fig.7-39: Setting the control's IP address

In order to structure the communication, the variables were distributed in various driver sheets:

1. **read cyclic fast:** The values of these variables are to be read cyclically at short intervals (e.g. diagnostic numbers, axis positions, ...)
2. **read only once:** These variables are read once after the visualization is started (e.g. axis names)

3. **read after switching phase:** These variables are read only once after a phase switch (e.g. if the axis was activated)
4. **read - write cyclic:** The values of these variables are to be both read and written (e.g. machine velocity, axis velocity for Setup mode, ...)
5. **write only:** These variables are only to be written through the visualization (trigger signals for clearing errors, ...)

To extend a driver sheet, the BR-WS browser can search the offline symbol file (*.XML) and the corresponding variable can be selected.

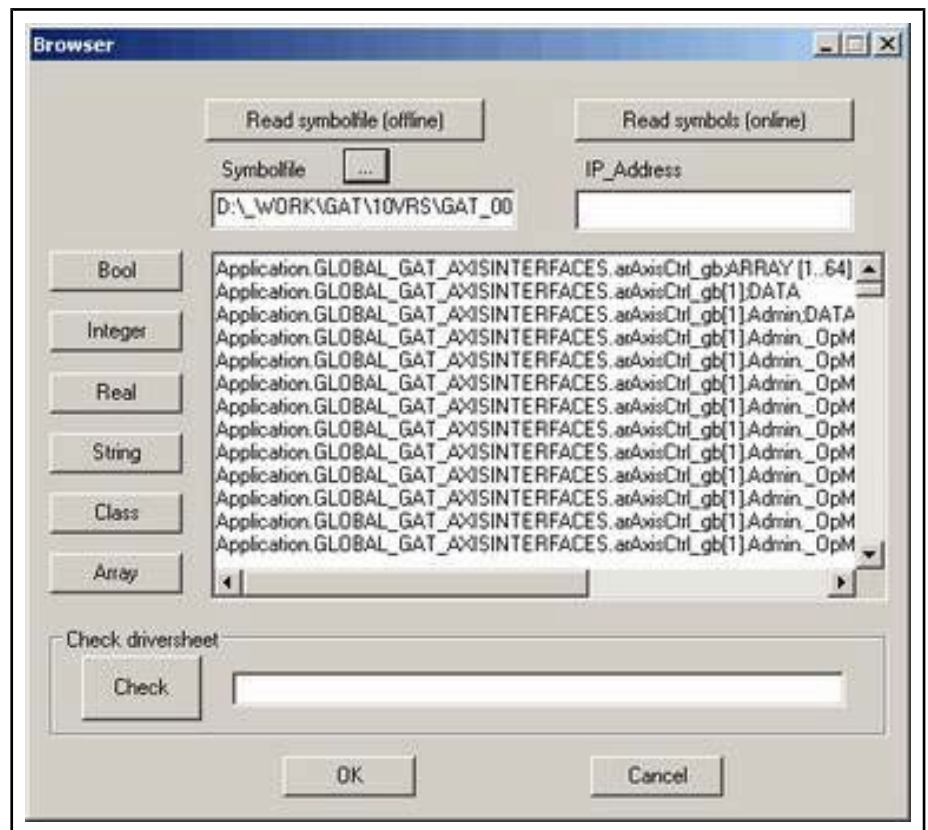


Fig. 7-40: Extending the driver sheet

Multiple Language Capability

Multiple language capability for the visualization has been realized with the translation functionality integrated into WinStudio.

The translation file is stored as a *.CSV table, which provides a column for each language.

The table is predefined to include multiple languages, but only the "German" and "English" columns are completely filled out. For the rest of the languages only the drive diagnostics are included.

GATcompact

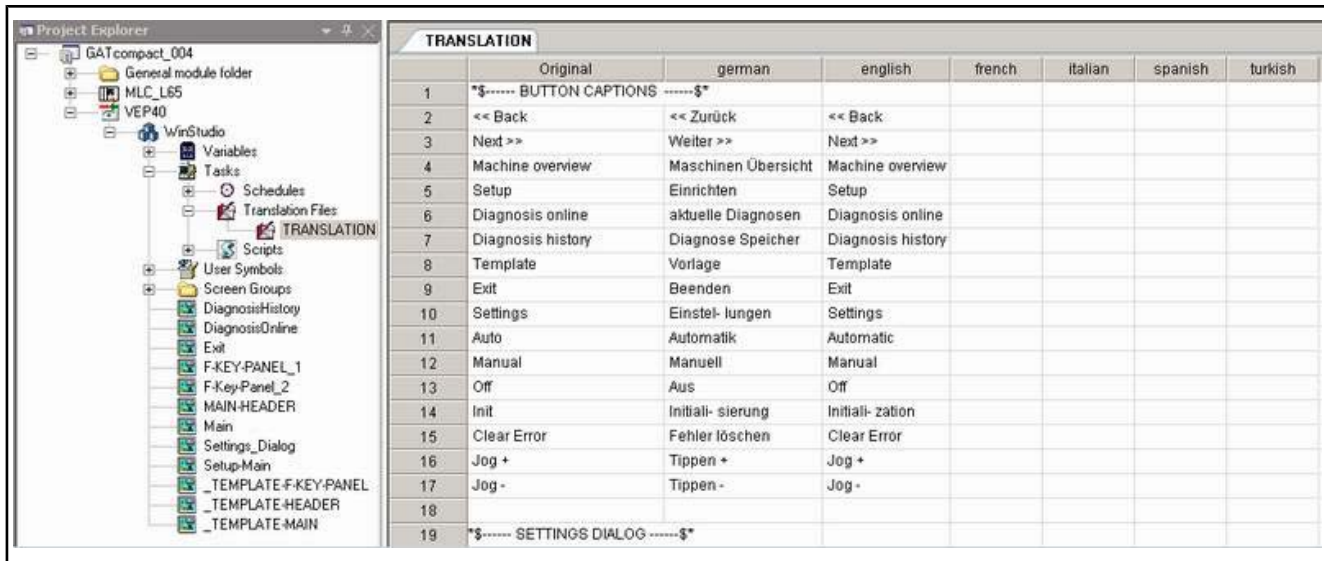


Fig.7-41: Editing the translation file

7.9 Interfaces to GATcompact

7.9.1 General Information

Most of the interface data for GATcompact is gathered together in the "UserVar-Global" variable list. Part of the data is made available in libraries.

In the following section, the data from libraries in the "Name of the variables" column is marked as such. All other data is located in the "UserVarGlobal" variable list.

7.9.2 Data for the State Machine

I/O type	Name	Data type	Description
E	stCtrl_gb	TE_MACHINE_MODE_CTRL	Command for machine operation mode
E	bSkipColdstartAtStartup_gb	BOOL	When starting the GATcompact program, a switch is made into parameterization mode (P2) to carry out the parameterization required for the application. This phase switch can be skipped with TRUE if the application does not require parameterization in P2.
A	stStatus_gb	TE_MACHINE_MODE_STATUS	Feedback regarding the current machine operation mode

E Input for writing
A Output for reading

7.9.3 Data for Error Handling

I/O type	Name	Data type	Description
E	bClearError_gb	BOOL	TRUE = Global clear error
E	bDisableErrorHandling_gb	BOOL	TRUE = In case of error, the process does not switch into the ERROR state The error display continues and clearing errors is still possible.
A	bError_gb	BOOL	TRUE = Global error has occurred

GATcompact

I/O type	Name	Data type	Description
A	dwDiagNo_gb	DWORD	Diagnostic number according to IndraLogic XLC-/ IndraMotion MLC specification DiagNo = 16#F0AE xxxx: Application error (e.g. safety door is open) DiagNo = 16#F0FB 0000: FB error DiagNo = 16#F0AA xxxx: GAT internal error
A	enErrorID_gb	ERROR_CODE	Detailed information if DiagNo = 16#F0FB 0000 / 16#F0AA xxxx
A	stErrorIdent_gb	ERROR_STRUCT	Detailed information if DiagNo = 16#F0FB 0000 / 16#F0AA xxxx
A	strDiagMessage_gb	STRING(60)	Diagnostic text of the control / axis
A	arAppError_gb	ARRAY OF BOOL	Application-specific error (e.g. safety door is open)

E Input for writing
A Output for reading

7.9.4 Axis Data

I/O type	Name	Data type	Description
E	arAxisCtrl_gb	ARRAY OF TE_AXIS_CONTROL_TYPE01	Commands for the axes
A	arAxisStatus_gb	ARRAY OF TE_AXIS_STATUS_TYPE01	Feedback from the axes
I/O	AxisData (GVL_Base from the ML_Base library)	ARRAY OF ML_AXISDATA_SM	Firmware structure with axis inputs and outputs

E Input for writing
A Output for reading
I/O Structure with inputs and outputs



Elements for arAxisCtrl_gb are written in the "Machine" FB and for this reason they can no longer be used externally.

7.9.5 Control Data

I/O type	Name	Data type	Description
E	ImcCtrl (Global_ImcInterface from the ML_TechInterface library)	MB_IMC_CONTROL_TYPE01	Commands for the control
A	ImcStatus (Global_ImcInterface from the ML_TechInterface library)	MB_IMC_STATUS_TYPE01	Feedback from the control

E Input for writing
A Output for reading



Elements for ImcCtrl_gb are written in the "Machine" FB and for this reason they can no longer be used externally.

GATcompact

7.10 How To: Typical User Activities

7.10.1 Access to Axis Data

In the FB "Machine", the structure **Axes** is defined as the central access point for accessing axis data. All relevant data for an axis can be addressed using the Axes structure.

The structure is designed such that the axis data is accessed using the name of the axes. This generates program code that is easily readable.

The following data are available below "Axes":

- Axes.Name.AxisData => Axis data structure: Status data for the firmware and access to the axis data container
- Axes.Name.Ctrl => Control structure for the AxisInterface
- Axes.Name.Status => Status structure for the AxisInterface
- Axes.Name.DV => Access to direct variables (function variables)
- Axes.Name.AxisRef => Axis reference structure for FB calls (e.g. MB_ReadParameter)

Array Axes.arAll[] can also be used to access the data listed above. That can be advantageous for loop programming (for an example, see actModeOff_entry).

In both versions, references to the same storage locations are accessed.

Example: the accesses

- Axes.Name.AxisData.rActualPosition_i
- Axes.arAll[Name.AxisNo].AxisData.rActualPosition_i

are absolutely equivalent and deliver the same result.

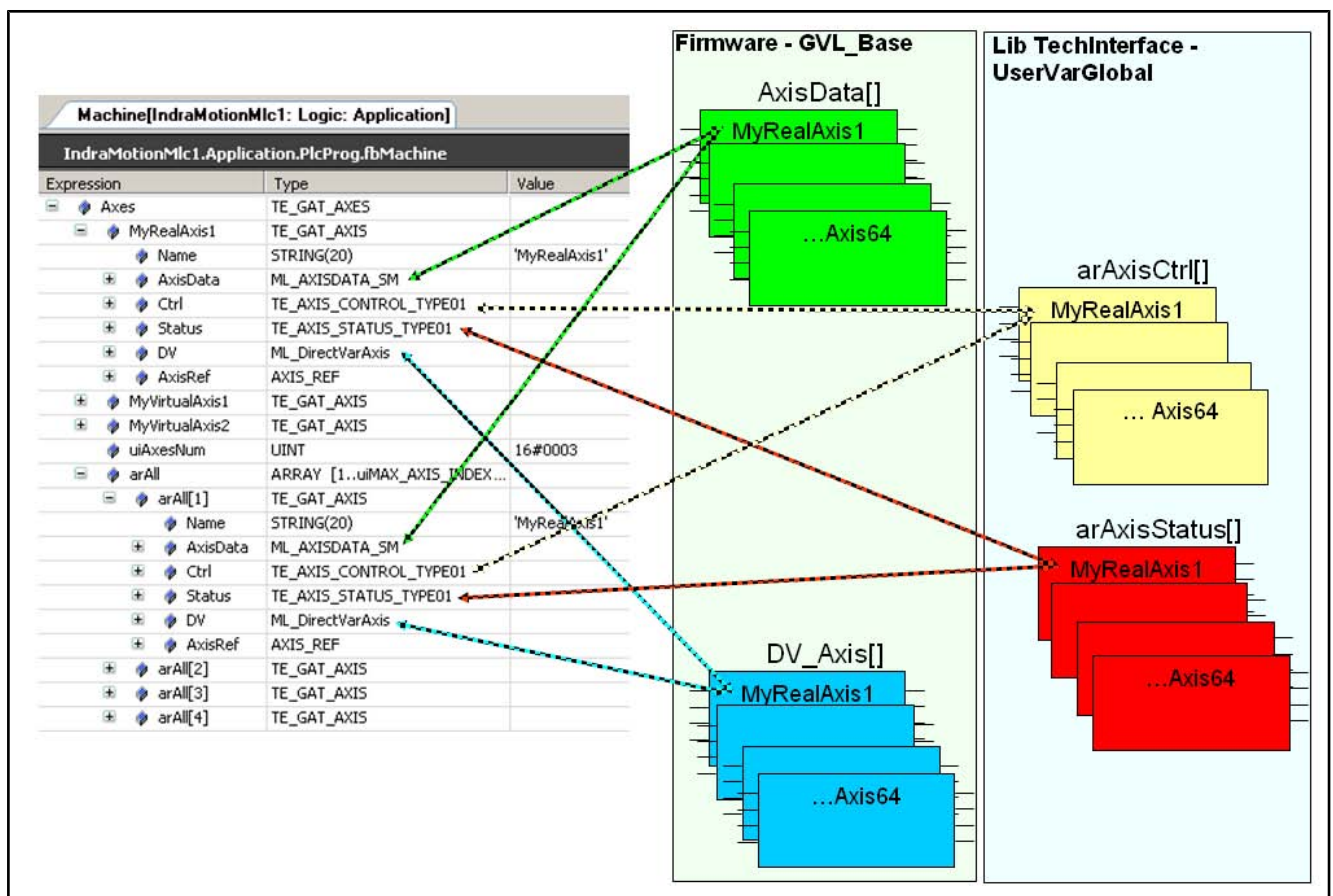


Fig. 7-42: Overview of axis data

7.10.2 Adding an Axis

The following steps are required for adding an axis:

- Add the axis into the IndraWorks project ("Drag to the SERCOS node...")
- When using the SystemOverview_All_Axis visualization: add a frame for the new axis and set the reference to the new axis in the properties (see also [chapter "System_Overview" on page 142](#)).
- Add the axis using the GAT wizard (see the figure below)

GATcompact

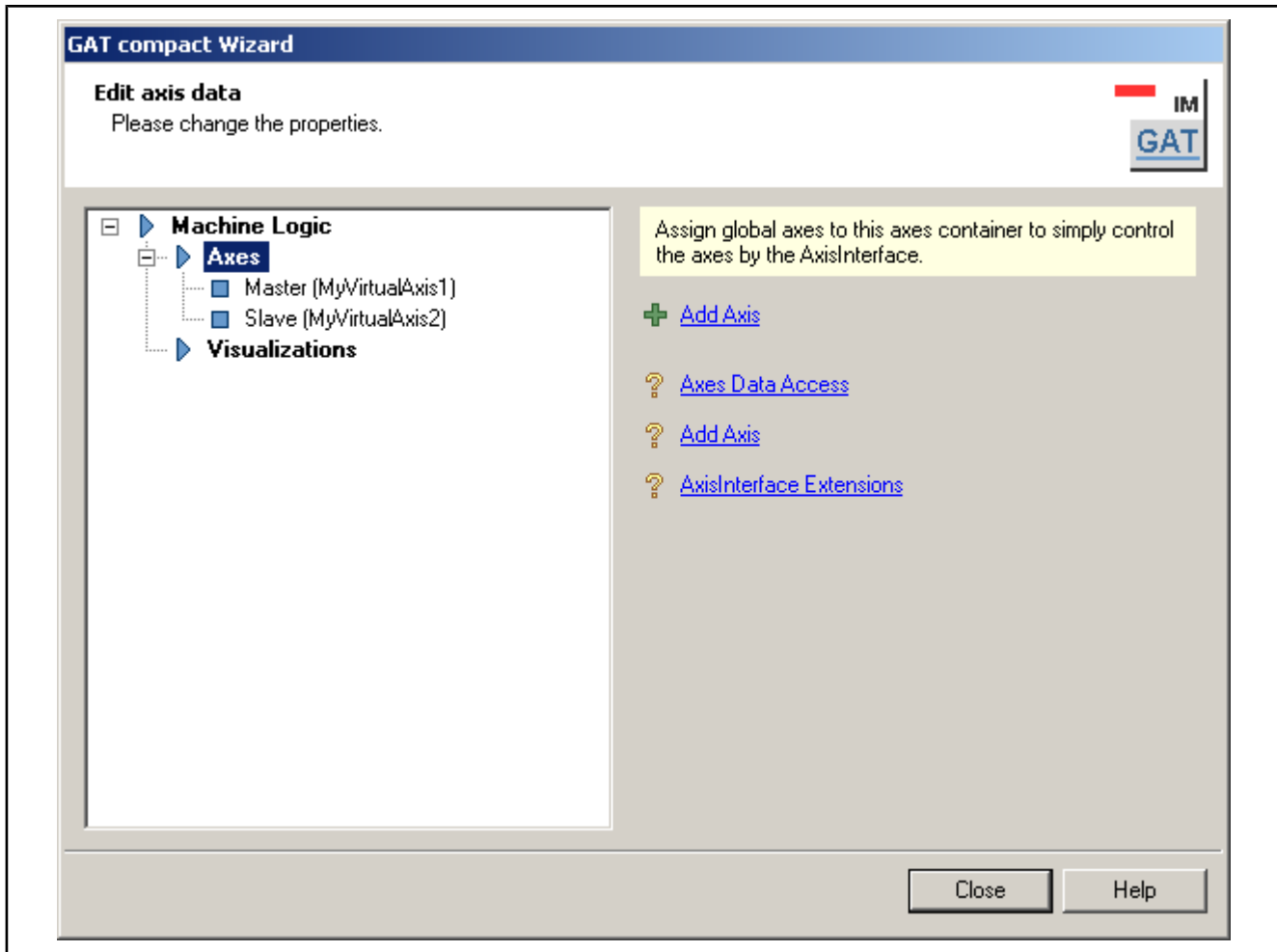


Fig. 7-43: Adding an axis using the GAT wizard

7.10.3 Removing/renaming an Axis



The wizard changes only the name in the axis declaration. If the axis is already being used in the implementation or internal visualization, the new axis name must be changed at these positions using "Find/Replace".

The following steps are required for removing/renaming an axis:

- Removing an axis from/renaming an axis in the IndraWorks project
- When using the `SystemOverview_All_Axis` visualization: remove the frame for the deleted axis and change the reference to the axis in the properties.
- Remove/rename the axis using the GAT wizard (see the figure below).

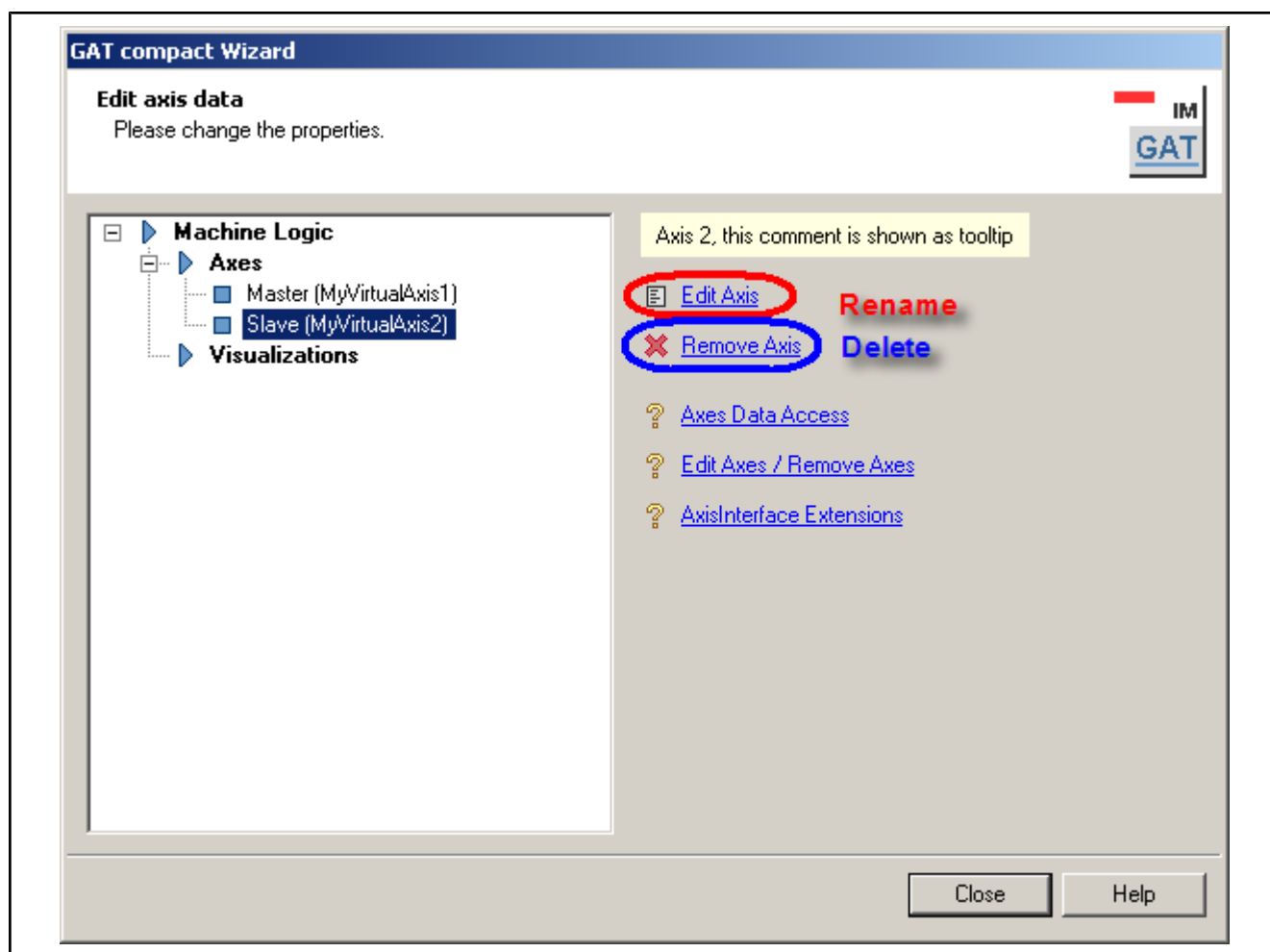


Fig. 7-44: Removing/renameing the axis with the GAT wizard

GATcompact

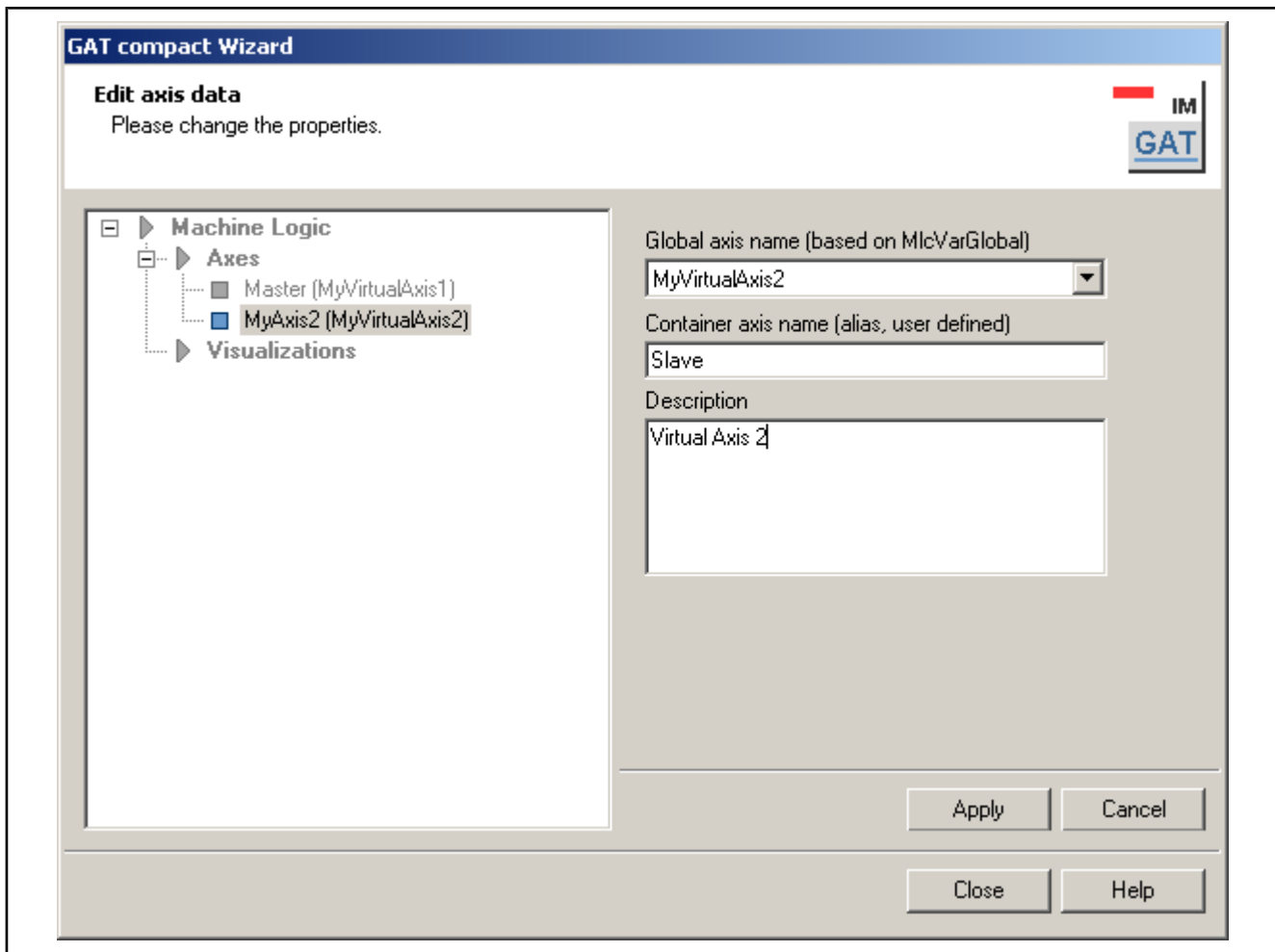


Fig. 7-45: Removing/renaming the axis with the GAT wizard

7.10.4 Adding an Operation Mode

GATcompact has the following predefined operation modes:

- OFF
- MANUAL
- AUTO

As an example, in the following section, the **"CLEANING"** operation mode (cleaning the machine) shall also be implemented.

To do this, the following steps are required:

- TYPE TE_MACHINE_MODE (GATcompact folder): extend with the **MODE_CLEANING** element
- FB "Machine":
 1. Copy the transition traAuto, rename it as traCleaning and in tra-Cleaning, replace MODE_AUTO with MODE_CLEANING; result: (Ctrl=MODE_CLEANING;)
 2. Copy the transition traLeaveAuto, rename it as traLeaveCleaning and in traLeaveCleaning, replace MODE_AUTO with MODE_CLEANING; result: NOT(Ctrl=MODE_CLEANING;)
 3. Add a "ModeCleaning" branch parallel to ModeAuto
 4. Use the transitions listed above at ModeCleaning

5. Add an input action `actModeCleaning_entry()` at `ModeCleaning`. In the action, add the following code as the first line: `Status:=MODE_CLEANING;`

6. After the transitions, add the same jump targets as for `ModeAuto`.

Results of the extension:

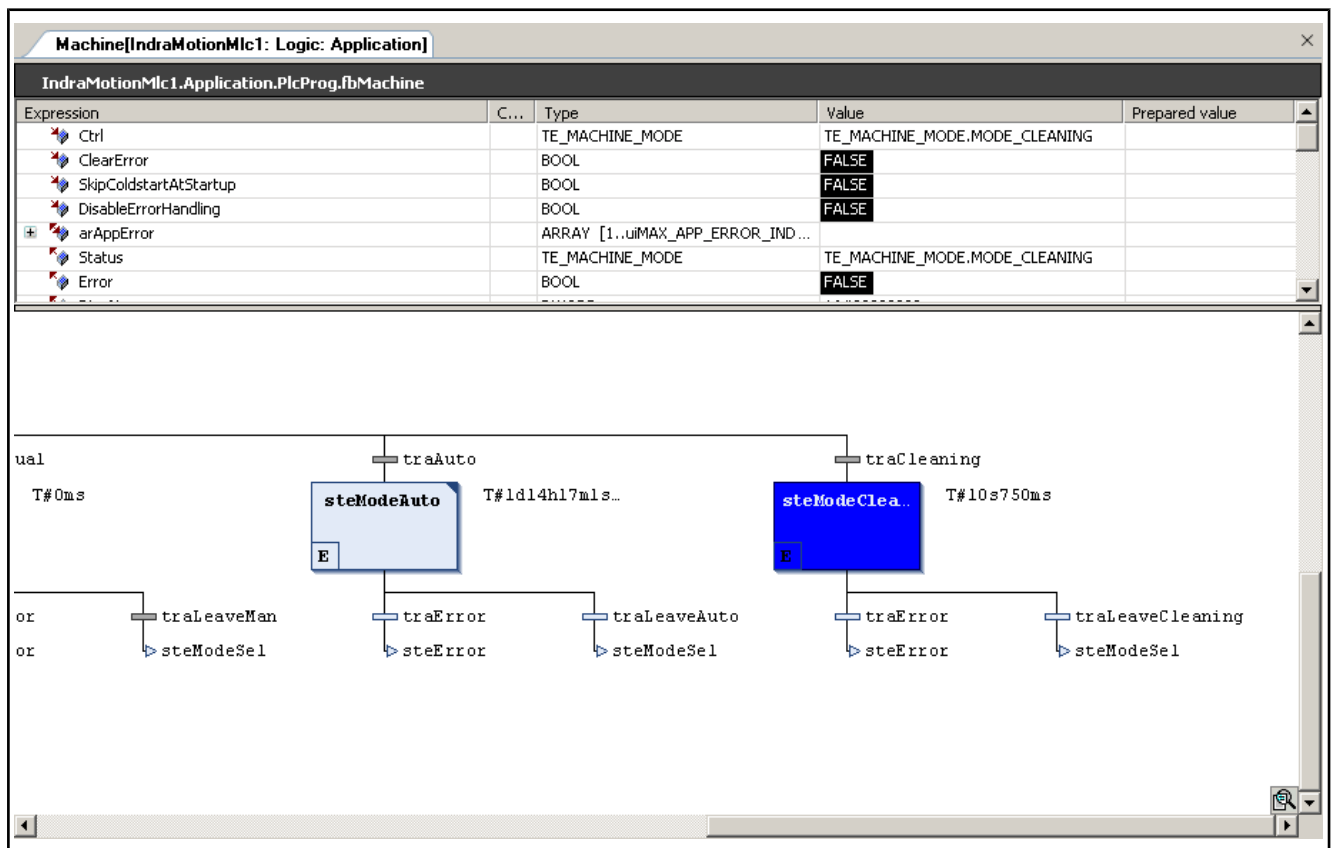


Fig. 7-46: Result after operation mode has been added

The desired behavior can be implemented in the action `actModeCleaning_entry()` or in an optional action `actModeCleaning_active()`.

7.10.5 Processing in MotionTask

For processing in the `MotionTask`, the FB "Machine" can be extended to include the method "`mMotionCyclic`". All of the FB variables (e.g. central axis entry point `Axes...`) can be accessed from this method.

It is important that the axes controlled here are updated in the `MotionTask` (see [chapter 7.6.2 "Adding a MotionTask" on page 135](#)).

7.10.6 AxisInterface Extensions

GATcompact allows almost any extension of the `AxisInterface` structures. Additional substructures can also be added and the existing substructures extended.

As an example, the following structure elements are programmed as user extensions in the `AxisCtrl` structure.

- `SetupMode`: additional substructure in the `AxisCtrl` structure
 - `Enable`: enable Setup mode
 - `JogPlus`: Jog +
 - `JogMinus`: Jog -

GATcompact

- Vel: Jog / reference travel velocity
- Accel: Jog/reference travel deceleration/acceleration
- Homing: Activates ModeHoming|SetAbsRef|ModesPosAbs depending on axis type/configuration
- SyncMode: Extension of the existing substructure in the AxisCtrl structure
 - AngleOffset: Phase offset start of table -> parameter A-0-2730 is described with this value
 - PhaseShift: Offset relative to the master axis before the gear (PhaseOffset: Offset relative to the master axis after the gear)
 - PhaseShiftVel: Velocity for adjusting the PhaseShift
 - PhaseShiftAcc: Acceleration for adjusting the PhaseShift
 - SyncVelocity: Phase shift velocity -> parameter A-0-2790 is described with this value
 - SyncAcceleration: Phase shift acceleration -> this value is written on parameter A-0-2791

As an example, the following structure elements are programmed as user extensions in the AxisStatus structure:

- SetupMode: additional substructure in the AxisStatus structure
 - EnableAck: Setup mode is active
 - Homingack: Homing is finished.
- SyncMode: extension of the existing substructure in the AxisStatus structure
 - AngleOffsetDone: Transfer of phase offset start of table -> parameter A-0-2730 was described
 - PhasingMasterDone: PhaseShift was completed.
 - SyncVelocityDone: Transfer of synchronization velocity -> parameter A-0-2790 was transferred.
 - SyncAccelerationDone: Transfer of synchronization acceleration -> parameter A-0-2791 was transferred.

The code for these extensions can be found in the action of the TE_AxisInterface() block in the GAT^{compact}/AxisInterface folder. The related structures can be found in the GAT^{compact}/AxisInterface/Type/Control and GAT^{compact}/AxisInterface/Type/Status folders.

Users can add their own extensions (to do so, see [chapter 6.7 "AxisInterface User Extensions" on page 88](#)).

When new error codes are introduced, a diagnostic message can be assigned for the VI logic visualization by extending the structure "arAXIS_DIAG_MESSAGES" in the GatConstGlobal variable list in the folder \GAT compact\GAT-Utilities.

7.11 GAT^{compact} Brief Overview

7.11.1 Structure Overview

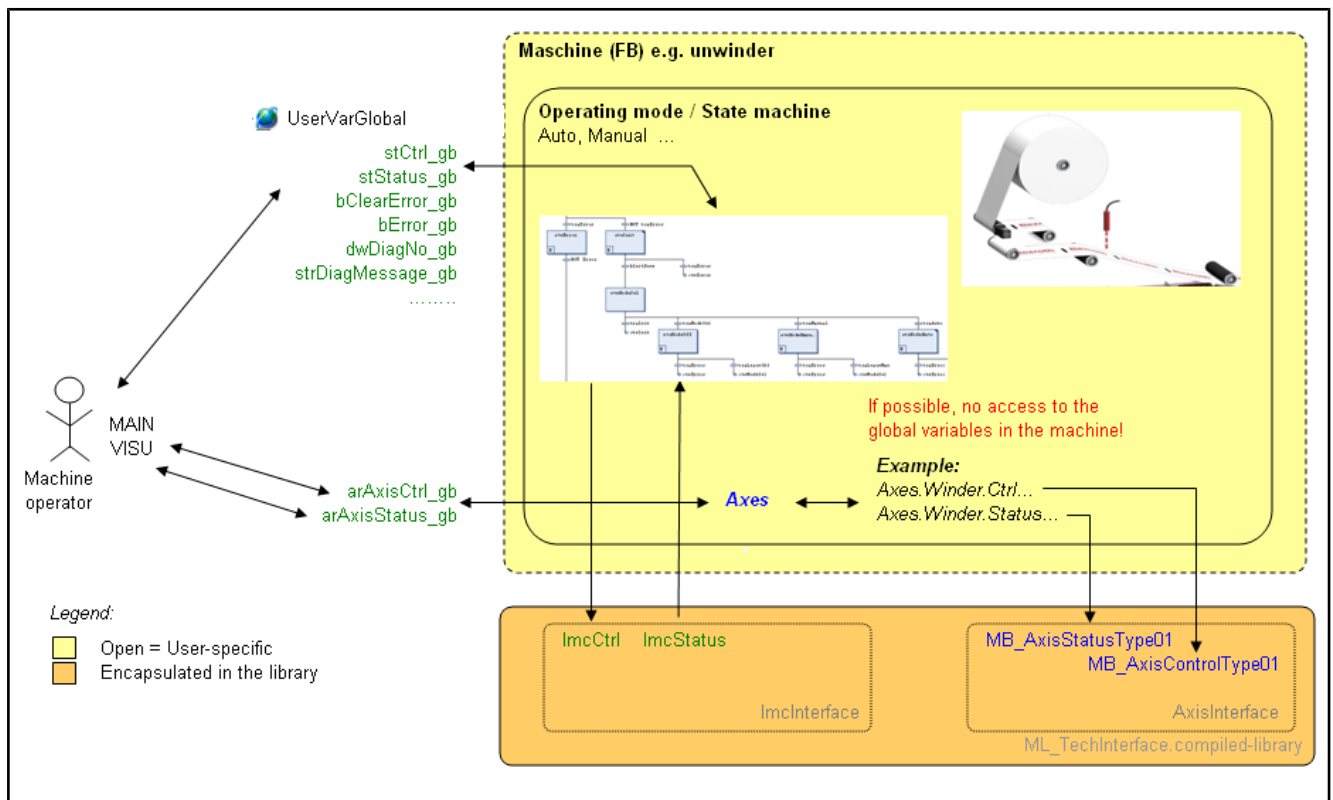


Fig.7-47: GAT^{compact} structure overview

7.11.2 Overview of the Machine's Most Important Global Variables

See the global variable list "UserVarGlobal":

Variable	Description	Access
stCtrl_gb	Machine control (operation mode selection, default operation mode)	RD/WR
stStatus_gb	Machine status (current operation mode)	RD
bClearError_gb	Clearing an Error	RD/WR
bDisableErrorHandling_gb	In case of error, do not switch into the error state	RD/WR
bSkipColdstartAtStartup_gb	When starting the program, a "cold start" (switch into P2) is not carried out.	RD/WR
bError_gb	TRUE indicates an error.	RD
dwDiagNo_gb	Diagnostic number according to IndraLogic XLC-/IndraMotion MLC documentation	RD
enErrorID_gb	Detailed information if dwDiagNo_gb = 16#F0FB 0000 (function block error)	RD
stErrorIdent_gb	Detailed information if dwDiagNo_gb = 16#F0FB 0000 (function block error)	RD

GATcompact

Variable	Description	Access
strDiagMessage_gb	Control or axis diagnostic text	RD
arAppError_gb	Array of BOOL with user-specific machine error flags	RD

RD

Read:

WR

Write

7.11.3 Overview of the Most Important Global Variables of the Axes

See the global variable list "UserVarGlobal":

Variable	Description	Access
arAxisCtrl_gb	Array with the AxisInterface command data	RD/WR
arAxisStatus_gb	Array with the AxisInterface status data	RD

RD

Read:

WR

Write

7.11.4 Overview of the Machine's Methods and Actions

Action/method	Call	Related task	Description / example
actCyclic_active	In every state	PlcTask	Comprehensive logic, e.g. safety technology
actError_entry	For a cycle during the transition into the MODE_ERROR state	PlcTask	Error reaction, e.g. switching into ModeAH (full stop with power on)
actInit_entry	For a cycle during the transition into the MODE_INIT state	PlcTask	Preparing the MODE_INIT state, e.g. switching off all axes in order to enable the phase switch into P2.
actInit_active	Only in the MODE_INIT state	PlcTask	Switching into parameterization mode P2, possible description of parameters in P2. When exiting the MODE_INIT, switch into operating mode (bb), possible description of parameters when arriving in bb.
actModeOff_entry	For a cycle during the transition into the MODE_OFF state	PlcTask	Preparing the MODE_OFF state, usually switching off all axes (ModeAb).
actModeManual_entry	For a cycle during the transition into the MODE_MANUAL state	PlcTask	Preparing the MODE_MANUAL state, usually stopping all axes (ModeAH).
actModeAuto_entry	For a cycle during the transition into the MODE_AUTO state	PlcTask	Preparing the MODE_AUTO state, e.g. activating a synchronous operation mode (e.g. ModeSyncVel) for all slave axes and enabling a master axis (e.g. ModeVel)
actModeAuto_active	Only in the MODE_AUTO state	PlcTask	e.g. monitoring in Automatic mode, calling process controller function blocks.
FB_init	Once before starting/resetting a PLC program	No specific task assignment	Internal system use, may not be modified

7.11.5 Access to Axes within the Machine

Always use **Axes.<axisname>**

GATcompact

Example:

Program:

```
// Command axis in velocity control mode
Axes.MyAxis1.Ctrl.Admin._OpMode.en := ModeVel;

// If axis has reached the velocity
IF (Axes.MyAxis1.Status.PtrAxisData^.Axis_InVelocity = TRUE) THEN;
```

8 GAT with Central State Machine

8.1 General Information

The Generic Application Template is a software tool that is integrated into IndraWorks for creating PLC applications. Based on a program framework in accordance with IEC61131-3, together with a dialog-based wizard it can be used to develop applications quickly and easily. It also supports the specific features of Motion applications such as the handling of axes.



The GAT with central operation mode SFC requires an IndraMotion MLC control. The IndraLogic XLC control is not currently supported by the GAT.

8.2 Overview

8.2.1 Basic Idea

Using the GAT is based on the idea that an application runs on a machine that coordinates one or more functional units in a various operation modes.

The following example is used to explain this idea.

An application will be created for a simple packaging machine that consists of a infeed unit, a sorting unit and a packaging unit (see the following figure).

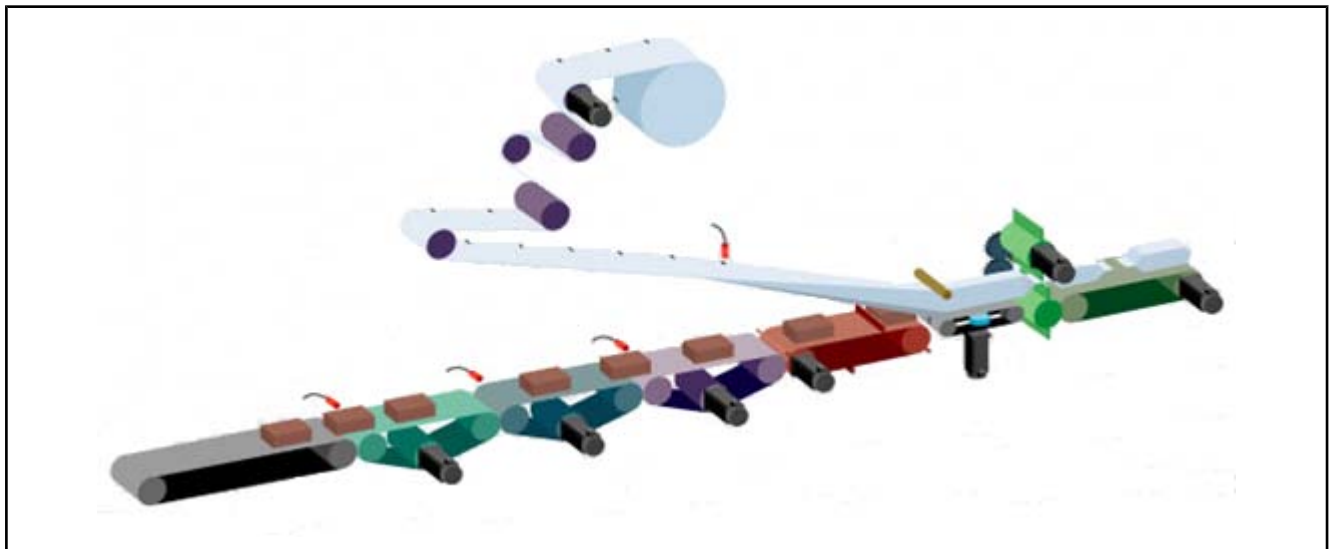


Fig. 8-1: Example: horizontal form fill and seal machinery

In order to apply the GAT, this packaging machine is divided into 3 functional units (modules).

GAT with Central State Machine

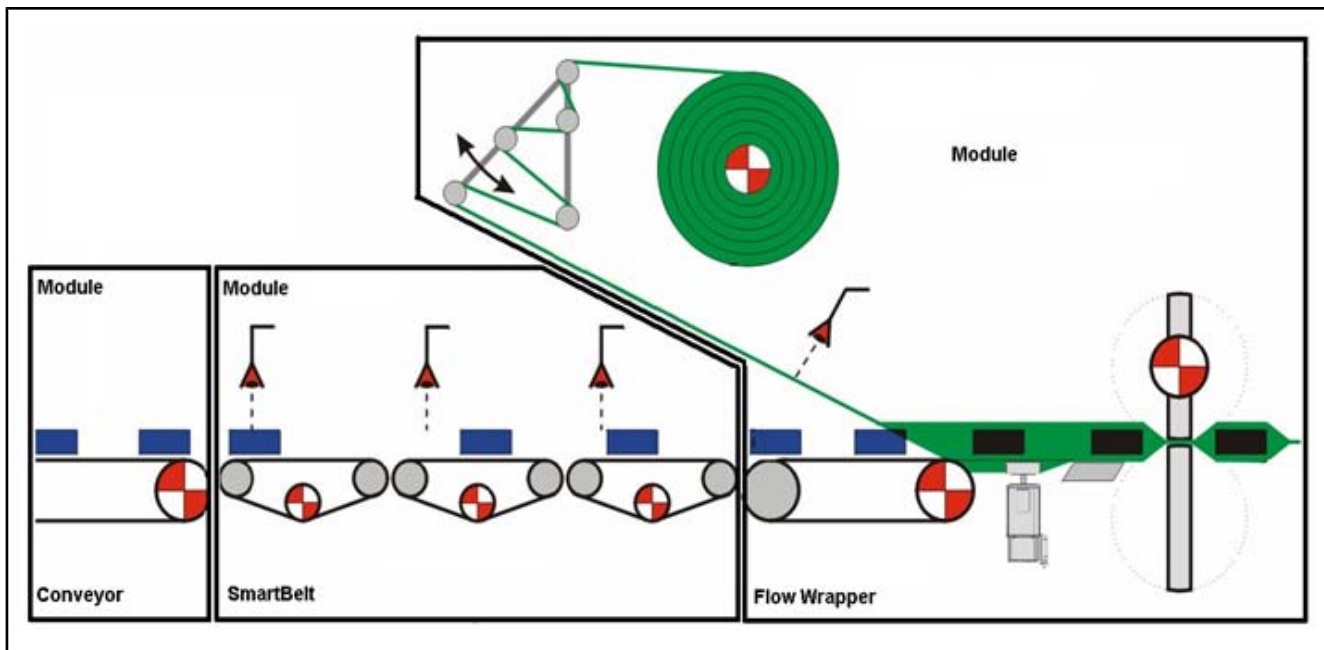


Fig. 8-2: Functional units of the horizontal form fill and seal machinery

The result is the following software architecture, which can be implemented directly in the real machine's corresponding software by using the GAT.

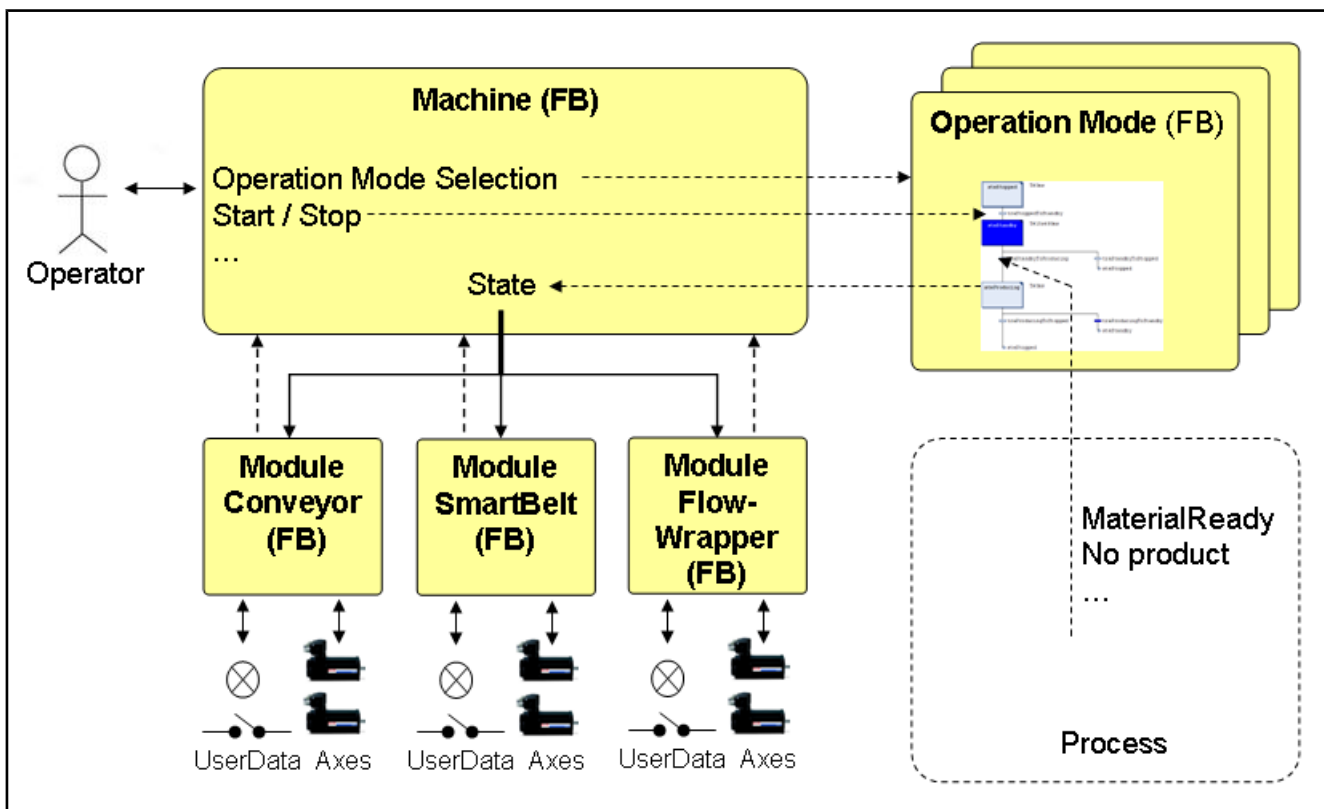


Fig. 8-3: Software architecture when using the GAT

8.2.2 Requirements for Use

Already presented to some extent in the previous section, the prerequisites for using the GAT are summarized again here.

GAT with Central State Machine

Before a Motion application is developed with the GAT, the user must clarify the following questions with respect to the application:

- Which operation modes are to be realized? Which states are required in each respective operation mode?
- Which functional units (modules) make up the machine? Which axes and I/Os are associated with each respective functional unit?

Using the information, the wizard can generate the code framework for the actual Motion application in a very short time.

The following section describes the parts of the GAT, how these parts work together and where the user has to make additions in order to realize the actual application.

8.3 First Steps - GAT with Central State Machine

8.3.1 Preparation

The GAT is integrated into the IndraWorks Suite installation package and is automatically installed.

The GAT can be used if a project was created with an IndraMotion MLC control.

The GAT wizard is opened using the GAT button  in the IndraWorks Toolbar.



The GAT wizard always refers to the active application in the IndraWorks project.

8.3.2 Creating a Simple Motion Application

General Information

This section describes how to create an application using the GAT wizard.

Topic

A machine is to consist of two functional units (modules). Starting from a stop state, in the first functional unit an axis should move continuously at 100 rev./min. in Automatic mode in a producing state. In the other unit, a second axis works operates phase synchronously to the axis in the first module during producing Automatic operation. The following figure provides a schematic representation of the tasks.

GAT with Central State Machine

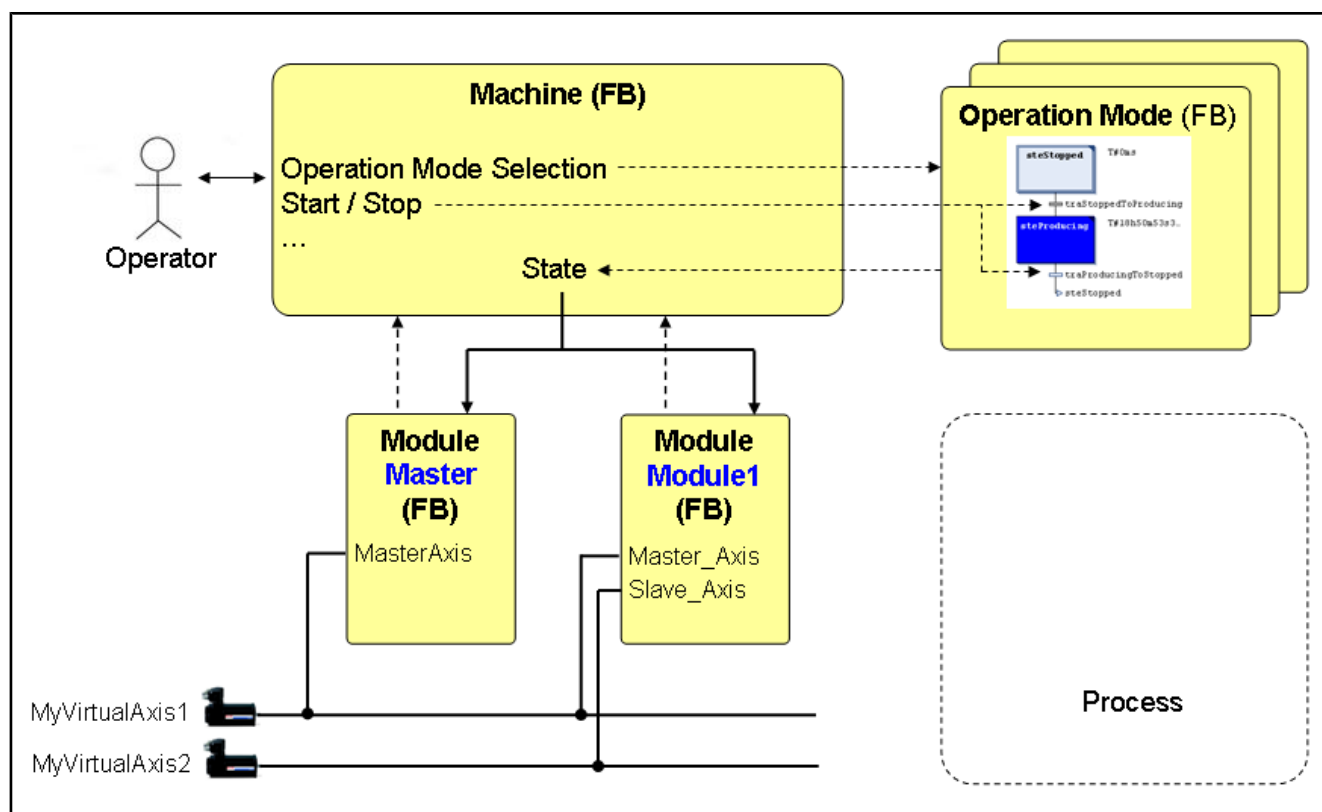


Fig.8-4: Schematic assignment of tasks

Operating Sequence

As described earlier, the GAT wizard starts. In the following dialog, select "GAT (central state machine)" and then click the **Finish** button.

GAT with Central State Machine

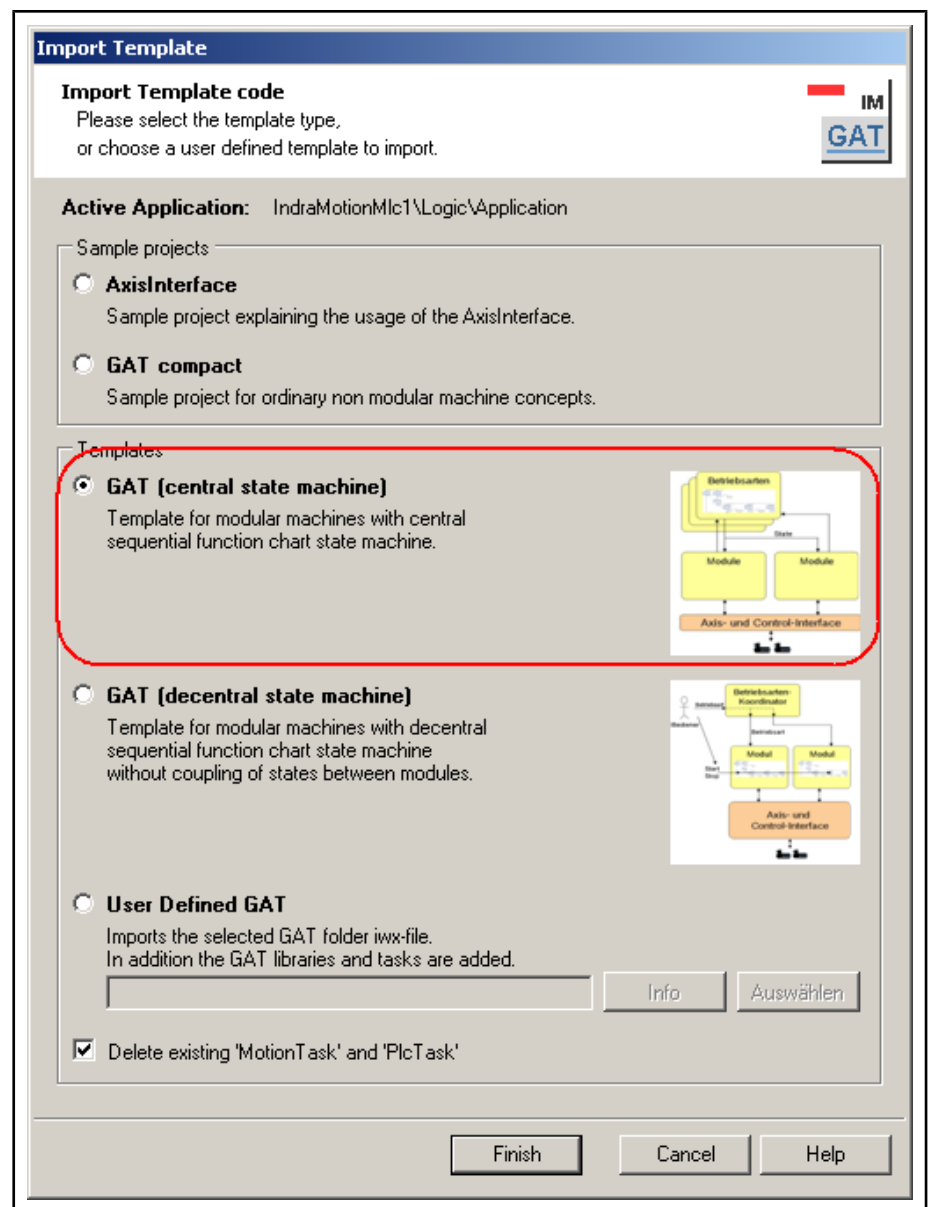


Fig. 8-5: Importing GAT with central state machine

By selecting "GAT with central state machine", a special project is imported, in which two global virtual axes (MyVirtualAxis1, MyVirtualAxis2) have already been configured in addition to a PLC program template.

