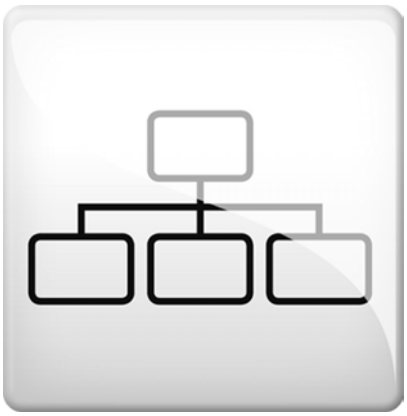
FAST Application Software

Application Template



Lenze Standard _____

Software Manual EN

13558104

Lenze

Contents

Content

1 About this documentation

This documentation describes how to create a structured program from the machine idea all the way through to the executable program in the »PLC Designer« by means of the Lenze FAST Application Template.

It shows how to create individual machine modules and interconnect them in the Machine Module Tree (MMT). In order to implement the machine application, FAST technology modules are integrated.

This documentation is part of the "Controller-based Automation" manual collection. It consists of the following sets of documentation:

Documentation type	Subject
Product catalogue	Controller-based Automation (system overview, sample topologies) Lenze Controller (product information, technical data)
System manuals	Visualisation (system overview/sample topologies)
Communication manuals **Online helps**	Bus systems • Controller-based Automation EtherCAT® • Controller-based Automation CANopen® • Controller-based Automation PROFIBUS® • Controller-based Automation PROFINET®
Reference manuals **Online helps**	Lenze Controllers: • Controller 3200 C • Controller c300 • Controller p300 • Controller p500
Software manuals **Online helps**	Lenze Engineering Tools: • »PLC Designer« (programming) • »Engineer« (parameter setting, configuration, diagnostics) • »VisiWinNET® Smart« (visualisation) • »Backup & Restore« (backup, restore, update)

More technical documentation for Lenze components

Further information on Lenze products which can be used in conjunction with Controller-based Automation can be found in the following sets of documentation:

Planning / configuration / technical data	
☐	**Product catalogues** • Controller-based Automation • Controllers • Inverter Drives/Servo Drives
Mounting and wiring	
🕮	**Mounting instructions** • Controllers • Communication cards (MC-xxx) • I/O system 1000 (EPM-Sxxx) • Inverter Drives/Servo Drives • Communication modules
☐	**Hardware manuals** • Inverter Drives/Servo Drives
Parameter setting / configuration / commissioning	
☐	**Online help/reference manuals** • Controllers • Inverter Drives/Servo Drives • I/O system 1000 (EPM-Sxxx)
☐	**Online help/communication manuals** • Bus systems • Communication modules
Sample applications and templates	
☐	**Online help / software and reference manuals** • i700 application sample • Application Samples 8400/9400 • FAST Application Template • FAST technology modules

Symbols:

- 🕮 Printed documentation
- ☐ PDF file / online help in the Lenze engineering tool

Tip!

Current documentation and software updates with regard to Lenze products can be found in the download area at:

www.lenze.com

Target group

This documentation is intended for all persons who plan, program and commission a Lenze automation system on the basis of the Lenze FAST Application Software.

Screenshots/application examples

All screenshots in this documentation are application examples. Depending on the firmware version of the Lenze devices used and the software version of the Engineering tools installed (e.g. »PLC Designer«), screenshots in this documentation may differ from the representation on the screen.

Information regarding the validity

The information in this documentation is valid for the following Lenze software:

Software	From software version
»PLC Designer« (**L_EATP_ApplicationTemplate** library)	3.13

1.1 Document history

Version			Description
4.0	10/2018	TD29	General revision
3.0	06/2016	TD17	Updated to »PLC Designer« V3.13 • General revision
2.0	12/2015	TD17	Updated to »PLC Designer« V3.12 • General revision
1.6	05/2015	TD17	Updated to »PLC Designer« V3.10
1.5	12/2014	TD11	Updated to »PLC Designer« V3.9
1.4	10/2013	TD11	Updated to »PLC Designer« V3.6 • Optimisations from usability tests (user group) • System error messages have been added. • L_EATP_MMD_Base structure has been added. • "Create MM Instance" command has been added.
1.3	04/2013	TD11	Updated to »PLC Designer« V3.5 • Software update of "Application Template Counter"/"Application Template". • New: Application example "flying saw".
1.2	11/2012	TD11	Updated to »PLC Designer« V3.3.2 • New: "Application Template Counter" sample project (Lenze standard)
1.1	07/2012	TD11	Updated • General correction • Adaptation to VISU layout according to the Lenze programming style guide for function blocks.
1.0	04/2012	TD11	First edition

1.2 Conventions used

This documentation uses the following conventions to distinguish between different types of information:

Type of information	Highlighting	Examples/notes
Spelling of numbers		
Decimal separator	Point	The decimal point is always used. For example: 1234.56
Text		
Version information	Blue text colour	All information that only applies to a certain controller software version or higher is identified accordingly in this documentation. Example: This function extension is available from software version V3.0 onwards!
Program name	» «	Lenze »PLC Designer«...
Window	*italics*	The *message window*... / The *Options* dialog box ...
Variable names		Setting *bEnable* to TRUE...
Control element	**bold**	The **OK** button ... / The **Copy** command ... / The Properties tab ... / The **Name** input field ...
Sequence of menu commands		If several commands must be used in sequence to carry out a function, the individual commands are separated by an arrow: Select **File→Open** to...
Shortcut	**<bold>**	Use **<F1>** to open the online help.
		If a shortcut is required for a command to be executed, a "+" has been put between the key identifiers: With **<Shift>+<ESC>** ...

Program code	`Courier`	`IF var1 < var2 THEN`
		`  a = a + 1`
		`  END IF`

Keyword	**`Courier bold`**	
Hyperlink	underlined	Optically highlighted reference to another topic. It is activated with a mouse-click in this online documentation.
Icons		
Page reference	(📖 7)	Reference to further information: Page number in PDF file.
Step-by-step instructions		Step-by-step instructions are indicated by a pictograph.

1.3 Terminology used

Term	Meaning
Controllers	The Controller is the central component of the Lenze automation system which controls the motion sequences by means of the operating system. The Controller communicates with the field devices (inverters) via the fieldbus.
Engineering PC	The Engineering PC and the Engineering tools installed serve to configure and parameterise the system. The Engineering PC communicates with the controller via Ethernet.
Inverters	Generic term for Lenze frequency inverters, servo inverters
MFB	Machine function block A machine function block contains the functions of a machine module (MM) in the »PLC Designer«.
MM	Machine module A machine module maps a subfunction of the machine/system in the »PLC Designer«. Machine modules are interconnected via the corresponding machine function blocks (MFB).
MMT	Machine Module Tree The machine module tree maps the structure of the automation system. The individual machine modules (MM) are arranged hierarchically in the tree topology.
PLC	Programmable Logic Controller (German designation: SPS - Speicherprogrammierbare Steuerung)

1.4 Notes used

The following signal words and symbols are used in this documentation to indicate dangers and important information:

Safety instructions

Layout of the safety instructions:

> **Pictograph and signal word!**
>
> (characterise the type and severity of danger)
>
> **Note**
>
> (describes the danger and gives information about how to prevent dangerous situations)

Pictograph	Signal word	Meaning
	Danger!	**Danger of personal injury through dangerous electrical voltage** Reference to an imminent danger that may result in death or serious personal injury if the corresponding measures are not taken.
	Danger!	**Danger of personal injury through a general source of danger** Reference to an imminent danger that may result in death or serious personal injury if the corresponding measures are not taken.
STOP	**Stop!**	**Danger of property damage** Reference to a possible danger that may result in property damage if the corresponding measures are not taken.

Application notes

Pictograph	Signal word	Meaning
	Note!	Important note to ensure trouble-free operation
	Tip!	Useful tip for easy handling
		Reference to another document

2 Safety instructions

Please observe the safety instructions in this documentation when you want to commission an automation system or a plant with a Lenze Controller.

The device documentation contains safety instructions which must be observed!

Read the documentation supplied with the components of the automation system carefully before you start commissioning the Controller and the connected devices.

Danger!

High electrical voltage

Injury to persons caused by dangerous electrical voltage

Possible consequences

Death or severe injuries

Protective measures

Switch off the voltage supply before working on the components of the automation system.

After switching off the voltage supply, do not touch live device parts and power terminals immediately because capacitors may be charged.

Observe the corresponding information plates on the device.

Danger!

Injury to persons

Risk of injury is caused by ...

- unpredictable motor movements (e.g. unintended direction of rotation, too high velocities or jerky movement);
- impermissible operating states during the parameterisation while there is an active online connection to the device.

Possible consequences

Death or severe injuries

Protective measures

- If required, provide systems with installed inverters with additional monitoring and protective devices according to the safety regulations valid in each case (e.g. law on technical equipment, regulations for the prevention of accidents).
- During commissioning, maintain an adequate safety distance to the motor or the machine parts driven by the motor.

Stop!

Damage or destruction of machine parts

Damage or destruction of machine parts can be caused by ...

- Short circuit or static discharges (ESD);
- unpredictable motor movements (e.g. unintended direction of rotation, too high velocities or jerky movement);
- impermissible operating states during the parameterisation while there is an active online connection to the device.

Protective measures

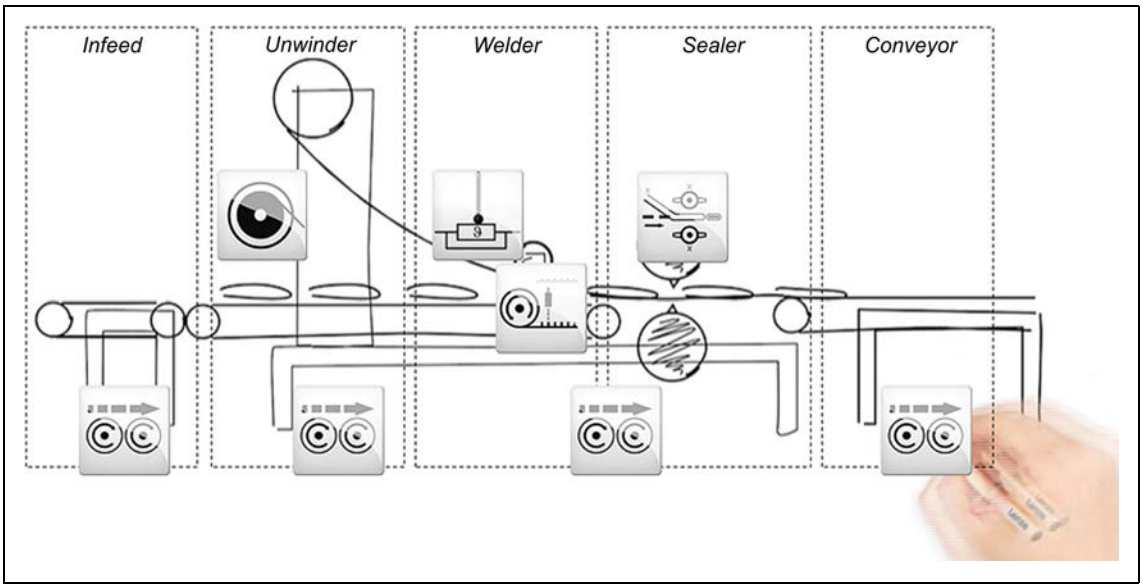
- Always switch off the voltage supply before working on the components of the automation system.
- Do not touch electronic components and contacts unless ESD measures were taken beforehand.
- If required, provide systems with installed inverters with additional monitoring and protective devices according to the safety regulations valid in each case (e.g. law on technical equipment, regulations for the prevention of accidents).

3 System requirements

	Engineering PC	Lenze Controller
Hardware	PC/notebook	PLC (Logic) from firmware V3.13
Operating system of	• Microsoft® Windows® XP Professional (32 bits) from SP3 • Microsoft® Windows® 7 (32 and 64 bits)	Microsoft® Windows® CE
Required Lenze software	»PLC Designer« from V3.13 and installed **L_EATP_ApplicationTemplate** library (from V3.13)	Runtime Software Motion For this purpose, the project information has to be updated: "Update devices" command.

4 Structured programming with the Application Template

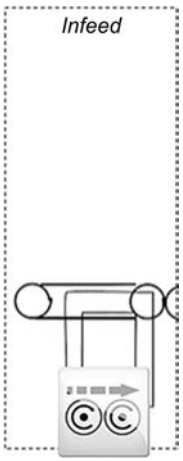
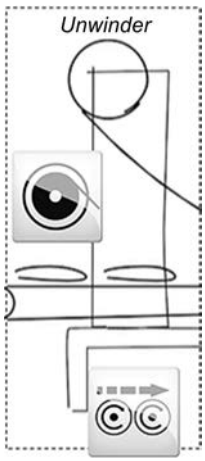
Everything starts with a machine idea. In order to convert this idea into a machine that is to be implemented with the Application Template, individual machine modules are created for the different drive tasks.



[4-1] Machine idea split into machine modules

Figure [4-1] shows a form-fill-seal machine that has been split into several machine modules.

In the further course, the documentation only refers to a sample implementation of the module "Infeed" and "Unwinder" in the »PLC Designer«:

Infeed		Unwinder	
	The "Infeed" machine module contains a drive that is mapped by the FAST technology module "Electrical Shaft Position".		The "Unwinder" machine module contains a drive that is also mapped by the FAST technology module "Electrical Shaft Position". The "Winder Dancer-controlled" technology module is not implemented for this example.

4.1 Procedure

Step	Activity
1.	Creating a new »PLC Designer« project – opening the Application Template (📖 15)
2nd	Creating machine modules (📖 16) or Copy machine module (📖 21)
3rd	Creating machine module instances (📖 24)
4.	Integrating machine modules into the Machine Module Tree (MMT) (📖 28)
5th	Implementing and connecting visualisation (📖 33)
6.	Inserting FAST technology modules (📖 35)
7.	Connecting axes (📖 38)
8.	Establishing the communication channel (ACD Slave Access) (📖 40)
9.	Using operation modes (📖 41)
10.	Using the communication channel (📖 43)
11.	Creating and processing error messages (📖 44)

Other tasks

- Save machine module as template (📖 19)
- Renaming a machine module (📖 26)
- Deleting machine modules (📖 32)
- Deleting machine module references (📖 31)

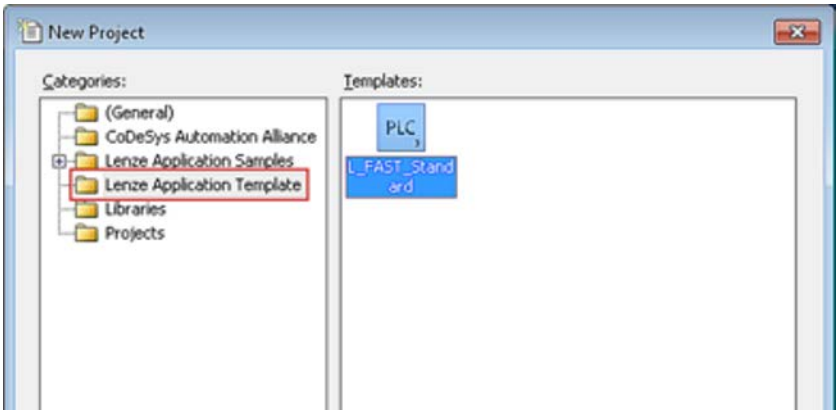
The individual tasks are described in detail in the following sections.

4.2 Creating a new »PLC Designer« project – opening the Application Template

How to proceed

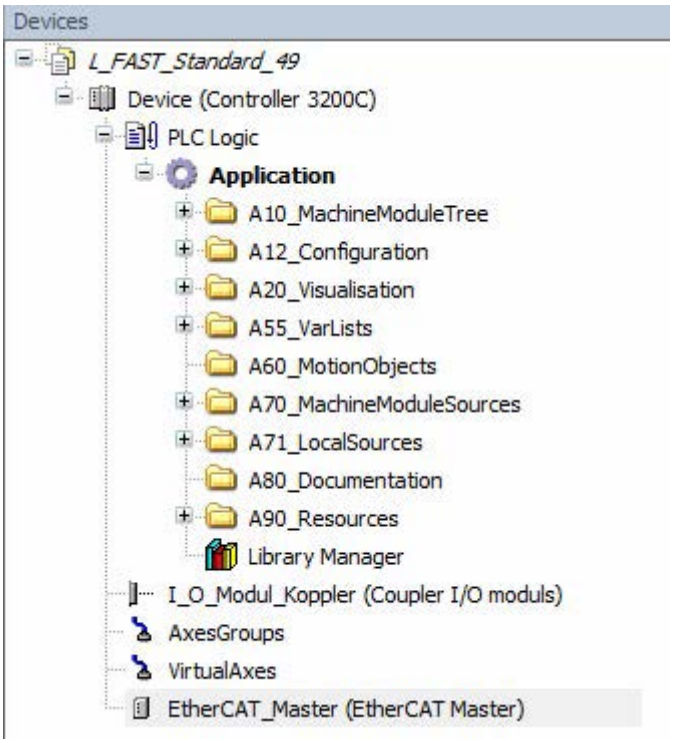
1. Start the »PLC Designer«.
2. Create a new project with the **File → New project** menu command.

   Select the "L_FAST_Standard" template from the "Lenze Application Template" category:



3. Confirm the entries with **OK**.

   The project is opened with this device tree structure:

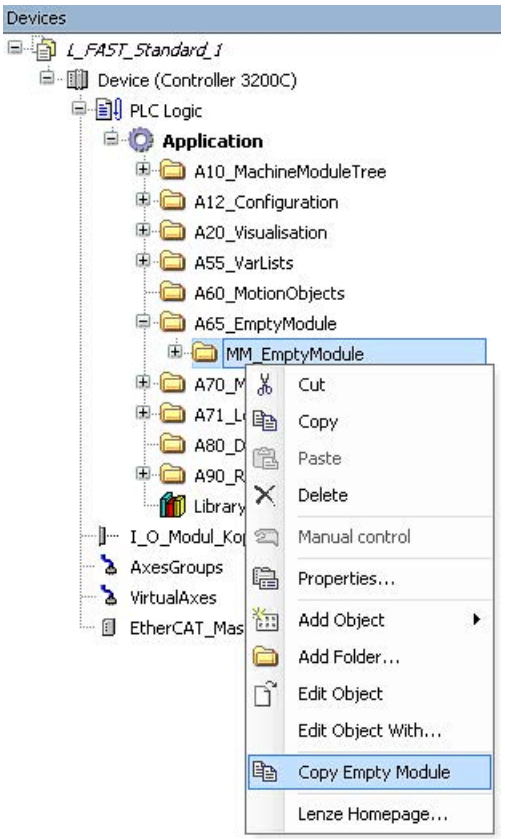


"Application" contains the Structure of the Application Template (📖 51).

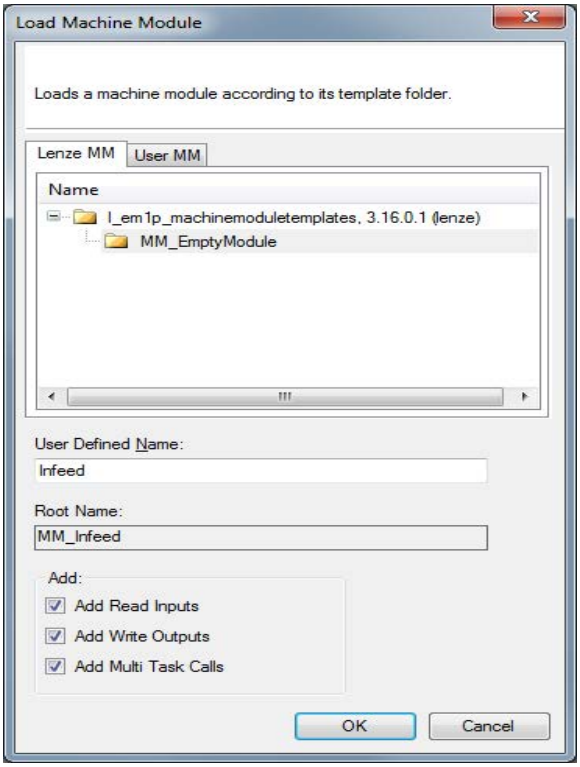
4.3 Creating machine modules

How to proceed

1. Right-click on the folder **A70_MachineModuleSources** and then select the menu command **Load Machine Module**.



2. In the "Load Machine Module" dialog box that appears, select tab "Lenze MM". In this tab, open group **l_em1p_machinemoduletemplates** and select the entry **MM_EmptyModule**.



3. Enter a name for the machine module in the lower section of the dialog box.

   The module name must not contain "MM_" or any special characters. Only the characters "A...Z", "a...z", "0...9" are permitted.

4. If necessary, assign optional actions in the "Add" section by checking the control fields.
   - "Add Read Inputs" → I01_ReadInputs (📖 75)
   - "Add Write Outputs" → O01_WriteOutputs (📖 79)
   - "Add Multi Task Calls" → MTC01_TaskMid / MTC02_TaskFree (📖 80)

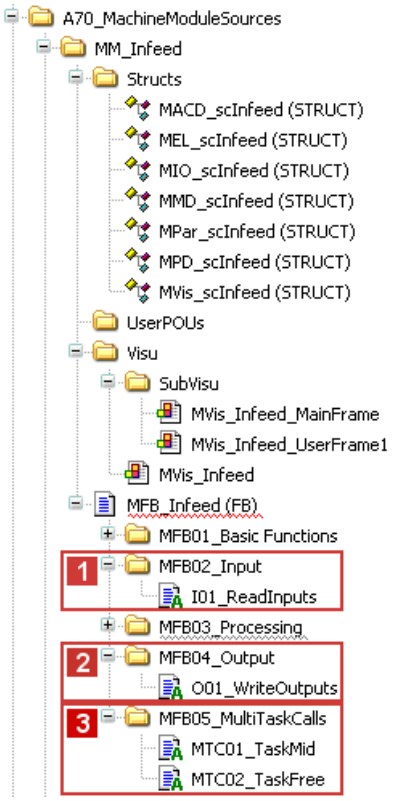
5. Confirm the entries with **OK**.

   The machine module is inserted with the name "MM_Infeed" under **A70_MachineModuleSources**.

