Contents

8 Working with the ApplicationTemplate ................................................. 40
  8.1 Mapping the actual machine structure in the »PLC Designer« .................. 41
    8.1.1 Adding devices - EtherCAT bus system .................................... 41
    8.1.2 Adding devices - CANopen bus system (optional) ......................... 44
  8.2 Creating your own machine modules: Copy/insert MM_EmptyModule ............ 52
  8.3 Creating instances of machine modules: Create MM instance .................. 53
  8.4 Integrating a machine module into the MMT ..................................... 55
  8.5 Assigning the module application (MAP) to the task ........................... 56
  8.6 Remove the instances of a machine module ...................................... 58
  8.7 Removing machine modules ......................................................... 58
  8.8 Module ID ................................................................. 58
  8.9 Inserting an axis ......................................................................... 60
  8.10 Integrating I/O modules of the I/O system 1000 with a machine module .... 62
  8.11 Creating module applications ..................................................... 64
    8.11.1 Adapting the function of the module application inserted - overview ... 65
    8.11.2 Frequently asked question: Connection between MFB and MAP .......... 66
    8.11.3 Programming with the module application .................................. 67
    8.11.4 Integrating a module application ........................................... 68
  8.12 Online change ........................................................................... 70
    8.12.1 When can an "Online change" be used in the ApplicationTemplate? .... 70
    8.12.2 When is it not possible to use an "Online change" in the ApplicationTemplate? 71
    8.12.3 General programming recommendations which enable an "Online change" 71

9 Architecture: The ApplicationTemplate in detail .................................. 73
  9.1 Accessing structure variables of machine modules ............................... 73
    9.1.1 Accessing module's intrinsic structure variables ............................ 73
    9.1.2 Accessing structures of other machine modules ............................ 74
  9.2 The AppChannelData structure (ACD) ............................................. 75
    9.2.1 Declaring/recording the ACD structure in the MFB ........................ 75
    9.2.2 Accessing the ACD structure - by means of the MFB module application ... 76
    9.2.3 Accessing an ACD structure - by means of the module application of the MFB master module .................................................. 77
  9.3 State machine ............................................................................ 79
    9.3.1 State transitions and conditions - overview ................................... 79
    9.3.2 Initial state of the state transitions ............................................ 80
    9.3.3 The state transitions in detail ................................................... 81
    9.3.4 Mapping of the states - Enum L_EATP_SMStates .......................... 84
    9.3.5 Activating the states - baseChannelSetNominalState method ........... 84
Contents

11.2 1_POUs - Program Organization Units ......................................................... 121
   11.2.1 L_EATP_Application ................................................................. 121
   11.2.2 L_EATP_CriticalSection ......................................................... 123
   11.2.3 L_EATP_ErrorListHandler ....................................................... 124
   11.2.4 L_EATP_ErrorSet ........................................................................ 125
       11.2.4.1 Adapting the error handling system .................................... 126
       11.2.4.2 Acknowledging errors ....................................................... 126
   11.2.5 L_EATP_Module ................................................................. 127
       11.2.5.1 Base() ........................................................................ 127
   11.2.6 RegisterACD() ........................................................................ 128
   11.2.7 setComplIAndVersion() ................................................................. 128
   11.2.8 L_EATP_ModuleDiag ................................................................. 129
   11.2.9 L_EATP_ModuleErrorHandler .................................................. 130
   11.2.10 L_EATP_ModulRelations ......................................................... 131
   11.2.11 L_EATP_SMAccess .................................................................... 132

11.3 2_Structs_Types ................................................................. 133
   11.3.1 L_EATP_ACD_Base (ACD structure) ........................................... 133
   11.3.2 L_EATP_BaseChannel ............................................................... 134
   11.3.3 L_EATP_ErrorList ................................................................. 134
   11.3.4 L_EATP_ErrorListEntry .......................................................... 134
   11.3.5 L_EATP_MMD_Base (MMD structure) ........................................ 135
   11.3.6 L_EATP_MM_Address ............................................................... 136
   11.3.7 L_EATP_MVis ................................................................. 136
   11.3.8 L_EATP_scBase ....................................................................... 136

11.4 3Enums ................................................................. 136
   11.4.1 L_EATP_ErrorReactionType .................................................. 136
   11.4.2 L_EATP_SMStates ................................................................. 137

12 Structuring an automation system: Example - flying saw ........................................... 138
   12.1 Preparation: Dividing the automation system into subfunctions ............................. 138
       12.1.1 Identifying subfunctions of the machine structure ............................ 139
       12.1.2 Representing subfunctions of the machine structure as machine modules .... 139
       12.1.3 Representing machine modules in a tree structure ............................. 139
   12.2 Mapping the actual machine structure in the »PLC Designer« ................................. 140
       12.2.1 Device structure: Adding master and slave .................................... 140
       12.2.2 Setting the cycle time .............................................................. 141
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.3</td>
<td>Creating and integrating machine modules</td>
<td>141</td>
</tr>
<tr>
<td>12.3.1</td>
<td>Creating machine modules</td>
<td>141</td>
</tr>
<tr>
<td>12.3.2</td>
<td>Integrating machine modules in the machine structure tree</td>
<td>143</td>
</tr>
<tr>
<td>12.3.3</td>
<td>Managing module applications of the machine modules</td>
<td>144</td>
</tr>
<tr>
<td>12.3.4</td>
<td>Inserting an axis into the module application</td>
<td>145</td>
</tr>
<tr>
<td>12.3.5</td>
<td>Assigning the module application to the tasks</td>
<td>145</td>
</tr>
<tr>
<td>12.4</td>
<td>Setting up communication between master modules and slave modules</td>
<td>147</td>
</tr>
<tr>
<td>12.5</td>
<td>Application example: Extending the visualisation</td>
<td>148</td>
</tr>
<tr>
<td>12.6</td>
<td>State machine</td>
<td>151</td>
</tr>
<tr>
<td>12.6.1</td>
<td>Controlling the state transition in the machine control module</td>
<td>151</td>
</tr>
<tr>
<td>12.6.2</td>
<td>Controlling state transitions in the slave modules</td>
<td>153</td>
</tr>
<tr>
<td>12.7</td>
<td>Manual jog of the axes</td>
<td>154</td>
</tr>
<tr>
<td>12.8</td>
<td>Error handling: Configuring the error handling</td>
<td>155</td>
</tr>
<tr>
<td>13</td>
<td>The &quot;ApplicationTemplateCounter&quot; sample project</td>
<td>159</td>
</tr>
<tr>
<td>13.1</td>
<td>Sample programs in the machine module MM_Module1/MM_Module2</td>
<td>160</td>
</tr>
<tr>
<td>13.1.1</td>
<td>&quot;Automatic&quot; status</td>
<td>160</td>
</tr>
<tr>
<td>13.1.2</td>
<td>&quot;Mode1&quot; status</td>
<td>161</td>
</tr>
<tr>
<td>13.1.3</td>
<td>&quot;Mode2&quot; status</td>
<td>161</td>
</tr>
<tr>
<td>13.1.4</td>
<td>&quot;Service&quot; status</td>
<td>161</td>
</tr>
<tr>
<td>13.1.5</td>
<td>&quot;Fault&quot;/&quot;System fault&quot; status</td>
<td>162</td>
</tr>
<tr>
<td>13.2</td>
<td>Sample program MM_Machine machine module</td>
<td>163</td>
</tr>
<tr>
<td>13.3</td>
<td>Simulation in the ApplicationTemplateCounter</td>
<td>164</td>
</tr>
</tbody>
</table>
Contents

14 Appendix .................................................................................................................. 165
  14.1 Method overview - ApplicationTemplate ............................................................ 165
    14.1.1 Accessing the state machine - the methods of the BaseChannel .................. 165
    14.1.2 Inhibiting/enabling module-specific state transitions .................................. 167
    14.1.3 Activate / deactivate quick stop response of the state machine ................. 167
    14.1.4 Activating/deactivating default coupling ................................................. 168
  14.2 Tips&tricks .......................................................................................................... 168
    14.2.1 Renaming a machine module ...................................................................... 168
    14.2.2 Next steps: Renaming the visualization buttons ......................................... 170
    14.2.3 Recipe manager .......................................................................................... 172
    14.2.4 Condition .................................................................................................... 173
    14.2.5 Creating the recipe definition ..................................................................... 173
    14.2.6 Recipe definition - assigning variables ....................................................... 174
    14.2.7 Inserting recipes into a recipe definition ...................................................... 175
    14.2.8 Managing recipes with RecipeManCommands .............................................. 176
      14.2.8.1 Create Recipe ....................................................................................... 176
      14.2.8.2 ReadAndSaveRecipeAs ......................................................................... 177
      14.2.8.3 LoadFromAndWriteRecipe ................................................................. 178
      14.2.8.4 WriteRecipe ......................................................................................... 179
      14.2.8.5 ReadRecipe ......................................................................................... 180
      14.2.8.6 GetRecipeCount .................................................................................... 181

Your opinion is important to us .................................................................................. 185
1 About this documentation

This documentation describes the operating mode of the Lenze application template "ApplicationTemplate" which serves as a basis for programming a Lenze automation system afterwards. The used automation system consists of a PLC for the "Controller-based Automation" system and drive components which are connected via the bus system.

⚠️ Tip!
Current information on the Lenze products can be found in the download area at: http://www.Lenze.com

⚠️ Note!
This documentation is a supplement to the »PLC Designer« online help.

Read the Mounting Instructions accompanying the controller first before you start working!
The mounting instructions include safety instructions which must be observed!

This manual is part of the "Controller-based Automation" manual collection. The manual collection consists of the documents:

<table>
<thead>
<tr>
<th>Documentation/abbreviation</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>System manuals (SHB)</td>
<td>• Controller-based Automation</td>
</tr>
<tr>
<td></td>
<td>• Visualization</td>
</tr>
<tr>
<td>Communication manuals (KHB)</td>
<td>• Controller-based Automation EtherCAT®</td>
</tr>
<tr>
<td></td>
<td>• Controller-based Automation CANopen®</td>
</tr>
<tr>
<td></td>
<td>• Controller-based Automation PROFIBUS®</td>
</tr>
<tr>
<td>Software manuals (SW)</td>
<td>• Controller</td>
</tr>
<tr>
<td></td>
<td>• »PLC Designer«</td>
</tr>
<tr>
<td></td>
<td>• »Engineer«</td>
</tr>
<tr>
<td></td>
<td>• »VisiWinNET® Smart«</td>
</tr>
<tr>
<td></td>
<td>• »Backup &amp; Restore«</td>
</tr>
</tbody>
</table>

⚠️ Information on the use of the controller beyond the field of "Controller-based Automation" can be found in the system manuals tailored to the application case.
Further technical documentation on Lenze products

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

<table>
<thead>
<tr>
<th>Mounting &amp; wiring</th>
<th>Symbols:</th>
<th>Abbreviation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ MAs for the controller</td>
<td>□ Printed documentation</td>
<td>BA Operating instructions</td>
</tr>
<tr>
<td>□ MA i700 Servo Inverter</td>
<td>□ Online help / PDF file</td>
<td>KHB Communication manual</td>
</tr>
<tr>
<td>□ MAs for Servo Drives 9400</td>
<td></td>
<td>MA Mounting instructions</td>
</tr>
<tr>
<td>□ MAs for Inverter Drives 8400</td>
<td></td>
<td>SW Software manual</td>
</tr>
<tr>
<td>□ MA I/O system 1000 (EPM-Sxxx)</td>
<td></td>
<td>SHB System manual</td>
</tr>
<tr>
<td>□ MAs for communication cards (MC-xxx)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>□ MAs for communication modules</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Parameterisation, configuration, commissioning         |                                               |                               |
| □ SW Controller                                       |                                               |                               |
| □ SW for the i700 servo inverter                       |                                               |                               |
| □ SW for the Servo Drive 9400 HighLine/PLC/Regenerative power supply module | |                               |
| □ 9400 HighLine commissioning guidelines               |                                               |                               |
| □ SW Inverter Drives 8400 StateLine/HighLine/TopLine   |                                               |                               |
| □ SHB I/O system 1000 (EPM-Sxxx)                       |                                               |                               |
| □ BAs for servo system ECS (ECSxE, ECSxM)              |                                               |                               |
| □ KHBs for communication modules                       |                                               |                               |

| Programming                                             |                                               |                               |
| □ SW 9400 function library                             |                                               |                               |

| Reuse                                                  |                                               |                               |
| □ SW Application Sample i700                           |                                               |                               |
| □ SW Application Samples                               |                                               |                               |
| □ SW ApplicationTemplate                                |                                               |                               |

Target group
This documentation addresses to all persons who plan, commission, and program a Lenze automation system on the basis of the Lenze "ApplicationTemplate" as part of the "Controller-based Automation".

Screenshots/application examples
All screenshots in this documentation are application examples. Depending on the firmware version of the Lenze device and the software version of the engineering tools installed (here: »PLC Designer«), the representation of the actual screen display may deviate.
Information regarding the validity

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

<table>
<thead>
<tr>
<th>Software</th>
<th>from software version</th>
</tr>
</thead>
<tbody>
<tr>
<td>»PLC Designer«</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Valid for the following Lenze templates:
- "ApplicationTemplate Counter" sample project: L_ApplicationTemplateCounter
- "Application Template" application template: L_ApplicationTemplate

1.1 Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 04/2012 TD11</td>
<td>First edition</td>
</tr>
</tbody>
</table>
| 1.1 07/2012 TD11 | Updated to M5 of the ApplicationTemplate  
  • General correction  
  • Adaptation to VISU layout according to the Lenze programming style guide for FBs. |
| 1.2 11/2012 TD11 | Updated to »PLC Designer« V3.3.2  
  • New: ApplicationTemplateCounter sample project (Lenze standard) |
| 1.3 04/2013 TD11 | Updated to »PLC Designer« V3.5  
  • Software update of ApplicationTemplateCounter/ApplicationTemplate.  
  • New: Application example "flying saw". |
| 1.4 11/2013 TD11 | Updated to »PLC Designer« V3.6  
  • Optimisations from usability tests (user group) were implemented.  
  • System error messages have been added.  
  • L_EATP_MMD_Base structure has been added.  
  • "Create MM Instance" command has been added. |
### 1.2 Conventions used

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

<table>
<thead>
<tr>
<th>Type of information</th>
<th>Highlighting</th>
<th>Examples/notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spelling of numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal separators</td>
<td>Point</td>
<td>The decimal point is generally used. For example: 1234.56</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version information</td>
<td>Text colour blue</td>
<td>All pieces of information that only apply to or from a specific software version of the inverter are highlighted correspondingly in this documentation. Example: This function extension is available from software version V3.0!</td>
</tr>
<tr>
<td>Program name</td>
<td>» «</td>
<td>»PLC Designer«...</td>
</tr>
<tr>
<td>Window</td>
<td>italics</td>
<td>The message window.../The Options dialog box...</td>
</tr>
<tr>
<td>Variable names</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control element</td>
<td>bold</td>
<td>The OK button.../The Copy command.../The Properties tab.../The Name input field...</td>
</tr>
<tr>
<td>Sequence of menu commands</td>
<td></td>
<td>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...</td>
</tr>
<tr>
<td>Shortcut</td>
<td>&lt;bold&gt;</td>
<td>Use &lt;F1&gt; to open the online help. If a command requires a combination of keys, a &quot;+&quot; is placed between the key symbols: With &lt;Shift&gt;+&lt;ESC&gt; you can...</td>
</tr>
<tr>
<td>Hyperlink</td>
<td>Underlined</td>
<td>Reference to further information: Hyperlink to further information.</td>
</tr>
<tr>
<td>Symbols</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step-by-step instructions</td>
<td></td>
<td>Step-by-step instructions are marked by a pictograph.</td>
</tr>
</tbody>
</table>
1.3 Notes used

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

Safety instructions
Structure of the safety instructions:

⚠️ Pictograph and signal word!
(characterises the type and severity of danger)

Note
(describing the danger and explains how to avoid it.)

<table>
<thead>
<tr>
<th>Pictograph</th>
<th>Signal word</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>⚠️ Danger!</td>
<td>Danger of personal injuries through electrical voltage Reference to an imminent danger that may result in death or serious personal injury unless the corresponding measures are taken.</td>
<td></td>
</tr>
<tr>
<td>⚠️ Danger!</td>
<td>Danger of personal injury through a general source of danger Reference to an imminent danger that may result in death or serious personal injury unless the corresponding measures are taken.</td>
<td></td>
</tr>
<tr>
<td>⏹️ Stop!</td>
<td>Danger of material damage Indicates a potential danger that may lead to material damage unless the corresponding measures are taken.</td>
<td></td>
</tr>
</tbody>
</table>

Application notes

<table>
<thead>
<tr>
<th>Pictograph</th>
<th>Signal word</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>📝 Note!</td>
<td>Important note to ensure troublefree operation</td>
<td></td>
</tr>
<tr>
<td>🕯️ Tip!</td>
<td>Useful tip for easy handling</td>
<td></td>
</tr>
<tr>
<td>📜 Reference to another document</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 1.4 Terminology used (presented according to the order in the device view)

<table>
<thead>
<tr>
<th>Term/abbreviation</th>
<th>Position in the device view</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine module tree MMT</td>
<td><img src="#" alt="A10_MachineModuleTree" /></td>
<td>The “MachineModuleTree” (MMT) maps the structure of the automation system in the form of machine modules.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• In the &quot;MachineModuleTree&quot;, all machine modules required for the machine are interconnected hierarchically according to the mechatronic interaction.</td>
</tr>
<tr>
<td>ModuleApplicationCalls MAC</td>
<td><img src="#" alt="A11_ModuleAppCalls" /></td>
<td>The module applications are to be assigned to the corresponding task within the “ModuleApplicationCalls”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Thus it is defined which module application is to be processed within the individual tasks.</td>
</tr>
<tr>
<td>Machine module MM</td>
<td><img src="#" alt="A70_MachineModuleSources" /></td>
<td>A machine module maps a unit of the real machine structure in the »PLC Designer«.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The machine module is part of the MachineModuleTree (MMT) within which the individual machine modules are interconnected.</td>
</tr>
<tr>
<td>Machine module application MAP</td>
<td></td>
<td>The machine module application provides the functionality of a machine module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A machine module can contain one or several machine module applications.</td>
</tr>
<tr>
<td>Machine function block MFB</td>
<td></td>
<td>The machine function block represents the machine module in the machine module tree (MMT).</td>
</tr>
</tbody>
</table>
Safety instructions

Please observe the following safety instructions when you want to commission a controller or system.