An AUTO operation mode with the states "Stopped" and "Producing" is predefined in the GAT. Switching from the "Stopped" state to the "Producing" state occurs when the global Boolean variable "bStart_gb" is set (see the following figure; to define the global variable "bStart_gb" see the global variable list "Global_GAT_Machine" below the "Machine" folder).

GAT with Central State Machine

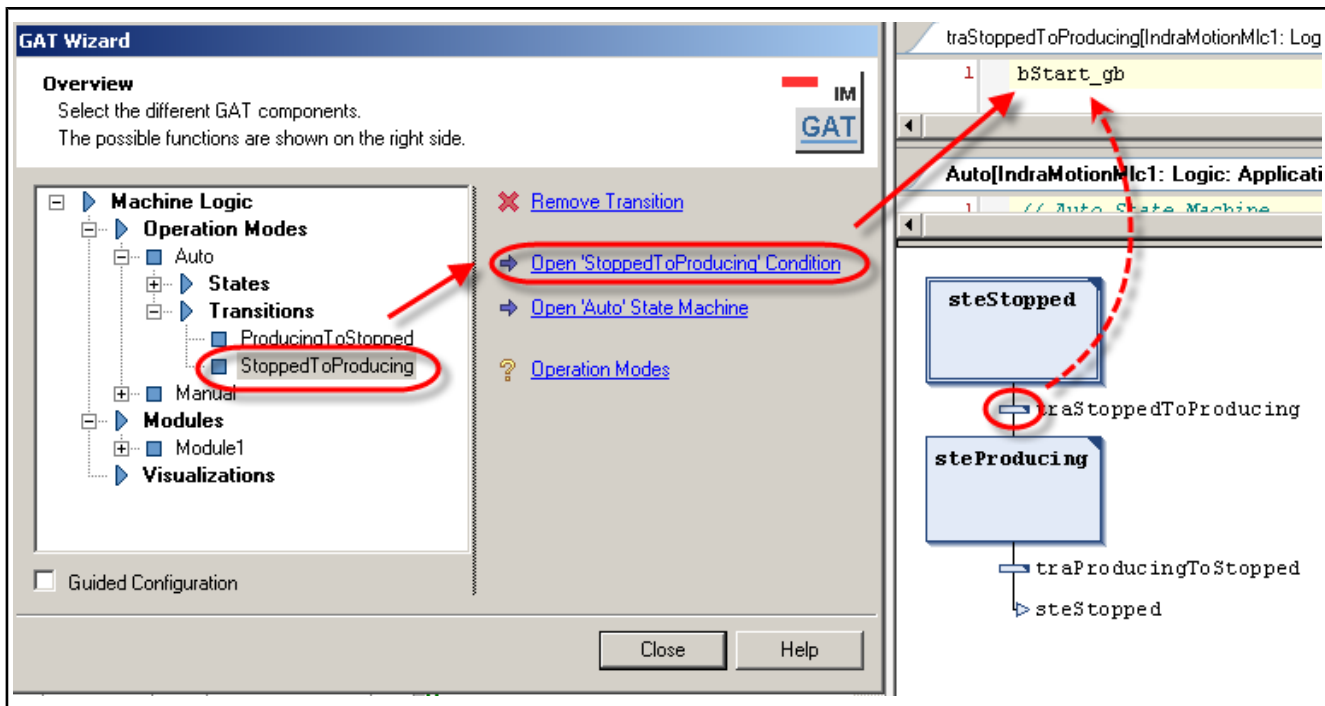


Fig.8-6: Editing transitions

Nothing else is required in order to realize the tasks with respect to defining the operation mode or states.

A module is already applied in the GAT (Module1). Another module is required to complete the topic. To do this, highlight the "Module" node in the GAT wizard directory tree and select "Add module" (see following figure).

GAT with Central State Machine

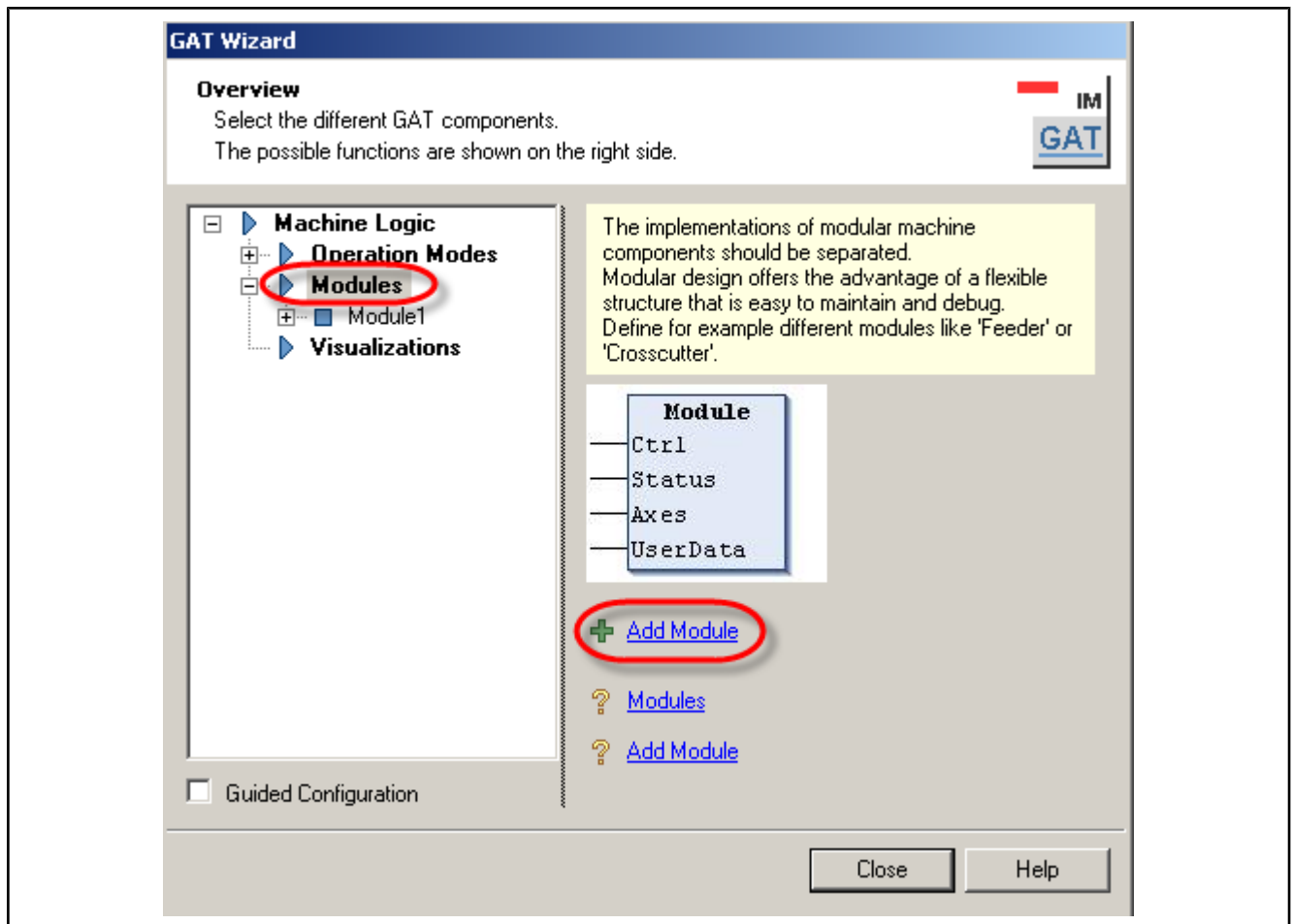


Fig. 8-7: Adding a module

In a subsequent dialog, the name of the new module is specified as "Module_Master".

GAT with Central State Machine

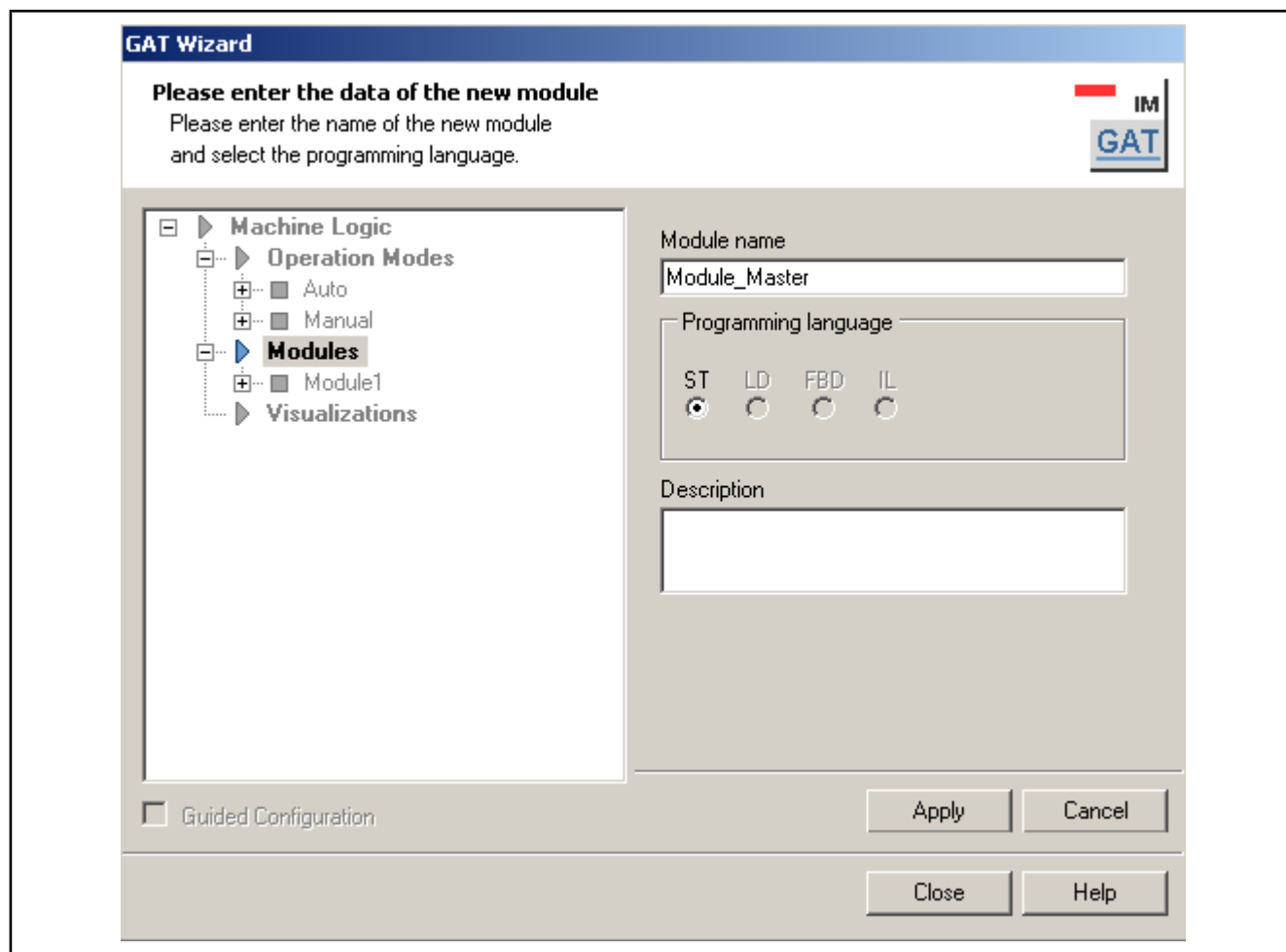


Fig. 8-8: Specifying a module name

Now the axis is to be created and implemented in the "Module_Master" module. This axis is to move continuously at 100 rev./min. in a producing state.

To do this, the wizard is used to create a new axis for the "Module_Master" module (see the following figure).

GAT with Central State Machine

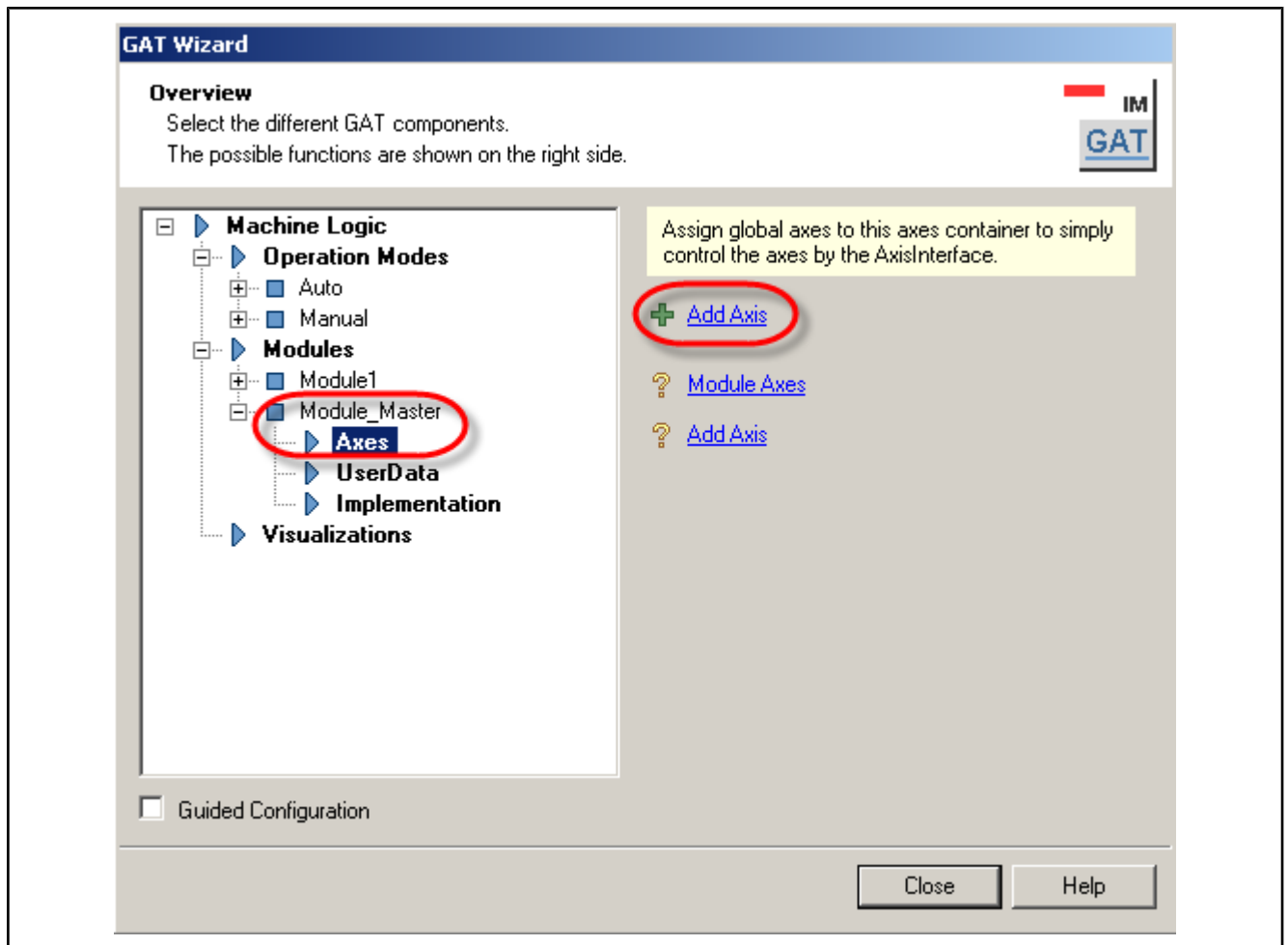


Fig. 8-9: Adding an axis with the GAT wizard

The axis in the module shall be called "MasterAxis". This module axis is assigned to the globally configured axis "MyVirtualAxis1".

GAT with Central State Machine

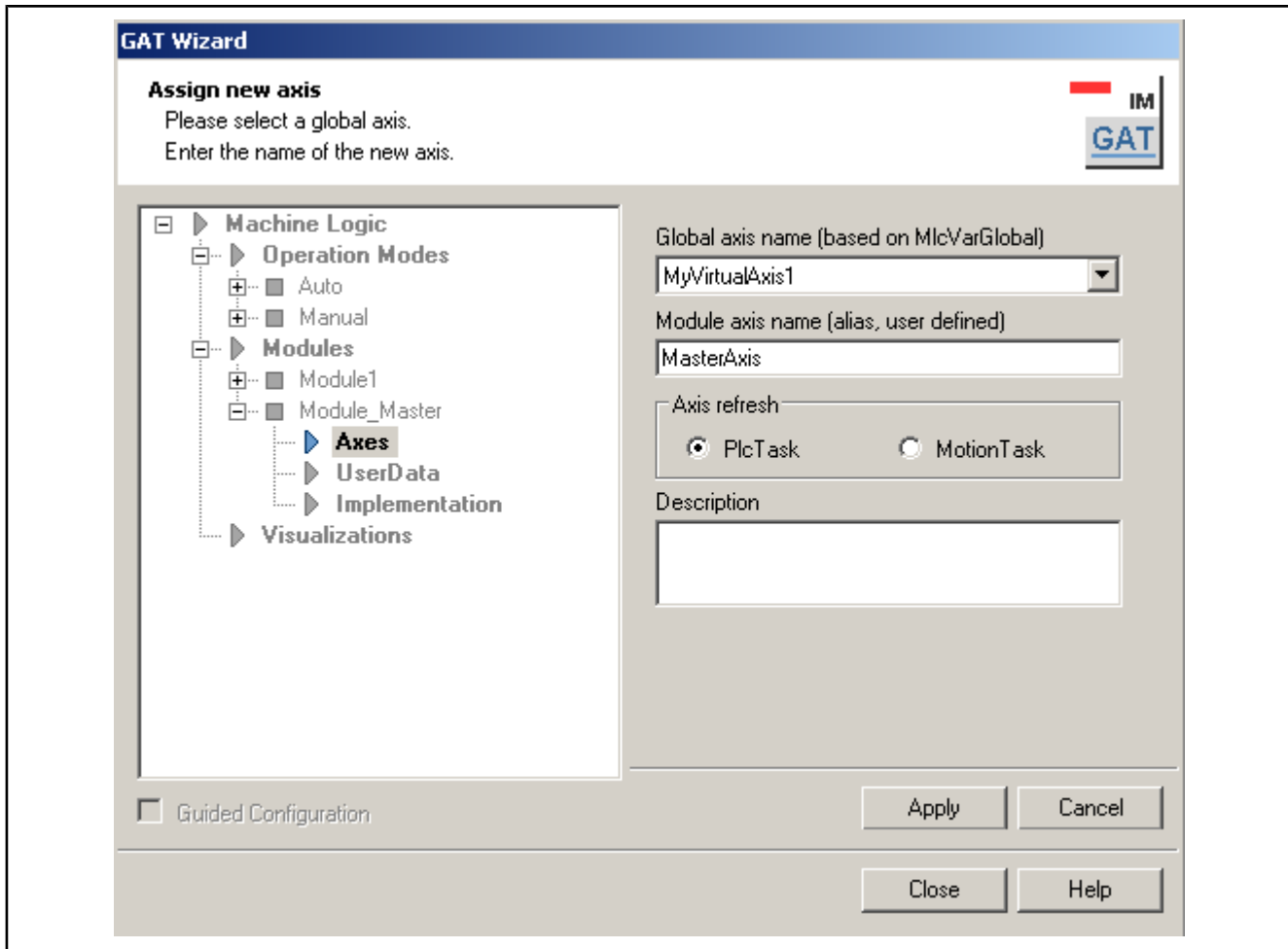


Fig. 8-10: Assigning the module axis to the globally configured axis

In the next step, the axis must be implemented in the module such that it moves in the "Producing" state at a constant velocity of 100 rev./min. (unit is specified by scaling the axis) and stops in the "Stopped" state. The code position in the module that corresponds with the state can be highlighted quickly and easily using the wizard. After the "Implementation" node below the module in the wizard directory tree is selected, all of the links in the implementation are displayed (see the following figure). The lines of code found on the right side of the figure must be entered in the result.

GAT with Central State Machine

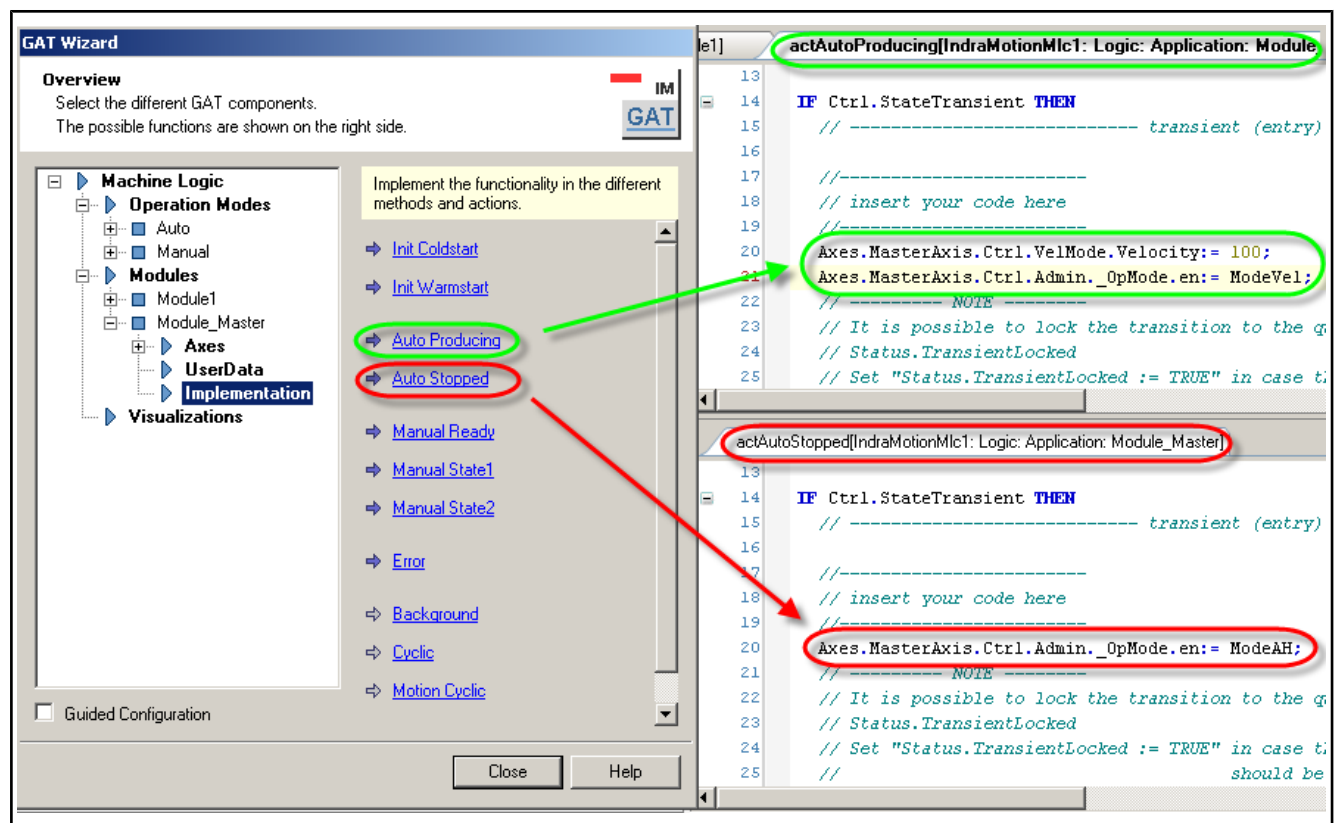


Fig.8-11: Implementing the code in the states' actions

Referencing an axis within a module is always done using "Axes". In every module, "Axes" is the name of the axis container. Enter a "." (period) after "Axes" to display all of the axes contained in the axis container (see the following figure). All I/O data for the module can be accessed in the same way using "UserData".

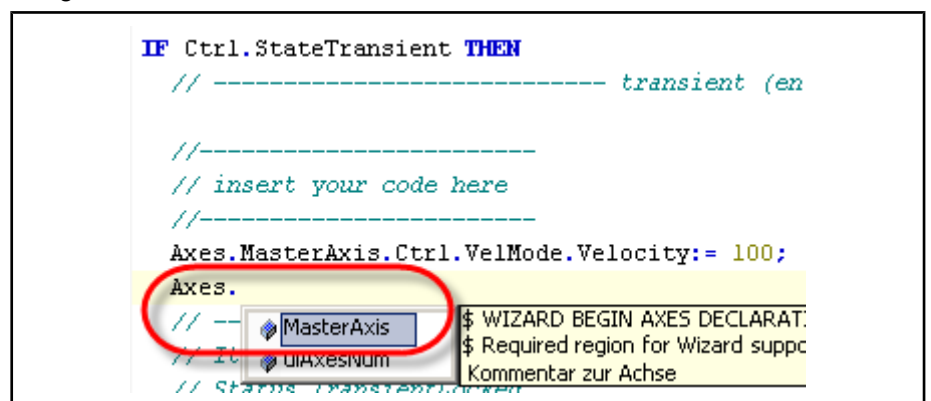


Fig.8-12: Commanding axes and using Intellisense

Further, by entering the period operator after the axis name, all elements that contain data regarding this axis are displayed for selection (see the following figure). To move to the next level, enter a period after the name of the previous level.

GAT with Central State Machine

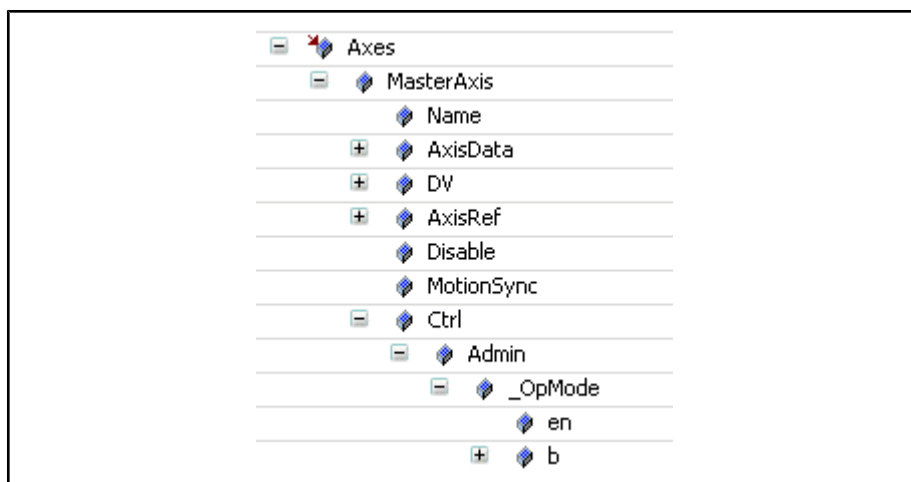


Fig.8-13: Available structure elements for an axis

Use the same procedure to create module axes "Master_Axis" and "Slave_Axis" for the second module "Module1".

In the original "Module1" module, there are already two module axes, "MyAxis1" and "MyAxis2". The globally configured axes with the same names are assigned to them, respectively. Using "Edit Axis" (see the following figures), the names of the module axes must be modified such that the globally configured axis "MyVirtualAxis1" is assigned to the module axis "Master_Axis" and the global axis "MyVirtualAxis2" is assigned to the module axis "Slave_Axis".

GAT with Central State Machine

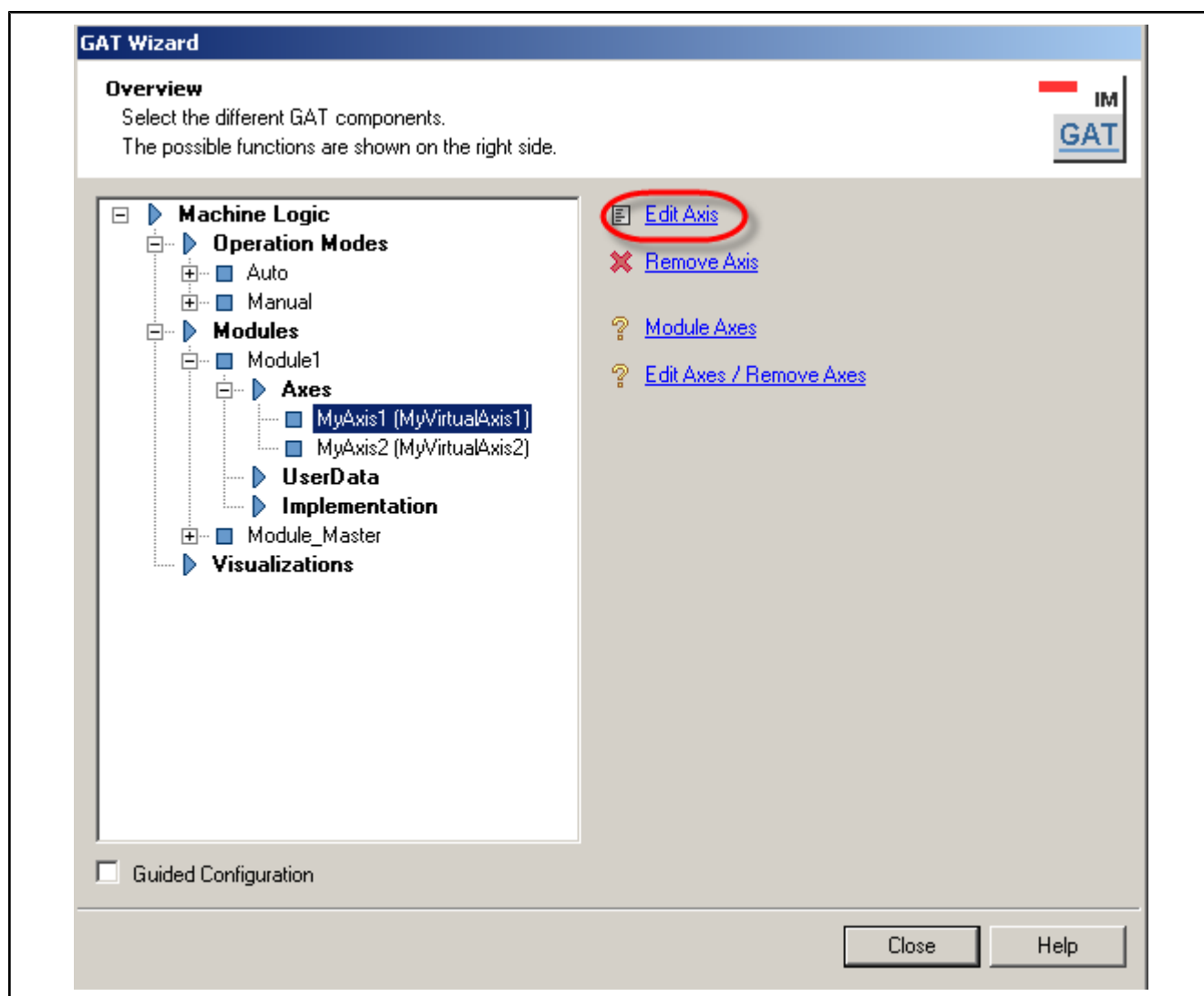


Fig. 8-14: Editing configured axes

GAT with Central State Machine

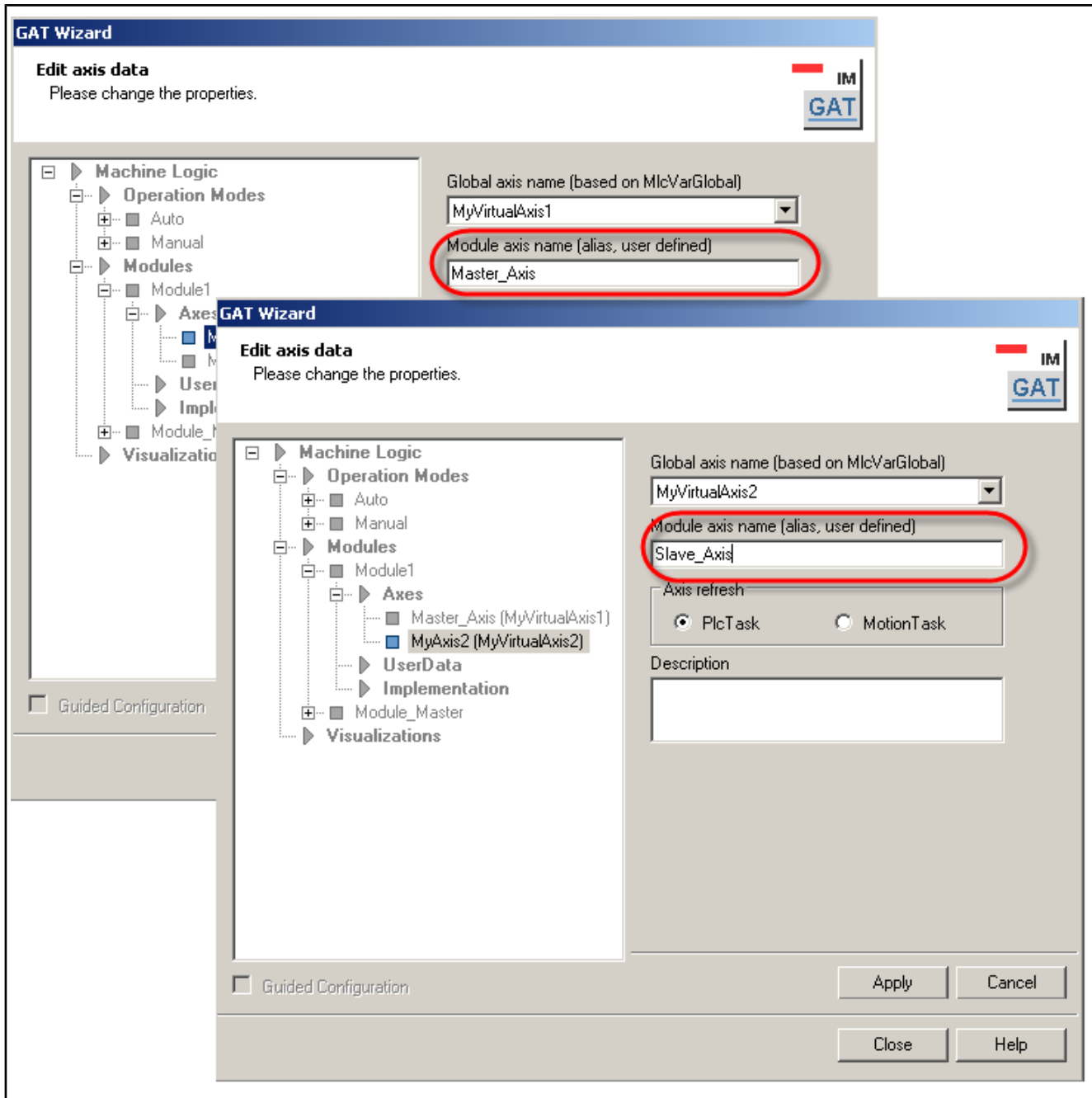


Fig.8-15: Editing configured axes

In the final step, the axis "Slave_Axis" in the "Producing" state in module "Module1" must be synchronized to the axis "MasterAxis" (see the following figure). To do this, the lines of code found on the right side of the figure must be entered.

In addition, the axis "Slave_Axis" must be commanded into the "AH" state by activating the "Auto Stopped" state. See the following note regarding the reason for this.

GAT with Central State Machine



Depending on the system, switching into the Init operation mode (and therefore parameterization mode) is only permitted when all axes acknowledge the "Ab" or "AH" state. Before switching from a User operation mode into the Init operation mode, all axes must be commanded into the "Ab" or "AH" state. For this reason, a corresponding axis command is recommended in the transient section of the initial state "Stopped".

The screenshot shows the GAT Wizard interface. On the left, the 'Machine Logic' tree is expanded to 'Implementation'. The 'Auto Producing' state is selected and highlighted with a green circle. A red arrow points from this state to the 'actAutoProducing' code block on the right. The code block contains the following logic:

```

actAutoProducing[IndraMotionMlc1: Logic: Application: Module1]
9      It is possible to lock the transient and
10     the module, see notes below
11
12     (*$
13     IF Ctrl.StateTransient THEN
14         // ----- transient (entry) state part -----
15         // -----
16         // insert your code here
17         // -----
18         Axes.Slave_Axis.Ctrl.SyncMode.Master:= Axes.Master_Axis.AxisRef;
19         Axes.Slave_Axis.Ctrl.Admin._OpMode.en:= ModeSyncPhase;
20         // NOTE
21         // It is possible to lock the transition to the quiescent state pa
22         // Status.TransientLocked

```

Below this, the 'actAutoStopped' code block is shown, with the line 'Axes.Slave_Axis.Ctrl.Admin._OpMode.en:= ModeAH;' highlighted with a red circle. A red arrow points from the 'Auto Stopped' state in the left tree to this line.

Fig.8-16: Implementing the code in the action of the "Auto_Producing" state

The implementation is complete.

After the configuration (1) and the PLC project (2) have been downloaded, the PLC can be started (3) (see the following figure).



Fig.8-17: Downloading (configuration/PLC project) and starting the PLC

When the pre-defined internal visualization "MAIN_VISU" is used, after the initialization is completed, press the **START** button to command the application into the "Stopped" state.

GAT with Central State Machine

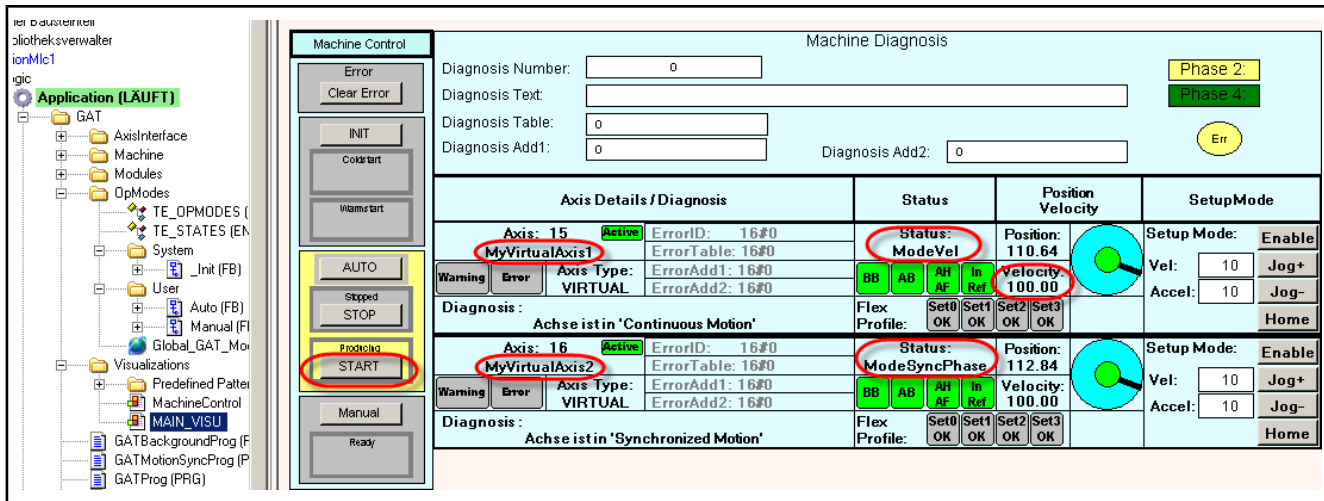


Fig.8-18: Using the integrated IndraLogic commissioning visualization

Press the **START** button to set the global variable "bStart_gb". As explained above, this causes a switch from the "Stopped" state to "Producing".

In the "Producing" state of the "AUTO" operation mode, the global axis "MyVirtualAxis1" ("MasterAxis" axis in the "Module_Master" module or "Master_Axis" axis in the "Module1" module) moves continuously at 100 rev./min. The global axis "MyVirtualAxis2" ("SlaveAxis" axis in the "Module1" module) follows phase synchronously.

This meets the requirements of the topic.

8.4 Functional Principle

In order to generate the Motion application simply and effectively, the program code for the GAT is divided into two sections. One section is located in a closed library (ML_TechTemplate.library) and access to the other section is open.

Using the wizard, the user places the special code for the application in the open section. The code section in the library organizes the calls of operation modes and states, modules and axes. The library works generically, i.e. even if new operation modes, states, modules or axes are added or deleted or their names modified, the consistency of these elements is ensured.

This functional principle yields the following advantages:

- Creation of compact and clearly arranged user code:
-> Through the use of simple interfaces and clear structures based on an optimized program template
- Increased robustness:
-> Through the use of tested and optimized code sections
- Improved maintainability
-> Due to easy library exchange in case of errors

8.5 Initialization

Initialization is completed at machine start up. After a successful initialization, the machine is in operating mode.

Initialization is executed in two steps:

1. Cold start: switches the control into parameterization mode (P2).
2. Warm start: switches the control into operation mode (BB).

GAT with Central State Machine

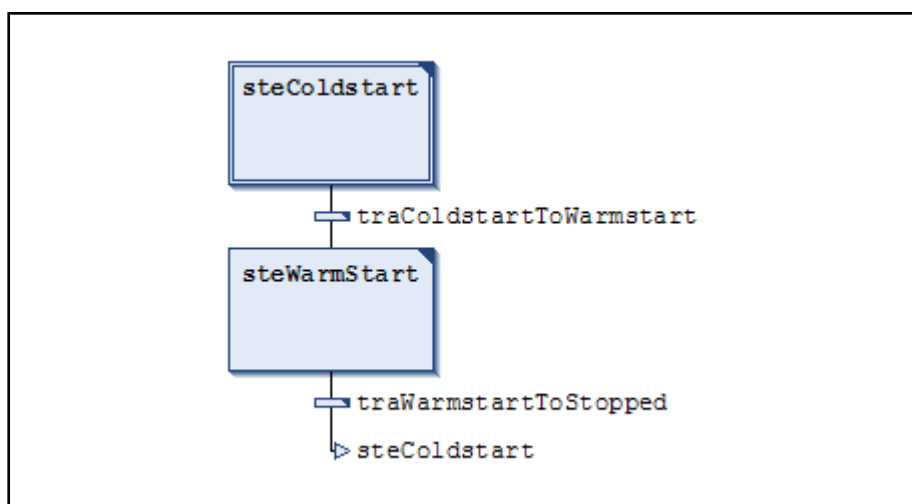


Fig. 8-19: OpMode_Init state machine

SkipColdstartAtStartup/mOneTimeInit

After PLC project download or a PLC reset, the "cold start" state of the Init operation mode is activated by default. Particularly for initial machine commissioning, this behavior can cause disturbances because switching to "cold start" = parameterization mode is time-consuming. For this reason, the **"SkipColdstartAtStartup = TRUE"** setting can be used to prevent the switch to "cold start" in these cases. With this setting, after PLC download or reset, "cold start" is skipped and only "warm start" is processed. However, if the Init machine operation mode is specified using the "OpMode" input, the cold start will be completed even with the setting "SkipColdstartAtStartup = TRUE".

Modifications to the **"SkipColdstartAtStartup"** setting can be made in the assignment of the "mOneTimeInit" method at the "Machine" FB (see the following figure). After the modified PLC project is downloaded, the new settings become effective.



In the state machine (SFC) for the initialization (see "GAT/OpModes/System/_Init") no states/transitions may be removed!

8.6 Operation Modes

8.6.1 General Information

The GAT includes the predefined operation modes "_Init", "Auto" and "Manual". The operation mode "_Init" for initializing the machine was already discussed in the previous chapter.

GAT with Central State Machine

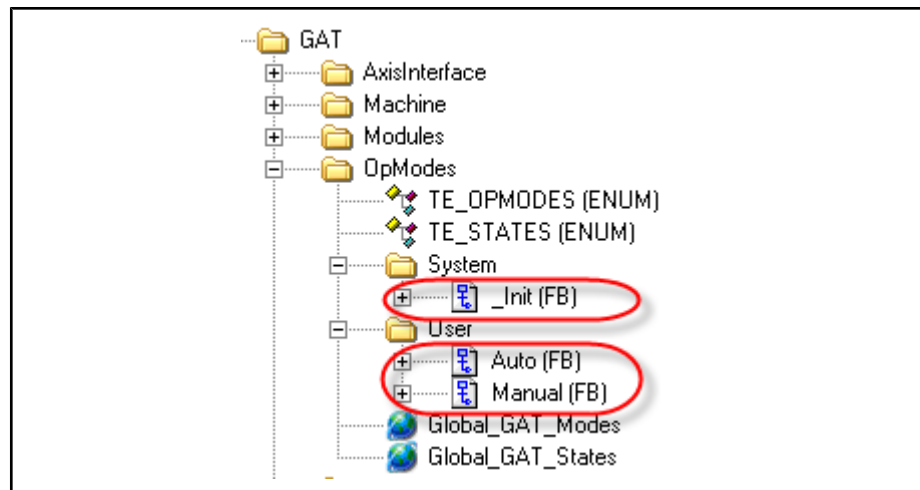


Fig. 8-20: Configured operation modes

Among operation modes, a distinction is generally made between

- **System operation modes**
such as "_Init", which are provided by the system and which users cannot modify
- **User operation modes**
such as "Auto" and "Manual", which users can modify.

All of the [chapter 8.7 "Modules" on page 186](#) of a GAT application are generally located in the same operation mode.

Depending on the actual application, users can defined up to 8 further user operation modes (see [chapter 8.12.4 "Adding an Operation Mode" on page 214](#)).

The operation modes are handled by the machine organization unit in the ML_TechTemplate.library. For this reason, users do not have to call an operation mode within the GAT.

8.6.2 States of an Operation Mode

In order to structure the Motion application in a clear arrangement, several states can be defined in one operation mode (see [chapter 8.12.5 "Selecting the Desired Operation Mode after Clear Error" on page 215](#)).

In this way, the predefined user operation mode "Auto", for example, is identified based on the combination of the "Stopped" and "Producing" states as in the following figure. If the actual application requires it, the user can add further states, rename or clear them.

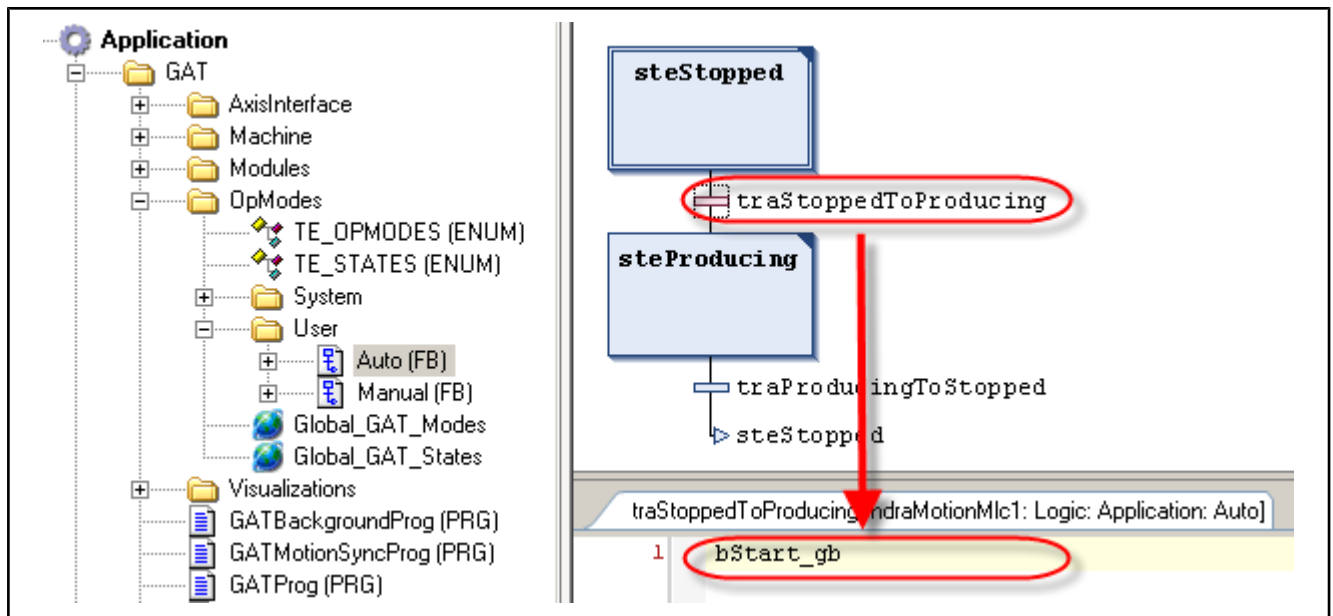


Fig.8-21: Operation mode states and transitions

A state change within an operation mode is carried out if the transition condition that follows the active state is TRUE. The user can provide all of the transitions with an application-specific condition. In this way, for example, in the predefined user operation mode "Auto", a switch from the "Stopped" state to the "Producing" state takes place if the "Stopped" state is active and the global variable "bStart_gb" has the value TRUE (see the previous figure).

8.6.3 Setting the Target Operation Mode after Initialization

The target operation mode is specified after initialization using the variable **fbMachine_gb.Ctrl.Init_Successor**. The default value is **OPMODE_AUTO**. The GAT remains in the this operation mode until **fbMachine_gb.Ctrl.OpMode** is used to command another operation mode. The following figure shows where these elements are located in the GAT. To create further operation modes, see [chapter 8.12.4 "Adding an Operation Mode" on page 214](#).

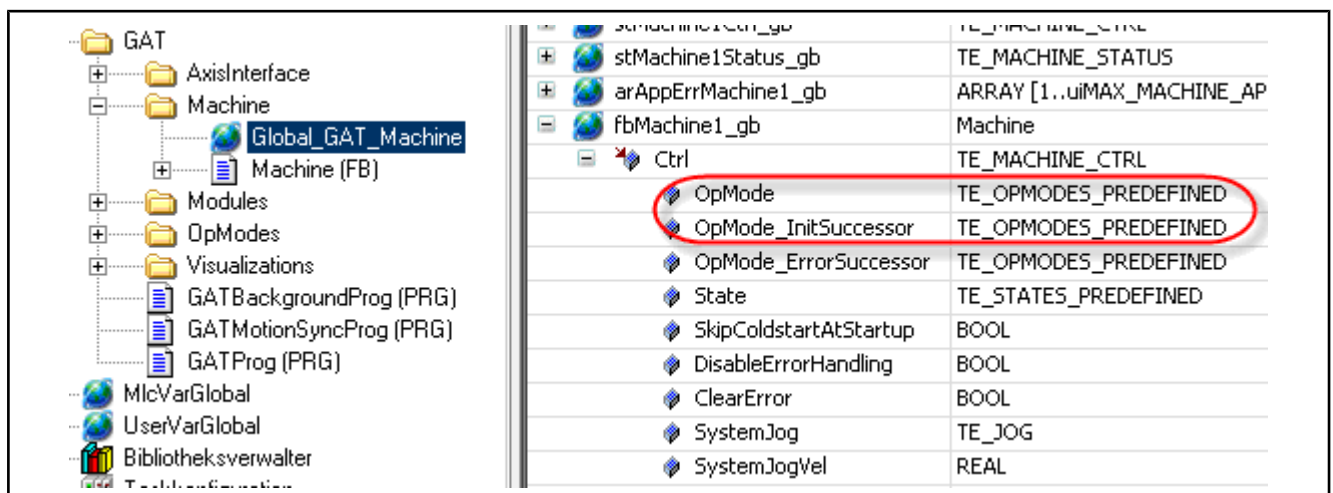


Fig.8-22: Setting the target operation mode after initialization

8.6.4 Changing the Operation Mode

A change of the operation mode is introduced when **fbMachine_gb.Ctrl.OpMode** is assigned a value (an operation mode) that differs from the value of the

GAT with Central State Machine

variable **fbMachine_gb.Status.OpMode**. But this is only possible when the initial state (STATE_<OPMODE>_STOPPED) is active in the current operation mode. In this case, there is a transition from the initial state of the current operation mode to the initial state of the new operation mode.

mOpModeChangeAllowed

As an option, the operation mode switch can be blocked with the "**mOpModeChangeAllowed**" method of the "Machine" FB. This method is automatically called by the GAT when an operation mode switch is introduced. The operation mode switch is blocked as long as the method returns the value FALSE. If the method returns the value TRUE, the operation mode switch is not blocked (default).

Program:

```

METHOD mOpModeChangeAllowed : BOOL
// Method is called in case of operation mode change
// mOpModeChangeAllowed := FALSE blocks the operation mode switching
// mOpModeChangeAllowed := TRUE allows the operation mode switching

// Declaration
VAR_INPUT
    OpMode: TE_OPMODES_PREDEFINED; // Current operation mode
END_VAR

// Implementation
(* IF (OpMode=OPMODE_AUTO) THEN
// Switching from AUTO Mode to another mode is allowed only if the ServiceKey is enabled
mOpModeChangeAllowed := bServiceKeyEnabled;
END_IF
*)

// END Implementation

```



Depending on the system, switching into the Init operation mode (and therefore parameterization mode) is only permitted when all axes acknowledge the Ab or AH state. Before switching from a User operation mode into the Init operation mode, all axes must be commanded into the Ab or AH state. For this reason, a corresponding axis command is recommended in the transient section of the initial state "Stopped".

8.6.5 User Activities when Exiting an Operation Mode

mExitOpMode

In special use cases, certain activities must be carried out when exiting an operation mode (e.g. to reset PLC variables). Functionalities have already been prepared for such use cases and they can be used as follows:

The method "**mExitOpMode**" can be added to every module FB / machine FB. This method is called from GAT when exiting an operation mode for a cycle. The exited operation mode is transferred from GAT using the method's "OpMode" input. For this reason, the following declaration of the method is required:

Program:

```

METHOD mExitOpMode
// Method is called for one cycle on operation mode exit

// Declaration
VAR_INPUT
    OpMode: TE_OPMODES_PREDEFINED; // Old operation mode
END_VAR

// Implementation
(*
CASE OpMode OF
    OPMODE_AUTO:
        ; // Activities in order to reset all variables of operation mode AUTO
    OPMODE_MANUAL:
        ; // Activities in order to reset all variables of operation mode MANUAL
ELSE
    ;
END_CASE
*)

```

```
*)
// END Implementation
```

Within the method, the operation mode can then be queried and the corresponding PLC variables reset.

8.6.6 User Activities when Starting an Operation Mode

mEntryOpMode In special use cases, certain activities must be carried out when starting an operation mode (e.g. to set PLC variables). Functionalities have already been prepared for such use cases and they can be used as follows:

The method "**mEntryOpMode**" can be added to each module FB / machine FB. This method is called from GAT when starting an operation mode for a cycle. The newly started operation mode is transferred from GAT using the method's "OpMode" input. For this reason, the following declaration of the method is required:

Program:

```
METHOD mEntryOpMode
// Method is called for one cycle on operation mode entry

// Declaration
VAR_INPUT
OpMode: TE_OPMODES_PREDEFINED; // New operation mode
END_VAR

// Implementation
(
CASE OpMode OF
OPMODE_AUTO:
; // Activities in order to set all variables of operation mode AUTO
OPMODE_MANUAL:
; // Activities in order to set all variables of operation mode MANUAL
ELSE
;
END_CASE
*)
// END Implementation
```

Within the method, the operation mode can then be queried and the corresponding PLC variables reset.

8.6.7 Bypassing the Central Operation Mode

When switching from one operation mode to another, all modules are generally switched into the new operation mode. Under certain circumstances, however, it may not be useful for all machine modules to follow a centrally specified operation mode because maintenance work might have to be carried out on the module, for example.

mForceOpMode(), mReleaseOpMode()

It is possible to switch one or more modules into another operation mode as specified with **fbMachine_gb.Ctrl.OpMode** of the FB "Machine". To do this, the module methods **mForceOpMode()** and **mReleaseOpMode()** are available.

Both functions are called directly at the module:

Program:

```
fbModule1_gb.mForceOpMode(OPMODE_MANUAL); // Switch to OpMode "Manual"
fbModule1_gb.mReleaseOpMode(); // Switch back to global OpMode
```

The manual switching of individual modules is only possible in a user-defined operation mode. When switching the operation mode, the same rules apply as when switching the machine's central operation mode (see [chapter 8.6.4 "Changing the Operation Mode" on page 183](#)). For this reason, the switch into the specified operation mode is only implemented if the machine allows the operation mode to be switched.

GAT with Central State Machine

If the machine has to handle an error, all modules are affected. As the initialization is completed (OpMode _Init), the manual bypass of the operation mode is canceled for all modules.



Bypassing the central operation mode can lead to unintended changes in the behavior of the entire machine if individual modules are interdependent.

Name	Parameters	Return value	Description
mForceOpMode	OpMode	BOOL	Puts the module into the OpMode specified operation mode
mReleaseOpMode		BOOL	Puts the module into the machine's operation mode

Fig.8-23: Functions for manually bypassing the operation mode of a module

8.7 Modules

8.7.1 General Information

A module in the GAT represents a functional unit of the machine.

Users do not have to call a module within the GAT. The localized machine organization unit in the ML_TechTemplate library does this automatically.

[chapter 8.12.8 "Adding an New Module" on page 217](#) describes how the wizard can be used to add and remove modules.

The provided GAT already contains the module **Module1** (see the following figure). This name can be easily changed by using the GAT wizard (see [chapter 8.12.8 "Adding an New Module" on page 217](#)).