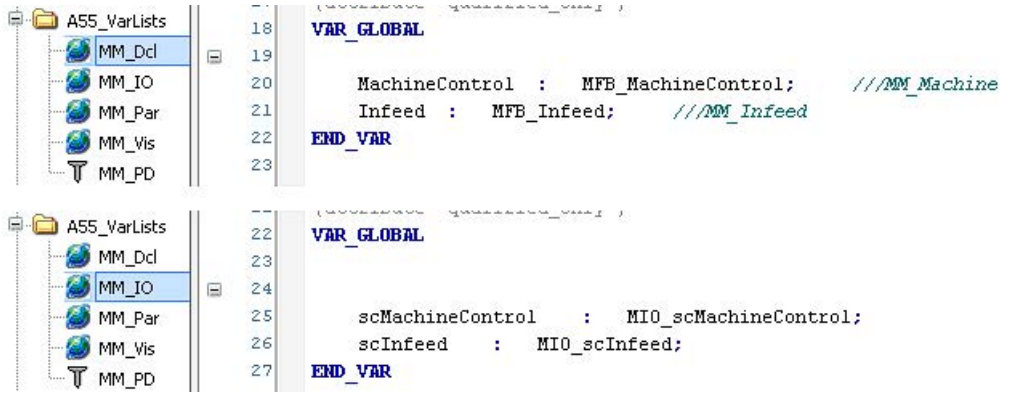
   The structures, visualisations and MFBs contained are instantiated with the name that was assigned.

   The selected (optional) actions are inserted.

   - "Add Read Inputs" ➞ 1 I01_ReadInputs (📖 75)
   - "Add Write Outputs" ➞ 2 O01_WriteOutputs (📖 79)
   - "Add Multi Task Calls" ➞ 3 MTC01_TaskMid / MTC02_TaskFree (📖 80)



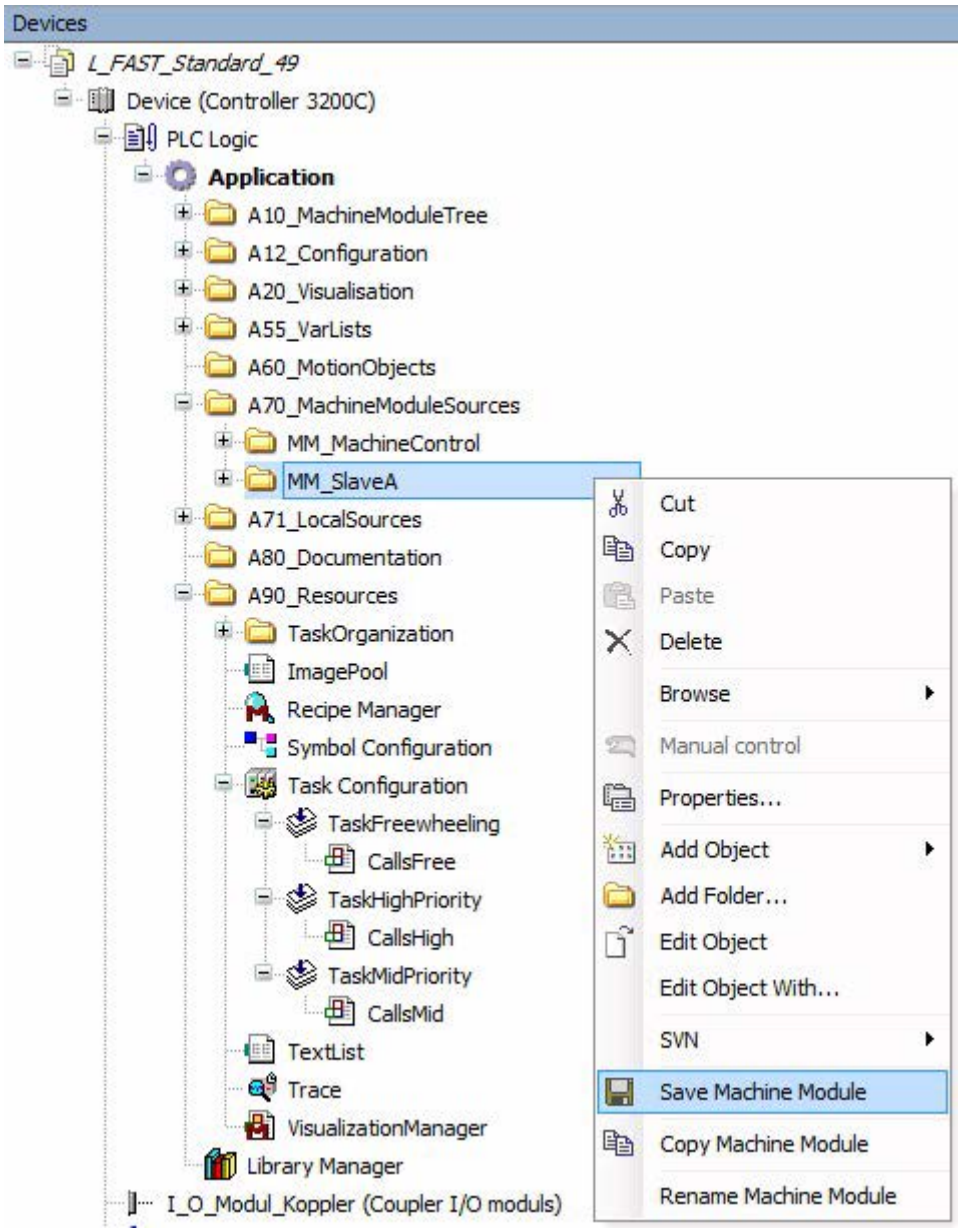
   In the predefined global variable lists **A55_VarLists**, the corresponding instance and the variable structure are declared as well:



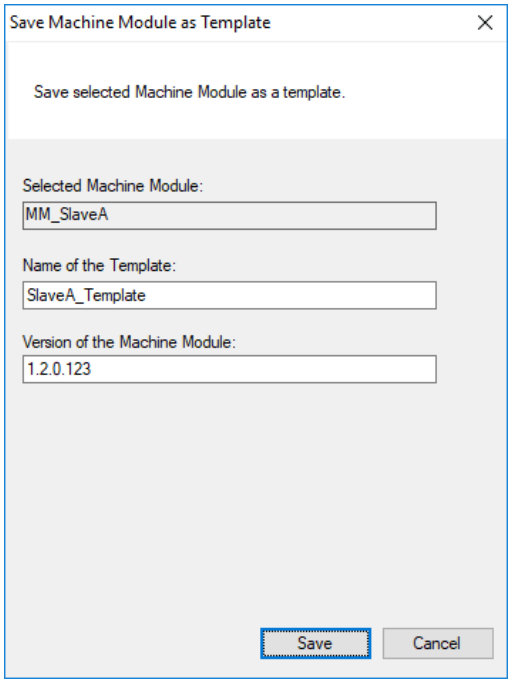
4.4 Save machine module as template

How to proceed

1. Right-click on the folder of the machine module to be copied ("MM_SlaveA" in the example) and execute the menu command **Save Machine Module**.



2. In the "Save Machine Module as Template" dialog box which then appears, assign a name and version number for the machine module template.



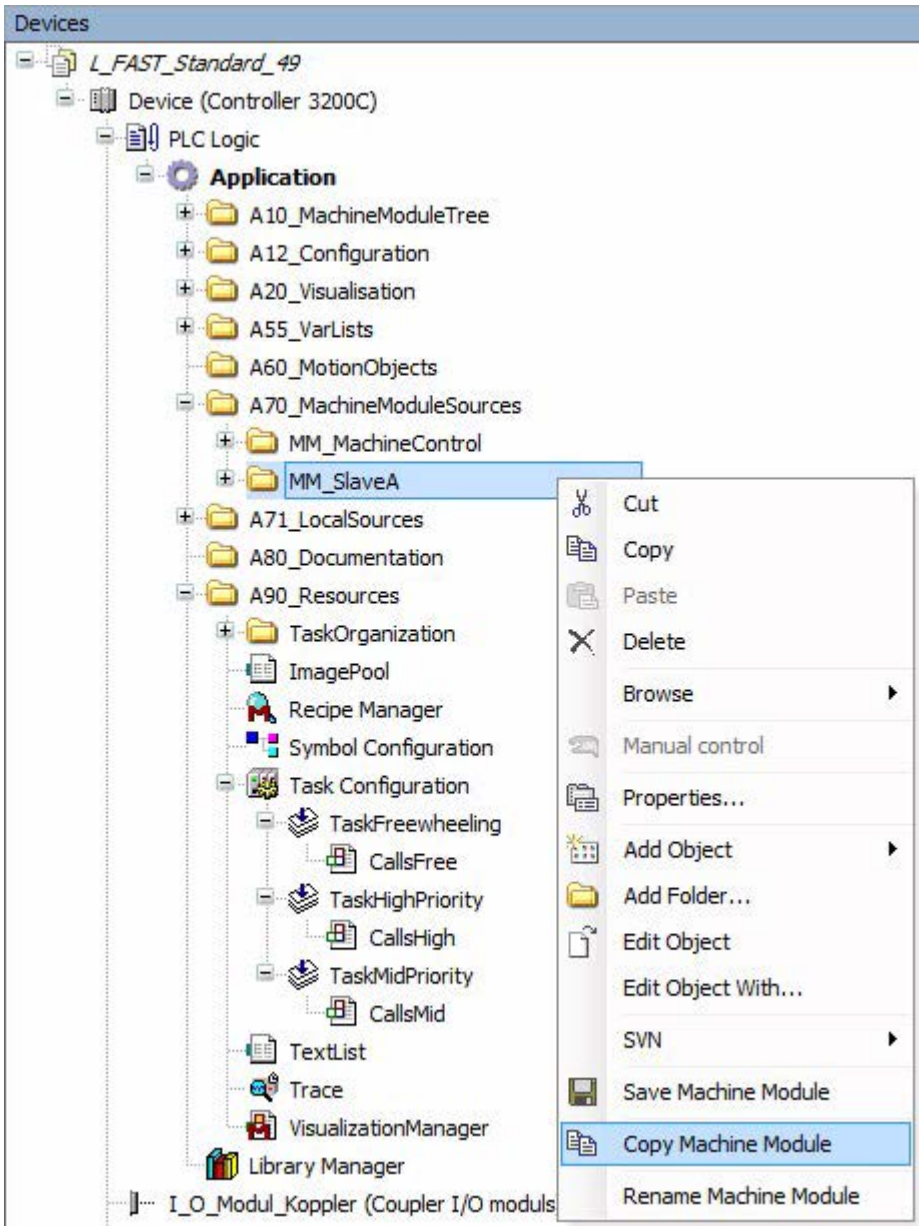
3. Then close the dialog with **OK**.

   When executing the **Create machine module** command, this machine module will from now on be available in the "User MM" tab. ▸ Creating machine modules (📖 16)

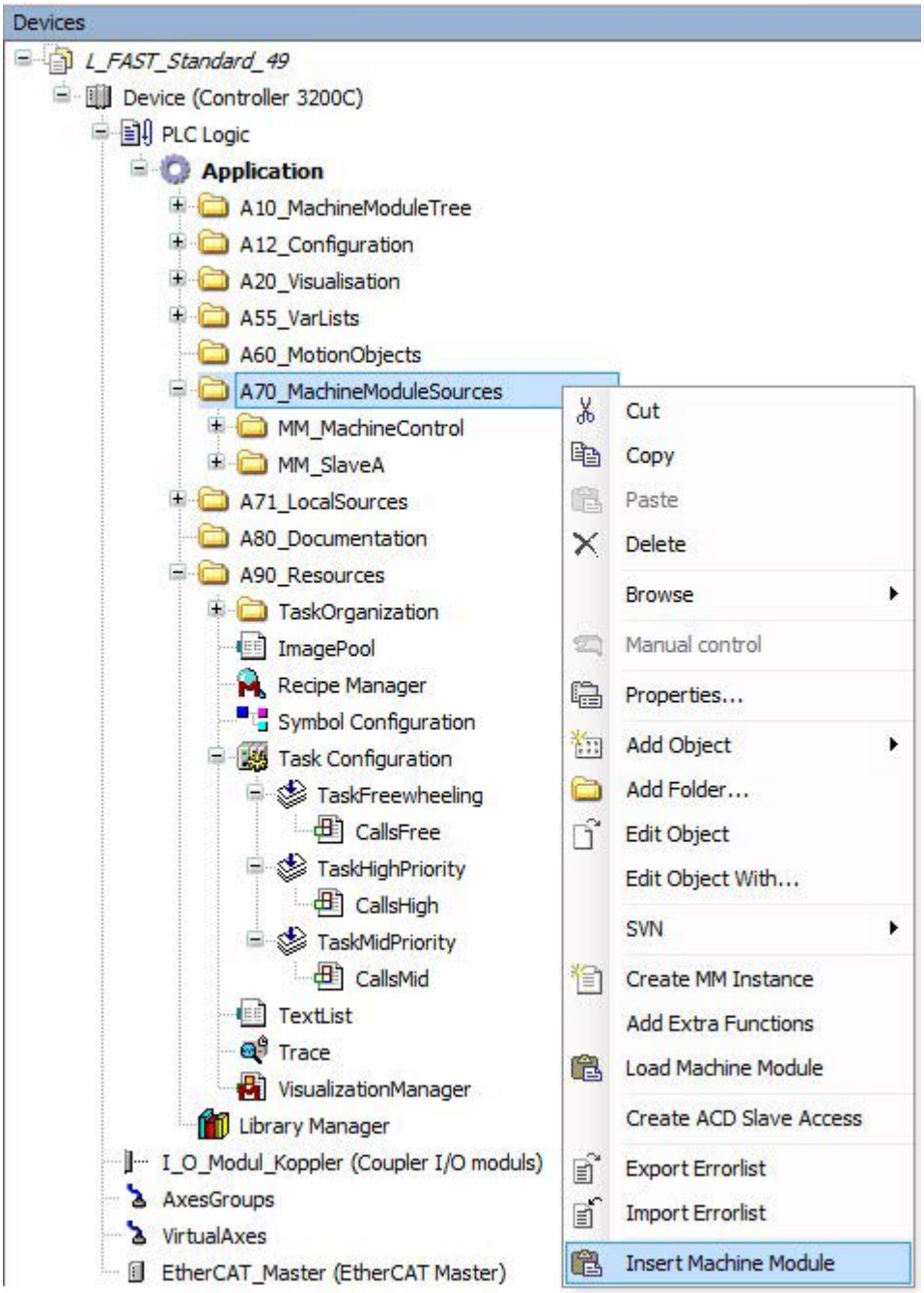
4.5 Copy machine module

How to proceed

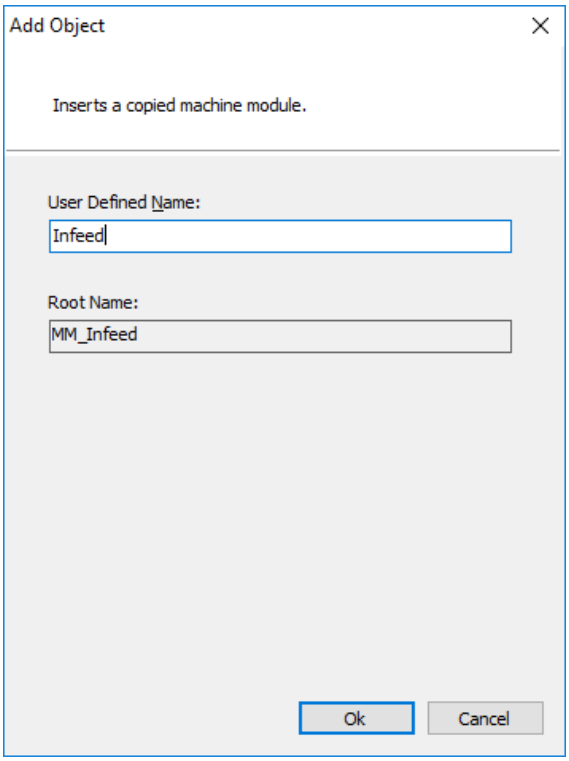
1. Right-click on the folder of the machine module to be copied ("MM_SlaveA" in the example) and use the menu command **Copy Machine Module** to copy the module to the clipboard as a template.



2. Right-click on the **A70_MachineModuleSources folder** and execute the menu command **Insert Machine Module**.



3. In the "Add Object" dialog box which then appears, enter a name for the new module in the **"User Defined Name"** field, e.g. "Infeed".

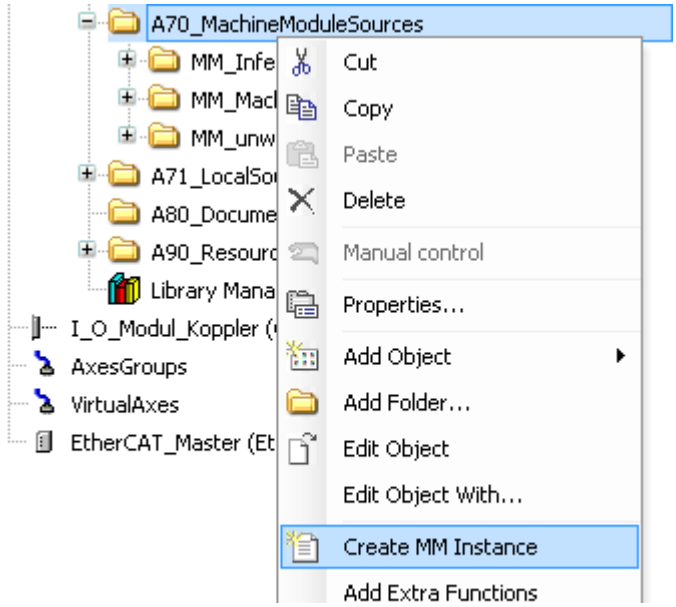


4. Then close the dialog with **OK**.
   The new machine module will be added to the folder **A70_MachineModuleSources**.

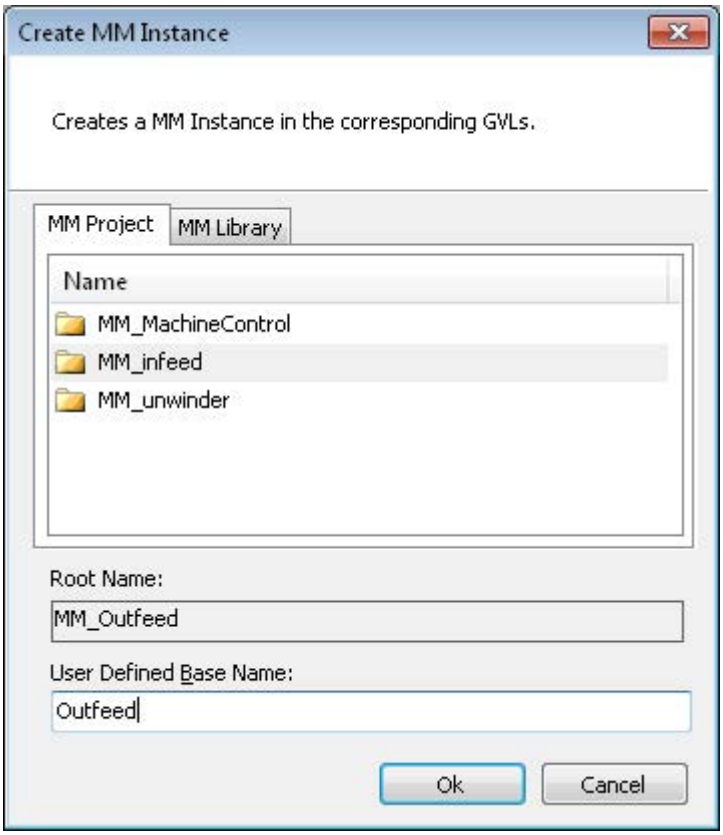
4.6 Creating machine module instances

How to proceed

1. Right-click the **A70_MachineModuleSources** folder and execute the **"Create MM Instance"** menu command.



2. Highlight the machine module from which an instance is to be created in the appearing dialog.



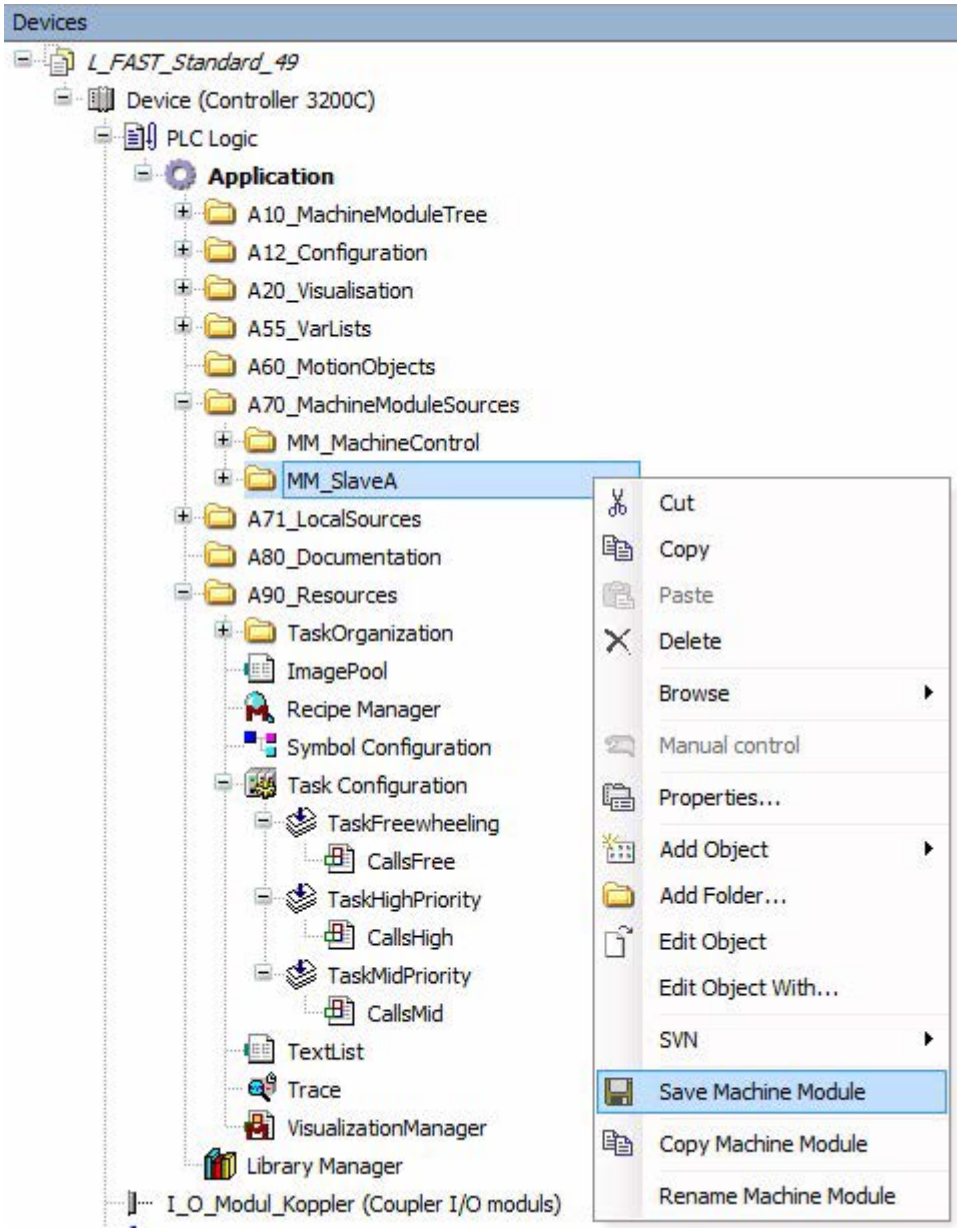
3. Enter an instance name in the "User Defined Base Name" input field (in the "Outfeed" example).
4. Click **OK**.