Read the documentation supplied with the controller or the individual components of the system carefully before you start to commission the devices!

The device documentation contains safety instructions which must be observed!

Danger!

According to today's scientific knowledge it is not possible to ensure absolute freedom from defects of a software.

If required, systems with integrated controllers have to be equipped with additional monitoring and protective equipment in accordance with the safety regulations valid in each case (e.g. law on technical equipment, regulations for the prevention of accidents), so that an impermissible operating status does not endanger persons or equipment.

During commissioning persons must keep a safe distance from the motor or the machine parts driven by the motor. Otherwise there would be a risk of injury by the moving machine parts.

Stop!

If you change parameters in the »PLC Designer« while an online connection to the device is established, the changes are directly accepted in the device!

An incorrect parameterisation can result in unpredictable motor movements. By an unintentional direction of rotation, too high speeds or jerky operation, powered machine parts can be damaged!
3 Preconditions

3.1 System requirements

![Diagram of Engineering PC and Controller connected via Ethernet]

### Engineering PC
- PC/notebook
- Windows XP
- »PLC Designer« from V3.3
  - Contains the ApplicationTemplate
  - Contains the Lenze library "L_EATP_ApplicationTemplate.compiled-library"

### Controller
- PLC (Logic) from firmware V3.3
- Windows CE
- Runtime software
  - Logic
  - Motion (for this purpose, the project data must be updated: "Update Device")

### Further requirements
- Depending on the application case:
  - CAN or EtherCAT bus system
  - CAN or EtherCAT node

#### Setting up communication to the Controller

- Connect the Engineering PC with the controller via a network cable. The »PLC Designer« accesses the controller via Ethernet.
- Make the IP settings with the »PLC Designer«.

##### How to check the communication settings:
1. Double-click the desired controller in the Devices view.
2. Make the desired settings on the Communication settings tab.
   - Click the Add gateway button to insert a gateway.
Preconditions
Setting up communication to the Controller

- Enter the desired IP address of the Controller.

![Gateway setup](image)

[3-1] Example: Enter the IP address of the Controller, standard setting: 192.168.5.99

3. Click **OK** to add the controller as gateway.
4. By double-clicking the desired channel (or clicking the **Set active path** button), set the channel selected in the device view below the gateway as active path to the controller.
   - Thus, all communication actions directly refer to this channel.
   - The currently active path is represented in **bold** in the list and "(active)" is attached:

![Gateway view](image)

5. A device represented in **italics** is set as active path but has not been found during the last network scan.

**Note!**

During initial commissioning, observe the predefined IP address: **192.168.5.99**

Further information can be found in the following documentation:

**Controller - Parameter setting & configuration**
What is the ApplicationTemplate?

Objective of the ApplicationTemplate

4  What is the ApplicationTemplate?

The ApplicationTemplate is a Lenze application template for standardised and convenient programming in the »PLC Designer«.

• The ApplicationTemplate is included in the »PLC Designer« (from version 3.3) as project template.  ‣ Create a new project - open the ApplicationTemplate (п 33)
• The L_EATP_ApplicationTemplate.compiled-library library includes the structure and the basic functionality of the ApplicationTemplate.  ‣ The L_EATP_ApplicationTemplate library (п 116)

4.1  Objective of the ApplicationTemplate

The ApplicationTemplate...

• ...helps to implement the mechatronic structure of an automation system (which is available as a tree structure) in a modular manner.
• ...enables the integration of predefined machine modules with prepared applications (for instance a cross cutter).
• ...simplifies and speeds up the creation of PLC programs in the long term by re-use of a standardised and modularised folder structure.

What are the advantages of the ApplicationTemplate?

The ApplicationTemplate facilitates programming with the »PLC Designer« ...

• ...by a predefined folder structure which "cleans up" and which can be extended individually.
• ...renders the navigation for extending or creating machine programming easier.
• The ApplicationTemplate contains ready-made, re-usable machine modules and module applications which minimise the risk of compilation errors in order to thus reduce time and costs.
4.2 Features of the ApplicationTemplate at a glance

The following functions facilitate implementing a machine application in a PLC:
- State machine (23)
- Error handling (94)
- Multitasking (104)

Further advantages if the ApplicationTemplate is used:
- Consistent data transfer between the tasks.
- Diagnostic function for every machine module ("generic module diagnosis").
- A defined standard response ("DefaultCoupling") of the state machine. Default coupling (85)
- Decouples one (or several machine modules. Internal Control (107)

For more information on the respective functions, please see the corresponding subchapter.
4.3 Elements of the ApplicationTemplate

4.3.1 Machine Module Tree - MMT

In order to map the desired automation system in the »PLC Designer« on the basis of the ApplicationTemplate, the structure of the whole machine application must be created in the »PLC Designer«.

- In a first step, the machine structure must be divided into machine modules.

- The A10_MachineModuleTree machine module tree (MMT) shows the machine modules in the form of a tree structure from left to right.

The ApplicationTemplate...

- ...supports two to five hierarchy levels of machine modules.
- ...supports up to 30 machine modules.
4.3.2 **Machine modules (MM)**

The overall functionality of the automation system is structured in a modular manner in the ApplicationTemplate. This means that every subfunction of the machine is included in one of the machine modules. Due to the modular structure, individual (or multiple) subfunctions of a machine can be reused. Advantage: The respective function does not have to be recreated for further machine parts.

- A machine module represents the function of a machine part; for instance a conveying belt, or a cross cutter.
- The overall functionality of, for example, a bag form, fill, and seal machine, contains the "Cross cutter" and "Transport unit" subfunctions. The two subfunctions are to be converted to a separate machine module each.

**Machine module in the ApplicationTemplate**

Every machine module contains the BaseChannel ("Base Data") which serves as a data channel for the basic functions of the ApplicationTemplate.

- The basic functions of the ApplicationTemplate are the State machine and the Error handling.

Every machine module has an AppChannelData structure (ACD structure). An ACD structure can be defined in a machine module if necessary.

- Via the ACD structure, data are provided to/received from the higher-level machine module.
- Via the ACD structure, process data can be exchanged between the user's own module applications.

A machine module (MFB) always contains at least one module application (MAP). Up to three MAPs per MFB are possible.

- Via the MM_IO, MM_Par, MM_Vis, MM_PD structures, the module application (MAP) is to be connected to the "outside world" (the respective sub-function of the automation system).
- By means of the MM_IO structure, the inputs/outputs of the terminals/the fieldbus are to be connected.
- The MM_Par structure contains all variables that are to be managed by the recipe manager.
- The MM_Vis structure contains all variables that can be controlled or are to be displayed via an external visualization.
- The MM_PD structure contains all persistent variables.
What is the ApplicationTemplate?
Elements of the ApplicationTemplate

4.3.3 Addressing the machine modules

Every machine module has an `MM_Address` input which serves to assign the relative address to the machine module.

The following must be observed when relative addresses are assigned to the machine modules:

- The relative address is to be assigned to every machine module (value range: 1...29).
- During the initialisation phase, the »PLC Designer« generates an absolute address for every machine module.
- Example of the relative and absolute module addressing:

The diagram shows the absolute module address (black) and the relative module address (white).

- In the event of an error, the absolute address enables an error analysis. This for instance makes it possible to retrace the module which has caused the error in each case.  » Error handling (§ 94)

4.3.4 Module application (MAP)

The module application (MAP) contains the function of the corresponding machine module.

- The ApplicationTemplate supports up to three tasks. Hence, up to three MAPs can be used per machine module.

- In the `A11_ModuleAppCalls` folder, the MAPs are to be assigned to the tasks: ModuleAppCalls (MAC).
4.3.5 Communication between the machine modules

The machine modules (MM_xxx) communicate with each other via the MM_Machine machine control module by means of the BaseChannel communication channel and the AppChannelData structure.

- The communication channels provide for a bidirectional data exchange.
- The BaseChannel is defined as a structure in the ApplicationTemplate.

One or several slave modules are always exactly connected to one higher-level master module. However, the master only always communicates with one slave module. Slave modules cannot communicate directly with each other, but only via the higher-level master module.

The higher-level machine module (master) communicates with the lower-level machine modules (slaves) via data channels (channels). During the initialisation, the ApplicationTemplate generates a BaseChannel and an AppChannelData (ACD) structure.

BaseChannel: exchange of control and status data (Control/Status basic data)

The BaseChannel...
- ...contains the control / status information of the state machine.
- ...contains the error handling.

The ACD structure...
- ...serves to exchange application process data between machine modules.
- ...is a data structure for the definition of own process data.
- ...must always be derived from the L_EATP_ACD_Base (ACD structure) structure.
4.3.6  State machine

Every machine module features one state machine.  

- If a machine module is integrated in the machine module tree (if it is not decoupled from the MMT), the state machine of the master module controls all state machines of the lower-level machine modules.
- When all slave modules have changed to the requested state, the master module also changes to the requested state.  
  **Exception:** During the initialisation, all machine modules are in the "INIT" state and, independently of the slave or master, change to the "READY" state if it is enabled:

```plaintext
SMEInitEnableToReady(TRUE);
```

**State transitions - overview**

The following status diagram illustrates the possible operating states of the state machine:

- After the start (switch-on / re-initialisation), the module is in the "Init" state.
- In standby/waiting mode, the module is in the "Ready" state.
- "Warning" is a special status (independent, "orthogonal" state), which does not influence the operational performance of the machine module.
4.3.7 Error handling

The ApplicationTemplate includes an error handling which enables the diagnosis of the machine modules.

The error handling of the ApplicationTemplate provides mechanisms by means of which responses (errors, warnings, messages) can be defined and triggered in the module applications (MAP) of the machine modules (MM).

Further mechanisms are:
- The forwarding of error states in the MachineModuleTree (MMT).
- An application-global error list with the current error status of all machine modules contained in the MMT.
- Transmission of errors and events to the central logbook of the controller.

Further information can be found under: Error handling (§ 94)
5 Structuring the automation system: Standard procedure

This section describes the standard procedure to create an application with the »PLC Designer« based on the ApplicationTemplate.

- Use the following recommendations as a guide in order to be able to then create a PLC project in the »PLC Designer« in a structured manner using the ApplicationTemplate and to program it effectively.

- Due to the structured layout of the ApplicationTemplate (the consistency in these structures and the compliance with these structures), applications can be created more quickly and hence integrated in an existing PLC program more quickly.

<table>
<thead>
<tr>
<th>Step</th>
<th>Activity</th>
<th>What has to be done?</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1    | Divide machine structure into machine modules (MM) | Gain an overview of the overall functionality of the machine structure.  
- Divide the overall functionality of the machine structure into subfunctions.  
- Transfer the identified subfunctions of the machine structure to machine modules. | In this project phase, programming is not yet required!  *Assign the relative address to the machine modules.* (§ 27) |

[5-1] Recommended procedure for creating a project efficiently.
Structuring the automation system: Standard procedure

<table>
<thead>
<tr>
<th>Step</th>
<th>Activity</th>
<th>What has to be done?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Create machine modules (MM)</td>
<td>Create machine modules containing the subfunctions of the machine structure in each case: one subfunction = one machine module.</td>
<td>• In order to be able to call machine functions in different tasks, corresponding module applications have to be created. &lt;br&gt;• More information about structuring within a module application: Structuring within a machine module ([28])&lt;br&gt;• Define the interfaces for the module applications (MAPs).&lt;br&gt;• Optionally create the visualization for the respective machine module.&lt;br&gt;• Each machine module is provided with a state machine. Irrespective of the active status, the module application (MAP) calls a corresponding action. The action is subordinated to the module application.&lt;br&gt;• Within these actions, create the logic which is to be executed if the machine module (MM) is in the corresponding status.</td>
</tr>
<tr>
<td>3</td>
<td>Instance structures and machine function block (MFB) globally</td>
<td>• Integrate newly created machine modules into the MMT (machine module tree).&lt;br&gt;• Assign the relative address to the machine modules.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Call module application in the MachineAppCalls (MAC)</td>
<td>Creating module applications ([64])</td>
<td></td>
</tr>
</tbody>
</table>
Structuring the automation system: Standard procedure
Assign the relative address to the machine modules.

5.1 Assign the relative address to the machine modules.

In order to modularise programming of a machine system, the individual subfunctions of the overall functionality of the automation system have to be mapped in the form of machine modules.

Example: Bag form, fill, and seal machine ("Flow Packer")

- It is helpful to outline the machine structure with the individual subfunctions in a tree structure.
- For this, the individual subfunctions of the machine have to be transferred to corresponding machine modules.

Examples of machine modules
- "Virtual master"
- "Infeed" (feeder)
- "Outfeed" (extractor)

- If the individual subfunctions are structured in the form of machine modules, the interfaces are to be assigned to the module application (MAP). See Creating your own machine modules: Copy/insertMM_EmptyModule (Page 52)
- Assign the input and output variables...
  ...in variable lists MM_IO, MM_Par, MM_Visu and MM_PD and...
  ...to the variables of the AppChannelData (ACD) structure.
5.2 Structuring within a machine module

In order to create a clearly arranged module application, it is advisable to divide the module applications (MAP) into subfunctions and to structure them correspondingly.

5.2.1 Assigning the MAP subfunctions to individual tasks

In a first step, the functions are to be assigned to the individual tasks. The ApplicationTemplate supports multitasking with three tasks. More information can be found under: Multitasking (104)

- Task "High" (standard value: 2 ms)
- Task "Mid" (standard value: 6 ms)
- Task "Free" (unsolicited)

One module application can be used per task.

- Task and module application are assigned in the A1_ModuleAppCalls folder.
- The MAC_Task_High program part for instance calls all module applications which are to pass through a high priority task Task_High.

ApplicationTemplateCounter sample project: MAC_Task_High calls the Module1_App1 module application.
Recommendation - which function is to be called in which task?

<table>
<thead>
<tr>
<th>Task/priority</th>
<th>Function (example)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;High&quot;</td>
<td>HighPriority Execution of Motion functions</td>
</tr>
<tr>
<td>&quot;Mid&quot;</td>
<td>MidPriority Conversion for an external visualization</td>
</tr>
<tr>
<td>&quot;Free&quot;</td>
<td>Unsolicited NRT Ethernet communication</td>
</tr>
</tbody>
</table>

5.2.2 Programming recommendations for structuring

The following table presents the structuring possibilities of the ApplicationTemplate as a decision recommendation:

Which features of the ApplicationTemplate should I use and for what purpose?

<table>
<thead>
<tr>
<th>What would you like to do?</th>
<th>FB</th>
<th>Action</th>
<th>Method</th>
<th>More suitable is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work with local variables</td>
<td>●</td>
<td>Local variables are to be declared in the FB assigned.</td>
<td>Local variables (if possible) are to be declared in the FB assigned.</td>
<td>-</td>
</tr>
<tr>
<td>Debugging</td>
<td>●</td>
<td>●</td>
<td>No local variables</td>
<td>Action, FB</td>
</tr>
<tr>
<td>Reuse:</td>
<td>●</td>
<td>●</td>
<td>To a limited extent - for instance if FBs are integrated in the method. Example: &quot;Statemachine&quot; in the ApplicationTemplate</td>
<td></td>
</tr>
<tr>
<td>Instancing</td>
<td>●</td>
<td>●</td>
<td>-</td>
<td>Action, method</td>
</tr>
<tr>
<td>Access all data types</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Action, FB, method</td>
</tr>
</tbody>
</table>
Actions

For easy structuring, Lenze recommend to use the Actions of the ApplicationTemplate.

- The Actions must be lower level to the module application.
- Actions are marked with the icon.

Example of structuring a module application:

The SMDispatcher method calls the S01-S11 actions. The actions can be viewed in the ModApp1/States folder.

An Action is always connected to a POU (example: function block of an MAP module application).

- Thereby the action only uses the data of function block. This means that the action does not feature an own declaration part. The variables used in the action are declared in the function block (i.e. in the MAP module application).
- The action does not feature any local variables.
The Lenze ApplicationTemplate facilitates programming with the »PLC Designer«.