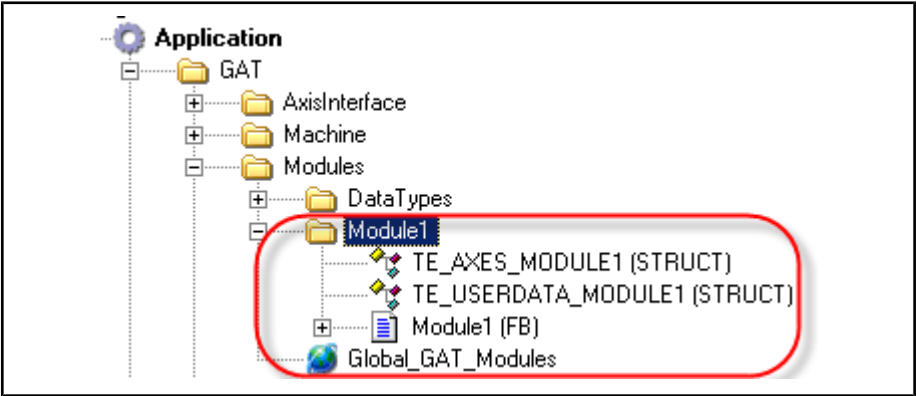


Fig.8-24: Module files in the project directory

8.7.2 Creation Rules

In order that the GAT module be easy to maintain and reuse, it should be designed such that:

- Dependencies between the modules are kept to a minimum
- No global variables are used within a module

To achieve this, each module has the following interface.

8.7.3 Module Interface

General Information

The following figure shows the uniform module interface.

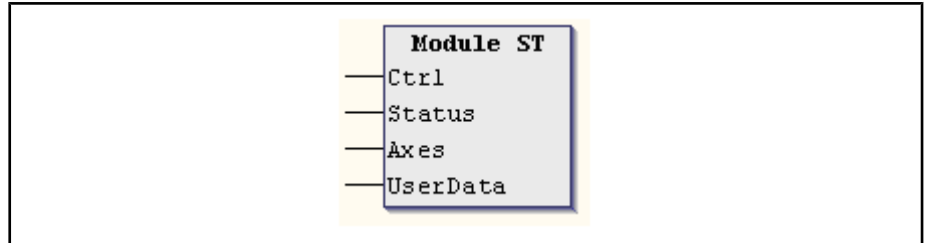


Fig. 8-25: Module interface

For effective, high-performance data handling, all of the elements of the module interface are executed as references, i.e. the data required in the module is not copied into the module, but instead transferred in by a reference to the module.

The components of this interface are described in the following section.

Control and Status Structure

Each module has a variable for control (**Ctrl**) and a variable for reporting status (**Status**) (see the following figure).

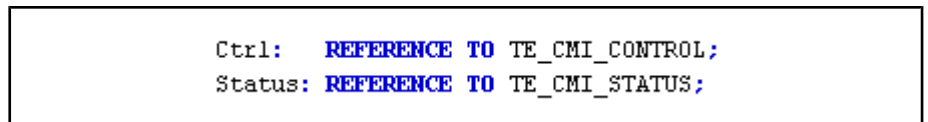


Fig. 8-26: Module control and status structure

The names and types of these variables are identical in each module. [chapter 8.11.4 "Module Data" on page 208](#) provides an overview of which elements belong to these structures, which can be used by the user and which are reserved for use in the internal machine organization unit.

Module axes

Each module's interface has an axis container (**Axes**) that contains all of the axes used in the module. The name of the axis container is the same for each module, but the contents (type) differ because not every module uses the same axes.

Each axis in the axis container (module axis) is type TE_GAT_AXIS, which is based on type TE_BASE_MODULE_AXIS (see ML_TechTemplate library). The following table shows all of the elements that are available in structure type TE_GAT_AXIS.

Element	Type	Description
Name	STRING(20)	Name of the axis as character string (Initialized when the system is initialized; can be modified by the user)
AxisData	REFERENCE TO ML_AXISDATA_SM	Reference to the corresponding element in the global array AxisData (see ML_Base library)
DV	REFERENCE TO ML_DirectVarAxis	In case there is a local axis: Reference to the corresponding element in the global array DV_Axis (see ML_Base library) In case there is a link axis (axis from another control): dummy element

GAT with Central State Machine

Element	Type	Description
GV	REFERENCE TO ML_GlobalVarAxis	Reference to the corresponding element in the global array GV_Axis (see ML_Base library)
AxisRef	AXIS_REF	Axis reference
Disable	BOOL	TRUE: the corresponding AxisInterface data set will no longer be updated FALSE: Default A change to the value of this variable will first take effect in INIT mode ((STATE_INIT_COLDSTART or STATE_INIT_WARMSTART)
MotionSync	BOOL	TRUE: The axis data for this axis are updated in the Motion-Task FALSE: The axis data for this axis are updated in the PlcTask
Ctrl	REFERENCE TO TE_AXISINTERFACE_CONTROL	Reference to the control structure of the AxisInterface of the corresponding axis (for the reference to the corresponding entry of the global array arAxisCtrl_gb , see the global variable list Global_GAT_AxisInterfaces in the AxisInterface folder)
Status	REFERENCE TO TE_AXISINTERFACE_STATUS	Reference to the status structure of the AxisInterface of the corresponding axis (for the reference to the corresponding entry of the global array arAxisStatus_gb , see the global variable list Global_GAT_AxisInterfaces in the AxisInterface folder)

Fig. 8-27: All of the available elements of type **TE_GAT_AXIS**. Within the module, an element can be accessed from **Axes.<AxisName>.<Element>**

The options for deactivating an axis within the GAT programming environment (Element **Disable**) or for assigning it to the MotionTask (Element **MotionSync**), can be selected within the GAT wizard when a module axis is defined or re-named (see [chapter 8.12.3 "Removing/renaming an Axis " on page 212](#)).



An axis must be assigned to the MotionTask if an activity is running in the MotionTask that uses this axis!

All relevant data for an axis can be addressed using the **Axes** axis container.

The container is designed such that the axis data is accessed using the name of the axes. This generates program code that is easily readable.

The following table shows how to access axes in the axis container from within and outside a module.

Access to a module axis (e.g. axis "vAxis" in the axis container of the "Module1" module)	Assigning the update of the axis data to the MotionTask	Disable axis
Access from outside the module	stAxesModule1_gb.vAxis.MotionSync := TRUE;	stAxesModule1_gb.vAxis.Disable := TRUE;
Access from within the module	Axes.vAxis.MotionSync := TRUE;	Axes.vAxis.Disable := TRUE;

In the method "mOneTimeInit" (this method is processed only once during the initialization of the GAT) of the superordinate machine, the wizard is used to assign a physical axis (real, virtual, ...) to the corresponding axis in the module (see the following figure and see [chapter 8.12.2 "Adding an Axis" on page 211](#)).

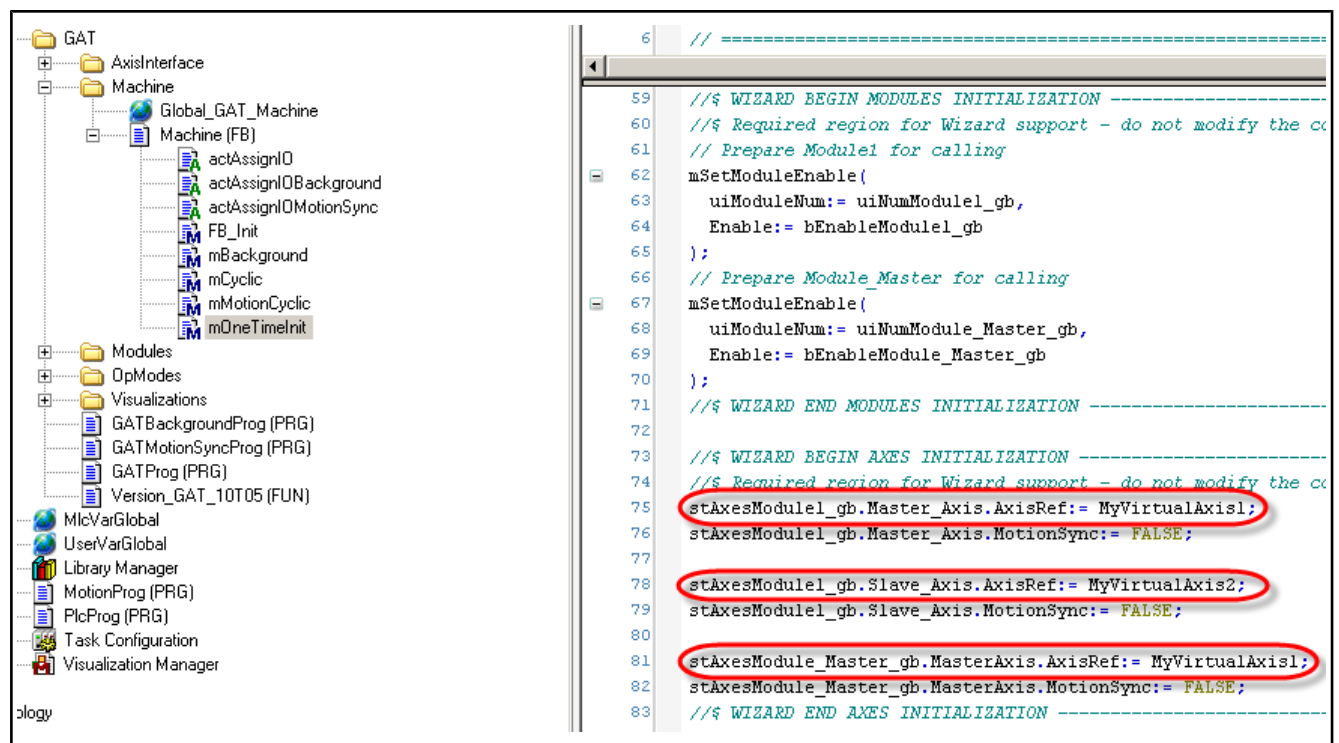


Fig. 8-28: Axis assignment in the "mOneTimelnit" method

An example from the figure above: the global axis generated in IndraWorks, "MyVirtualAxis1", is assigned the axis "Master_Axis" of the module "Module1".

Passive axes

In principle, it is recommended that all axes that are **not actively** commanded in a module be transferred using the I/O data container **UserData** specific to the module. Typical examples for axes that are not actively commanded are master axes or encoder, link and C2C consumed axes.

Using the UserData container is associated with the following advantages:

- Link axes and C2C consumed axes can be transferred to a module.
- Performance advantage because the AxisInterface is not called for the passive axes.
- Easier for users because it can be seen more quickly which axes are actively commanded in which module.

The following procedure is recommended to configure a passive axis in the UserData container:

- Declare passive axes in the module's UserData container.

GAT with Central State Machine

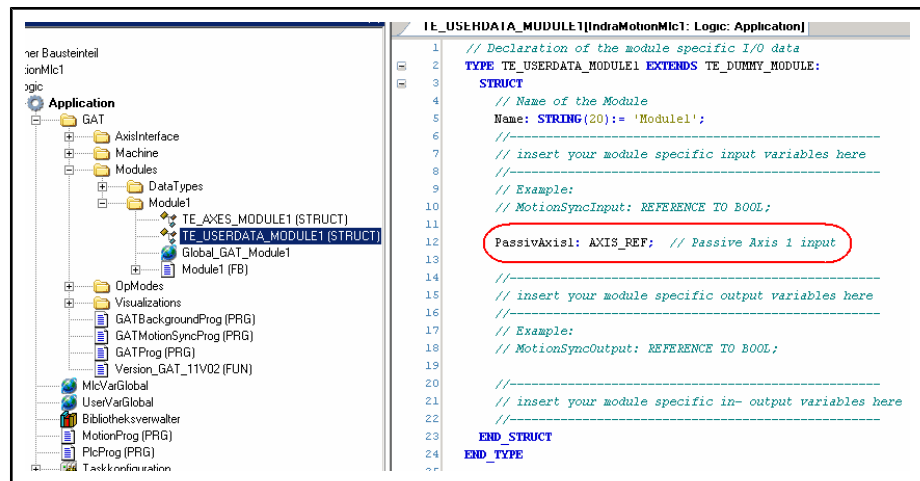


Fig.8-29: Declaration of a passive axis "PassiveAxis1" in the UserData container of a module

- Assign the global configured axis to the module axis.

In the following example, the global axis "MyMasterAxis" is assigned to module axis "PassiveAxis1". The global axis "MyMasterAxis" must be configured with IndraWorks in another step. Exceptions are C2C consumed axes that are not configured with IndraWorks.

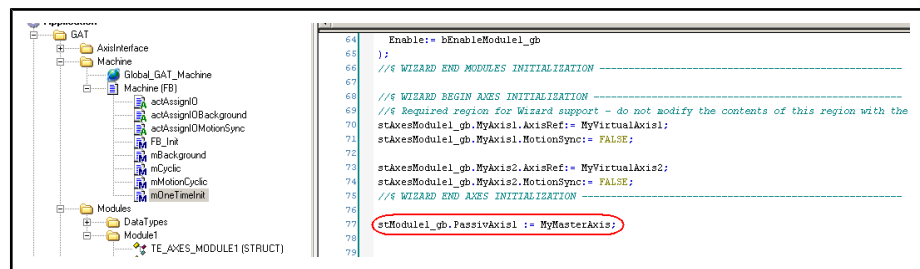


Fig.8-30: Example assignment of the global axis "MyMasterAxis"

- The declaration of global C2C consumed axes is implemented by the user with an entry in a global variable list. The following example of code shows the declaration of the global C2C consumed axis "MyMasterAxis", which is assigned to Axis 1 of Control 2.

Program:

```
VAR_GLOBAL MyMasterAxis: AXIS_REF := (AxisNo := 1, CntrlNo := 2); END_VAR
```

- If the passive axis is a C2C consumed axis, the axis data (e.g. position, velocity) can be read in the module using the variable Variable GV_Link-Axis[]. The passive axis can also be used in the module as a master axis.

Module-specific I/O data

If a module is to be reused, global data may not be accessed in that module. All I/Os required in the modules are made available using the I/O container **UserData**. The name of the I/O container is the same for each module, but the contents (type) differ because not every module uses the same I/O data.

The variable UserData is a data structure for which the user must define the elements.

The following figure shows where the I/O data for the "Module1" module contained in the GAT are to be defined:

GAT with Central State Machine

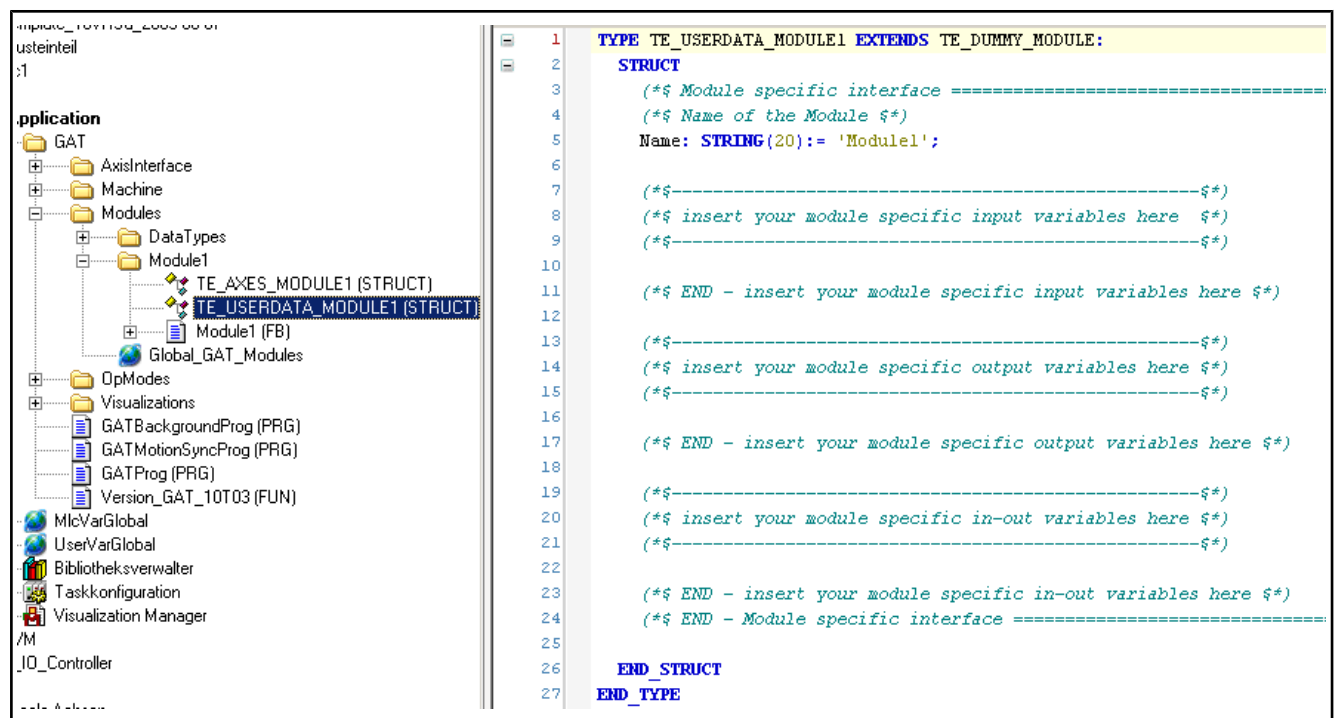


Fig. 8-31: Module-specific I/O data

Module Instances

Further instances can be created from an existing module. These instances differ only in the assigned IndraWorks axes and assigned I/Os. The module instantiation is supported by the GAT wizard and is described in section [chapter 8.12.9 "Adding Another Instance of a Module" on page 218](#).

8.7.4 States in the Module

When adding or deleting a state within an operation mode, the wizard ensures that an action corresponding to the state is created in each module. This action is only processed if the corresponding state is active in the related operation mode. In this way, the action "actAutoProducing" is processed in the module "Module1" if the "Producing" state in the "AUTO" operation mode is active.

Each action within a module that refers to a state can be divided into a **transient** and a **stationary** section.

The transient section of the action that refers to a state is specified by the **Ctrl.StateTransient** condition. Here the user should coordinate the activities that are required only for a certain period in order to detect a state (e.g. switch on a temperature controller in the AUTO_PRODUCING state). If this is not possible in a PLC cycle, the user can set **Status.TransientLocked:= TRUE** so that the process remains in this section (e.g. until the nominal temperature of the temperature controller in the AUTO_PRODUCING state is reached).

If **Ctrl.StateTransient=FALSE**, the stationary section of the action that refers to a state is processed. In this case, the user should code all activities that are to be executed when this state is already detected (e.g. synchronous coupling/decoupling in case of missing products) by setting **Status.StateLocked:=TRUE**, a switch into another state in this operation mode can be prevented, even if the corresponding transition condition has been met.

GAT with Central State Machine

8.7.5 Module Coordination

Because a module does not contain any information on other modules, the coordination of the modules must be carried out at a superordinate position.

The machine organization unit of the ML_TechTemplate.library library calls the machine's **"mCyclic"** method in each PLC cycle (see the following figure). In this cyclically called method, I/O information is to be exchanged between the modules and from the modules to the I/O points of the machine.

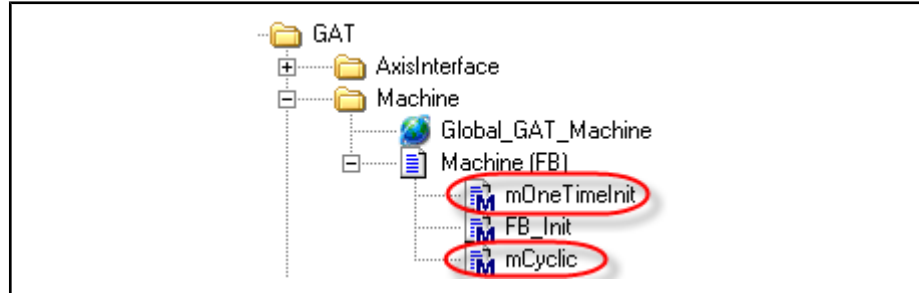


Fig. 8-32: Method "mCyclic" of the machine organization unit

Because the module configuration or the assignment of global axis to a corresponding module axis does not change during runtime, these activities are carried out in the machine's **"mOneTimeInit"** method. This method is called once for initializing the GAT using the machine organization unit (see previous figure).

The code contained in the "mOneTimeInit" method is handled by the wizard. When adding or removing modules or module axes, the corresponding entries are added or removed.

8.8 Task Handling

8.8.1 Task Configuration, GAT

Several tasks run simultaneously within the PLC. Each of these tasks executes its own PLC code. The processing of the tasks by the control processor is specified based on task priorities and cycle time.

The MotionTask, PlcTask and BackgroundTask are all contained in the standard GAT.

	Priority	Cycle time	Watchdog time	Watchdog sensitivity	Program call
GATMotionSyncTask	2	=SERCOS cycle time	T#2ms	1	GATMotionSyncProg
GATProgTask	5	T#10ms	T#20ms	2	GATProg
GATBackgroundProg-Task	6	T#40ms	-	-	GATBackgroundProg

8.8.2 GATMotionSyncTask

In order to access axes consistently from the MotionTask, the AxisInterface must also be called in the MotionTask. Processing all of the axes in the MotionTask would cause an unnecessary load on the MotionTask. For this reason, an axis can be added by using the GAT wizard to assign the processing of the AxisInterface to the PlcTask or MotionTask.

See also [chapter "Typical Tasks for the MotionTask" on page 35](#).

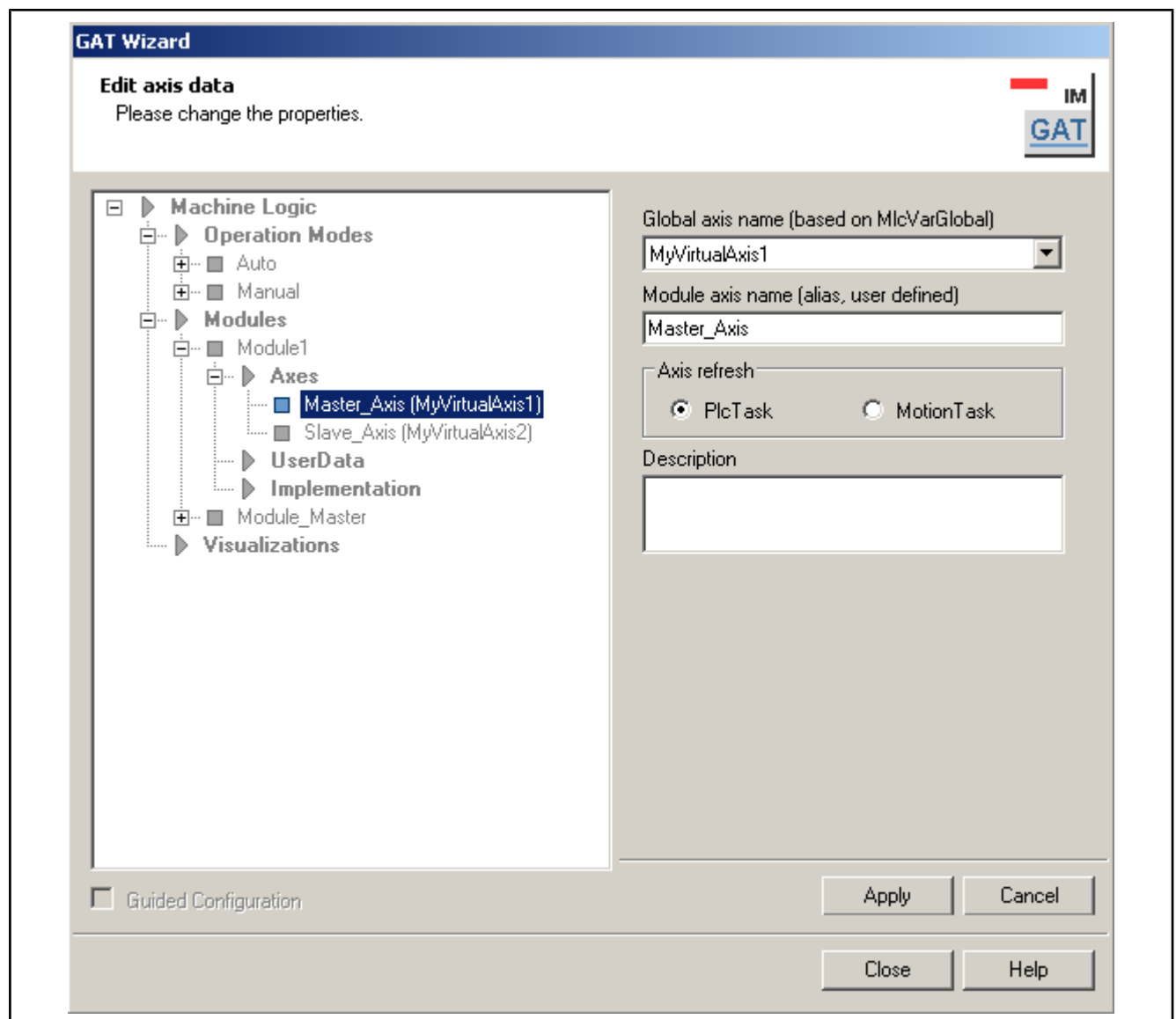


Fig. 8-33: Assignment of the AxisInterface processing



In principle, the axis data should be updated in the PlcTask.

An axis must be assigned to the MotionTask if an activity is running in the MotionTask that uses this axis! (e.g. A process controller is processed in the MotionTask. Then all of the axes assigned to the process controller must also be updated in the MotionTask!)

For processing in the MotionTask, each module / machine has the **"mMotion-Cyclic"** method. All of the module / machine variables (e.g. central axis entry point Axes...) can be accessed from this method.

8.8.3 GATBackgroundProgTask

If the runtime for the PlcTask is too long, the load on the PlcTask can be reduced by moving tasks that are not time critical to a background task. Please note, however, that a greater load is placed on the entire system due to the additional task Switch PlcTask <-> BackgroundTask (see also [chapter "Typical Tasks for the BackgroundTask" on page 36](#)).

GAT with Central State Machine

For processing in the BackgroundTask, each module / machine has the **"mBackground"** method. All of the module methods can be accessed from this method.

8.9 Error Handling

8.9.1 Error Detection and Diagnostics

The program template GAT contains predefined functionalities for detecting, handling and clearing errors at a **central location**. Errors from various sources (e.g. control errors, application errors, FB errors, ...) are automatically gathered together with the cross-machine FB "Machine" and are made available using the status structure of the FB "Machine".

If several errors occur simultaneously, the most severe is displayed. If several errors with the same degree of severity have occurred simultaneously, the most recently detected error is output.

The machine status structure contains the following described structure elements for diagnostic purposes:

Element	Type	Description
Error	BOOL	TRUE indicates an error
DiagNo	DWORD	Number according to the IndraMotion MLC documentation with the following extensions: 16#F0AE xxxx: Application error (e.g. safety door is open) (xxxx = can be defined by the user) 16#F0FB 0000: FB error (e.g. error at a TechFB) -> for details, see ErrorID / ErrorIdent 16#F0AA 0000: GAT internal error -> for details, see ErrorID / ErrorIdent
ErrorID	ERROR_CODE	Detailed diagnostics in case of FB + GAT errors (DiagNo = 16#F0FB/AA 0000)
ErrorIdent	ERROR_STRUCT	Detailed diagnostics in case of FB + GAT errors (DiagNo = 16#F0FB/AA 0000)
DiagMessage	STRING(60)	It contains the diagnostic text for drive and control errors

The **DiagNo** element is only for use on an HMI. The error texts are assigned the numerical information in any desired language there.

DiagMessage, **ErrorID** and **ErrorIdent** provide additional information that is primarily displayed during commissioning by the programmer, e.g. using an IndraLogic visualization.

Note regarding application-specific errors

mSetAppError

By calling the method mSetAppError(Number), application errors (e.g. safety door is open) can be signaled. The "number" transferred at the method input corresponds with the application error number.

If an application error was signaled, the central diagnostic system in the module / machine performs cyclic monitoring of the application errors that have been signaled and generates the application error DiagNo = Error 16#F0AE xxxx (xxxx = application error number).

Numbers in the range 1 to uiMAX_MODULE_APP_ERROR_INDEX are valid for application errors in modules. The constant "uiMAX_MODULE_APP_ERROR_INDEX" has the value 64 by default. If necessary, this

GAT with Central State Machine

constant can be adjusted in the global variable list "Global_GAT_Machine" below the "Machine" folder.

Numbers in the range 1 to uiMAX_MACHINE_APP_ERROR_INDEX are valid for application errors in the machine. The constant "uiMAX_MACHINE_APP_ERROR_INDEX" has the value 64 by default. If necessary, this constant can be adjusted in the global variable list "Global_GAT_Machine" below the "Machine" folder.

Code example for signaling an application error:

Program:

```
// Recommended code to issue user specific errors (Example)
IF bDoor10open THEN
    mSetAppError(1);
END_IF
```

Note regarding severe errors

The control diagnostic system defines 10 degrees of "severity" (see IndraMotion MLC diagnostic system). The degree of severity affects the error reaction (see [chapter 5.6.4 "Error Reaction" on page 41](#)).

- DiagNo = 16#F0xx xxxx is the lowest degree of severity (e.g. application errors, FB errors)
- DiagNo = 16#F9xx xxxx is the highest degree of severity (fatal system errors)

The error with the highest degree of severity is displayed if several errors occur simultaneously.

Note regarding function block errors

mSetFBError

If errors occur in called function blocks (e.g. technology FBs), calling the method "mSetFBError(ErrorID, ErrorIdent)" is recommended along with transferring the "ErrorID" and "ErrorIdent" outputs of the FB that caused the error to the method. This ensures that the error is forwarded to the diagnostic system of the module / machine and the "ErrorID" and "ErrorIdent" are made available to the diagnostic outputs of the module / machine.

Code example for signaling function block error:

Program:

```
// Recommended code a used FB issues an error (Example)
IF fbCrossCutter.Error THEN
    mSetFBError(ErrorID:=fbCrossCutter.ErrorID,
                ErrorIdent:=fbCrossCutter.ErrorIdent);
END_IF
```

8.9.2 Error Reaction

Depending on the system, the control system automatically executes an error reaction only for axis and control errors starting at a severity degree of 2 (see [chapter 5.6.4 "Error Reaction" on page 41](#)). For all other errors and lower degrees of severity (e.g. application errors, FB errors...) an automatic error reaction is not executed.

DisableErrorHandling

If GAT error handling is activated (Machine control structure element "DisableErrorHandling" = FALSE), in case of an error, the selected operation mode is always interrupted and the error reaction is active.

Additional application-specific error reactions can be programmed in this case in the action "actError" of the module. When error handling is switched off ("DisableErrorHandling" = TRUE), the selected operation mode remains active even if the error output is set. The application-specific error reaction "actError" is not processed in this case.

GAT with Central State Machine



If the error reaction is already active and if the structure element "DisableErrorHandling = TRUE" is currently set, the system exits the error reaction and branches into the following operation mode (see OpMode_ErrorSuccessor setting in the machine control structure).

Error filtering

Not every error on the machine must necessarily lead to shutting down the entire system. For this reasons, as an option errors can be processed centrally by the machine or in a decentralized manner by the individual modules. To do this, the mIsCriticalError method is available at the FB "Machine" and at every module FB.

If the machine detects an error, the following steps are processed:

1. The diagnostic number (DiagNo) and, if applicable, the axis that caused the error are determined.
2. mIsCriticalError(DiagNo, AxisRef) of the FB "Machine" is called up.
If mIsCriticalError() returns FALSE, the error is ignored.
If mIsCriticalError() returns TRUE, the next step is processed.
3. mIsCriticalError(DiagNo, AxisRef) is called up at every active module.
If all modules return FALSE for this call, the error is ignored.
If at least one module returns TRUE, normal error handling continues.

By default, mIsCriticalError() returns TRUE, i.e. every error is classified as critical. The behavior can be influenced by implementing the mIsCriticalError method under the FB "Machine" or at the module FB.

Example implementation of the mIsCriticalError() method at the FB "Machine":

Program:

```
METHOD mIsCriticalError : BOOL
VAR_INPUT
    DiagNo: DWORD;    // diagnostic number
    Axis: AXIS_REF;    // axis
END_VAR

CASE DiagNo OF
    // My uncritical errors
    16#F0AE0001, 16#F0AE0002: mIsCriticalError := FALSE;
ELSE
    // All other errors
    mIsCriticalError := TRUE;
END_CASE
```

Example implementation of the mIsCriticalError() method at a module FB:

Program:

```
METHOD mIsCriticalError : BOOL
VAR_INPUT
    DiagNo: DWORD;    // diagnostic number
    Axis: AXIS_REF;    // axis
END_VAR

mIsCriticalError := TRUE;

IF Axis.AxisNo = 0 THEN
    // error with no axis supplied
    mIsCriticalError := FALSE;
ELSIF mIsMyAxis(Axis) THEN
    // error at my own axis
    IF DiagNo=16#FFAA0002 THEN
        mIsCriticalError := FALSE;
```

	<pre> END_IF END_IF </pre>
Decelerating all axes	For standard cases, the deceleration of all axes is programmed with "ModeAH" as an example in the following section (ModeAH = deceleration to full stop). If the GAT error handling is switched off (machine control structure element "DisableErrorHandling" = TRUE), the selected operation mode is not interrupted (the application-specific error reaction "actError" is not started). The diagnostic outputs, however, are set in this case as well and can be deleted using "ClearError". Standard case for the error reaction in the module action actError (all axes decelerate): <i>Program:</i> <pre> // Error reaction example 1 // Decelerate all module axes to standstill => switch all axes to ModeAH FOR uiLoopIndex:=1 TO Axes.uiAxesNum DO IF mGetAxis(AxisNumber:=uiLoopIndex, PointerToAxis=>pCurrentAxis) THEN pCurrentAxis^.Ctrl.Admin._OpMode.en := ModeAH; END_IF END_FOR </pre>
Synchronous deceleration	Often it is required that axes be decelerated synchronously (to prevent film or paper from tearing, for example). In these applications, it is recommended that if at all possible, all axes remain connected in synchronous movement, and that only the master is decelerated. This use case is shown in the following example: <i>Program:</i> <pre> // Error reaction example 2 // Keep all axes in synchronization with the master axis // and decelerate only the master axis FOR uiLoopIndex:=1 TO Axes.uiAxesNum DO IF mGetAxis(AxisNumber:=uiLoopIndex, PointerToAxis=>pCurrentAxis) THEN IF pCurrentAxis^.AxisData.SynchronizedMotion THEN // Axis is in synchronized motion => no reaction as the master // axis decelerates to standstill and the slave axis follows in synchr. ELSE // Axis is not in synchronized motion => command to standstill pCurrentAxis^.Ctrl.Admin._OpMode.en := ModeAH; END_IF END_IF END_FOR </pre> For synchronous deceleration, the module bus reaction for the real axes must also be switched off (see the "Error reaction" section in the drive documentation). Furthermore, in the "best possible deceleration" of the real axes, "30s NC reaction" must be set. In addition, bit 0 of A-0-0651 = TRUE must be set so that the control does not execute an automatic error reaction (see also the "Error reaction" section in the "Rexroth IndraMotion MLC xxxVRS function description"). xx - Version

8.9.3 Clearing Errors

The central clearing of an error is automatically executed with a positive edge at the structure element "ClearError" of the machine control structure. The "Machine" FB forwards the command to all subordinate components (e.g. IMC interface¹⁾, module, AxisInterface ...) in order to bring the entire system into a trouble-free state. The structure element "enOpModeCtrl_ErrorSuccessor" (see machine control structure) determines which operation mode becomes active after clearing the error.

¹⁾ *IndraMotionControl Interface = IndraLogic library with an interface to the control, e.g. clear errors*

GAT with Central State Machine

8.10 Visualization

8.10.1 Internal Visualization for Commissioning Support

General Information

The GAT contains predefined IndraLogic visualization objects that offer assistance during commissioning as well as extended diagnostic possibilities.

AXIS_GAT

The visualization element "AXIS_GAT" can be instantiated for individual axes. This is implemented as an example in the "MAIN_VISU" visualization for the "MyVirtualAxis1" and "MyVirtualAxis2" axes.

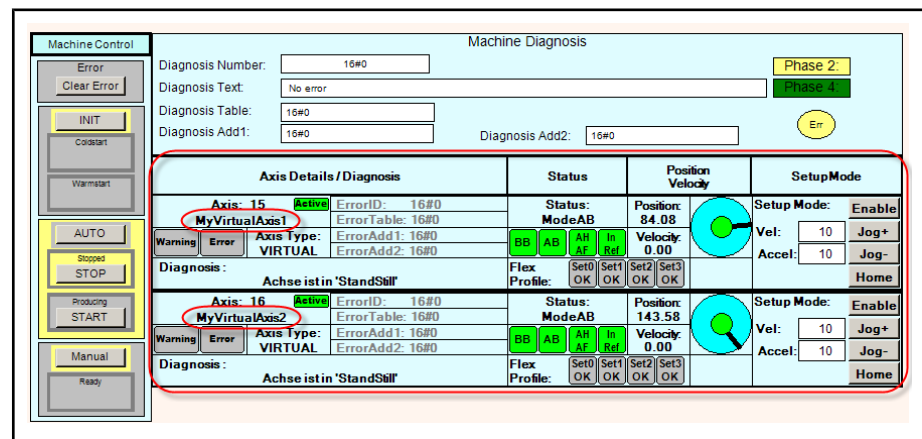


Fig. 8-34: Internal visualization for assistance during start up

The "Frame" object can be used to create an instance of the visualization object AXIS_GAT in the visualization.

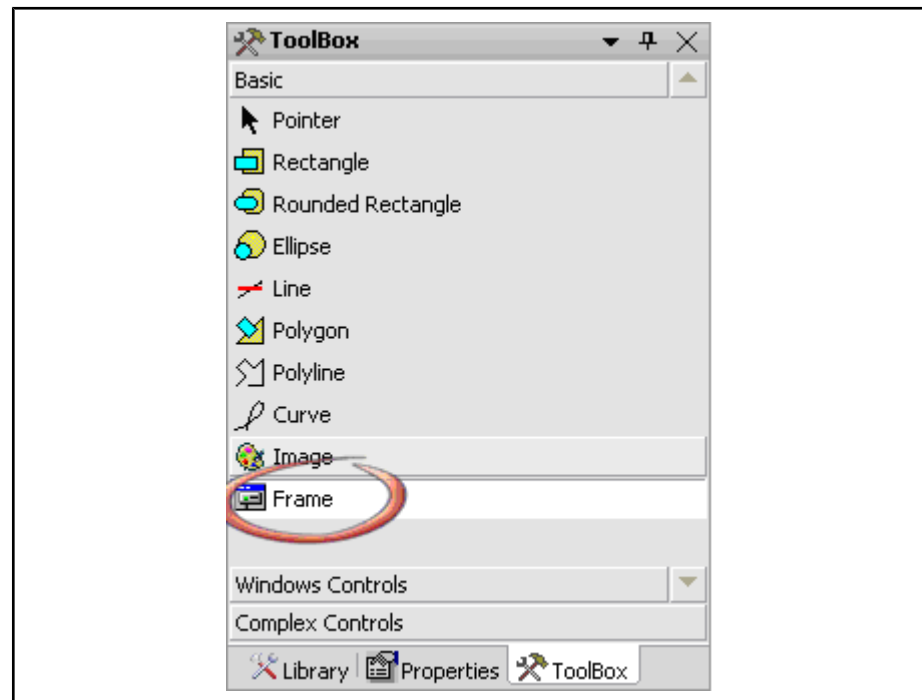


Fig. 8-35: Creating an instance of the visualization object "AXIS_GAT"

Now the corresponding axis reference must be entered under Properties - References.

GAT with Central State Machine

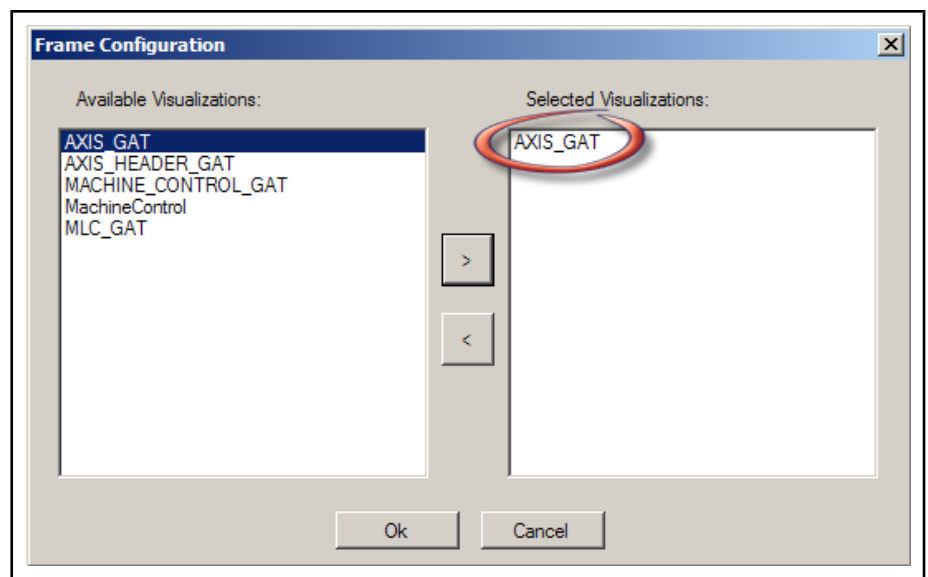


Fig. 8-36: Creating an instance of the visualization object "AXIS_GAT"

8.10.2 External Visualization with Separate HMI (WinStudio Visualization)

General Information

A predefined WinStudio visualization is available for the GAT. A wizard can be used to add this visualization into the project.



To edit the WinStudio visualization, at least the following development environment license (or higher) is required:

WinStudio development license, max. 1,500 variables (Mat.No. R911323596)

For execution on a VEP, at least the following runtime license is required:

WinStudio runtime license for devices using Windows CE as an operating system, max. 1,500 variables (Mat. No. R911323620)

Adding the WinStudio visualization



In order to add the WinStudio visualization to the project, a corresponding visualization device must be present in the project.

Use "drag and drop" to add a VEP from the device library into the project tree and configure it as follows:

- Device type: VEPxx (CE)
- Application type: WinStudio application

GAT with Central State Machine

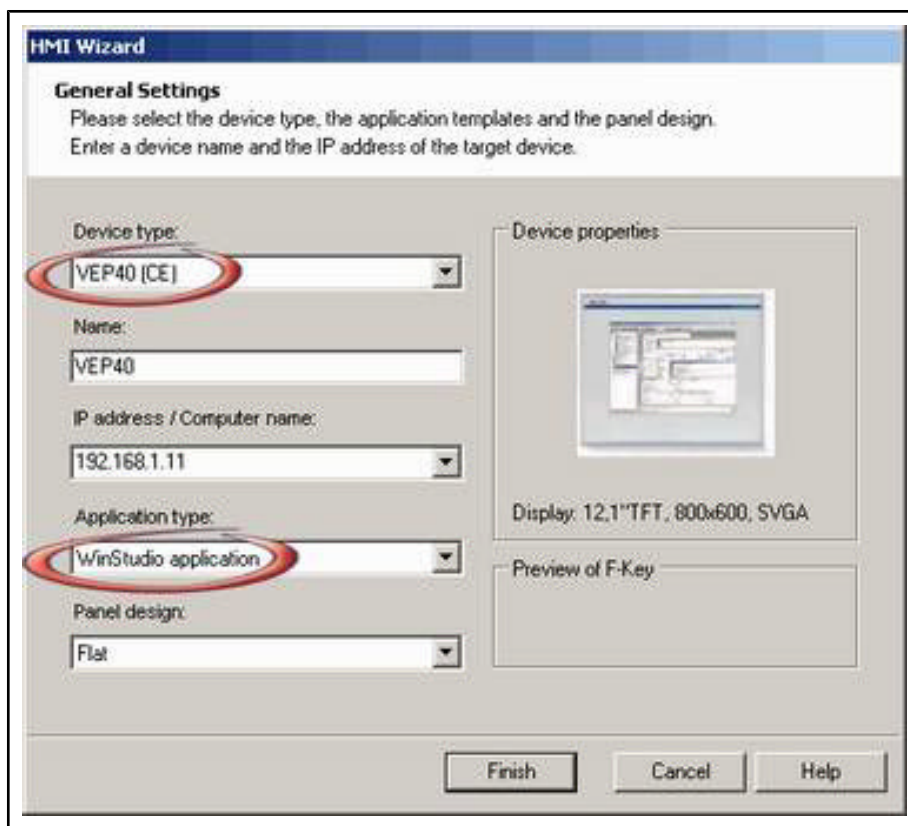


Fig.8-37: Adding a visualization device

If a corresponding visualization device is present, the WinStudio visualization is added using the following link and the wizard.

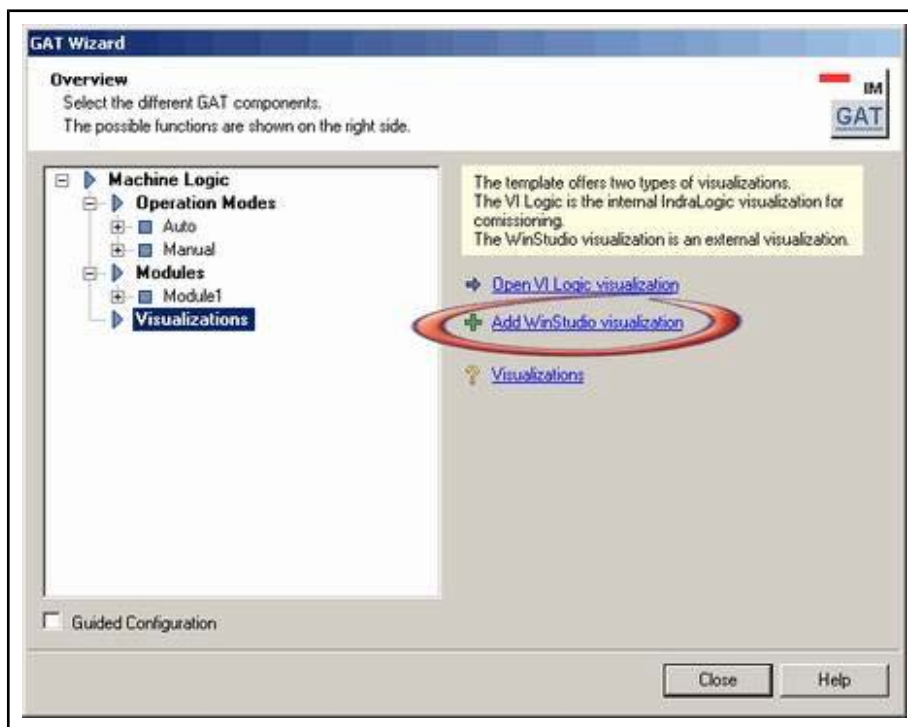


Fig.8-38: Adding a WinStudio visualization

All of the project's visualization devices that correspond to the given configuration are included in the following wizard dialog.

GAT with Central State Machine

After the corresponding visualization device is selected, click the **Apply** button to add the WinStudio visualization.

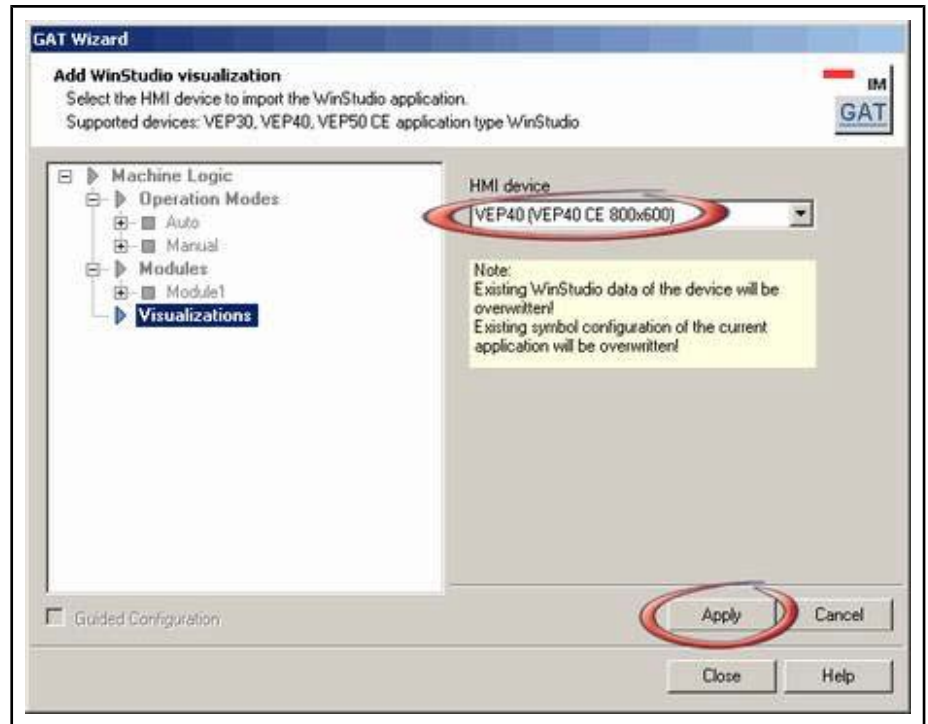


Fig. 8-39: Selecting a visualization device and adding a WinStudio visualization



Existing visualization data from the visualization device along with the symbol configuration of the control are overwritten when the WinStudio visualization is added!

After the visualization is added, its data can be edited.

The control address is configured in the BR-WS driver properties dialog:

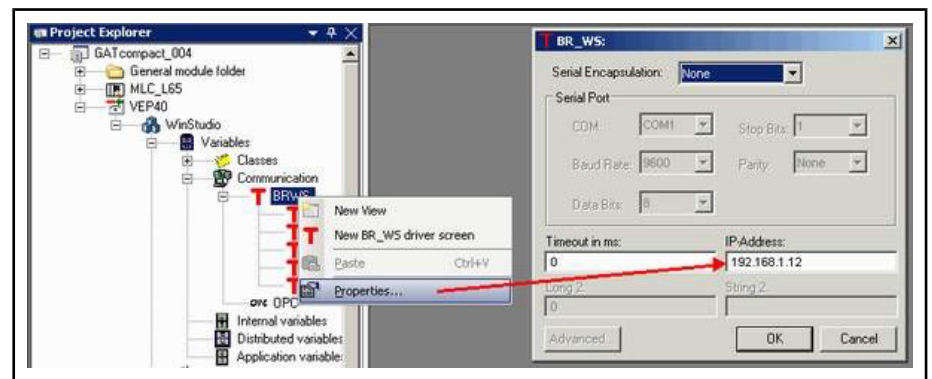


Fig. 8-40: Setting the control's IP address

Now use  "Start application" in the WinStudio node context menu to start the emulation of the WinStudio visualization.

Dialog Structure

The operation screens are divided into 3 subsections:

- Header with status information
- Main screen with menu-related displays and function keys

GAT with Central State Machine

- Footer with keys for switching from one user map to another

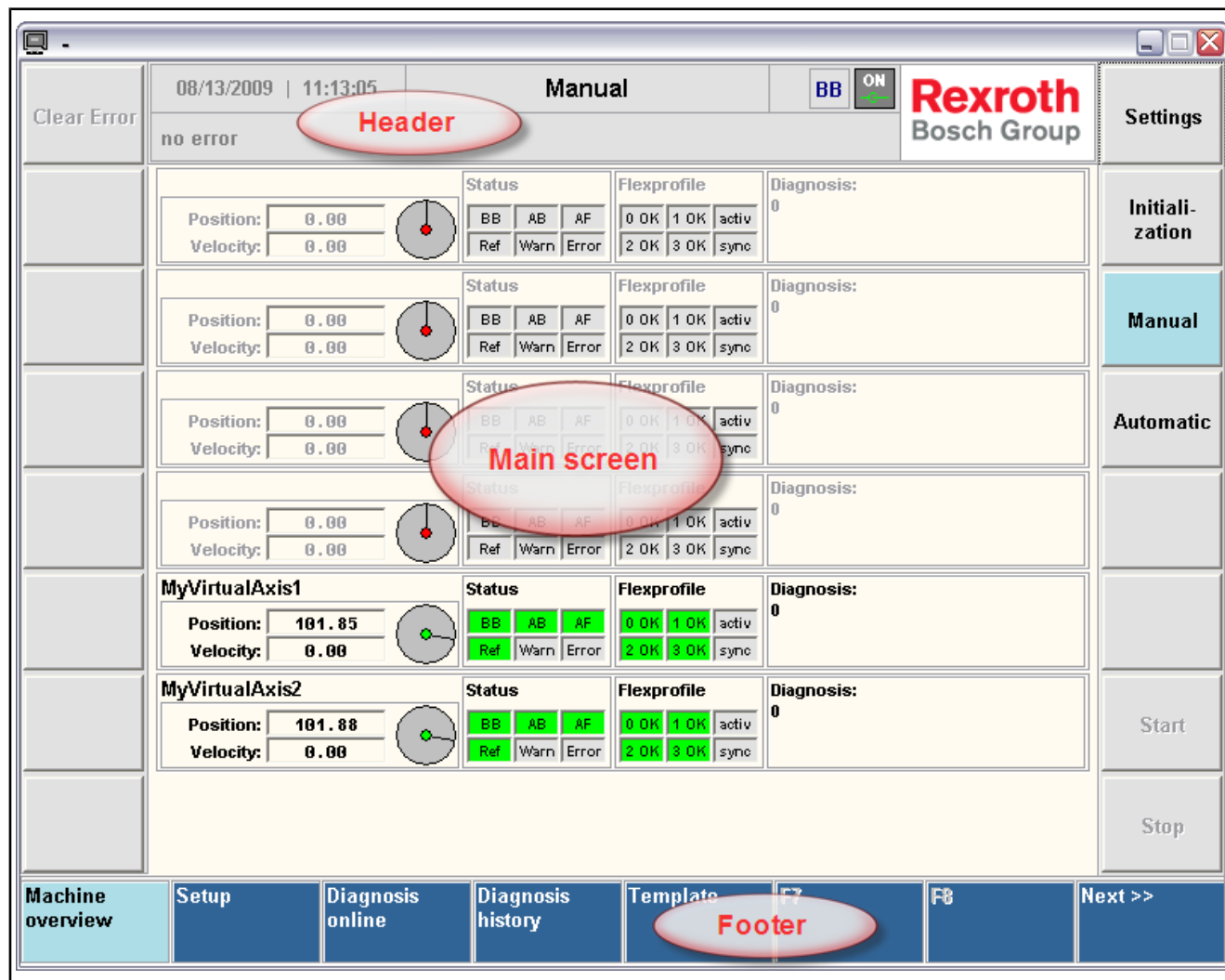


Fig.8-41: WinStudio visualization

The **header** contains:

- Operation mode display
- Global function keys for "clearing errors" and opening the "Settings" dialog (for switching languages, setting the decimal place for numerical displays)
- Display of current machine diagnostics
- Display of current SERCOS phase (including active phase switching)
- Display of communication status

The **main screen** contains displays and function keys related to user maps (e.g. axis information, operation mode switching, ...)

The **footer** contains the keys for switching from one user map to another.

Extending the Visualization

Adding an operation screen

To extend the visualization, new user maps can be created by copying the existing "template" screens and inserting them with new names.

GAT with Central State Machine

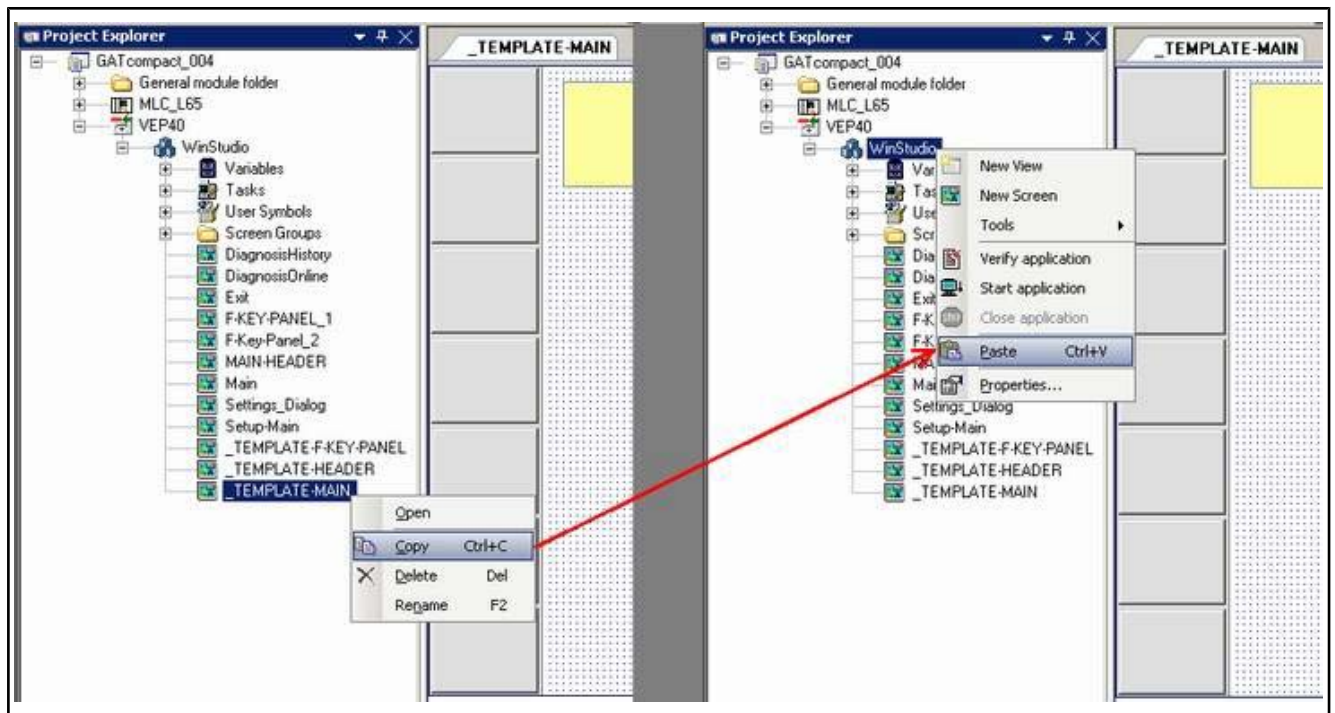


Fig. 8-42: Duplicating an operation screen

The elements for displaying the axis data are grouped individual objects that can be duplicated by "copying" and "pasting". To reconfigure the inserted object for a different axis the Replace function must now be used for the respective object to adapt the axis array index (see the following figures).