   In all global variable lists in the **A55_VarLists** folder, an instance of the machine module and its structures is created.

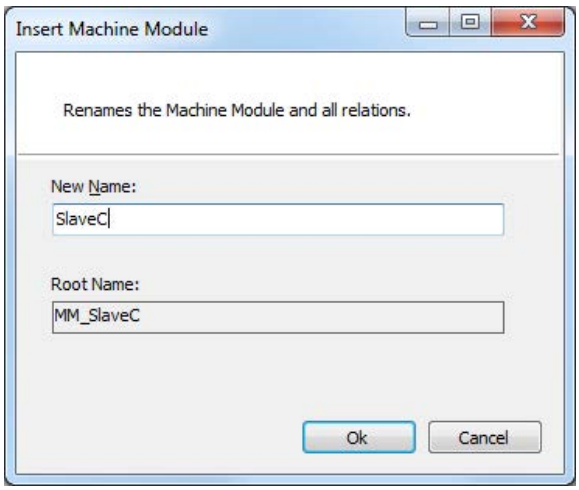
4.7 Renaming a machine module

How to proceed

1. Right-click on the folder of the machine module to be renamed and execute the menu command **Rename Machine Module**.



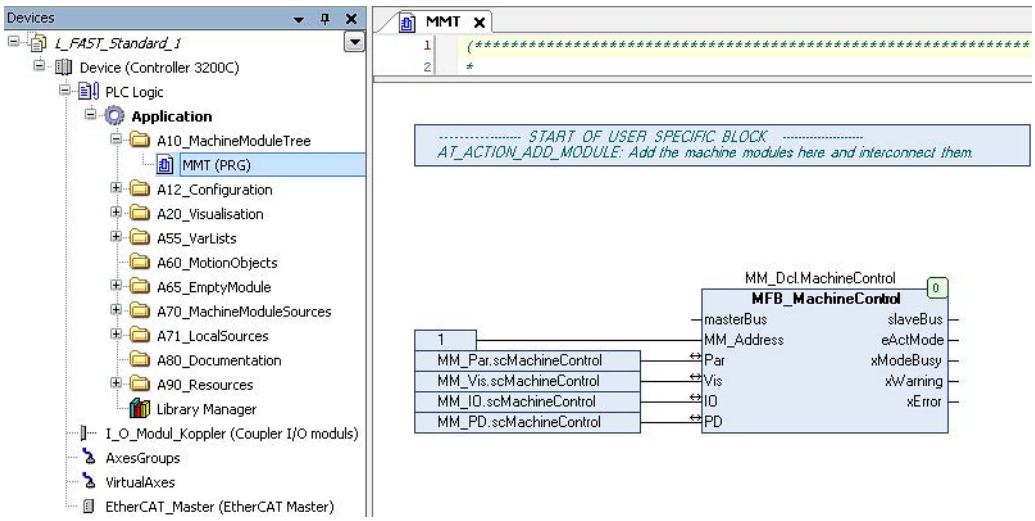
2. In the "Insert Machine Module" dialog box which then appears, enter a name for the new module in the **"New Name"** field, e.g. "SlaveC".



3. Then close the dialog with **OK**.
   The module is renamed.

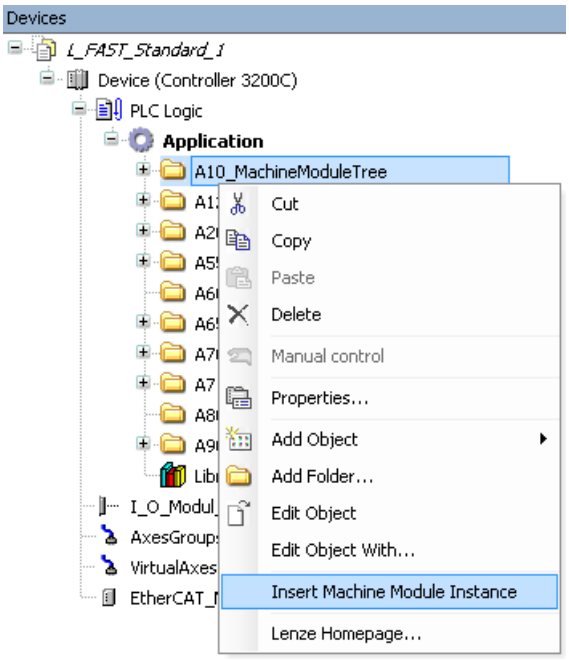
4.8 Integrating machine modules into the Machine Module Tree (MMT)

The prepared instances of the machine modules are called under **A10_MachineModuleTree** in the **MMT (PRG)** program. The template already contains the "MachineControl" master module in which the higher-level logic functions are programmed.



How to proceed

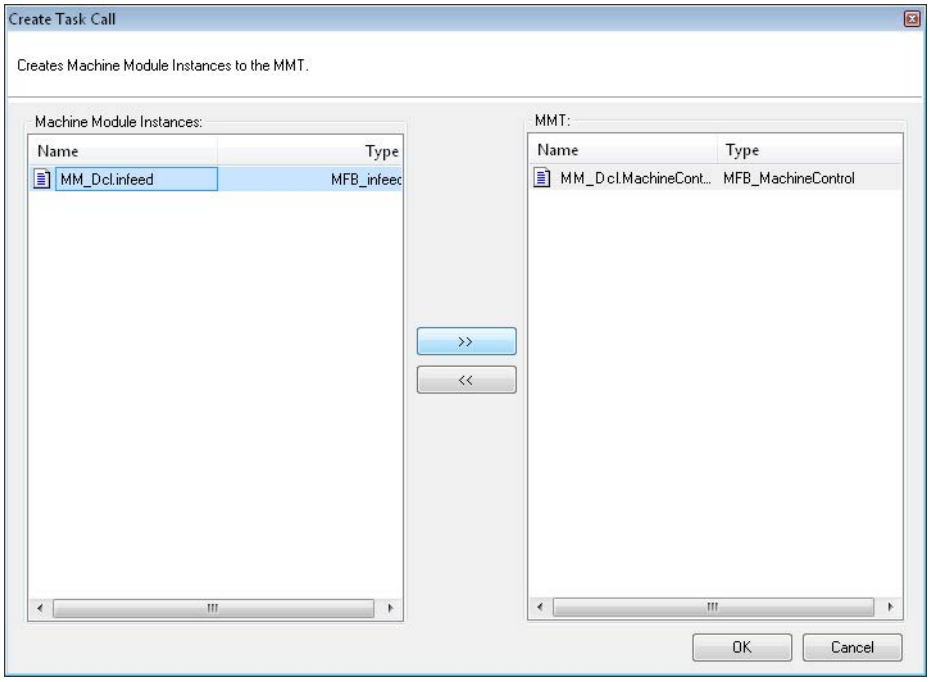
1. Go to **A10_MachineModuleTree** and execute the **"Insert Machine Module Instance"** menu command.



2. In the appearing dialog, move the new instance "MM_Dcl.Infeed" with the **">>"** button to the "MMT" list.

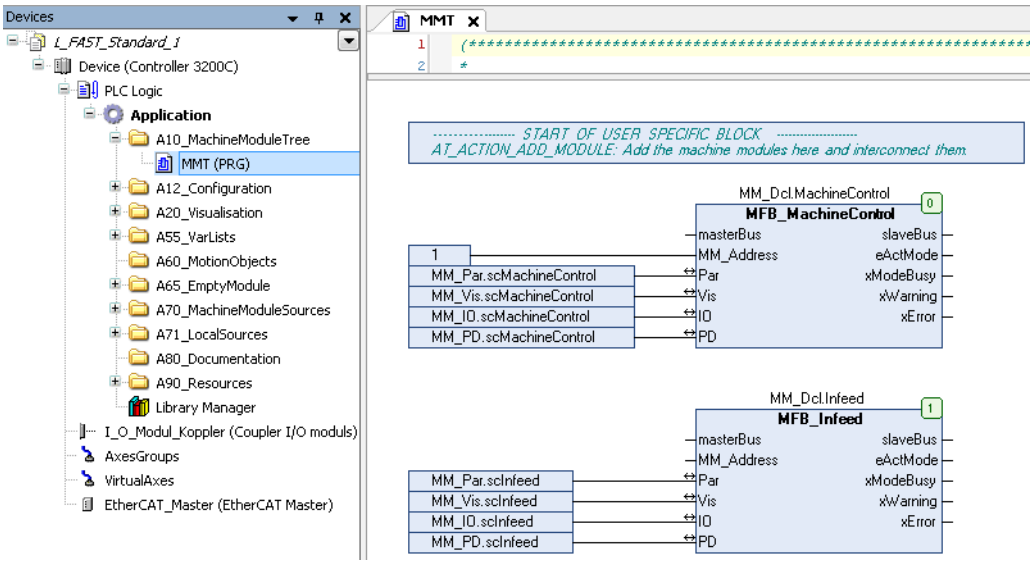
   The "Machine Module Instances" list (on the left) displays module instances that can be implemented into the Machine Module Tree.

   The "MMT" list (on the right) displays the module instances that are to be added to the Machine Module Tree.



3. Confirm the selection with **OK**.

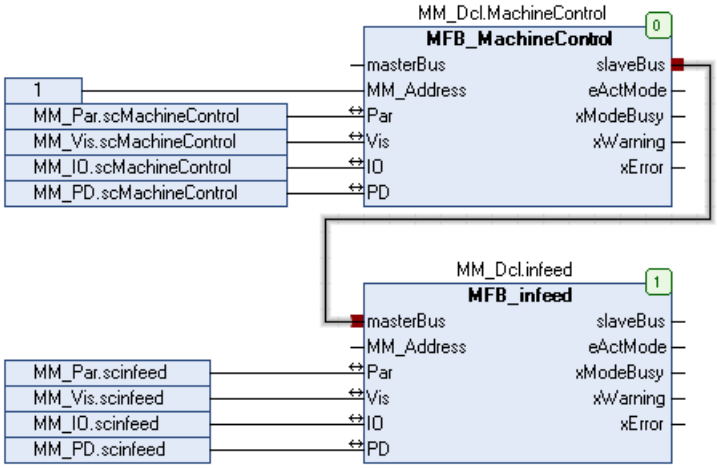
   The new "Infeed" instance is placed below the already available "MachineControl" master module and can be moved to the desired position via "Drag & Drop".



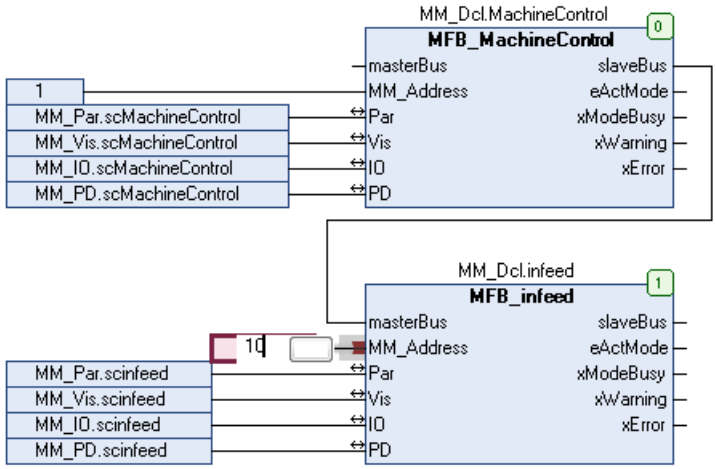
The associated structures (MACD, MEL, MIO, MMD, MPar, MPD, MVis) are automatically connected to the inputs of the module.

The execution order is displayed at the top-right corner of the module. A right-click on the MMT screen and the **"Execution Order"** context menu command serves to change the order.

4. Draw a line between the bus connections *slaveBus* and *masterBus* to establish communication between the "MachineControl" master module and the "Infeed" machine module.



5. Specify the address of the slave via the *MM_Address* input.
   - Click the line at the input.
   - Enter the desired address.



In this state, the "Infeed" machine module already responds to operation-mode commands of the "MachineControl" master module.

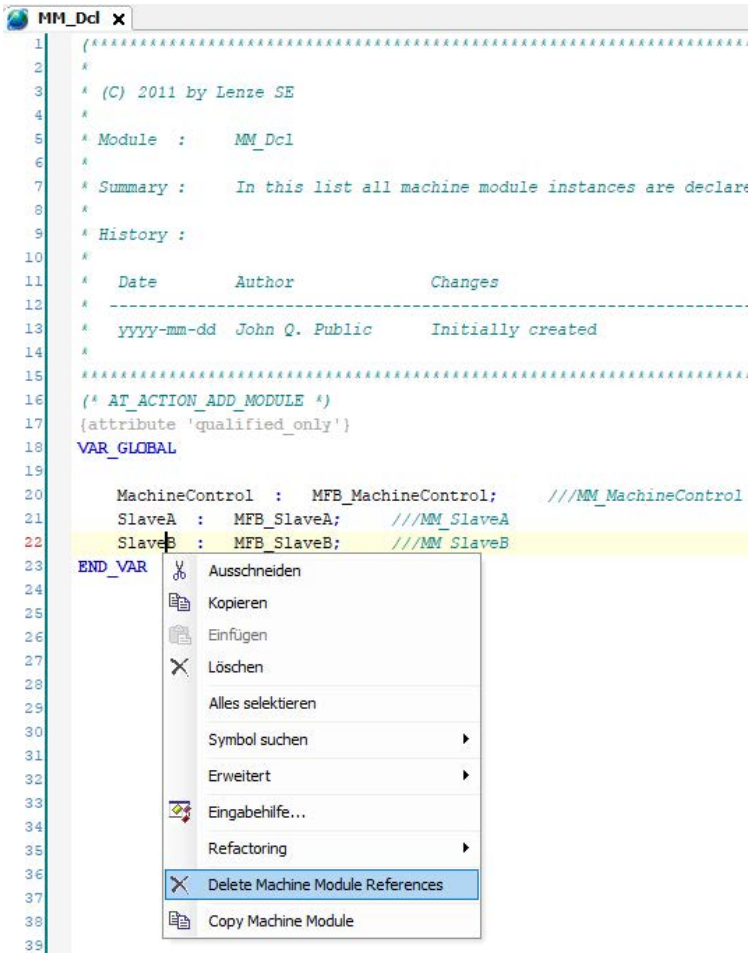
4.9 Deleting machine module references

How to proceed

1. Open the global variable list **A55_VarLists->MM_Dcl**.
2. Right-click on the machine module instance to be deleted (here: "SlaveB") and execute the menu command **Delete Machine Module References**.

Note!

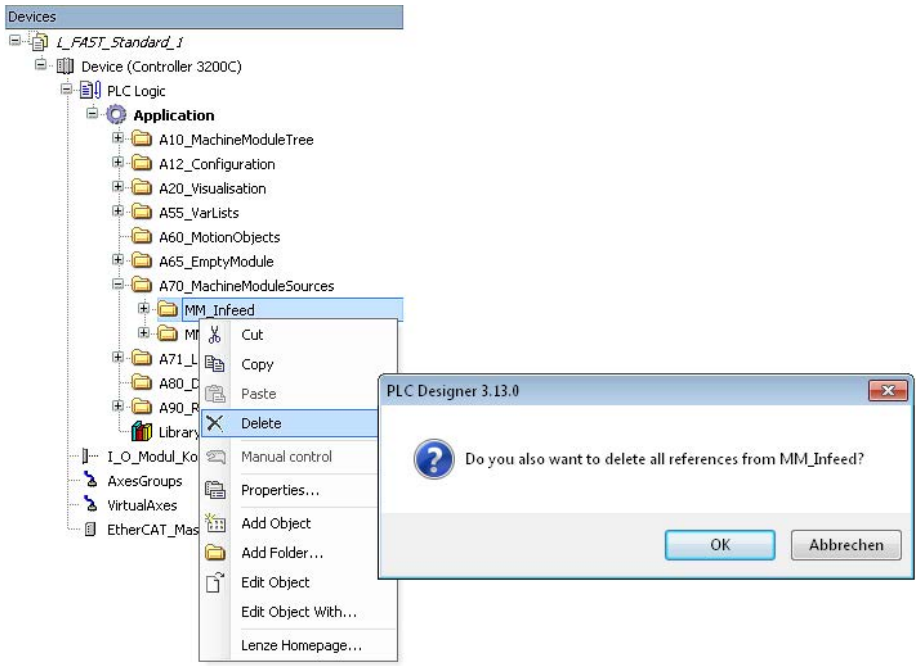
In order for the command **Delete Machine Module References** to appear, the cursor must be located within the instance name, not before or after.



All references to the corresponding machine module will be deleted.

4.10 Deleting machine modules

If you delete a machine module folder under **A70_MachineModuleSources**, the automatically generated declarations and the call in the Machine Module Tree are deleted after a query.

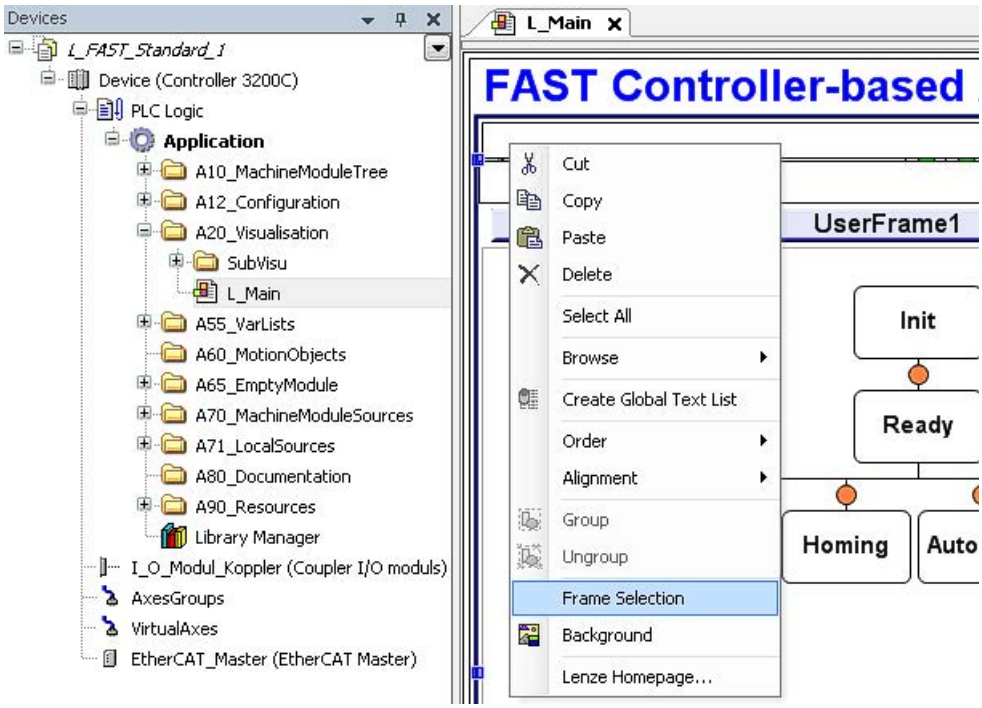


4.11 Implementing and connecting visualisation

In the Application Template, the visualisations of the individual machine modules must be added to the **L_Main** main visualisation. The visualisation for the "MachineControl" master module is already included.

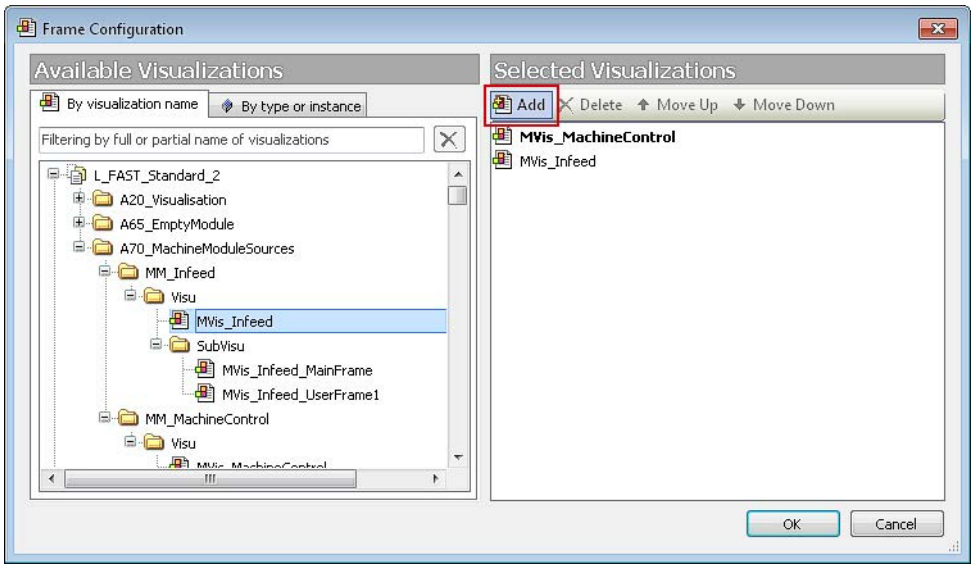
How to add the visualisation of the "MM_Infeed" machine module:

1. Go to **A20_Visualisation** and open the **L_Main** main visualisation.
2. Use the **Frame Selection** menu command to open the dialog for configuring the frame visualisations.



3. Go to "Available Visualizations", select the **MVis_Infeed** visualisation in the folder of the machine module **A70_MachineModuleSources/MM_Infeed/Visu** and click the **"Add"** button.

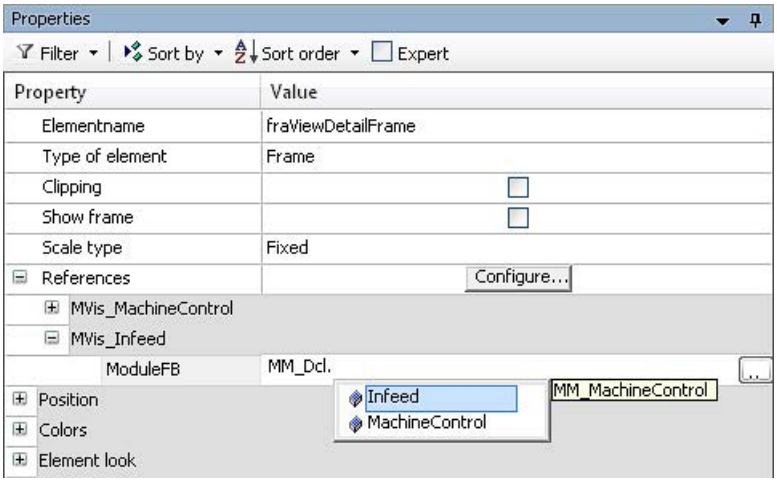
   The visualisation is inserted into the "Selected Visualizations".