- The ApplicationTemplateCounter has the following predefined structure:
  
<table>
<thead>
<tr>
<th>File Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A10_MachineModuleTree</td>
<td>Machine module tree maps the mechatronic functionality of the machine structure in the form of machine modules (MM).</td>
</tr>
<tr>
<td>A11_ModuleAppCalls</td>
<td>Contains the assignments of module applications (MAP) to the tasks.</td>
</tr>
<tr>
<td>A20_Visualisation</td>
<td>Contains the visualizations for the device-independent functions.</td>
</tr>
<tr>
<td>A55_VarLists</td>
<td>Contains the declarations of the global variables: Machine modules used: MM_Dcl, IO variables: MM_IO, Parameters: MM_Par, Variables for an external visualization: MM_Vis, Persistent data: MM_PD.</td>
</tr>
<tr>
<td>A60_MotionObjects</td>
<td>Predefined folder for motion-relevant data, example: CAM profiles.</td>
</tr>
<tr>
<td>A65_EmptyModule</td>
<td>Contains the machine module sources.</td>
</tr>
<tr>
<td>A70_MachineModuleSources</td>
<td>Contains the individually created machine modules: Machine modules (MM).</td>
</tr>
<tr>
<td>A71_LocalSources</td>
<td>Storage location for machine-independent enumerations, function blocks, structures, visualizations.</td>
</tr>
<tr>
<td>A80_Documentation</td>
<td>Predefined folders for &quot;project history&quot; documents, example: version information, changes.</td>
</tr>
<tr>
<td>A90_Resources</td>
<td>Contains the system information such as: task settings, used libraries, Recipe manager, Visualization manager.</td>
</tr>
</tbody>
</table>

**Tip!**

Combine the "local sources" from the A71_LocalSources folder in one library.
7 Opening the ApplicationTemplate

The ApplicationTemplateCounter includes a sample program with three machine modules and a predefined visualization. The sample program serves to test the basic functions of the ApplicationTemplate. ▶ The "ApplicationTemplateCounter" sample project

General procedure

The main steps are summarised in the following table:

<table>
<thead>
<tr>
<th>Step</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Create a new project - open the ApplicationTemplate</td>
</tr>
<tr>
<td>2nd</td>
<td>Update the controller in the project (optional)</td>
</tr>
<tr>
<td>3rd</td>
<td>Going online</td>
</tr>
<tr>
<td></td>
<td>Compiling the project data</td>
</tr>
<tr>
<td></td>
<td>Transferring the project to the control - &quot;Log in&quot;</td>
</tr>
<tr>
<td>4th</td>
<td>Downloading and starting the PLC program</td>
</tr>
<tr>
<td>5th</td>
<td>Getting started - operating the ApplicationTemplate</td>
</tr>
</tbody>
</table>

Further information on the parameter setting and configuration of the respective bus system can be found in the following communication manuals (KHB):

- Communication manual **EtherCAT**: "Controller-based Automation" EtherCAT
- Communication manual **CANopen**: "Controller-based Automation" CANopen
- Communication manual **PROFIBUS**: "Controller-based Automation" PROFIBUS

The commissioning steps in detail

The following section provides a detailed description of every commissioning step.

Please follow the instructions below carefully to commission your automation system.
7.1 Create a new project - open the ApplicationTemplate

The ApplicationTemplate is included as a project template (*.project) in »PLC Designer« from V3.3 onwards. In order to call the ApplicationTemplate, a new project has to be created, taking the ApplicationTemplate as template.

How to proceed:

1. Creating a new project:
   - File → New project
     - Select category **Lenze Application Template**
     - Open template **L_ApplicationTemplate**

<table>
<thead>
<tr>
<th>Which template do you want to use?</th>
<th>Function</th>
</tr>
</thead>
</table>
| ApplicationTemplate               | Use Lenze application template **L_ApplicationTemplate**. It includes a structure predefined by Lenze which serves to:
  - ... standardise applications by means of a defined folder structure.
  - ... structure applications with machine modules. |
| ApplicationTemplateCounter        | Sample project with the Lenze application template **L_ApplicationTemplate**.
  - It includes two lower-level machine modules which are integrated into the MachineModuleTree (MMT).
  - Contains one counter example (Automatic Modus).
  The sample project is located in the **Lenze Application Samples\Lenze Standard category** |
7.2 Update the controller in the project (optional)

Cases in which the project must be updated

The controller in the »PLC Designer« must be updated if ...

• ...the project contains firmware information that is older than the hardware to be used or
• ...a controller other than the integrated 3200 C controller is desired (example: p500).

If the controller is marked with the ? icon after the project is opened, the device information of the »PLC Designer« project have to be updated.

Determining the firmware of the controller

How to proceed:

• Use the »WebConfig« to check which firmware is used by the controller to select the appropriate device information in the »PLC Designer«.
• If the controller must be updated, the dynamic memory settings of the application must be adapted.

Adapting the memory settings

Highlight the desired controller.

• Execute the Update device command in the context menu.
• Double-click the applicable controller in the Update device dialog window to update the controller in the Device view:

```
1. In the context menu of Application, execute the Properties command.
   • On the Application build options tab, activate the Use dynamic memory allocation option.
   • For Maximum size of memory, enter a value of 100000:

   • Click OK to confirm.
```
Opening the ApplicationTemplate

Going online

7.3 Going online

In order to be able to establish an online connection to the controller, the communication settings (Set active path) must be commissioned before. ➤ Setting up communication to the Controller

7.3.1 Compiling the project data

To compile the project data, select the Build→Build menu command or press the <F11> function key.

- If errors occur during the compilation, they are to be localised on the basis of the »PLC Designer« error messages and corrected correspondingly. Recompile the project data afterwards.
- If no errors occur during the compilation, the »PLC Designer« project must be saved:

File→Save project

7.3.2 Transferring the project to the control - "Log in"

Note!

To "log in" the PLC program must be error-free. For this it must be possible to execute the Build→Build (F11) menu command without an error message.

The desired project must be transferred to the PLC device by "Logging in" to the controller: Call menu command Online→Log in.

7.4 Downloading and starting the PLC program

- Load the PLC program to the controller: Call Online→Load menu command.
- Start the PLC program: Call Online→Start menu command.
- As an alternative, you can execute the Debug→Start menu command or press <F5>.

Tip!

In order to load a project automatically after a device is restarted, it can be defined as "boot project".

Setting up the project as boot application

How to install the project as boot project:

1. Select the Online→Generate boot project for L-force Controller menu command.
7.5 Getting started - operating the ApplicationTemplate

In the Device view, select the A20_Visualisation folder. Double-click the L_Main visualization.

Welcome page - L_Main visualization

The user interface of the visualization is divided into the following areas:

- **A** Select machine module
- **B** Detailed view of the machine modules
- **C** Select error overview
- **D** State machine
- **E** Error block visualization
- **F** ApplicationTemplate visualization
- **G** Buttons: Select status/acknowledge error message

Example: ApplicationTemplateCounter with two machine modules (module 1, module 2)
Opening the ApplicationTemplate
Getting started - operating the ApplicationTemplate

- The buttons in area A can be used to select the visualization of the desired machine module.
- **Machine / machine control** calls the highest machine control module.

- Button 1 calls the detailed view of the machine modules.
- Button 2 calls the global error overview.

- Area D displays the state machine (with the current status).  
  ➤ *State machine (\\(\text{\textsection } 23)*)

- Area E displays the **L_EATP_ErrorSet** FB.  
  ➤ *L_EATP_ErrorSet (\\(\text{\textsection } 125))*

---

Lenze · ApplicationTemplate · 1.4 EN · 11/2013
Opening the ApplicationTemplate  
Visualisation of the machine modules

7.6 Visualisation of the machine modules

- Area visualises communication between the individual machine modules by the AppChannelData structure.
- The AppChannelData structure provides the infrastructure for communication of the application data.  
  - The AppChannelData structure (ACD) ([75])
- ApplicationTemplateCounter: In this area, the demo visualization of the sample programs is shown.  
  - The "ApplicationTemplateCounter" sample project ([159])
- The Lock data button inhibits a data area of the ACD structure/enables the data area again.
  - Methods for inhibit/enable
    - Lock(): Inhibit data area of the ACD structure.
      - L_EATP_CriticalSection ([123])
    - Unlock(): Enable inhibited data area of the ACD structure.
      - L_EATP_ErrorListHandler ([124])
      - Consistent data transfer ([105])
- The buttons in area activate the desired status (state)/enable the acknowledgement of errors.
- The Error Quit button acknowledges the error message.  
  - Error handling ([94])

- If e.g. MM_Modul 1 is selected, errors can be triggered via fields xError[01...04] of L_EATP_ErrorSet.  
  - L_EATP_ErrorSet ([125])

...
• Depending on the standard response, triggering an error at L_EATP_ErrorSet results in a state change in area ③.
• Possible error responses are: Error, SystemFault, Warning, Information.

• Default setting of the ApplicationTemplate (can be deactivated, if required):
  When an error is triggered, the higher-level machine module changes to the "Quick stop" state.
  • The machine module forwards the state to the higher-level machine module.
  • The machine module on the top level sets all lower-level machine modules to the "Quick stop" status.

• The Module List button calls the detailed view for the machine modules.
• Click the desired machine module to show the respective status and further details.

• The Error List button calls the global error list which provides an overview of the errors that have occurred.
  • The cause of the fault activation must be eliminated. Then the corresponding error must be reset.
  • To reset the error, the Error Quit button is to be pressed.
This chapter provides information on how to create modular machine functions / machine modules with the machine module template in the ApplicationTemplate. The **MM_EmptyModule** machine module is a template for the creation of individual machine modules.

**Programming with the ApplicationTemplate: What has to be done?**

<table>
<thead>
<tr>
<th>Step</th>
<th>Activity</th>
<th>Detailed information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Structuring the automation system</td>
<td>• The overall functionality (machine application) of the automation system is to be mapped modularly: One subfunction = one machine module • In this project phase, programming is not yet required!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Assign the relative address to the machine modules. (لين 27)</td>
</tr>
<tr>
<td>2nd</td>
<td>Starting the ApplicationTemplate</td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td>Updating the project (optional)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Adjust the device information version in the »PLC Designer« project to the firmware version of the controller. • Integrate another controller in the project if required. The controller included is the 3200 C.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Update the controller in the project (optional) (لين 34)</td>
</tr>
<tr>
<td>4th</td>
<td>Mapping the actual machine structure in the »PLC Designer«</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Mapping the actual machine structure in the »PLC Designer« (لين 41)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Adding devices - EtherCAT bus system (لين 41) • Adding devices - CANopen bus system (optional) (لين 44) • Setting of CAN parameters and PDO mapping (لين 49)</td>
</tr>
<tr>
<td>6.</td>
<td>Creating/integrating individual machine modules</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creating your own machine modules: Copy/insertMM_EmptyModule (لين 52)</td>
</tr>
<tr>
<td>7.</td>
<td>Integrating devices</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Inserting an axis (لين 60) • Integrating I/O modules of the I/O system 1000 with a machine module (لين 62) • Integrating a module application (لين 68)</td>
</tr>
<tr>
<td>8.</td>
<td>Going online</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Going online (لين 35)</td>
</tr>
<tr>
<td>9.</td>
<td>Starting the PLC program</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Downloading and starting the PLC program (لين 35)</td>
</tr>
<tr>
<td></td>
<td>Carrying out an online change</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Online change (لين 70)</td>
</tr>
</tbody>
</table>
8.1 Mapping the actual machine structure in the »PLC Designer«

The ApplicationTemplate contains a predefined structure that can be extended by the individual requirements. Carry out the following steps to map the actual machine structure.

8.1.1 Adding devices - EtherCAT bus system

Note!

The following points have to be observed before starting to set up an EtherCAT configuration in the »PLC Designer«:

- The sequence of the EtherCAT slaves in the device view must correspond to the physical arrangement of the EtherCAT topology.
- Select the cycle times according to the technical data, from 1 ... 10 ms.

How to create the control configuration in the »PLC Designer«:

1. Open the context menu of the target system and execute the command **Append device** in order to extend the control configuration with **EtherCAT Master**.
2. Add an EtherCAT slave below the EtherCAT Master: Right-click the EtherCAT Master

1. **Add device:**

Select the desired device from the selection list.

The «PLC Designer» provides a "fieldbus scan" during which the devices connected to the fieldbus are automatically detected.

Further information is provided in the "Controller-based Automation EtherCAT" section of the online help for the «PLC Designer» and in the **Controller-based Automation EtherCAT** communication manual (KHB).

Repeat the **Add device** until all slaves connected to the fieldbus are implemented in the device view.
3. Allocate unique designations to the slaves inserted (example: "L_94_HL_ActuatorSpeed").

The names can be selected freely and must ...
• only contain the characters "A ... Z", "a ... z", "0 ... 9", or "_";
• not start with a digit.

You can enter a name by clicking the element.

Example:

```
  EtherCAT_Master (EtherCAT Master)
  L_94_HL_ActuatorSpeed (9400 Highline C - Actuator Speed)
  LC_Drive (LenzeLogicDrive_ETC_94HL)
```

4. Setting the cycle time

• The value for the EtherCAT master cycle time has to be defined according to the cycle time of the quickest task.

```
<table>
<thead>
<tr>
<th>Master</th>
<th>Diagnostic Master</th>
<th>Diagnostic Slaves</th>
<th>EtherCAT 10 Mapping</th>
<th>Status</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  • Use LHW instead of LHW/USD
  • Enable messages per task
  • Auto restart slaves

  Cycle time: 2000 μs
```

• The 🔄 icon in front of the respective device indicates the successful EtherCAT communication.

**Note:** The EtherCAT cycle time is to be set to the quickest task cycle time set. In this ApplicationTemplate, the quickest task cycle time is set to 2 ms, therefore 2000 μs have to be set here.
• If the "Distributed Clocks" option is activated for all controllers and communication is successful, the EtherCAT Master provides the "DC In-Sync" message:

8.1.2 Adding devices - CANopen bus system (optional)

In order to map the desired machine structure on the basis of the ApplicationTemplate, the corresponding devices have to be added in the Device view.

How to proceed:

1. Select the controller

• In the context menu, execute the Add device command to add "CANbus" to the bus system.
Working with the ApplicationTemplate
Mapping the actual machine structure in the »PLC Designer«

2. Set the baud rate via the **CANbus** tab.

   ![CANbus tab](image)

   **Note:** The value of the transfer rate set in the »PLC Designer« overwrites any existing transfer values of the devices (set via »WebConfig«/»Engineer«).

3. In a CANopen network, set the same baud rate for all nodes.

4. Use the 1 **Add Device** command to extend the control configuration by 2 **"CANopen Manager"**.

   ![Add Device](image)
5. Go to the **CANopen_Manager** tab to set the parameters for sync generation.

![CANopen_Manager](image)

The sync producing is required if ...
- ...at least one PDO with sync-controlled processing is used on the bus;
- ...the applications are to run synchronously in several devices;
- ...motion devices are to be connected to the bus.

If you want to use CAN synchronisation, check the **EnableSync Producing** input field.

1. **Set Sync cycle time: 2 ms (input value: 2000 µs)**
   - The value for the CANopen master cycle time has to be defined according to the cycle time of the quickest task.
6. Use the **Add Device** command to insert a logic device (slave) under the CANopen_Manager (master).

Select the desired CAN node from the **2** selection list.

**Tip!**
A general device description is provided for the integration of inverters: **Lenze Generic Drive**.

Repeat the **1 Add device** command until all slaves connected to the bus are implemented in the device view. Optionally, you can **Copy** and **Paste** a node already inserted into the context menu of the device. The communication settings (node ID and baud rate as well as further parameters) must be adapted manually afterwards.
7. Allocate unique designations to the slaves inserted (example: "L_94_HL_ActuatorSpeed").

   The names can be selected freely and must …
   • only contain the characters "A ... Z", "a ... z", "0 ... 9", or "_";
   • not start with a digit.

8. You can enter a name by clicking the element.

   Example:

   ![Diagram showing allocation of unique designations]

   • Below the selected slave, the axis data are available in the form of an additional node.

   In order to be able to link the process data manually (instead of the automatic link to the L_LCB_Axis_REF instance), the Manual I/O mapping option has to be activated.
Setting of CAN parameters and PDO mapping
Set the CAN parameters and the PDO mapping for each Logic device connected to the bus.

How to set CAN parameters and CAN mapping:
1. Change to the CANopen Remote Device tab of the corresponding slave.

Assign the suitable node addresses of the devices in the Node ID input field:

Only if the Expert settings option is set, the following settings are possible and visible:
- "Node Guarding"
- "Emergency"
- "Heartbeat"
- "Checks at Startup"

Make the settings required for your application here.
2. Change to the **PDO Mapping** tab.

By default, the PDO mapping is optimised for the corresponding application. It is possible to change this mapping (set checkmark). Due to the limited bandwidth of the CAN bus, this is only sensible in special cases. The PDO properties are pre-assigned sensibly as well and should not be changed.

By double-clicking a single PDO, you can see its transmission properties.

- The **Transmission Type** "cyclic - synchronous (type 1-240)" and the information at which sync the PDOs are to be transmitted must not be changed.

- The settings of the **Inhibit time** and the **Event time** are not evaluated.
- Confirm the settings with **OK**.

The **CANopen I/O Mapping** tab serves to assign PLC variables to the process image.

---

**Note!**

**Bus operation without CAN synchronisation**

- The controller always transmits asynchronous PDOs from an unsolicited task in an event-controlled way. To achieve a time-controlled transmission of asynchronous PDOs by the controller, you must assign the CAN master to a cyclic task.
- The controller does not support any monitoring times for asynchronous receipt PDOs. These monitoring times have to be set in the devices.
3. The selection of a special bus cycle task on the CANopen I/O image of the CANopen manager is not mandatory.

The standard setting uses the task with the shortest cycle time that accesses the devices of the CAN master as bus cycle task:
Creating your own machine modules: Copy/insertMM_EmptyModule

**Tip!**

If individual machine modules are created, the ApplicationTemplate automatically assigns the corresponding machine module-internal names.

Creating machine modules: What has to be done?

The **EmptyModule** machine module template...

- ...has to be copied in the A65 EmptyModule folder using the Copy Empty Module command and
- ...then using the Insert Empty Module command to insert it into the A70_MachineModuleSources folder.

The machine control module (MM_Machine) has to be created on your own according to the individual requirements.

**How to proceed:**

1. Copy the **MM_EmptyModule** machine module:

   - Right-click the A65 EmptyModule\MM_EmptyModule folder → Copy Empty Module.

2. Insert the previously copied machine module (Empty Module) below the A70_MachineModuleSources folder:

   - Right-click the A70_MachineModuleSources → Insert Empty Module
3. Enter the desired module name.
   The module name can be selected freely and ...
   • must not contain "MM_"
   • may only contain the characters "A ... Z", "a ... z", "0 ... 9"
   • must not contain any special characters.

4. Click Insert to insert the machine module.
   • The machine module has been inserted with the applicable names of the MAPs/MFB, structures, and visualization.

5. The machine module inserted (MFB_*) is to be inserted in the MMT in the A10_MachineModuleTree folder.

6. The module application (MAP_*) in the A11_ModuleAppCalls folder is to be inserted in the desired module application call (MAC_Task_*).