Fig. 8-43: Reconfiguring the inserted object

GAT with Central State Machine

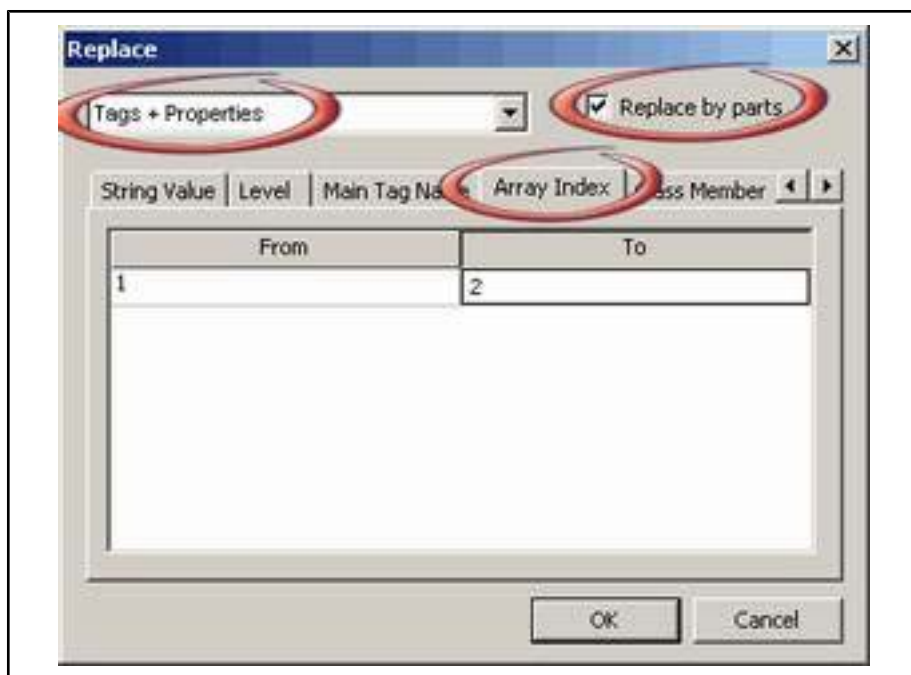


Fig.8-44: Reconfiguring the inserted object

Extending the footer

The keys used to switch from one user map to another are stored as graphic objects that can be parameterized (linked symbol). They have the following transfer parameters:

- Caption -> Key label
- Disable -> Input for blocking key functionality (grayed out label)
- ScreenName -> Screen name for the operation screens to be opened (or screen group)
- ScreenNumber -> Number to be assigned to the screen (to highlight the respective key)

Communication Connection

The communication between visualization and control is executed using the BR-WS driver. To do this, generate the symbol entries for the PLC variables to be visualized (see [chapter "Communication Configuration" on page 46](#)).

The required symbol configuration for the predefined WinStudio visualization contains the following entries:

GAT with Central State Machine

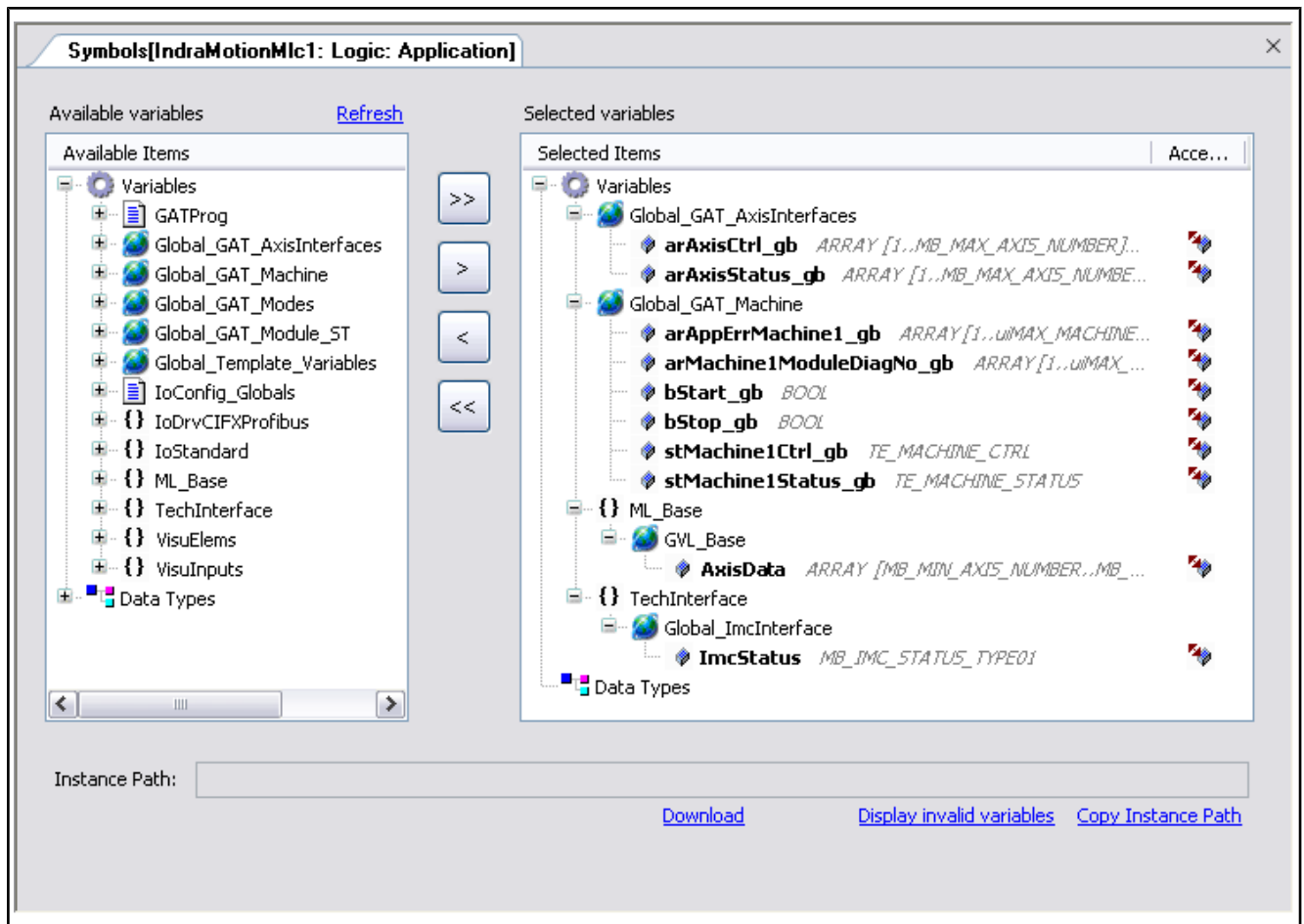


Fig. 8-45: Symbol configuration of the predefined WinStudio visualization



Symbol entries for variables from global variable lists that come from a library must be reinserted in the symbol configuration after the respective library version has been modified.

Setting the control's IP address:

The control address is configured in the BR-WS driver properties dialog:

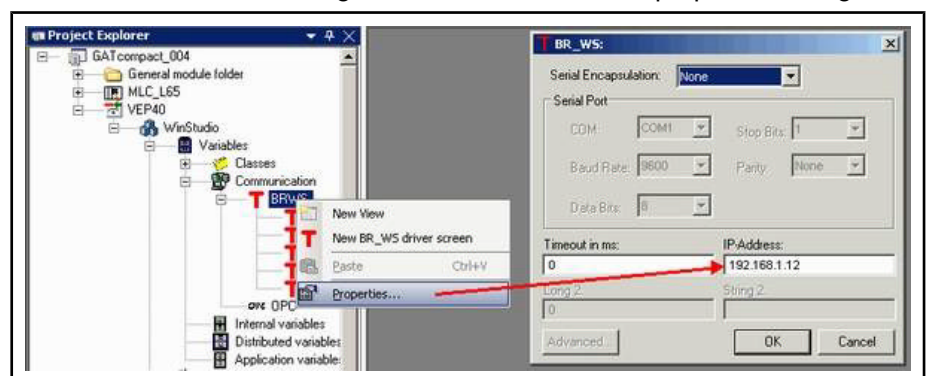


Fig. 8-46: Setting the control's IP address

In order to structure the communication, the variables were distributed in various driver sheets:

1. read cyclic fast: -> The values of these variables are to be read cyclically at short intervals (e.g. diagnostic numbers, axis positions, ...)

GAT with Central State Machine

2. read only once: -> These variables are read once after the visualization is started (e.g. axis names)
3. read after switching phase: These variables are read only once after a phase switch (e.g. if the axis was activated)
4. read - write cyclic: -> The values of these variables are to be both read and written (e.g. machine velocity, axis velocity for Setup mode, ...)
5. write only: -> These variables are only to be written through the visualization (trigger signals for clearing errors, ...)

To extend a driver sheet, the BR-WS browser can search the offline symbol file (*.XML).

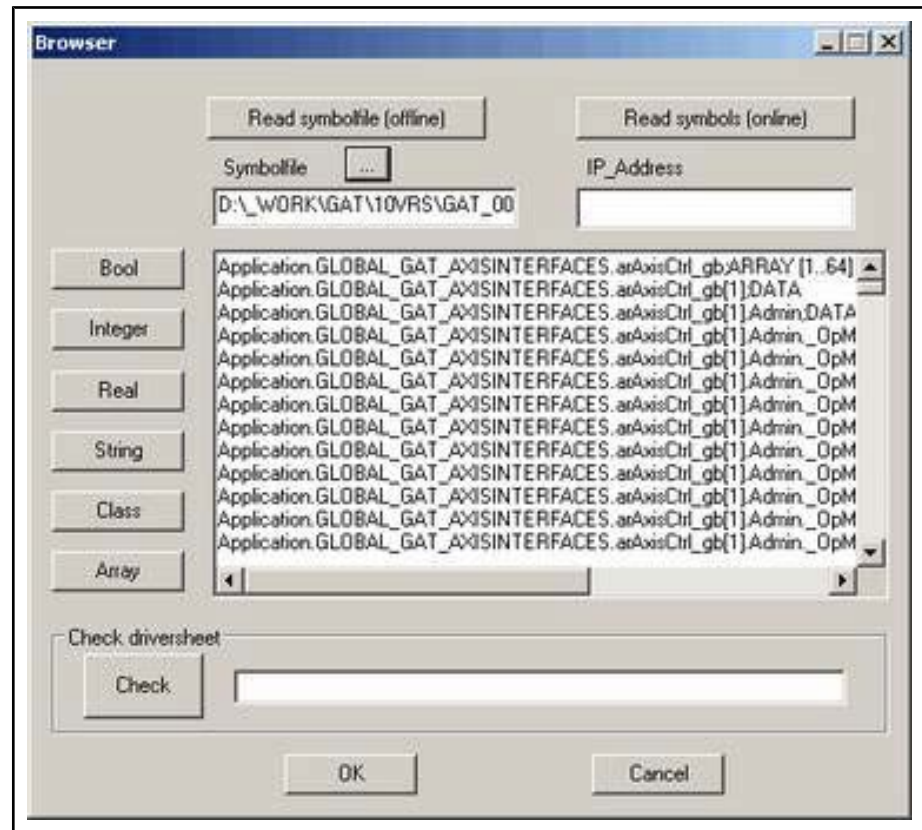


Fig.8-47: Extending the driver sheet

Multiple Language Capability

Multiple language capability for the visualization has been realized with the translation functionality integrated into WinStudio.

The translation file is stored as a *.CSV table, which provides a column for each language.

The table is predefined to include multiple languages, but only the "German" and "English" columns are completely filled out. For the rest of the languages only the drive diagnostics are included.

GAT with Central State Machine

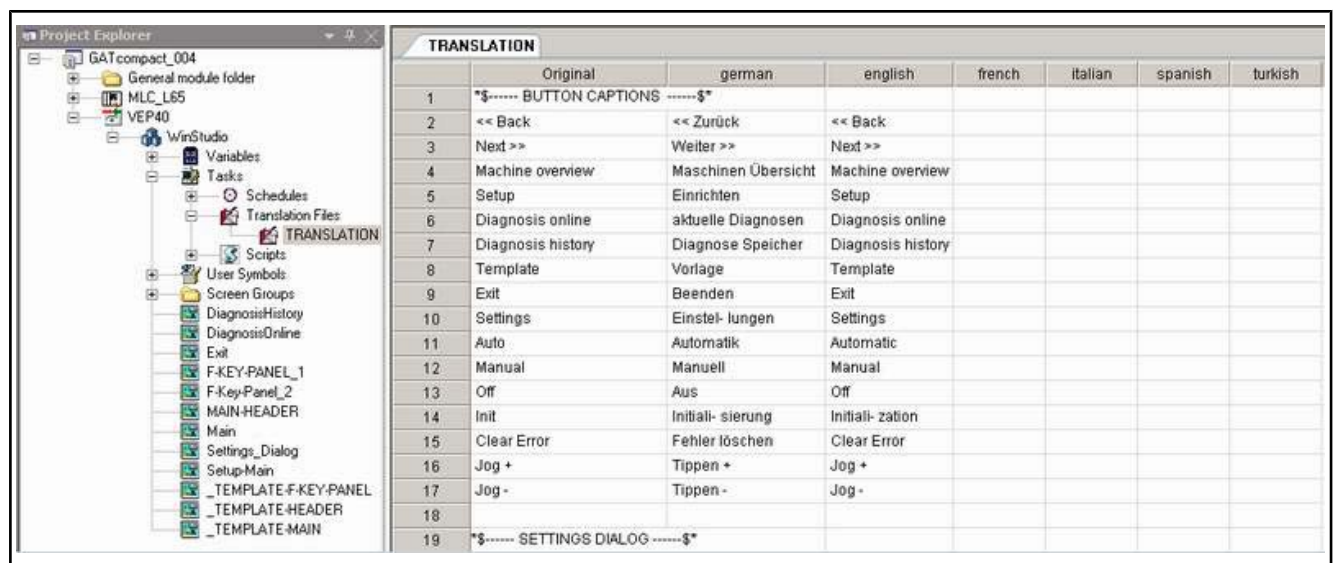


Fig. 8-48: Editing the translation file

8.11 Interfaces to GAT

8.11.1 General Information

The GAT interface data are made available in a variety of variable lists. Part of the data is made available in libraries.

8.11.2 Machine Data

The machine input data are summarized in the stMachine1Ctrl_gb (Global_GAT_MachineControl) structure of type TE_MACHINE_CTRL.

The following table describes the elements of the stMachine1Ctrl_gb structure.

I/O type	Name	Data type	Description
E	OpMode	TE_OPMODES_PREDEFINED	Specification for machine operation mode
E	OpMode_InitSuccessor	TE_OPMODES_PREDEFINED	Machine operation mode after successful initialization
E	OpMode_ErrorSuccessor	TE_OPMODES_PREDEFINED	Machine operation mode after successfully clearing the error
E ^{*)}	StateCtrl	TE_STATES_PREDEFINED	Internal use / Machine states
E	SkipColdstartAtStartup	BOOL	When starting the GAT program, a switch is made into parameterization mode (P2) to carry out the parameterization required for the application. This phase switch can be skipped with TRUE if the application does not require parameterization in P2.
E	DisableErrorHandling	BOOL	TRUE = In case of error, the process does not switch into the ERROR state. The error display continues and clearing the error is still possible.
E	ClearError	BOOL	Global clear error with a positive edge

^{*)} May not be described by the user. Setting this input is reserved for the machine organization unit.

E Input for writing

GAT with Central State Machine

The machine output data are summarized in the stMachine1Status_gb (Global_GAT_MachineControl) structure of type TE_MACHINE_STATUS.

The following table describes the elements of the stMachine1Status_gb structure.

I/O type	Name	Data type	Description
A	OpMode	TE_OPMODES_PREDEFINED	Feedback regarding the current machine operation mode
A	State	TE_STATES_PREDEFINED	Internal use / Feedback regarding the current machine state
A	Error	BOOL	TRUE = Global error has occurred
A	ErrorID	ERROR_CODE	Detailed information if DiagNo = 16#F0FB 0000 / 16#F0AA xxxx
A	ErrorIdent	ERROR_STRUCT	Detailed information if DiagNo = 16#F0FB 0000 / 16#F0AA xxxx
A	DiagNo	DWORD	Diagnostics number according to the IndraMotion MLC specification DiagNo = 16#F0AE xxxx: Application error (e.g. safety door is open) DiagNo = 16#F0FB 0000: FB error DiagNo = 16#F0AA xxxx: GAT internal error
A	DiagMessage	STRING(60)	Diagnostic text of the control / axis

A Output for reading

8.11.3 Data for Error Handling

The inputs and outputs for error handling are contained in the machine structures mentioned above.

8.11.4 Module Data

The modules are controlled by the machine organization unit. The module data can be accessed from an external position for diagnostic purposes.

The input data for all modules are summarized in an array arModuleCtrl of type TE_CMI_CONTROL within the function block bMachine1_gb (Global_GAT_MachineControl).

The following table describes the elements of the Global_GAT_MachineControl.fbMachine1_gb.arModuleCtrl structure.

I/O type	Name	Data type	Description
E	Enable	BOOL	TRUE = Module processing starts
E	ClearError	BOOL	TRUE = Module clear error
E*)	State	TE_STATES_PREDEFINED	Command for module state
E*)	StateTransient	BOOL	TRUE = Module is in transition to the commanded state
E*)	DisableTransient	BOOL	TRUE = Module deactivation was started

E*) Input for writing
May not be described by the user. Setting this input is reserved for the machine organization unit.

GAT with Central State Machine



Elements for arModuleCtrl are written in the "Machine" FB and for this reason, from an external position they can only be read.

The output data for all modules are summarized in an array arModuleStatus of type TE_CMI_STATUS within the function block bMachine1_gb (Global_GAT_MachineControl).

The following table describes the elements of the Global_GAT_MachineControl.fbMachine1_gb.arModuleStatus structure.

I/O type	Name	Data type	Description
A*)	InOperation	BOOL	TRUE = the module is processed
A*)	State	TE_STATES_PREDEFINED	Feedback regarding module state
A	StateLocked	BOOL	TRUE = Switching to the following state has been blocked by the user
A	TransientLocked	BOOL	TRUE = The transition to the stationary section of the commanded state was blocked by the user.
A*)	TransientState	TE_STATES_PREDEFINED	Display of the state into which the module is currently switching (commanded state not yet reached, related activities are running)
A	DisableLocked	BOOL	TRUE = Activities for module deactivation are still running
A*)	Error	BOOL	TRUE = Module error has occurred
A*)	ErrorID	ERROR_CODE	Detailed information if DiagNo = 16#F0FB 0000
A*)	ErrorIdent	ERROR_STRUCT	Detailed information if DiagNo = 16#F0FB 0000
A*)	DiagNo	DWORD	Diagnostics number according to the IndraMotion MLC specification DiagNo = 16#F0AE xxxx: Application error (e.g. safety door is open) DiagNo = 16#F0FB 0000: FB error DiagNo = 16#F0AA xxxx: GAT internal error
A*)	DiagMessage	STRING(60)	Diagnostic text of the control / axis

A
*)

Output for reading
May not be described by the user. Setting this output is reserved for the machine organization unit.

8.11.5 Axis Data

I/O type	Name	Data type	Description
E	arAxisCtrl_gb (Global_GAT_AxisInterfaces)	ARRAY OF TE_AXIS_CONTROL_TYPE01	Commands for the axes
A	arAxisStatus_gb (Global_GAT_AxisInterfaces)	ARRAY OF TE_AXIS_STATUS_TYPE01	Feedback from the axes
I/O	AxisData (GVL_Base from the ML_Base library)	ARRAY OF ML_AXISDATA_SM	Firmware structure with axis inputs and outputs

E
A
I/O

Input for writing
Output for reading
Structure with inputs and outputs

GAT with Central State Machine



Elements for arAxisCtrl_gb are written in the "Machine" FB and for this reason they can no longer be used externally.

8.11.6 Control Data

I/O type	Name	Data type	Description
E	ImcCtrl (Global_ImcInterface from the ML_TechInterface library)	MB_IMC_CONTROL_TYPE01	Commands for the control
A	ImcStatus (Global_ImcInterface from the ML_TechInterface library)	MB_IMC_STATUS_TYPE01	Feedback from the control

E

Input for writing

A

Output for reading



Elements for ImcCtrl_gb are written in the "Machine" FB and for this reason they can no longer be used externally.

8.12 How To: Typical User Activities

8.12.1 Access to Axis Data

For access to axis data, in every module the **Axes** structure is defined as the central access point for all of the axes assigned to the module. All relevant data for an axis can be addressed using the Axes structure.

The structure is designed such that the axis data is accessed using the name of the axis. This generates program code that is easily readable.

The following data are available below "Axes":

- Axes.<Name>.AxisData => Axis data structure: Status data for the firmware and access to the axis data container
- Axes.<Name>.Ctrl => Control structure for the AxisInterface
- Axes.<Name>.Status => Status structure for the AxisInterface
- Axes.<Name>.DV => Access to direct variables (function variables)
- Axes.<Name>.AxisRef => Axis reference structure for FB calls (e.g. MB_ReadParameter)

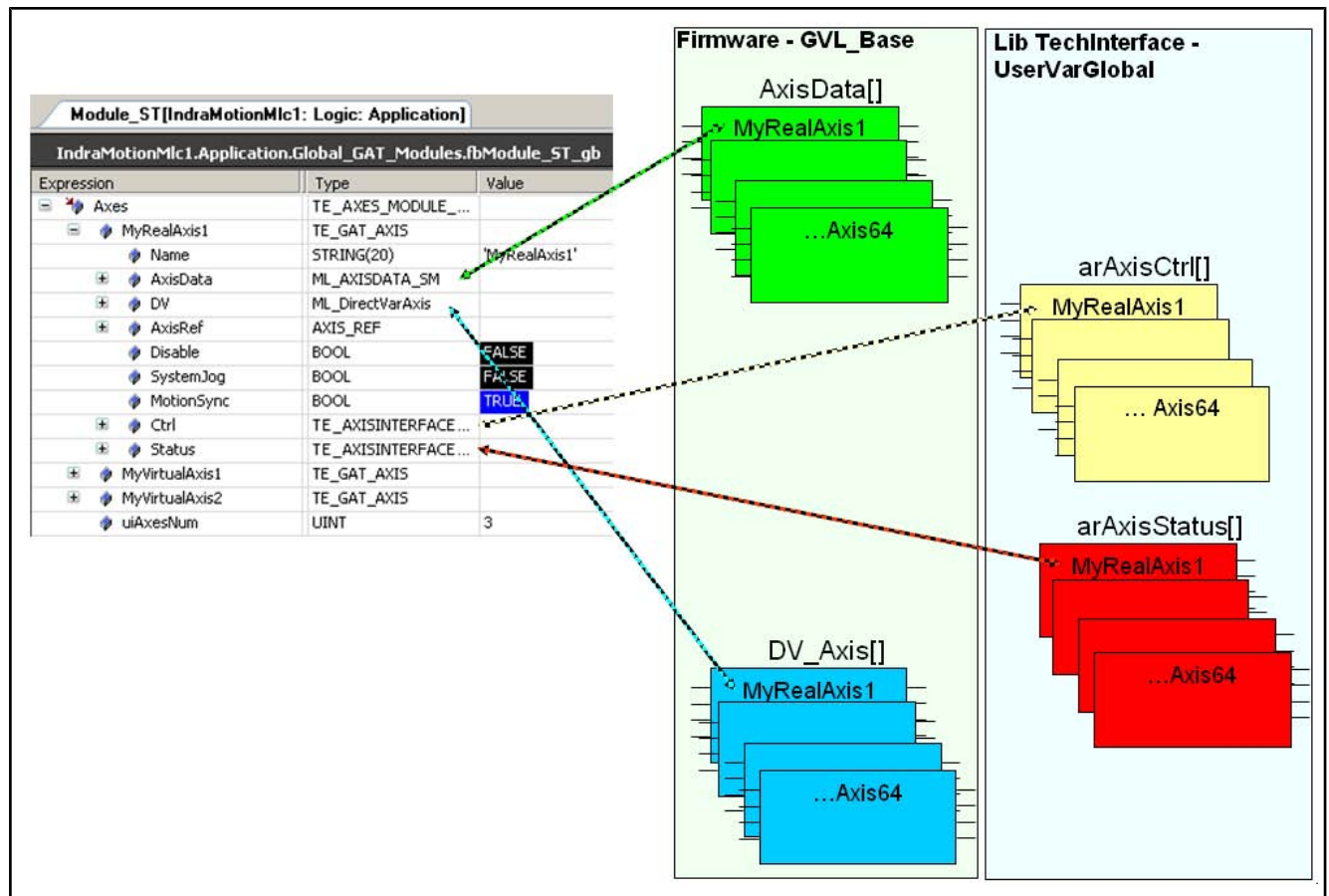


Fig.8-49: Overview / axis data access

To program loops across all of the axes of a module, each module has the method **mGetAxis()** and the element **Axes.uiAxesNum**.

Example: Switching all of a module's axes into ModeAb:

Declaration:

Program:

```
pCurrentAxis:    POINTER TO TE_GAT_AXIS;
uiLoopIndex:     UINT; // For loop index
```

Implementation:

Program:

```
// ----- Error reaction example 1 -----
// Decelerate all module axes to standstill and turn power off
// --> switch all axes off to ModeAb
// -----
FOR uiLoopIndex := 1 TO Axes.uiAxesNum DO
  IF mGetAxis(AxisNumber:=uiLoopIndex, PointerToAxis => pCurrentAxis) THEN
    pCurrentAxis^.Ctrl.Admin._OpMode.en := ModeAb;
  END_IF
END_FOR
```

8.12.2 Adding an Axis

The following steps are required for adding an axis:

- Add the axis into the IndraWorks project ("Drag to the SERCOS node...")

GAT with Central State Machine

- When using the SystemOverview_All_Axis visualization: add a frame for the new axis and set the reference to the new axis in the properties (see also [chapter 7.10.2 "Adding an Axis" on page 155](#)).
- Add the axis using the GAT wizard (see the figure below)

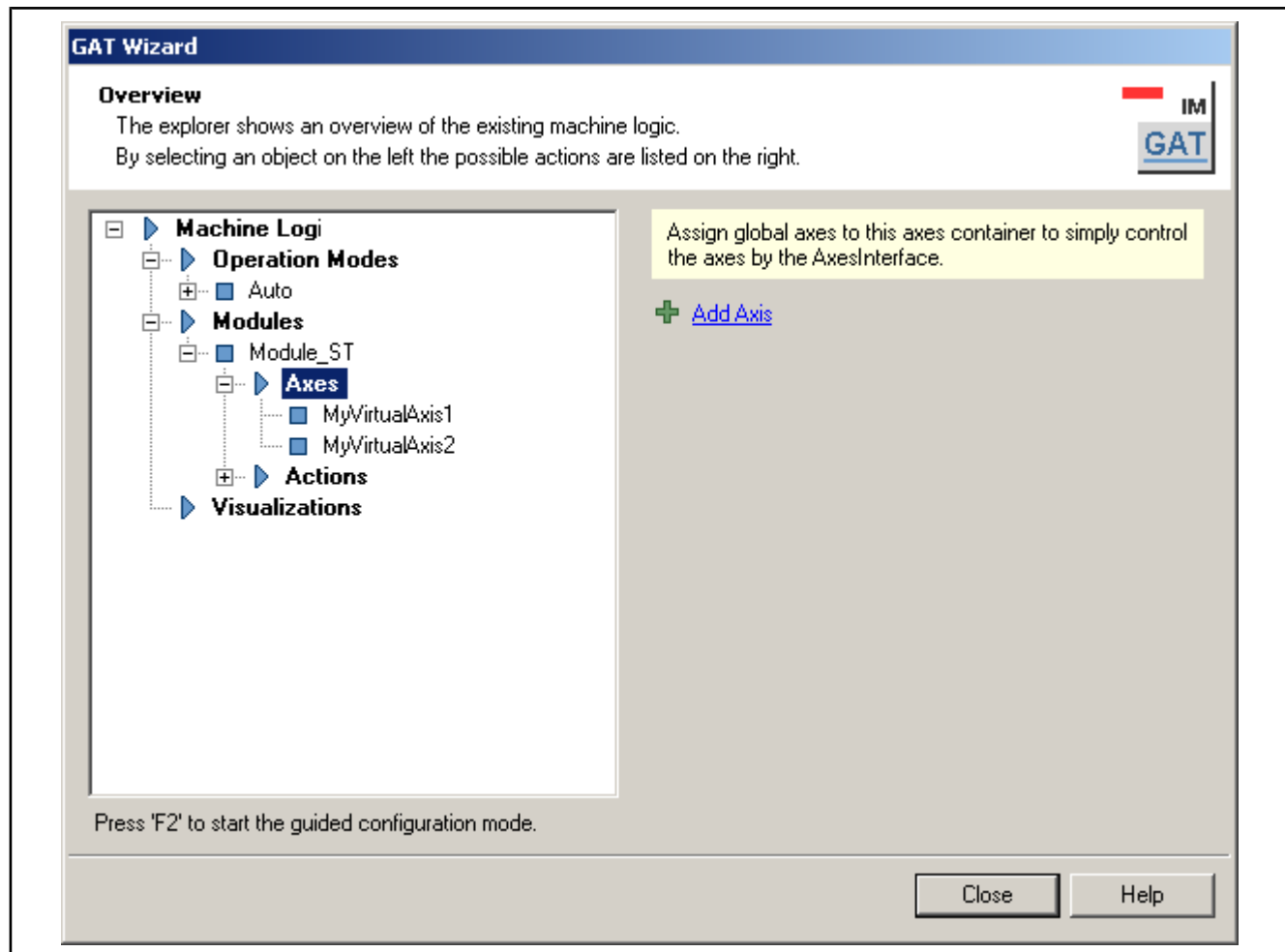


Fig. 8-50: Adding an axis using the GAT wizard

8.12.3 Removing/renameing an Axis



The wizard changes only the name in the axis declaration. If the axis is already being used in the implementation or internal visualization, the new axis name must be changed at these positions using "Find/Replace".

The following steps are required for removing/renameing an axis:

- Removing an axis from/renameing an axis in the IndraWorks project
- When using the SystemOverview_All_Axis visualization: remove the frame for the deleted axis and change the reference to the axis in the properties.
- Remove/rename the axis using the GAT wizard (see the figure below).

GAT with Central State Machine

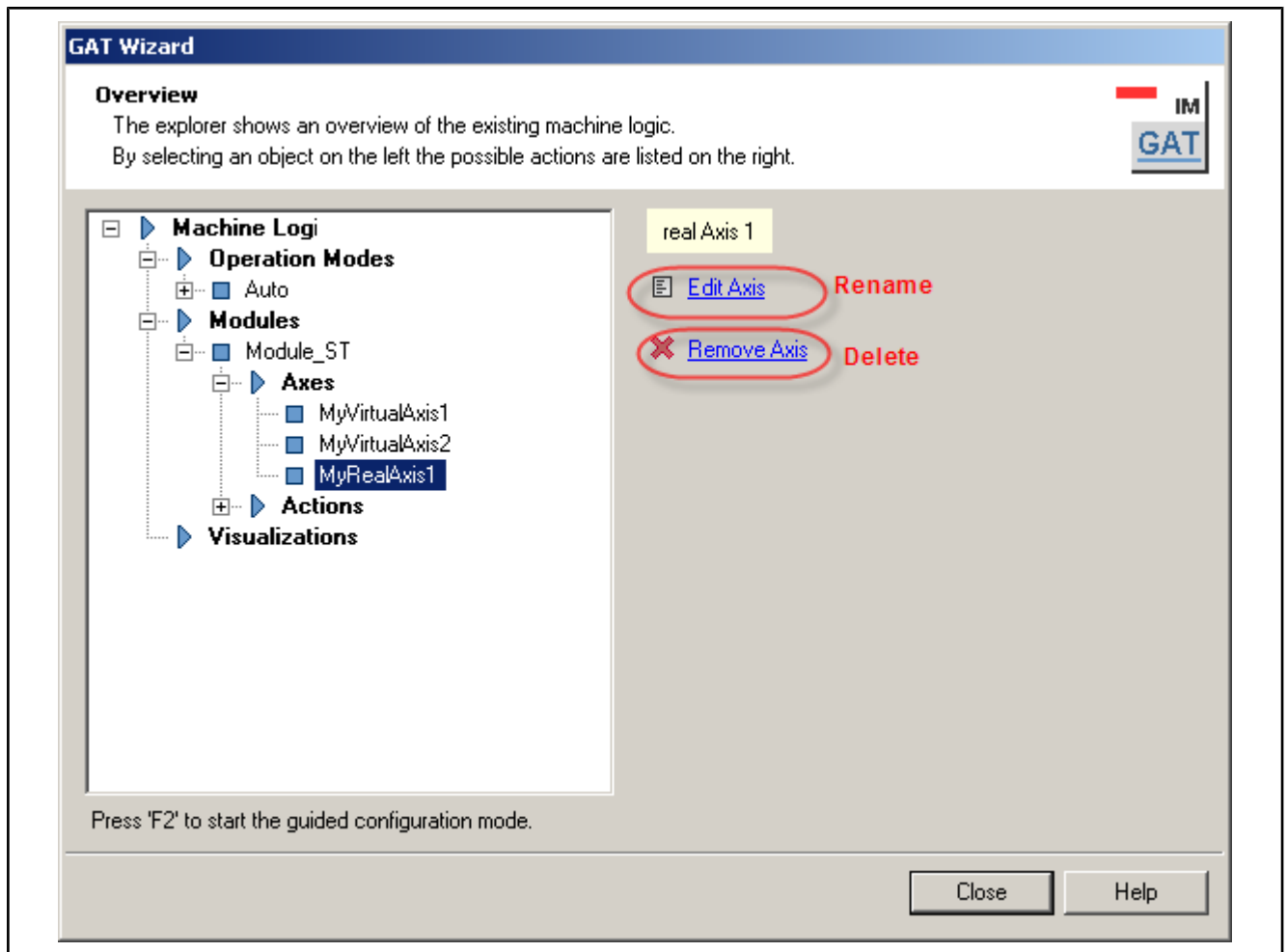


Fig.8-51: Removing/renaming the axis with the GAT wizard

GAT with Central State Machine

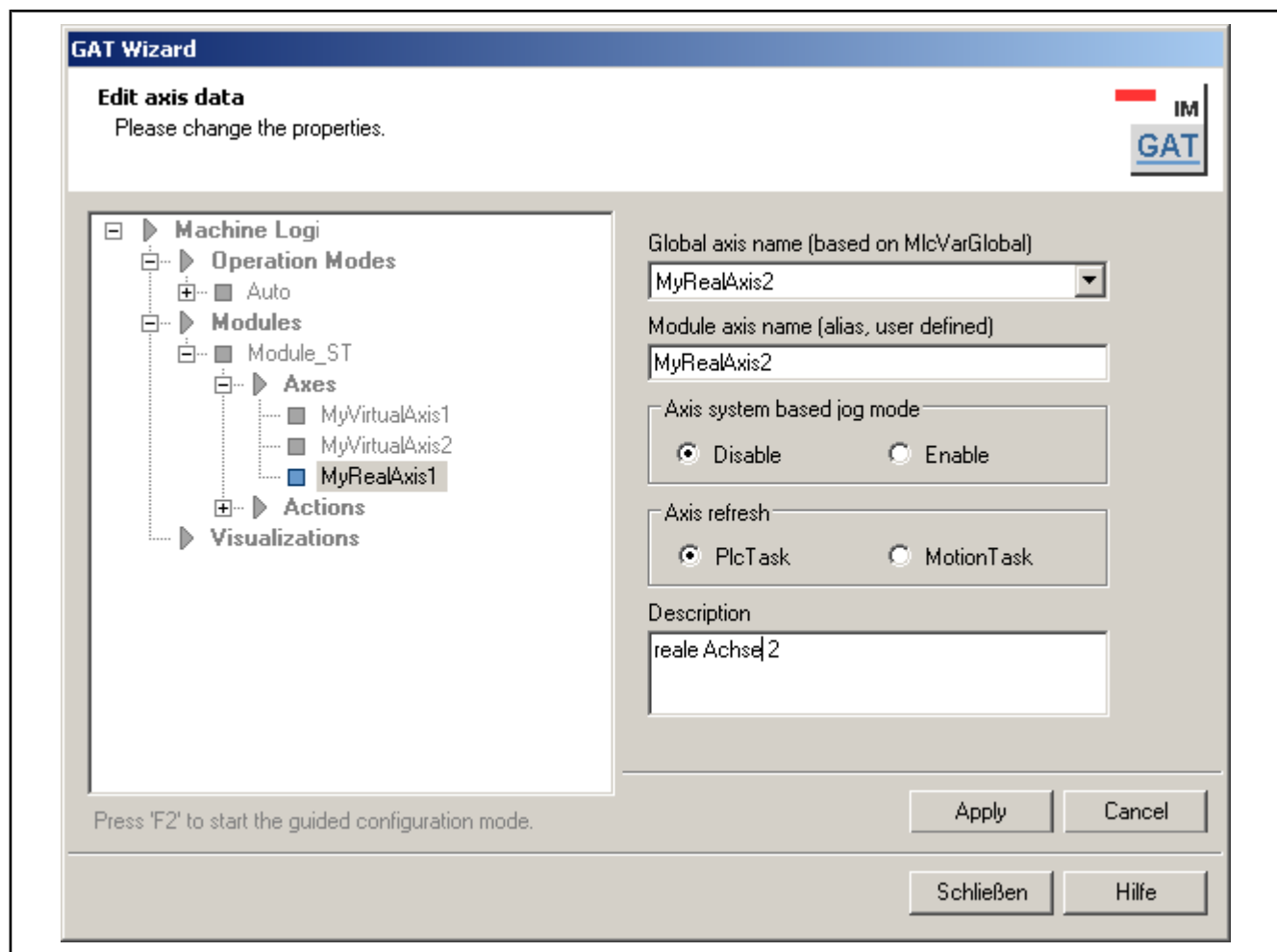


Fig.8-52: Removing/renameing the axis with the GAT wizard

8.12.4 Adding an Operation Mode

The GAT wizard makes it easy to add operation modes to the machine. The wizard can also be used to delete or rename the operation modes.

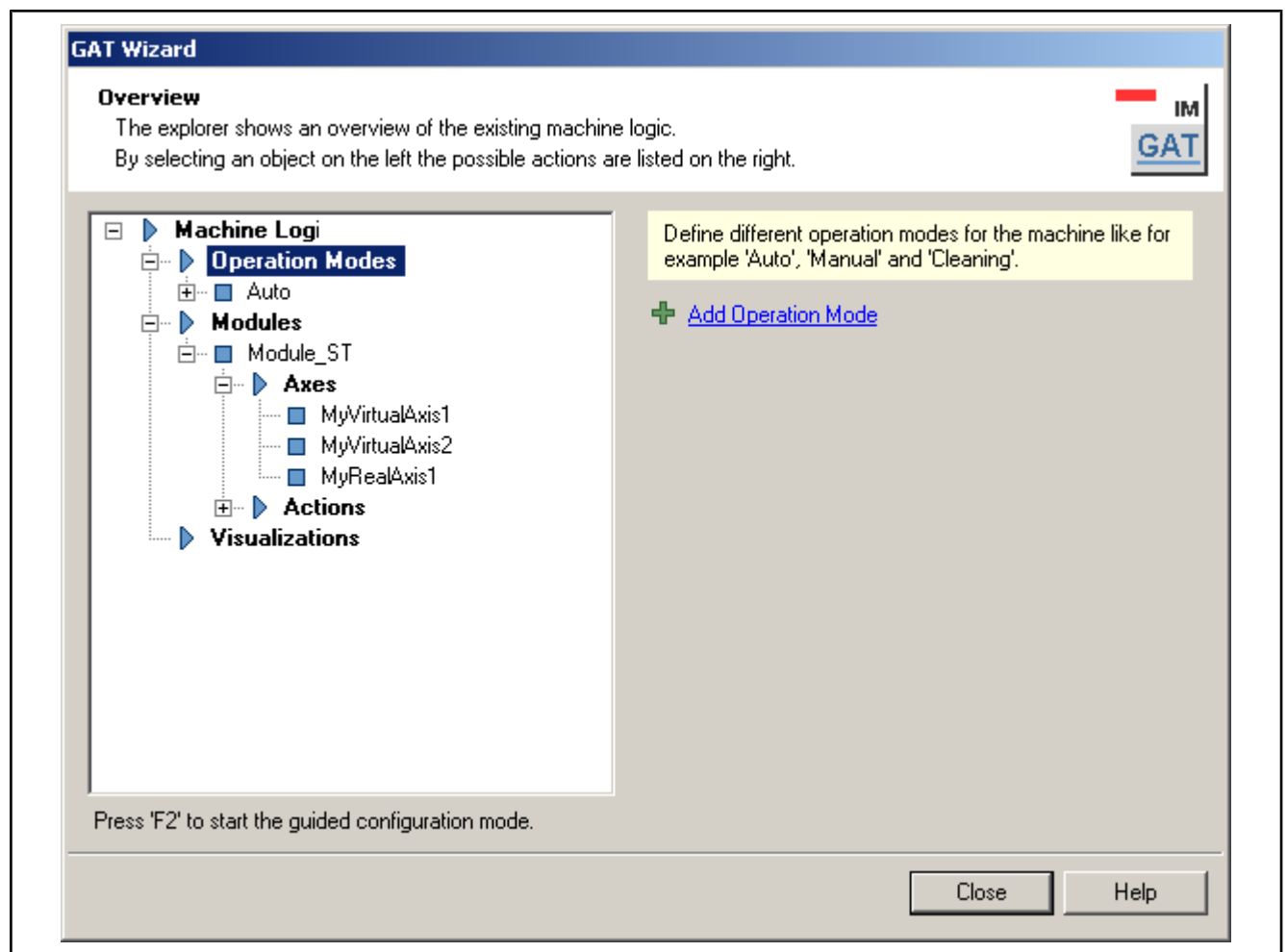


Fig. 8-53: Adding an operation mode with the GAT wizard

8.12.5 Selecting the Desired Operation Mode after Clear Error

The variable **fbMachine_gb.Ctrl. OpMode_ErrorSuccessor** affects the behavior of the application after clear error. The user can change the value of this variable at any time. The variable is evaluated by the machine organization unit immediately after clearing an error.

By default, the value of this variable is "STATE_UNDEFINED". If an error is reset under this conditions, the application exits the error state and transitions through the STATE_INIT_WARMSTART state into the STOPPED operation mode state that was previously active.

For example, the PRODUCING state of the AUTO operation mode was active before an error occurred. If the error is deleted, the application switches into the STATE_INIT_WARMSTART state and then into the STATE_AUTO_STOPPED state.

If the value of the variable **fbMachine_gb.Ctrl. OpMode_ErrorSuccessor** is a specific operation mode, then the application is commanded into the STOPPED state of this operation mode after the error is cleared.

For example, before an error occurred, the PRODUCING state of the AUTO operation mode was active. In addition, "fbMachine_gb.Ctrl. OpMode_ErrorSuccessor=OPMODE_MANUAL" applies. After the error is cleared, the application switches directly into the STATE_MANUAL_STOPPED state.

The initial state of INIT is STATE_INIT_COLDSTART.

GAT with Central State Machine

If an error occurs while the application is in the STATE_INIT_COLDSTART or STATE_INIT_WARSTART state, the application returns to exactly this state after clearing the error

8.12.6 Superordinate Logic for Operation Mode Specification

In some cases, selecting the machine's operation mode depends on additional conditions. The recommended place for implementing this logic is in the program "GATProg" before calling the "Machine" FB. Here the conditions can be queried and the operation mode specified using the global variable **stMachine1Ctrl_gb.OpMode**.

8.12.7 Adding States and Transitions

If the activities to be executed within an operation mode are very complex, a division into several states is recommended. In this way, the application is clearly structured and easier to understand.

The GAT wizard makes it easy to add states and conditions between states (transitions).

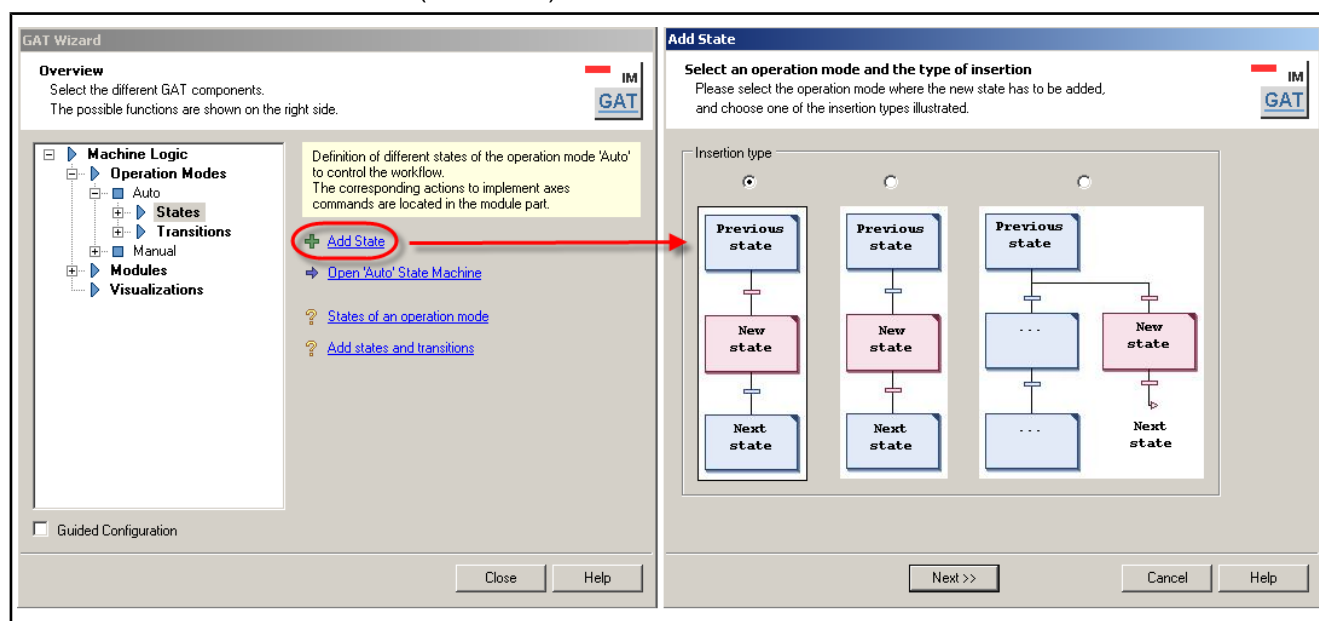


Fig. 8-54: Adding states

When adding states within an operation mode, it must be decided if a new state is to be added after a highlighted state or after a highlighted condition (transition). In addition, branches can be created (see the previous figure).

When a new state is generated, actions that correspond to this state are created in all of the modules that exist at this point.

The generation of transitions allows one or more states to be bypassed, depending on a specific condition (see the following figure).

GAT with Central State Machine

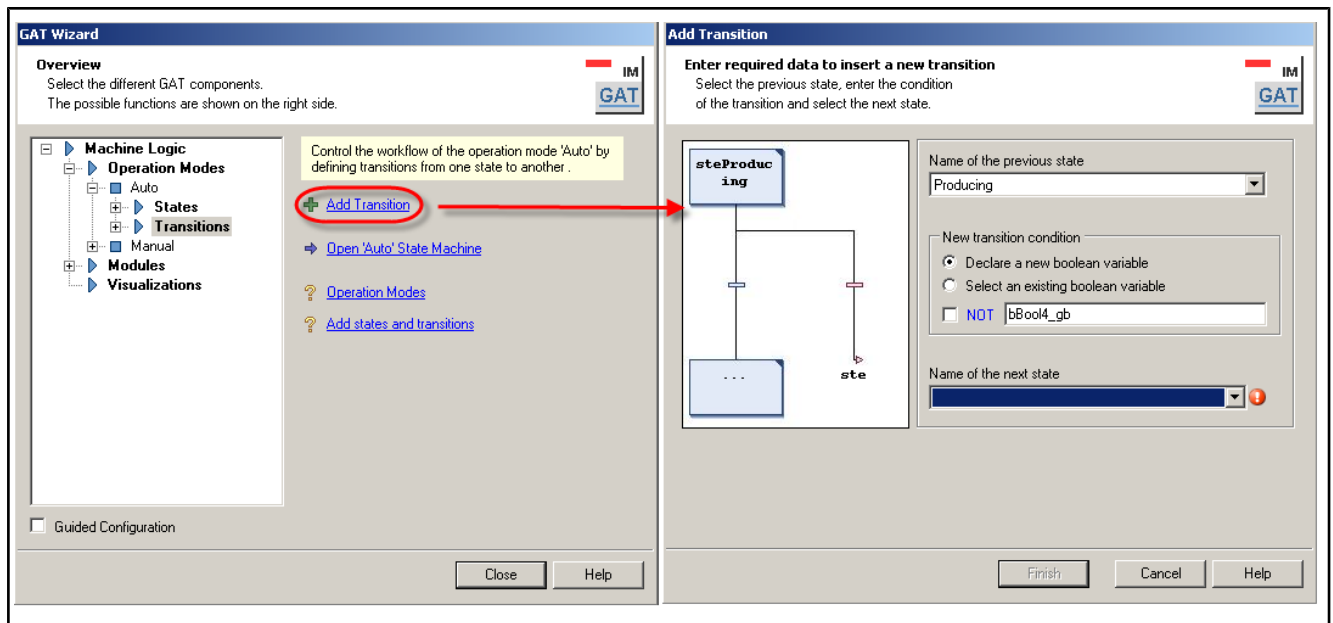


Fig. 8-55: Adding transitions

8.12.8 Adding an New Module

The GAT wizard makes is easy to add modules to the machine. The wizard can also be used to delete or rename the modules.

In the new module, actions that correspond to the state are created in all of the states that exist at this point.

GAT with Central State Machine

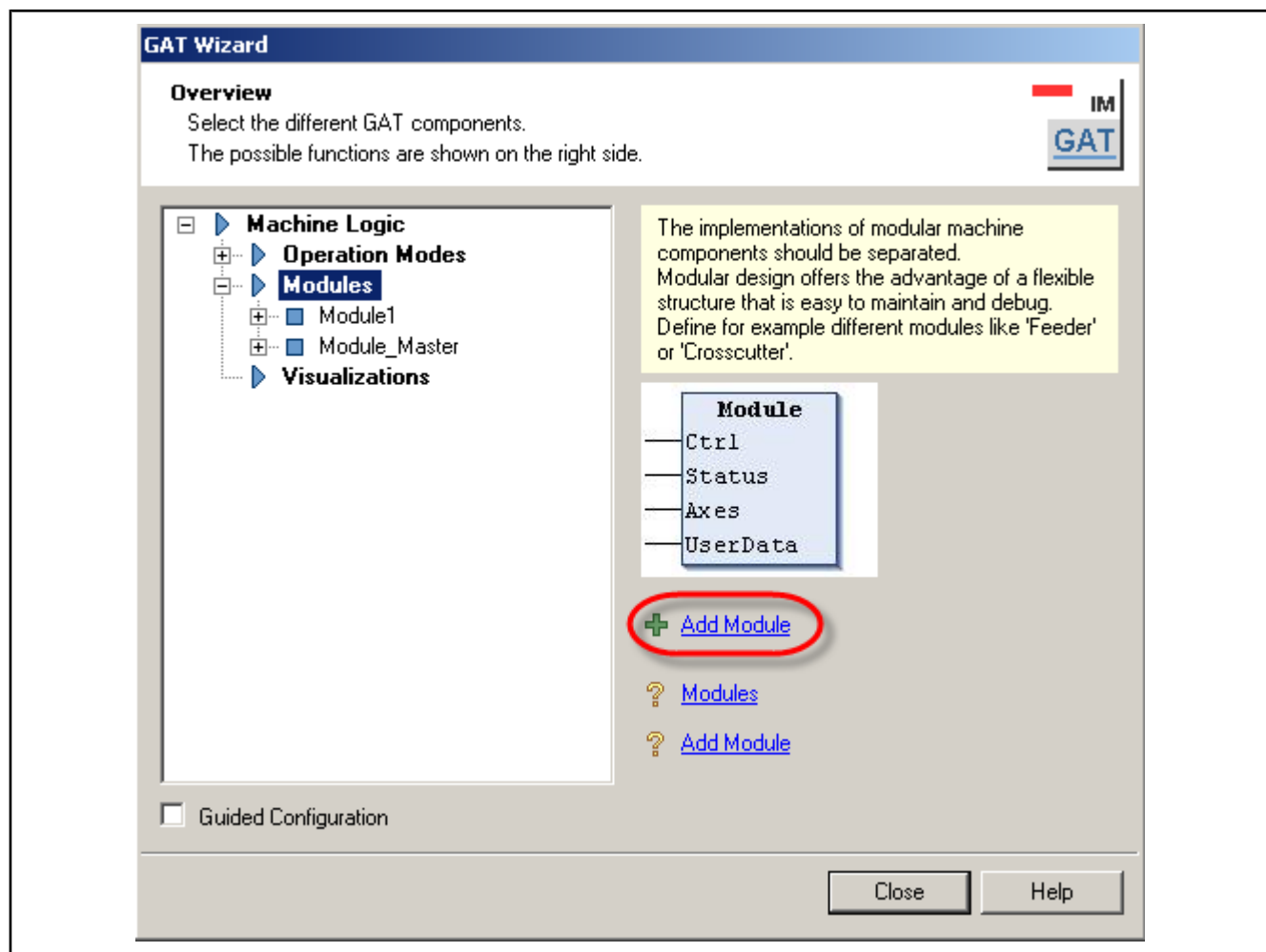


Fig.8-56: Adding a module

8.12.9 Adding Another Instance of a Module

The GAT wizard makes it easy to add further instances of existing modules to the machine.

The interfaces remain the same, but other global axes (an axes) and I/O data (an UserData) must be assigned.

GAT with Central State Machine

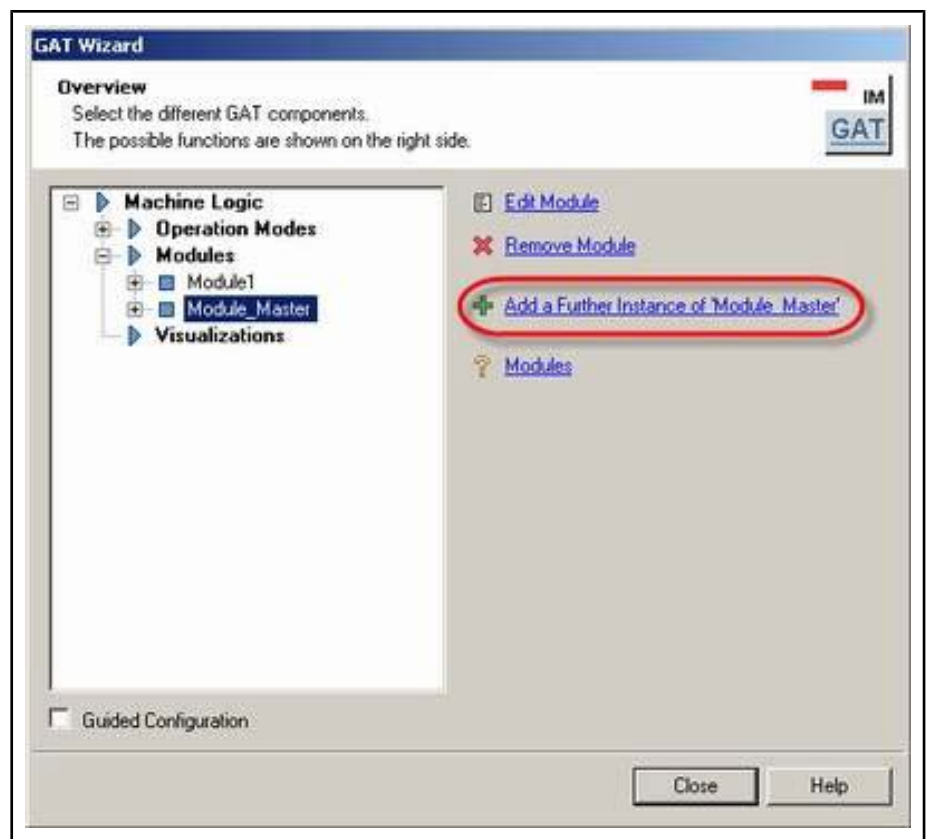
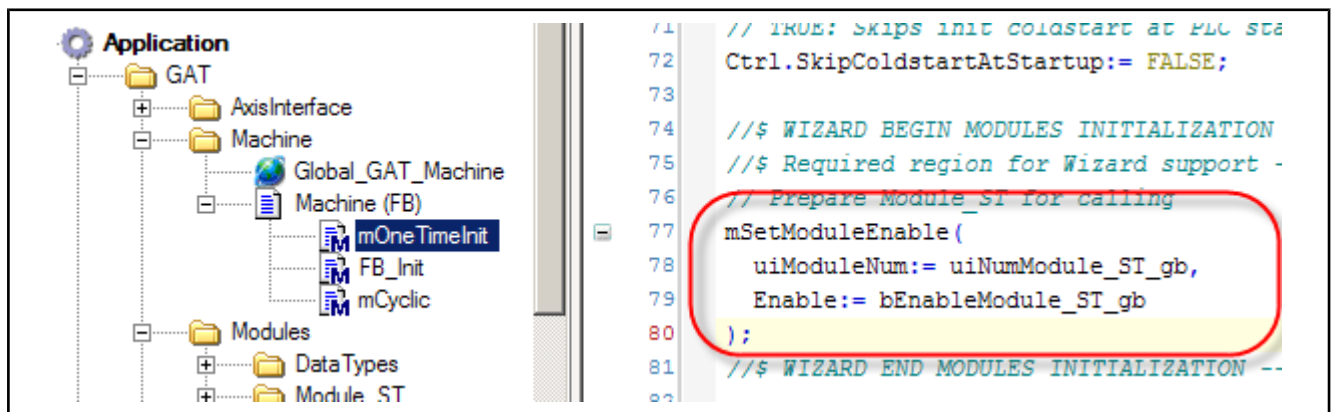


Fig. 8-57: Adding another instance of a module

8.12.10 Deactivating a Module

In order to exclude modules from processing, they can be deactivated. If a module is deactivated, the module code will no longer be processed. The method "mSetModuleEnable()" can be used to activate or deactivate modules. By default, when a module is added using the GAT wizard, it is activated in the "mOneTimeInit" method of the Machine FB.