4. Confirm the selection with **OK**.

   The machine module instance of the **MM_Infeed** module which is declared in the global **MM_Dcl** variable list serves to supply the visualisation with data.

   The Intellisense function serves to directly display the available instances included in the **MM_Dcl** variable list after setting the point.



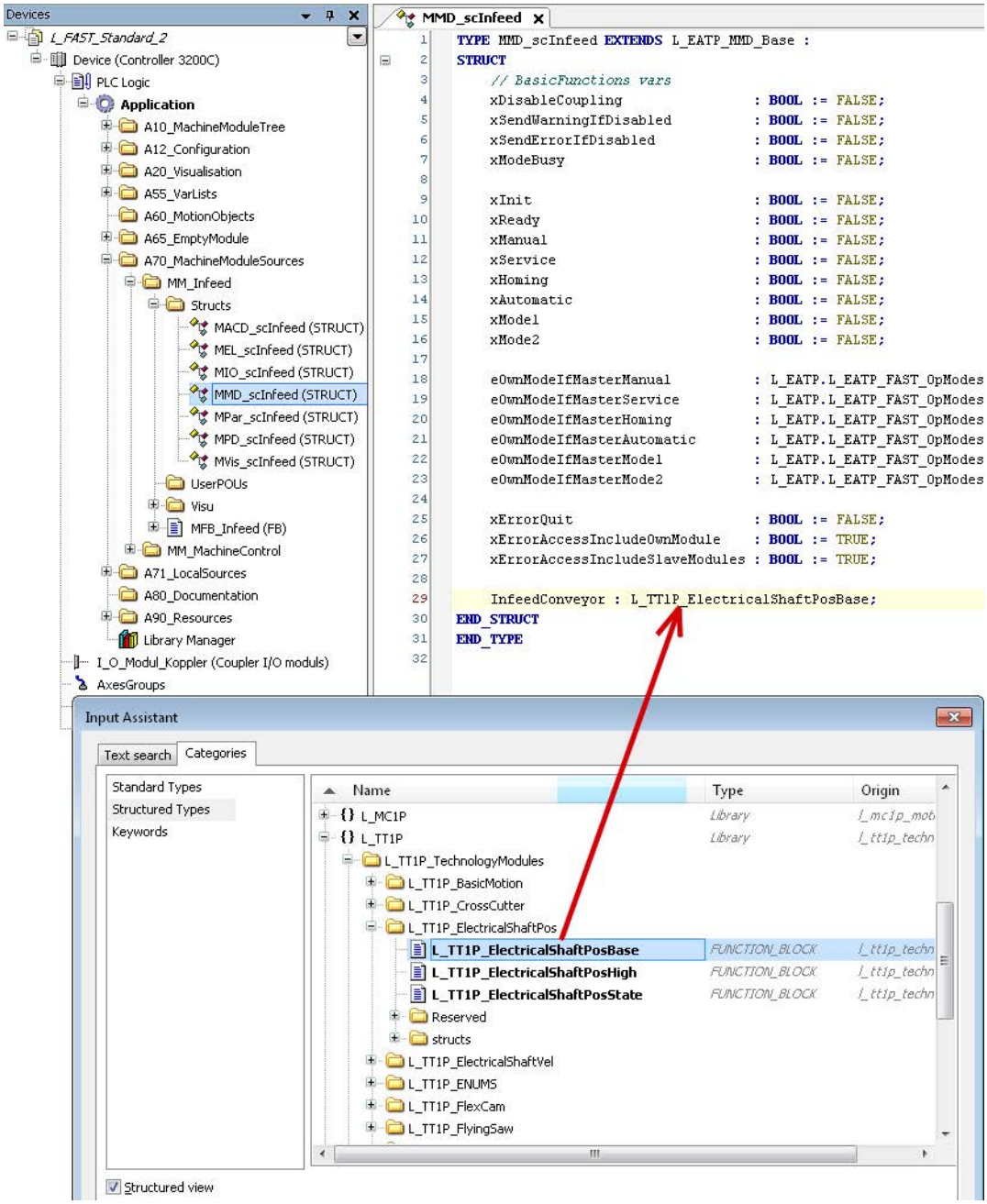
4.12 Inserting FAST technology modules

How to proceed

1. Declare the "Electrical Shaft Position" FAST technology module to be inserted in the **MMD_scInfeed** structure.

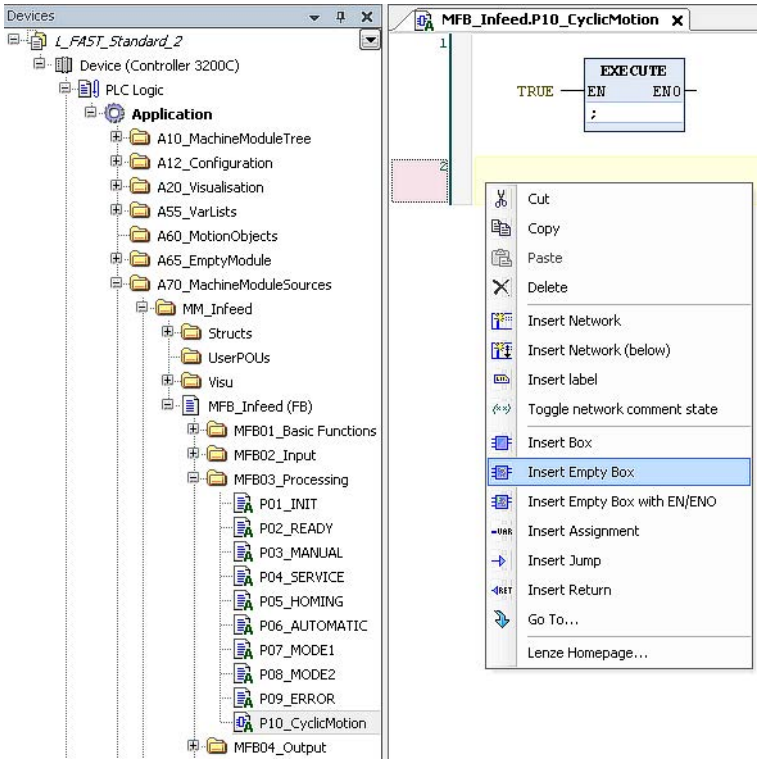
   Entry: `InfeedConveyor : L_TT1P_ElectricalShaftPosBase;`

   Tip: Right-clicking in the input area opens a menu from which the input assistance can be opened. The FAST technology modules can be found under category "Structured Types" ➞ L_TT1P.

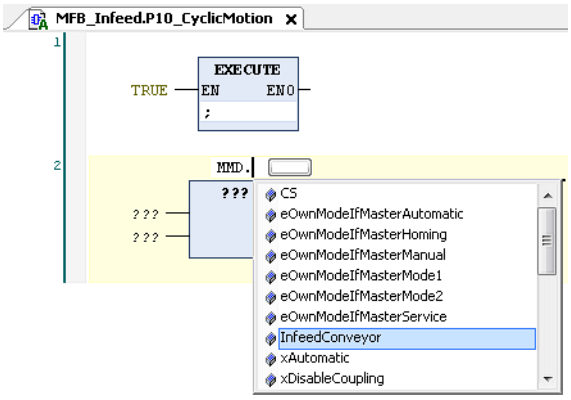


2. Insert an empty box under **MFB03_Processing → P10_CyclicMotion**.

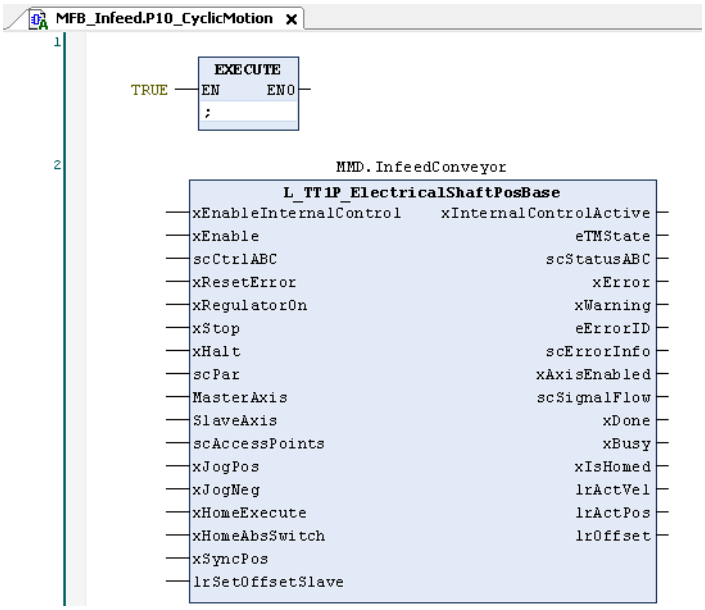
   Right-click the input area and execute the **"Insert Empty Box"** menu command.



3. Enter "MMD." at the "???" position at the top of the block and double-click **InfeedConveyor** in the appearing selection list.



The new **InfeedConveyor** block now contains the "Electrical Shaft Position" technology module.



The axes of the technology module are led to the outside via the interfaces of the machine module and only there connected to the real axes in the Machine Module Tree.

4.13 Connecting axes

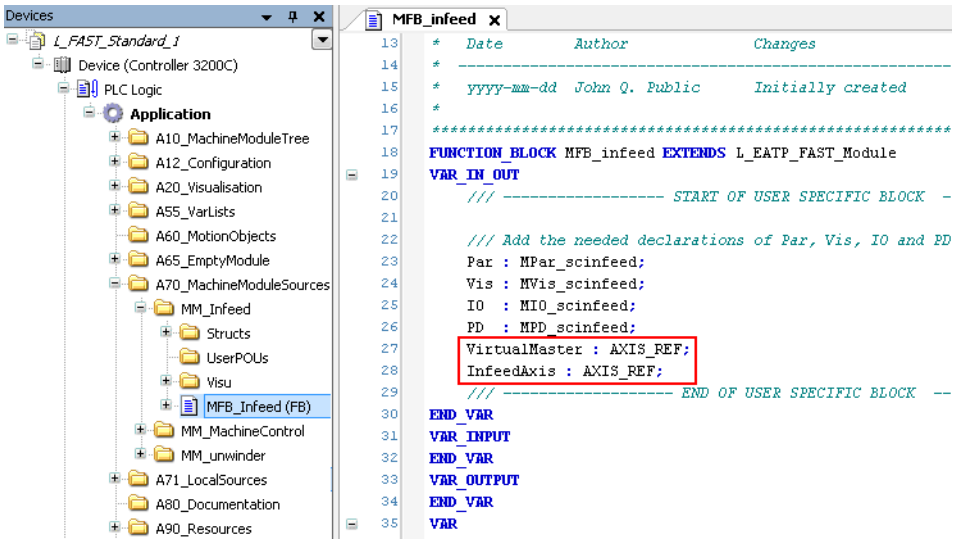
In the following example, the FAST technology modules "Electrical Shaft Position" and "Virtual Master" are connected to the processes "Infeed" and "Unwinder"

The "Virtual Master" integrated in the "MachineControl" master module defines the master position for the machine modules "Infeed" and "Unwinder".

The machine modules "Infeed" and "Unwinder" each contain an ElectricalShaftPosBase module. These modules are to be clutched into the master position after the final speed of the "Virtual Master" has been reached. For this purpose, a "Handshake" is required between the machine modules "Infeed" and "Unwinder" and the "MachineControl" master module (see Using the communication channel (📖 43)).

How to proceed

1. Create AXIS_REF variables under **A70_MachineModuleSources** in the inserted machine modules.

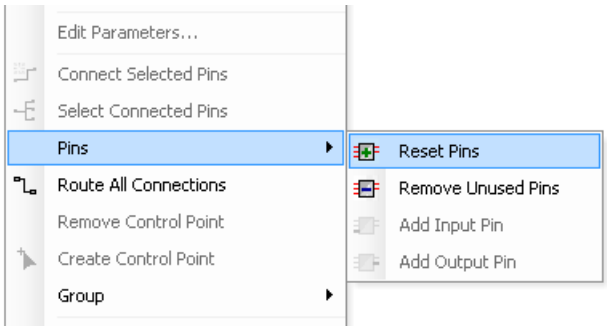


Entries

MFB_MachineControl	MFB_infeed	MFB_unwinder
`VirtualMaster : AXIS_REF;`	`VirtualMaster : AXIS_REF;` `InfeedAxis : AXIS_REF;`	`VirtualMaster : AXIS_REF;` `UnwinderAxis : AXIS_REF;`

2. Go to **A10_MachineModuleTree** and open the **MMT (PRG)** program.

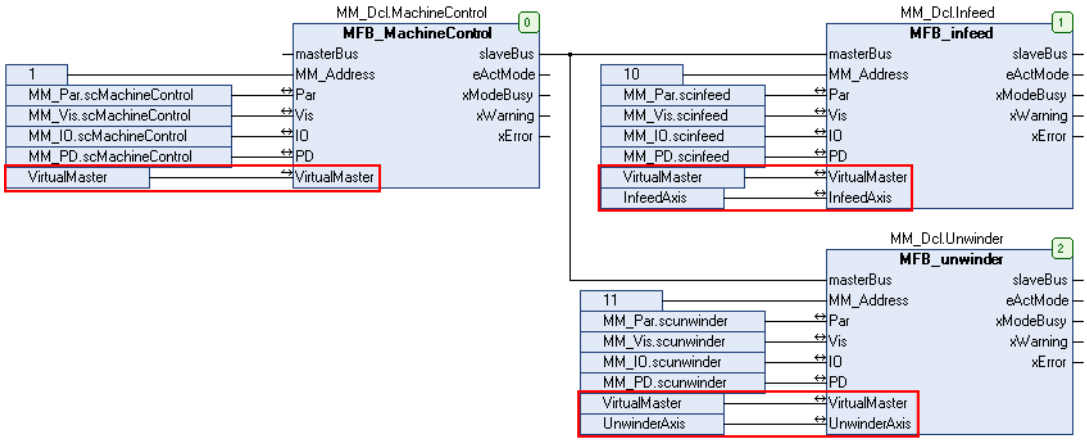
3. Right-click the "Infeed" machine module and execute the **"Pins → Reset Pins"** menu command.



Now, all technology modules are calculated.

The AXIS_REF variables are inserted into the **MMT (PRG)** program.

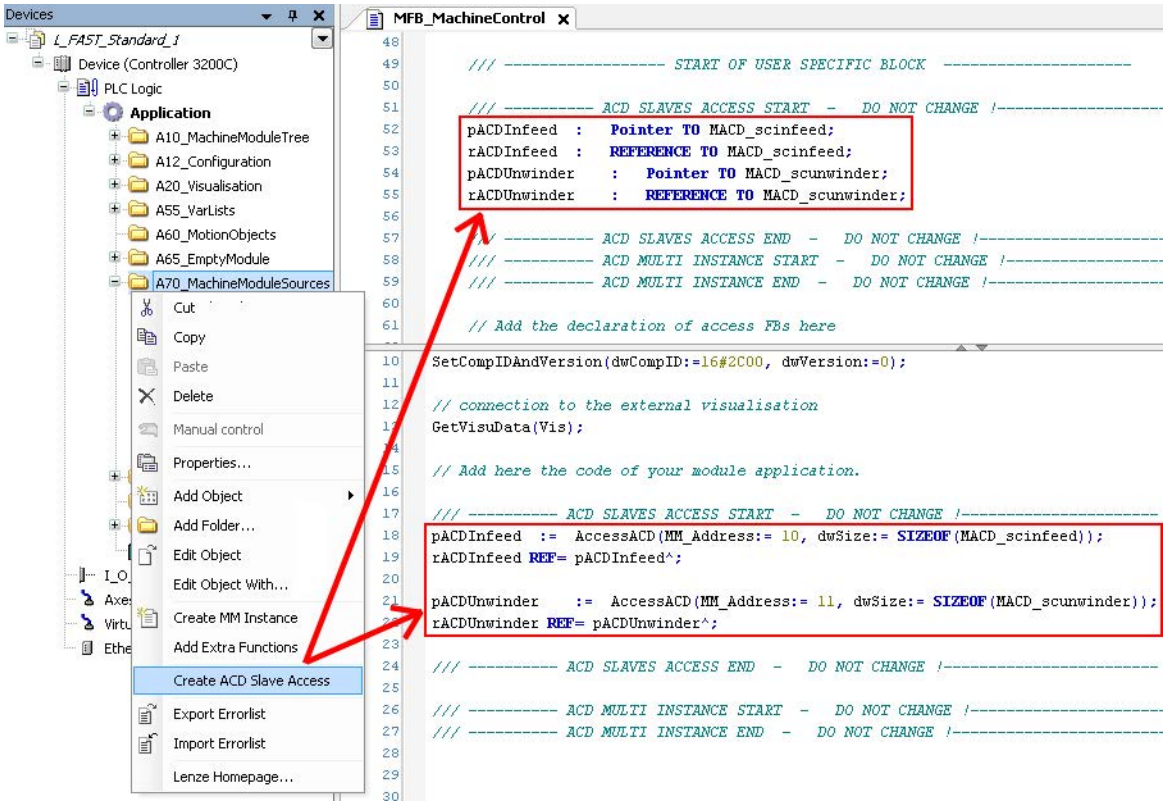
4. Connect the real and virtual axes to the blocks.



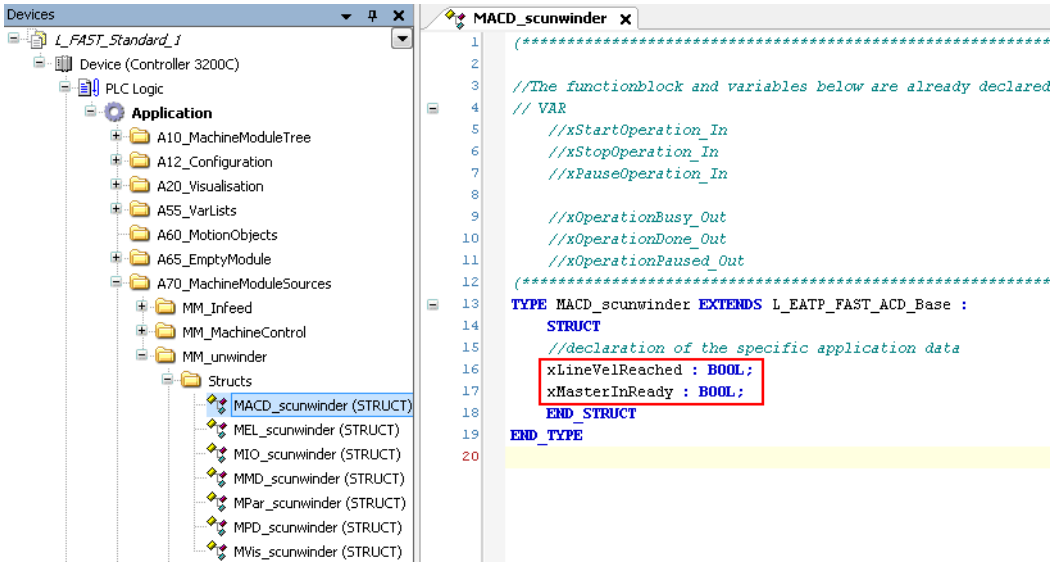
4.14 Establishing the communication channel (ACD Slave Access)

Right-click the **A70_MachineModuleSources** folder to open the context menu and execute the **"Create ACD Slave Access"** menu command.

The ACD channel is automatically created and can be used immediately. The "MachineControl" master module can access the ACD structures of "Infeed" and "Unwinder" with *rACDInfeed* and *rACDUnwinder*.



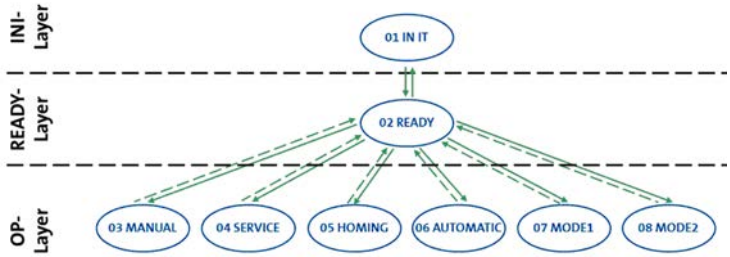
For extending the interface, variables can be entered in the MACD structures of the respective machine module:



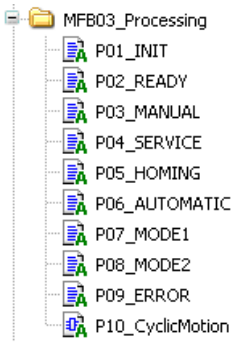
▸ Using the communication channel (📖 43)

4.15 Using operation modes

The operation modes for a machine module are predefined and cannot be extended. They are subdivided into the initialisation mode (01 INIT/INI-Layer), the standby mode (02 READY/READY-Layer), and six named work modes (03 MANUAL ... 08 MODE 2 / OP-Layer).



For each of the eight operation modes, the folder **MFB03_Processing** contains an assigned, predefined action which can be added to the program logic. The mode model for switching operation modes is the same in all machine modules and is controlled centrally via the master in most applications.



Each mode contains already predefined program codes. The user code can be written into this frame.

There are three areas into which a program code can be inserted within a mode:

1 "ModeEntry" is executed for one cycle when the mode is entered.

2 "Cyclic Area" is executed until the mode is changed over.

3 "ModeExit"' is executed when the mode is quit.

```
(* AT_ACTION_CREATE_NEW_MODULE *)
// First pulse, last pulse and cyclic program
1 IF xModeEntry THEN
    // Do init steps

    // Set ModeBusy
    MMD.xModeBusy := TRUE;

3 ELSIF xModeExit THEN
    // Do exit steps

2 ELSE
    // Do cyclic things

    // Reset own ModeBusy
    MMD.xModeBusy := FALSE;

    // Check if nobody is busy
    IF NOT OpModeControl.xIsBusy THEN
        // No module is busy
        ;
    END_IF
END_IF
```

Example

Program code for switching on the technology module (ElectricalShaftPosBase) of the infeed axis in the "READY" mode:

MFB_Infeed.P02_READY

```
(* AT_ACTION_CREATE_NEW_MODULE *)
// First pulse, last pulse and cyclic program
IF xModeEntry THEN
    // Do init steps

    // Set ModeBusy
    MMD.xModeBusy := TRUE;

ELSIF xModeExit THEN
    // Do exit steps

ELSE
    // Do cyclic things
    MMD.InfeedConveyor.xEnable := TRUE;
    MMD.InfeedConveyor.xRegulatorOn := TRUE;
    // Reset own ModeBusy
    IF MMD.InfeedConveyor.xAxisEnabled THEN
        MMD.xModeBusy := FALSE;
    END_IF

    // Check if nobody is busy
    IF NOT OpModeControl.xIsBusy THEN
        // No module is busy
        ;
    END_IF
END_IF
```

First, the axis is switched on in the "READY" mode. For this purpose, the inputs *xEnable* and *xRegulatorOn* are set to TRUE.