8.3 Creating instances of machine modules: Create MM instance

For creating further machine module instances, use the Create MM Instance command in the ApplicationTemplate. Example: Creating an instance of the machine module Module1 in the Application Template Counter.

How to proceed:
1. Instancing the machine module MM_Module1:
   • Right-click the folder A70_MachineModulesSources → Create MM Instance:
2. Afterwards, the MM Project tab shows the machine modules which are located in the 
A70_MachineModulesSources folder.

Note: The MM Library tab serves to create instances of machine modules of a library (in preparation!).

• Mark the desired machine module from which you want to create an instance.
• Enter instance names in the "User Defined Base Name" input field, example: TestModule.

3. Click Insert to create an instance of the machine module and its structures in all global variable lists.
8.4 **Integrating a machine module into the MMT**

The machine module (created on the basis of the **MM_EmptyModule** template) has to be integrated into the **MachineModuleTree (MMT)** to implement it in the **ApplicationTemplate** in a serviceable state.

How to proceed:

1. Double-click the **A10_MachineModuleTree** folder in the device view.
   - Double-click **MMT (PRG)**.
   - **Note:** The programming language of MMT (PRG) is CFC (Continuous Function Chart).
   - In the Tools dialog window, click the **Block** button.
   - Create the new FB via drag-and-drop.
   - Double-click the **??** area of the FB. Click the **___** button.

   Use the **Input assistance** from the **Application** element...
   - ...to assign the **MFB_CrossCutter** FB.
   - ...to assign the **CrossCutter** instance.
   - **Note:** Go to the **Input assistant** and select the **Insert with namespace prefix** when assigning the instance name.

2. Specify the relative address for the machine module.
   - In the Tools dialog window, click the **Input** button.
   - Add the new input at **MM_Address**
   - Assign the relative address (example: 3).

3. Connect FBs **MFB_MachineControl** and **MFB_CrossCutter** to each other.
   - Example (ApplicationTemplate Counter):

   **Note:** After inserting a machine module, the processing order of the machine modules which results after inserting the further machine module is to be checked.

   For changing the Processing order, adapt the element number in the top right corner of the corresponding machine module.
   - For this, call the **Processing order** menu item in the context menu of the element
Assigning the module application (MAP) to the task

For machine modules that have been created using the EmptyModule template, the MAP and task can be easily assigned via a dialog window.

**Tip!**
For creating machine modules, use the EmptyModule template in the A65_EmptyModule folder to assign the module applications to the tasks easier.

**How to proceed:**

1. Right-click the A11_ModuleAppCalls folder:
   - **Call Create Task Call.**
2. Go to the following dialog window and mark the module application in the area (example: MM_NewModule) and in area, mark the task to be assigned to the MAP.
   - Assign/unassign the task to the respective MAP using << / >>.

[8-1] Example: ModuleApp1 is assigned to the task with the highest priority Task_High.

- Confirm assignment by clicking **OK**.
## Manual assignment

**How to proceed:**

1. Double-click the `A11_ModuleAppCalls` folder.
2. Add the module application call in the program part of the corresponding task (example: `MAC_Task_high`).

**Note:** The programming language of `MAC_Task_xxx` (PRG) is FBD (function block diagram).

- Use the `Input assistance` to assign the structure `MM_Par.sc<module name>` (element: `Application`).

**Note:** Activate the `Insert with namespace prefix` option in the `Input assistant`.

### Results:
The machine module...

- ...is now integrated into the machine module tree (MMT).
- ...can be controlled by the internal state machine.
8.6 Remove the instances of a machine module

The previously inserted instances of a machine module have to be removed from the global variable lists in the □ A55_VarLists folder.

How to proceed:
1. Double-click the □ A55_VarLists folder.
2. Double-click the global □ MM_Dcl variable list.
3. Right-click the instance to be deleted (example: NewModule):

8.7 Removing machine modules

In order to remove a machine module including all instances from the »PLC Designer« project, execute the □ Delete command.

How to proceed:
Right-click the module name in the □ A70_MachineModuleSources folder and execute the □ Delete command.

8.8 Module ID

• Each machine module can be identified by a unique module ID which is stored in every MFB of the machine module in each case.
• The module ID is defined with the name CompID in the BaseChannel and can be set by the setCompIDAndVersion() method.

[8-2] Example: Structured overview of the components of an MFB_MachineControl of the MM_Machine machine module
• The respective module ID (CompID) can be viewed in the visualization under *Machine Module Details*: Folder A20_Visualisation→ L_Main, Module List button.

Tip!
Allocate a suitable module ID *CompID* for every newly created machine module.
When revising a module, update the *Version* accordingly.
8.9 Inserting an axis

In order to connect an axis to a machine module, the following has to be observed when creating a machine module.

- The following example shows how an axis is to be connected to the **MM_Module2** machine module.
- The axis is to be connected to the task with the highest priority **MAC_Task_High**.

**How to proceed:**

1. Call the module application within which an axis is to be connected:
   - Extend the module application by the declaration: `AXIS: AXIS_REF_SM3`

2. In the **A11_ModuleAppCalls** folder, update the task into which the module application is integrated: **MAC_Task_High**
   - Delete the present module name (here: **MAP2_Modul2_App1**) in the FB of the module application:

3. Assign the name of the module application (example: **MAP_Module2_App1**) to the FB:
4. Assign the name of the instance (example: MM_Dcl.Module2.App1) to the module application:

5. Assign the axis to the module application. The precondition for this is that the actual machine structure is mapped in the »PLC Designer« project. [Mapping the actual machine structure in the »PLC Designer« (LJ 41)] (The dwTestCounter input does not receive a transfer variable.)

- Select the desired axis (example: SM_Drive_ETC_9400HL)
- Click OK to insert the axis.
- The axis is connected to the machine module:
8.10 Integrating I/O modules of the I/O system 1000 with a machine module

This chapter describes in which way the modules of an I/O system connected via CAN can be accessed from a machine module.

Example: Access three digital inputs/outputs and one analog input /O system 1000 from the MM_Module1 machine module.

How to proceed:

1. Right-click I_O_module_coupler
   - Execute the Start Search menu command

The following dialog window shows all I/O modules identified.

- Click the Copy all device to project button to insert all devices found into the project.
2. The variable names that are used in the machine module and that are connected to the I/O system are summarised in the MM_IO structure.

3. Assign the variables to the physical I/O modules.

4. Activate the **Always update variables** option by ticking the checkbox (✓):
8.11 Creating module applications

This chapter describes how to create a module application.

Tip!
In the ApplicationTemplate, all positions that are to be edited if a new module application is to be created are marked with the `AT_ACTION_CREATE_NEW_MODULEAPPLICATION` keyword.

The search function in the »PLC Designer« makes it easier to find the positions to be edited:
- Execute the menu command `Edit→Find&replace→Find`
- Enter the required keyword and search/continue to search in order to navigate to the corresponding positions in the ApplicationTemplate.

Example: Extend the `MM_Module1` machine module by a module application.

- Initial situation:
  - A machine module (example: `MM_Module1`) is to be extended by a module application.
  - Create the module application in the following folder of the machine module: `MM_Module1\ModApp1`

How to create a module application:

1. Copy the `MM_Module1\ModApp1` folder (right click: Copy) and insert it in the `MM_Module1` folder (right click: Paste).

Rename the following elements (by right click Properties):
- The folder name (previously: `ModApp1`): `ModApp2`
Working with the ApplicationTemplate
Creating module applications

• The FB name (previously: MAP_Module1_App1): MAP_Module1_App2

The module application has now been inserted.

The module application inserted is to be declared in the MFB_Module1 function block:

8.11.1 Adapting the function of the module application inserted - overview

Programming has to be adapted in the following folders to achieve the desired serviceability of the module application:

1. Adapt the required parameters such as error responses and error numbers.

2. If the state machine of this module application is to work differently than module application 1, the states are to be programmed correspondingly.
Frequently asked question: Connection between MFB and MAP

**MFB_Module1 function block**

The MachineModuleTree (MMT) maps the mechatronic structure of the automation system by means of the MFB function blocks (MFB_Modulexx).

- Instancing of the **MFB_Module1** must be executed in the **MM_Dcl** structure of the **A55_VarLists** folder.
- Insert the instance in the desired position of the MMT.

![Machine module tree example](image)

**Example: Machine module tree in the ApplicationTemplateCounter**

- A machine module encloses the module applications in the machine module.
- Therefore the module applications are to be declared within the machine module.

Instance the **MAP_Module1_App1** module application with the following declaration line, so that the instance can be called according to the task in the module application of ModuleAppCalls:

```plaintext
App1: MAP_Module1_App1 := (parentModule := THIS);
```

- The `parentModule := THIS` instruction connects the **App1** module application to the basic functions of all machine modules (e.g. state machine, error handling, BaseChannel). That way, the **App1** module application can access the basic functions of the ApplicationTemplate.
- Assign the ACD structure manually. [The AppChannelData structure (ACD) (§ 75)]

![ACD structure example](image)
Working with the ApplicationTemplate

Creating module applications

8.11.3 Programming with the module application

In order to structure the subfunction of a module application, the ApplicationTemplate actions are to be used. Tip! Programming recommendations for structuring (page 29)

Tip!

For programming with the ApplicationTemplate it is recommended to use the methods included in the ApplicationTemplate.

- State machine (page 79)
- Error handling (page 94)
- The AppChannelData structure(ACD) (page 75)

Note!

A machine module must contain at least one module application which is assigned to a ModulAppCall (and thus to a task)!
8.11.4 Integrating a module application

This section describes the procedure for integrating a newly created module application into a machine module.

**Tip!**

In the ApplicationTemplate, all positions that are to be edited if a new module application is to be integrated are marked with the AT_ACTION_ADD_MODULEAPPLICATION keyword. The search function in the »PLC Designer« makes it easier to find the positions to be edited:

- Execute the menu command Edit→Find&replace→Find.
- Enter the required keyword and search/continue to search in order to navigate to the corresponding positions in the ApplicationTemplate.

**Example:** Integration of the module application into the Task_Mid task.

How to integrate a module application:

Taking the MM_Module1 machine module as a basis, the example shows how the newly created App2 module application is to be integrated into the ApplicationTemplate.

1. Call of the newly created module application in the corresponding task (A11_ModuleAppCalls folder, MAC_Task_Mid block).
   - Highlight the MAP_Module block.
   - Copy and Paste the block (right click: Copy/Paste)
Working with the ApplicationTemplate

Creating module applications

2. Integrate the new module application (MAP).
   • Click the FB name
     ![Diagram showing FB connections]
   • Call the *Input assistance* by clicking the *** button.
   • Use the *Input assistance* of the *Application* element to select the desired module application which is to be integrated into the task.

   Example: MAP_Module1_App2 module application

3. Click OK to integrate the module application.
4. Adapt the instance name correspondingly:
   • Specify applicable instance name, i.e.: MM_Dcl.Module1.App2

   ![Diagram showing instance name change]
Further information about extending the visualization can be found in the following section:  
- Extending the visualization ([110])

8.12 Online change

“Online change” means only loading the changed program/data parts of a running application to the controller.

- Thus no re-initialisation (warm restart, reset) takes place.
- The "Online change" takes place after all tasks have been processed (task interval).

**Note!**

An "Online change" is only possible if a compiler listing is available. After the **Clean all** and **Clean application** commands have been executed, an "Online change" is not possible, since the cleaning process deletes all compile information (compiler listing).

8.12.1 When can an "Online change" be used in the ApplicationTemplate?

In the following application cases, an "Online change" can be executed without restrictions:

- Changes on the **Program code** and application data which have no impact on the following areas:
  ...the Machine Module Tree (MMT) structure
  ...the integration of machine module basic functions (for example the **L_EATP_ErrorSet** function block for triggering errors).
- When the structure is changed and when machine module basic functions are integrated into the ApplicationTemplate, an **Online change is not to be used!**
8.12.2 When is it not possible to use an "Online change" in the ApplicationTemplate?

### Note!

Especially with regard to time-critical Motion applications, an Online change must only be executed if all controllers with Motion function are at standstill.
If they are at standstill, a short-time interruption of the task timing (normally) does not take effect.

In the following application cases, it is to be avoided that an "Online change" is executed:
- If the structure of the MMT is changed: e.g. when machine modules are added / removed.
- If the access to basic function blocks is changed: e.g. additional instances of the `L_EATP_ErrorSet` function block for triggering errors.

### Technical background information

A regeneration of the ApplicationTemplate structure (due to changes regarding the structure) is only possible after a restart (at least warm restart). This requires a new Log in including a download.
- An "Online change" executes the changes on the application program during the "task interval" (all tasks have completed their process).
- The exchange of the corresponding program blocks/data blocks by the runtime system causes a time delay, so that negative effects on the task timing might occur.

### Information regarding the time response

The following factors have the effect that the task process immediately after the online change starts with a delay/is ineffective.

In the case of a high task utilisation (ratio of the task runtime and the task interval) and a resulting short task interval

and/or

the shifting of variables required for an Online change.

8.12.3 General programming recommendations which enable an "Online change"

The following factors must be observed during programming to enable an online change:
- Avoid pointers and references

Basically avoid the use of pointers and references. Then no special precautions have to be made for an online change to ensure normal operation of the application.

- Avoid pointers and references in every clock cycle

If pointer and reference values have to be re-assigned in each clock cycle, it has to be ensured that the referenced memory addresses have the correct value in the first clock cycle after an online change.

Please note that, when assigning a reference or pointer value to another cycle, the source value has to be updated in the clock cycle first, since otherwise the target of the assignment will show the previous (old) value for at least one clock cycle.

- Use {attribute ‘init_on_onlchange’}

Note!

Especially with regard to time-critical Motion applications, an Online change must only be executed if all controllers with Motion function are at standstill.
If they are at standstill, a short-time interruption of the task timing (normally) does not take effect.

In the following application cases, it is to be avoided that an "Online change" is executed:
- If the structure of the MMT is changed: e.g. when machine modules are added / removed.
- If the access to basic function blocks is changed: e.g. additional instances of the `L_EATP_ErrorSet` function block for triggering errors.

### Technical background information

A regeneration of the ApplicationTemplate structure (due to changes regarding the structure) is only possible after a restart (at least warm restart). This requires a new Log in including a download.
- An "Online change" executes the changes on the application program during the "task interval" (all tasks have completed their process).
- The exchange of the corresponding program blocks/data blocks by the runtime system causes a time delay, so that negative effects on the task timing might occur.

### Information regarding the time response

The following factors have the effect that the task process immediately after the online change starts with a delay/is ineffective.

In the case of a high task utilisation (ratio of the task runtime and the task interval) and a resulting short task interval

and/or

the shifting of variables required for an Online change.

8.12.3 General programming recommendations which enable an "Online change"

The following factors must be observed during programming to enable an online change:
- Avoid pointers and references

Basically avoid the use of pointers and references. Then no special precautions have to be made for an online change to ensure normal operation of the application.

- Avoid pointers and references in every clock cycle

If pointer and reference values have to be re-assigned in each clock cycle, it has to be ensured that the referenced memory addresses have the correct value in the first clock cycle after an online change.

Please note that, when assigning a reference or pointer value to another cycle, the source value has to be updated in the clock cycle first, since otherwise the target of the assignment will show the previous (old) value for at least one clock cycle.

- Use {attribute ‘init_on_onlchange’}
Reference or pointer variables (as well as other variables) provided with this attribute will be set to their initial value after an online change.

- Thus the occurrence of an online change can be identified.
- Thus the initialisation processes can be executed again (after an online change has been completed).

- Use interface references

During an online change, interface references (interfaces) are automatically checked and adapted.

When interfaces and interface references are used for the ApplicationTemplate, please note that the use is not required in a purely functional way and that it is only required for the purpose of the online change.

In order to be able to access an FB instance, an additional method call of the interface (pointer/reference to the FB instance) is required.
Architecture: The ApplicationTemplate in detail

9 Accessing structure variables of machine modules

9.1 Accessing module’s intrinsic structure variables

The structures of the machine modules contain the input and output variables of the machine modules for integrating data in the complete system:

- I/O data: Structure MIO
- Parameters / data of the recipe manager: Structure MPar
- Persistent data: Structure MPD
- Visualization data: Structure MVIs

The following example shows how a structure variable for an external visualization (assigned to the MAP) within a module application is to be called (when a module with automatically matching names is assigned, the matching entries will be added automatically):

```plaintext
// MVIs_scMachine
// for MAP1
diPieces : DINT;

// for MAP2
diLength : DINT;
```

[9-1] Extract from the MVIS_scMachine structure

```plaintext
VAR_IN_OUT
/// ------------------ START OF USER SPECIFIC BLOCK ---------------------

/// Add the needed declarations of Par, Vis, IO and PD access here
Par: MPar_scMachine;
Vis: MVIs_scMachine;
IO: MIO_scMachine;
PD: MPD_scMachine;

/// ------------------- END OF USER SPECIFIC BLOCK ---------------------

END_VAR
```

[9-2] Extract from the module application (MAP)
The associated program call in the machine application has the following syntax:

```
Vis.diPieces := 4711;
```

[9-3] Sample program which assigns the value 4711 to the `diPieces` visualization variable.

### 9.1.2 Accessing structures of other machine modules

The global machine module structures are instanced in the `A5S_VarLists` folder.

- Like this, the structures and machine modules of the ApplicationTemplate can be reused.

- The structure variables of an optional machine module can be used from another machine module.

- In order to be able to use the variables of a global structure of machine module 1 (example: `MM_Module1`) in another machine application of machine module 2 (example: `MM_Module2`), the following program call is required:

```
diPiecesModul1 := MM_Vis.scModule1;
```

9.2 The AppChannelData structure (ACD)

If required, the data structure of the MACD_sc type module name can be assigned within the desired machine module (MFB).