A module can be deactivated either directly after the PLC project is downloaded again or following runtime. By changing the initial value of the "Enable" variable to FALSE (see the following figure), the module is deactivated directly after the PLC download. This procedure allows just one project to be maintained, even for various machine versions, while simply switching off modules that are not present.

GAT with Central State Machine

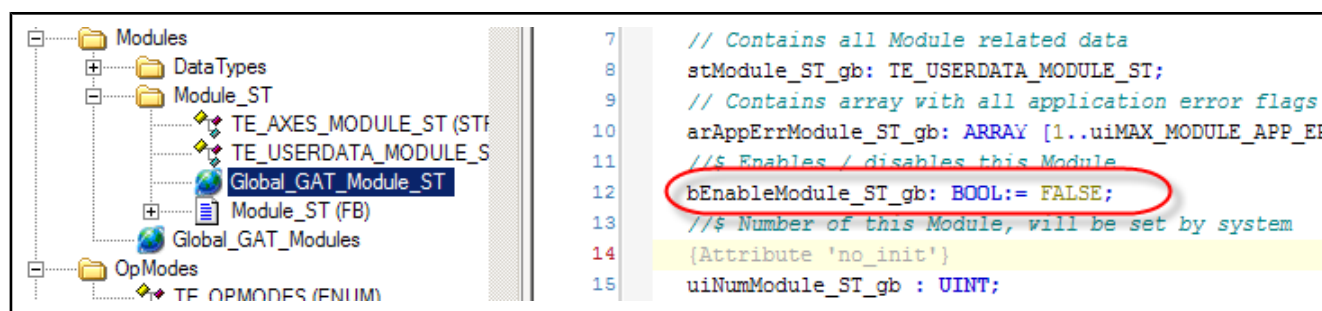


Fig.8-59: Activating / deactivating a module by changing the initial value

The module can be deactivated during runtime even if the program is running. To do this, the following procedure must be used:

- First, the user must bring the machine (and all of the assigned modules) into a secure initial state (e.g. the STOPPED state in Automatic operation mode). For safety-related reasons, the command for deactivating the module is not executed if the secure initial machine state is not achieved.
- Then the "mSetModuleEnable(Enable:=FALSE...)" method is used to initiate the function for module deactivation from the application program.
- The GAT sets the "DisableTransient" control element in the control structure module to TRUE for at least one cycle.
- The action "actDisable" is then processed in the deactivated module. In this action, the code necessary for deactivating the module can be implemented. The action "actDisable" is processed for at least one cycle. If the module deactivation requires several cycles, the processing can be extended by setting the module status element "DisableLocked := TRUE". The action "actDisable" is processed until "DisableLocked" is reset to FALSE again. Then the module code will no longer be processed.

8.12.11 Reusing Modules

The GAT is designed such that the functional units (modules) developed can also be reused in other projects built using the GAT.

To do this, simply select the folder of the desired module and export the related data by using the context menu (see the following figure).

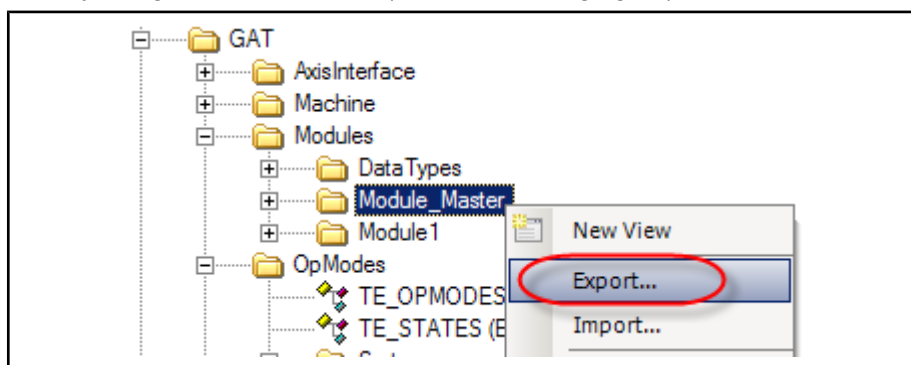


Fig.8-60: Exporting a module

Select all of the elements in the following dialog.

Import the file generated during the export into the new project. To do this, highlight the "Modules" folder and use the "Import" functionality in the related context menu (see the following figure).

GAT with Central State Machine

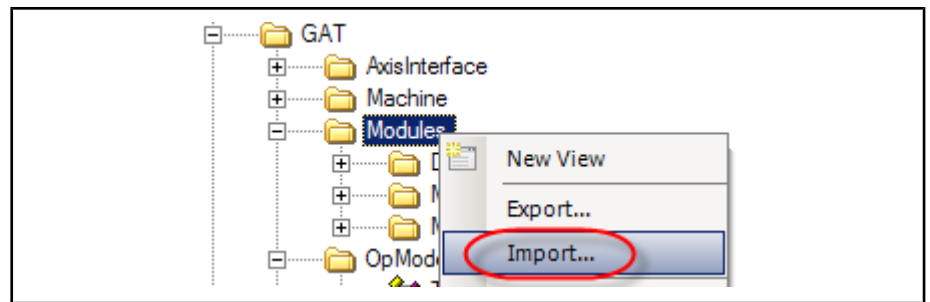


Fig. 8-61: Importing a module

When using a module in another project (importing), make sure that all states contained in the module are also defined in the new machine (project) (see the following figure).

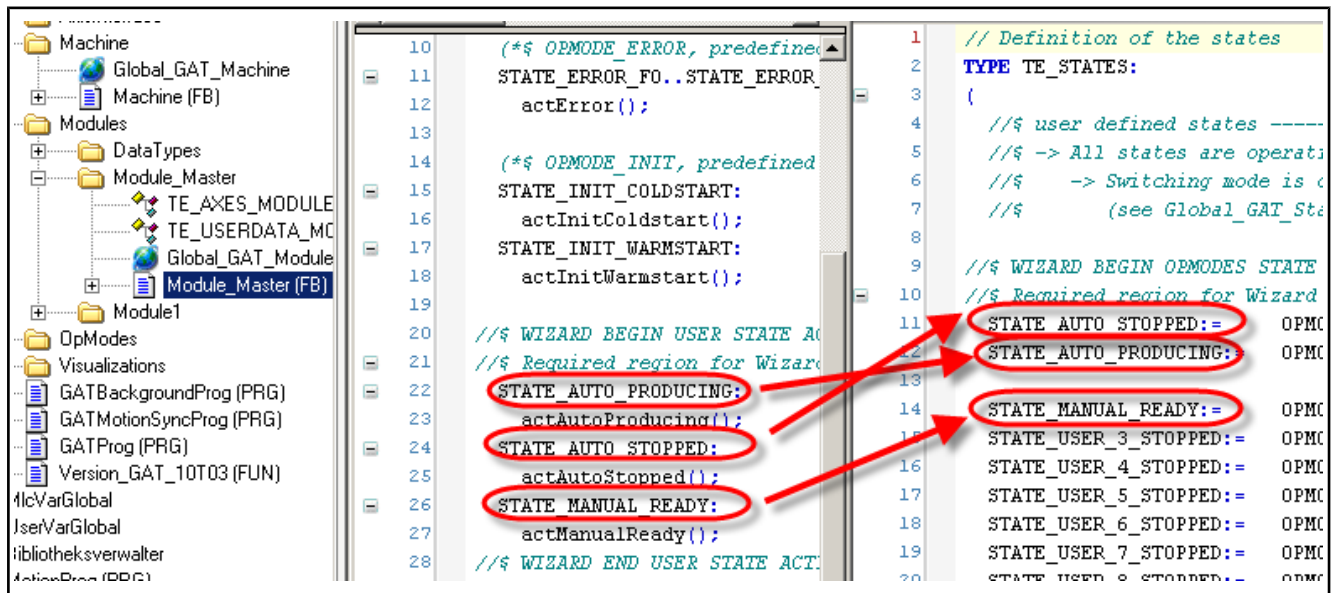


Fig. 8-62: Declaration of the states

After the module is imported, an associated entry must be added to the "mOneTimeInit" method of the FB "Machine" in order to activate/deactivate the module (see the following figure).

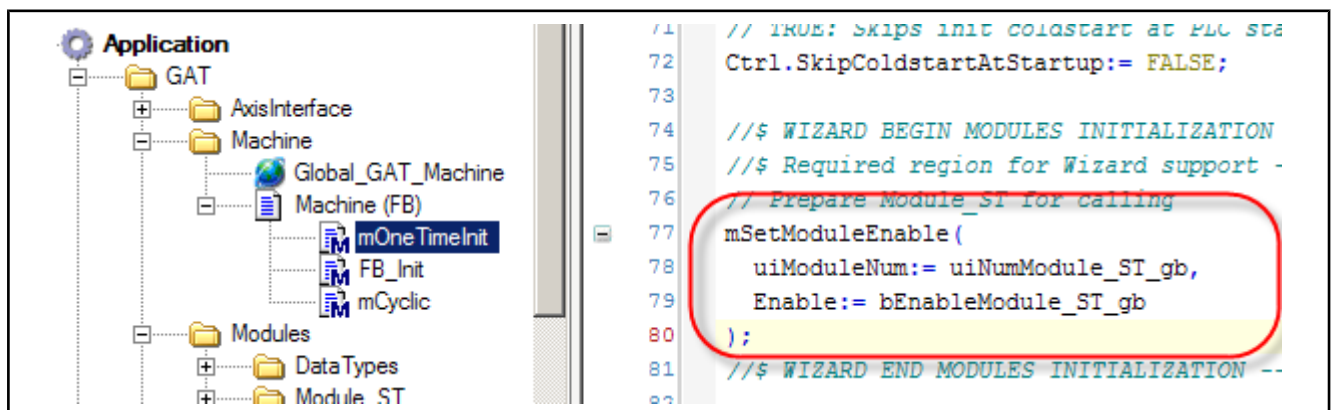


Fig. 8-63: Extending the method "mOneTimeInit" of the FB Machine after importing a module

GAT with Central State Machine

8.12.12 Using Global and Address Variables in Modules

Because a module can be used more than once in an application (instantiation) and because it can be reused in another application, there may be no direct access to a global or address variable in a module.

All global and address variables that are to be used in the module must be declared as interface variables in the module data container **"UserData"**. The assignment of the global or address variable to the corresponding element in the UserData container is carried out only once in the machine. The element within the module is accessed using **UserData.<Element>**.

The following example demonstrates how access is handled:

Two variables are declared in the global variable list **MlcVarGlobal**: a global input and a global output (see the following figure)

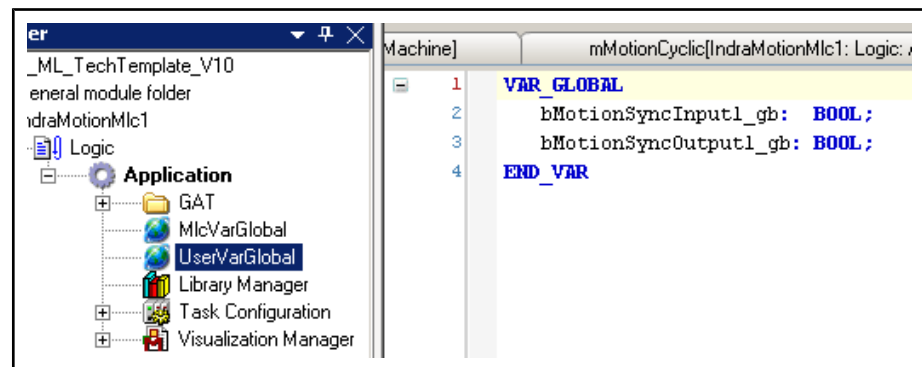


Fig. 8-64: Declaration of one Boolean input and one Boolean output variable in the global variable list **UserVarGlobal**

If necessary, these variables can also be mapped into physical addresses. In this specific case, the global input variable is mapped to the first bit of the onboard inputs and the global output variable to the first bit of the onboard outputs of the control (see the following figure).

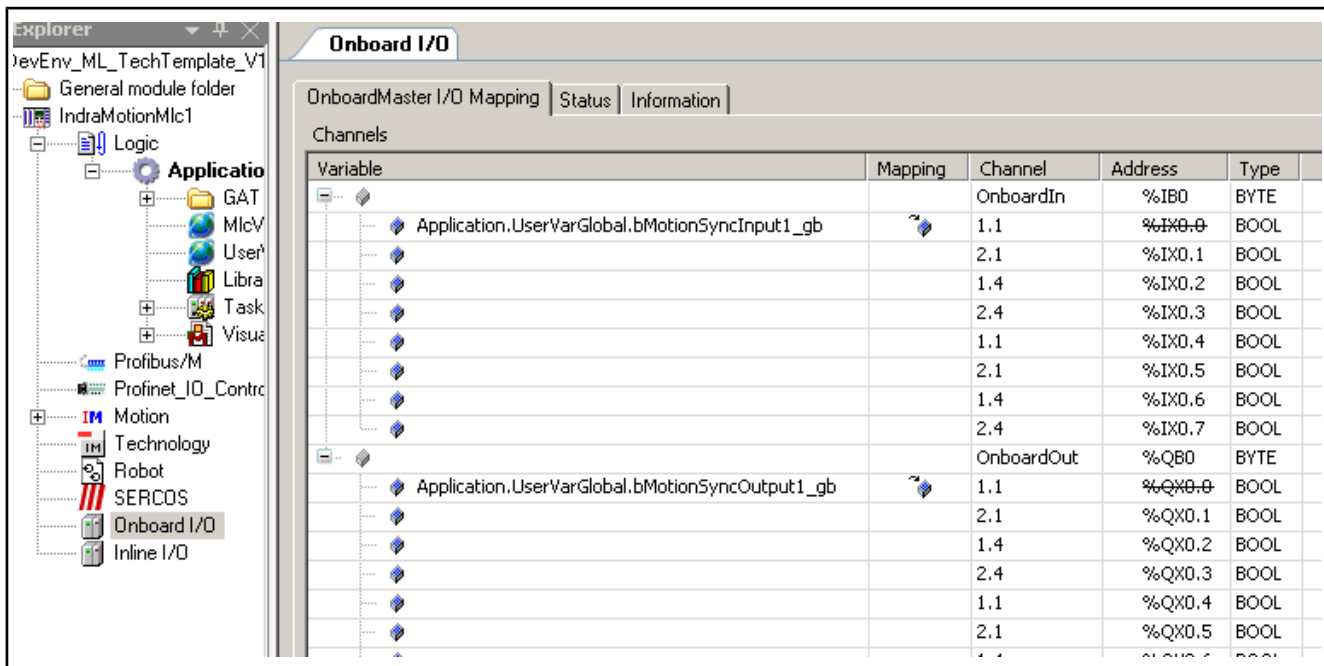


Fig. 8-65: Mapping of Boolean global variables on onboard inputs/outputs

In the module in which the global variables are to be used, the reference variables must be declared in the UserData data structure. The reference variables

GAT with Central State Machine

must correspond with the type of the global variables to be used (see the following figure).

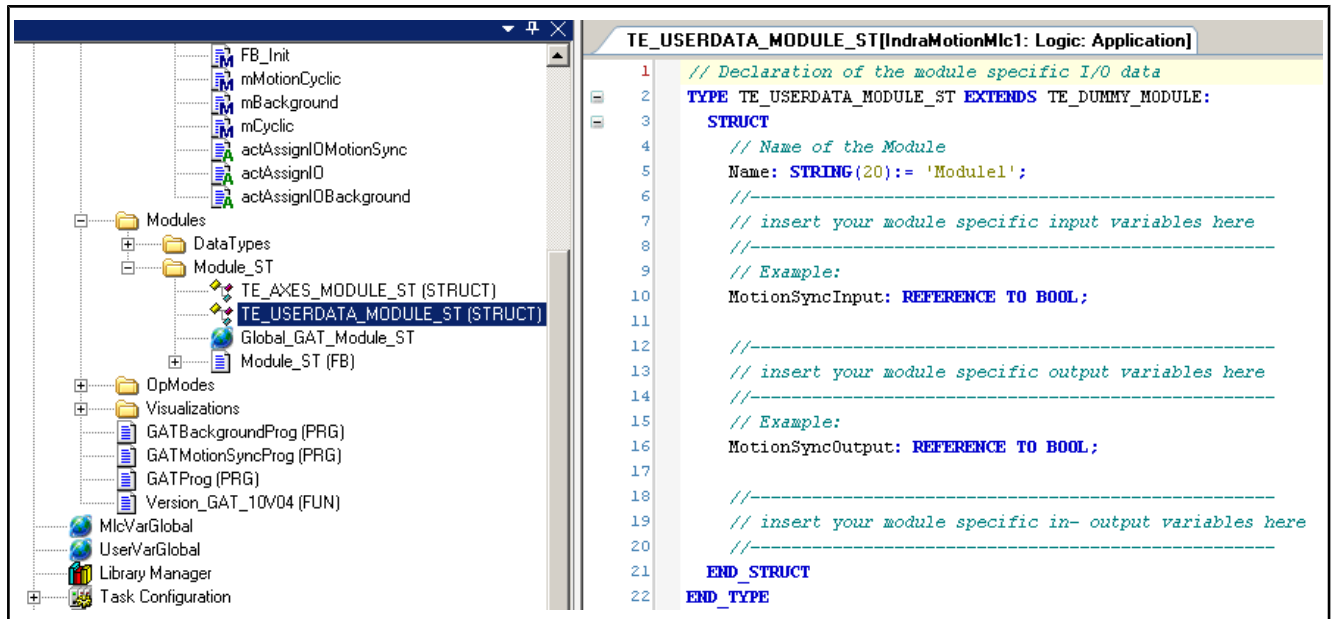


Fig. 8-66: Declaration of reference variables in the module for the global variables that are to be used in the module

Finally, the connection between the global variables and the corresponding interface variables (reference variables) for the module must be made. In principle, global and address variables are updated with reference to tasks. For this reason, there are task-related actions in the **FB Machine** in which the global and address variables are linked with the corresponding module variables. For this reason, the variables to be updated in the

- **MotionSyncTask** in the **actAssignIOMotionSync** action
- **PlcTask** in the **actAssignIO** action
- **BackgroundTask** in the **actAssignIOBackground** action

must be connected with each other.

The following figure shows how the global input and output variables declared at the input can be linked with the corresponding module variables

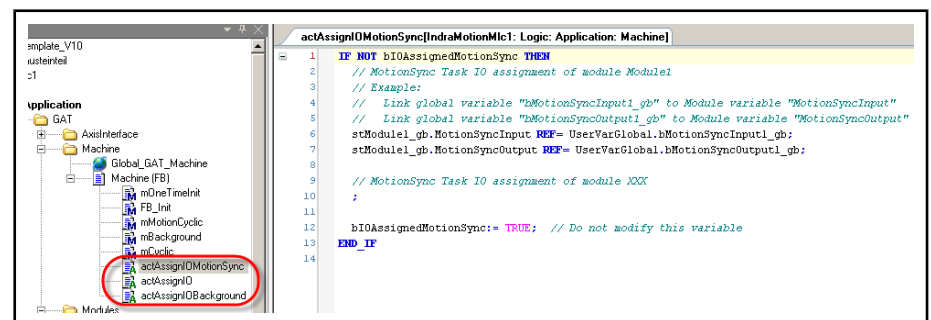


Fig. 8-67: Connection of the global input and output from the global variable list **UserVarGlobal** with the corresponding variables in the data container of the module **Module1**

Because this link is carried out in the **actAssignIOMotionSync** action, these variables are updated using the **MotionSyncTask**.

Based on the previous declarations and assignments, access to the global or address variable within the module is now carried out using the data container **UserData** (see the following figure).

GAT with Central State Machine

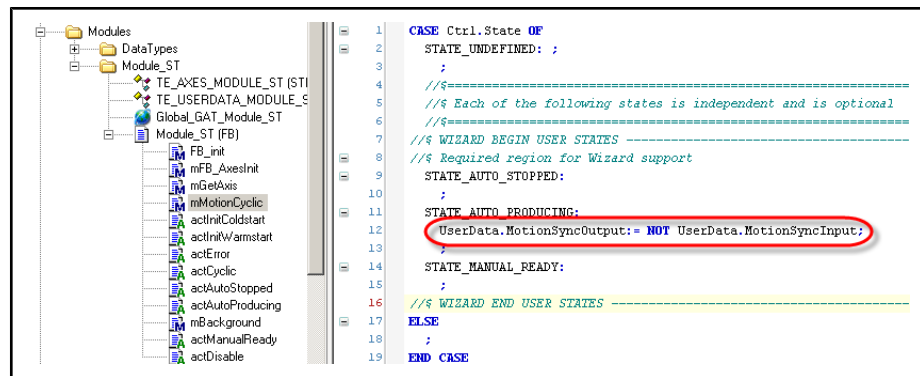


Fig. 8-68: Access to global or address variables within the module using the data container **UserData**, which refers to the module

8.12.13 Processing in MotionTask

For processing in the MotionTask, each module / machine has "mMotionCyclic". All of the module / machine variables (e.g. central axis entry point Axes...) can be accessed from this method.

It is important that the axes controlled here are updated in the MotionTask (see also [chapter 5.5 "Task Management" on page 34](#)).

8.12.14 AxisInterface Extensions

The GAT allows almost any extension of the AxisInterface structures. Additional substructures can be added and the existing substructures extended.

As an example, the following structure elements are programmed as user extensions in the AxisCtrl structure.

- SetupMode: Additional substructure in the AxisCtrl structure
 - Enable: Enable Setup mode
 - JogPlus: Jog +
 - JogMinus: Jog –
 - Vel: Jog / reference travel velocity
 - Accel: Jog/Reference travel deceleration/acceleration
 - Homing: Activates ModeHoming|SetAbsRef|ModesPosAbs depending on axis type/configuration
- SyncMode: Extension of the existing substructure in the AxisCtrl structure
 - AngleOffset: Phase offset start of table -> parameter A-0-2730 is described with this value
 - PhaseShift: Offset relative to the master axis before the gear (PhaseOffset: Offset relative to the master axis after the gear)
 - PhaseShiftVel: Velocity for adjusting the PhaseShift
 - PhaseShiftAcc: Acceleration for adjusting the PhaseShift
 - SyncVelocity: Phase shift velocity -> parameter A-0-2790 is described with this value
 - SyncAcceleration: Phase shift acceleration -> this value is written on parameter A-0-2791

As an example, the following structure elements are programmed as user extensions in the AxisStatus structure:

- SetupMode: Additional substructure in the AxisStatus structure
 - EnableAck: Setup mode is active

GAT with Central State Machine

- Homingack: Homing is finished.
- SyncMode: Extension of the existing substructure in the AxisStatus structure
 - AngleOffsetDone: Transfer of phase offset start of table -> parameter A-0-2730 was described
 - PhasingMasterDone: PhaseShift was completed.
 - SyncVelocityDone: Transfer of synchronization velocity -> parameter A-0-2790 was transferred.
 - SyncAccelerationDone: Transfer of synchronization acceleration -> parameter A-0-2791 was transferred.

The code for these extensions can be found in the action of the TE_AxisInterface() block in the GAT/AxisInterface folder. The related structures can be found in the GAT/AxisInterface/Type/Control and GAT/AxisInterface/Type/Status folders.

Users can add their own extensions (to do so, see [chapter 6.7 "AxisInterface User Extensions" on page 88](#)).

8.12.15 Generating Your Own Program Framework Based on the GAT

An application developed based on the GAT can also be used in turn as a template (program framework) for developing another, more complex application. In this way, users can provide their own program frameworks for various machine types, for example. However, these frameworks differ from the standard GAT due to more advanced developments (e.g. additional operation modes and modules). The GAT wizard can then be used to work with the more advanced program frameworks.

To generate your own template based on an application developed with the GAT, use the following procedure:

1. Export the entire GAT folder; leave all check boxes selected (generate an *.iwx file). In the Export dialog is a "Description" input field. The text entered here can be displayed in the Import function of the GAT wizard.

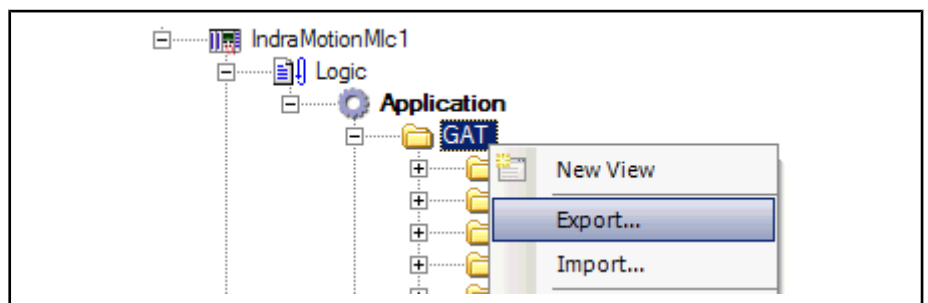


Fig. 8-69: Exporting the GAT folder

2. The *.iwx file generated in step 1 contains the adapted / extended template and can now be used with the GAT wizard as follows for creating new projects:

GAT with Central State Machine

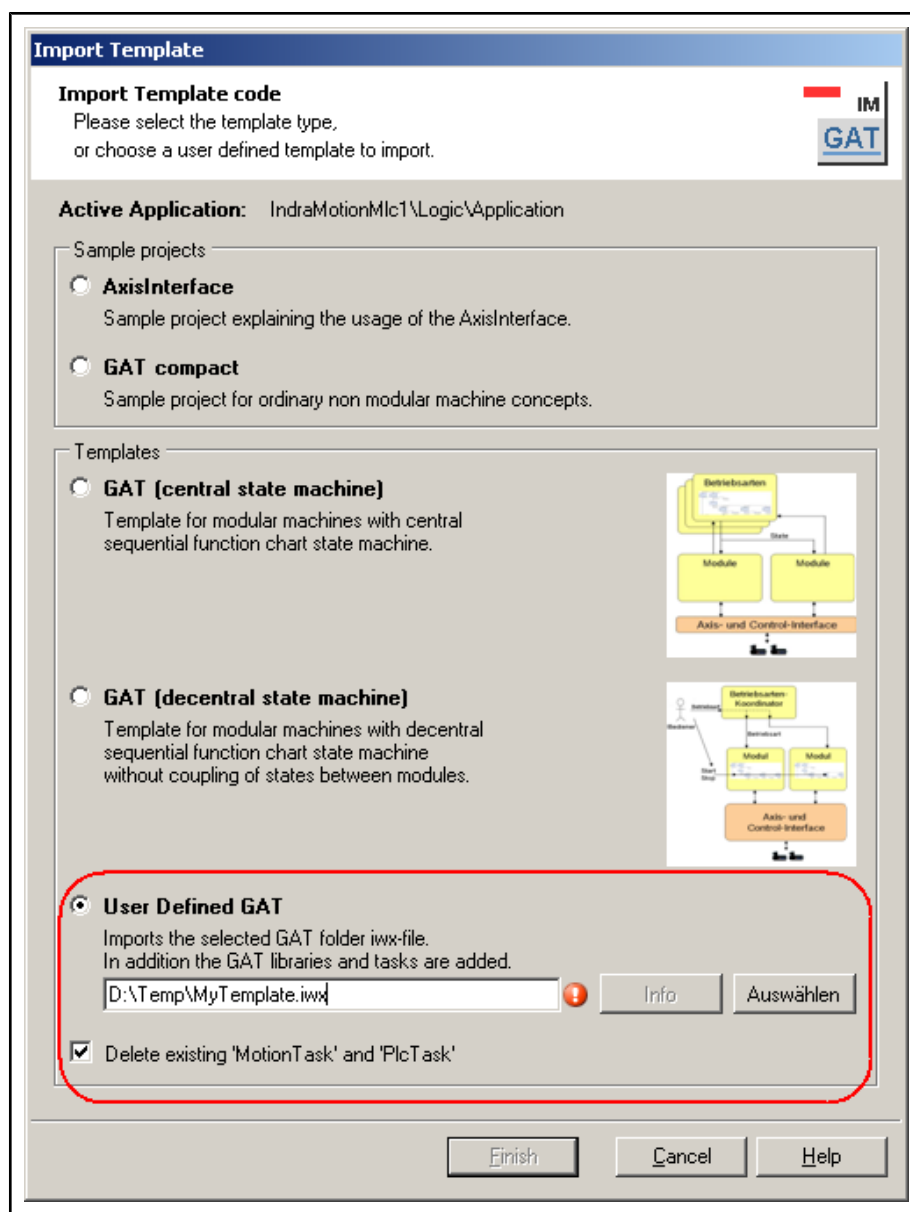


Fig.8-70: Importing a user-defined template

8.12.16 PackML V3.0 Program Framework Based on GAT

A prefabricated program framework is provided with the installation, which maps the modes and states defined in PackML V3.0.

The PackML V3.0 program framework can be loaded using the **User-defined GAT** selection after calling the GAT wizard by selecting the "GAT-PackML-V3.iwx" file to be imported. The "GAT-PackML-V3.iwx" file is delivered with the IndraWorks Installation and is stored in the corresponding GAT wizard directory.

All PackML V3.0 definitions are located below the GAT folder in the PackML_V3 folder.

GAT with Central State Machine

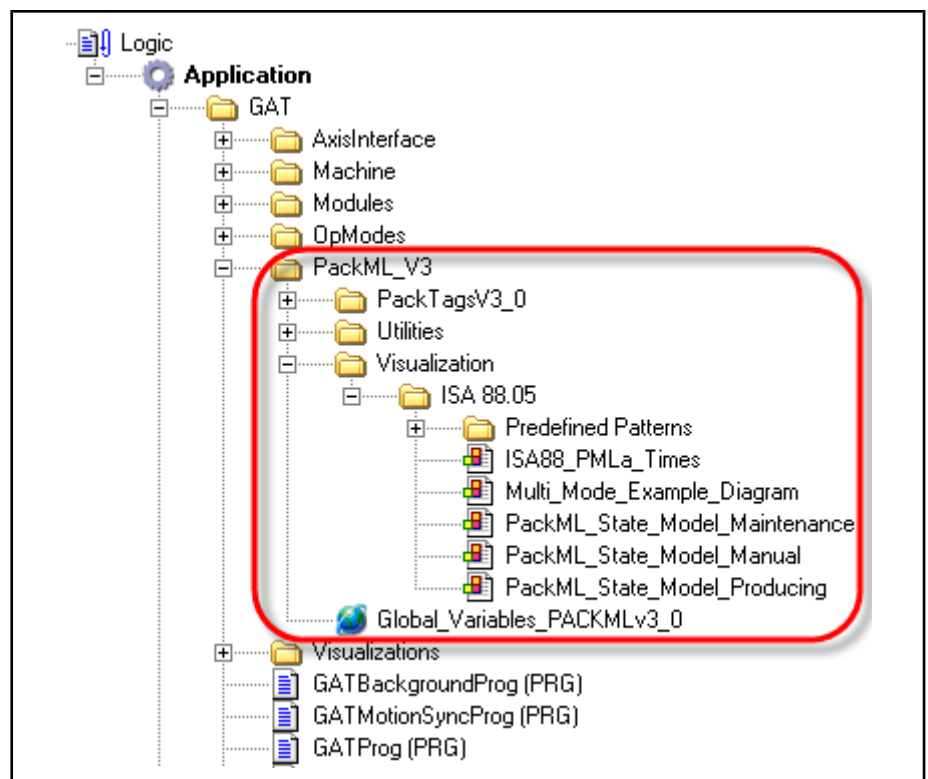


Fig. 8-71: Location of the PackML V3.0 definitions in the pre-defined program framework

In addition to the modes and states defined in PackML V3.0, this program template also contains the visualizations defined in the standard template.

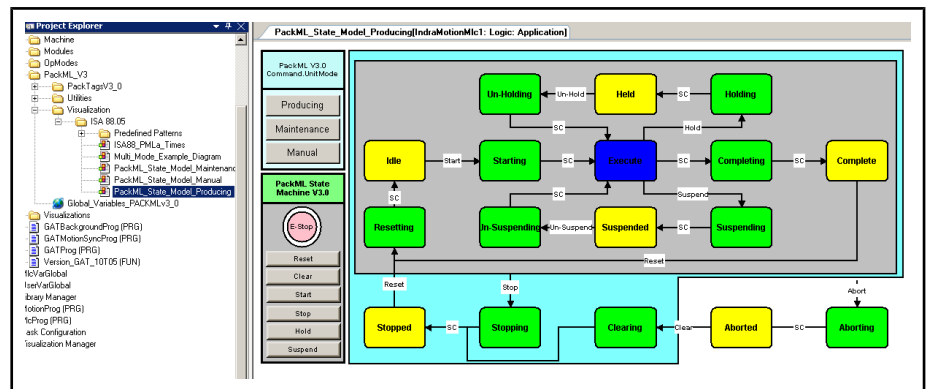


Fig. 8-72: PackML V3.0 visualization for the **Producing** mode within the prefabricated program framework

Because this program template is based on the GAT, all of the processing rules and possibilities mentioned in this document regarding the GAT also apply.



To implement **transient** PackML states (<statename>ing), the relevant code must be placed in the **transient** section of a GAT action that corresponds to the state (see [chapter 8.7.4 "States in the Module"](#) on page 191).

To implement **stationary** PackML states, the relevant code must be placed in the **stationary** section of a GAT action that corresponds to the state (see [chapter 8.7.4 "States in the Module"](#) on page 191).

GAT with Central State Machine

For more information on PackML V3.0, see www.isa.org/link/PWG.

8.12.17 Time Sensitive, Application-specific Error Reaction

In the standard version of the GAT, the operation modes and state machines are processed in the context of the "GATProgTask" (10 ms cyclical task). Therefore, an error is detected and taken care of within this relatively lengthy period of time.

For applications that require faster error reaction, it is recommended that the operation modes and state machines be processed in the context of the "GAT-MotionSyncTask" (cyclical processing synchronous to the Motion cycle). The time period between error detection and error reaction is then one Motion cycle. The following steps are required:

1. Clearing "GATProgTask" from the task configuration
2. Additional program call "GATProg" in the "GATMotionSyncTask"

8.12.18 Switching off GAT Error Reaction for F0 Errors

For errors with a severity degree of "F0", on the control side the control firmware does not automatically react to the error reaction or disengage the axes. However, the GAT detects the error, interrupts the currently selected operation mode and branches into error handling. With default GAT settings, in this error handling the axes are decelerated with a ramp using the "ModeAH" command.

If the GAT does not react to F0 control errors, the following code must be implemented in the "mCyclic" method of the **Machine** FB:

Program:

```
Ctrl.DisableErrorHandling := TRUE; // Turn error reaction off

MB_IMCInterface(); // Refresh control diagnosis number

IF (TE_GetErrorSeverity(IMCStatus.Diag.Number) > 16#F0) THEN
// Turn error reaction on if errors > F0 appear
Ctrl.DisableErrorHandling := FALSE;
END_IF
```

8.12.19 Updating from Version 10/11 to 12

The GAT can be updated from version 10/11 to 12 without changes to the application program because the functions and interfaces are compatible.

Version 12 of the GAT wizard required the current control firmware version 12VRS. For this reason, when the GAT wizard is to be used, the control firmware of controls with firmware 10/11 must be updated.

To update the control firmware and the control configuration the following steps are required:

1. Execute a firmware update to version 12 using the firmware management.
2. Select firmware version 12VRS in the control configuration.
3. In the Library Manager select the libraries ML_TechInterface and ML_TechTemplate, version 12 (using the placeholder TECHINTERFACE or TECHTEMPLATE).

8.13 GAT, Brief Overview

8.13.1 Structure Overview

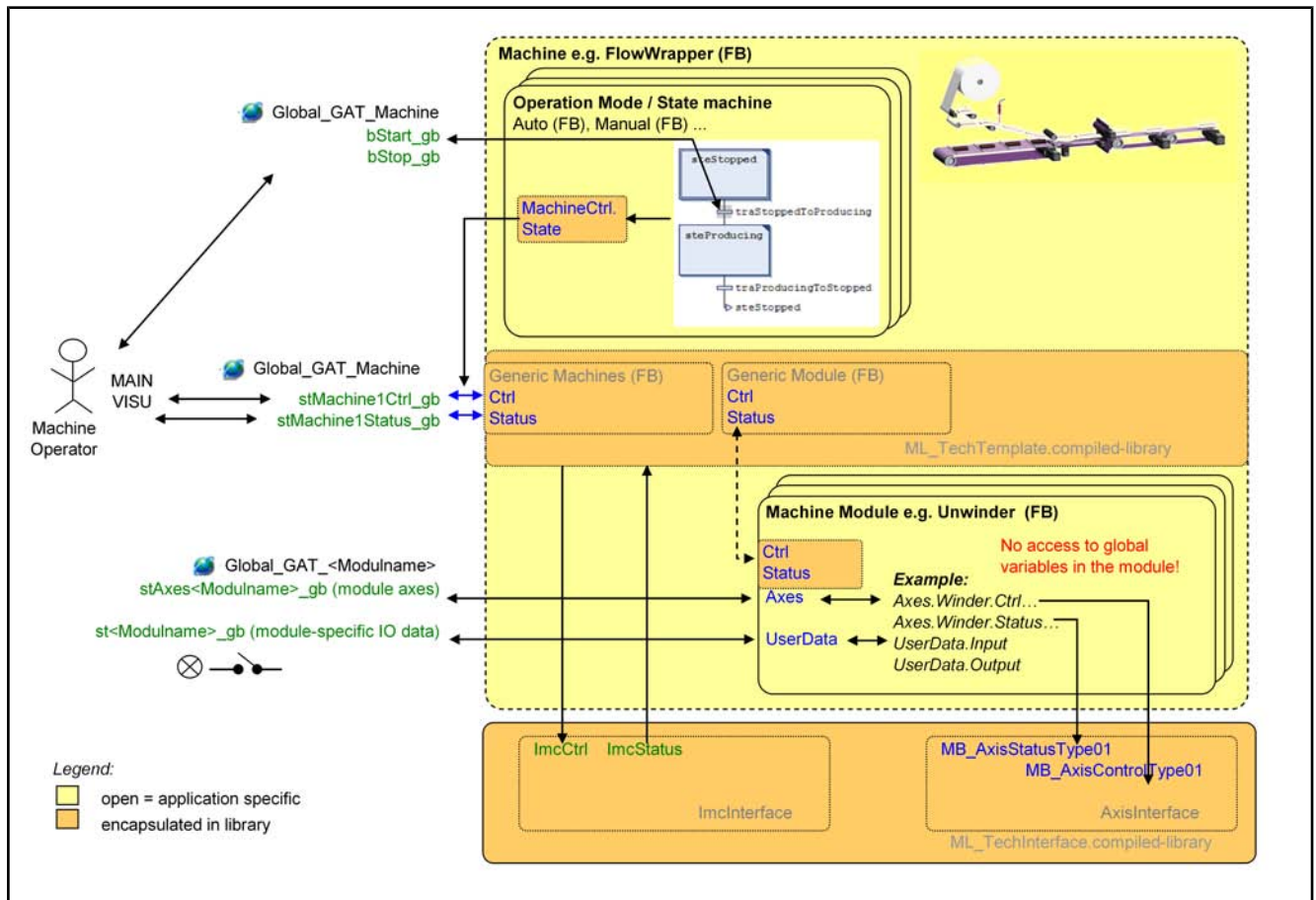


Fig.8-73: GAT, structure overview

8.13.2 Overview of the Machine's Most Important Global Variables

See the global variable list "GAT/Machine/Global_GAT_Machine":

Variable	Description	Access
stMachine1Ctrl_gb	Machine control (e.g. operation mode selection, central "clear error", subsequent operation mode after initialization / error reset, switching off error handling, initialization settings)	RD/WR
stMachine1Status_gb	Machine status (e.g. current operation mode and state, error and diagnostic outputs)	RD
arAppErrMachine1_gb	Array of BOOL with user-specific machine error flags	RD
arMachine1ModuleDiag-No_gb	Array with the diagnostic numbers of all modules	RD
bStart_gb	Start signal for Auto operation mode / Transition from AUTO_STOPPED to AUTO_PRODUCING	RD/WR
bStop_gb	Stop signal for Auto operation mode / Transition from AUTO_PRODUCING to AUTO_STOPPED	RD/WR

RD Read:
WR Write

GAT with Central State Machine

8.13.3 Overview of the Module's Most Important Global Variables

See the global variable list "GAT/Module/Global_GAT_<Modulename>":

Variable	Description	Access
st<Modulename>_gb	Module-specific I/O	RD/WR
arAppErr<Modulename>_gb	Array of BOOL with user-specific module error flags	RD
stAxes<Modulename>_gb	Container with module axes	RD

RD
WR

Read:
Write

8.13.4 Overview of the Most Important Global Variables of the Axes

See the global variable list "GAT/ AxisInterface/ Global_GAT_AxisInterfaces":

Variable	Description	Access
arAxisCtrl_gb	Array with the AxisInterface command data	RD/WR
arAxisStatus_gb	Array with the AxisInterface status data	RD

RD
WR

Read:
Write

8.13.5 Overview of the Machine's Methods and Actions

Action/method	Call	Related task	Description / example
mMotionCyclic	In every state	GATMotionSyncTask	Time sensitive, cross module logic in the context of the GATMotionSyncTask (e.g. cross module, time sensitive monitoring)
mCyclic	In every state	GATProgTask	Cyclical cross module logic in the context of the GATProgTask
mBackground	In every state	GATBackgroundProgTask	Cross module background activities that are not time sensitive in the context of the GATBackgroundProgTask (e.g. HMI data preparation, statistics)
mOneTimeInit	For one cycle when the GAT is initialized. Execution before cold start	GATProgTask	Activating / deactivating modules, initializing settings for "SkipCold-Start", assigning module axes to the global IndraWorks axes
mExitOpMode	For one cycle when exiting an operation mode	GATProgTask	Method is optional: activities when exiting an operation mode (e.g. re-setting variables)
mEntryOpMode	For one cycle when activating an operation mode	GATProgTask	Method is optional: activities when activating an operation mode (e.g. initializing variables)
mOpModeChangeAllowed	Before switching operation modes	GATProgTask	Method is optional: the method is called by GAT before switching operation modes. The operation mode switch is blocked as long as the return value = FALSE.

GAT with Central State Machine

Action/method	Call	Related task	Description / example
mlsCriticalError	In case of error	GATProgTask	Called if an error occurs and allows the rejection of an error for the machine
actAssignIOMotion-Sync	Once after starting/resetting a PLC program	GATMotionSyncTask	I/O assignment at the "GATMotion-SyncTask"
actAssignIO	Once after starting/resetting a PLC program	GATProgTask	I/O assignment at the "GATProgTask"
actAssignIOBack-ground	Once after starting/resetting a PLC program	GATBackgroundProgTask	I/O assignment at the "GATBackgroundProgTask"
FB_init	Once before starting/resetting a PLC program	No specific task assignment	Internal system use, may not be modified

8.13.6 Overview of the Machine's Methods and Actions

Action/method	Call	Related task	Description / example
actInitColdstart	Only in the INIT_COLDSTART state	GATProgTask	Initializing / writing parameters in parameterization mode (P2)
actInitWarmstart	Only in the INIT_WARM-START state	GATProgTask	Initializing / writing parameters, power activation for axes
actAutoStopped	Only in the AUTO_STOPPED state	GATProgTask	Stop axis movement
actAutoProducing	Only in the AUTO_PRODUCING state	GATProgTask	Start axis movement
actManualReady	Only in the MANUAL_READY state	GATProgTask	Monitoring while jogging the axes
actError	Only in the ERROR_F0.. state ERROR_F9	GATProgTask	Additional user-specific error reaction (e.g. synchronized deceleration of the axes)
mMotionCyclic	In every state	GATMotionSyncTask	Cyclical call in the context of the GATMotionSyncTask. Call of time sensitive process controller FBs (e.g. RegisterController)
mCyclic	In every state	GATProgTask	Cyclical call in the context of the GATProgTask for module-specific cyclical logic
mBackground	In every state	GATBackgroundProgTask	Cyclical call in the context of the GATBackgroundProgTask for background activities that are not time sensitive (e.g. HMI data preparation, statistics)
mlsMyAxis	No automatic call	No specific task assignment	Checks if an axis belongs to the module
mlsCriticalError	In case of error	GATProgTask	Called if an error occurs and allows the rejection of an error for this module

GAT with Central State Machine

Action/method	Call	Related task	Description / example
mForceOpMode	No automatic call	No specific task assignment	• Puts the module in another operation mode than that specified by the machine. • Resets the operation mode of the module to the operation mode of the machine
mExitOpMode	For one cycle when exiting an operation mode	GATProgTask	Method is optional: activities when exiting an operation mode (e.g. re-setting variables)
mEntryOpMode	For one cycle when activating an operation mode	GATProgTask	Method is optional: activities when activating an operation mode (e.g. initializing variables)
actDisable	In every state while deactivating a module until the deactivation is complete	GATProgTask	Resetting variables or module outputs during module deactivation
mGetAxis	No automatic call	No specific task assignment	Internal system use, may not be modified. Delivers the pointer to a module axis
mFB_AxesInit	During axis initialization	GATProgTask	Internal system use, may not be modified
FB_init	Once before starting/resetting a PLC program	No specific task assignment	Internal system use, may not be modified

8.13.7 Access to Axes within the Module

Always use **Axes.<axisname>**

Example:

Program:

```
// Command axis in velocity control mode
Axes.MyAxis1.Ctrl.Admin._OpMode.en := ModeVel;

// If axis has reached the velocity
IF (Axes.MyAxis1.Status.PtrAxisData^.Axis_InVelocity = TRUE) THEN;
```

8.13.8 Access to I/O Data within the Module

Always use **UserData. !! No direct access to global variables !!**

Example:

Program:

```
IF (UserData.bInput1=TRUE) THEN // If "bInput1" is set...
UserData.bOutput1:=TRUE; // Set output "bOutput1"
END_IF
```

9 GAT with Decentral State Machine

9.1 General Information

The **Generic Application Template (GAT)** with decentral operation mode SFC is a simpler, more scalable, ready to use IndraWorks program framework based on IEC61131-3 that already contains all of the relevant components for controlling a machine and its drives.

The GAT wizard integrated into IndraWorks accelerates training because the user is immediately taken to all of relevant code positions.

The GAT already contains all of the important basic functions of a machine from operation mode management, error handling and a diagnostic system to product and machine data handling and simplified Motion command using the AxisInterface. The GAT can be used intuitively even in basic version due to prepared visualizations.

Defined interfaces and wizard support make adjustments and additions easy.



The GAT with decentral operation mode SFC requires an IndraMotion MLC control.

9.2 Overview

9.2.1 Basic Idea

Using the GAT is based on the idea that an application runs on a machine that coordinates one or more functional "units" in a various operation modes. In the simplest case, the selected operation mode is forwarded to the units. However, it is also possible to specify different operation modes for each unit.

The GAT consists of the following elements:

GAT with Decentral State Machine

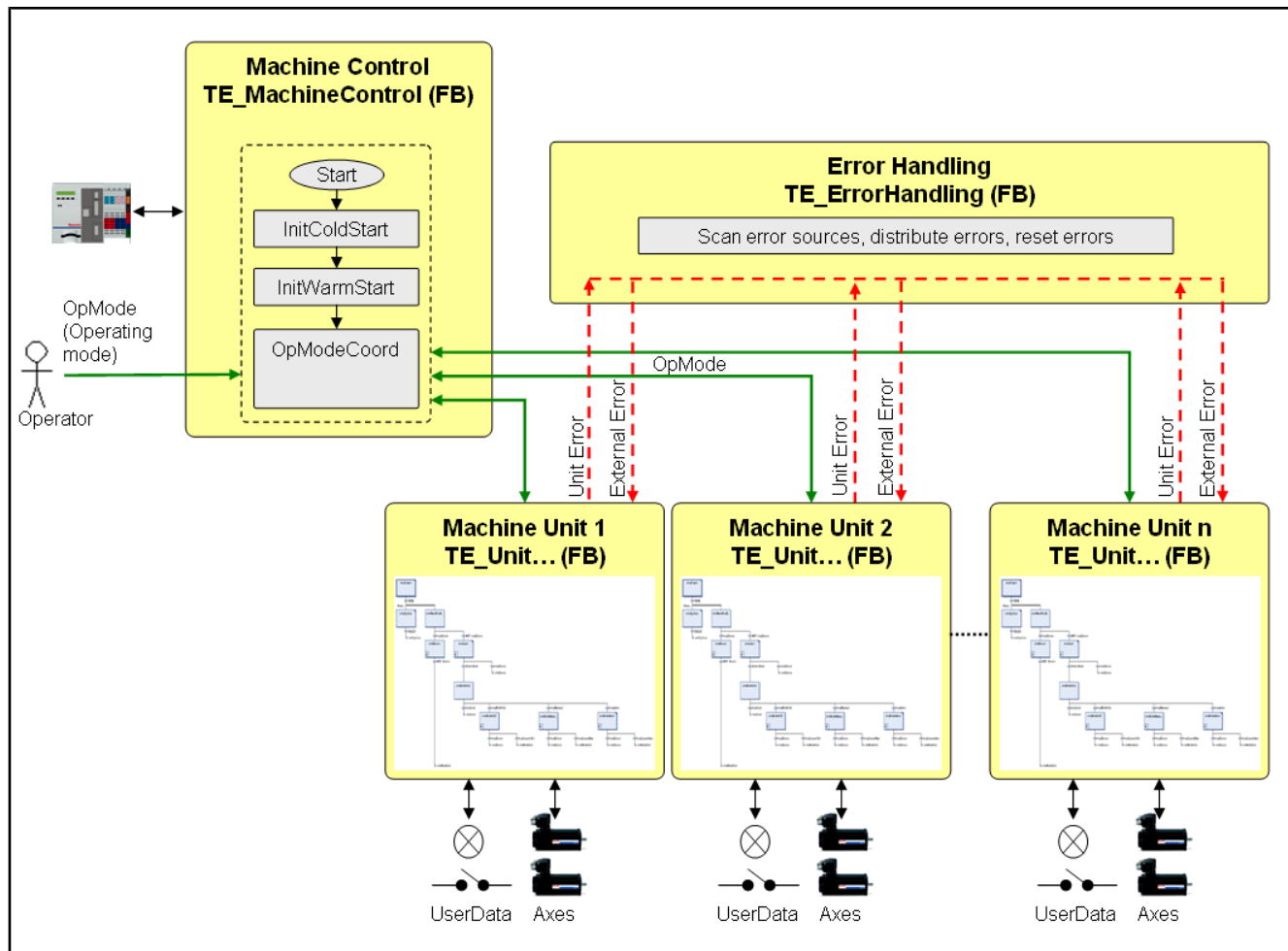


Fig.9-1: Software architecture of the GAT with decentral state machine

Machine Handler

The Machine Handler (FB **TE_MachineControl**) forwards the operation mode selected by the operator to the respective unit. During system initialization, it commands the phase switching of the control and in so doing, brings all units through the respective initialization steps. After the initialization is complete, in the simplest case the operation mode commanded by the operator is forwarded to all units.

The Machine Handler is divided into the following parts:

- Sequencing for the initialization (action **actOPModeSFC**)
- Data Handler for visualization and field bus connection with machine and product data handling (method **mDataHandler**)
- Primary operation mode determination (method **mOpModeSet**)
- Operation mode distribution to the respective units (method **mOpMode-Distribution**)
- Permanent cyclical processing (method **mCyclic**)

Error Handler

The Error Handler (FB **TE_ErrorHandling**) is the central position where all errors are collected and forwarded. In addition, the error handling performs the central error clearing.

The Error Handler is divided into the following parts:

- In the **error detection** section all detected sources of error (e.g. control, unit, field bus or inline errors) are scanned (method **mScanError**).

GAT with Decentral State Machine

- In the **error distribution** section, the error information is distributed to all units. It is also possible to forward it selectively to specified units (method **mDistributeError**).
- In the **error acknowledgement** section, the order to clear errors is forwarded to all subordinate sources of error (method **mClearError**).

Units

A machine unit, called unit in the following, is an encapsulated unit for modularizing the software (FB **TE_Unit...**). All units have uniform interfaces in order to ensure user-friendly replacement and maintenance.

Each unit queries the error information of the associated components (e.g. axes) and forward this information to the Error Handler. In addition, the **error reaction** is executed in the unit.

9.2.2 Requirements for Use

Already presented to some extent in the previous section, the prerequisites for using the GAT are summarized again here.

Before a Motion application is developed with the GAT, the user must clarify the following questions with respect to the application:

- Which operation modes are to be realized?
- Which functional units make up the machine? Which axes and I/Os are associated with each respective functional unit?

Using the information, the wizard can generate the code framework for the actual Motion application in a very short time.

The following section describes the parts of the GAT, how these parts work together and where the user has to make additions in order to realize the actual application.

9.3 First Steps - GAT with Decentral State Machine

9.3.1 Preparation

The GAT is integrated into the IndraWorks Suite installation package and is automatically installed.

The GAT with decentral operation mode SFC can be used if a project was created with an IndraMotion MLC control.

The GAT wizard is opened using the GAT button  in the IndraWorks Toolbar.



The GAT wizard always refers to the active application in the IndraWorks project.

9.3.2 Creating a Simple Motion Application

General Information

This section describes how to create a GAT application with a decentral state machine using the GAT wizard.

Topic

A machine is to consist of two functional units.

In the unit named "UnitMaster" the unit axis "Master_Axis" is to move continuously at 100 rev./min. in Automatic operation mode.

GAT with Decentral State Machine

In the unit named "Unit1" the unit axis "Slave_Axis" is to move phase synchronously with the continually moving axis from "UnitMaster" in the Automatic operation mode.

The following figure provides a schematic representation of the tasks.

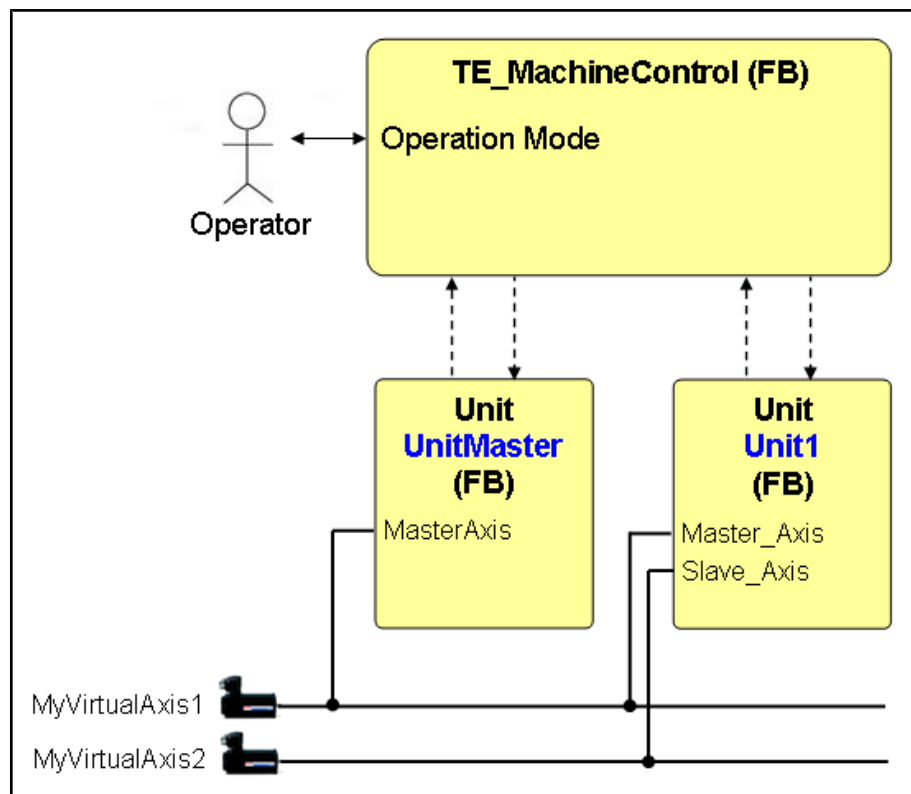


Fig. 9-2: Schematic assignment of tasks

Operating Sequence

As described earlier, the GAT wizard starts. In the following dialog, select "GAT (decentral state machine)" and then click the **Finish** button.

GAT with Decentral State Machine

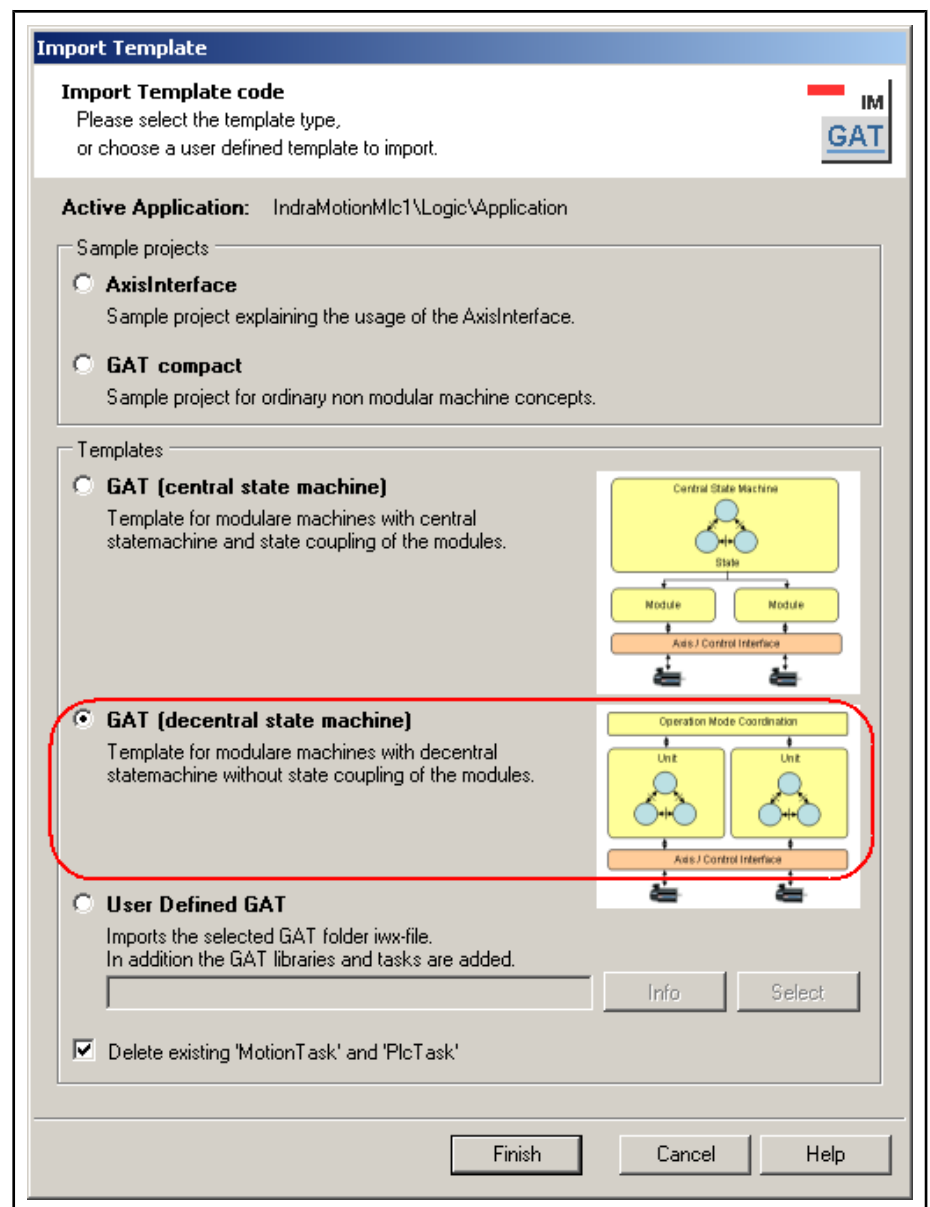


Fig.9-3: Importing GAT with decentral state machine

By selecting "GAT with decentral state machine", a special project is imported, in which two global virtual axes (MyVirtualAxis1, MyVirtualAxis2) have already been configured in addition to a PLC program template.