As long as the *xAxisEnabled* output is set to FALSE, the *xModeBusy* bit is not reset to FALSE. This means, as long as *xModeBusy* is set to TRUE, the change-over to another mode is blocked. This serves to prevent very easily that, for instance, the mode is changed over from "READY" to "AUTOMATIC" without all axes being ready for operation.

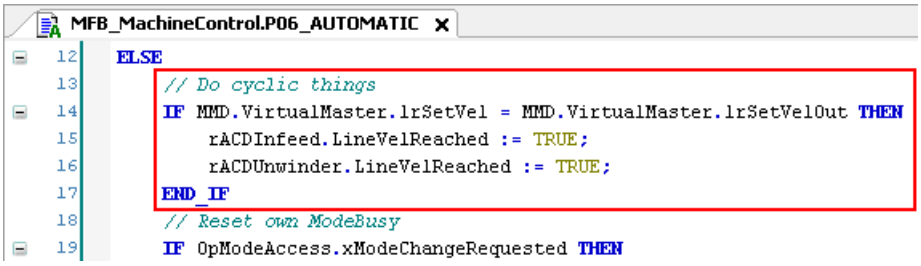
4.16 Using the communication channel

The ACD channel serves to exchange handshake signals between the master axis (MachineControl/ VirtualMaster) and the slave axes (Infeed, Unwinder).

The ACD channel only exists between the master and slave. Cross communication is not intended.

▸ Establishing the communication channel (ACD Slave Access) (📖 40)

The sample project uses the ACD channel to start the synchronisation to the master axis only when the master axis has reached its setpoint speed. For this purpose, a query has been programmed in the "Automatic" mode in the Cyclic area whether the *lrSetVel* setpoint velocity equals the *lrSetVelOut* output setpoint velocity:



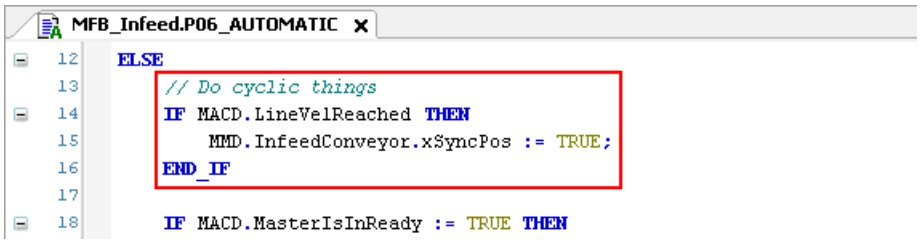
When the setpoint speed has been reached, the *LineVelReached* variable from the ACD channel is set to TRUE.

In the slave, this value is queried by the ACD channel and then the synchronisation is started.

From the master to the slave, the ACD channel is used via the *rACD[Slave instance name]* reference. If the *LineVelReached* bit is to be set for the "Infeed" slave, the program line would be:

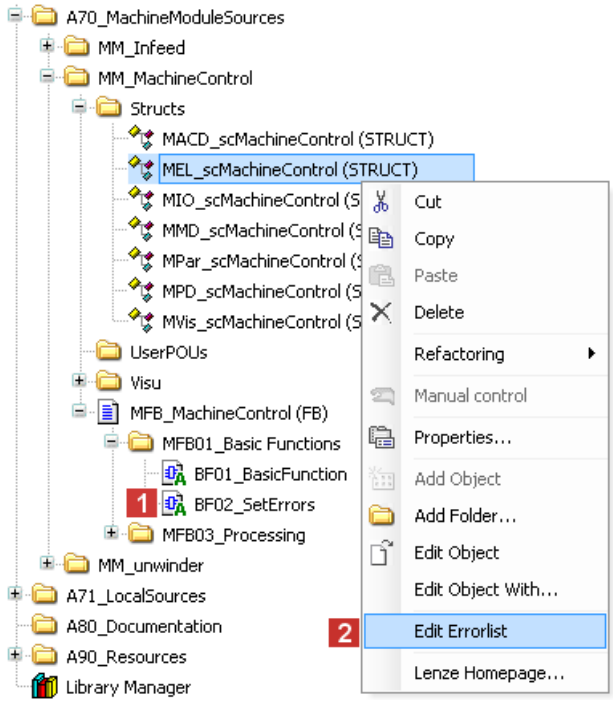
```
rACDZufuehrung.LineVelReached := TRUE;
```

In order to read this value in the slave, the MACD structure is accessed. Then the value is read out with `MACD.LineVelReached`:



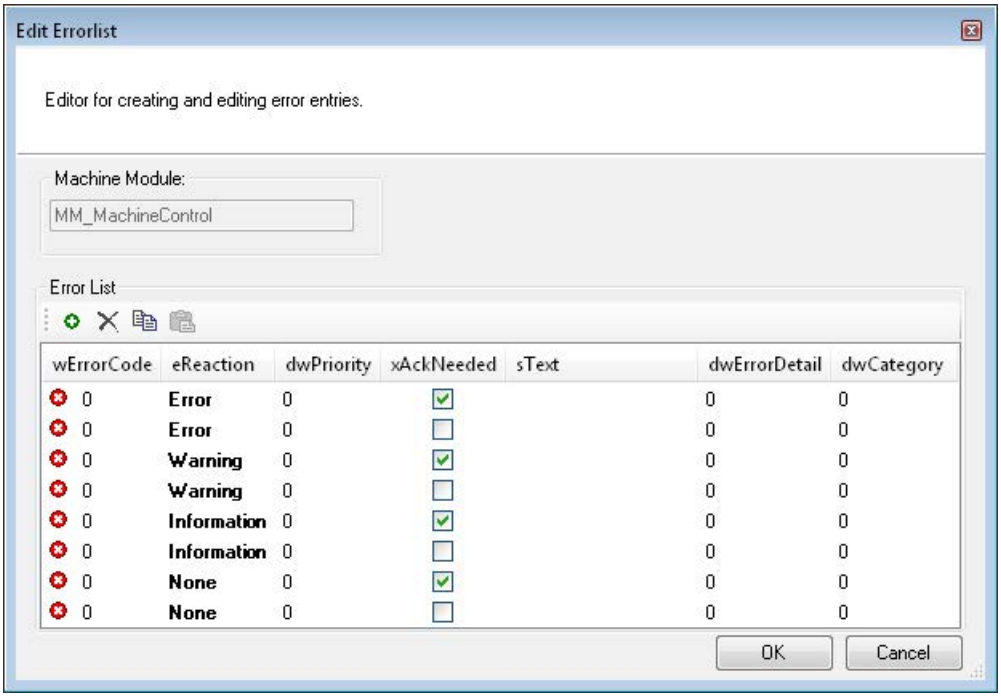
4.17 Creating and processing error messages

Each machine module comes with call modules for error handling and a structure for defining error messages, shown here using the example of the "MachineControl" master module:

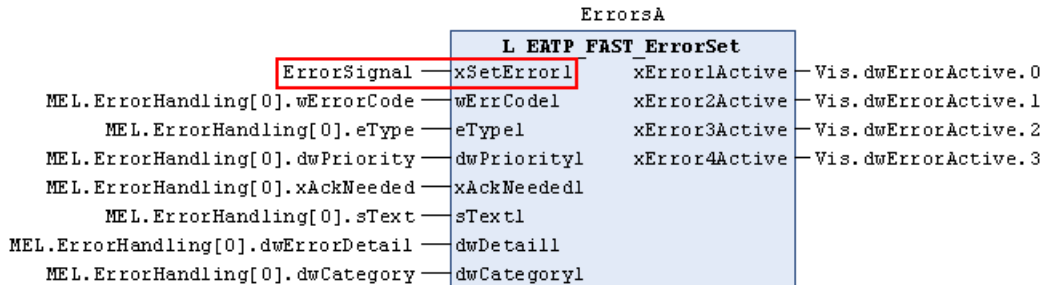


The **MFB01_BasicFunctions** folder contains the 1 **BF02_SetErrors** action which contains the call modules for error handling. The project template already includes two calls which enable 8 error messages to be triggered.

A right-click on the MEL structure and the 2 **"Edit Errorlist"** menu command allow you to add and delete error messages in a dialog window and define them with all properties required.



The triggering error signal is applied to the block under **BF02_SetErrors** at the *xSetError* input.



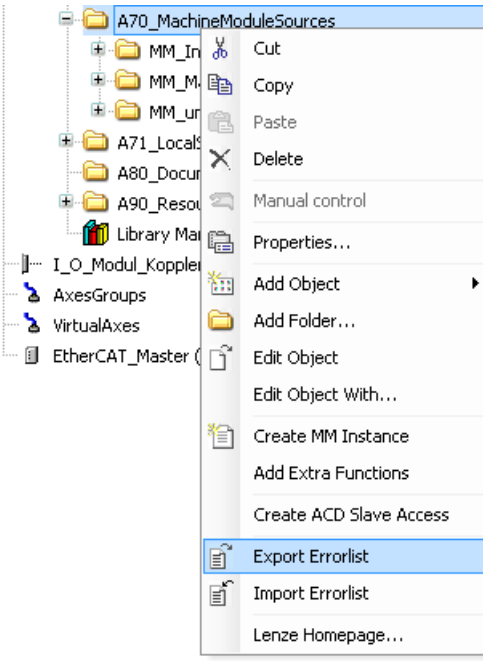
If the error signal is triggered, the "Error" response occurs. The P09_Error action is executed in parallel to the current operation mode.

4.17.1 Exporting the error list (XML file)

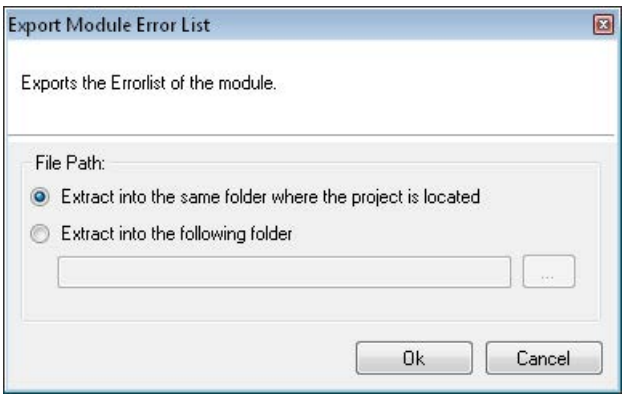
An error list is an XML file containing all errors defined in the individual machine modules located in folder **A70_MachineModuleSources**. The structured arrangement of the error information in the XML file corresponds to the order of the machine modules in the project folder.

How to proceed

1. Right-click the **A70_MachineModuleSources** folder and execute the **Export Errorlist** menu command.



2. Set the target directory for the XML file in the dialog that appears.

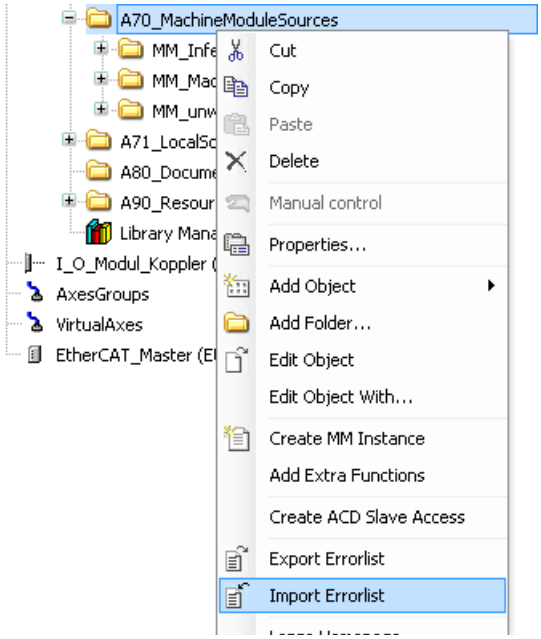


3. Click **OK** to execute the export.

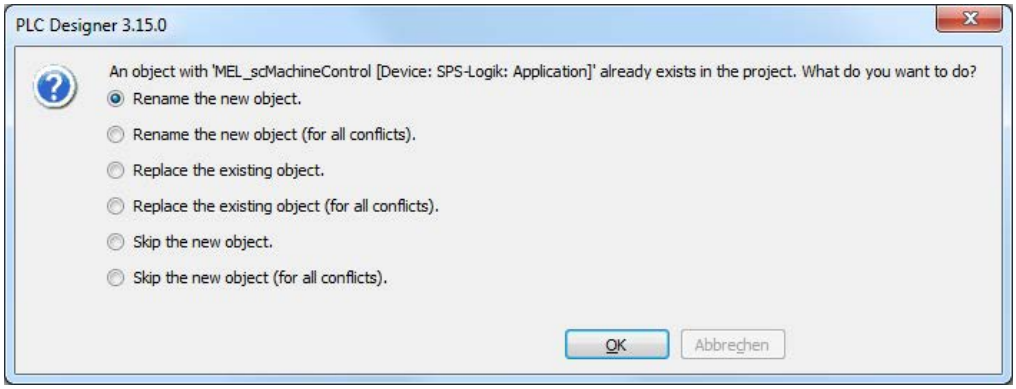
4.17.2 Importing the error list (XML file)

How to proceed

1. Right-click the **A70_MachineModuleSources** folder and execute the **Import Errorlist** menu command.



2. Select the source directory and the XML file to be imported in the dialog that appears.
3. Click on **Open** to execute the error list import.
4. Select actions for the error information:



For each module error list, one of the following actions can be selected:

Rename the new object: The error information contained in the XML file for this machine module will be saved under a new name in the project. This option should only be used in exceptional cases.

Replace the existing object: The error information on this machine module contained in the project will be overwritten with the values from the XML file. For most application scenarios in which the error information is processed with an external tool, this is the default action.

Skip the new object: The error information on this machine module contained in the XML file will be discarded. The error information in the project remains unchanged.

Note!

Generally, it is not a good idea to import an error list which was created with another arrangement and/or designation of machine modules in the folder A70_MachineModuleSources.

In the XML file, the attributes for each error entry are stored once as individual XML tags. Additionally, an XML tag "InterfaceAsPlainText" is output which contains the complete error information for a machine module as structured text (ST).

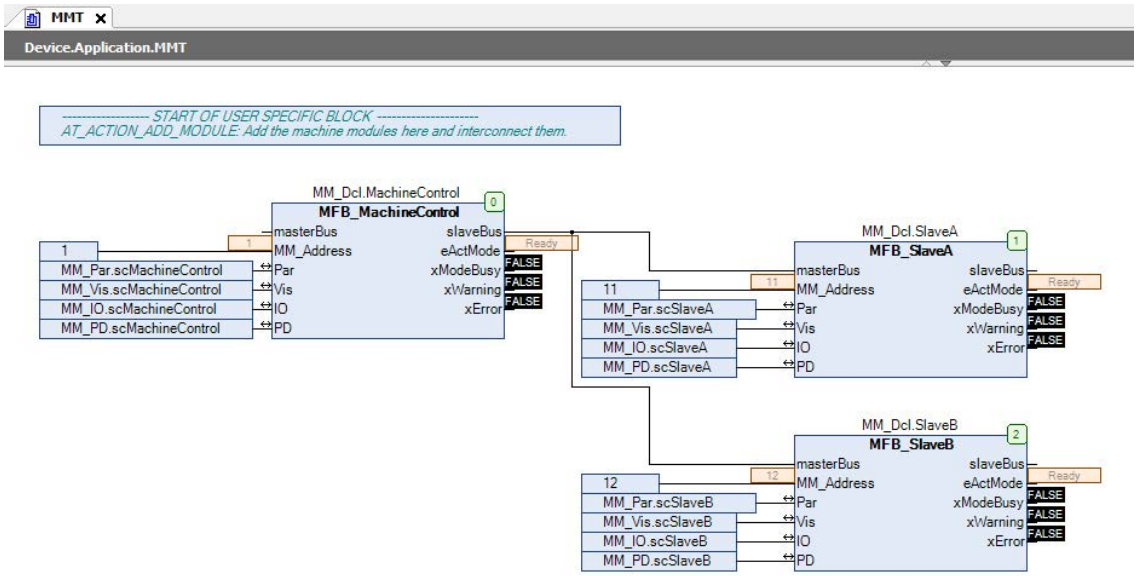
If the error attributes are to be processed with an external tool, processing must be performed within this XML tag. Only this XML tag will be considered when (re-)importing the error list.

4.17.3 Using module coupling

Uncoupling a machine module from its higher-level module results in the operation mode of the decoupled module no longer following the master, thereby allowing it to be defined independently.

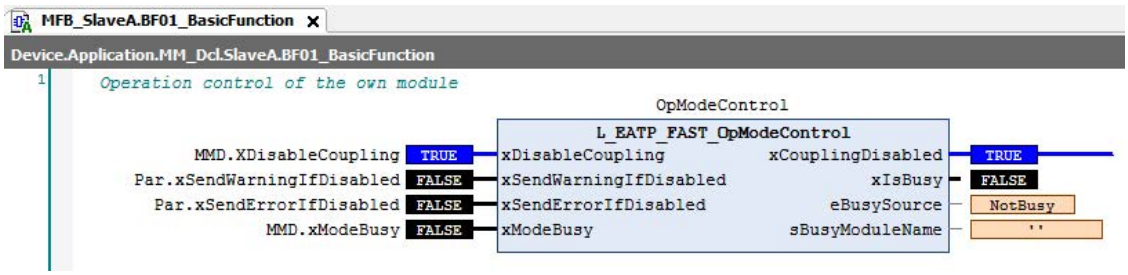
Example

The following structure consists of a machine module **MachineControl** and two subordinate modules **SlaveA** and **SlaveB**. The module **SlaveA** is to be decoupled.



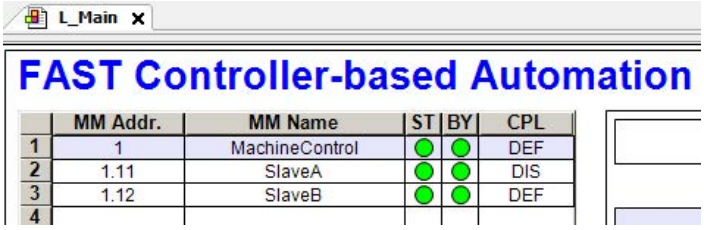
The signal *MMD.xDisableCoupling* in module **SlaveA** is set to TRUE.

This signal is at the input *xDisableCoupling* of block **OpModeControl** within the action *MFB_SlaveA.BF01_BasicFunction*.

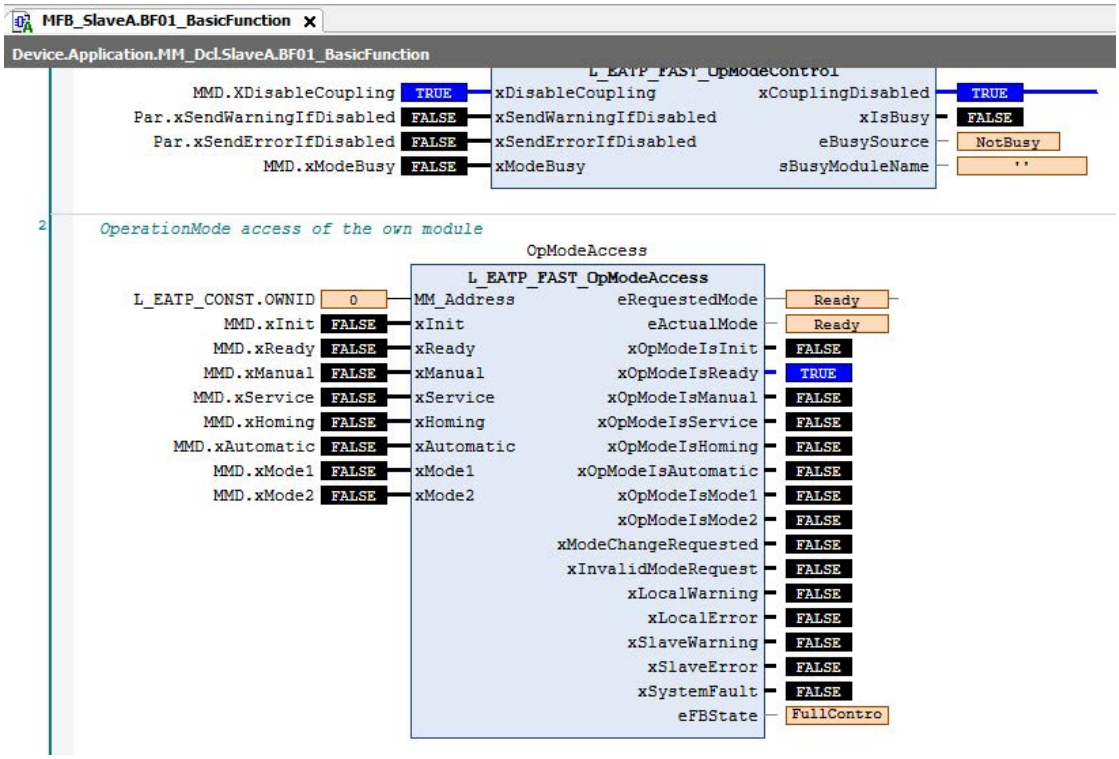


The output *xCouplingDisabled* in the same block shows the current status of the module coupling for the module.

The visualisation **L_Main** indicates "DIS" for the decoupled module **SlaveA** in the column "CPL" of the module list.

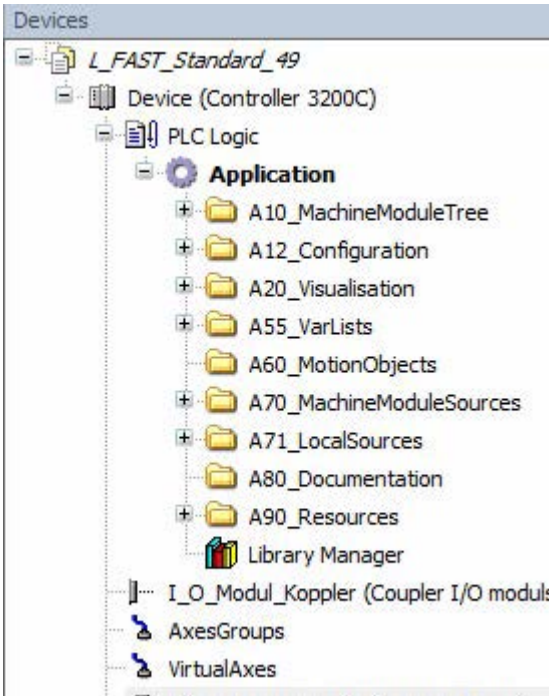


The operation mode of module **SlaveA** can now be controlled via the inputs of block **OpModeAccess** (action *MFB_SlaveA._ BF01_BasicFunction*) independently of the higher-level module **MachineControl**.



To restore the coupling, the signal *MMD.xDisableCoupling* is reset to the standard value FALSE. As soon as both machine modules are in the same operation mode, the coupling is restored.