The data structure must inherit the L_EATP_ACD_Base contained in the L_EATP_ApplicationTemplate library. ▶ The L_EATP_ApplicationTemplate library ( Libya 116)

- The MACD_sc Module structure defines the diCounter variable of the DINT data type.

```
TYPE MACD_scModule EXTENDS L_EATP_ACD_Base :
STRUCT
  // declare here the specific application data
  diCounter : DINT;
END_STRUCT
END_TYPE
```

[9-5] Configuration of the ACD structure with an inherited L_EATP_ACD_Base structure

- The MACD_sc Module structure can be individually extended by the data types required.
- By the inherited structure, specific control and status bits are provided in the ACD structure.
- In the structure, the L_EATP_CriticalSection block which ensures a consistent data exchange is instanced (if the FB is used correctly). ▶ L_EATP_CriticalSection ( Libya 123)

More information about the L_EATP_ACD_Base structure: ▶ L_EATP_ACD_Base (ACD structure) ( Libya 133)

9.2.1 Declaring/recording the ACD structure in the MFB

In order to be able to use the ACD structure in the module applications (MAPs) of an MFB and its master, the instance of the ACD structure has to be declared locally in the MFB.

```
FUNCTION BLOCK MFB_Module1 EXTENDS L_EATP_Module
  VAR TN IN
  END_VAR
  VAR INIT
  END_VAR
  VAR OUTPUT
  END_VAR
  VAR
    // Declaration of the Module Application Channel data (ACD) instance
    // ACD : MACD_scModule1;
END_FUNCTION_BLOCK
```

[9-6] Declaration of the ACD structure in the MFB Module

The ACD structure is recorded in the program part of the MFB to activate safety mechanisms.

- The instance name and size of the structure are to be transferred to the RegisterACD() method.
- The structure is to be newly recorded during every cycle. (A missing recording triggers an infrastructure error.)
9.2.2 Accessing the ACD structure - by means of the MFB module application

In order to access the ACD structure by means of the module application (MAP) of the MFB, ...
- ...a POINTER to the ACD structure and
- ...a reference to the ACD structure have to be declared in the desired module application.

Declaration of the POINTER and REFERENCE to the ACD structure in the MAP of the MM_Module1 machine module
In the program part of the module application (MAP), access to the ACD structure by the `AccessACD()` method is to be ensured:

This renders a read/write access by means of the reference and the Intellisense function to the data contents of the ACD structure possible:

9.2.3 Accessing an ACD structure - by means of the module application of the MFB master module

In order that an application of the master module can access the ACD structure of a lower-level machine module, a pointer and reference to the ACD structure of the slave have to be declared in the declaration part of the master module application.
In the program part of the master module application (MAP), access to the ACD structure by the `AccessACD()` method is to be ensured:

This renders read/write access by means of the reference and the Intellisense function to the ACD structure of the slave possible:

Command for reading the data field of the ACD structure
9.3 State machine

The machine module (which is on the highest level of the MachineModuleTree MMT) controls the state machine of all lower-level machine modules.

- The resulting state transitions have an impact on the subordinated machine modules.

9.3.1 State transitions and conditions - overview

This section illustrates the individual state transitions of the ApplicationTemplate state machine. For each state transition, the conditions required in each case are specified. The state transitions are numbered and then described in the text:

<table>
<thead>
<tr>
<th>Labelled with</th>
<th>State/identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialisation</td>
<td>Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initialisation of the module / module application is always required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• After a RESET, the &quot;Init&quot; → &quot;Ready&quot; state transition is inhibited.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Then the &quot;Init&quot; → &quot;Ready&quot; state transition is to be enabled by the module application (when initialisation has been completed).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The SMEnableInitToReady(TRUE/FALSE) method enables / inhibits the state transition.</td>
</tr>
<tr>
<td>2</td>
<td>Idle state</td>
<td>Ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ready for operation. This is the basic state of the state machine.</td>
</tr>
<tr>
<td>3</td>
<td>Manual operation</td>
<td>Manual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enable/inhibit mechanism</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• In this state, the controller can be controlled in manual mode, for instance for cleaning or changing the tool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• By setting the individual control bits, the controller can be controlled manually (Jog1, Jog2, QSP, ErrorReset...).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• This state permits the module applications to inhibit / enable the underlying state transition.</td>
</tr>
<tr>
<td>4</td>
<td>Automatic</td>
<td>Automatic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Automatic operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MM_Module1: Incrementing (50 units per second)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MM_Module2: Incrementing (20 units per second)</td>
</tr>
</tbody>
</table>
Special "Warning" status

"Warning" is a special status (independent, "orthogonal" state), which does not influence the operational performance of the machine module.

- Possible states:
  - ...active: "Warning" (or: ...not active: "No Warning")

9.3.2 Initial state of the state transitions

The state transitions between the module state machine and the module applications are in the initial state as follows:

<table>
<thead>
<tr>
<th>State transition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-13</td>
<td>Are enabled in the initial state (after RESET)</td>
</tr>
</tbody>
</table>

By means of methods, the state transitions can be influenced (enable/inhibit).
### The state transitions in detail

The "Init" status is reached immediately after mains switching/device reset.

- The module application executes the initialisation. After the initialisation phase, the state machine is to be set to the "Ready" status by setting the $x_{\text{EnableInitToReady}}$ input.
- The following input conditions must be met so that the respective state transition is reached:

<table>
<thead>
<tr>
<th>Number</th>
<th>State transition</th>
<th>Condition</th>
</tr>
</thead>
</table>
| 1      | "Init"→"Ready"   | • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM\_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Ready}); and • State transition is enabled by the module application ($x_{\text{EnableInitToReady}}$ input = TRUE)  
**Note:** When this method is passed through by every machine module (MM) during the initialisation phase, all MMs are in the "Ready" status. |
<p>| 2      | &quot;Ready&quot;→&quot;Manual&quot; | • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Manual}); and • State transition is to be enabled by the module application in every subordinated machine module ($x_{\text{DisableReadyToManual}}$ input = FALSE) |
| 3      | &quot;Ready&quot;→&quot;Service&quot;| • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Service}); and • State transition is enabled by the module application ($x_{\text{DisableReadyToService}}$ input = FALSE) |
| 4      | &quot;Ready&quot;→&quot;Automatic&quot;| • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Automatic}); and • State transition is enabled by the module application ($x_{\text{DisableReadyToAutomatic}}$ input = FALSE) |
| 5...7  | &quot;Ready&quot;→&quot;Mode(1...2)&quot; | • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Mode(1...2)}); and • State transition is enabled by the module application: ($x_{\text{DisableReadyToMode(1...2)}}$ input = FALSE) |
| 8      | &quot;Manual&quot;→&quot;Ready&quot; | • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Ready}); and • Enable of the state transition by the module application ($x_{\text{DisableManualToReady}}$ input = FALSE) |
| 9      | &quot;Service&quot;→&quot;Ready&quot; | • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Ready}); and • State transition is enabled by the module application ($x_{\text{DisableServiceToReady}}$ input = FALSE) |
| 10     | &quot;Automatic&quot;→&quot;Ready&quot; | • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Ready}); and • State transition is enabled by the module application ($x_{\text{DisableAutomaticToReady}}$ input = FALSE) |
| 11...13| &quot;Mode 1...2&quot;→&quot;Ready&quot; | • $\text{SetState}$ request by the master baseChannelSetNominalState($\text{MM_Address}$:= 0, $\text{eState}$:=L_EATP_SMStates.\text{Ready}); and • State transition is enabled by the module application: ($x_{\text{DisableMode(1...2)}To\text{Ready}}$ input = FALSE) |</p>
<table>
<thead>
<tr>
<th>Number</th>
<th>State transition</th>
<th>Condition</th>
</tr>
</thead>
</table>
| 14 A   | “Ready” → “Quick stop” | • The module’s intrinsic error handling sets the state machine to the “Quick stop” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Quick stop);` |
| 14 B   | “Ready” → “Fault” | • The module’s intrinsic error handling sets the state machine to the “Fault” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Fault);` |
| 14 C   | “Ready” → “SystemFault” | • The module’s intrinsic error handling sets the state machine to the “SystemFault” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.SystemFault);` |
| 15 A   | “Manual” → “Quick stop” | • The module’s intrinsic error handling sets the state machine to the “Quick stop” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Quick stop);` |
| 15 B   | “Manual” → “Fault” | • The module’s intrinsic error handling sets the state machine to the “Fault” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Fault);` |
| 15 C   | “Manual” → “SystemFault” | • The module’s intrinsic error handling sets the state machine to the “SystemFault” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.SystemFault);` |
| 16 A   | “Service” → “QuickStop” | • The module’s intrinsic error handling sets the state machine to the “Quick stop” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Quick stop);` |
| 16 B   | “Service” → “Fault” | • The module’s intrinsic error handling sets the state machine to the “Fault” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Fault);` |
| 16 C   | “Service” → “SystemFault” | • The module’s intrinsic error handling sets the state machine to the “SystemFault” status  
|        |                  | or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.SystemFault);` |
| 17 A   | “Automatic” → “QuickStop” | • The module’s intrinsic error handling sets the state machine to the “Quick stop” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Quick stop);` |
| 17 B   | “Automatic” → “Fault” | • The module’s intrinsic error handling sets the state machine to the “Fault” status or  
|        |                  | • `SetState` request by the master  
|        |                  | `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Fault);` |
## Architecture: The ApplicationTemplate in detail

### State machine

<table>
<thead>
<tr>
<th>Number</th>
<th>State transition</th>
<th>Condition</th>
</tr>
</thead>
</table>
| 17     | *Automatic*   → *"SystemFault"* | • The module's intrinsic error handling sets the state machine to the "SystemFault" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.SystemFault);` |
| 18     | *Mode 1*   → *"Quick stop"* | • The module's intrinsic error handling sets the state machine to the "Quick stop" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Quick stop);` |
| 18     | *Mode 1*   → *"Fault"* | • The module's intrinsic error handling sets the state machine to the "Fault" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Fault);` |
| 18     | *Mode 1*   → *"SystemFault"* | • The module's intrinsic error handling sets the state machine to the "SystemFault" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.SystemFault);` |
| 19     | *Mode 2*   → *"QuickStop"* | • The module's intrinsic error handling sets the state machine to the "Quick stop" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Quick stop);` |
| 19     | *Mode 2*   → *"Fault"* | • The module's intrinsic error handling sets the state machine to the "Fault" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Fault);` |
| 19     | *Mode 2*   → *"SystemFault"* | • The module's intrinsic error handling sets the state machine to the "SystemFault" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.SystemFault);` |
| 20     | *Homing*  → *"QuickStop"* | • The module's intrinsic error handling sets the state machine to the "Quick stop" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Quick stop);` |
| 20     | *Homing*  → *"Fault"* | • The module's intrinsic error handling sets the state machine to the "Fault" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.Fault);` |
| 20     | *Homing*  → *"SystemFault"* | • The module's intrinsic error handling sets the state machine to the "SystemFault" status or  
  • `SetState` request by the master  
    `baseChannelSetNominalState(MM_Address:= 0, eState:=L_EATP_SMStates.SystemFault);` |
| 21     | *"Quick stop"*  → *"Ready"* | • After acknowledging the error (eliminate error cause and execute "ErrorQuit"), and  
  • Automatically after a quick stop (Execute "Quickstop")  
  [Error handling (LJ 94)] |
| 22     | *"Fault"*   → *"Ready"* |  |
| 23     | *"No Warning"* → *"Warning"* | • The module's intrinsic error handling sets the state machine to the "Warning" status. |
| 24     | *"Warning"* → *"No Warning"* | • The module's own error handling sets the state machine to the "No Warning" status. |
9.3.4 Mapping of the states - Enum L_EATP_SMStates

The different states of the state machine are mapped in the L_EATP_SMStates enumeration.

- **L_EATP_SMStates** (137)

9.3.5 Activating the states - baseChannelSetNominalState method

The ApplicationTemplate provides methods to set the machine modules to a specific state.

**Note!**

This method can be used if...

- ...the .DisableDefaultCouplingSlave function (module's intrinsic target status) is activated, or
- ...the .DisableDefaultCouplingMaster function (target status of a slave module) is activated.
- ...the machine control module (MM in the top level of the MachineModuleTree (MMT)) sets itself to a specific state, so that the subordinated modules change to the same status.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseChannelSetNominalState</td>
<td></td>
<td>Set the target status of the state machine for the corresponding slave module/own module.</td>
</tr>
<tr>
<td>MM_Address</td>
<td>L_EATP_MM_Address</td>
<td>• Address of the module the current state of which is to be queried.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The value &quot;0&quot; and the L_EATP_CONST.OWNID constant address the module's intrinsic ID.</td>
</tr>
<tr>
<td>eState</td>
<td>L_EATP_SMStates</td>
<td>• Setpoint status to be set.</td>
</tr>
</tbody>
</table>
Example

The MachineModuleTree has the following exemplary structure:

```
+---------------------------------+             +---------------------------------+
| M_F8_Machine                    |             | M_F8_Machine                    |
|                                 | 0            |                                 |
| masterBus                      |             | masterBus                      |
| MM_Address                     |             | MM_Address                     |
| slaveBus                       |             | slaveBus                       |
+---------------------------------+             +---------------------------------+
|                                 | 1            |                                 |
| masterBus                      |             | masterBus                      |
| MM_Address                     |             | MM_Address                     |
| slaveBus                       |             | slaveBus                       |
+---------------------------------+             +---------------------------------+
|                                 | 2            |                                 |
| masterBus                      |             | masterBus                      |
| MM_Address                     |             | MM_Address                     |
| slaveBus                       |             | slaveBus                       |
```

Example

The MFB_MachineControl machine control module sets the “Automatic” state for the lower-level machine modules:

```
baseChannelSetNominalState(MM_Address := L_EATP_CONST.OWNID, eState := L_EATP_SMStates.Automatic)
```

An overview of all BaseChannel methods can be found under: Method overview - ApplicationTemplate (§ 165)

9.4 Default coupling

The state machine in the ApplicationTemplate is provided with a standard response. In this section you'll be provided with information about how subordinated (slave) machine modules are to be decoupled from the standard response.

- The higher-level machine module specifies setpoint states for the lower-level slaves and, in doing this, checks the current actual states of the slaves.
- If a subordinated slave module detects an error...
  - ...this module responds with an error response (according to the error response programmed), and
  - ...forwards this information to the higher-level machine module which then changes to the "Quick stop" status.
- If there is another higher-level machine module, the information is forwarded to the top level and this module correspondingly responds with the "Quick stop" status. The top machine module then transfers the information back to its subordinated machine modules, which change to the "Quick stop" status.

This standard coupling can be deactivated by corresponding programming. This requires that the user defines the error behaviour by programming.

9.4.1 Standard mechanisms of the ApplicationTemplate

- The standard behaviour is active if the module in the machine module tree (MMT) is provided with at least one subordinated slave module.
- The standard behaviour can be controlled by means of the corresponding methods ("TRUE" is the initial value):

```
SMDisableDefaultCouplingMaster(TRUE)
SMDisableDefaultCouplingSlave (TRUE)
```
• Individual programming of the module application is not required.

**Rules defined by the standard mechanism:**

1. If at least one slave module or the machine module itself shows an error status ("Quick stop", "Fault", or "System fault"), the machine module sets the actual state to "Quick stop". All slaves receive the request to adopt the "Quick stop" state.

   **Example:**
   A machine module M features four subordinated slaves: S1, S2, S3, S4.
   • S2 goes to the "ErrorState FAULT" error status; S4 to "Quick stop".
   By this,
   • ...M changes to the "Quick stop" status,
   • ...S1, S2, S3 receive the request to change to the "Quick stop" status (S4 already is in the "Quick stop" status)

2. If no error is pending, the machine module forwards the module's intrinsic status request to all slaves. The module's intrinsic actual state transition only takes place when all slaves have changed to the status.

   **Example:**
   A machine module M features four subordinated slaves: S1, S2, S3, S4.
   • By the higher-level master module, M receives the request to change from the "Ready" status to the "Automatic" status.
   • Machine module M forwards the state transition to all slaves.
   • If all slaves (S1 ... S4) have completed the state transition and the module's intrinsic conditions for the state transition have been met, M changes to the "Automatic" status.
### 9.4.2 The "DisableDefaultCouplingMaster" function

The "DisableDefaultCouplingMaster" function (during operation) decouples the state machine of the higher-level master module from the associated slave modules.

- This function is required if...
  - ...specific machine parts / machine modules are to be decoupled specifically (during operation).
  - ...a different status is required for the decoupled machine parts / machine modules than for the other machine modules.
- Due to the addressed method access of the master, the state machine of the slave can be influenced. Alternatively, the state machine can be programmed within the slave. The error handling of the slave does not affect the state machine of the master.

**Example**

![Diagram](image)

The M1.2 master decouples its state machine from the state machines of the subordinated slaves M1.2.1 and M1.2.2. This means that the setpoint states for the subordinated slave modules M1.2.1 and M1.2.2 have to be programmed manually in the M1.2 machine module.

- The `SMDisableDefaultCouplingMaster(TRUE)` method call from a module application (MAP) activates the function.
- The `SMDisableDefaultCouplingMaster(FALSE)` method call deactivates the function.
- The method call has no impact on the connection to the ACD structure of the respective machine module.
- Optionally, the `baseChannelSetErrorQuit` method forwards the error acknowledgement to the slaves. The `L_EATP_GVL.xDisableGlobalErrorReset` variable serves to activate / deactivate a global error reset:

```plaintext
L_EATP_GVL.xDisableGlobalErrorReset := FALSE; //default setting
```

- The master of the respective state machine can read out the status of a decoupled module (via the corresponding methods).
- The use of this mechanism is only advisable if the module is provided with at least one subordinated slave module.

```plaintext
SMDisableDefaultCouplingMaster(xValue := TRUE);
```

**Sample program: Deactivating "DefaultCouplingMaster"**

- The master of the respective state machine can read out the status of a decoupled module (via the corresponding methods).
- The use of this mechanism is only advisable if the module is provided with at least one subordinated slave module.
"DefaultCouplingMaster" methods - overview

<table>
<thead>
<tr>
<th>Identifier/function</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseChannelGetDefaultCouplingMaster Disabled</td>
<td>MM_Address <code>L_EATP_MM_Address</code></td>
<td>- Queries the status of the <code>SMDisableDefaultCouplingMaster</code> method of the respective module.</td>
</tr>
<tr>
<td>SMDisableDefaultCouplingMaster</td>
<td>xValue</td>
<td>- Activates/deactivates the standard coupling of the master.</td>
</tr>
</tbody>
</table>

9.4.3 The "DisableDefaultCouplingSlave" function

The "DisableDefaultCouplingSlave" function has the effect that the slave module decouples from the state machine of the master module (during operation).