Nothing else is required in order to realize the tasks with respect to defining the operation modes because the Automatic operation mode is contained in the preconfiguration.

A coupled unit is already created in the GAT (Unit1). Another unit is required to complete the topic. To do this, highlight the "Units" node in the GAT wizard directory tree and add a coupled unit with code for axis jogging and homing in manual mode.

GAT with Decentral State Machine

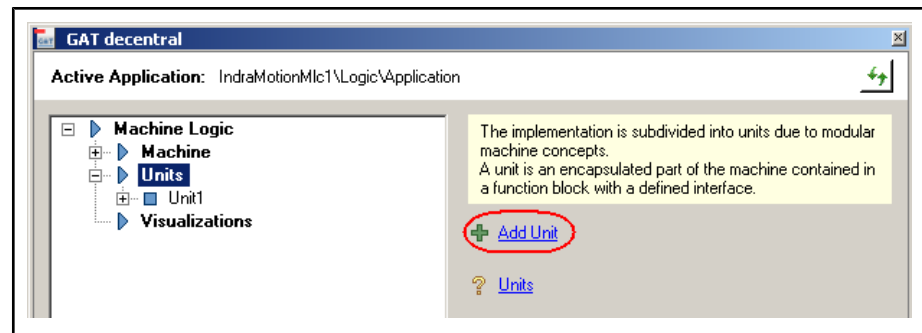


Fig.9-4: Adding a unit

In a subsequent dialog, the name of the new unit is specified as "Unit_Master".

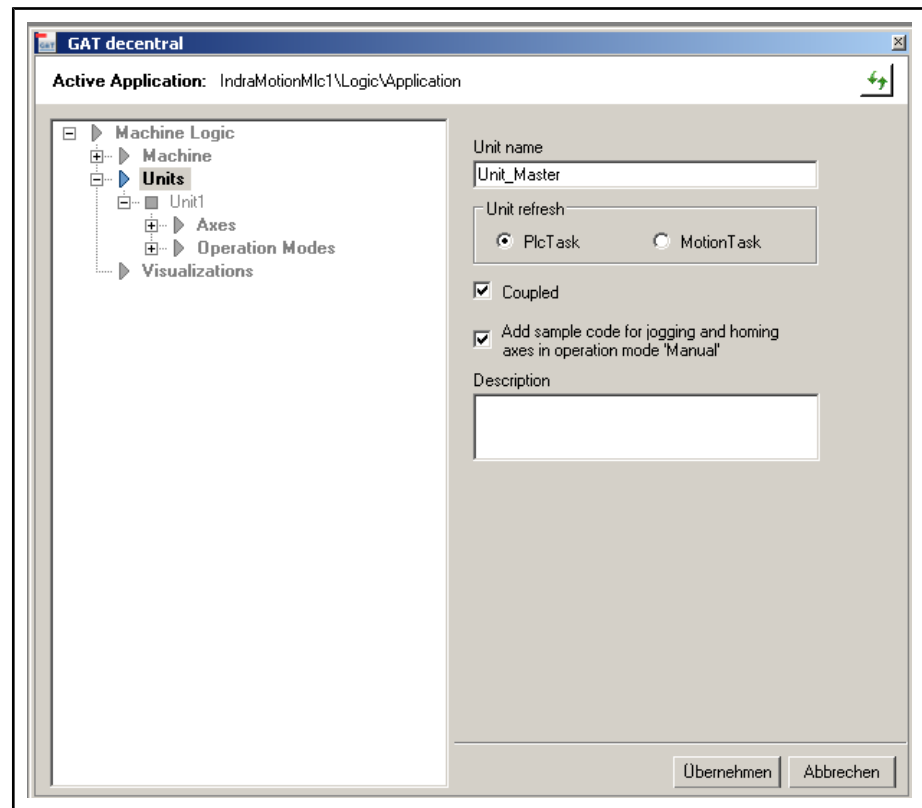


Fig.9-5: Specifying the unit name

Now the unit axis is to be created and implemented in the "Unit_Master" unit. This axis is to move continuously at 100 rev./min. in Automatic mode.

To do this, the wizard is used to create a new unit axis for the "Unit_Master" unit.

GAT with Decentral State Machine

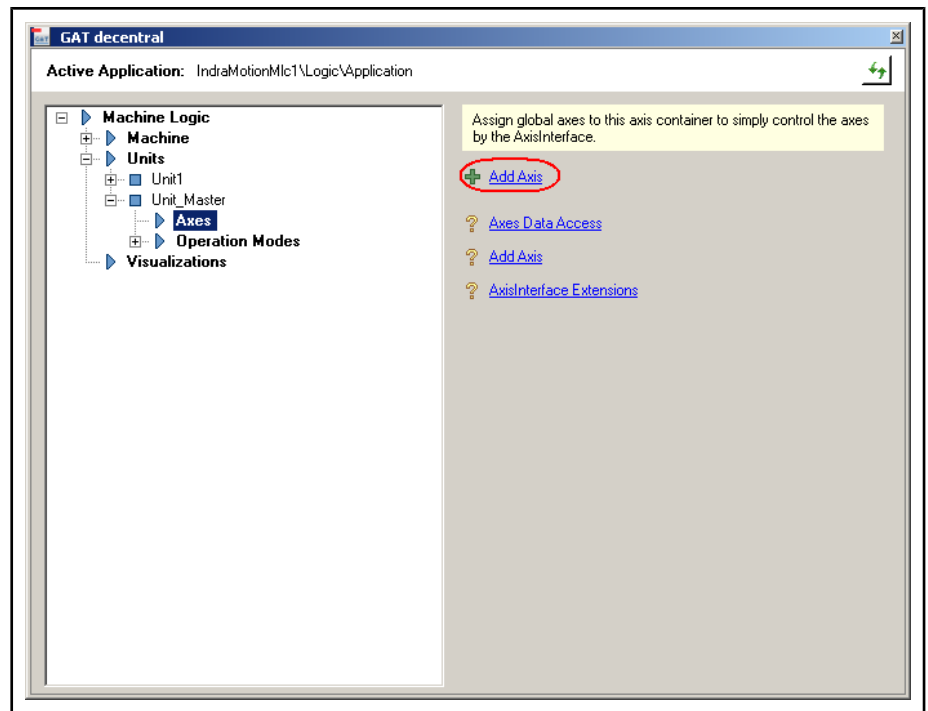


Fig.9-6: Adding an unit axis with the GAT wizard

The axis in the unit shall be called "MasterAxis". This unit axis is assigned to the globally configured axis "MyVirtualAxis1".

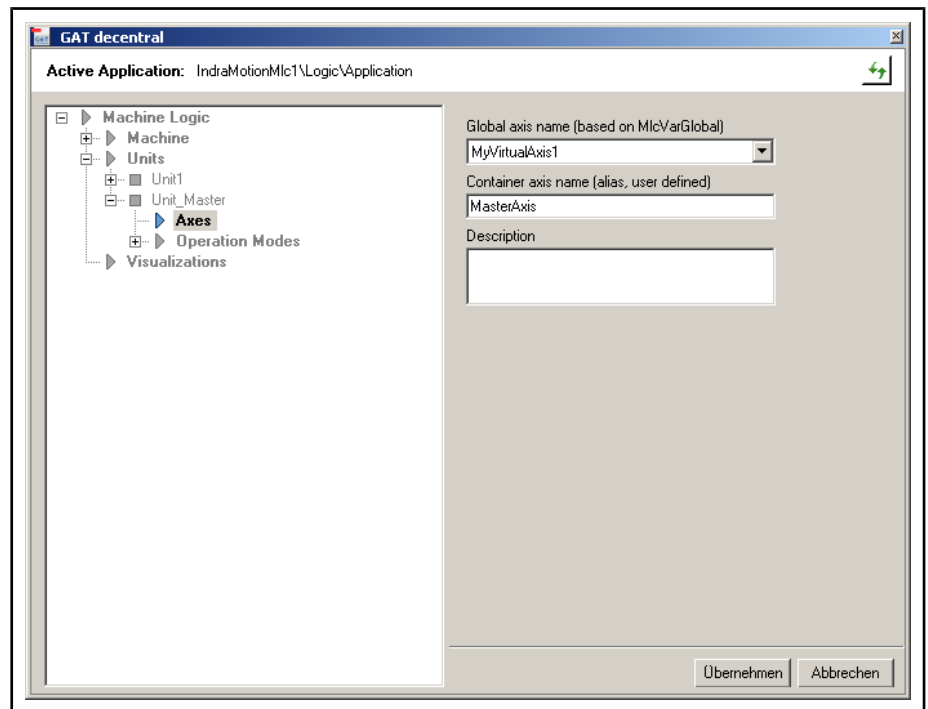


Fig.9-7: Assigning the unit axis to the globally configured axis

In the next step, the axis must be implemented in the unit such that it moves in the Automatic operation mode at a constant velocity of 100 rev./min. (unit is specified by scaling the axis).

The corresponding code position for the Automatic operation mode can be found easily using the GAT wizard.

GAT with Decentral State Machine

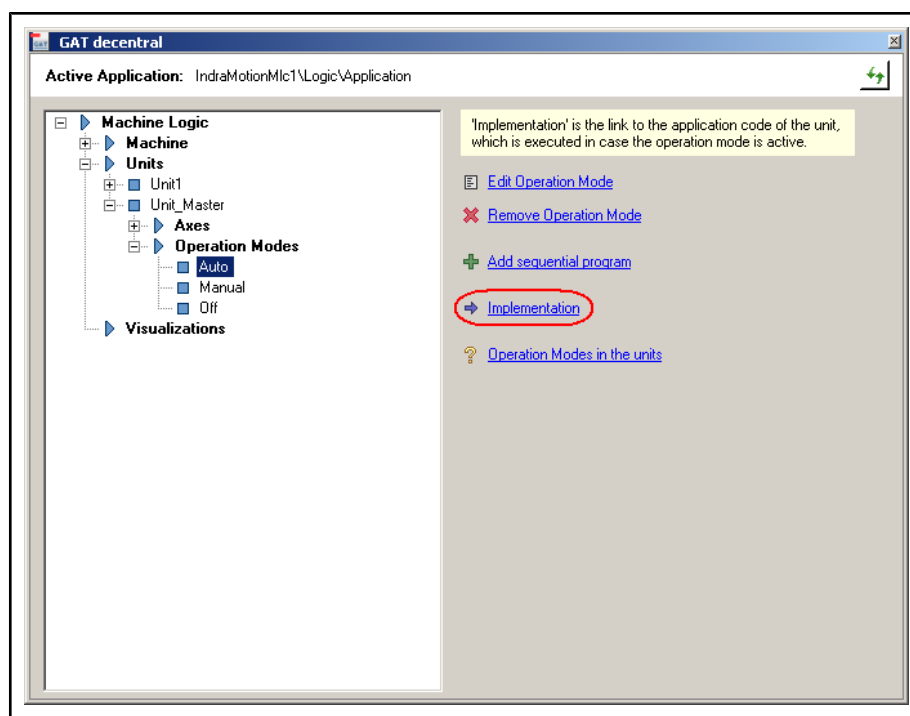


Fig.9-8: Finding the code position for the Automatic operation mode using the GAT wizard

The MasterAxis is to move with constant velocity in the Automatic operation mode. For this reason, the "MasterAxis" must be commanded accordingly. The axis data container "Axes" is used for commanding axes. By adding the following lines of code into the Automatic operation mode, the "MasterAxis" will move with a constant velocity of 100 rev./min. with velocity control.

Program:

```
//-----
// insert your code here
//-----
Axes.MasterAxis.Ctrl.Admin._OpMode.en := ModeVel;
Axes.MasterAxis.Ctrl.VelMode.Velocity := 100;
```

Axis data access using Axes

All of the axes in a unit can be addressed using the "Axes" data container in the unit. All axis data is also accessible via this data container.

Referencing an axis within a unit is always done using "Axes". In every unit, "Axes" is the name of the axis container. Enter a "." (period) after "Axes" to display all of the axes contained in the axis container (see the following figure). All I/O data for the unit can be accessed in the same way using "UserData".

GAT with Decentral State Machine

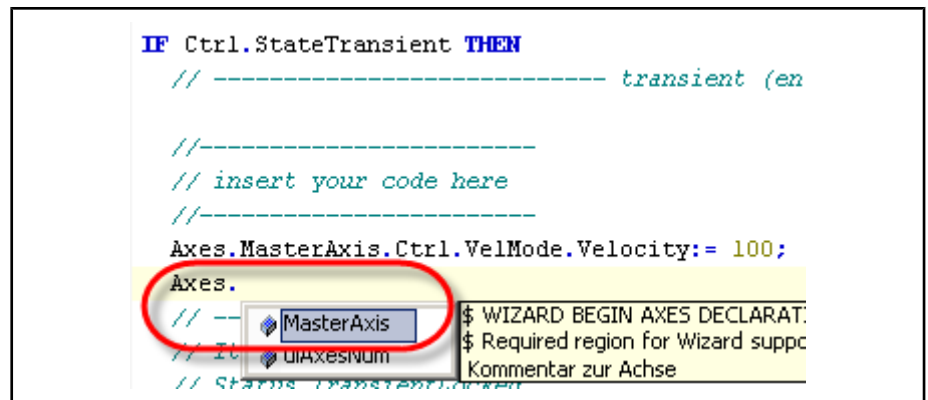


Fig.9-9: Commanding axes within a unit and Intellisense

Further, by entering the period operator after the axis name, all elements that contain data regarding this axis are displayed for selection (see the following figure). To move to the next level, enter a period after the name of the previous level.

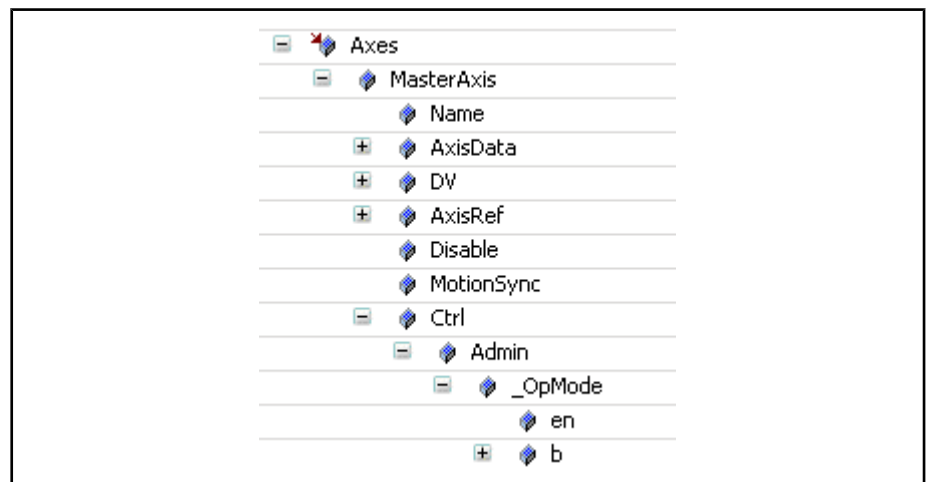


Fig.9-10: Available structure elements for a unit axis

Use the same procedure to create a unit axis "Master_Axis" and a unit axis "Slave_Axis" for the other unit "Unit1".

In the original "Unit1" unit, there are already two unit axes, "MyAxis1" and "MyAxis2". The globally configured axes with the same names are assigned to them, respectively. Using "Edit axes" (see the following figures), the names of the unit axes must be modified such that the globally configured axis "MyVirtualAxis1" is assigned to the unit axis "Master_Axis" and the global axis "MyVirtualAxis2" is assigned to the unit axis "Slave_Axis".

GAT with Decentral State Machine

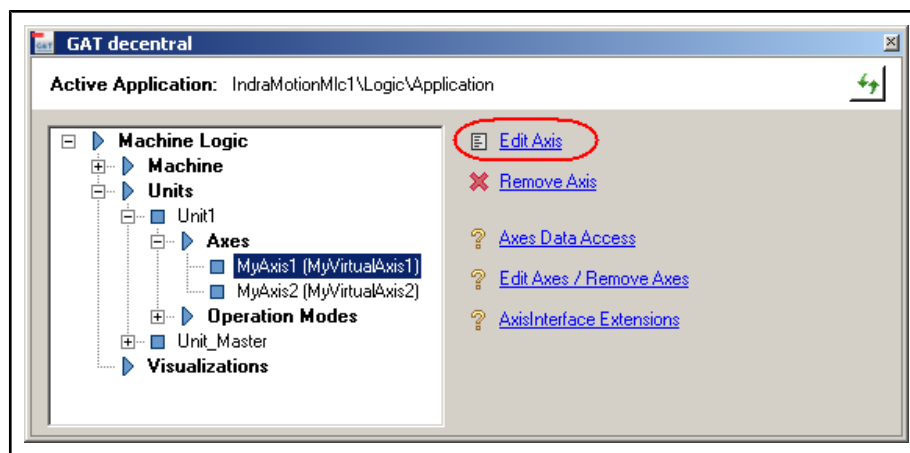


Fig.9-11: Editing configured axes

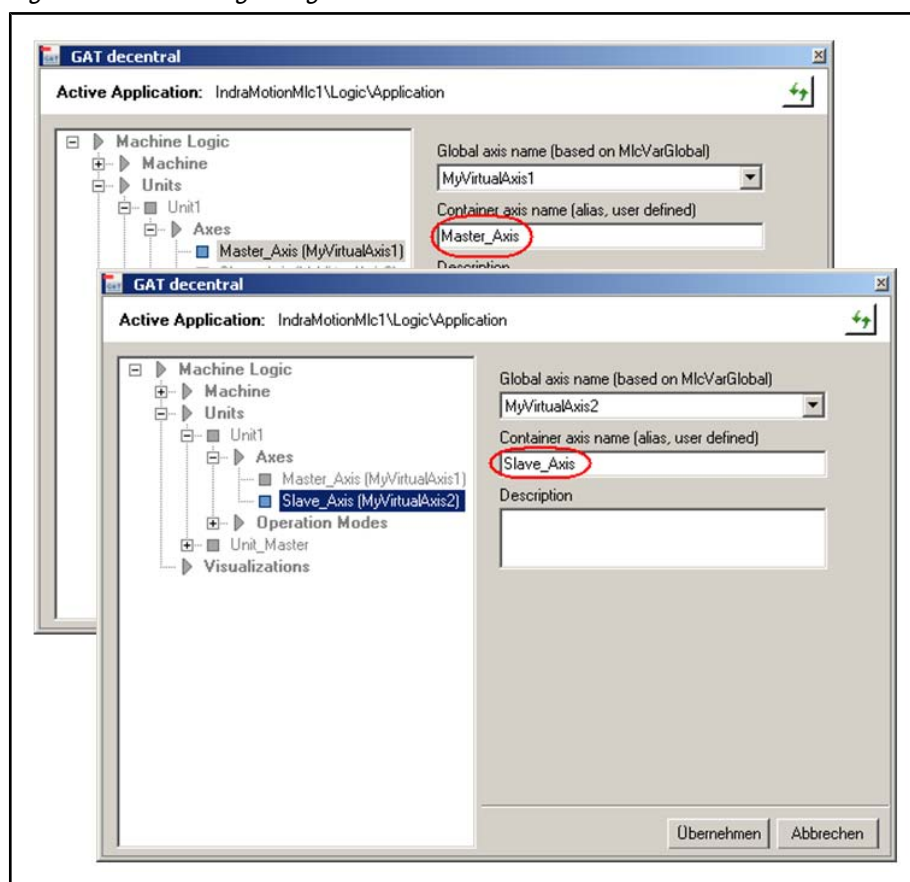


Fig.9-12: Editing configured axes

In the final step, the axis "Slave_Axis" in the Automatic operation must be synchronized with the unit "Unit1" on the axis "MasterAxis". To do this, the GAT wizard is used to open the Automatic operation mode of "Unit1" and to enter the following program code there.

GAT with Decentral State Machine

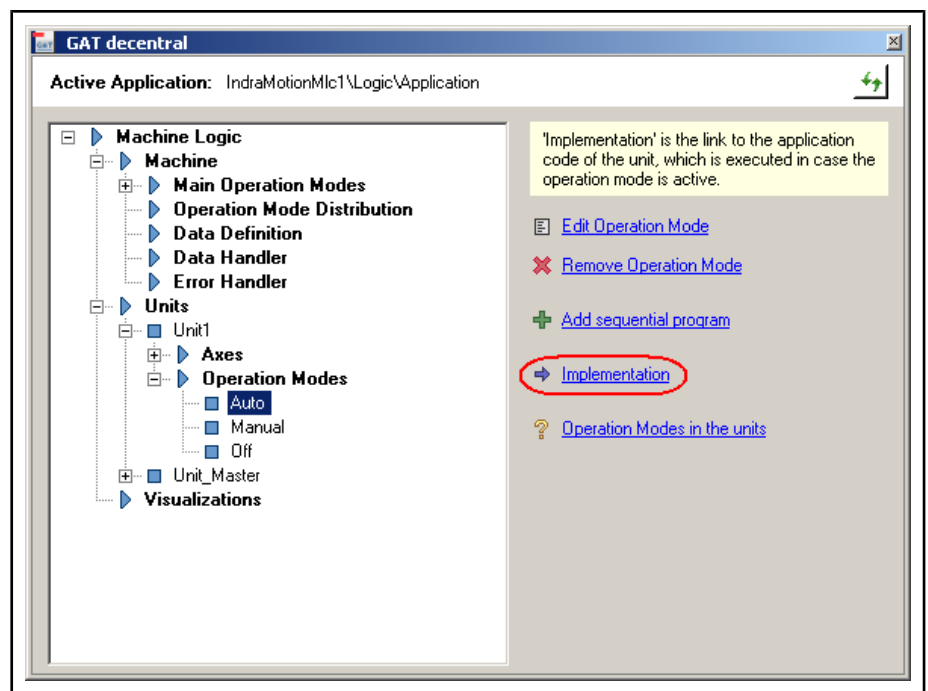


Fig.9-13: Opening the Automatic operation mode of Unit1

Program:

```
//-----  
// insert your code here  
//-----  
Axes.Slave_Axis.Ctrl.SyncMode.Master := Axes.Master_Axis.AxisRef;  
Axes.Slave_Axis.Ctrl.Admin._OpMode.en := ModeSyncPhase;
```

The implementation is complete.

After the configuration (1) and the PLC project (2) have been downloaded, the PLC can be started (3) (see the following figure).



Fig.9-14: Downloading (configuration/PLC project) and starting the PLC

After the PLC project has been started, the GAT can be operated using the predefined internal visualization "System_Overview_All_Axis". First, click the **Remote On** button of the visualization to take control of the GAT.

Then the Automatic operation can be activated with the **Auto** button and the axes move accordingly.

If "Manual" operation mode is selected, the axes are halted and the Setup mode for the respective axis can be activated with the **Enable** button. The axes can be jogged or homed in activated Setup mode using the respective buttons in the visualization.

9.4 Functional Principle

In order to generate the Motion application simply and effectively, the program code for the GAT is divided into two sections.

GAT with Decentral State Machine

Program code that can be generalized for various applications is located in the basic types and basic blocks of the GAT library (ML_TechTemplate.compiled-library).

With wizard support, the user places application-specific program code at the position intended for it in the application program.

This functional principle yields the following advantages:

- Creation of compact and clearly arranged user code:
 - Through the use of simple interfaces and clear structures based on an optimized program template
- Increased robustness:
 - Through the use of tested and optimized code sections
- Improved maintainability
 - Due to easy library exchange when eliminating errors

9.5 Initialization

The initialization is completed at machine start up and consists of 2 steps:

- In the first step, the cold start (parameterization mode/P2) is activated and transferred to all units. The application-specific program code for initialization is located in the action "**actInit**" of the respective unit and is processed in this state. Switching to the next step occurs when the parameterization mode/P2 of the control is active and all units enable the switch (see note below).
- In the second step, the warm start (operating mode/BB) is activated and transferred to all units. The initialization is completed when the operating mode/BB of the control is active and all units enable the switch (see note below). Then the operation mode selected by the machine operator becomes active. The initialization can be executed again at any time by selecting the initialization operation modes.



Switching to the next initialization step can be blocked or enabled by using the functionality "**LockOpModeAcknowledge**" within the unit.

By setting "**LockOpModeAcknowledge := TRUE**", the switch to the next step can be blocked for several cycles, e.g. to read or write drive parameters.

SkipColdstartAtStartup

After PLC project download or a PLC reset, the "cold start" of the Init operation mode is activated by default. Particularly for initial machine commissioning, this behavior can cause disturbances because switching to "cold start" = parameterization mode is time-consuming. For this reason, the "**SkipColdstartAtStartup = TRUE**" setting can be used to prevent the switch to "cold start" in these cases. With this setting, after PLC download or reset, "cold start" is skipped and only "warm start" is processed.

Modifications to the "**SkipColdstartAtStartup**" setting can be made in the assignment of the "**mOneTimeInit**" method at the TE_MachineControl FB (see the following figure). After the modified PLC project is downloaded, the new settings become effective.

GAT with Decentral State Machine

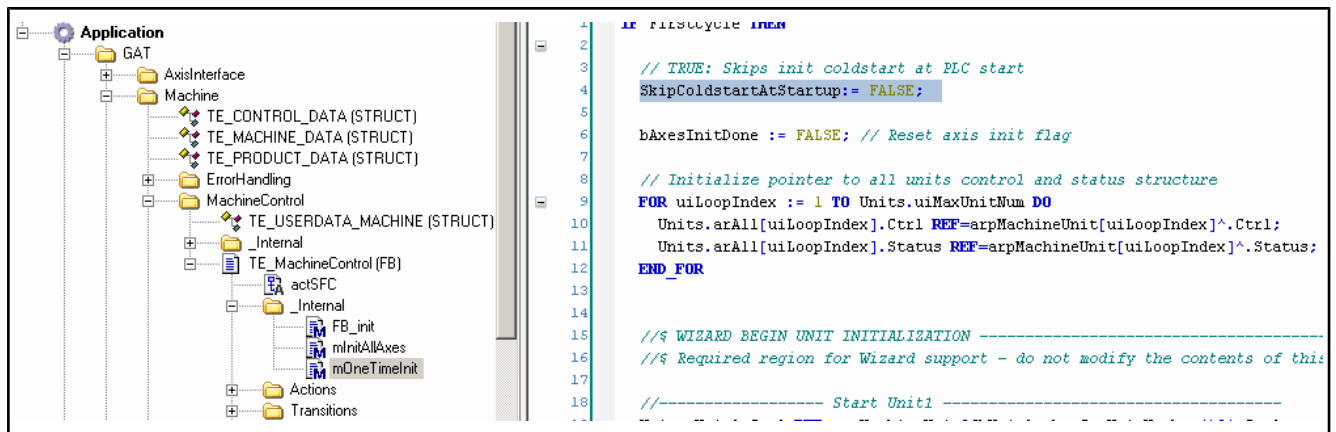


Fig. 9-15: Configuration of the initialization using the "SkipColdstartAtStartup" structure element

9.6 Operation Modes

General Information

At the initial import, the GAT already contains the predefined primary operation modes "MODEOFF", "AUTO" and "MANUAL". The initialization was already discussed in the previous chapter.

In principle, the operation modes are differentiated as follows:

- **Primary operation modes** are forwarded to all **coupled** units, which contain them as branches in the operation mode SFC.
- **Unit-specific operation modes** respectively specific operation modes for **uncoupled** units, which contain them as branches in the respective operation mode SFC.

Target operation mode after initialization/operation mode switch

After initialization is completed, the primary operation mode commanded by the operator is forwarded to all coupled units and becomes effective.

For all uncoupled units, a unit-specific operation mode is specified for each unit using the program code in the method "mOpmodeDistribution" of the FB "TE_MachineControl".

Operation mode acknowledgement

The selected primary operation mode is acknowledged at the machine's status output (variable "stMachine_Status_gb") only after all units (coupled and uncoupled) acknowledge the commanded operation mode.

With the functionality "LockOpModeAcknowledge", each individual unit has the option to delay the acknowledgement of the operation mode.

9.7 Units

9.7.1 General Information

Operation mode SFC

Each unit contains an operation mode SFC (actOPModeSFC), which is called cyclically. Depending on the commanded operation mode, the corresponding step in the SFC becomes active and the related PLC code is processed (e.g. for Automatic mode).

Local error handling

In case of error, the operation mode is interrupted and the SFC branches into error handling. Each unit is responsible for the error reaction of the components assigned to it (e.g. corresponding axes). When an error occurs within a unit, the error is forwarded to the superordinate Error Handler so that the information can be distributed. As an option, forwarding of the local error to the superordinate Error Handler in a unit can be suppressed (see "Local error handling in a unit" on page 254).

GAT with Decentral State Machine

Axis handling If axes are present in the unit, they are initialized at startup and the AxisInterface is called during runtime. Depending on the system, the AxisInterface for the respective unit axes is automatically called in the same task as the corresponding unit.

In the application program, it is easy to use the wizard to add, delete and modify units.

In principle, the units are differentiated as follows:

Coupled units The primary operation mode commanded by the operator is automatically transferred to all coupled units. For this reason, in coupled units all primary operation modes are contained as branches in the operation mode SFC.

The operation mode distribution to the units is carried out in the method **mOp-modeDistribution** of TE_MachineControl. Here the instruction "mSetUnitsOpModes(OpMode := OpMode, CoupledUnitsOnly := TRUE);" is used to forward the primary operation mode to all coupled units.

With coupled units it is possible to disable the operation mode coupling during runtime and to establish it again. For example, for maintenance purposes this function can be used to remove a unit from the operation mode link while all other units continue production in Automatic mode.

The instruction "Units.<UnitName>.Ctrl.OpModeCoupled := FALSE;" within TE_MachineControl is used to cancel the operation mode coupling. In this case, the primary operation mode selected on the machine is no longer forwarded to this unit. In this way, the user program can be used in the method **mOpmode-Distribution** to specify any desired primary operation mode of the unit.

The instruction "Units.<UnitName>.Ctrl.OpModeCoupled := TRUE;" is used to establish the operation mode coupling again and the unit now receives the commanded primary operation mode automatically again.

Uncoupled units After initialization is completed, the primary operation mode commanded by the operator is not automatically forwarded to the uncoupled units. The operation mode is specified by the application program using the method "mOpmode-Distribution" in the FB "TE_MachineControl".

After the initial GAT import, one coupled unit **"Unit1"** is already available (see the following figure). The GAT wizard can be used to modify the unit name.

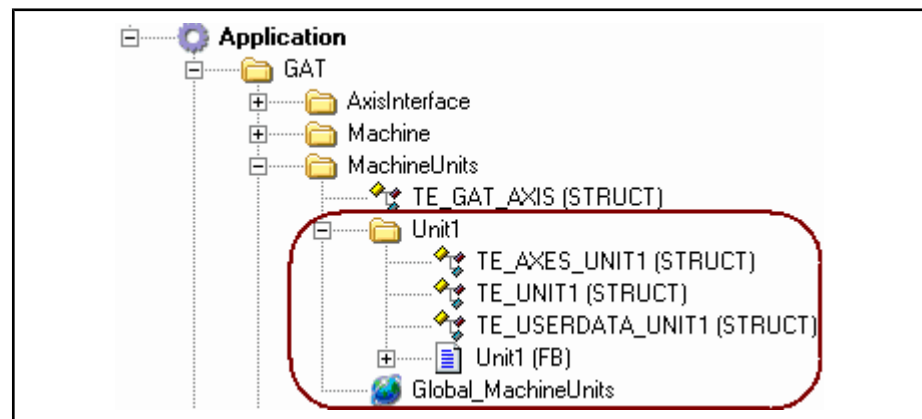


Fig.9-16: Unit folder in the project directory

9.7.2 Creation Rules

In order that the GAT unit be easy to maintain and reuse, it should be designed such that:

- Dependencies between the units are kept to a minimum
- No global variables are used within a unit

To achieve this, each unit has the following interface.

9.7.3 Unit Interface

General Information

The following figure shows the uniform unit interface.

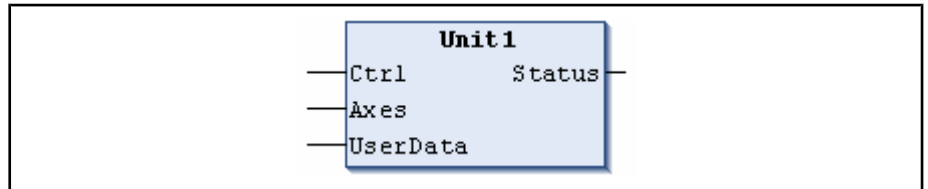


Fig.9-17: Unit interface

For high-performance data handling, the elements "Axes" and "UserData" of the unit interface are executed as references, i.e. the data required in the unit is not copied into the unit, but instead transferred in by reference to the unit.

The components of this interface are described in the following section.

Control and Status Structure

Each unit has a variable for control (**Ctrl**) and a variable for reporting status (**Status**). These elements are contained in the basic block of the GAT library and are managed from there.

Unit Axes

Each unit's interface has an axis container (**Axes**) that contains all of the axes used in the unit. The name of the axis container is the same for each unit, but the contents (type) differ because not every unit uses the same axes.

Each axis in the axis container (unit axis) is type TE_GAT_AXIS, which is based on type TE_BASE_UNIT_AXIS (see ML_TechTemplate library). The following table shows all of the elements that are available in structure type TE_GAT_AXIS.

Element	Type	Description
Name	STRING(20)	Name of the axis as character string (Initialized when the system is initialized; can be modified by the user)
AxisData	REFERENCE TO ML_AXISDATA_SM	Reference to the corresponding element in the global array AxisData (see ML_Base library)
DV	REFERENCE TO ML_DirectVarAxis	In case there is a local axis: Reference to the corresponding element in the global array DV_Axis (see ML_Base library) In case there is a link axis (axis from another control): dummy element
GV	REFERENCE TO ML_GlobalVarAxis	Reference to the corresponding element in the global array GV_Axis (see ML_Base library)
AxisRef	AXIS_REF	Axis reference
Disable	BOOL	TRUE: the corresponding AxisInterface data set will no longer be updated FALSE: Default A change to the value of this variable will first take effect in INIT mode ((STATE_INIT_COLDSTART or STATE_INIT_WARMSTART)

GAT with Decentral State Machine

Element	Type	Description
Ctrl	REFERENCE TO TE_AXISINTER-FACE_CONTROL	Reference to the control structure of the AxisInterface of the corresponding axis (for the reference to the corresponding entry of the global array arAxisCtrl_gb , see the global variable list Global_GAT_AxisInterfaces in the AxisInterface folder
Status	REFERENCE TO TE_AXISINTER-FACE_STATUS	Reference to the status structure of the AxisInterface of the corresponding axis (for the reference to the corresponding entry of the global array arAxisStatus_gb , see the global variable list Global_GAT_AxisInterfaces in the AxisInterface folder

Fig.9-18: All of the available elements of type **TE_GAT_AXIS**. Within the unit, an element can be accessed from **Axes.<AxisName>.<Element>**

All relevant data for an axis can be addressed using the **Axes** axis container.

The container is designed such that the axis data is accessed using the name of the axes. This generates program code that is easily readable.

The following table shows how to access axes in the axis container from within and outside a unit.

Access to a unit axis (e.g. axis "vAxis" in the axis container of the unit "Unit1")	Disables axis
Access outside the unit	<code>stAxesUnit1_gb.vAxis.Disable := TRUE;</code>
Access inside the unit	<code>Axes.vAxis.Disable := TRUE;</code>

The GAT wizard establishes the assignment of the unit axis and the axis configured in IndraWorks (real, virtual, ...). To do this, the corresponding physical axis is assigned to the unit axis in the method "mOneTimeInit" of the FB **TE_MachineControl** (see the following figure and [chapter 9.13.2 "Adding an Axis" on page 275](#)).

```

29  //§ WIZARD BEGIN AXES INITIALIZATION -----
30  //§ Required region for Wizard support - do not modify the contents of this region with the
31
32  //----- Start Axes of Unit1 -----
33  // set the references for all axes to avoid exceptions when accessing missing axes
34  FOR uiLoop := 1 TO uiMAX_AXIS_INDEX DO
35    stAxesUnit1_gb.arAll[uiLoop].AxisData REF= AxisData[uiLoop];
36    stAxesUnit1_gb.arAll[uiLoop].Ctrl REF= arAxisCtrl_gb[uiLoop];
37    stAxesUnit1_gb.arAll[uiLoop].Status REF= arAxisStatus_gb[uiLoop];
38    stAxesUnit1_gb.arAll[uiLoop].DV REF= DV_Axis[uiLoop];
39    stAxesUnit1_gb.arAll[uiLoop].AxisRef.AxisNo := NO_OBJECT; // set to not valid
40  END FOR
41
42  //----- Start Unit1 / MyAxis1 -----
43  stAxesUnit1_gb.MyAxis1.AxisData REF= AxisData[MlcVarGlobal.MyVirtualAxis1.AxisNo];
44  stAxesUnit1_gb.MyAxis1.Ctrl REF= arAxisCtrl_gb[MlcVarGlobal.MyVirtualAxis1.AxisNo];
45  stAxesUnit1_gb.MyAxis1.Status REF= arAxisStatus_gb[MlcVarGlobal.MyVirtualAxis1.AxisNo];
46  stAxesUnit1_gb.MyAxis1.DV REF= DV_Axis[MlcVarGlobal.MyVirtualAxis1.AxisNo];
47  stAxesUnit1_gb.MyAxis1.AxisRef := MlcVarGlobal.MyVirtualAxis1;
48  stAxesUnit1_gb.arAll[MlcVarGlobal.MyVirtualAxis1.AxisNo] := stAxesUnit1_gb.MyAxis1;
49  //----- End Unit1 / MyAxis1 -----
50
51  //----- Start Unit1 / MyAxis2 -----
52  stAxesUnit1_gb.MyAxis2.AxisData REF= AxisData[MlcVarGlobal.MyVirtualAxis2.AxisNo];
53  stAxesUnit1_gb.MyAxis2.Ctrl REF= arAxisCtrl_gb[MlcVarGlobal.MyVirtualAxis2.AxisNo];
54  stAxesUnit1_gb.MyAxis2.Status REF= arAxisStatus_gb[MlcVarGlobal.MyVirtualAxis2.AxisNo];
55  stAxesUnit1_gb.MyAxis2.DV REF= DV_Axis[MlcVarGlobal.MyVirtualAxis2.AxisNo];
56  stAxesUnit1_gb.MyAxis2.AxisRef := MlcVarGlobal.MyVirtualAxis2;
57  stAxesUnit1_gb.arAll[MlcVarGlobal.MyVirtualAxis2.AxisNo] := stAxesUnit1_gb.MyAxis2;
58  //----- End Unit1 / MyAxis2 -----

```

Fig.9-19: Axis assignment in the method "mOneTimeInit" in **TE_MachineControl**

An example from the figure above: the global axis generated in IndraWorks, "MyVirtualAxis1", is assigned the axis "MyAxis1" of the unit "Unit1".

GAT with Decentral State Machine

Passive axes

In principle, it is recommended that all axes that are **not actively** commanded in a unit be transferred using the I/O data container **UserData** specific to the unit. Typical examples for axes that are not actively commanded are master axes or encoder, link and C2C consumed axes.

Using the UserData container is associated with the following advantages:

- Link axes and C2C consumed axes can be transferred to a unit.
- Performance advantage because the AxisInterface is not called for the passive axes.
- Easier for users because it can be seen more quickly which axes are actively commanded in which unit.

The following procedure is recommended to configure a passive axis in the UserData container:

- Declare passive axes in the unit's UserData container.



The screenshot shows the GAT project structure on the left and the declaration of a passive axis in the UserData container on the right.

Project Structure (Left):

- Application
 - GAT
 - AxisInterface
 - Machine
 - MachineUnits
 - Unit1
 - TE_GAT_AXIS (STRUCT)
 - TE_AXES_UNIT1 (STRUCT)
 - TE_UNIT1 (STRUCT)
 - TE_USERDATA_UNIT1 (STRUCT)
 - Global_Unit1
 - Unit1 (FB)
 - Visualizations
 - GATMotionSyncProg (PRG)
 - GATProg (PRG)
 - Version_GAT_Decimal_12T01 (FUN)
 - MlcVarGlobal
 - UserVarGlobal
 - Bibliotheksverwalter
 - MotionProg (PRG)
 - PlcProg (PRG)
 - Taskkonfiguration
 - Visualization Manager

Declaration of a passive axis (Right):

```

3 // Declaration of the unit specific I/O data
4 TYPE TE_USERDATA_UNIT1 EXTENDS TE_BASE_USERDATA:
5   STRUCT
6     // $ -----Unit specific interface -----
7     // Name of the Unit
8     Name: STRING(20) := 'Module1';
9     //-----
10    // insert your unit specific input variables here...
11    //-----
12    ;
13    PassiveAxis1: AXIS_REF;
14    //-----
15    // insert your unit specific output variables here...
16    //-----
17    ;
18    //-----
19    // insert your unit specific in- output variables here...
20    //-----
21    ;
22    //-----
23    // Control, machine and product data, do not modify !
24    //-----
25    stControlDataActive : REFERENCE TO TE_CONTROL_DATA;
26    stMachineDataActive : REFERENCE TO TE_MACHINE_DATA;
27    stProductDataActive : REFERENCE TO TE_PRODUCT_DATA;
28
29  END_STRUCT
30 END_TYPE

```

Fig.9-20: Declaration of a passive axis "PassiveAxis1" in the UserData container of a unit

- Assign the global configured axis to the unit axis.

In the following example, the global axis "MyMasterAxis" is assigned to unit axis "PassiveAxis1". The global axis "MyMasterAxis" must be configured with IndraWorks in another step. Exceptions are C2C consumed axes that are not configured with IndraWorks.

GAT with Decentral State Machine

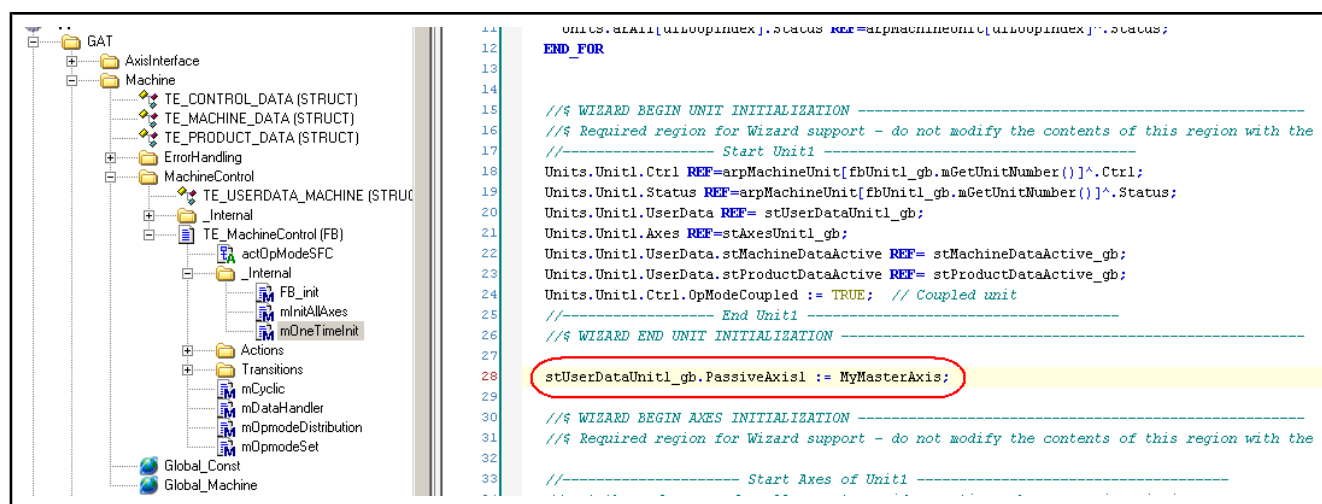


Fig.9-21: Example assignment of the global axis "MyMasterAxis"

- The declaration of global C2C consumed axes is implemented by the user with an entry in a global variable list. The following example of code shows the declaration of the global C2C consumed axis "MyMasterAxis", which is assigned to Axis 1 of Control 2.

Program:

```
VAR_GLOBAL MyMasterAxis: AXIS_REF := (AxisNo := 1, CntrlNo := 2); END_VAR
```

- If the passive axis is a C2C consumed axis, the axis data (e.g. position, velocity) can be read in the unit using the variable Variable GV_LinkAxis[]. The passive axis can also be used in the unit as a master axis.

Unit-specific I/O Data

In order to ensure reusability and multiple instantiation, accessing global data in a unit is not allowed. All I/Os required in units are made available using the I/O container **UserData**. The name of the I/O container is the same for each unit, but the contents (type) differ because not every unit uses the same I/O data.

The variable UserData is a data structure for which the user must define the elements.

The following figure shows where the I/O data for the Unit1 contained in the GAT are defined:

GAT with Decentral State Machine

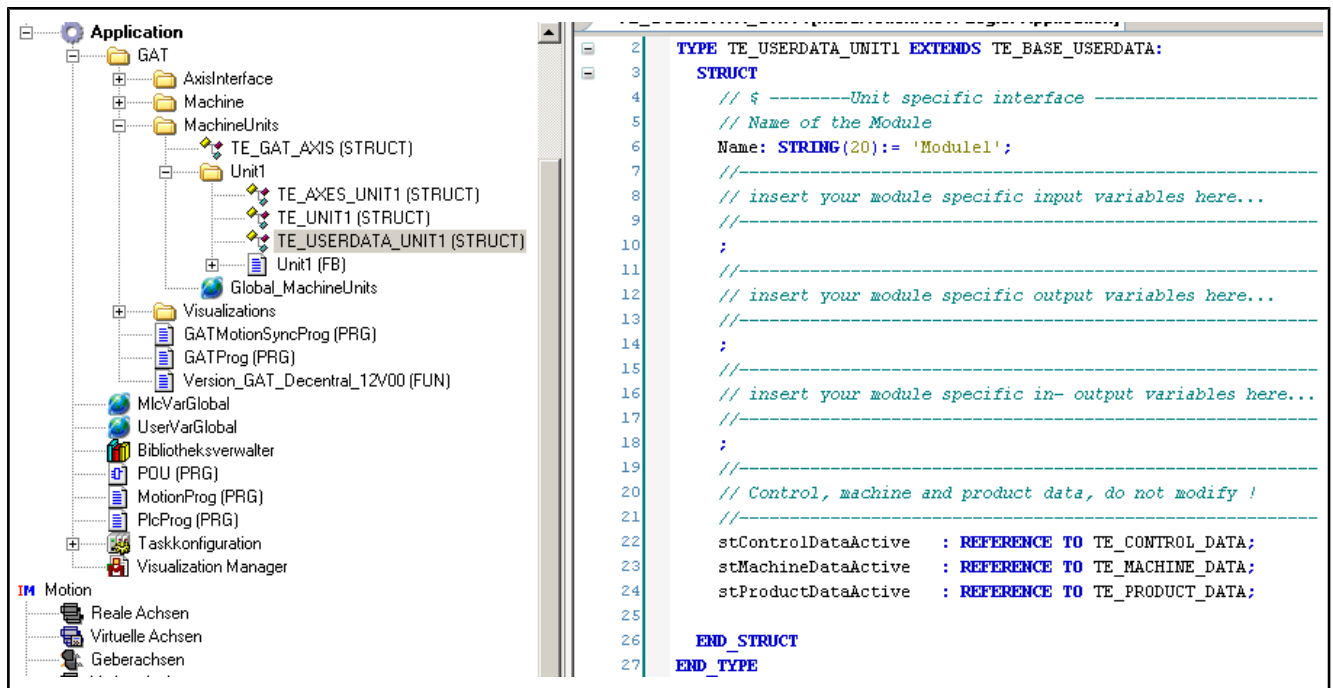


Fig. 9-22: Unit-specific I/O data

Unit Instances

Further instances can be created from an existing unit. These instances differ only in the assigned IndraWorks axes and assigned I/Os. The GAT wizard supports unit instantiation.

9.7.4 Operation Modes in the Unit

When adding a new primary operation mode, the wizard makes sure that a corresponding branch is created in the operation mode SFC in all coupled units. This step is only processed after the corresponding primary operation mode is selected.

With the functionality "LockOpModeAcknowledge", each individual unit has the option to delay the operation mode acknowledgement. In this way, switching to the next initialization step can also be delayed, for example, if the related activities required several cycles (e.g. writing parameters).

The operation mode acknowledgement is executed in the method "mOpModeAcknowledge" of the unit. This method is created by the wizard and modified when the primary operation modes are changed.

9.7.5 Sequential Programming

Introduction to sequential programming

The sequential programming aims to facilitate sequential programming in the IndraMotion MLC, in particular the automatic processes. The IEC - 61131 (Structured Text) programming language is used for sequential commands. Thus, no additional programming environment is required, as the editor integrated in IndraWorks can also be used for sequential programming. The sequential programming is completely integrated in the GAT (with a decentral state machine). The GAT wizard was extended for the configuration of sequential programs. The user can use the wizard to create or remove sequential programs. The corresponding axes to execute sequential commands are automatically prepared when a sequential program is created.

GAT with Decentral State Machine

Interaction between sequential programming and the GAT

Sequential programs are assigned to exactly one operating mode of a unit. Before the commissioning, the programs and the axes have to be configured. The sequential programming is configured by the GAT wizard as described in the following.

- The sequential programs of an operating mode are saved in a folder (Mode <Operating mode>) below the unit folder <UnitName>. The sequential sequence is programmed in the programs.
- Each sequential program is assigned to an individual task. The tasks can be found in the task configuration.
- In order to start and stop sequential programs, one instance of the Seq-TaskControl block is required for each. The instances of this block are inserted in the Global Variable List Global <UnitName> of the corresponding unit. By default, the start and stop command of the sequential program is inserted in the input or output step of the respective operating mode.

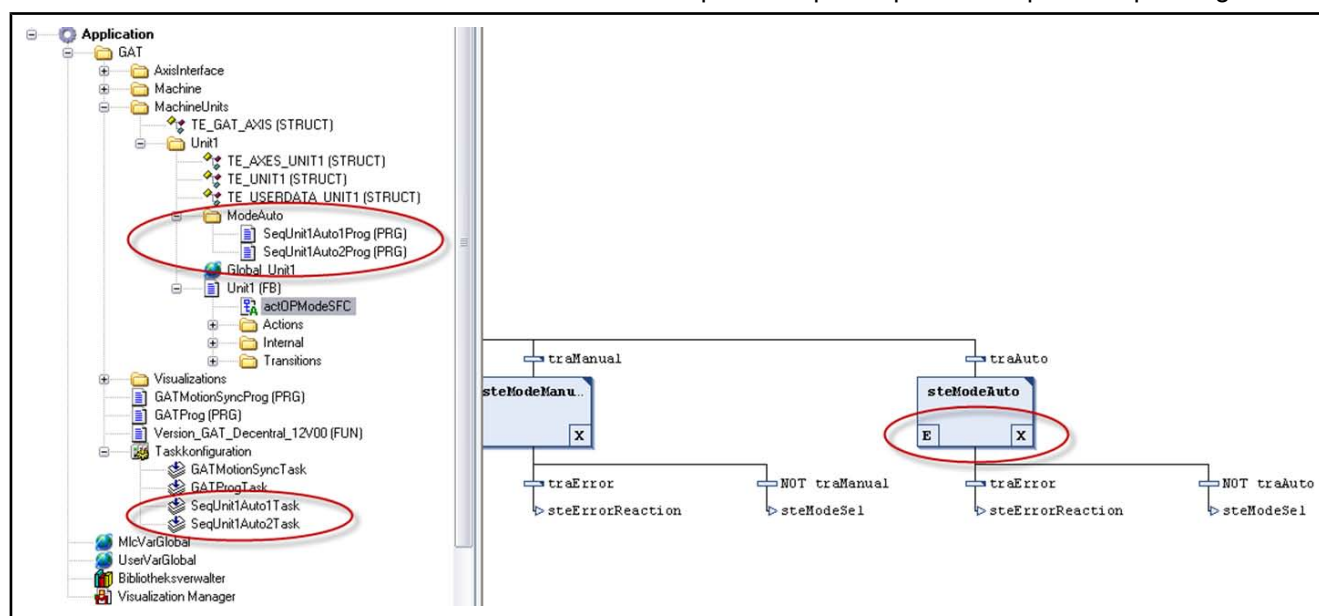


Fig. 9-23: Input and output action by sequential programming

- The GAT wizard assigns the unit axes to the sequential programming. To do this, the axis references of the unit axes are assigned to the Seq[...] field in the "mOneTimeInit" method of the TE_MachineControl function block.
- The sequential axis handlers are called motion-synchronously in the GAT-MotionSyncProg.
- The command Seq[...].Disable() is entered for each axis by the GAT wizard in the output step of the operating mode. This command ends the access of the sequential programming to the axes when the operating mode is closed.

More notes on sequential programming

More notes on sequential programming, e.g. that the function blocks, commands and interfaces are contained in the function description for the sequential programming (see the documentation "Rexroth IndraLogic XLC IndraMotion MLC xxVRS Sequential Programming").

xx - Version

9.7.6 Unit Coordination

Because a unit does not contain any information about other units, unit coordination must be carried out at a superordinate position.

GAT with Decentral State Machine

The following function blocks are specified for superordinate unit coordination:

- **TE_ErrorHandling** coordinates the error distribution between the units. In the basic version, all errors are forwarded to all units.
- **TE_MachineControl** processes all cross unit activities that are not related to errors, e.g. comprehensive operation mode management.

The function block **TE_MachineControl** already contains the method "**mCyclic**", which is called in every cycle. In this cyclically called method, I/O information is to be exchanged between the units and from the units to the I/O points of the machine.

Because the unit configuration or the assignment of global axis to a corresponding unit axis does not change during runtime, these activities are carried out in the method "**mOneTimeInit**" of **TE_MachineControl**. This method is called for initializing the GAT using the machine organization unit.

The code contained in the "**mOneTimeInit**" method is managed by the wizard. When adding or removing units or unit axes, the corresponding entries are added or removed.

9.8 Task Management

9.8.1 Task Configuration, GAT

By default, the GAT contains the following tasks.

	Priority	Cycle time	Watchdog time	Watchdog sensitivity	Program call
GATMotionSync-Task	2	=Motion cycle time	T#2ms	1	GATMotionSyncProg
GATProgTask	5	T#10ms	T#20ms	2	GATProg

9.8.2 Adding a BackgroundTask

If the runtime for the **PlcTask** is too long, the load on the **PlcTask** can be reduced by moving tasks that are not time critical to a background task. Please note, however, that a greater load is placed on the entire system due to the additional task **Switch PlcTask <> BackgroundTask** (see also [chapter 5.5 "Task Management" on page 34](#)).

	Priority	Cycle time	Watchdog time	Watchdog sensitivity
BackgroundTask	~6	T#40ms	-	-

In order to add a **BackgroundTask**, the following steps are required:

- Task configuration, adding the task "**BackgroundTask**", for settings see the table above.
- Add the **BackgroundProg** (program) and call **BackgroundProg** in the configuration of the **BackgroundTask**
- Add the method "**mBackground**" at the FB **TE_MachineControl**. Then call this method in the **BackgroundProg**. Program line with call: **fbTE_MachineControl_gb.mBackground()**. All variables of the FB **TE_MachineControl** are available in the method "**mBackground**".