5 Structure of the Application Template

Structure of the Application Template	
	A10_MachineModuleTree (📖 52) Linkage of the individual machine modules
	A12_Configuration (📖 54) Basic configuration of the Application Template
	A20_Visualisation (📖 55) Main visualisation of the Application Template for implementing the individual machine module visualisations
	A55_VarLists (📖 59) Declaration of the machine module instances and their structures
	A60_MotionObjects (📖 59) Motion-relevant data (e.g. CAM profiles)
	A70_MachineModuleSources (📖 60) Machine modules
	A71_LocalSources Machine-independent enumerations, function blocks, structures, visualisations
	A80_Documentation Documentation folder for storing documents for project history (e.g. version information, changes) and operating instructions etc.
	A90_Recources System information such as used libraries and task settings

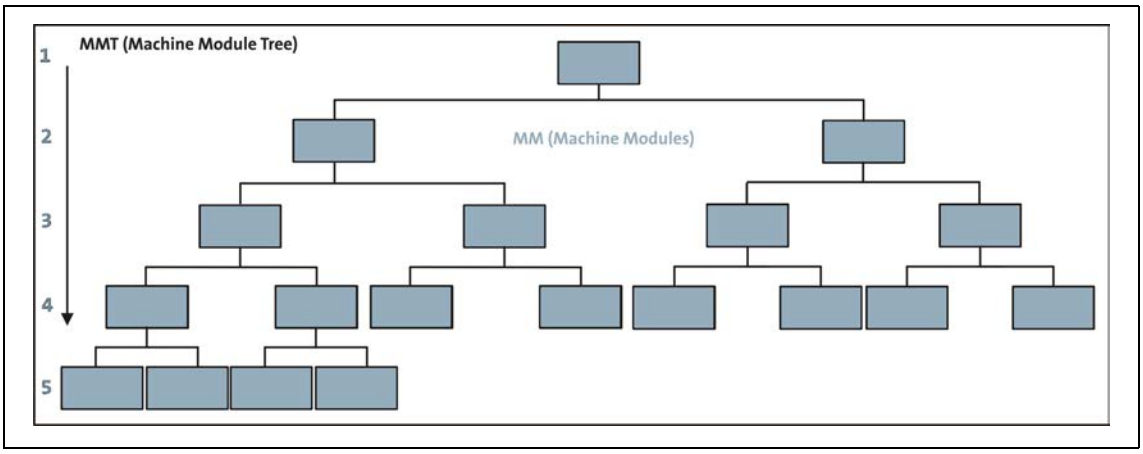
5.1 A10_MachineModuleTree

In order to map the desired automation system based on the Application Template in the »PLC Designer«, first you have to divide the total mechatronic functionality of the machine into individual subfunctions of the machine.

The subfunctions can be used to create individual reusable machine modules which can be mapped as a tree topology – the "Machine Module Tree". Here, the top module is the machine control module. All other machine modules are subordinated to this master module.

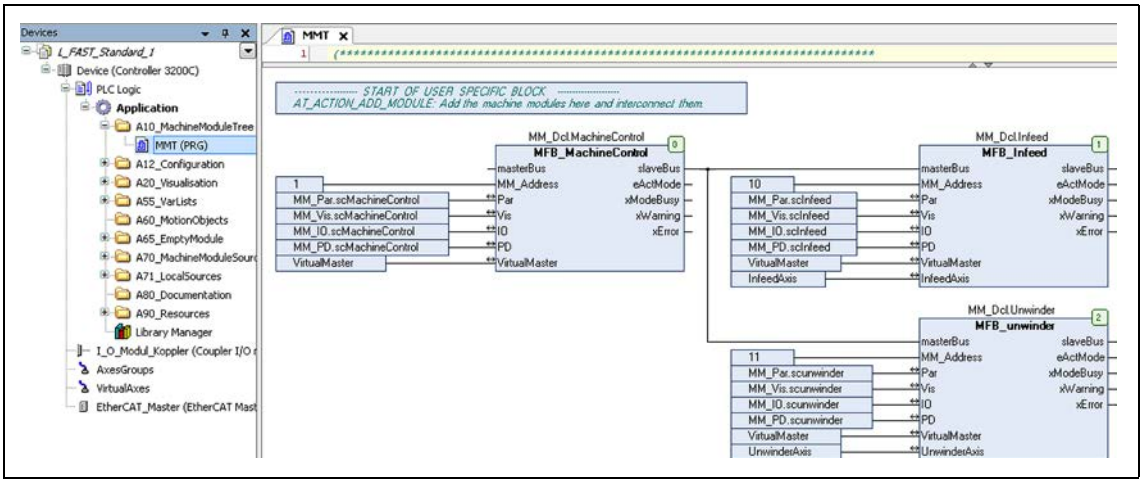
The Application Template supports ...

- two to five hierarchy levels of machine modules;
- up to 40 machine modules.



[5-1] Machine Module Tree (MMT) with five hierarchy levels

In the »PLC Designer« under the **A10_MachineModuleTree → MMT (PRG)** folder, the tree topology is displayed from left to right:

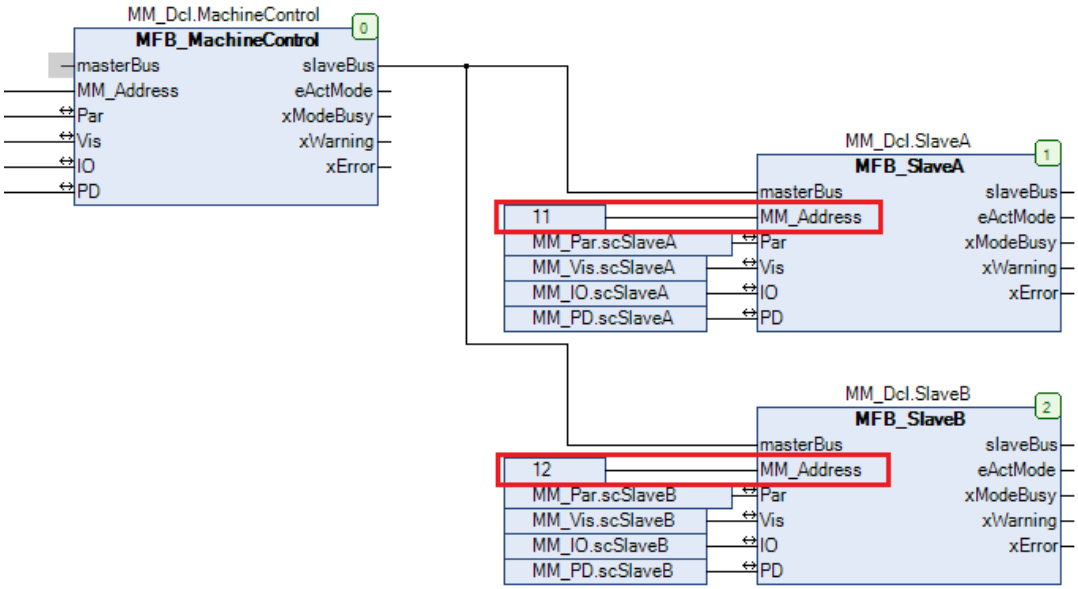


[5-2] Sample project: MMT (PRG)

▸ Creating machine modules (📖 16)
▸ Integrating machine modules into the Machine Module Tree (MMT) (📖 28)

MM_Address

Each machine module has an input *MM_Address* which needs to be assigned by the user. This address uniquely identifies it to its higher-level master module.



The following basic conditions apply when assigning addresses:

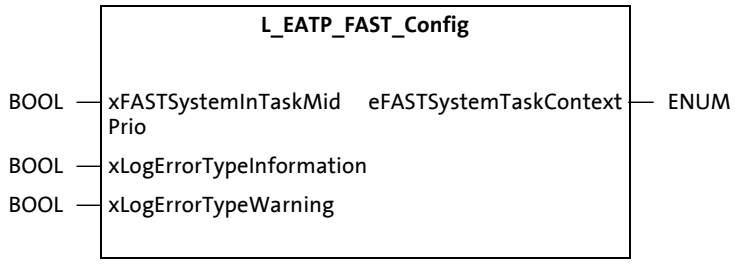
- The address is a positive integer (>0).
- All machine modules which are connected to the same higher-level master module must have different addresses.
- It is not necessary for the addresses to be consecutively numbered or start at 1.

5.2 A12_Configuration

Located in this folder is the function block **L_EATP_FAST_Config**, which allows a basic configuration of the Application Template to be performed.

Note!

Settings at this function block can only be made before the PLC is started and not during the runtime.



inputs

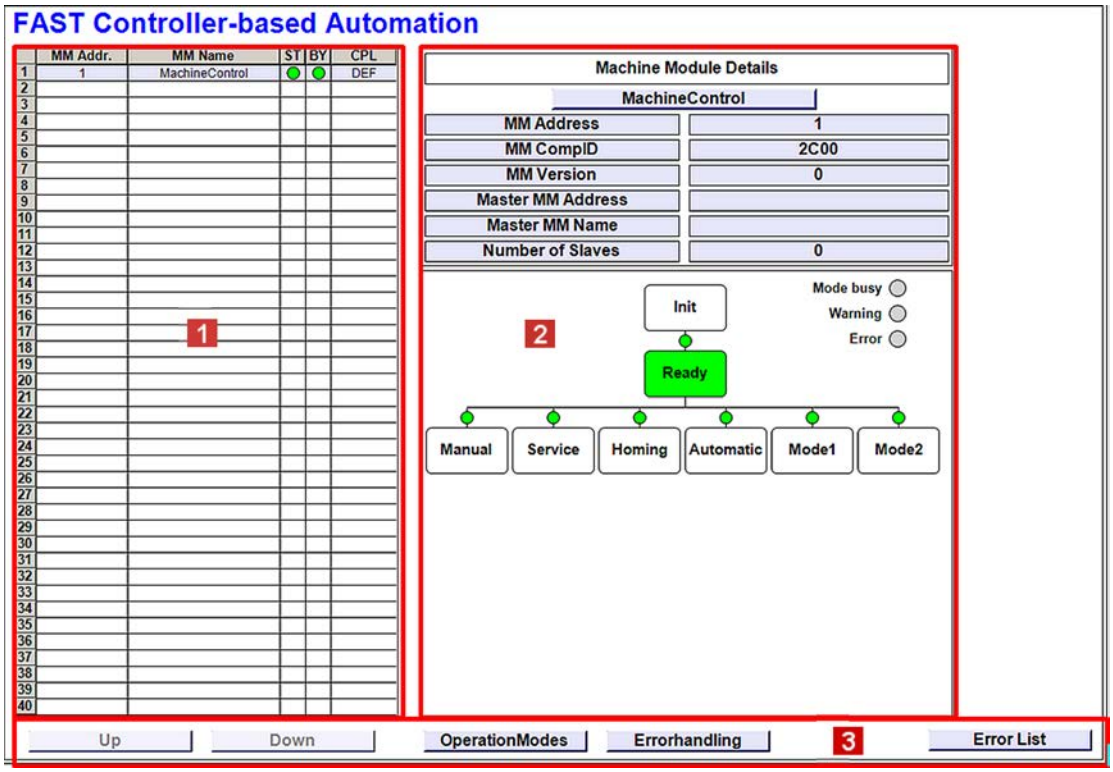
Designator Data type		Description
xFASTSystemInTaskMidPrio BOOL	TRUE	The logic and infrastructure functions of the Application Template are calculated in the mid task. This unloads the high task.
xLogErrorTypeInformation BOOL	TRUE	The information generated via the ErrorSe block (BF02_SetErrors) are entered in the logbook.
xLogErrorTypeWarning BOOL	TRUE	The warnings generated via the ErrorSet block (BF02_SetErrors) are entered in the logbook.

outputs

Designator Data type	Description
eFASTSystemTaskContext ENUM	Display of the task in which the logic and infrastructure functions of the Application Template are calculated.

5.3 A20_Visualisation

This folder contains the main visualisation **L_Main** in which all visualisations of the individual machine modules are added.



The main visualisation consists of three areas:

1 Machine module list

Display of all machine modules that are available in the Machine Module Tree.

MM Addr.: machine module address
MN Name: machine module name
ST: machine module status (green: no error; red: error pending)
BY: machine module status (green: ready for operation; red: not ready for operation)
CPL: machine module coupling mode

- "DEF": module coupling is switched on.
- "DIS": module coupling is switched off. The IC module has status "Internal Control".

The sequence in the list corresponds to the call sequence.

2 "Machine Module Details"

Display detailed information bout the module selected in the Machine Module List.

3 Buttons

OperationModes: activates "Operation Modes" details.
Errorhandling: activates "Error handling" details.
ErrorList: opens error list.
Up/Down: moves cursor up and down the list.

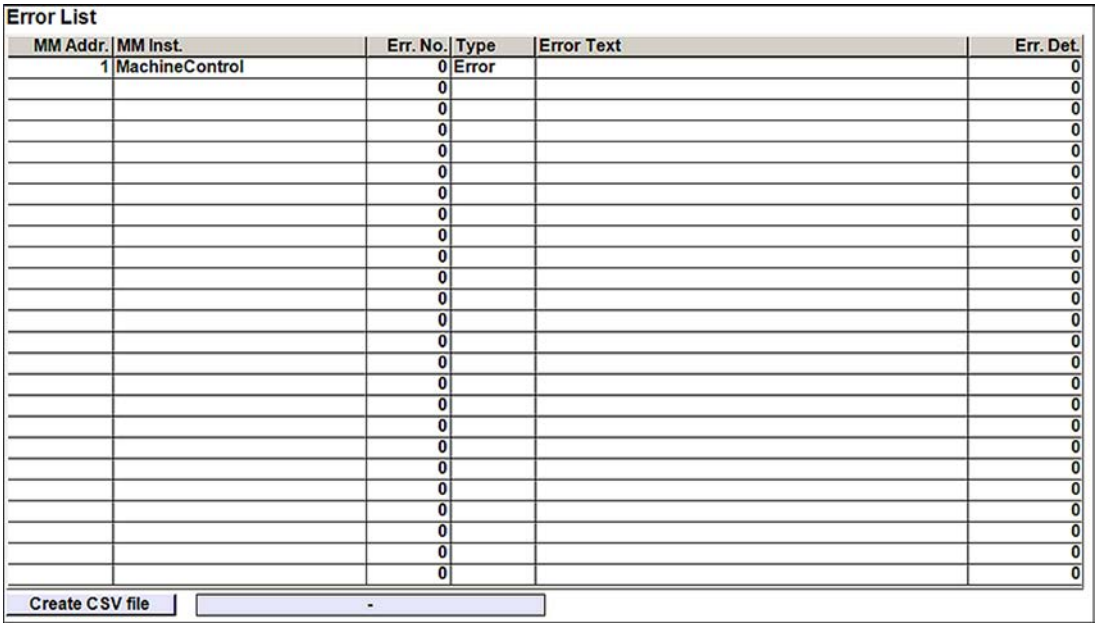
Generic visualisations

- L_EATP_FAST_VisErrorList (📖 56)
- L_EATP_FAST_VisModuleList (📖 57)
- L_EATP_FAST_VisModuleDetail (📖 58)

5.3.1 L_EATP_FAST_VisErrorList

This visualisation shows the contents of the *ErrorList* array.

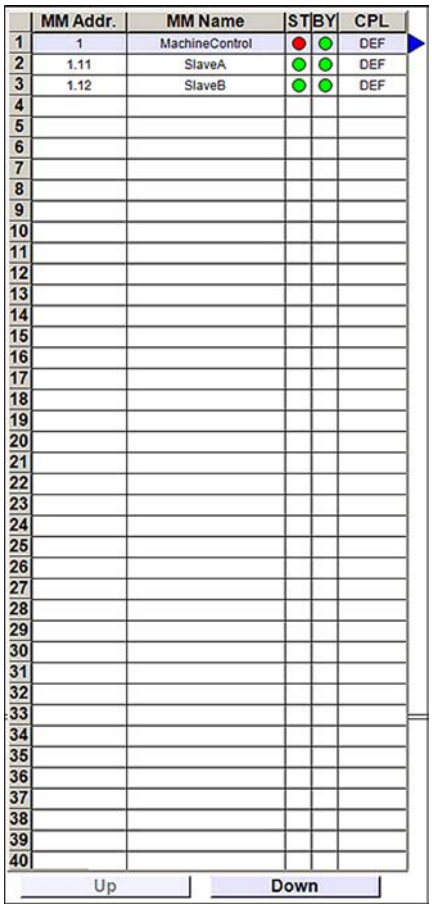
The **"Create CSV file"** button creates a CSV file with all the errors defined in the machine modules.



5.3.2 L_EATP_FAST_VisModuleList

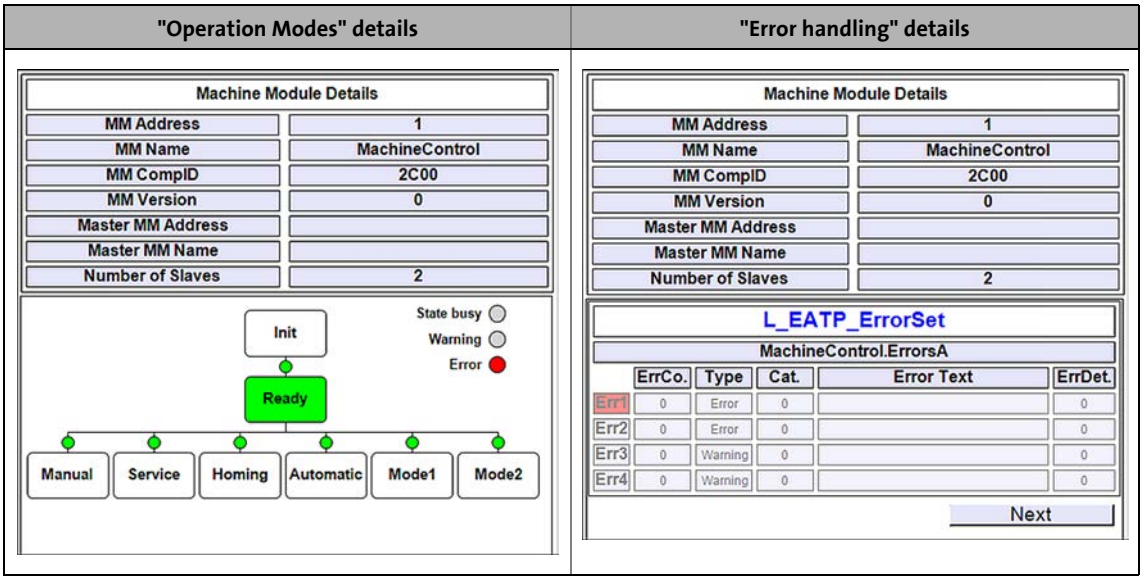
This visualisation shows all machine modules implemented in the Machine Module Tree.

The **"Up"** and **"Down"** buttons serve to move the selection mark in the list by one element upwards or downwards. A selection is also directly possible via mouse-click on a list element.



5.3.3 L_EATP_FAST_VisModuleDetail

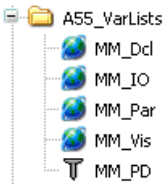
This visualisation shows the detailed information of a machine module which has been selected via the L_EATP_FAST_VisModuleList (📖 57) visualisation.



Previous/Next (in "Errorhandling" details): shows the previous/next instance of the function block L_EATP_FAST_ErrorSet.

5.4 A55_VarLists

The instances of the used machine modules and the respective structures are declared in the variable lists.

Variable lists		Description
	MM_Dcl	Declaration of the machine module instances The first instance of a machine module is created with the "Insert Empty Module" command. Further instances of machine modules can be created with the "Create MM Instance" command. Both commands can be executed via the context menu which appears with a right-click on the "A70_MachineModuleSources" folder.
	MM_IO	Declaration of instances of MIO structures in the machine modules
	MM_Par	Declaration of instances of MPar structures in the machine modules
	MM_Vis	Declaration of instances of MVis structures in the machine modules
	MM_PD	Declaration of instances of MPD structures in the machine modules

The structures (MIO, MPar, MVis, MPD) are automatically created as soon as an instance of a machine module has been created with the commands "Insert Empty Module" or "Create MM Instance".

5.5 A60_MotionObjects

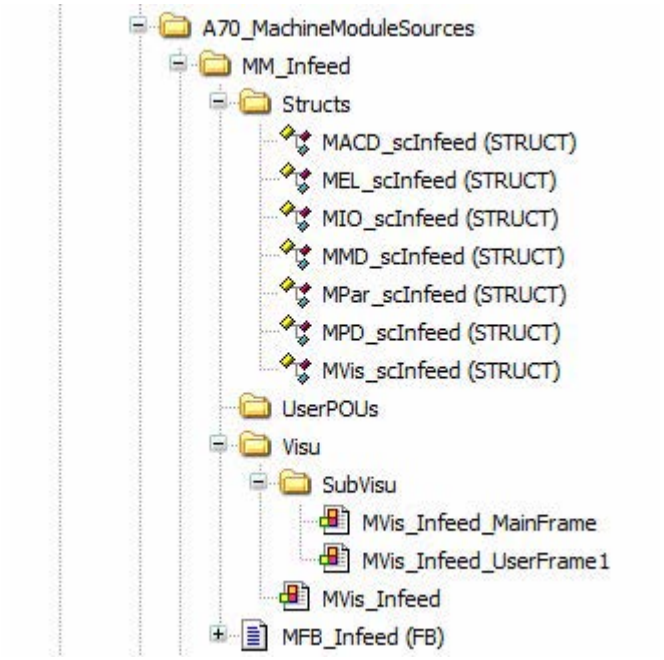
This folder is for storing your own function blocks, functions etc. which are used in the machine modules in the **A70_MachineModuleSources** folder.

If the folder is not required, it can be deleted.

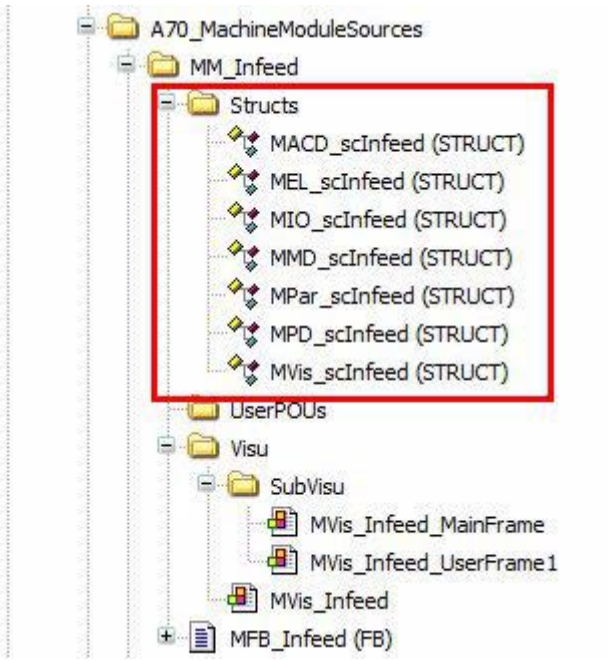
5.6 A70_MachineModuleSources

This folder contains previously created machine modules. See Creating machine modules (📖 16).