- This function is required if...
  - specific machine parts / machine modules are to be decoupled (during operation).
  - a different response is required for the decoupled machine parts/machine modules than for the higher-level machine module.

- The `SMDisableDefaultCouplingSlave(TRUE)` method call of a module application (MAP) activates the function to decouple the slave from the master.
- The `SMDisableDefaultCouplingSlave(FALSE)` method call deactivates the function.
- The module's intrinsic state machine cannot be coupled to the higher-level master module.
  - The state machine can control the subordinated slave modules (locally from the module application) via the BaseChannel methods.
  - That way, the MFB_MachineControl state machine of the lower-level slave modules acts just like the machine module with the activated `SMDisableDefaultCouplingSlave` method.
- The higher-level master module ignores the status of a slave module for which the `DisableDefaultCouplingSlave` function is activated.
- Optionally, the error acknowledgement can be set via the higher-level master:

```plaintext
L_EATP_GVL.xDisableGlobalErrorReset := FALSE; //default setting
```
Architecture: The ApplicationTemplate in detail

Default coupling

- The method call has no impact on the connection of the ACD structure of the respective machine module.
- The master of the state machine can read out a decoupled module via the corresponding `baseChannelGetActualState` methods.
  - Accessing the state machine - the methods of the BaseChannel (§ 165)
- The master of the respective state machine can read out the status of a decoupled module (via the corresponding methods). The use of this mechanism is advisable if the module is provided with at least one higher-level master module.

Error response - response for the occurrence of an error

- The `DisableDefaultCouplingSlave` method decouples the slave module from the state machine of the master module; example: decoupling M1.2.2 from M1.2.
- The Quick stop error status of M1.2 has no impact on M1 and M1.1 (the M1.2 module is decoupled from M1).
  - The state machine of the master module M1 no longer has an impact on the module M1.2 (due to the decoupling).
  - Error acknowledgement by the M1 master module no longer has an impact on M1.2 and its slave modules.

```c
SMDisableDefaultCouplingSlave(xValue := TRUE);
```

[9-19] Sample program: Deactivating "DefaultCouplingSlave"

"DefaultCouplingSlave" methods - overview

<table>
<thead>
<tr>
<th>Identifier/function</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseChannelGetDefaultCouplingSlaveDisabled</td>
<td>MM_Address _L_EATP_MM_Address</td>
<td>• Queries the status of the DisableDefaultCouplingSlave method of the respective module.</td>
</tr>
<tr>
<td>Return value/data type</td>
<td>BOOL</td>
<td>• Address of the module the current state of which is to be queried.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The value &quot;0&quot; and the _L_EATP_CONSTOWNID constant address the module's intrinsic ID.</td>
</tr>
<tr>
<td>SMDisableDefaultCouplingSlave</td>
<td>xValue _BOOL</td>
<td>• Activates/deactivates the standard coupling of the slave.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TRUE: Deactivate the standard coupling of the master.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FALSE: Activate the standard coupling of the master.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TRUE: Deactivate the standard coupling of the slave.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FALSE: Activate the standard coupling of the slave.</td>
</tr>
</tbody>
</table>
9.5 Influencing state transitions

The methods of the ApplicationTemplate can inhibit and re-enable the state transitions.

• All methods are provided with a request parameter of the BOOL data type. By setting "TRUE", the function of the respective methods (according to the method name) is executed.

<table>
<thead>
<tr>
<th>Method/function</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMDisableAutomaticToReady</td>
<td>xValue</td>
<td>• Controls the &quot;Automatic&quot;→&quot;Ready&quot; state transition</td>
</tr>
<tr>
<td>SMDisableReadyToAutomatic</td>
<td>xValue</td>
<td>• Controls the &quot;Ready&quot;→&quot;Automatic&quot; state transition</td>
</tr>
<tr>
<td>SMDisableReadyToManual</td>
<td>xValue</td>
<td>• Controls the &quot;Ready&quot;→&quot;Manual&quot; state transition</td>
</tr>
<tr>
<td>SMDisableReadyToMode1</td>
<td>xValue</td>
<td>• Controls the &quot;Ready&quot;→&quot;Mode1&quot; state transition</td>
</tr>
<tr>
<td>SMDisableReadyToMode2</td>
<td>xValue</td>
<td>• Controls the &quot;Ready&quot;→&quot;Mode2&quot; state transition</td>
</tr>
<tr>
<td>SMDisableReadyToHoming</td>
<td>xValue</td>
<td>• Controls the &quot;Ready&quot;→&quot;Homing&quot; state transition</td>
</tr>
<tr>
<td>SMDisableReadyToService</td>
<td>xValue</td>
<td>• Controls the &quot;Ready&quot;→&quot;Service&quot; state transition</td>
</tr>
<tr>
<td>SMDisableManualToReady</td>
<td>xValue</td>
<td>• Controls the &quot;Manual&quot;→&quot;Ready&quot; state transition</td>
</tr>
<tr>
<td>SMDisableMode1ToReady</td>
<td>xValue</td>
<td>• Controls the &quot;Mode1&quot;→&quot;Ready&quot; state transition</td>
</tr>
<tr>
<td>SMDisableMode2ToReady</td>
<td>xValue</td>
<td>• Controls the &quot;Mode2&quot;→&quot;Ready&quot; state transition</td>
</tr>
<tr>
<td>SMDisableHomingToReady</td>
<td>xValue</td>
<td>• Controls the &quot;Homing&quot;→&quot;Ready&quot; state transition</td>
</tr>
<tr>
<td>SMDisableServiceToReady</td>
<td>xValue</td>
<td>• Controls the &quot;Service&quot;→&quot;Ready&quot; state transition</td>
</tr>
<tr>
<td>SMEnableInitToReady</td>
<td>xValue</td>
<td>• Controls the &quot;Init&quot;→&quot;Ready&quot; state transition</td>
</tr>
</tbody>
</table>

Sample program: Inhibiting the "Service"→"Ready" state transition

SMDisableServiceToReady(TRUE)

Sample program: Enabling the "Service"→"Ready" state transition

SMDisableServiceToReady(FALSE)

Controlling the state transitions - an overview of the methods
9.6 Displaying the states of the state machine - FB L_EATP_SMAccess

The Statemachine FUP method serves to display the current state of the state machine from a module application (MAP). → L_EATP_SMAccess (Lj132)

- In the ApplicationTemplate, every machine module features one module application for which an instance of this block with the SMAccess instance name is created.
- The Statemachine FUP method calls the instance of the L_EATP_SMAccess(SMAccess) block.

9.7 Stater machine: Query examples

This section shows (by the use of program examples) how...

- ...to access the state machine.
- ...to query the states.

<table>
<thead>
<tr>
<th>Objective/call</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Querying the current state</strong></td>
<td></td>
</tr>
<tr>
<td>• IF condition</td>
<td>IF baseChannelGetActualState (MM_Address:= ) = L_EATP_SMStates.Manual THEN // Do something if Statemachine is in state MANUAL END_IF</td>
</tr>
<tr>
<td>• Module’s intrinsic/subordinated module status enquiry by method</td>
<td></td>
</tr>
<tr>
<td>• IF condition</td>
<td>IF StateMachine.xActStateIsAutomatic THEN // Do something if Statemachine is in state AUTOMATIC END_IF</td>
</tr>
<tr>
<td>• Query the module’s intrinsic status</td>
<td></td>
</tr>
<tr>
<td>• CASE instruction</td>
<td>CASE baseChannelGetActualState (MM_Address:= ) OF L_EATP_SMStates.Ready: // Do something if Statemachine is in state READY ; L_EATP_SMStates.Service: // Do something else if Statemachine is in state SERVICE ; L_EATP_SMStates.Quickstop: // Do something else if Statemachine is in state QUICKSTOP ; END_CASE</td>
</tr>
<tr>
<td>• Module’s intrinsic/subordinated module status enquiry by method</td>
<td></td>
</tr>
<tr>
<td><strong>Querying an active warning</strong></td>
<td></td>
</tr>
<tr>
<td>• IF condition</td>
<td>IF StateMachine.xWarning THEN // Do something if warning is active ELSE // Do something else if warning is not active END_IF</td>
</tr>
<tr>
<td>• Query the module’s intrinsic warning</td>
<td></td>
</tr>
<tr>
<td><strong>Querying the setpoint state of the master</strong></td>
<td></td>
</tr>
<tr>
<td>• IF condition</td>
<td>IF StateMachine.xActStateIsAutomatic THEN IF StateMachine.eSetState = L_EATP_SMStates.Ready THEN // Do something if actual state is AUTOMATIC and set state to READY ; END_IF END_IF</td>
</tr>
<tr>
<td>• Based on signals with eSetState signal</td>
<td></td>
</tr>
<tr>
<td><strong>Enabling state transition from “Init” to “Ready” (initialisation completed)</strong></td>
<td>SMEnableIlitToReady (xValue := TRUE);</td>
</tr>
</tbody>
</table>
9.8 Where can the response of a machine module be programmed?

The response of a machine module (in a specific state) can be programmed in the prepared Actions of the ApplicationTemplate.

- Actions are marked with the icon.

During every cycle, always the action representing the current status of the machine module’s state machine is passed through.

- Program parts which are only required if the machine module is in a specific state have to be stored directly within these predefined actions.

- The MM_EmptyModule module template contains a predefined structure with (empty) actions.

<table>
<thead>
<tr>
<th>Objective/call</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabling/inhibiting state transition from &quot;Automatic&quot; to &quot;Ready&quot;</td>
<td>Depending on a BOOL variable (here: &quot;xAutomaticEndOk&quot;)&lt;br&gt;IF xAutomaticEndOk = TRUE THEN&lt;br&gt;SMDisableAutomaticToReady (xValue := FALSE)&lt;br&gt;ELSE&lt;br&gt;SMDisableAutomaticToReady (xValue := TRUE);&lt;br&gt;END_IF</td>
</tr>
</tbody>
</table>

Example: Status-related actions of the MAP_Module1_App1 FB
9.8.1 State transition (state entry/state exit)

Every change of a status-related action (within an action or from an action) sets a specific flag for a task cycle to "TRUE".

- **xStateEntry**: The status-related action sets this flag to "TRUE" during the entry for one task cycle.
- **xStateExit**: The status-related action sets this flag to "TRUE" during the exit for one task cycle.

```plaintext
IF xStateEntry THEN
   // Put state entry code here
   // ...
END_IF

// Put cyclic code here
// ...
```

**Program example:** *S04_SERVICE action*

---

**Note!**

The following procedure is to be taken into consideration for task timing-related program commands: At a state transition, actions are assigned to the individual task cycles in the same way as for the sequential function chart AS:

- **At the "Ready"→"Automatic" state transition**, the task cycle...
  - ...first passes through the *S02_READY* action with a set xStateExit flag.
  - ...then passes through the *S05_AUTOMATIC* action with a set xStateEntry flag.
9.9 Error handling

The ApplicationTemplate contains an error handling for the output of error messages. It is defined by default and can be changed if required.

The error handling has the following functions:

- Errors/warnings/information can be defined in the module applications (MAP) of the machine modules.
- Forwarding of error states within the MachineModuleTree (MMT).
- An application-global error list with the current error status of all machine modules contained in the MMT.
- The error handling is connected to the logbook of the controller, so that the error messages can be viewed in the logbook.

9.9.1 Defining errors

The SetErrors method defines the errors with the corresponding properties in the respective module application.

The ApplicationTemplate contains the L_EATP_ErrorSet FB for changing the error handling. At the xSetError input of the L_EATP_ErrorSet FB, application-specific errors can be triggered. 

![L_EATP_ErrorSet FB](image)

- The L_EATP_ErrorSet FB has four inputs, xSetError[1...4], for setting errors.
- For every xSetError input, one error can be set.

An error can be defined on the FB by the following characteristics:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
<th>Further information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error number</td>
<td>Unique error number (ID)</td>
<td>16 bit error number (WORD data type)</td>
</tr>
<tr>
<td>Error response</td>
<td>Error response</td>
<td>Configuring an error response (LJ 95)</td>
</tr>
<tr>
<td></td>
<td>• Can be selected from the list.</td>
<td></td>
</tr>
<tr>
<td>Acknowledgement response</td>
<td>Information (yes/no) stating whether the error tripped is to be acknowledged.</td>
<td>The acknowledgement response cannot be changed for all error response types. Configuring an error response (LJ 95) Acknowledging errors (LJ 95)</td>
</tr>
</tbody>
</table>
9.9.2 Configuring an error response

The error response types of the error handler define the state of the module's intrinsic application. If an error occurs, the standard behaviour triggers the configured error response in the other machine modules.

If a different error response of the machine module is desired, the possibility of programming the desired error response in the corresponding machine module is provided.

- Further information on the DefaultCoupling (predefined standard response) can be found in the following section: Default coupling
- The error states that can be defined in the module applications (MAP) are divided into the following error categories:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
<th>Further information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error text</td>
<td>Text description of the error.</td>
<td>The (English) error text is stored in the table of attributes. The error text always is to be used if no higher-level mechanism for implementation of the error number in national languages is activated (example: higher-level visualization).</td>
</tr>
<tr>
<td>Detailed error information</td>
<td>For each error, an additional attribute (type: DWORD) can be used, which (in connection with the error number) can be interpreted.</td>
<td>For a “Drive Error” group error, for instance, the detailed error information of the corresponding controller can be used.</td>
</tr>
<tr>
<td>Priority</td>
<td>Assign priority for errors of the same error response type.</td>
<td></td>
</tr>
</tbody>
</table>

More information:
- L_EATP_ErrorSet
- L_EATP_ModuleErrorHandler

9.9.3 Acknowledging errors

The acknowledgement of errors deletes all error messages / error causes that have been defined as requiring acknowledgement from the current error list (Error List).

The acknowledgement of errors can be activated by the following procedures:

1. Permanent activation: Within the module application via the following program command:
2. Activation for one cycle: Via the `baseChannelSetErrorQuit` method, rising edge `FALSE` to `TRUE`. The method sets the `xQuitErrors` input to `TRUE` for one cycle.

If multiple errors occur, they cannot be acknowledged individually. Further information:

- **L_EATP_ModuleErrorHandler** (§ 130)

### 9.9.4 Acknowledging errors: Response in the machine module

The `xErrorQuitActive` output on the **L_EATP_ModuleErrorHandler** block can be used within a module application to acknowledge errors on an integrated function block.

#### Defining errors - standard procedure

- The `SetErrors` method is to be extended by one function block.
- The **MAP_EmptyModul_App1** FB is to be extended by the corresponding declaration.
- The `xSetError[1..4]` inputs must be connected to the module application.
- Assign the corresponding parameter to the respective error input:
  - Error number,
  - Error text,
  - Error priority,
  - Acknowledgement response
- More information:
  - **L_EATP_ErrorSet** (§ 125)
9.9.5 Triggering errors

Example: Triggering / forwarding errors - within a machine module

Program line for triggering an error within the machine application:

```
ErrorsA.xSetErr1 := TRUE;
```

Example: Trigger error Error1 of the Errors A FB

Within the respective machine module, the error handling transfers the error response with the highest priority of the current errors to the module's intrinsic state machine.

- By this, the state machine changes to the defined status of the error response type with the highest priority.
- Then the module application processes the state change (just like every further state change).
9.9.6 Central Error handling in the ApplicationTemplate: Error List

The error handling of the ApplicationTemplate transmits all errors which occur in the machine modules to the global error list, Error List.

- How to start the global error list:
  1. Call the A20_Visualisation folder, L_Main.
  2. Click the Error List button to display the global error list:

- The error list shows the errors defined as requiring acknowledgement, which are still pending/to be acknowledged.
- The error list shows maximally 25 error entries. In case of more than 25 entries, the warning reports (1007) that too many entries are in the error list: "Errorlist is full, probably missing entries"
- Remedy: Reduce the number of entries in the error list by acknowledging the errors.
### Information in the error list

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM addr.</td>
<td>Absolute address of the machine module. Example: Module2 has the absolute address 1.2. The means: The machine module is in the MMT on the second level with the relative address 2.</td>
</tr>
<tr>
<td>MM Inst.</td>
<td>Indicates the instance name of the machine module.</td>
</tr>
<tr>
<td>Err. no.</td>
<td>Error number which is defined at the wErrCode input of the L_EATP_ErrorSet block.</td>
</tr>
<tr>
<td>Type</td>
<td>Error response which is defined at the eReaction input of the L_EATP_ErrorSet block.</td>
</tr>
<tr>
<td>Error text</td>
<td>Error text which is defined at the sText input of the L_EATP_ErrorSet block.</td>
</tr>
<tr>
<td>Err. Det.</td>
<td>Detail number of the error which is defined at the dwErrDetail input of the L_EATP_ErrorSet block.</td>
</tr>
</tbody>
</table>

#### 9.9.7 Overview system error messages

The ApplicationTemplate contains the following system error messages. In the event of an error, the corresponding remedy has to be carried out.