GAT with Decentral State Machine

9.9 Error Handling

9.9.1 Error Detection and Diagnostics

	The program template GAT contains predefined functionalities for detecting, handling and deleting errors. The following describes the basic procedure:
Local error handling in a unit	Each unit scans its assigned components for errors. If, for example, a drive of the unit has an error, the unit also has a local error. By default, an error output is set which forwards this local error to the Error Handler so that the other units can also react to the error. In case of error, the error bit in the unit status output is set automatically in the standard case. If this is not desired in certain cases, the error forwarding can be suppressed by implementing the method mForwardLocalError in the unit. The method mForwardLocalError of the unit is called automatically by the GAT in case of error and also receives the diagnostic number of the detected local error. Based on this information, the application code in this method decides if the error output is set (mForwardLocalError := TRUE) or is not set (mForwardLocalError := FALSE). Even if the error is not forwarded, a local error reaction occurs in the unit immediately, in the same cycle. The following are taken into consideration as sources of error within a unit: • Axis error (e.g. axis lag error) • Application-specific error (e.g. safety door is open) • Function block errors (e.g. due to an error of a called function block)
DisableErrorReaction	Setting the control bit DisableErrorReaction allows each unit to be set individually with respect to calling the error reaction. If this control bit is set, no error reaction occurs in the unit. This control bit is switched off by default.
Cross unit error handling	The FB TE_ErrorHandling monitors all sources of errors across the units from a central position. If an error is detected, it is forwarded to all units as an external error (unit inputs Ctrl.ExtError and Ctrl.ExtErrDiagNo). As an option, the error distribution to the respective units in the application program can also be suppressed. To do this, the method mDistributeError of TE_Errorhandling must be adapted accordingly. Then, in the next cycle, a corresponding error reaction occurs in the respective units, The function block TE_ErrorHandling continues to coordinate the acknowledgement of the error and forwards this information from the central position to all units. The units are then responsible for executing the error acknowledgement at the assigned components. If several errors occur simultaneously, the most severe is displayed. If several errors with the same degree of severity have occurred simultaneously, the most recently detected error is output. The diagnostics of the machine are output at the output of TE_ErrorHandling using the variable stErr_Handling_Status_gb:

GAT with Decentral State Machine

Element	Type	Description
Error	BOOL	TRUE indicates an error
DiagNo	DWORD	Number according to the IndraMotion MLC diagnostics with the following extensions: 16#F0AE xxxx: Application error (e.g. safety door is open) (xxxx = can be defined by the user) 16#F0FB 0000: FB error (e.g. error at a TechFB) -> for details, see ErrorID / ErrorIdent 16#F0AA 0000: GAT internal error -> for details, see ErrorID/ErrorIdent 16#F0B1 0000: Local inline error 16#F0B2 0000: Profibus error
ErrorID	ERROR_CODE	Detailed diagnostics in case of FB + GAT errors (DiagNo = 16#F0FB/AA 0000)
ErrorIdent	ERROR_STRUCT	Detailed diagnostics in case of FB + GAT errors (DiagNo = 16#F0FB/AA 0000)
DiagMessage	STRING(60)	It contains the diagnostic text for drive and control errors

The **DiagNo** element is only for use on an HMI. The error texts are assigned the numerical information in any desired language there.

DiagMessage, **ErrorID** and **ErrorIdent** provide additional information that is primarily displayed during commissioning by the programmer, e.g. using an IndraLogic visualization.

Note regarding application-specific errors

mSetAppError

By calling the method `mSetAppError(Number)`, application errors (e.g. safety door is open) can be signaled. The "number" transferred at the method input corresponds with the application error number.

If an application error was signaled, the central diagnostic system in the unit performs cyclic monitoring of the application errors that have been signaled and generates the application error `DiagNo = Error 16#F0AE xxxx` (xxxx = application error number).

Numbers in the range 1 to `uiMAX_MODULE_APP_ERROR_INDEX` are valid for application errors in the units. The constant "`uiMAX_MODULE_APP_ERROR_INDEX`" has the value 64 by default. If necessary, this constant can be adjusted in the global variable list "Global_Const" below the "Machine" folder.

Numbers in the range 1 to `uiMAX_MACHINE_APP_ERROR_INDEX` are valid for application errors in the machine. The constant "`uiMAX_MACHINE_APP_ERROR_INDEX`" has the value 64 by default. If necessary, this constant can be adjusted in the global variable list "Global_Const" below the "Machine" folder.

Code example for signaling an application error:

Program:

```
// Recommended code to issue user specific errors (Example)
IF bDoor10open THEN
    mSetAppError(1);
END_IF
```

Note regarding severe errors

GAT with Decentral State Machine

The control diagnostic system defines 10 degrees of "severity" (see IndraMotion MLC diagnostics). The degree of severity affects the error reaction (see [chapter 5.6 "Error Management" on page 40](#)).

- DiagNo = 16#F0xx xxxx is the lowest degree of severity (e.g. application errors, FB errors)
- DiagNo = 16#F9xx xxxx is the highest degree of severity (fatal system errors)

The error with the highest degree of severity is displayed if several errors occur simultaneously.

Note regarding function block errors

mSetFBError If errors occur in called function blocks (e.g. technology FBs), calling the method "mSetFBError(ErrorID, ErrorIdent)" is recommended along with transferring the "ErrorID" and "ErrorIdent" outputs of the FB that caused the error to the method. This ensures that the error is forwarded to the diagnostic system of the unit and the "ErrorID" and "ErrorIdent" are made available to the diagnostic outputs of the unit.

Code example for signaling function block error:

Program:

```
// Recommended code a used FB issues an error (Example)
IF fbCrossCutter.Error THEN
    mSetFBError(ErrorID:=fbCrossCutter.ErrorID,
                ErrorIdent:=fbCrossCutter.ErrorIdent);
END_IF
```

9.9.2 Error Reaction

The error reaction is contained in the action **actErrorReaction_entry** of the unit. In the default setting, each unit automatically executes a user-specific error reaction that halts all unit axes using "ModeAH". This application-specific error reaction can be modified in the units.



It is recommended that hydraulic axes be brought to a standstill using "ModeAb".

In addition, the control firmware reacts to severe errors as follows:

For axis errors with a severity degree less than 2, the axes are not brought to a standstill on the firmware side.

For errors with a severity degree greater than 2, all axes are brought to a standstill by the control firmware.

DisableErrorReaction If the user-specific error reaction of the unit is not desired, it can be switched off using the unit input "DisableErrorReaction". The following assignment in the FB TE_ErrorHandling can be used to switch off the error reaction of the unit:

Program:

```
Units.<UnitName>.Ctrl.DisableErrorReaction := TRUE;
```

Switching off the error reaction can be used for a service operation mode, for example, in order to move axes even in the error state.

Decelerating all axes The following error reaction is contained in the action **actErrorReaction_entry** of the unit in the initial delivery. It decelerate all axes to a standstill using "ModeAH".

Program:

```
// Error reaction example 1 // Decelerate all unit axes to standstill => switch all axes to ModeAH
FOR uiLoopIndex:=1 TO Axes.uiAxesNum DO
  IF mGetAxis(AxisNumber:=uiLoopIndex, PointerToAxis=>pCurrentAxis) THEN
    pCurrentAxis^.Ctrl.Admin._OpMode.en := ModeAH; // Set ModeAb for hydraulic axis
  END_IF
END_FOR
```

Synchronous deceleration

Often it is required that axes be decelerated **synchronously** (to prevent film or paper from tearing, for example). In these applications, it is recommended that if at all possible, all axes remain connected in synchronous movement, and that only the master is decelerated. This use case is shown in the following example:

Program:

```
// Error reaction example 2
// Keep all axes in synchronization with the master axis
// and decelerate only the master axis
FOR uiLoopIndex:=1 TO Axes.uiAxesNum DO
  IF mGetAxis(AxisNumber:=uiLoopIndex, PointerToAxis=>pCurrentAxis) THEN
    IF pCurrentAxis^.AxisData.SynchronizedMotion THEN
      // Axis is in synchronized motion => no reaction as the master axis
      // decelerates to standstill and the slave axis follows in synchronization
    ELSE
      // Axis is not in synchronized motion => command to standstill
      pCurrentAxis^.Ctrl.Admin._OpMode.en := ModeAH;
    END_IF
  END_IF
END_FOR
```

If axes are to be brought to a standstill synchronously, the following must also be observed:

- For synchronous deceleration, the module bus reaction for the real axes must also be switched off (see the "Error reaction" section in the drive documentation). Furthermore, in the "best possible deceleration" of the real axes, "30s NC reaction" must be set.
- In addition, bit 0 of A-0-0651 = TRUE must be set so that the control does not execute an automatic error reaction (see also the "Error reaction" section in the "Rexroth IndraMotion MLC xxxVRS function description").

xx - Version

9.9.3 Clearing Errors

The central error clearing is triggered by a positive edge on the element **"Clear-AllError"** of the variable **"stErr_Handling_Ctrl_gb"** (input of TE_ErrorHandling) and it is automatically forwarded to all subordinate components (e.g. IMC interface¹⁾, Units, AxisInterface ...).

If an error-free state is achieved, the operation mode selected by the operator becomes active again.

9.10 Product and Machine Data Handling

9.10.1 Introduction

The GAT provides functions in order to prepare machine and product data as a data set that is activated at a precisely defined point in time. Machine and product data are viewed and managed separately. The data handling described in the following is implemented in the method "mDataHandler" of the TE_MachineControl as an open source concept and can be extended or modified accordingly.

¹⁾ *IndraMotionControl Interface = IndraLogic library with an interface to the control, e.g. clear errors*

GAT with Decentral State Machine

9.10.2 Machine Data Handling

Machine data are the data that refer to the machine and can be changed during operation. Examples include production velocity, acceleration and deceleration ramps...

These machine data are edited/prepared using the variable "stMachineDataEdit_gb" of the variable list "Global_Machine". The content of these variables is application-specific and is specified using the data structure "TE_MACHINE_DATA".

A positive edge on the input "stControlDataActive_gb.bSetMachineData" causes these data to be written on the variable "stMachineDataActive_gb" and thus become effective. The acknowledgement is carried out using "stControlDataActive_gb.bSetMachineDataDone".

Conversely, a positive edge at the input "stControlDataActive_gb.bGetMachineData" causes the active data set to be copied to the data set "stMachineDataEdit_gb", which can be edited, and it is then acknowledged using "stControlDataActive_gb.bGetMachineDataDone".

9.10.3 Product Data Handling

Product data are data that describe the product and are required for manufacturing. Examples include product length, recipe data...

These product data are edited/prepared using the variable "stProductDataEdit_gb" of the variable list "Global_Machine". The content of these variables is application-specific and is specified using the data structure "TE_PRODUCT_DATA".

A positive edge on the input "stControlDataActive_gb.bSetProductData" causes these data to be written on the variable "stProductDataActive_gb" and thus become effective. The acknowledgement is carried out using "stControlDataActive_gb.bSetProductDataDone".

Conversely, a positive edge at the input "stControlDataActive_gb.bGetMachineData" causes the active data set to be copied to the data set "stProductDataEdit_gb", which can be edited, and it is then acknowledged using "stControlDataActive_gb.bGetProductDataDone".

9.11 Visualization

9.11.1 Switching the Visualization

In the GAT, there are functions prepared for controlling the machine either with the integrated commissioning visualization or with an external visualization. However, only one visualization is allowed to be active at a time. To ensure this, both the integrated commissioning visualization and the external visualization each have their own variables.

The integrated commissioning visualization uses the variable "**stControlDataVisuL_gb**" from the global variable list "GlobalMachine".

The external visualization (e.g. via a field bus or customer HMI) uses the variable "**stControlDataCustomer_gb**" from the global variable list "GlobalMachine".

In the application program, the variable "**stControlDataActive_gb**" is effective, which references either the integrated or the external visualization. The integrated commissioning visualization can switch the reference using the **Remote On** button and thus either take over or give up control.

GAT with Decentral State Machine

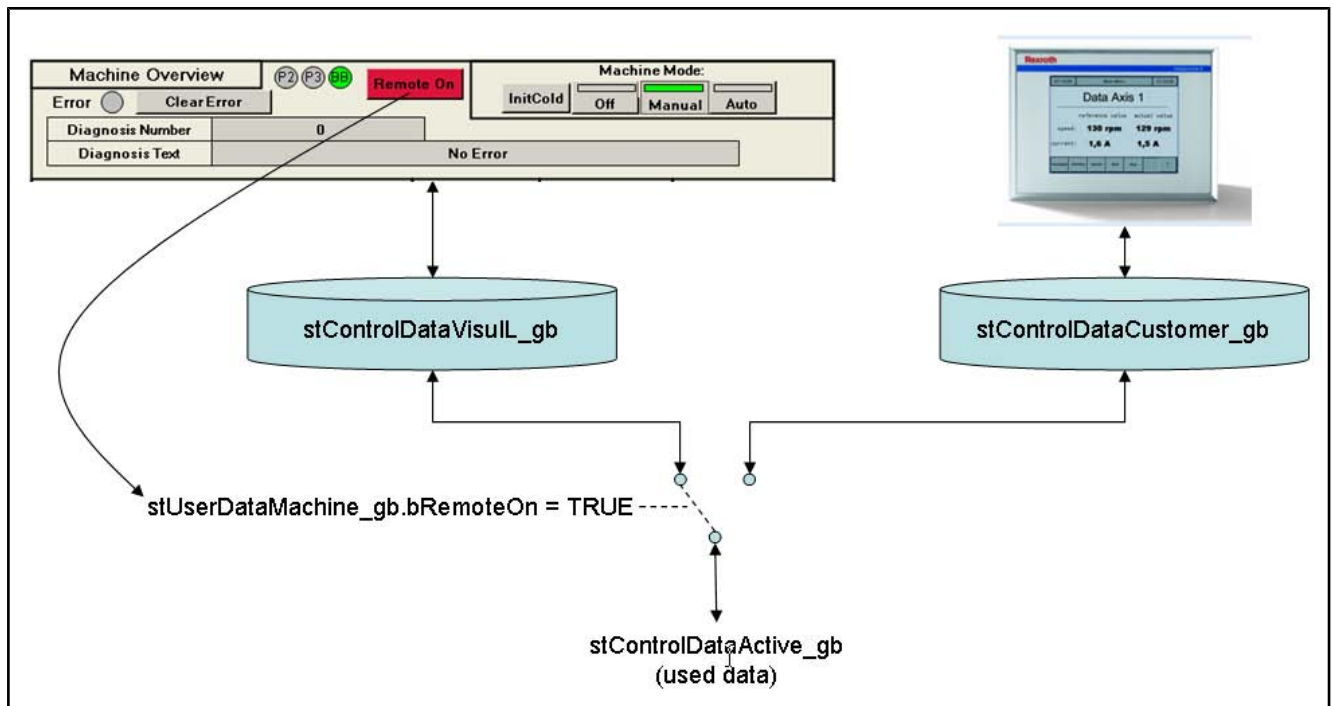


Fig. 9-24: Switching between the commissioning and external visualizations

9.11.2 Internal Visualization for Commissioning Support

General Information

The GAT contains predefined IndraLogic visualization objects that offer assistance during commissioning as well as extended diagnostic possibilities.

System_Overview

The visualization element "System_Overview" can be instantiated for individual axes.

This is implemented as an example in the "System_overview_All_Axis" visualization for the "MyVirtualAxis1" and "MyVirtualAxis2" axes.

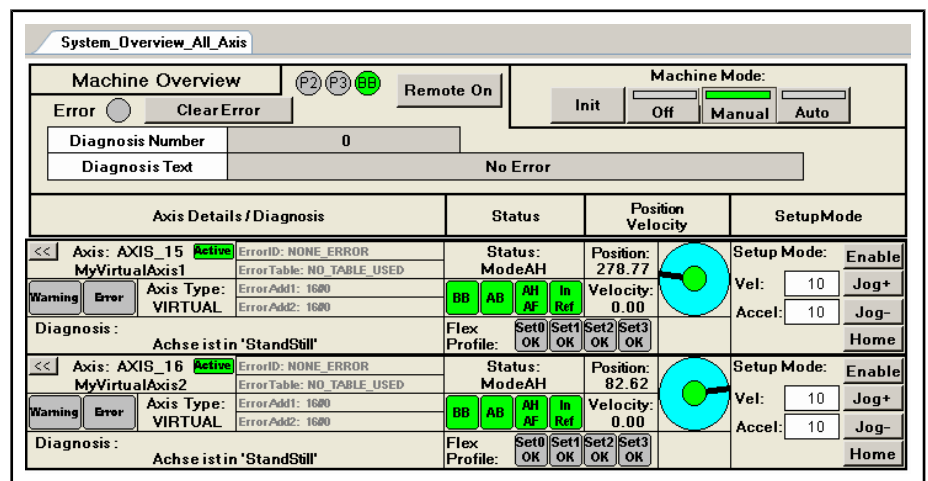


Fig. 9-25: Internal visualization for assistance during start up



The commissioning visualization is effective only after the **Remote On** button has been clicked.

GAT with Decentral State Machine

The "Frame" object can be used to create an instance of the visualization object System_Overview in the visualization.

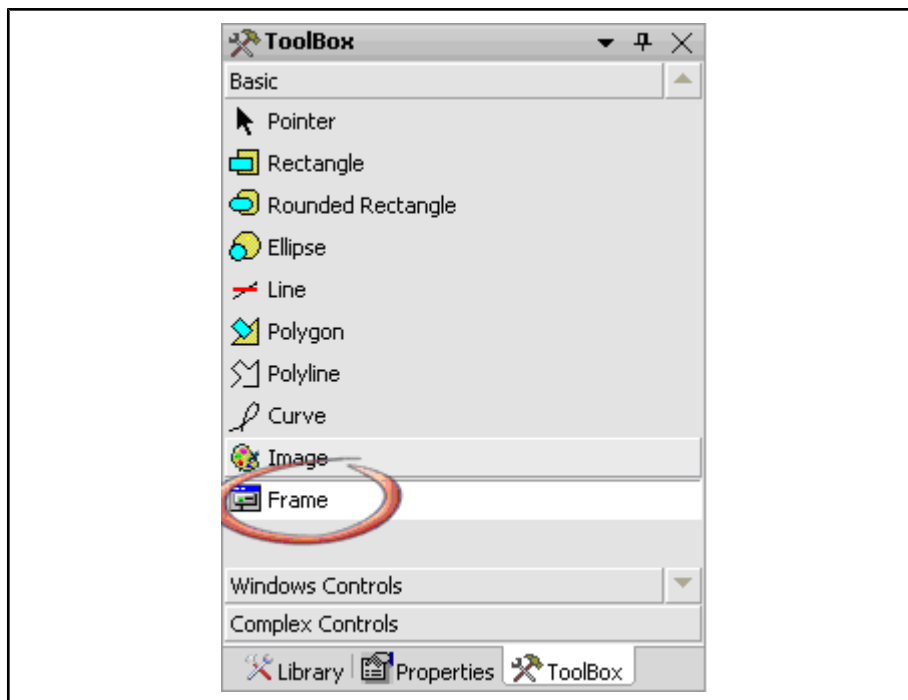


Fig.9-26: Creating an instance of the visualization object "System_Overview"

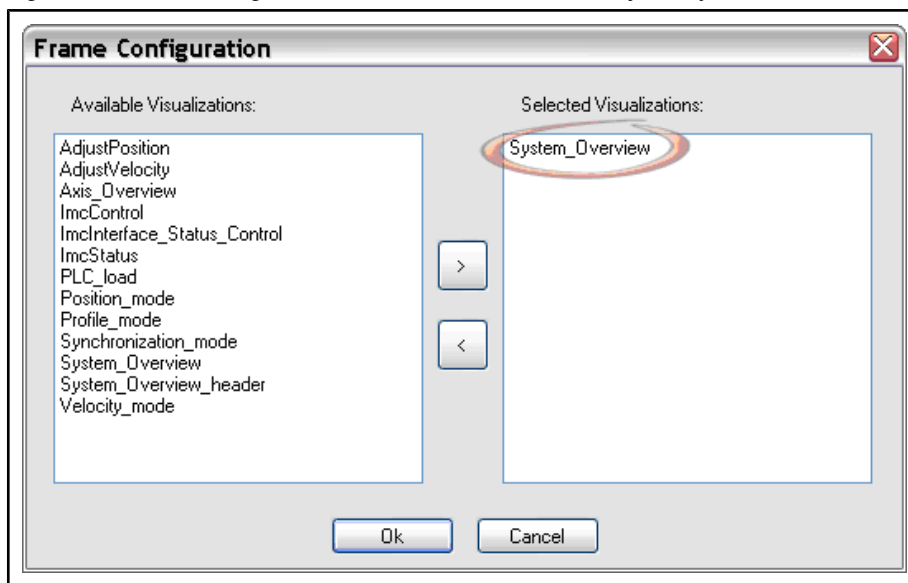


Fig.9-27: Creating an instance of the visualization object "System_Overview"

Now the corresponding axis reference must be entered under Properties - References.

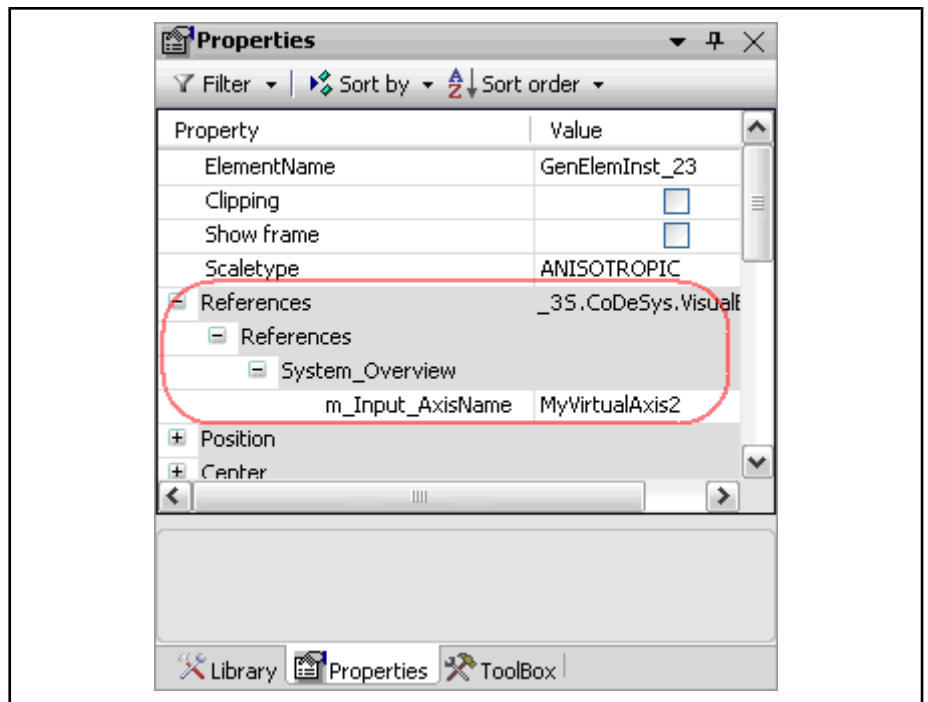


Fig. 9-28: Creating an instance of the visualization object "System_Overview"

9.11.3 External Visualization with Separate HMI (WinStudio Visualization)

General Information

A predefined WinStudio visualization is available for the GAT. A wizard can be used to add this visualization into the project.



To edit the WinStudio visualization, at least the following development environment license (or higher) is required:

WinStudio development license, max. 1,500 variables (Mat.No. R911323596)

For execution on a VEP, at least the following runtime license is required:

WinStudio runtime license for devices using Windows CE as an operating system, max. 1,500 variables (Mat. No. R911323620)

Adding the WinStudio visualization



In order to add the WinStudio visualization to the project, a corresponding visualization device must be present in the project.

Use "drag and drop" to add a VEP from the device library into the project tree and configure it as follows:

- Device type: VEPxx (CE)
- Application type: WinStudio application

GAT with Decentral State Machine

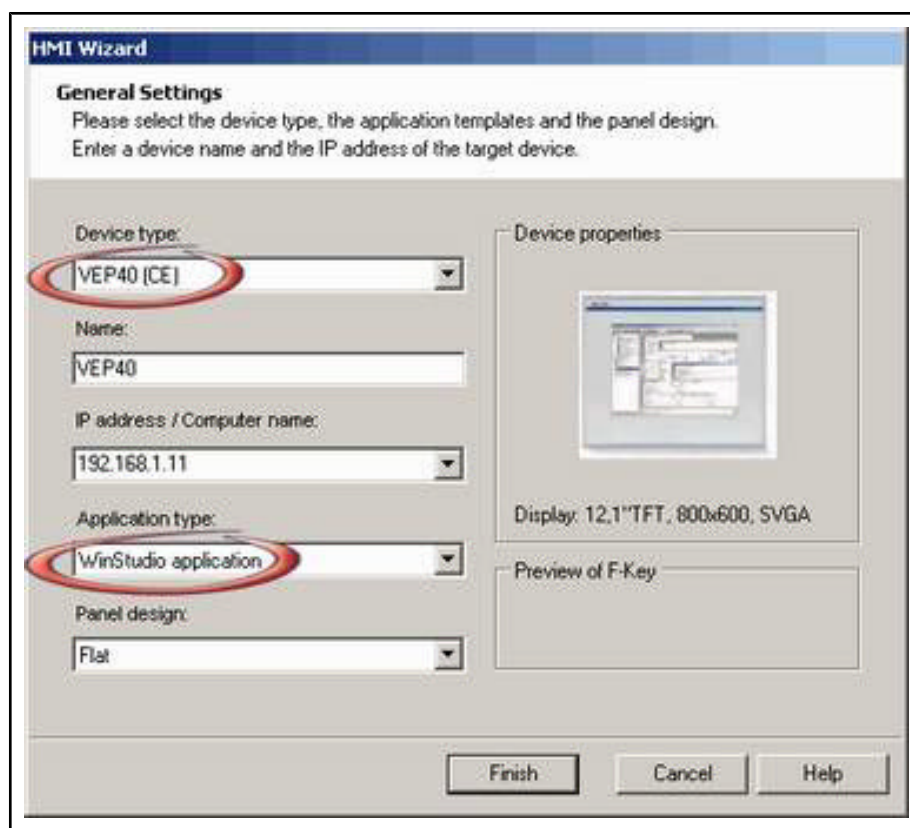


Fig.9-29: Adding a visualization device

If a corresponding visualization device is present, the WinStudio visualization is added using the following link and the wizard.

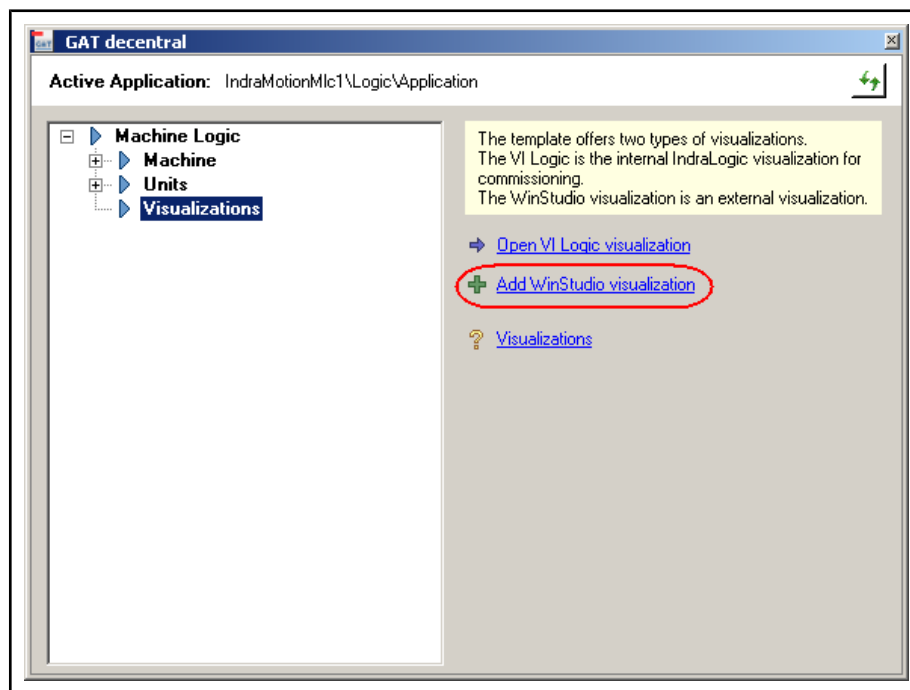


Fig.9-30: Adding a WinStudio visualization

All of the project's visualization devices that correspond to the given configuration are included in the following wizard dialog.

GAT with Decentral State Machine

After the corresponding visualization device is selected, click **Apply** to add the WinStudio visualization.

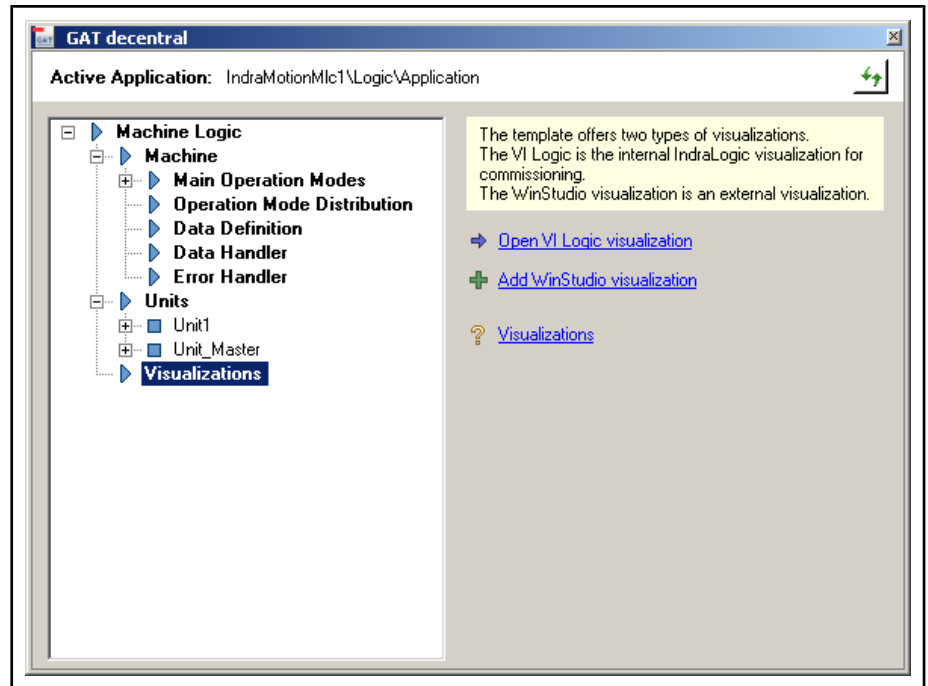


Fig.9-31: Selecting a visualization device and adding a WinStudio visualization



Existing visualization data from the visualization device along with the symbol configuration of the control are overwritten when the WinStudio visualization is added!

After the visualization is added, its data can be edited.

The control address is configured in the BR-WS driver properties dialog:

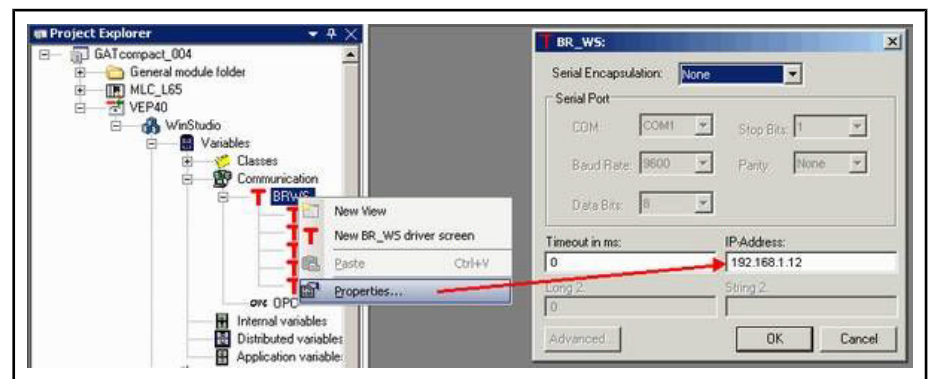


Fig.9-32: Setting the control's IP address

Now use  "Start application" in the WinStudio node context menu to start the emulation of the WinStudio visualization.

Dialog Structure

The operation screens are divided into 3 subsections:

- Header with status information
- Main screen with menu-related displays and function keys
- Footer with keys for switching from one user map to another

GAT with Decentral State Machine

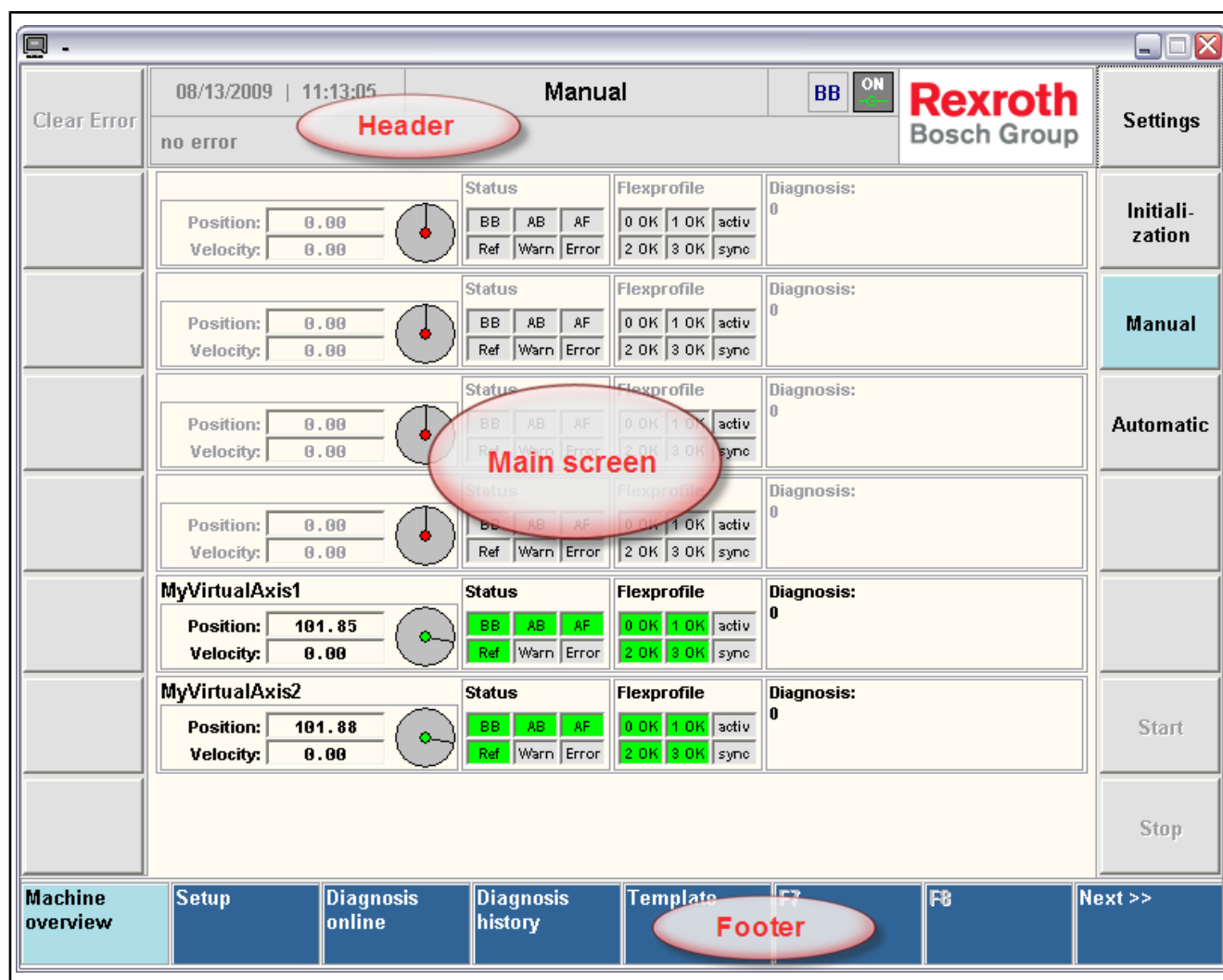


Fig.9-33: WinStudio visualization

The **header** contains:

- Operation mode display
- Global function keys for "deleting errors" and opening the "Settings" dialog (for switching languages, setting the decimal place for numerical displays)
- Display of current machine diagnostics
- Display of current SERCOS phase (including active phase switching)
- Display of communication status

The **main screen** contains displays and function keys related to user maps (e.g. axis information, operation mode switching, ...)

The **footer** contains the keys for switching from one user map to another.

Extending the Visualization

Adding an operation screen

To extend the visualization, new user maps can be created by copying the existing "template" screens and inserting them with new names.

GAT with Decentral State Machine

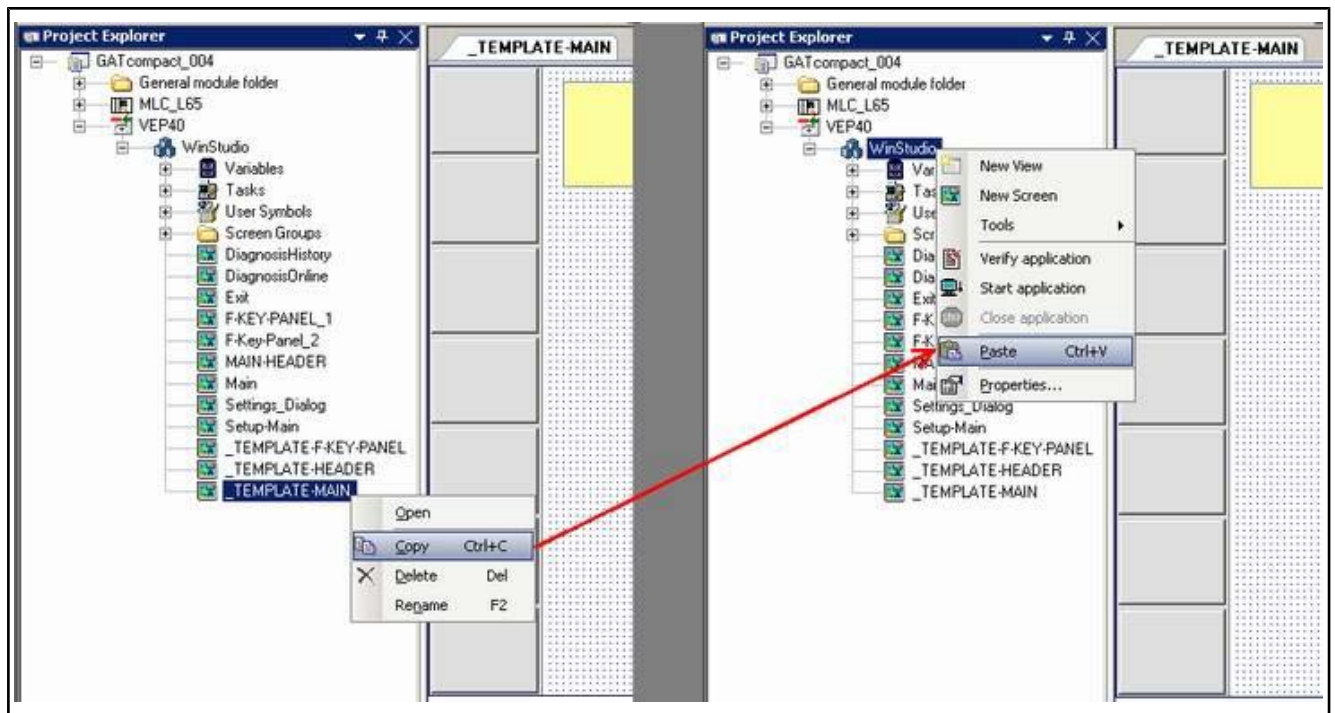


Fig.9-34: Duplicating an operation screen

The elements for displaying the axis data are grouped individual objects that can be duplicated by "copying" and "pasting". To reconfigure the inserted object for a different axis the Replace function must now be used for the respective object to adapt the axis array index (see the following figures).



Fig.9-35: Reconfiguring the inserted object

GAT with Decentral State Machine

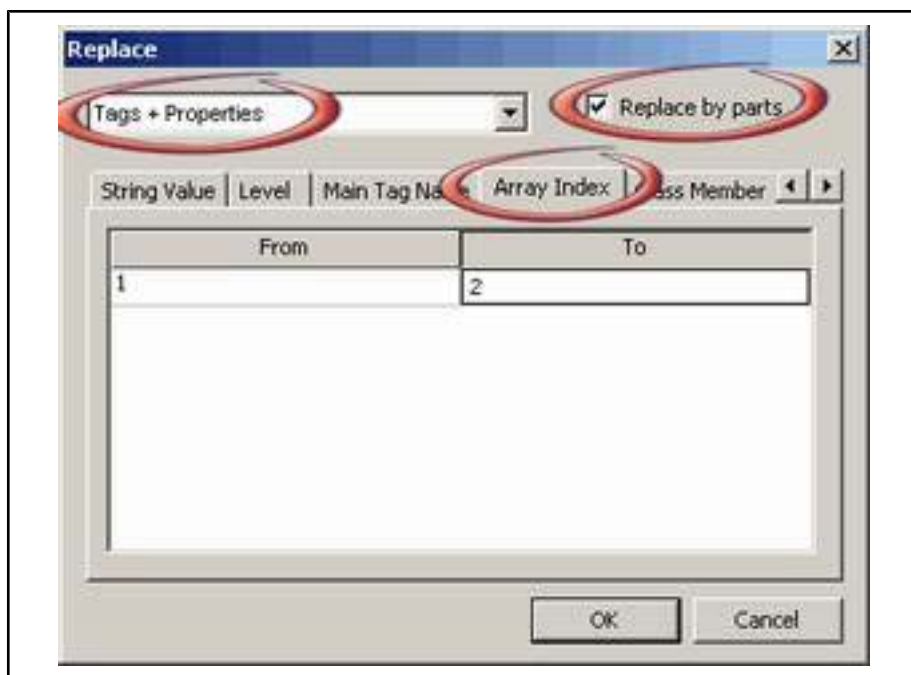


Fig.9-36: Reconfiguring the inserted object

Extending the footer

The keys used to switch from one user map to another are stored as graphic objects that can be parameterized (linked symbol). They have the following transfer parameters:

- Caption -> Key label
- Disable -> Input for blocking key functionality (grayed out label)
- ScreenName -> Screen name for the operation screens to be opened (or screen group)
- ScreenNumber -> Number to be assigned to the screen (to highlight the respective key)

Extending the Maximum Number of Axes

After import, the visualization is designed for 16 axes. This corresponds with the maximum number of axes for the IndraControl L25. If more axes are required, the corresponding driver sheets of the visualization must be called up.

GAT with Decentral State Machine

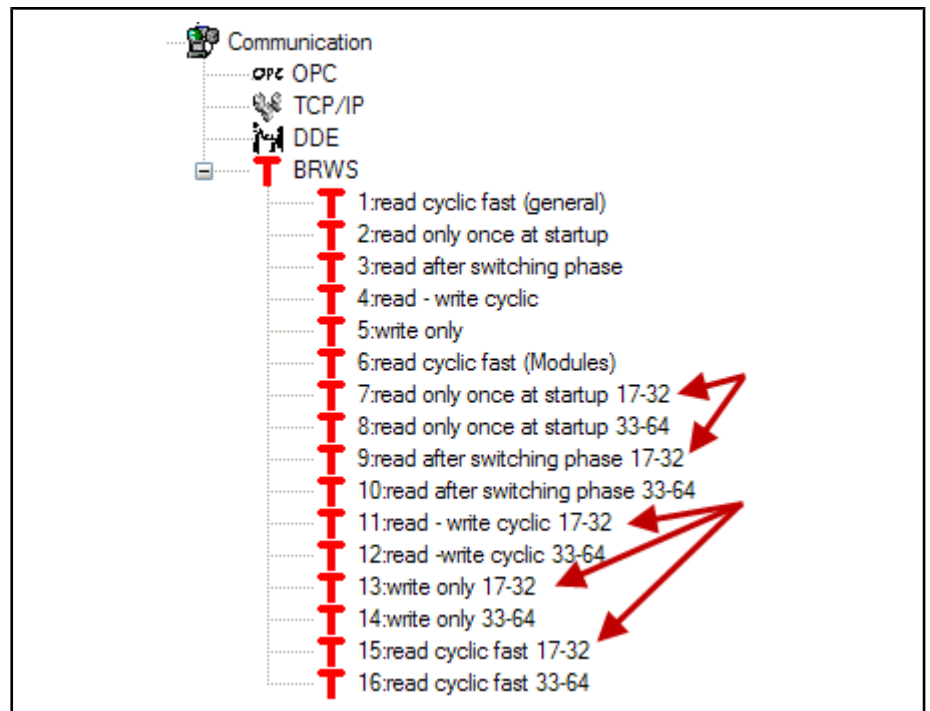


Fig.9-37: Overview of driver sheets

The settings can be opened by double-clicking on the respective driver sheet. For the inactive driver sheets, the read and write trigger variables are empty and the entries from the active driver sheets must be added.

Example based on the driver sheet "read only once at startup 17-32":

There the entry "stCom.TrigOnce" from the driver sheet "read only once at startup" must be copied into the driver sheet "read only once at startup 17-32" as a trigger variable.

This screenshot shows the settings for the active driver sheet "read only once at startup". The "Read Trigger" field is circled in red and contains the text "stCom.TrigOnce". Other fields include "Enable Read when Idle", "Read Completed", "Read Status", "Write Trigger", "Enable Write on Tag Change", "Write Completed", "Write Status", "Station", "Header", "Sheet_2", and "Increase priority" checkbox.

Fig.9-38: Driver sheet settings of the active driver sheet

This screenshot shows the settings for the driver sheet "read only once at startup 17-32". The "Read Trigger" field is circled in red and is empty. A red arrow points to the "Description" field, which contains the text "read only once at startup 17-32". Other fields include "Enable Read when Idle", "Read Completed", "Read Status", "Write Trigger", "Enable Write on Tag Change", "Write Completed", "Write Status", "Station", "Header", "Sheet_7", and "Increase priority" checkbox.

Fig.9-39: Driver sheet settings for extending to 32 axes

GAT with Decentral State Machine

The respective entries must be adjusted to match for all driver sheets. The numbers following the driver sheets indicate the axes for which these driver sheets were created.

Communication Connection

The communication between visualization and control is executed symbolically using the BR-WS driver. To do this, generate the symbol entries for the PLC variables to be visualized (see [chapter 5.7.3 "Providing Data Using the PLC" on page 44](#)).

The required symbol configuration for the predefined WinStudio visualization contains the following entries:

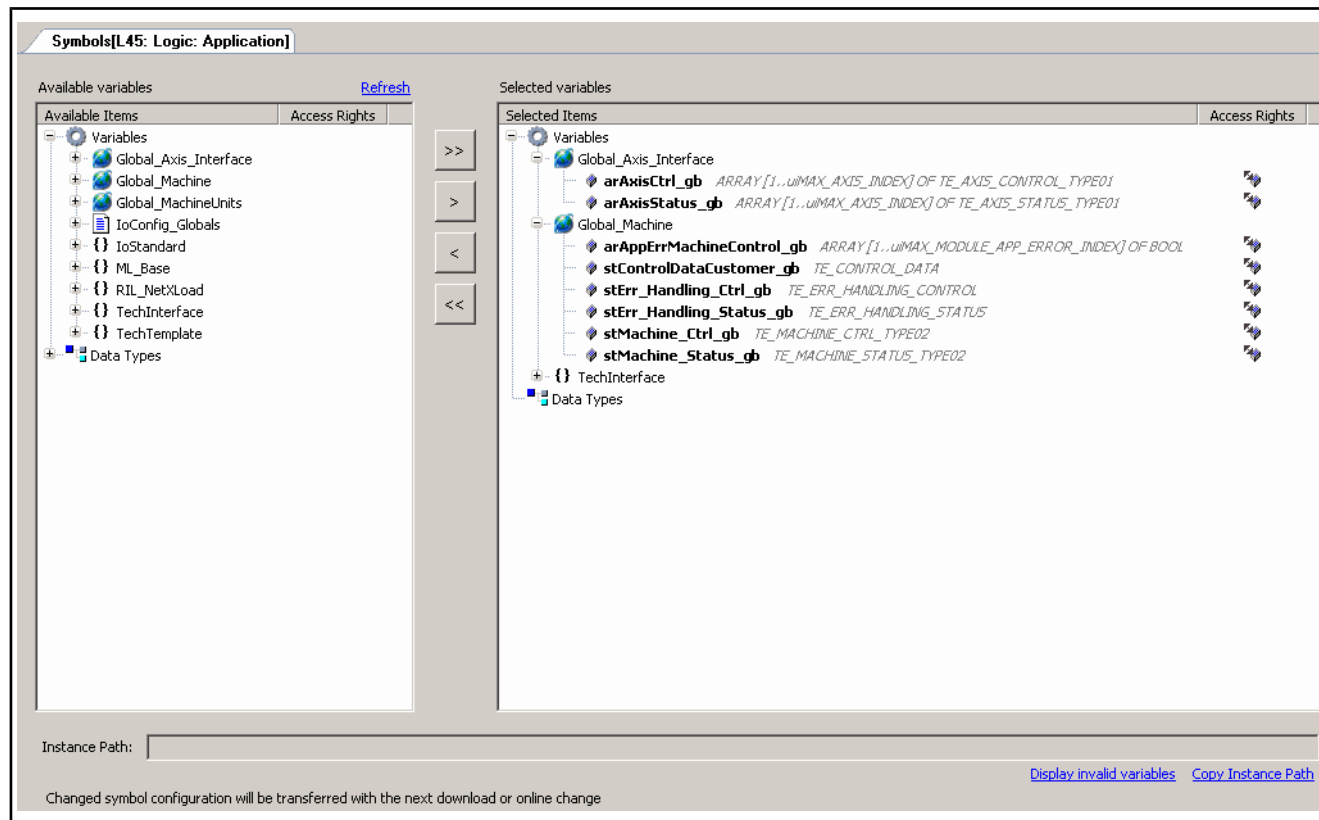


Fig.9-40: Symbol configuration of the predefined WinStudio visualization



Symbol entries for variables from global variable lists that come from a library must be reinserted in the symbol configuration after the respective library version has been modified.

Setting the control's IP address:

The control address is configured in the BR-WS driver properties dialog:

GAT with Decentral State Machine

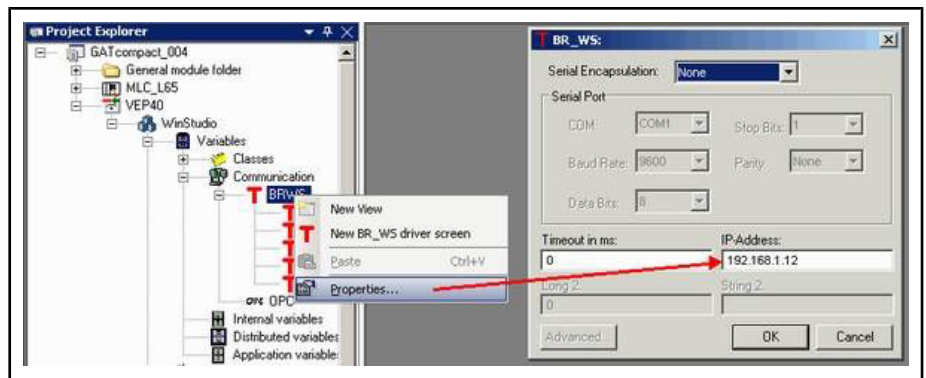


Fig.9-41: Setting the control's IP address

In order to structure the communication, the variables were distributed in various driver sheets:

1. read cyclic fast: -> The values of these variables are to be read cyclically at short intervals (e.g. diagnostic numbers, axis positions, ...)
2. read only once: -> These variables are read once after the visualization is started (e.g. axis names)
3. read after switching phase: These variables are read only once after a phase switch (e.g. if the axis was activated)
4. read - write cyclic: -> The values of these variables are to be both read and written (e.g. machine velocity, axis velocity for Setup mode, ...)
5. write only: -> These variables are only to be written through the visualization (trigger signals for deleting errors, ...)

To extend a driver sheet, the BR-WS browser can search the offline symbol file (*.XML) and the corresponding variable can be selected.

GAT with Decentral State Machine

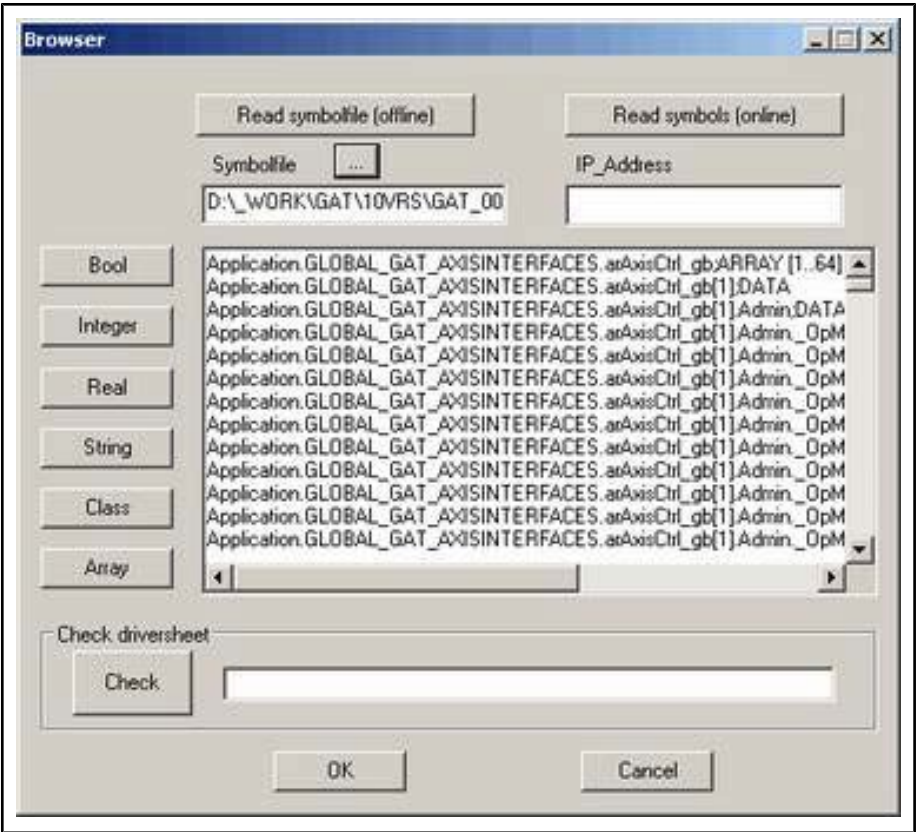


Fig.9-42: Extending the driver sheet

Multiple Language Capability

Multiple language capability for the visualization has been realized with the translation functionality integrated into WinStudio.

The translation file is stored as a *.CSV table, which provides a column for each language.

The table is predefined to include multiple languages, but only the "German" and "English" columns are completely filled out. For the rest of the languages only the drive diagnostics are included.

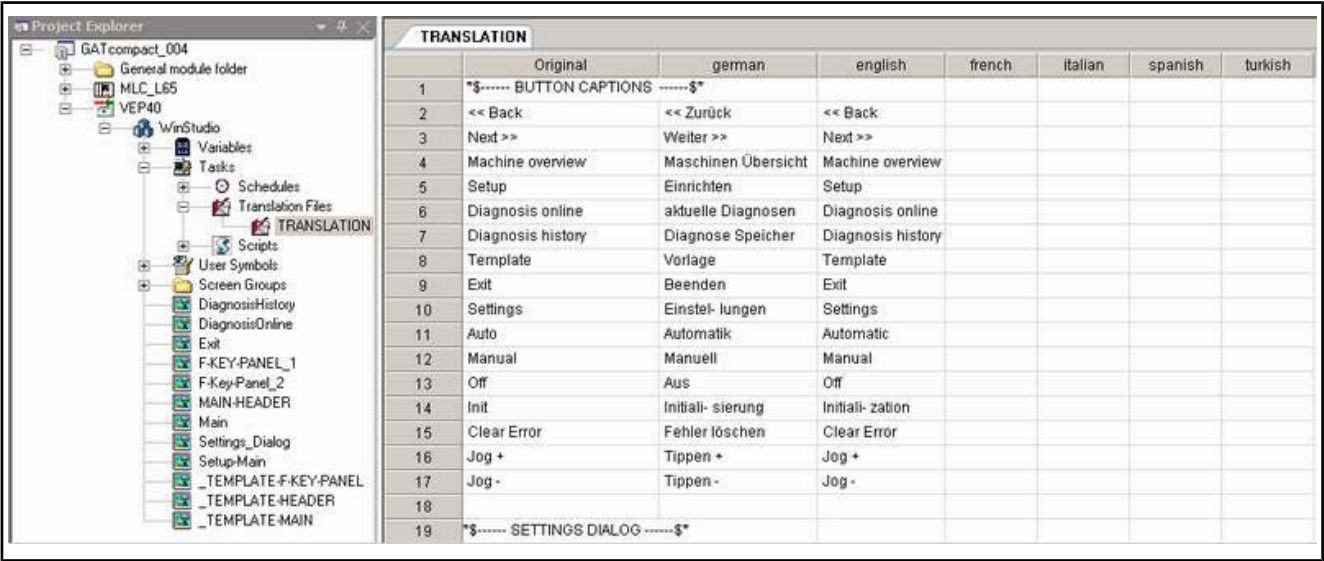


Fig.9-43: Editing the translation file

9.12 Interfaces to GAT with Decentral State Machine

9.12.1 General Information

The GAT interface data are made available in variable lists. Part of the data is made available in libraries.

9.12.2 Machine Machine Handler Data (TE_MachineControl)

TE_MachineControl input data The input data of the TE_MachineControl are combined in the structure **stMachineCtrl_gb** (variable list "Global_Machine") of type **TE_MACHINE_CTRL_TYPE02**.

The following table describes the elements of the input structure.

I/O type	Name	Data type	Description
E	Enable	BOOL	Processing enabled
E	ClearError	BOOL	Positive edge executes local error clearing

E Input (write and read access)

TE_MachineControl output data The output data of the TE_MachineControl are combined in the structure **stMachineStatus_gb** (variable list "Global_Machine") of type **TE_MACHINE_STATUS_TYPE02**.

The following table describes the elements of the output structure.

I/O type	Name	Data type	Description
A	InOperation	BOOL	Processing active
A	OpMode	TE_OPMODES_PREDEFINED_TYPE02	Acknowledgement of the machine's current operation mode
A	Error	BOOL	Local error present
A	ErrorID	ERROR_CODE	Detailed information if DiagNo = 16#F0FB 0000 or 16#F0AA xxxx
A	ErrorIdent	ERROR_STRUCT	Detailed information if DiagNo = 16#F0FB 0000 or 16#F0AA xxxx
A	DiagNo	DWORD	Diagnostic number according to IndraLogic XLC-/IndraMotion MLC specification DiagNo = 16#F0AE xxxx: Application error (e.g. safety door is open) DiagNo = 16#F0FB 0000: FB error DiagNo = 16#F0AA xxxx: GAT internal error
A	DiagMessage	STRING(60)	Diagnostic text of the control / axis

A Output (read access)

9.12.3 Central Error Handling Data (TE_ErrorHandling)

TE_ErrorHandling input data The input data of the TE_MachineControl are combined in the structure **stErr_Handling_Ctrl_gb** (variable list "Global_Machine") of type **TE_ERR_HANDLING_CONTROL**.

The following table describes the elements of the input structure.