The command **"Load Machine Module"** is used to load machine modules into the project.



5.6.1 Structures



The **Structs** folder contains the structures of the machine module.

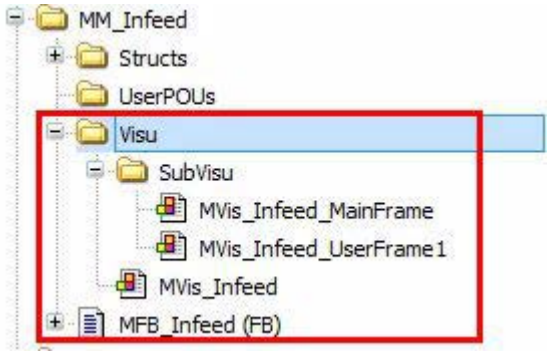
Structures	Description		
MMD_scInfeed	This structure serves to instance machine module data. This comprises all function blocks and variables that are used internally in the machine module for the application. These values can be accessed with "MMD.".		
MACD_scInfeed	This structure serves to exchange data between the master and slave machine module. In this structure, the following variables are predefined.		
	Variable	**Data type**	**Description**
	xStartOperation_In	BOOL	Control signal for "Start"
	xStopOperation_In	BOOL	Control signal for "Stop"
	xPauseOperation_In	BOOL	Control signal "Pause"
	xOperationBusy_Out	BOOL	Status signal "Busy"
	xOperationDone_Out	BOOL	Status signal "Done"
	xOperationPaused_Out	BOOL	Status signal "Paused"
	These values can be accessed by the master module with "rACDModulName" and in the slave module with "MACD". The "MACD." entry serves to display the contents of the structure by means of the Intellisense function. ▸ Establishing the communication channel (ACD Slave Access) (📖 40) ▸ Using the communication channel (📖 43)		
MEL_scInfeed	This structure contains the error entries that are connected to the errorset block. These error entries can be processed with the "Edit Errorlist" command. ▸ Creating and processing error messages (📖 44)		
MIO_scInfeed	This structure contains IO variables that are externally connected to the machine module.		
MPar_scInfeed	This structure contains parameters for parameterising the machine module.		
MPD_scInfeed	This structure contains persistent variables/parameters.		
MVis_scInfeed	This structure contains visualisation variables which are automatically written into the symbol configuration.		

5.6.2 User POUs

In this folder, function blocks, functions etc. can be stored which are especially used in the machine module

If the folder is not required, it can be deleted.

5.6.3 visualisations



The **Visu** folder contains the main visualisation **MVis_Infeed** and the sub-visualisations **MVis_Infeed_MainFrame** and **MVis_Infeed_UserFrame1**.

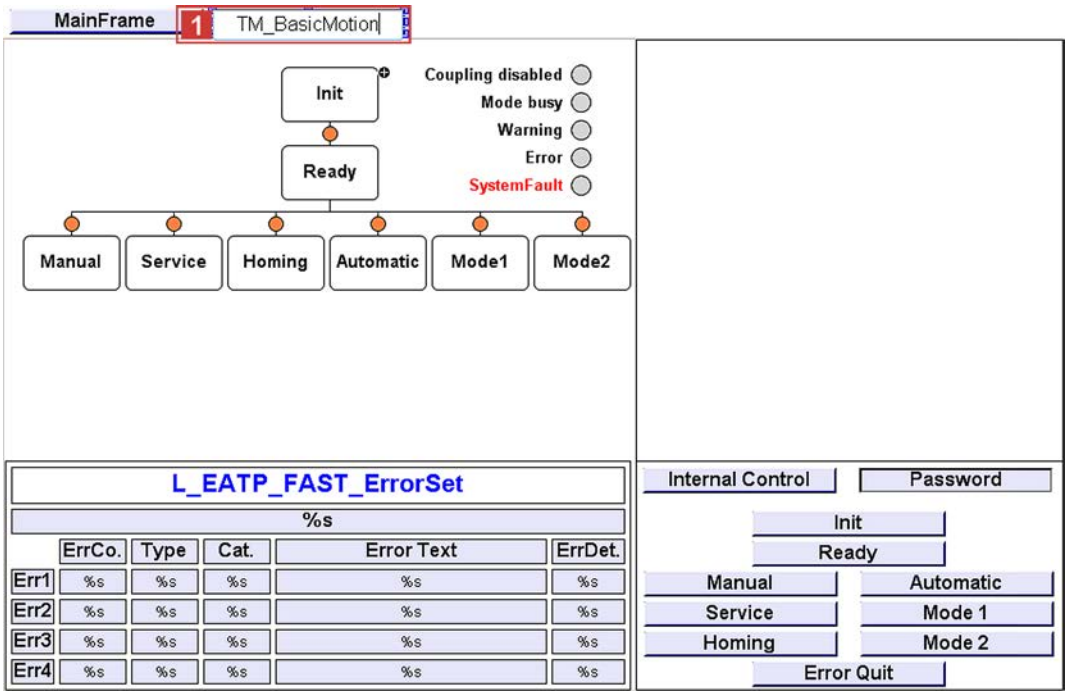
The visualisation **MVis_Infeed_UserFrame1** can be used to add custom content (e.g. the visualisation for a technology module).

The main visualisation **MVis_Infeed** is prepared such that it can be switched over to "User Frame".

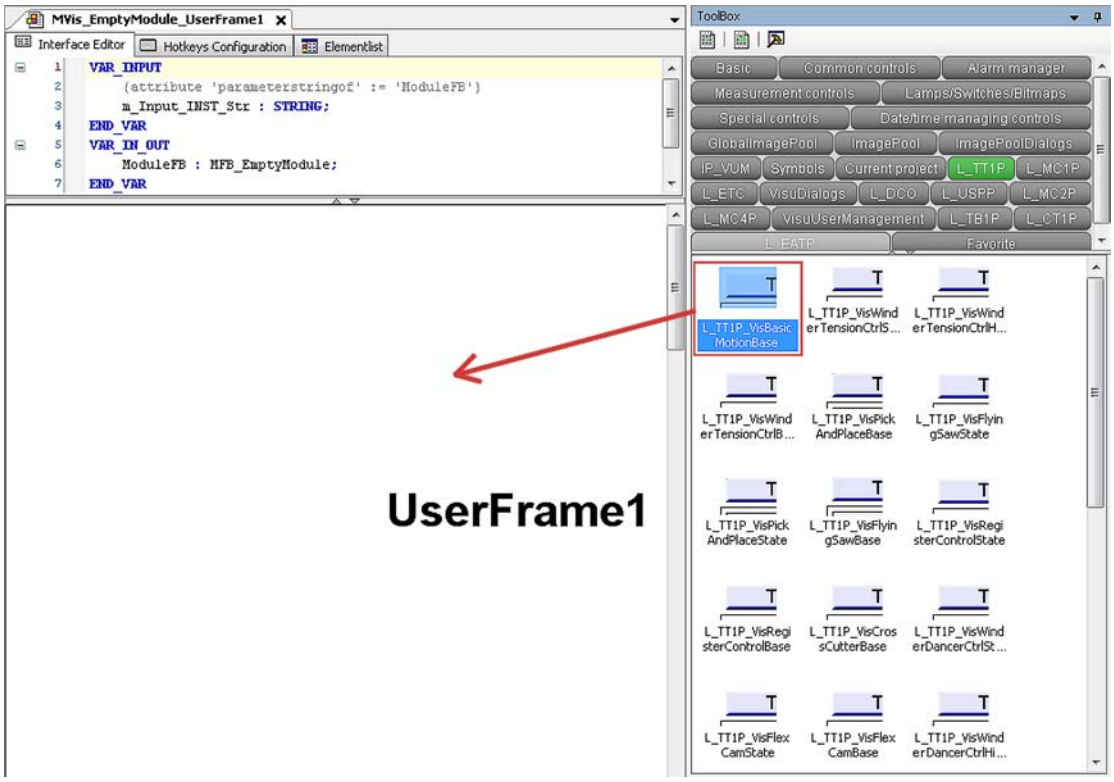
How to insert the visualisation of a technology module:

1. Open the main visualisation **MVis_[ModuleName]** with a double-click.
2. Rename the 1 **"UserFrame1"** button.

   In this case "TM_BasicMotion" for the "Basic Motion" technology module:

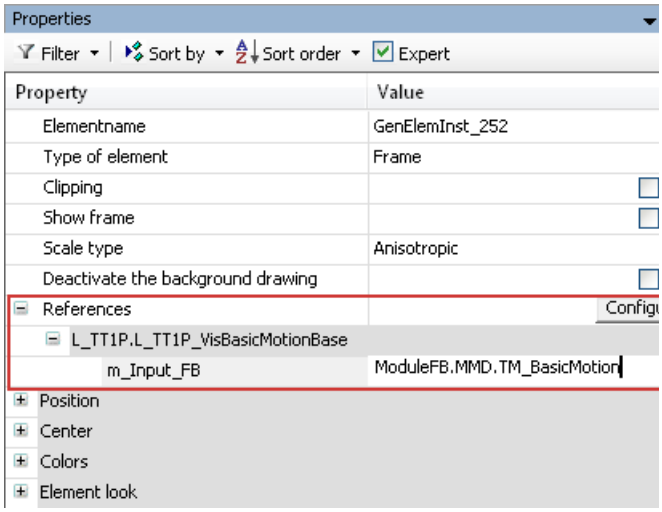


3. Open the **MVis_[ModuleName]_UserFrame1** visualisation with a double-click.
4. Move the visualisation of the technology module into the "UserFrame1" in the tools under **L_TT1P**.

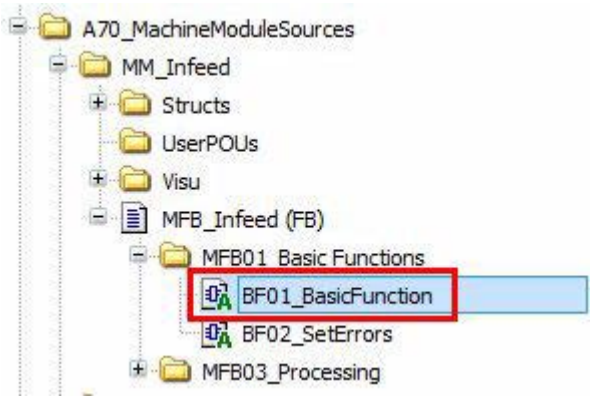


5. In a final step, indicate the reference of the instance of the technology module under the properties of the visualisation.

   In this case "ModuleFB.MMD.TM_BasicMotion" for the "Basic Motion" technology module:



5.6.4 BF01_BasicFunction



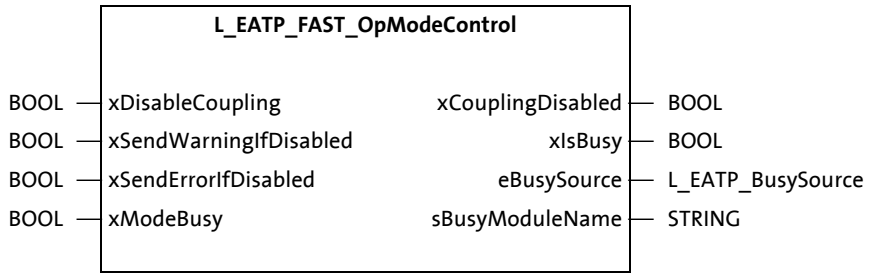
The **BF01_BasicFunction** action in the **MFB01_BasicFunctions** module folder contains the interface of the application program in a machine module to the application template system functions.

Note!

All inputs of the interface block described in the following are connected to data elements with the same name in the MMD structure (cp. Structures (📖 61)). These data elements serve to control system functions in the application program.

5.6.4.1 L_EATP_FAST_OpModeControl

Instances of this function block map the interface for configuring the operation mode model for mode changes.



inputs

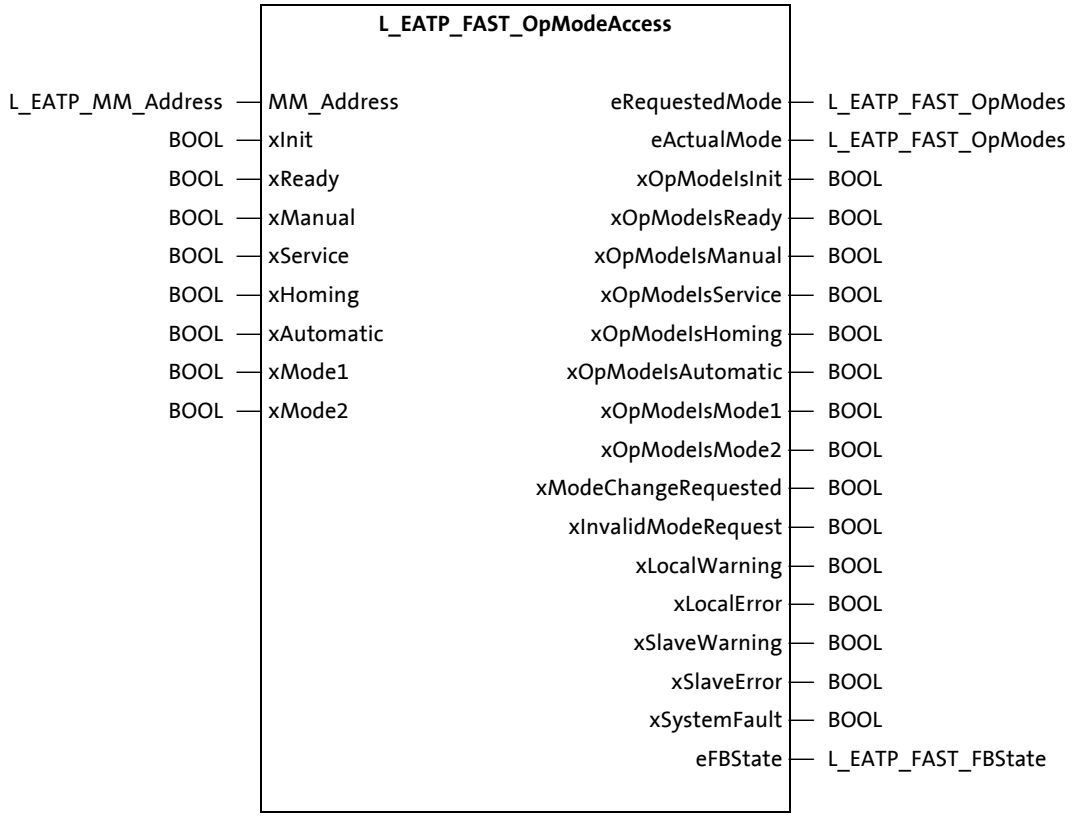
Designator Data type		Description
xDisableCoupling BOOL	FALSE ↗ TRUE	The standard coupling of the machine module to the master module is deactivated.
	TRUE↘ FALSE	The activation of the standard coupling of the machine module to the master module is enabled. **Note:** Coupling is only activated if both modules are in the same operation mode!
xSendWarningIfDisabled BOOL	TRUE	If the coupling of the machine module to the master module is deactivated, warnings are still forwarded to the master module.
	FALSE	If the coupling of the machine module to the master module is deactivated, warnings are not forwarded to the master module.
xSendErrorIfDisabled BOOL	TRUE	If the coupling of the machine module to the master module is deactivated, error messages are still forwarded to the master module.
	FALSE	If the coupling of the machine module to the master module is deactivated, error messages are not forwarded to the master module.
xSendErrorIfDisabled BOOL	TRUE	The machine module is in the "ModeBusy" state, i.e. mode changes are disabled.
	FALSE	The machine module is not in the "ModeBusy" state, i.e. mode changes are enabled.

outputs

Designator Data type		Description
xCouplingDisabled BOOL	TRUE	The standard coupling of the machine module to the master module is deactivated.
xIsBusy BOOL	TRUE	The machine module itself or one of its lower-level and coupled slave modules is in the "ModeBusy" state.
eBusySource L_EATP_BusySource	One of the following enumeration constants: • "NotBusy": No machine module is in the "ModeBusy" state. • "OwnModuleIsBusy": The local machine module is in the "ModeBusy" state. • "SlaveIsBusy": At least one coupled slave module is in the "ModeBusy" state.	
sBusyModuleName STRING	Instance name of the machine module which is in the "ModeBusy" state. **Note:** If several slave modules are in the "ModeBusy" state, the name of the slave module found first in the call hierarchy is indicated.	

5.6.4.2 L_EATP_FAST_OpModeAccess

Instances of this function block map the interface for changing over the operation modes.



inputs

Note!

At any time, only maximally one block input *xInit* ... *xMode2* may be connected to TRUE.

Designator Data type	Description	
MM_Address L_EATP_MM_Address	Module address of the target module Either L_EATP_CONST.OWNID for the local machine module or a valid address of a slave module	
xInit BOOL	TRUE	A change to the "INIT" mode is requested.
xReady BOOL	TRUE	A change to the "READY" mode is requested.
xManual BOOL	TRUE	A change to the "MANUAL" mode is requested.
xService BOOL	TRUE	A change to the "SERVICE" mode is requested.
xHoming BOOL	TRUE	A change to the "HOMING" mode is requested.
xAutomatic BOOL	TRUE	A change to the "AUTOMATIC" mode is requested.

Designator Data type	Description	
xMode1 BOOL	TRUE	A change to the "MODE1" mode is requested.
xMode2 BOOL	TRUE	A change to the "MODE2" mode is requested.

outputs

Designator Data type	Description	
eRequestedMode L_EATP_FAST_OpModes	The requested mode in the addressed machine module	
eActualMode L_EATP_FAST_OpModes	The active mode in the addressed machine module	
xOpModeIsInit BOOL	TRUE	The addressed module is in the "INIT" mode.
xOpModeIsReady BOOL	TRUE	The addressed module is in the "READY" mode.
xOpModeIsManual BOOL	TRUE	The addressed module is in the "MANUAL" mode.
xOpModeIsService BOOL	TRUE	The addressed module is in the "SERVICE" mode.
xOpModeIsHoming BOOL	TRUE	The addressed module is in the "HOMING" mode.
xOpModeIsAutomatic BOOL	TRUE	The addressed module is in the "AUTOMATIC" mode.
xOpModeIsMode1 BOOL	TRUE	The addressed module is in the "MODE1" mode.
xOpModeIsMode2 BOOL	TRUE	The addressed module is in the "MODE2" mode.
xModeChangeRequested BOOL	TRUE	A mode change has been requested.
xInvalidModeRequest BOOL	TRUE	There is an invalid request for a mode change. Possible causes: • More than on input of xInit ... xMode2 is set to TRUE. • The addressed module is in the standard coupling to its master module. The request of a mode change is only possible for the master module. • The FB instance is in the passive mode and there is a mode change request (see also eFBState output).
xLocalWarning BOOL	TRUE	A warning is active at the addressed module.
xLocalError BOOL	TRUE	An error message is active at the addressed module.
xSlaveWarning BOOL	TRUE	A warning is active at a lower-level slave module.
xSlaveError BOOL	TRUE	An error message is active at a lower-level slave module.

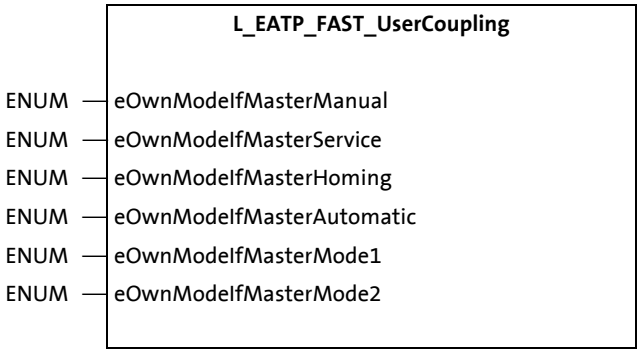
Designator Data type	Description	
xSystemFault BOOL	TRUE	The "SystemFault" state is active.
eFBState L_EATP_FAST_FBState	One of the following enumeration constants: • "FullControl": The FB instance has the full functional range. • "PassiveInternalControl": The FB instance is in passive mode (the block inputs are deactivated), because "InternalControl" is active for the machine module. • "PassiveCoupledToMaster": The FB instance is in passive mode because the machine module has a controlling master module. • "PassiveSecondInstance": The FB instance is in passive mode because it is a second or following instance for the addressed module. • "PassiveMasterOverride": The coupling of the module to its master module is deactivated. A block instance, however, is contained in the master module which executes the mode control.	

5.6.4.3 L_EATP_FAST_UserCoupling

Instances of this function block serve to change to a different mode in certain operation modes (depending on the mode of the master module).

Note!

- The pre-assignment of the data elements in the MMD structure is selected in such a way that in case of the slave module always the operation mode of the master module is selected.
- The operation modes "INIT" and "READY" of the master module are always accepted and cannot be changed over.



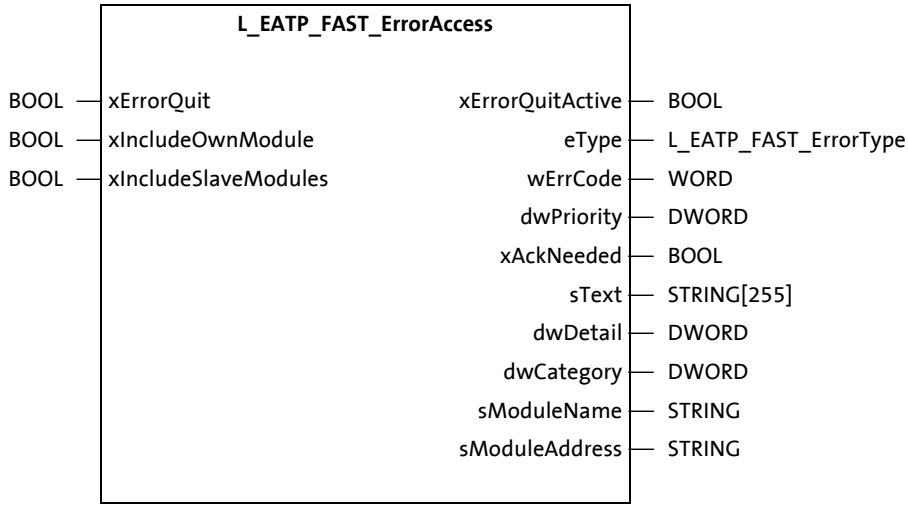
inputs

Designator Data type	Description
eOwnModeIfMasterManual ENUM	Defining the operation mode of the local machine module if the master module is in the "MANUAL" mode.
eOwnModeIfMasterService ENUM	Defining the operation mode of the local machine module if the master module is in the "SERVICE" mode.
eOwnModeIfMasterHoming ENUM	Defining the operation mode of the local machine module if the master module is in the "HOMING" mode.
eOwnModeIfMaster Automatic ENUM	Defining the operation mode of the local machine module if the master module is in the "AUTOMATIC" mode.
eOwnModeIfMasterMode1 ENUM	Defining the operation mode of the local machine module if the master module is in the "MODE1" mode.
eOwnModeIfMasterMode2 ENUM	Defining the operation mode of the local machine module if the master module is in the "MODE2" mode.