<table>
<thead>
<tr>
<th>Number</th>
<th>Error text</th>
<th>Meaning/cause</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Duplicated MM Address on the same level</td>
<td>Module address (MM Address) has been used multiple times within the same level of the machine module tree. On level 2,3,4 or 5, the same module address has been used multiple times.</td>
<td>Assign unambiguous module addresses/ change multiple-assigned module addresses. Each module address must be unambiguous within one level.</td>
</tr>
<tr>
<td>1001</td>
<td>Missing MM address</td>
<td>Missing module address of a machine module. No module address has been assigned to a machine module in the MMT.</td>
<td>A module address has to be assigned to the machine module in the MMT.</td>
</tr>
<tr>
<td>1002</td>
<td>Too many machine modules are used</td>
<td>Too many machine modules are used. Maximally 30 machine modules can be used in the MMT.</td>
<td>Reduce the number of machine modules. Use maximally 30 modules in the MMT, optionally contact the Lenze support. ➤ Machine Module Tree - MMT (19)</td>
</tr>
<tr>
<td>1003</td>
<td>MMT has more than 5 levels</td>
<td>The machine modules are arranged on more than five levels in the MMT. The MMT maximally supports hierarchy levels in the MMT.</td>
<td>Reduce the number of levels in the MMT. Use maximally five levels in the MMT, optionally contact the Lenze support. ➤ Machine Module Tree - MMT (19)</td>
</tr>
<tr>
<td>1004</td>
<td>Duplicated TopLevel module</td>
<td>More than one Toplevel machine module/machine control module is available (example: &quot;Machine Control&quot; module). Two machine modules do not have a master: MasterBus connection of a slave module is not connected to the SlaveBus terminal of the master module.</td>
<td>Create the missing connection in the MMT: Connect the MasterBus terminal of the slave module to the SlaveBus terminal of the master module. Optionally, delete the redundant machine module.</td>
</tr>
<tr>
<td>1005</td>
<td>AccessACD does not find MM addr.</td>
<td>AccessACD method uses the wrong slave address. The AccessACD method in the MAP has received a non-existing module address of a slave module.</td>
<td>Check the syntax of the AccessACD method in the MAP.</td>
</tr>
</tbody>
</table>
9.9.8 Export error overview of all machine modules: CSV file

The ApplicationTemplate provides the possibility to create a list of all errors/error definitions in one *.CSV file.

- The CSV-based text file contains the error definitions of all machine modules integrated in the machine module tree (MMT).
• The CSV file (file name: AT_DefinedErrors.CSV) after creation from the »PLC Designer« can be found in the main directory of the controller (volatile in the flash memory of the controller). Thus the file can be used during the project planning phase of a machine application (example: for an external visualization system).

How to write the CSV file:

On the visualization interface of L_Main, click the Create CSV File button.

Example: Structure of the exported CSV file

| Machine,10001,4,1,"Example for reaction type fault",1323394 |
| Machine,10002,3,1,"Example for reaction type Quickstop",0 |
| Machine,10003,2,1,"Missing slave(s)",0 |
| Module1.1,10001,4,1,"EVENT: Drive Error",0 |
| Module1.1,10002,4,1,"EVENT: Drive error of drive 2",0 |
| Module1.1,10003,2,0,"EVENT: Drive warning",0 |
| Module1.1,10004,1,0,"EVENT: Drive reaction type Information",0 |

[9-26] Example: Structure of the exported CSV file

For experts:

• Alternatively, the file export via xExecuteCreateCSVFile of the Error List Handler L_EATP_GVL.ErrorListHandler can be called from the application program (FALSE->TRUE edge).
• The following outputs provide information on the progress: xBusyCreateCSVFile, xDoneCreateCSVFile, and sInfoCreateCSVFile.

9.10 Logbook

The ApplicationTemplate transfers the error events to the logbook function of the controller.

• The logbook entries can be viewed in chronological order in the logbook of the controller.

How to display the logbook of the controller:

1. Double-click the controller in the device view.
2. Call the Log tab to show the contents of the logbook:

• Example: Logbook view (two warnings/errors). Click the button to update the view. The Description column contains information about the cause of the respective message.

The logbook entries (Description) column have the following structure:

{(absolute) machine module address}, <module name>, <error response with error ID>, <error text> <(optional) detailed error information>

Example: M1.2, Module2, F12002, Demo: Drive Error of drive 2
<table>
<thead>
<tr>
<th>Element</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1.2</td>
<td>Absolute ID of the machine module</td>
</tr>
<tr>
<td>Module2</td>
<td>Name of the machine module</td>
</tr>
<tr>
<td>F12002</td>
<td>Error ID, response type: ☐ error</td>
</tr>
<tr>
<td>Demo: Drive Error of drive 2</td>
<td>Error text</td>
</tr>
</tbody>
</table>
Module diagnostics

For the module diagnostics, a visualization is provided.

- The buttons in section **Machine Modules** call the details of the machine modules.
- Click the desired machine module to show the respective status and further details.
- The visualization for module diagnostics is provided as a separate visualization `L_EATP_VisModuleList` (for instance for own visualization processes).

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM addr.</td>
<td>Absolute address of the machine module. Example: Module2 has the absolute address 1.2. The means: The machine module is in the MMT on the second level with the relative address 2.</td>
</tr>
<tr>
<td>MM name</td>
<td>Instance name of the machine module</td>
</tr>
</tbody>
</table>
| ST       | Indicates the status of the machine module.  
- Green = everything is OK  
- Red = error (system fault, fault or quick stop) |
| CPL      | Coupling type of the machine module  
- DEF = Default Coupling active  
State machine of the machine module is coupled to the state machine of the Machine Control module.  
- CMD = Coupling Master Disabled  
All slave modules are decoupled from the Machine Control module. This serves to decouple the lower-level modules from the state machine of the Machine Control module.  
- CSD = Coupling Slave Disabled  
Slave module is decoupled from the master. Thus, the module is decoupled from the state machine of the master module. |
9.12 Multitasking

- The ApplicationTemplate is able to multitask.
- The following tasks are defined in the ApplicationTemplate:

<table>
<thead>
<tr>
<th>Task level</th>
<th>Priority</th>
<th>Type</th>
<th>Cycle time (bold = default value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task_High</td>
<td>High</td>
<td>Cyclic</td>
<td>Short (1, 2, 4 ms)</td>
</tr>
<tr>
<td>Task_Mid</td>
<td>Medium</td>
<td>Cyclic</td>
<td>Medium (4, 6, 8, 16 ms)</td>
</tr>
<tr>
<td>Task_Free</td>
<td>Low</td>
<td>Unsolicited</td>
<td>Unsolicited</td>
</tr>
</tbody>
</table>

- In the `A11_ModuleAppCalls` folder, the respective module applications can be assigned to the corresponding task.
- ModuleAppCalls (MAC) are the calls of a module application (MAP) by the associated machine modules (MM) of the corresponding task.

According to the task configurations, the associated programs (CallFree, CallHigh, and CallMid) are to be called, which, in turn, call the ModulAppCall programs (MAC_Task_Free, MAC_Task_High, and MAC_Task_Mid).

- The connection to the interface system (like for example the I/O system and visualization) is to be carried out in the corresponding ModulAppCall program (MAC).
- The module applications which are assigned to the corresponding tasks are stored...
  - ...in the `A70_MachineModuleSources` folder or
  - ...in the corresponding module libraries.
In order to be able to use the multitasking functionality in a machine module, the machine module must be provided with more than one module application.

### 9.13 Consistent data transfer

A defined data area is exchanged between two tasks so that it is transferred consistently to the other task.

Data consistency...

- ...is ensured depending on the data, or
- ...to be ensured for specific application cases. Depending on the application case, proceed as follows:

<table>
<thead>
<tr>
<th>Application case</th>
<th>Data element/data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data consistency within individual data elements</td>
<td>Integer</td>
<td>INT DINT</td>
</tr>
<tr>
<td></td>
<td>Bit fields</td>
<td>WORD DWORD</td>
</tr>
<tr>
<td>Data consistency within one data element</td>
<td>Floating point</td>
<td>LREAL</td>
</tr>
<tr>
<td>Data consistency for more than one data element</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Reserving/inhibiting data areas of the AppChannelData(ACD) structure

The ACD structure contains a `L_EATP_CriticalSection` block which ensures the data consistency of real-time critical data with the following methods. If several blocks are required, further instances of the block can be used in the ACD structure, instance name of the block **CS** (Critical Section).

- `L_EATP_CriticalSection (LJ 123)`

<table>
<thead>
<tr>
<th>Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock()</td>
<td>Reserve data area of the ACD structure.</td>
</tr>
<tr>
<td>Unlock()</td>
<td>Enable reserved data of the ACD structure.</td>
</tr>
<tr>
<td>LockState()</td>
<td>Query locked status of a data area.</td>
</tr>
</tbody>
</table>
Tip!

Call the **L_EATP_CriticalSection** block via the **CS** (Critical Section) instance name in the ACD structure.

Note!

**Avoiding performance loss**

Avoid inhibiting real time-critical data of the ACD structure by several MAPs within the same machine module by the **CriticalSection**.

- Reserve an individual area for every MAP for real time-critical data in the ACD structure, in order to avoid that the module application (MAP) of a low-priority task (**Task_Mid**) slows down the MAP of a higher-priority task (**Task_High**).

1. Before the data area to be transferred can be consistently accessed, the **Lock()** method has to be called in a task.
   - The **Lock()** method returns a **BOOL** value if...
     - another task has already called the method ("TRUE").
     - no task has called the method yet ("FALSE").

Note!

A task may only use the data area (reading or writing) if the **appChannelLock()** method has returned the **TRUE** value.

```
IF Lock() THEN
    For i:=0 TO 2000 by 1 DO
        rACDOwn.aCAMData[i] := aCAMData[i];
    END_FOR
IF Lock() THEN
    For i:=0 TO 2000 by 1 DO
        rACDOwn.aCAMData[i] := aCAMData[i];
    END_FOR
Unlock()
```

[9-1] Example: The **Lock()** method inhibits real time-critical data of the ACD structure.

2. After the last exclusive access to the data area, a task must call the **Unlock()** method to enable the data area.

```
IF Lock() THEN
    For i:=0 TO 2000 by 1 DO
        rACDOwn.aCAMData[i] := aCAMData[i];
    END_FOR
Unlock()
```

[9-2] Example: The **Unlock()** method enables a reserved data area der ACD structure.

Program in machine module 1 (**TaskMidPriority = 4 ms**):

```
IF Lock() THEN
    For i:=0 TO 2000 by 1 DO
        rACDOwn.aCAMData[i] := aCAMData[i];
    END_FOR
Unlock()
```

Program in machine module 2 (**TaskHighPriority = 2 ms**):
9.14 Internal Control

The "Internal Control" function makes it possible to decouple a slave module from its master. This is advisable during e.g. commissioning.

How to use the "Internal Control"

1. Call the A20_Visualisation folder, L_Main.
2. Activate the desired machine module for which the "Internal Control" is to be activated, example: MM_Module1.
3. Click the Internal Control button for activation.
4. Enter the password by clicking into the text field that is now visible.
   • Standard value: lenze
   • Confirm password entry with Enter.
5. The machine module is now decoupled from the state machine and can be operated manually via the buttons:

IF NOT LockState() THEN
   For i:=0 TO 2000 by 1 DO
      aCAMData[i] := rACDModule1.aCAMData[i];
   END_FOR
END_IF
9.14.1 Exiting the Internal Control

- In order to exit the activated "Internal Control", click the Internal Control button again.
- If "Internal Control" is active, the corresponding machine module can be operated manually by means of the buttons.
  - This has the effect that the machine module is decoupled and the state machine can be influenced.

9.14.2 What does the Internal Control do?

When the "Internal Control" of a machine module is active, the write operations of the module to the BaseChannel and the ACD structure by the higher-level master (i.e. from the outside) are deactivated.

- Thus, all actions involving access via the baseChannelSet() method and the ACD structure to the machine module from the outside have no effect.
- Access within the machine module is possible. Reading accesses (by the baseChannelGet() methods and the ACD structure remain unaffected by the "Internal Control" and still work from the outside.
- In the »PLC Designer« variable view, in the online mode (via the local baseChannel references), the baseChannel can be accessed.

- The following is highlighted:
  - The switches (TRUE/FALSE) for activating the "Internal Control" (xInternalControl)
  - The references to the module's intrinsic baseChannel (blue marking)

If "Internal Control" is active, ...

- ...a write operation by the master has no effect on the module. The write operation of the module's intrinsic application (MAP) will prevail instead.
- ...a read operation by the master or the module's intrinsic MAP supplies the current machine module values.

If "Internal Control" is deactivated, ...

- ...a write operation by the master to the module can be executed.
• ...a write operation of the module’s intrinsic MAP will prevail.
• ...a read operation by the master or the module’s intrinsic MAP supplies the current machine module values.
10 Visualising in the ApplicationTemplate

The ApplicationTemplate contains the predefined L_Main visualization in the A20_Visualisation folder. In order to create further visualization from the visualization elements of the ApplicationTemplate, the steps described in the following section have to be executed.

10.1 Extending the visualization

This section provides information about how to extend the L_Main visualization included in the ApplicationTemplateCounter sample project by further visualization pages. The procedure in the ApplicationTemplate is identical. The lower-level machine modules must be newly created.

- The precondition for extending the visualization in this example (ApplicationTemplateCounter) is that first another machine module has to be integrated into the MachineModuleTree (MMT). Integrating a machine module into the MMT (55)

- The other module application has to be integrated in the program part in the A11_ModulAppCalls folder.

- Call MAC_Task_High (example: MM_Dcl.Modul3.App1)
Visualising in the ApplicationTemplate
Extending the visualization

How to proceed:
1. Call the visualization:
   • Double-click the 20_Visualisation folder.
   • Double-click L_Main.
2. Open the visualization element list:
   • Call <Alt>+<F6> or the menu command Visualisation → Element list.
3. Go to the Element list tab and select the #0 Frame visualization.
   • Execute the Frame selection command by right-clicking the visualization frame.
   • Select Element list: Command Visualisation → Element list (or <Alt>+<F6>).
4. Call the Frame selection by right-clicking.
   • In the following dialog window, the frame visualizations are listed in the order in which the buttons/control elements are arranged.
   • Highlight the visualization of the new module (example: MVis_Module3).
   • Click the > button to select the visualization.
   • Confirm the selection by clicking OK.
5. Go to the Element list tab and select the #0 Frame visualization.
   • Go to the Properties dialog window and select the Referenced visualisations area.
   Assign the required data to the corresponding visualization:
   • Click the visualization name.
   • Click the desired field in the area on the right of m_Input_FB.
Visualising in the ApplicationTemplate

Defining the properties of buttons

10.2 Defining the properties of buttons

**How to proceed:**

1. Extend **Vis_eFrame** in the **A71_LocalSources\Structs ENUM** folder:

   ![Diagram](image1)

   Add the name of the visualization manually, example: **MVis_Module3**

   **Note:** The arrangement of entries in the ENUM **Vis_eFrame** has to correspond to the selected order of #0 Frame of the **L_Main** visualization.

2. Go to **A20_Visualisation\SubVisu** folder and select the **Keys_Main** visualization.

3. Create a new button.

   • Highlight the **Module 2** button:
Visualising in the ApplicationTemplate

Adding a visualization: Standard procedure

4. Go to the Properties dialog of the Module 3 button and select the Color variables\Toggle color variable.

5. Adapt the value of the button copied before to the new name (example: Module 3)

6. Go to OnMouseUp and change the ST code to

   MVis.scVisuIntern.diFrame:=Vis_eFrame.MVis_Module3

10.3 Adding a visualization: Standard procedure

   The visualization must be added in the A20 Visualisation folder:

   Right-click the A20 Visualisation folder and execute the following command:

   Add object → Visualization
Visualising in the ApplicationTemplate
Adding a visualization: Standard procedure

• Allocate the desired name of the visualization. Click Open.
• The visualization has been added in the A20 Visualisation folder.

• Create a Frame in the visualization: Create a frame from the Tools dialog box via drag&drop.

• Adapt the frame size.
• Execute the Frame selection command by right-clicking into the frame.
  • Accept the desired visualization to the selection under Available visualisations by clicking the > button.
  • Click OK to accept the visualization to the frame.
  • Example: In order to accept the global error list, select the L_EATP.L_EATP_VisErrorList entry.

• Adapt/position the size of the visualization.
• For the error list (Error List), no further assignments are required.
  Result: The visualization is now serviceable.
Visualising in the ApplicationTemplate

Adding a visualization: Standard procedure

- **Example:** In order to insert a window for activating errors for a specific machine module, select the _L_EATP.L_EATP_VisErrorSet_ visualization.

- **Example:** Assigning the visualization for the ErrorsA block of the App1 module application of Module1.

- **Tip:** The predefined visualization elements of the ApplicationTemplate require no manual data assignments.
  - Therefore (after the compilation process), logging in (going online) is possible without any extra effort involved.
The L_EATP_ApplicationTemplate library

11 The L_EATP_ApplicationTemplate library

This library contains the basic functionality for the ApplicationTemplate.

Overview of the functions and function blocks

The functions and function libraries of the L_EATP_ApplicationTemplate library are divided into different groups.