GAT with Decentral State Machine

I/O type	Name	Data type	Description
E	Enable	BOOL	Processing enabled
E	ClearAllError	BOOL	Positive edge executes global error clearing

E Input (write and read access)

TE_ErrorHandling output data The output data of the TE_MachineControl are combined in the structure **stErr_Handling_Status_gb** (variable list "Global_Machine") of type **TE_ERR_HANDLING_CONTROL**.

The following table describes the elements of the output structure.

I/O type	Name	Data type	Description
A	InOperation	BOOL	Processing active
A	Error	BOOL	Global error present
A	ErrorID	ERROR_CODE	Detailed information if DiagNo = 16#F0FB 0000 / 16#F0AA xxxx
A	ErrorIdent	ERROR_STRUCT	Detailed information if DiagNo = 16#F0FB 0000 / 16#F0AA xxxx
A	DiagNo	DWORD	Diagnostic number according to IndraLogic XLC-/ IndraMotion MLC specification: DiagNo = 16#F0AE xxxx: Application error (e.g. safety door is open) DiagNo = 16#F0FB 0000: FB error DiagNo = 16#F0AA xxxx: GAT internal error DiagNo = 16#F0B1 xxxx: Local inline error DiagNo = 16#F0B2 xxxx: PROFIBUS error
A	DiagMessage	STRING(60)	Diagnostic text of the control / axis

A Output (read access)

9.12.4 Unit Data

Unit input data Each unit has a control data structure (Ctrl) as an input that is specified by the operation mode handler (TE_MachineControl) and the Error Handler (TE_ErrorHandling). The elements ClearError, ExtError, and ExtErrDiagNo are specified by the Error Handler (TE_ErrorHandling). All other elements are specified by the operation mode handler (TE_MachineControl).

The elements of the input data structure of the unit **TE_UNIT_CONTROL** are listed in the following table:

I/O type	Name	Data type	Description
E	Enable	BOOL	Processing enabled
E	ClearError	BOOL	Error acknowledgement based on a positive edge (from TE_ErrorHandling)
E	OpMode	TE_OPMODES_PREDEFINED_TYPE02	Commanded operation mode
E	DisableErrorReaction	BOOL	Deactivates the error reaction
E	OpModeCoupled	BOOL	TRUE = coupled unit FALSE = uncoupled unit

GAT with Decentral State Machine

I/O type	Name	Data type	Description
E	ExtError	BOOL	External error (from TE_ErrorHandling)
E	ExtErrDiagNo	DWORD	External error number (from TE_ErrorHandling)

E Input (write and read access)

Unit output data

Each unit has a status data structure (Status) as an output that is evaluated by the operation mode handler (TE_MachineControl) and the Error Handler (TE_ErrorHandling). All Error... and Diag... elements are evaluated by the Error Handler (TE_ErrorHandling). All other elements are evaluated by the operation mode handler (TE_MachineControl).

The elements of the output data structure of the unit **TE_UNIT_STATUS** are listed in the following table:

I/O type	Name	Data type	Description
A	InOperation	BOOL	Processing active
A	OpMode	TE_OPMODES_PREDEFINED_TYPE02	Acknowledgement of the current operation mode
A	Error	BOOL	Local error
A	ErrorID	ERROR_CODE	Detailed information if DiagNo = 16#F0FB 0000 / 16#F0AA xxxx
A	ErrorIdent	ERROR_STRUCT	Detailed information if DiagNo = 16#F0FB 0000 / 16#F0AA xxxx
A	DiagNo	DWORD	Diagnostic number according to IndraLogic XLC-/ IndraMotion MLC specification DiagNo = 16#F0AE xxxx: Application error (e.g. safety door is open) DiagNo = 16#F0FB 0000: FB error DiagNo = 16#F0AA xxxx: GAT internal error
A	DiagMessage	STRING(60)	Diagnostic text

A Output (read access)

9.12.5 Axis Data

I/O type	Name	Data type	Description
E	arAxisCtrl_gb (Global_Axis_Interfaces)	ARRAY OF TE_AXIS_CONTROL_TYPE01	Commands for the axes
A	arAxisStatus_gb (Global_Axis_Interfaces)	ARRAY OF TE_AXIS_STATUS_TYPE01	Feedback from the axes
I/O	AxisData (GVL_Base from the ML_Base library)	ARRAY OF ML_AXISDATA_SM	Firmware structure with axis inputs and outputs

E Input (write and read access)

A Output (read access)

I/O Structure with inputs and outputs



Elements for arAxisCtrl_gb are overwritten by the units and for this reason they can no longer be used externally.

GAT with Decentral State Machine

9.12.6 Control Data

I/O type	Name	Data type	Description
E	ImcCtrl (Global_ImcInterface from the ML_TechInterface library)	MB_IMC_CONTROL_TYPE01	Commands for the control
A	ImcStatus (Global_ImcInterface from the ML_TechInterface library)	MB_IMC_STATUS_TYPE01	Feedback from the control

E Input (write and read access)

A Output (read access)



Elements for ImcCtrl_gb are described in TE_MachineControl and for this reason they can no longer be used externally.

9.13 How To: Typical User Activities

9.13.1 Access to Axis Data

For access to axis data, in every unit the **Axes** structure is defined as the central access point for all of the axes assigned to the unit. All relevant data for an axis can be addressed using the Axes structure.

The structure is designed such that the axis data is accessed using the name of the axis. This generates program code that is easily readable.

The following data are available below "Axes":

- Axes.<Name>.AxisData => Axis data structure: Status data for the firmware and access to the axis data container
- Axes.<Name>.Ctrl => Control structure for the AxisInterface
- Axes.<Name>.Status => Status structure for the AxisInterface
- Axes.<Name>.DV => Access to direct variables (function variables)
- Axes.<Name>.AxisRef => Axis reference structure for FB calls (e.g. MB_ReadParameter)

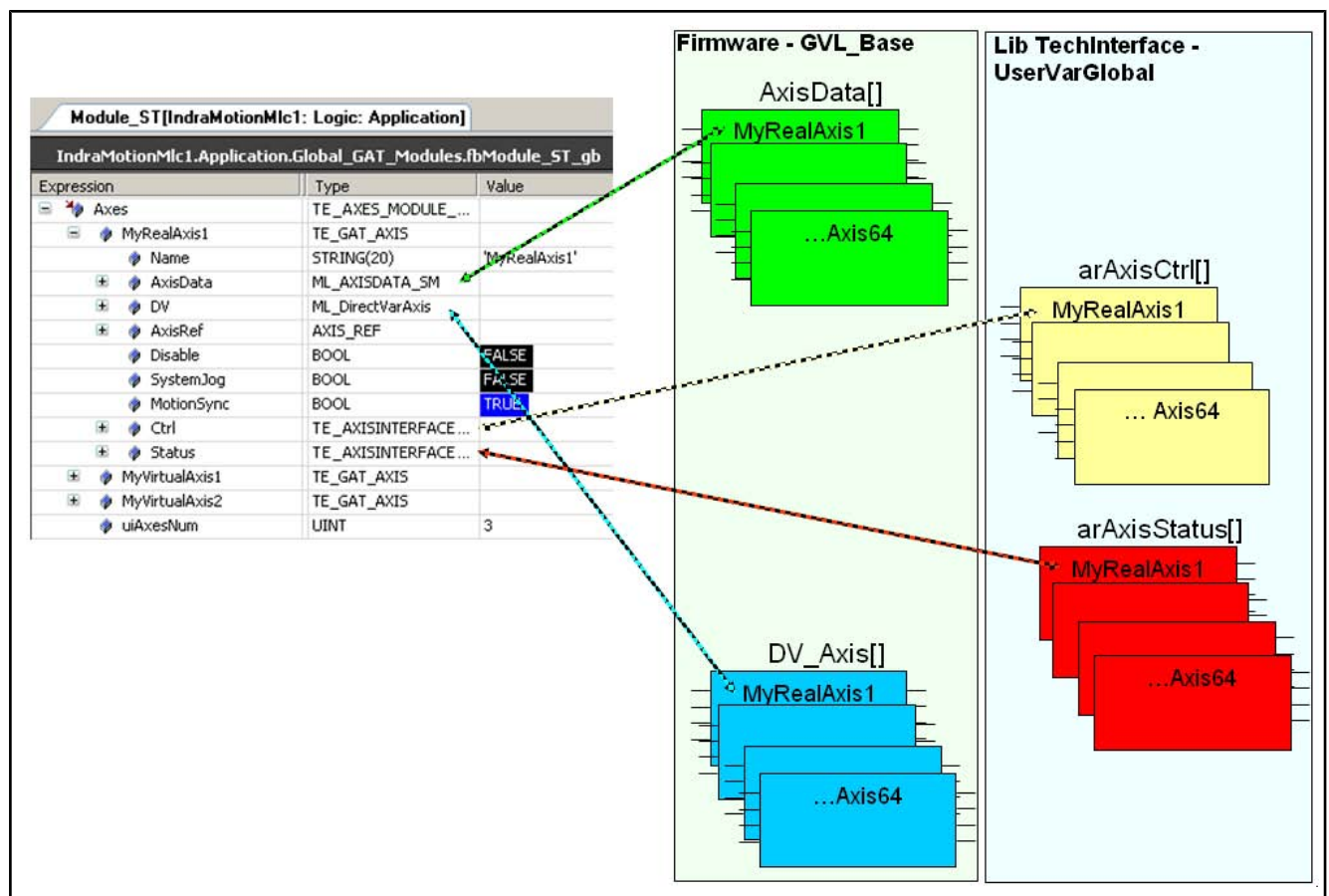


Fig.9-44: Overview / axis data access

9.13.2 Adding an Axis

The following steps are required for adding an axis:

- Add the axis into the IndraWorks project ("Drag to the SERCOS node...")
- When using the SystemOverview_All_Axis visualization: add a frame for the new axis and set the reference to the new axis in the properties (see also [chapter 7.8.1 "Internal Visualization for Assistance during Start-up"](#) on page 142).
- Add the unit axis using the GAT wizard (see the figure below)

GAT with Decentral State Machine

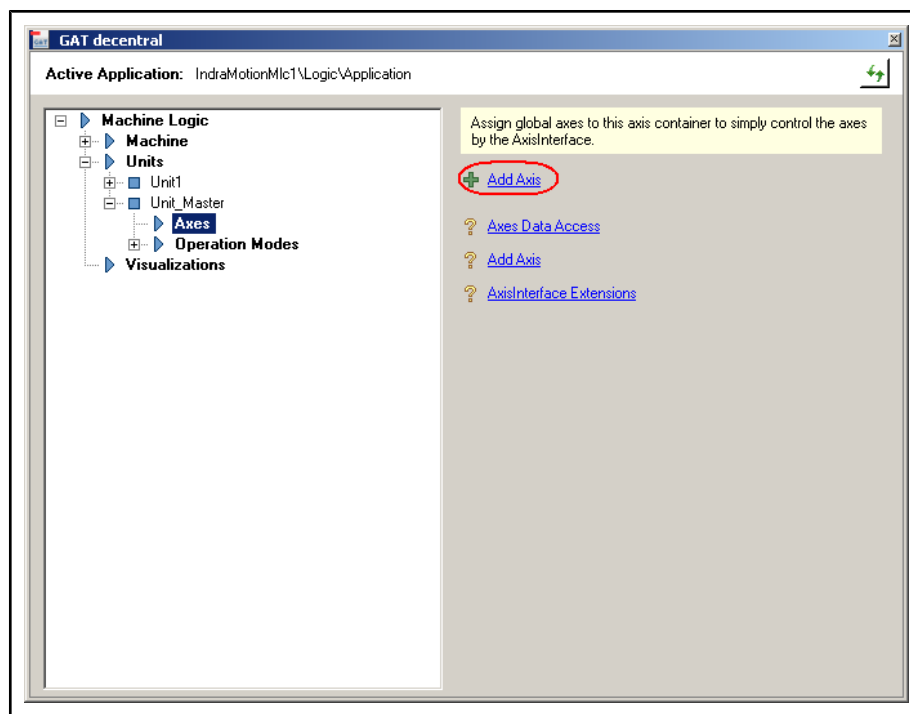


Fig.9-45: Adding an axis using the GAT wizard

9.13.3 Removing/renaming an Axis



The wizard changes only the name in the axis declaration. If the axis is already being used in the implementation or internal visualization, the new axis name must be changed at these positions using "Find/Replace".

The following steps are required for removing/renaming an axis:

- Removing an axis from/renaming an axis in the IndraWorks project
- When using the SystemOverview_All_Axis visualization: remove the frame for the deleted axis and change the reference to the axis in the properties.
- Removing/renaming the axis with the GAT wizard

9.13.4 Adding a Primary Operation Mode

The GAT wizard makes it easy to add primary operation modes to the machine. The wizard can also be used to delete or rename the operation modes.

GAT with Decentral State Machine

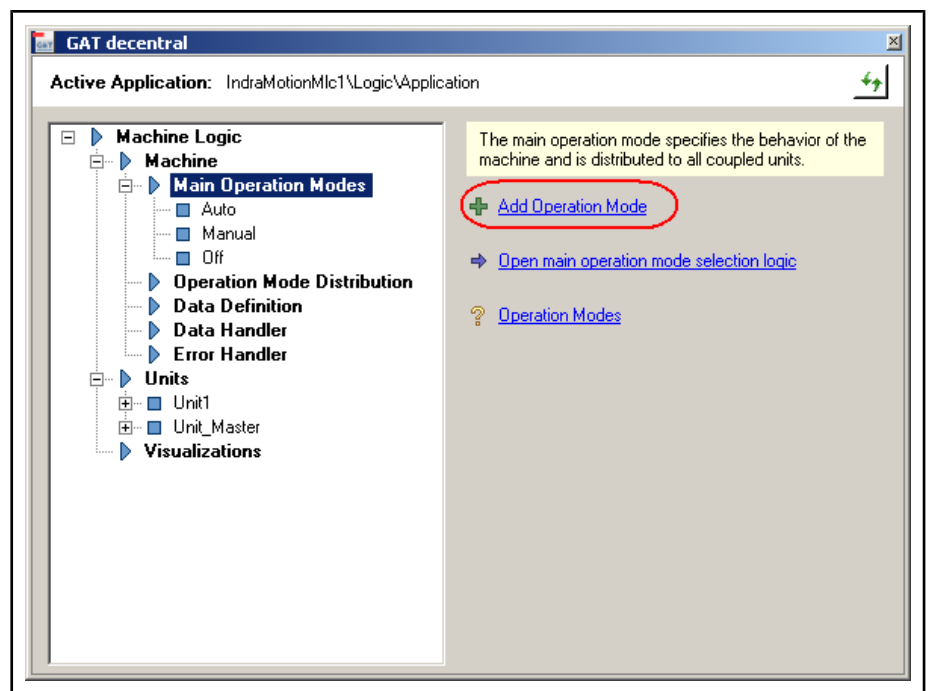


Fig.9-46: Adding an operation mode with the GAT wizard

9.13.5 Adding an New Unit

The GAT wizard makes it easy to add units to the machine. The wizard can also be used to delete or rename the units.

In the new coupled unit, corresponding branches are created in the operation mode SFC for all existing primary operation modes.

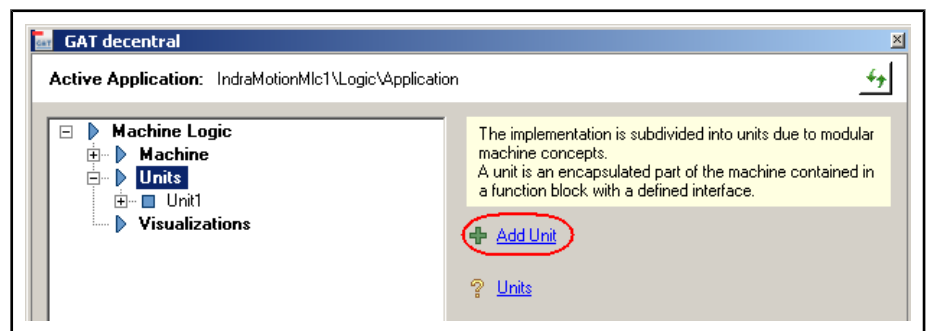


Fig.9-47: Adding a unit with the GAT wizard

9.13.6 Adding Another Instance of a Unit

The GAT wizard makes it easy to add further instances of existing units to the machine.

The interfaces remain the same, but other global axes (an axes) and I/O data (an UserData) must be assigned.

GAT with Decentral State Machine

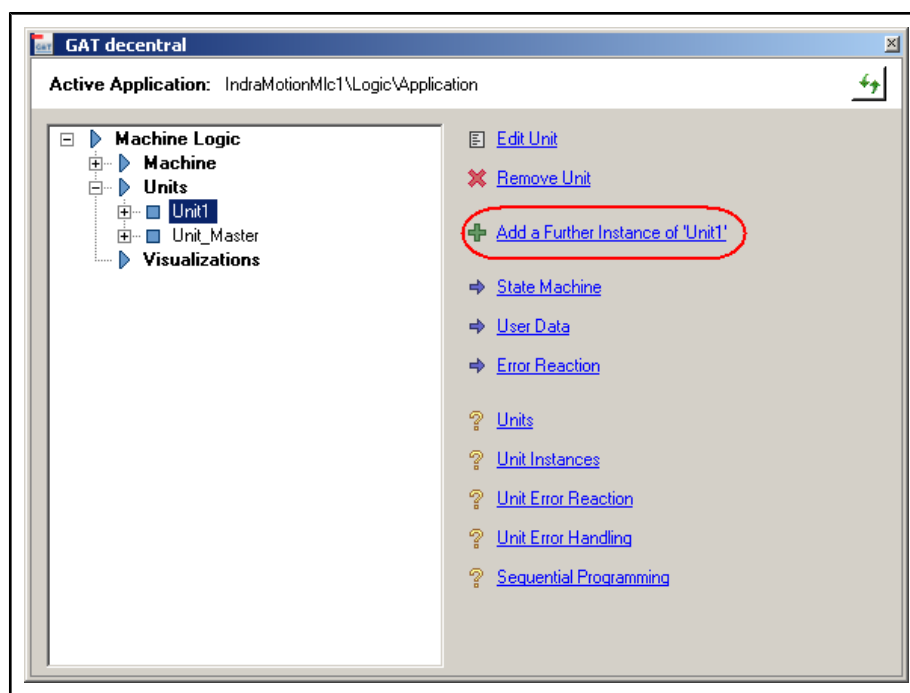


Fig.9-48: Adding another instance with the GAT wizard

9.13.7 Deactivating Units

In the action **actInitColdStart** of the FB **TE_MachineControl**, the instruction "mEnableUnits(Enable := TRUE);" is used to activate all units using the Enable input (see the following figure, highlight 1).

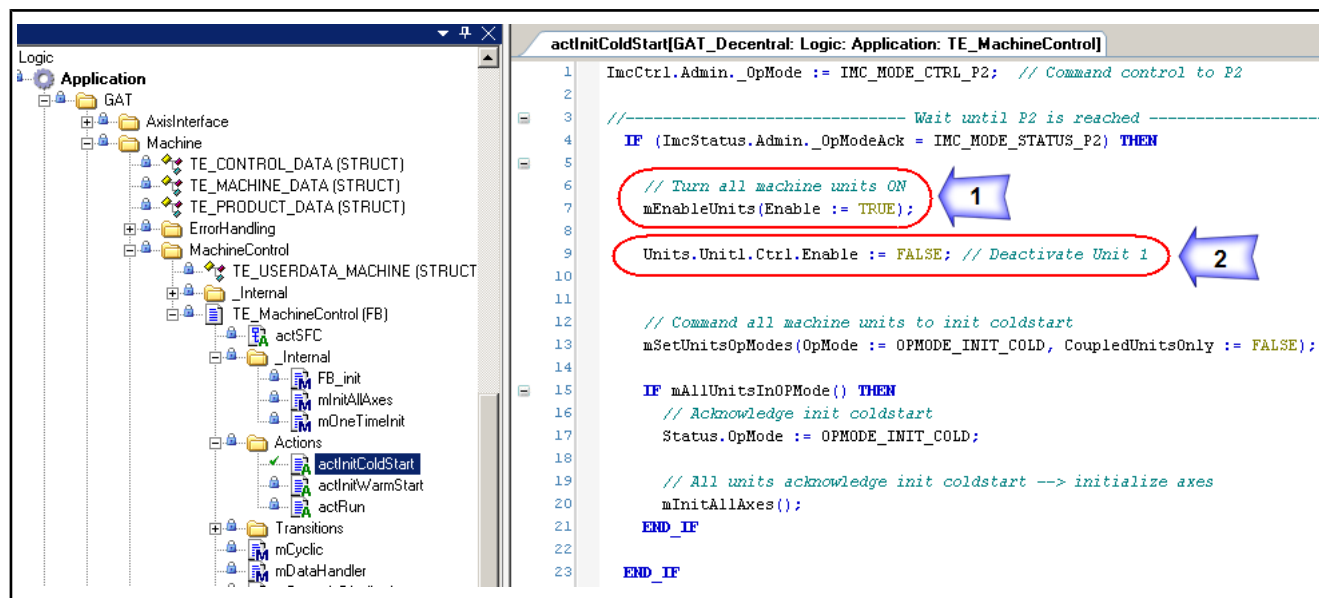


Fig.9-49: Example "Unit1 is deactivated"

Following the instruction listed above, individual units can be deactivated in a targeted manner using the instruction "Units.<UnitName>.Ctrl.Enable := FALSE" (see the figure above, highlight 2). The program code of deactivated units is excluded from processing. The unit can be deactivated at any time. The Status output "InOperation = FALSE" shows that this unit is deactivated.

9.13.8 Reusing Units

The GAT is designed such that developed units can also be reused in other projects built using the GAT.

To do this, select the folder of the desired unit and export the related data by using the context menu (see the following figure).

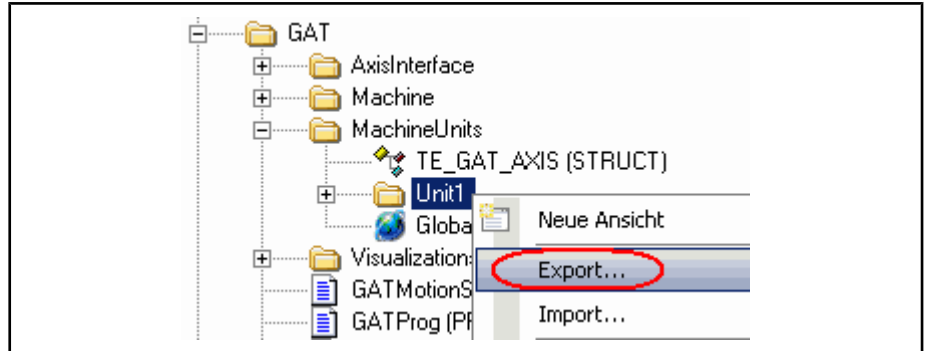


Fig. 9-50: Exporting a unit

Select all of the elements for the export in the following dialog.

Import the file generated during the export into the new project. To do this, highlight the "MachineUnits" folder and use the "Import" functionality in the related context menu (see the following figure).

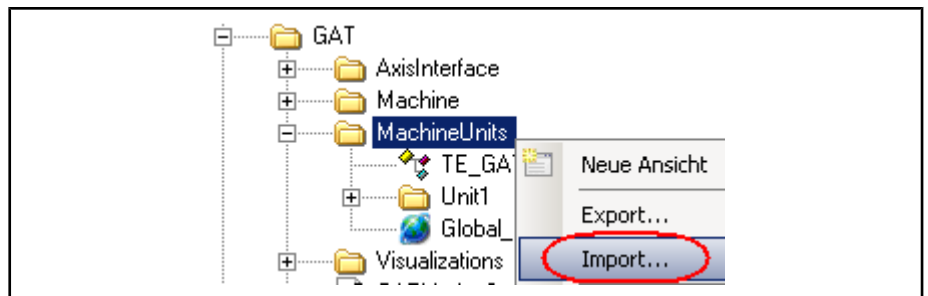


Fig. 9-51: Importing a unit

In the data structure **TE_MACHINE_UNITS**, the imported unit must be accepted as an element and the element **uiMaxUnitNum** must be increased by 1. The following figure shows an example of a unit named "MyImportUnit".

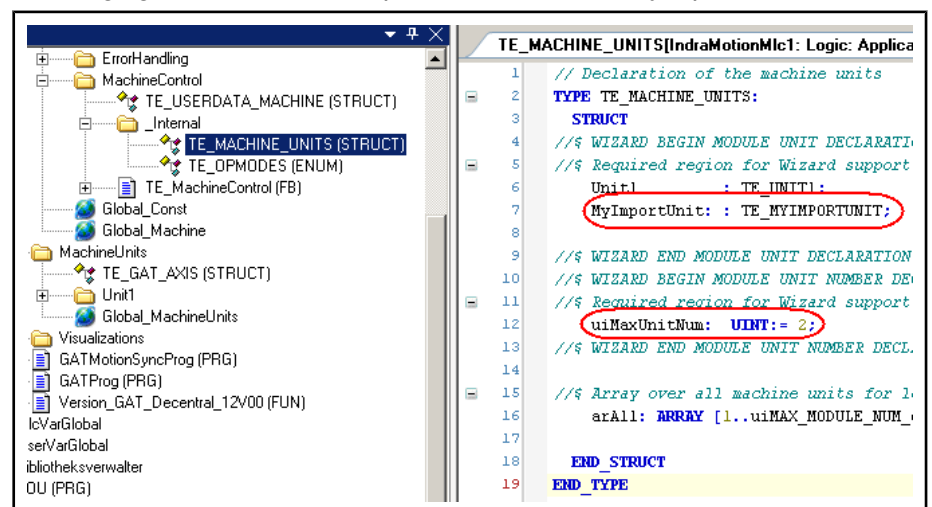


Fig. 9-52: Example adjustment of the data structure **TE_MACHINE_UNITS** after unit import

GAT with Decentral State Machine

The call of the imported unit instance must be added into the program "GAT-Prog" or "GATMotionSyncProg".

The unit references for the new instance are to be added in the method "mOn-eTimeInit" of the FB TE_MachineControl. The assignment for coupled units (Units.Unit1.Ctrl.OpModeCoupled := TRUE; // Coupled unit) and uncoupled units (Units.Unit1.Ctrl.OpModeCoupled := FALSE; // Uncoupled unit) must also be entered in this area.

In addition, the reference for the element of the UserData structure must also be added in the ELSE branch of this method (Units.<UnitName>.UserData.stControlDataActive REF= stControlDataActive_gb;).

The unit references for the new instance are to be added in the method "mOn-eTimeInit" of the FB TE_ErrorHandling.

For an uncoupled unit, the part for the uncoupled unit must still be added to the method "mOpModeDistribution".



After the import, check the coupled unit to see if all primary operation modes of the machine are also contained in the SFC of the unit. If this is not the case, the missing operation modes must be added to the SFC branch.

9.13.9 Using Global and Address Variables in Units

Because a unit can be used more than once in an application (instantiation) and because it can be reused in another application, there may be no direct access to a global or address variable in a unit.

All global and address variables that are to be used in the unit must be declared as interface variables in the unit data container "**UserData**". The element within the unit is accessed using **UserData.<Element>**.

9.13.10 AxisInterface Extensions

The GAT allows extensions of the AxisInterface structures. Additional substructures can be added and the existing substructures extended.

As an example, the following structure elements are programmed as user extensions in the AxisCtrl structure.

- SetupMode: Additional substructure in the AxisCtrl structure
 - Enable: Enable Setup mode
 - JogPlus: Jog +
 - JogMinus: Jog –
 - Vel: Jog / reference travel velocity
 - Accel: Jog/Reference travel deceleration/acceleration
 - Homing: Activates ModeHoming|SetAbsRef|ModesPosAbs depending on axis type/configuration
- SyncMode: Extension of the existing substructure in the AxisCtrl structure
 - AngleOffset: Phase offset start of table -> parameter A-0-2730 is described with this value
 - PhaseShift: Offset relative to the master axis before the gear (PhaseOffset: Offset relative to the master axis after the gear)
 - PhaseShiftVel: Velocity for adjusting the PhaseShift
 - PhaseShiftAcc: Acceleration for adjusting the PhaseShift
 - SyncVelocity: Phase shift velocity -> parameter A-0-2790 is described with this value

GAT with Decentral State Machine

- SyncAcceleration: Phase shift acceleration -> this value is written on parameter A-0-2791

As an example, the following structure elements are programmed as user extensions in the AxisStatus structure:

- SetupMode: Additional substructure in the AxisStatus structure
 - EnableAck: Setup mode is active
 - Homingack: Homing is finished.
- SyncMode: Extension of the existing substructure in the AxisStatus structure
 - AngleOffsetDone: Transfer of phase offset start of table -> parameter A-0-2730 was described
 - PhasingMasterDone: PhaseShift was completed.
 - SyncVelocityDone: Transfer of synchronization velocity -> parameter A-0-2790 was transferred.
 - SyncAccelerationDone: Transfer of synchronization acceleration -> parameter A-0-2791 was transferred.

The code for these extensions can be found in the action of the TE_AxisInterface() block in the GAT/AxisInterface folder. The related structures can be found in the GAT/AxisInterface/Type/Control and GAT/AxisInterface/Type/Status folders.

9.13.11 Generating Your Own Program Framework Based on the GAT

An application developed based on the GAT can also be used in turn as a template (program framework) for developing another, more complex application. In this way, users can provide their own program frameworks for various machine types, for example. However, these frameworks differ from the standard GAT due to more advanced developments (e.g. additional operation modes and units). The GAT wizard can then be used to work with the more advanced program frameworks.

To generate your own template based on an application developed with the GAT, use the following procedure:

1. Export the entire GAT folder; leave all check boxes selected (generate an *.iwx file). In the Export dialog is a "Description" input field. The text entered here can be displayed in the Import function of the GAT wizard.

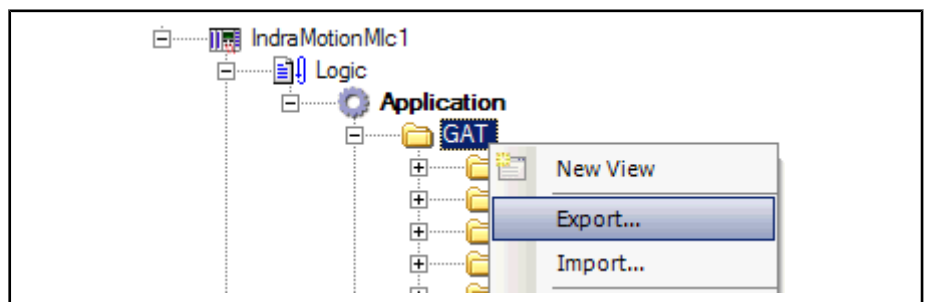


Fig.9-53: Exporting the GAT folder

2. The *.iwx file generated in step 1 contains the adapted / extended template and can now be used with the GAT wizard as follows for creating new projects:

GAT with Decentral State Machine

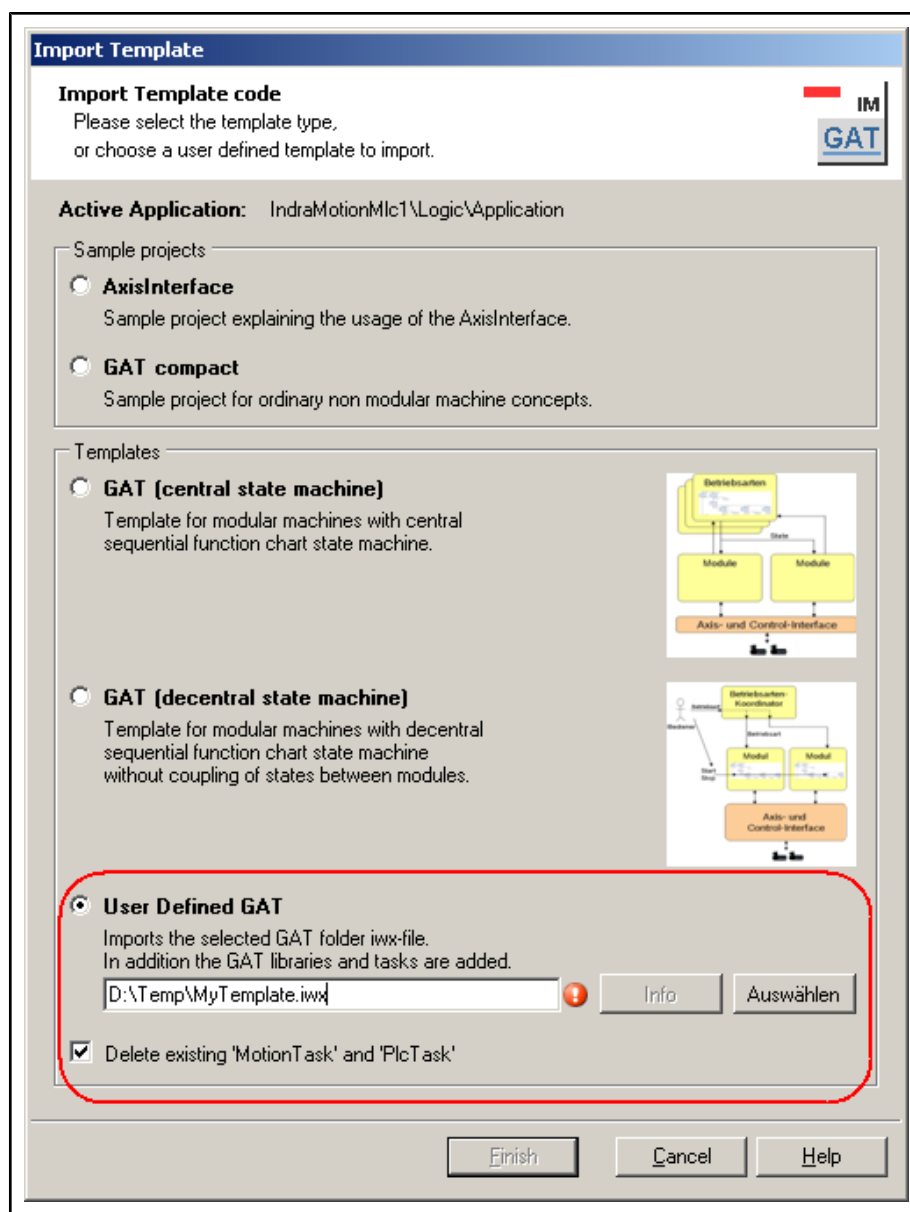


Fig.9-54: Importing a user-defined template

9.14 GAT - Decentral State Machine, Brief Overview

9.14.1 Structure Overview

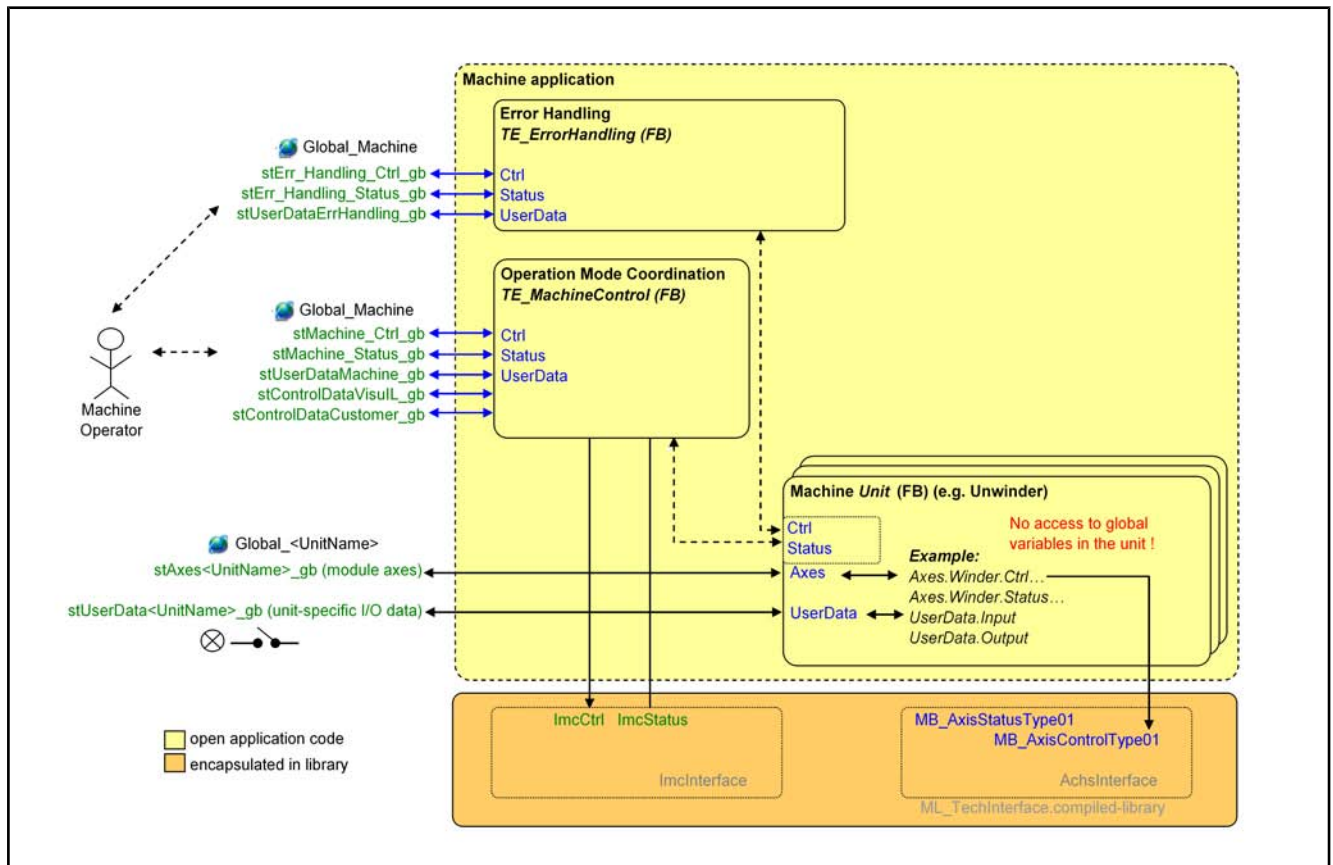


Fig.9-55: Structure overview, GAT with decentral state machine

9.14.2 Overview of the Machine's Most Important Global Variables and Error Handling

The following table shows the most important global variables of the machine and error handling.

The variables are contained in the global variable list "GAT/Machine/Global_Machine".

Variable	Description	Access
stControlDataVisuL_gb	Central machine control (e.g. operation mode selection), command using integrated commissioning visualization	RD/WR
stControlDataCustomer_gb	Central machine control (e.g. operation mode selection), command using external visualization	RD/WR
stControlDataActive_gb	Effective data of the central machine control (e.g. operation mode selection)	RD/WR
stMachineDataEdit_gb	Prepared machine-specific data	RD/WR
stMachineDataActive_gb	Effective machine-specific data	RD
stProductDataEdit_gb	Prepared product-specific data	RD/WR
stProductDataActive_gb_gb	Effective product-specific data	RD

GAT with Decentral State Machine

Variable	Description	Access
stMachine1Status_gb	Status of the machine (e.g. output of the current operation mode)	RD
stUserDataMachine_gb	User-specific I/O data of the machine	RD/WR
arAppErrMachineControl_gb	User-specific error flags of the machine	RD
stErr_Handling_Ctrl_gb	Central error handling, control structure (e.g. for global error clearing)	RD/WR
stErr_Handling_Status_gb	Central error handling, status structure (e.g. output of the central error bit)	RD
stUserDataErrHandling_gb	User-specific I/O data of the error handling	RD/WR

RD Read:
WR Write

9.14.3 Overview of the Most Important Global Variables of the Units

The following table shows the most important global variables of the units.

The variables are contained in the global variable list "GAT/MachineUnits/Global_<Unitname>".

Variable	Description	Access
fb<Unitname>_gb.Ctrl	Unit control structure (e.g. operation mode command)	RD/WR
fb<Unitname>_gb.Status	Unit status structure (e.g. operation mode acknowledgement)	RD
stUserData<Unitname>_gb	Unit-specific I/O	RD/WR
arAppErr<Unitname>_gb	Array with user-specific error flags of the unit	RD
stAxes<Unitname>_gb	Container with unit axes	RD

RD Read:
WR Write

9.14.4 Overview of the Most Important Global Variables of the Axes

The following table shows the most important global variables of the axes.

The variables are contained in the global variable list "GAT/ AxisInterface/Global_GAT_AxisInterfaces".

Variable	Description	Access
arAxisCtrl_gb	Array with the AxisInterface command data	RD/WR
arAxisStatus_gb	Array with the AxisInterface status data	RD

RD Read:
WR Write

9.14.5 Overview of the Methods and Actions of TE_MachineControl

The following table shows the most important methods and actions of the FB TE_MachineControl.

Action/method	Call	Description / example
actOPModeSFC	Cyclical on an ongoing basis	Operation mode SFC
mCyclic	Cyclical on an ongoing basis	Cross unit logic

GAT with Decentral State Machine

Action/method	Call	Description / example
mDataHandler	Cyclical on an ongoing basis	Data handling/switching between integrated commissioning visualization and external visualization, machine and product data handling
mOpModeDistrubution	Cyclical on an ongoing basis	Operation mode distribution to the units
mOpModeSet	Cyclical on an ongoing basis	Determining the primary operation mode
mOneTimeInit	For one cycle when the GAT is initialized. Execution before cold start	Initialize settings for "SkipColdstartAtStartup" Assign unit axes to the global IndraWorks axes Initialize references
mInitAllAxes	When exiting InitColdStart and in InitWarmStart	Initialize AxisInterface
mErrorHandling	Cyclical on an ongoing basis	Scan local sources of error, clear local error
FB_init	Once before starting/resetting a PLC program	Internal system use, may not be modified

9.14.6 Overview of the Methods and Actions of TE_ErrorHandling

The following table shows the most important methods and actions of the FB TE_ErrorHandling.

Action/method	Call	Description / example
mScanError	Cyclical on an ongoing basis	Scan sources of error
mDistributeError	Cyclical on an ongoing basis	Error distribution to units
mClearError	Cyclical on an ongoing basis	Central clear error
mOneTimeInit	For one cycle when the GAT is initialized. Execution before cold start	Initialize settings for "SkipColdstartAtStartup" Assign unit axes to the global IndraWorks axes Initialize references
mGetFieldbusError	Cyclical on an ongoing basis	Scan Profibus Slave errors
FB_init	Once before starting/resetting a PLC program	Internal system use, may not be modified

9.14.7 Overview of the Methods and Actions of the Units

The following table shows the most important methods and actions of the units.

Action/method	Call	Description / example
actOPModeSFC	Cyclical on an ongoing basis	Operation mode SFC
mErrorHandling	Cyclical on an ongoing basis	Scan local sources of error, clear local error
mRefreshAxes	Cyclical on an ongoing basis	Update AxisInterface data
mOpModeAcknowledge	Cyclical on an ongoing basis	Operation mode acknowledgement
FB_init	Once before starting/resetting a PLC program	Internal system use, may not be modified

GAT with Decentral State Machine

9.14.8 Access to Axes within the Unit

Always use **Axes.<axisname>**

Example:

Program:

```
// Command axis in velocity control mode
Axes.MyAxis1.Ctrl.Admin._OpMode.en := ModeVel;

// If axis has reached the velocity
IF (Axes.MyAxis1.Status.PtrAxisData^.Axis_InVelocity = TRUE) THEN;
```

9.14.9 Access to I/O Data within the Unit

Always use **UserData**. **!! No direct access to global variables !!**

Example:

Program:

```
IF (UserData.bInput1=TRUE) THEN // If "bInput1" is set...
UserData.bOutput1:=TRUE; // Set output "bOutput1"
END_IF
```

10 Service and Support

Our service helpdesk at our headquarters in Lohr, Germany and our worldwide service will assist you with all kinds of enquiries. You can reach us **around the clock - even on weekend and on holidays**.

	Helpdesk	Service Hotline Worldwide
Phone	+49 (0) 9352 40 50 60	Outwith Germany please contact our sales/service office in your area first.
Fax	+49 (0) 9352 40 49 41	
E-mail	service.svc@boschrexroth.de	For hotline numbers refer to the sales office addresses on the Internet.
Internet	http://www.boschrexroth.com You will also find additional notes regarding service, maintenance (e.g. delivery addresses) and training.	

Preparing Information

For quick and efficient help please have the following information ready:

- Detailed description of the fault and the circumstances
- Information on the type plate of the affected products, especially type codes and serial numbers
- Your phone, fax numbers and e-mail address so we can contact you in case of questions.

Index

Symbols

_Init..... 180

A

About this documentation..... 9
 Information representation 13
 Validity of the documentation 9
 actAssignIO..... 231
 actAssignIOBackground..... 231
 actAssignIOMotionSync..... 231
 actAutoProducing..... 231
 actAutoStopped..... 231
 actDisable..... 220, 232
 actError..... 195, 231
 actInitColdstart..... 231
 actInitWarmstart..... 231
 actManualReady..... 231
 actOpModeSFC..... 245
 Adding an axis to the system overview..... 77
 Address variables in the unit..... 280
 Application-specific watchdog..... 39
 Appropriate use
 Use cases 17
 Appropriate Use
 Introduction 17
 arAppErrMachine1_gb..... 229
 arAxisCtrl_gb..... 209
 arAxisStatus_gb..... 209
 arMachine1ModuleDiagNo_gb..... 229
 Automating Applications..... 33
 AXIS_GAT..... 198
 Axis_Ref..... 104
 AxisInterface
 Special features of the IndraMotion MLD ... 117
 AxisInterface extensions, GAT..... 224
 AxisInterface extensions, GAT, decentral..... 280
 AxisInterface function blocks..... 57
 AxisInterface operation modes..... 61
 AxisInterface structure elements
 Axis overview visualization 79
 ImcInterface control 87
 ImcInterface status 87
 Positioning mode visualization 80
 Synchronization mode visualization 83
 Synchronization mode with profiles 85
 Velocity mode visualization 82
 AxisInterface structures..... 93
 Axis_Mode 103
 MB_AXIS_ADJUST_DATA 105, 110
 MB_AXIS_ADJUST_STATUS 95, 98
 MB_AXIS_ADMINISTRATION 101
 MB_AXIS_CONFIG 96
 MB_AXIS_CONTROL_TYPE01 structure .. 100
 MB_AXIS_DIAGNOSIS 97
 MB_AXIS_MODE 96

A

...AxisInterface structures
 MB_AXIS_POS_STATUS 98
 MB_AXIS_POSITIONING 106
 MB_AXIS_STATUS_TYPE01 94
 MB_AXIS_SYNC_STATUS 100
 MB_AXIS_VEL_STATUS 99
 AxisInterface Structures
 MB_AXIS_CONFIG 96
 MB_AXIS_CONTROL_TYPE01 100
 MB_AXIS_PROFILE 113
 MB_AXIS_SYNCHRONISATION 110
 MB_AXIS_VELOCITY 108
 AxisInterface visualizations..... 76

B

bStart_gb..... 229
 bStop_gb..... 229

C

Changing the operation mode, GAT..... 183
 ClearError..... 197
 Configuration..... 53
 Converting AxisInterface projects..... 55
 Cyclical processing, general information..... 34

D

DisableErrorHandling..... 195
 DisableLocked..... 220
 DisableTransient..... 220
 DriveInterruptReaction..... 73
 Drive system..... 19

E

Electric drive system..... 19
 EnableCyclicScanning..... 49
 Error acknowledgement / restart after an error... 43
 Error detection..... 40
 Error diagnostics..... 42
 Error management, general information..... 40
 Error reaction..... 41
 Error reaction, GAT..... 195, 256

F

First steps, AxisInterface..... 75

G

GATBackgroundProg..... 193
 GATBackgroundProgTask..... 193
 GAT^{compact}
 Access to axes within the machine 162
 Access to axis data 154

Index

G

...GATcompact

Adding a BackgroundTask	137
Adding a MotionTask	135
Adding an axis	155
Adding an operation mode	158
Axis data	153
AxisInterface extensions	159
Clearing errors	142
Control data	153
Cyclical processing	135
Data for error handling	152
Data for the state machine	152
Error detection and diagnostics	138
Error handling	138
Error reaction	140
External visualization with separate HMI (WinStudio visualization)	144
First steps	122
Initialization	130
Interfaces to GAT	152
Operation modes	133
Overview of the machine's methods and actions	162
Overview of the machine's most important global variables	161
Overview of the most important global var- iables of the axes	162
Processing in MotionTask	159
Removing/renaming an axis	156
Structure overview	161
Task configuration	135
Task handling	135
Visualization	142

GAT compact

First steps, creating a simple motion ap- plication	122
First steps, preparation	122
GAT functional principle.....	180, 243
GAT initialization.....	180, 244
GATMotionSyncProg.....	192
GATMotionSyncTask.....	192
GAT operation modes.....	181
GATProg.....	216
GAT with central state machine.....	165
Access to axes within the module	232
Access to axis data	210
Access to I/O data within the module	232
Adding an axis	211
Adding a new module	217
Adding an operation mode	214
Adding another instance of a module	218
Adding states and transitions	216
Axis data	209
AxisInterface extensions	224
Control data	210
Data for error handling	208

G

...GAT with central state machine

Deactivating a module	219
External visualization with separate HMI (WinStudio visualization)	199
First steps	167
First steps, creating a simple motion ap- plication	167
First steps, preparation	167
Functional principle	180
GAT, brief overview, overview of the ma- chine's methods and actions	231
GAT error handling	194
GAT structure overview	229
Generating your own program framework based on the GAT	225
How to: typical user activities	210
Initialization	180
Interfaces to GAT	207
Internal visualization for commissioning support	198
Machine data	207
Module data	208
Modules	186
Operation modes	181
Overview	165
Overview of the machine's methods and actions	230
Overview of the machine's most important global variables	229
Overview of the module's most important global variables	230
Overview of the most important global var- iables of the axes	230
PackML V3.0 program framework based on GAT	226
Processing in MotionTask	224
Removing/renaming an axis	212
Reusing modules	220
Selecting the desired operation mode after clear error	215
Superordinate logic for operation mode specification	216
Switching off GAT error reaction for F0 er- rors	228
Task handling	192
Time sensitive, application-specific error reaction	228
Updating from version 10/11 to 12	228
Using global and address variables in modules	222
Visualization	198, 258
GAT with decentral state machine.....	233
Access to axes within the unit	286
Access to axis data	274
Access to I/O data within the unit	286
Adding a BackgroundTask	253

G

...GAT with decentral state machine	
Adding an axis	275
Adding a new unit	277
Adding another instance of a unit	277
Adding a primary operation mode	276
Axis data	273
AxisInterface extensions	280
Basic idea	233
Central error handling data (TE_ErrorHandling)	271
Clearing errors	257
Control data	274
Deactivating units	278
Error detection and diagnostics	254
Error handling	254
Error reaction	256
External visualization with separate HMI (WinStudio visualization)	261
First steps	235
First steps, creating a simple motion application	235
First steps, preparation	235
Functional principle	243
GAT structure overview	283
Generating your own program framework based on the GAT	281
How to: typical user activities	274
Initialization	244
Interfaces to GAT with decentral state machine	271
Internal visualization for commissioning support	259
Machine data handling	258
Machine Machine Handler data (TE_MachineControl)	271
Operation modes	245
Overview of the machine's most important global variables and error handling	283
Overview of the methods and actions of TE_ErrorHandling	285
Overview of the methods and actions of TE_MachineControl	284
Overview of the methods and actions of the units	285
Overview of the most important global variables of the axes	284
Overview of the most important global variables of the units	284
Product and machine data handling	257
Product data handling	258
Removing/renaming an axis	276
Requirements for use	235
Reusing units	279
Switching the visualization	258
Task management	253
Unit data	272

G

...GAT with decentral state machine	
Units	245
Using global and address variables in units	280
General information on the Automatic operation mode.....	34
General information on the Manual operation mode.....	34
Generation and displaying diagnostic messages.....	34
Global_GAT_MachineControl.....	209
Global_ImcInterface.....	74
Global_MB_AxisInterfaceVars.....	74
Global variables.....	74
Global variables in the unit.....	280

I

ImcCtrl.....	210
IMC interface structures.....	115
ImcStatus.....	210
Improper use.....	18
Inappropriate Use	
Consequences, Exclusion of Liability	17
Initialization.....	33
Initial TechInterface configuration.....	53
Interfaces to GAT.....	207
Interfaces to GAT, decentral.....	271
Introduction.....	31

L

Library	
ML_TechInterface.library	47

M

MB_AXIS_ADJUST_DATA.....	105, 110
MB_AXIS_ADJUST_STATUS.....	95, 98
MB_AXIS_ADMINISTRATION.....	101
MB_AXIS_CONFIG.....	96
MB_AXIS_CONTROL_TYPE01.....	100
MB_AXIS_DIAGNOSIS.....	97
MB_AXIS_MODE.....	96
MB_AXIS_POS_STATUS.....	98
MB_AXIS_POSITIONING.....	106
MB_AXIS_PROFILE.....	113
MB_AXIS_STATUS_TYPE01.....	94
MB_AXIS_SYNC_STATUS.....	100
MB_AXIS_SYNCHRONISATION.....	110
MB_AXIS_VEL_STATUS.....	99
MB_AXIS_VELOCITY.....	108
MB_AxisInitType01.....	57
MB_AxisInterfaceType01.....	59
MB_GetAxisInterfaceIndex.....	119
MB_IMC_Control_Type01.....	117
MB_IMC_Status_Type01.....	116
MB_Phasing.....	88

Index

M

MB_PhasingSlave.....	89
mBackground.....	194, 230, 231
mClearError.....	235
mCyclic.....	230, 231
mDataHandler.....	257
mDistributeError.....	235
Measuring cycle times.....	36
mEntryOpMode.....	185, 230, 232
mErrorHandling.....	254
mExitOpMode.....	184, 230, 232
mFB_AxesInit.....	232
mForceOpMode.....	185, 232
mGetAxis.....	211, 232
mIsCriticalError.....	196, 231
mIsMyAxis.....	231
ML_TechInterface.library.....	47
mMotionCyclic.....	193, 224, 230, 231
ModeAB.....	63
ModeAH.....	64
ModeExternalFB.....	73
ModeFlexProfile.....	71
ModeHoming.....	65
ModeInterrupted.....	72
ModePosAbs.....	65
ModePosAdd.....	67
ModePosRel.....	66
ModeStopping.....	64
ModeSyncCam.....	69
ModeSyncPhase.....	68
ModeSyncProfile.....	70
ModeSyncVel.....	69
ModeVel.....	67
Modularization rules.....	186, 246
Module axes.....	187
Module Coordination.....	192
Module instances.....	191
Module interface	187
Module reuse	220
Modules.....	186
Module-specific I/O Data	190
mOneTimeInit.....	181, 221, 230, 244
mOpModeChangeAllowed.....	184, 230
mReleaseOpMode.....	185
mScanError.....	234
mSetAppError.....	194, 255
mSetFBError.....	195, 256
mSetModuleEnable.....	219

N

Navigation system overview.....	78
---------------------------------	----

O

Operation mode independent logic.....	34
Operation modes	
Absolute positioning	65

O

...Operation modes	
Additive Positioning	67
Drive-controlled interruption	72
Drive Halt	64
Drive Homing	65
Drive ready	63
Drive Stop	64
Electronic cam	69
Electronic FlexProfile	71
Electronic MotionProfile	70
External function block	73
Phase synchronization	68
Relative Positioning	66
Velocity Control	67
Velocity synchronization	69
Operation modes, general information.....	33
Operation modes in the unit.....	251
OPMODE_AUTO.....	183
OpMode_ErrorSuccessor.....	215
OPMODE_INIT_COLD.....	244
OPMODE_INIT_WARM.....	244

P

PackML V3.0.....	226
PELV.....	24
PowerOn.....	63
Principles of error management.....	40
Principles of task management.....	34
Program for the AxisInterface.....	75
Protective extra-low voltage.....	24

R

Recommended task division when using IndraWorks/IndraLogic.....	35
--	----

S

Safety instructions for electric drives and con- trols.....	19
Sequential programming.....	251
SkipColdstartAtStartup.....	181
States in the Module.....	191
States of an Operation Mode.....	182
stControlDataActive_gb.....	258
stControlDataCustomer_gb.....	258
stControlDataVisuIL_gb.....	258
stMachine1Ctrl_gb.....	229
stMachine1Status_gb.....	229
Support	
see Service Hotline	287
System_Overview_All_Axis.....	76

T

Target mode after initializing the GAT.....	183
Task configuration, GAT.....	192

T

Task configuration, GAT, decentral.....	253
Task management, general information.....	34
TE_AxisInterfaceType01Prog.....	75
TE_BASE_MODULE_AXIS.....	187
TE_BaseUnit.....	245
TE_CMI_CONTROL.....	208
TE_CMI_STATUS.....	209
TE_DummyMode.....	181
TE_DummyModule.....	186
TE_ErrorHandling.....	254
TE_GAT_AXIS.....	187
TE_Machine.....	223
TE_MACHINE_CTRL.....	207
TE_MACHINE_DATA.....	258
TE_MACHINE_STATUS.....	208
TE_MachineControl.....	252
TE_OPMODES.....	245
TE_UNIT_CONTROL.....	247
TE_UNIT_STATUS.....	247
TechInterface	
Introduction and overview	47
The sample project GAT ^{compact}	121
Time sensitive, application-specific error re- action.....	228
Typical contents of a visualization.....	43

U

Unit axes.....	247
Unit coordination.....	252

U

Unit instances	251
Unit interface.....	247
Unit reuse	279
Units.....	245
Unit-specific I/O data	250
UpdateEveryInput.....	50
UserData.....	222, 232
User-defined GAT.....	225
User-defined GAT, decentral.....	281
User extension	
arAxisCtrl_gb[]	90
arStatusCtrl_gb[]	91
Function block	91
User extensions.....	88

V

Visualization.....	43
Axis overview	78
IMC interface control	86
IMC interface status	86
Overview	76
PLC load	88
Positioning mode	80
Synchronization mode	82
Synchronization mode with profiles	84
System overview	76
Velocity mode	81
Visualization, GAT, decentral.....	258
Visualization, providing data using the PLC.....	44

Notes

Bosch Rexroth AG
Electric Drives and Controls
P.O. Box 13 57
97803 Lohr, Germany
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr, Germany
Tel. +49 (0)93 52-40-0
Fax +49 (0)93 52-48 85
www.boschrexroth.com/electrics



R911334191