5.6.4.4 L_EATP_FAST_ErrorAccess

Instances of this function block map the following functions:

- Error acknowledgement
- Detection of the error information on the current error with the highest priority of the local machine module and/or its subordinate slave modules



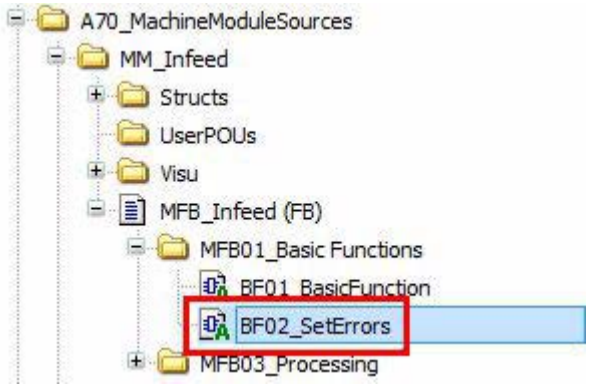
inputs

Designator Data type	Description	
xErrorQuit BOOL	FALSE ↗ TRUE	The errors in the own machine module and in all subordinate slave modules are acknowledged.
xIncludeOwnModule BOOL	TRUE	The local machine module is considered when the error with the highest priority is detected.
	FALSE	The local machine module is ignored when the error with the highest priority is detected.
xIncludeSlaveModules BOOL	TRUE	The subordinate slave modules are considered when the error with the highest priority is detected.
	FALSE	The subordinate slave modules are ignored when the error with the highest priority is detected.

outputs

Designator Data type	Description	
xErrorQuitActive BOOL	FALSE ↗ TRUE	The error acknowledgement is active. **Note:** This signal should be used for the error acknowledgement of local FB instances.
eType L_EATP_FAST_ErrorType	Error type of the current error with the highest priority	
wErrCode WORD	Error number of the current error with the highest priority	
dwPriority DWORD	Error priority of the current error with the highest priority	
xAckNeeded BOOL	Current error with the highest priority is subject to acknowledgement	
	TRUE	Acknowledgement required
	FALSE	Acknowledgement not required
sText STRING[255]	Error text of the current error with the highest priority	
dwDetail DWORD	Error detail of the current error with the highest priority	
dwCategory DWORD	Error category of the current error with the highest priority	
sModuleName STRING	Instance name of the machine module in which the current error with the highest priority is active.	
sModuleAddress STRING	Global address of the machine module in which the current error with the highest priority is active.	

5.6.5 BF02_SetErrors



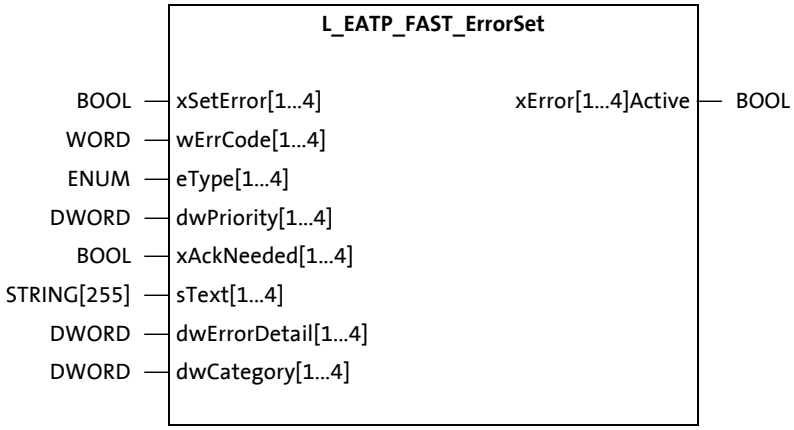
The **BF02_SetErrors** action in the **MFB01_BasicFunctions** module folder contains the predefined instances "ErrorsA" and "ErrorsB" of the **L_EATP_FAST_ErrorSet** function block for the error handling and the triggering of 8 error messages.

When the error occurs, the associated error message is entered into the logbook.

▸ A12_Configuration (🕮 54)

A direct programming or configuration usually does not take place in this action but by means of the **"Edit Errorlist"** command.

▸ Creating and processing error messages (🕮 44)



Note!

The inputs are connected to data elements with the same name in the MEL structure.

The outputs are connected to data elements with the same name in the MVis structure.

▸ Structures (🕮 61)

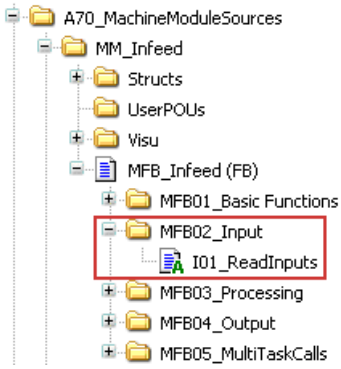
inputs

Designator Data type	Description	
xSetError[1...4] BOOL	FALSE ↗ TRUE	An error is active.
wErrCode[1...4] WORD	Error number of the current error with the highest priority	
eType[1...4] L_EATP_FAST_ErrorType	Error type of the current error with the highest priority	
dwPriority[1...4] DWORD	Error priority of the current error with the highest priority	
xAckNeeded[1...4] BOOL	Current error with the highest priority is subject to acknowledgement	
	TRUE	Acknowledgement required
	FALSE	Acknowledgement not required
sText[1...4] STRING[255]	Error text of the current error with the highest priority	
dwDetail[1...4] DWORD	Error detail of the current error with the highest priority	
dwCategory[1...4] DWORD	Error category of the current error with the highest priority	

outputs

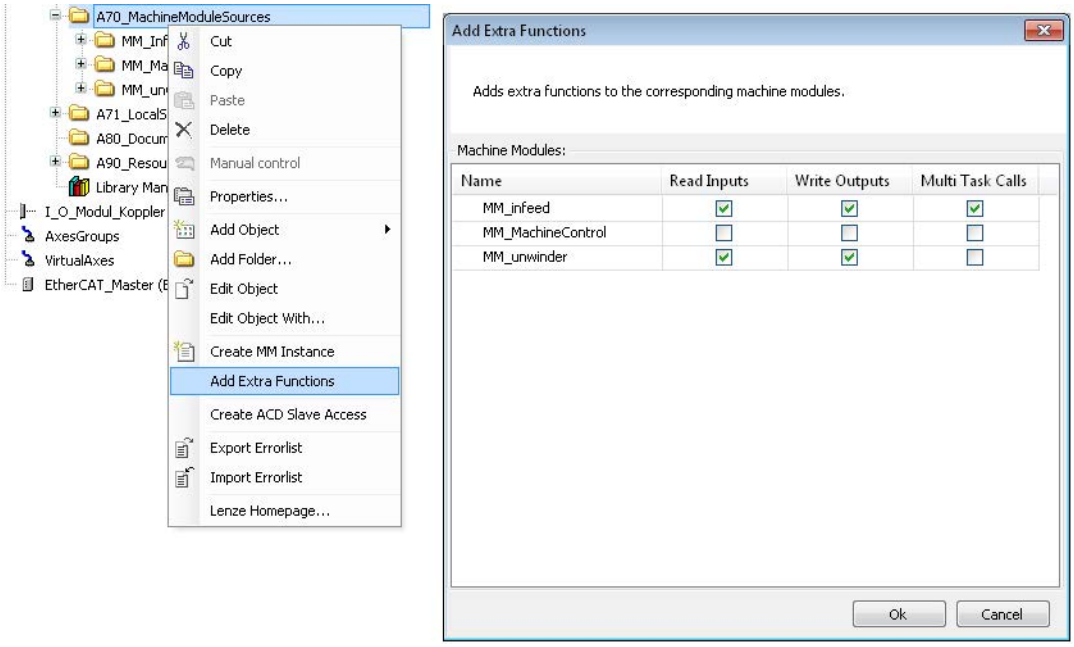
Designator Data type	Description	
xError[1...4]Active BOOL	TRUE	Output to the MVis structure (*dwErrorActive* variable) that an error is active.

5.6.6 I01_ReadInputs

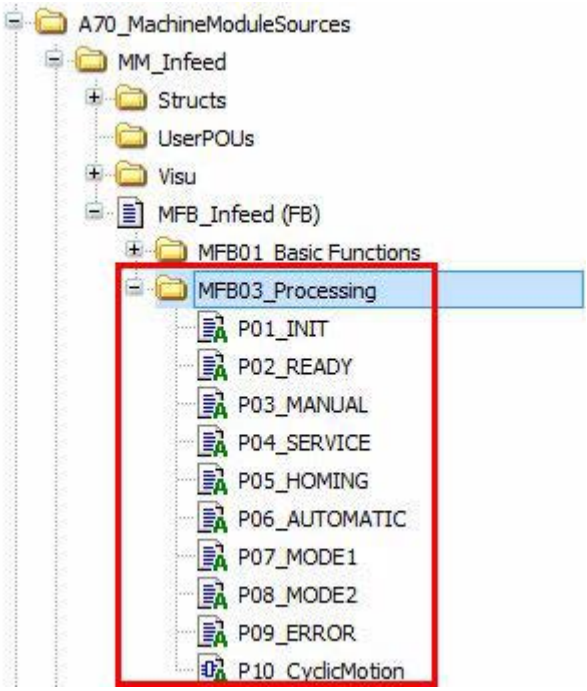


The **I01_ReadInputs** action in the **MFB02_Input** module folder can be used as structuring aid if the IPO model is used (input – processing – output). A detailed description of the IPO model can be found in the section MFB03_Processing (📖 76).

At first, the **MFB02_Input** module folder is not visible in the project tree. If required, it can be shown/hidden with the **"Add Extra Functions"** command under **A70_MachineModuleSources** if the IPO model is to be used. For this purpose, checkmarks can be set or removed under the appearing "Read Inputs" dialog.



5.6.7 MFB03_Processing



This module folder summarises the relevant actions that are intended for defining the application program. The actions are divided into ...

- Mode-related actions (📖 77) "P01_INIT" ... "P08_MODE2",
- Error action (📖 77) "P09_ERROR",
- Cyclic action (📖 77) "P10_CyclicMotion".

5.6.7.1 Mode-related actions

Depending on the active mode of a machine module, the related mode action **P01_INIT ... P08_MODE2** is passed through in each program cycle.

Each mode action contains an already predefined program structure to which the program code can be added.

There are three areas into which a program code can be inserted within a mode:

1 "ModeEntry" is executed for one cycle when the mode is entered.

2 "Cyclic Area" is executed until the mode is changed over.

3 "ModeExit"' is executed when the mode is quit.

```
1  (* AT_ACTION_CREATE_NEW_MODULE *)
2  // First pulse, last pulse and cyclic program
3  IF xModeEntry THEN
4      // Do init steps
5
6      // Set ModeBusy
7      MMD.xModeBusy := TRUE;
8
9  ELSIF xModeExit THEN
10     // Do exit steps
11
12 ELSE
13     // Do cyclic things
14
15     // Reset own ModeBusy
16     MMD.xModeBusy := FALSE;
17
18     // Check if nobody is busy
19     IF NOT OpModeControl.xIsBusy THEN
20         // No module is busy
21         ;
22     END_IF
23 END_IF
24
```

▸ Using operation modes (📖 41)

5.6.7.2 Error action

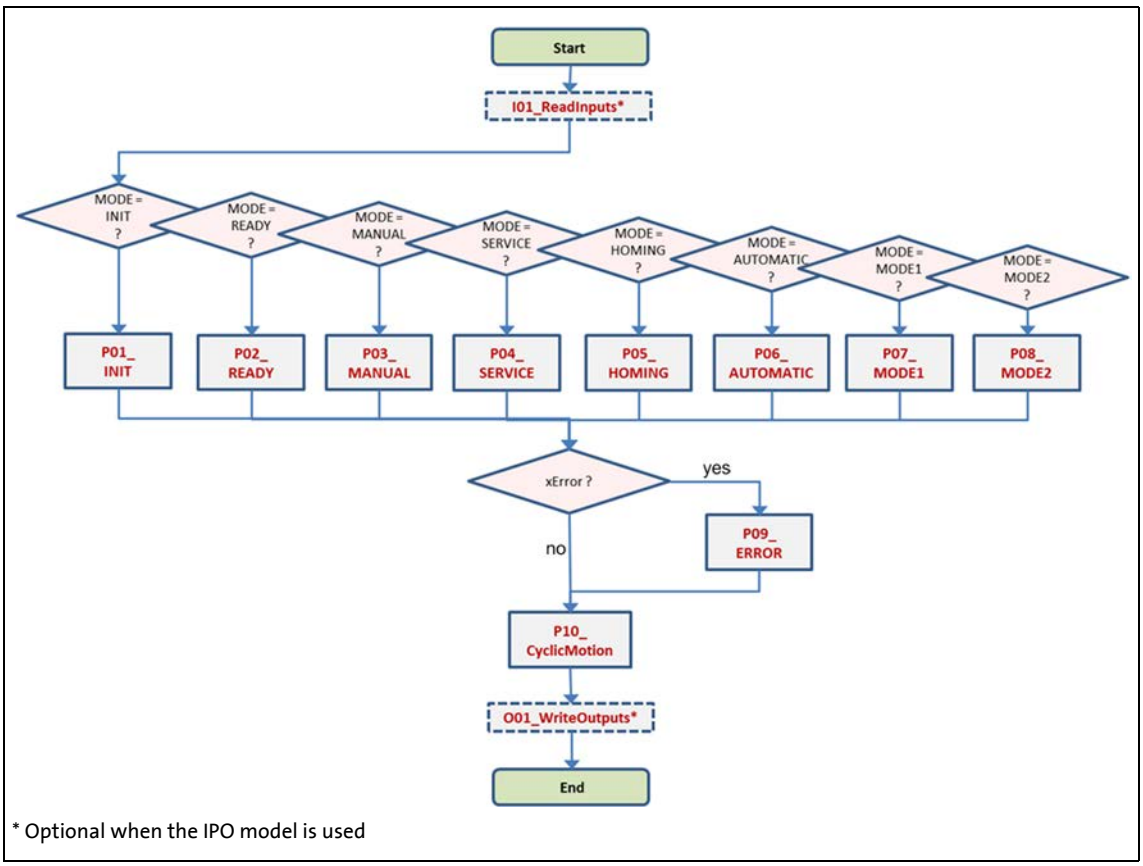
As long as an error is active in the machine module, the **P09_ERROR** action is called in each program cycle.

In case of active warnings, this action is not passed through.

5.6.7.3 Cyclic action

The **P10_CyclicMotion** action is called just once in each program cycle of the "HighPriority" task. Among other things, it serves to include the calls of motion blocks and other application parts.

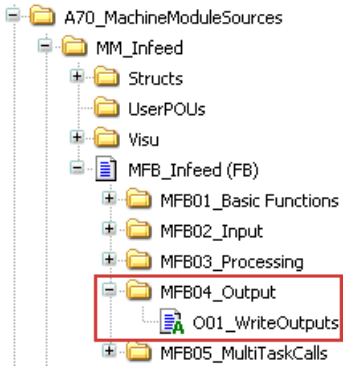
5.6.7.4 Sequence of actions



[5-3] Sequence: Program cycle in the machine module

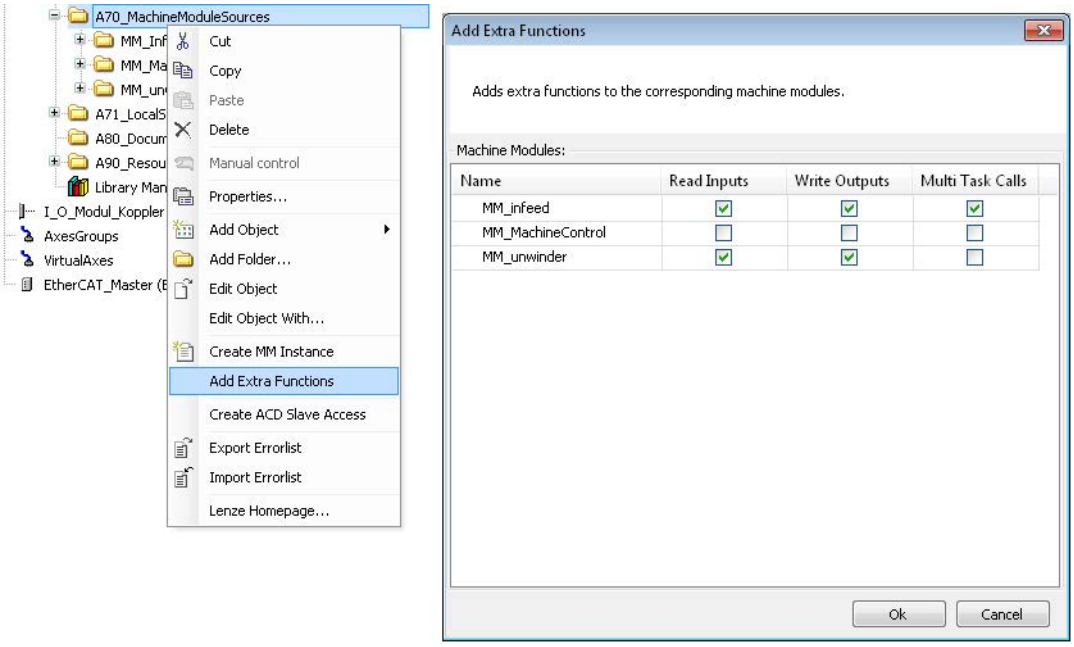
Figure [5-3] shows the entire sequence of a program cycle for the "HighPriority" task with the optional actions I01_ReadInputs (📖 75) and O01_WriteOutputs (📖 79) for using the IPO model.

5.6.8 O01_WriteOutputs

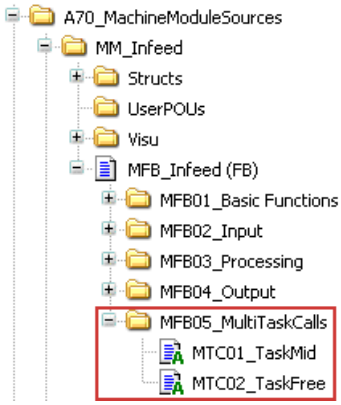


The **O01_WriteOutputs** action in the **MFB04_Output** module folder can be used as structuring aid if the IPO model is used (input – processing – output). A detailed description of the IPO model can be found in the section MFB03_Processing (📖 76).

At first, the **MFB04_Output** module folder is not visible in the project tree. If required, it can be shown/hidden with the **"Add Extra Functions"** command under **A70_MachineModuleSources** if the IPO model is to be used. For this purpose, checkmarks can be set or removed under the appearing "Write Outputs" dialog.



5.6.9 MTC01_TaskMid / MTC02_TaskFree



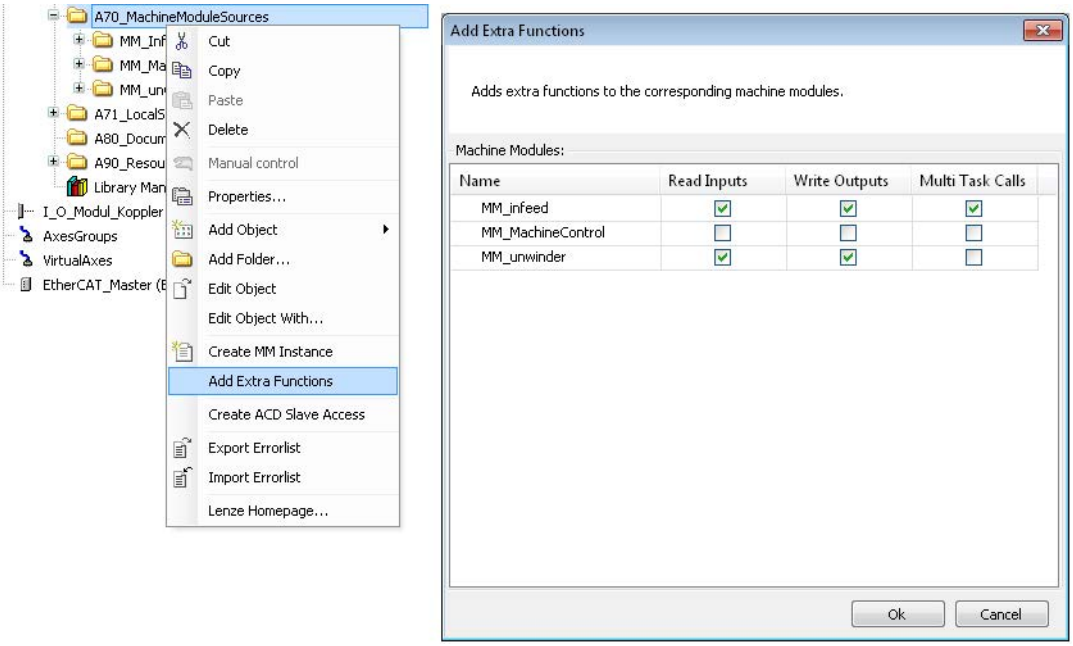
Normally, the entire application program can be programmed as single task solution in the "HighPriority" task context in the predefined actions of the Application Template.

In the exceptional case that parts of the module application have to be called in the task contexts "MidPriority" and/or "Freewheeling", the **MFB05_MultiTaskCalls** module folder provides the actions **MTC01_TaskMid** and **MTC02_TaskFree**.

Note!

If the application program contains cross-task data accesses, respective measures have to be taken to ensure the data consistency.

At first, the **MFB05_MultiTaskCalls** module folder is not visible in the project tree. If required, it can be shown/hidden with the **"Add Extra Functions"** command under **A70_MachineModuleSources** if the IPO model is to be used. For this purpose, checkmarks can be set or removed under the appearing "Multi Task Calls" dialog.



Index

R

S

T

U

V

Your opinion is important to us

These instructions were created to the best of our knowledge and belief to give you the best possible support for handling our product.

Perhaps we have not succeeded in achieving this objective in every respect. If you have suggestions for improvement, please e-mail us to:

feedback-docu@lenze.com

Thank you very much for your support.

Your Lenze documentation team

Lenze Automation GmbH
Postfach 10 13 52, 31763 Hameln
Hans-Lenze-Straße 1, 31855 Aerzen
GERMANY
HR Hannover B 205381
+49 5154 82-0
+49 5154 82-2800
sales.de@lenze.com
www.lenze.com

Service
Lenze Service GmbH
Breslauer Straße 3, 32699 Extertal
GERMANY
008000 24 46877 (24 h helpline)
+49 5154 82-1112
service.de@lenze.com