1.POUs (Program Organisation Units)
   - L_EATP_Application
   - L_EATP_CriticalSection
   - L_EATP_ErrorListHandler
   - L_EATP_ErrorSet
   - L_EATP_Module
   - L_EATP_ModuleDiag
   - L_EATP_ModuleErrorHandler
   - L_EATP_ModulRelations
   - L_EATP_SMAccess

2.Structs_Types
   - L_EATP_ACD_Base (ACD structure)
   - L_EATP_BaseChannel
   - L_EATP_ErrorList
   - L_EATP_MMD_Base (MMD structure)
   - L_EATP_MM_Address

3.Enums
   - L_EATP_ErrorReactionType
   - L_EATP_SMStates

GetBooleanProperty (automatically generated by "PLC Designer")
GetCompany (automatically generated by "PLC Designer")
GetNumberProperty (automatically generated by "PLC Designer")
GetTextProperty (automatically generated by "PLC Designer")
GetTitle (automatically generated by "PLC Designer")
GetVersion (automatically generated by "PLC Designer")
GetVersionProperty (automatically generated by "PLC Designer")

L_POUs (Program Organisation Units)

L_EATP_Application
L_EATP_CriticalSection
L_EATP_ErrorListHandler
L_EATP_ErrorSet
L_EATP_Module
L_EATP_ModuleDiag
L_EATP_ModuleErrorHandler
L_EATP_ModulRelations
L_EATP_SMAccess

L_Structs_Types

L_EATP_ACD_Base (ACD structure)
L_EATP_BaseChannel
L_EATP_ErrorList
L_EATP_MMD_Base (MMD structure)
L_EATP_MM_Address

LEnums

L_EATP_ErrorReactionType
L_EATP_SMStates
11.1  
**Automatically generated functions**

This section contains detailed information on functions that are generated automatically by the »PLC Designer«. The output values can be found under: Project → Project information

**Note!**

If the project information is changed, the following setting is required:

Call the project information: Project → Project information.
- Deactivate the "Automatically generate POU for property access" parameter.
- Click OK to close the project information.

Call the project information again: Project → Project information.
- Activate the "Automatically generate POU for property access" parameter.
- Click OK to close the project information.

The changed project information will be available in the project afterwards.

11.1.1  
**GetBooleanProperty (automatically generated by »PLC Designer«)**

The function provides boolean project information of the library. (Currently no boolean project information is used).

**Note!**

This function must always be used with the corresponding name area of the related library. (for instance `IOStandard.GetNumberProperty(stKey)`)

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>stKey</td>
<td>WSTRING Key of the project information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBooleanProperty</td>
<td>BOOL  Value of the project information</td>
</tr>
</tbody>
</table>
11.1.2 GetCompany (automatically generated by »PLC Designer«)

The function provides the name of the company that has created the library.

Note!

This function must always be used with the corresponding name area of the related library. (for instance IOStandard.GetCompany())

```
GetCompany
```

Output:

<table>
<thead>
<tr>
<th>Identifiers/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCompany</td>
<td>WSTRING Name of the company that created the library.</td>
</tr>
</tbody>
</table>

11.1.3 GetNumberProperty (automatically generated by »PLC Designer«)

The function provides numerical project information of the library. (Currently, no numerical project information is used).

Note!

This function must always be used with the corresponding name area of the related library. (for instance IOStandard.GetNumberProperty(stKey))

```
GetNumberProperty
```

Inputs:

<table>
<thead>
<tr>
<th>Identifiers/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>stKey</td>
<td>WSTRING Key of the project information</td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>Identifiers/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetNumberProperty</td>
<td>DINT Value of the project information</td>
</tr>
</tbody>
</table>
The L_EATP_ApplicationTemplate library
Automatically generated functions

11.1.4 **GetTextProperty (automatically generated by »PLC Designer«)**

The function provides textual project information of the library.

**Note!**

This function must always be used with the corresponding name area of the related library. (for instance `IOStandard.GetText("Author")`)

Inputs

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>stKey</td>
<td>WSTRING</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetTextProperty</td>
<td>Value of the project information</td>
</tr>
</tbody>
</table>

11.1.5 **GetTitle (automatically generated by »PLC Designer«)**

The function provides the title of the library.

**Note!**

This function must always be used with the corresponding name area of the related library. (for instance `IOStandard.GetTitle()`)
11.1.6 GetVersion (automatically generated by »PLC Designer«)

The function provides the version of the library.

**Note!**

This function must always be used with the corresponding name area of the related library. (for instance `IOStandard.GetVersion()`)  

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetVersion</td>
<td>VERSION</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetVersion</td>
<td>Version of the library</td>
</tr>
</tbody>
</table>

11.1.7 GetVersionProperty (automatically generated by »PLC Designer«)

The function provides the version of the library.

**Note!**

This function must always be used with the corresponding name area of the related library. (for instance `IOStandard.GetVersionProperty(stKey)`)  

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>stKey</td>
<td>WSTRING</td>
</tr>
</tbody>
</table>

**Inputs**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>stKey</td>
<td>Key of the project information</td>
</tr>
<tr>
<td></td>
<td>• Extended versioning information of the library</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetVersionProperty</td>
<td>VERSION</td>
</tr>
<tr>
<td></td>
<td>Version of the library</td>
</tr>
</tbody>
</table>
The L_EATP_ApplicationTemplate library

1_POUs - Program Organization Units

11.2 1_POUs - Program Organization Units

11.2.1 L_EATP_Application

This block enables the empty (abstract) basic function block of a machine module application (MAP).

- Each machine module application is derived from this FB.

Local variables

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentModule</td>
<td>Reference to the parent module containing the MAP.</td>
</tr>
<tr>
<td>L_EATP.IModule</td>
<td>• This reference has to be initialised within the MM during the creation of the MAP.</td>
</tr>
</tbody>
</table>

AccessACD

This method allocates an access structure to the ACD structure in the user's own MFB or a slave module.

Input

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_EATP.MM_Address</td>
<td>MM_Address AccessACD • Returns the pointer to the ACD structure to be accessed.</td>
</tr>
<tr>
<td>DWORD dwSize</td>
<td>Size of the structure transferred (e.g.: SIZEOF(MACD_scExample) )</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessACD</td>
<td>• Returns the pointer to the ACD structure to be accessed.</td>
</tr>
<tr>
<td>POINTER TO L_EATP.ACD.Base</td>
<td>(ACD structure)</td>
</tr>
</tbody>
</table>
Base
...is a management function for infrastructure processing.

- This method is to be called first in the body of the FB during every cycle.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Data type</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td></td>
<td>Gives a response as to whether the infrastructure has been established correctly. <strong>Note:</strong> All methods and data regarding the infrastructure must only be used if the return value is “TRUE” first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE</td>
</tr>
</tbody>
</table>

checkSlave
...checks whether a slave with the module address specified is available.

<table>
<thead>
<tr>
<th>Transfer variables</th>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM_Address</td>
<td>L_EATP_MM_Address</td>
<td>ID of the machine module which the method is to request.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th>Data type</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td></td>
<td>TRUE</td>
</tr>
</tbody>
</table>

SMDispatcherCaller
Method for calling the SMDispatcher method and transferring the current status of the state machine.
### 11.2.2 L_EATP_CriticalSection

This FB contains the **lock()**, **Unlock()**, and **LockState()** methods.

The methods render it possible to ensure consistent data transmission. Consistent data transmission for instance is required for multitasking systems. [Consistent data transfer](#105)

**Lock()**

...reserves a data area of the ACD structure (query the state of the flag).

<table>
<thead>
<tr>
<th>Return value</th>
<th>Data type</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td></td>
<td>Replies whether a data area has been reserved (flag set).</td>
</tr>
<tr>
<td></td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
<td>Data area reserved (flag set).</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>Data area not reserved (flag not set).</td>
</tr>
</tbody>
</table>

**Unlock()**

...releases the previously reserved data area again (reset flag).

<table>
<thead>
<tr>
<th>Return value</th>
<th>Data type</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td></td>
<td>Replies whether the data area has been released (flag reset).</td>
</tr>
<tr>
<td></td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
<td>Data area released (flag reset).</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>Data area not released (flag not reset).</td>
</tr>
</tbody>
</table>

**LockState()**

...queries the inhibit state of a data area (flag set/not set).

<table>
<thead>
<tr>
<th>Return value</th>
<th>Data type</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td></td>
<td>Replies whether flag is set.</td>
</tr>
<tr>
<td></td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
<td>Return value if flag is set.</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>Return value if flag is not set.</td>
</tr>
</tbody>
</table>
11.2.3  **L_EATP_ErrorListHandler**

This FB manages/updates the global error list.

- This FB enables the creation of a CSV file containing all errors defined.

```
L_EATP_ErrorListHandler

BOOL xExecuteCreateCSVFile

INT iDiagModuleCount

BOOL xBusyCreateCSVFile

BOOL xDoneCreateCSVFile

STRING sInfoCreateCSVFile
```

**Inputs**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>xExecuteCreateCSVFile</td>
<td>BOOL A rising edge at this input (FALSE -&gt; TRUE) starts generation of a CSV file. The CSV file contains all errors which are defined in the machine modules.</td>
</tr>
<tr>
<td>iDiagModuleCount</td>
<td>INT Provides the link to all subordinated machine modules (slaves). For slaves of the lowest hierarchy level, this output is not assigned.</td>
</tr>
<tr>
<td>xBusyCreateCSVFile</td>
<td>BOOL TRUE: Create CSV file (creation process running).</td>
</tr>
<tr>
<td>xDoneCreateCSVFile</td>
<td>BOOL FALSE -&gt; TRUE: CSV file created (creation process completed). Automatic reset of the signal: Three seconds after the CSV file creation process has been completed.</td>
</tr>
<tr>
<td>sInfoCreateCSVFile</td>
<td>STRING Detailed information about the CSV file creation process.</td>
</tr>
</tbody>
</table>

**Outputs**
11.2.4 **L_EATP_ErrorSet**

This FB can trigger up to four errors. The triggering of an error has an impact on the state machine.

- Depending on how you program the error response (eReaction), the state machine changes to the status defined.

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>xSetError[1...4]</td>
<td>BOOL Setting an error [1...4]</td>
</tr>
<tr>
<td></td>
<td>• Error source: module application (MAP)</td>
</tr>
<tr>
<td></td>
<td>TRUE Error [1...4] has been set.</td>
</tr>
<tr>
<td></td>
<td>FALSE Error [1...4] has not been set.</td>
</tr>
<tr>
<td>wErrCode[error number]</td>
<td>WORD Unique error number of error [1...4]</td>
</tr>
<tr>
<td>eReaction[error response]</td>
<td>L_EATP_ErrorReactionType Error response type of error [1...4]</td>
</tr>
<tr>
<td>dwPriority[priority]</td>
<td>DWORD Assignment of a priority [1...n] for error responses.</td>
</tr>
<tr>
<td></td>
<td>• Helps prioritise errors of the same error response type.</td>
</tr>
<tr>
<td></td>
<td>• Highest priority = 1.</td>
</tr>
<tr>
<td>xAckNeeded[acknowledgement required]</td>
<td>BOOL Note: Is ignored for the following error response types:</td>
</tr>
<tr>
<td></td>
<td>• &quot;Information&quot;: Acknowledgement not required.</td>
</tr>
<tr>
<td></td>
<td>• &quot;Fault&quot;: Acknowledgement is always required.</td>
</tr>
<tr>
<td></td>
<td>• &quot;SystemFault&quot;: No acknowledgement possible.</td>
</tr>
<tr>
<td></td>
<td>FALSE No acknowledgement of the error [1...4] required.</td>
</tr>
<tr>
<td></td>
<td>TRUE Acknowledgement of the error [1...4] is required.</td>
</tr>
<tr>
<td>sText[error text]</td>
<td>STRING Text of error [1...4]</td>
</tr>
<tr>
<td>dwErrDetail[error number]</td>
<td>DWORD Detail information on error [1...4]</td>
</tr>
</tbody>
</table>
11.2.4.1 Adapting the error handling system

How to change the error handling system:

1. Double-click the A70_MachineModuleSources folder in the device view.

2. Double-click the SetErrors method.
   - Add a further FB.
   - The declaration has to be carried out in the MAP_EmptyModul_App1 FB.

   • Connect the desired xError(1...4) error input to the application.
   • Assign the properties of the error to the corresponding inputs: error texts, error response type, acknowledgement behaviour, priority.

The visualization of the L_EATP_ErrorSet block facilitates the operation of the error handling in the ApplicationTemplate: Getting started - operating the ApplicationTemplate (\(\equiv\) 36)

11.2.4.2 Acknowledging errors

If the currently pending error requires acknowledgement and the error cause has been eliminated, the error, by default, can be acknowledged via the higher-level master module (xEHQuitError BaseChannel data element).

- Optionally, the error can be acknowledged via the ErrorHandler.xQuitErrors data element (from the module application) for the user's own module/subordinated slave modules.
- The following command line is to be called from the corresponding module application (signal is edge-based):

   ErrorHandler.xQuitErrors := TRUE;

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>xError[1...4]Active</td>
<td>BOOL</td>
</tr>
<tr>
<td></td>
<td>Status of the currently set error.</td>
</tr>
<tr>
<td></td>
<td>TRUE Setting of an error is active.</td>
</tr>
<tr>
<td></td>
<td>FALSE No error has been set.</td>
</tr>
</tbody>
</table>
• Optionally, the error can be acknowledged by means of the baseChannelSetErrorQuit method.

11.2.5 L_EATP_Module

This FB provides the empty basic FB of a machine module.

• All further machine modules are derived from this FB.

Inputs

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>masterBus</td>
<td>Provides a link to the higher level machine module (master). • For the highest machine module, this input is unassigned.</td>
</tr>
<tr>
<td>MM_Address</td>
<td>Relative module ID of the machine module. • Must be unique within a hierarchy level (a master with all directly connected slaves).</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>slaveBus</td>
<td>Provides the link to all subordinated machine modules (slaves). • For slaves of the lowest hierarchy level, this output is not assigned.</td>
</tr>
</tbody>
</table>

11.2.5.1 Base()

...is a management function for infrastructure processing.

• This method is to be called first in the body of the FB during every cycle.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Data type</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOL</td>
<td>Gives a response as to whether the infrastructure has been established correctly.</td>
</tr>
<tr>
<td>Note: All methods and data regarding the infrastructure must only be used if the return value is “TRUE” first.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE Infrastructure established correctly (Success)</td>
</tr>
<tr>
<td>FALSE Infrastructure not correct (Error)</td>
</tr>
</tbody>
</table>
11.2.6 RegisterACD()

This method records the ACD structure in the module FB (MFB).

```
RegisterACD

\[\text{L_EATP_ACD_Base (ACD structure)} \rightarrow \text{scStruct}
\]
\[\text{DWORD} \rightarrow \text{dwSize}\]
```

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>scStruct</td>
<td>Transferred structure is to be derived from the L_EATP_ACD_Base structure. (e.g.: TYPE MACD_scExample EXTENDS L_EATP_ACD_Base)</td>
</tr>
<tr>
<td>dwSize</td>
<td>Size of the structure transferred (e.g.: sizeof(MACD_scExample))</td>
</tr>
</tbody>
</table>

11.2.7 setCompIDAndVersion()

This method sets the module ID (ComplID) and version of the machine module.

**Inputs**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
</table>
| dwCompID             | Module ID
|                      | • The free area of the ComplID is between 16#3C00 – 16#3FF |
| dwVersion            | Version of the machine module |
### 11.2.8 L_EATP_ModuleDiag

This FB makes it possible to diagnose a machine module.

- The FB always exactly focuses on one individual machine module.
- After starting the project, automatically the highest master module is focused.

#### Inputs

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
</table>
| goToMaster | BOOL | In the case of a positive edge, the FB changes to the master of the currently focused module.  
  - If there is no master on the focused module, the “Current module has no master” message is visible at the statusText output. |
| goToSlave | INT | A non-zero input is considered as a module address. According to the module address, the FB changes to the slave of the machine module currently focused.  
  - If the machine module focused does not feature a slave with a corresponding module address, the “Current module has no slave with module ID xx” message is visible at the statusText output. |

#### Outputs

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleName</td>
<td>STRING</td>
</tr>
<tr>
<td>absoluteMM_Address</td>
<td>STRING</td>
</tr>
</tbody>
</table>
| dataConnector | L_EATP_DataConnector | Returns the Data Connector of the module selected  
  - By this, enables access to the BaseChannel. |
| statusText | STRING | Status/error message of the diagnostic FB. The following messages are possible:  
  - No root module was found  
  - Root module was entered  
  - Current module has no master  
  - Master module was entered  
  - Current module has no slave with module ID x  
  - Slave module was entered |
11.2.9 L_EATP_ModuleErrorHandler

This FB contains the central error handling in a machine module.

- An instance of this block called **ErrorHandler** is part of the basic block **L_EATP_Module**.

```
L_EATP_ModuleErrorHandler

BOOL xQuitErrors xErrorQuitActive BOOL
```

**Input**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>xQuitErrors</td>
<td><strong>BOOL</strong> Set the input to TRUE to activate the acknowledgement of errors (edge FALSE⇒TRUE). The module application (MAP) is the error source.</td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>xQuitErrorActive</td>
<td><strong>TRUE</strong> Shows whether the acknowledgement of errors is active. Is active...</td>
</tr>
<tr>
<td></td>
<td>• ... as long as xQuitErrors = TRUE or</td>
</tr>
<tr>
<td></td>
<td>• for one cycle if the acknowledgement of errors is activated by means of the <strong>baseChannelSetErrorQuit</strong> method.</td>
</tr>
<tr>
<td></td>
<td>† <strong>Method overview - ApplicationTemplate</strong> (LJ 165)</td>
</tr>
<tr>
<td></td>
<td><strong>FALSE</strong> The acknowledgement of errors is deactivated, xQuitErrors = FALSE.</td>
</tr>
</tbody>
</table>
11.2.10 L_EATP_ModulRelations

This FB provides an overview of how the machine modules are linked.

- This FB can for instance be used to retrace whether the interconnection of structures and the assignment of the module address have worked.

```
L_EATP_ModulRelations
```

**Inputs**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>BOOL In the case of a positive edge, the module analysis will start.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>numberOfModules</td>
<td>INT Number of machine modules identified</td>
</tr>
<tr>
<td>relationAsString</td>
<td>ARRAY [1...30] OF STRING For every master-slave relation identified, there is an entry with the corresponding master and slave in the array.</td>
</tr>
<tr>
<td></td>
<td>• The entry has the following structure: <code>&lt;master instance name&gt;/{&lt;master module address}&gt;</code> -&gt; <code>&lt;slave instance name&gt;/{&lt;slave module address&gt;}</code></td>
</tr>
<tr>
<td></td>
<td>Example: <code>MachineModule[1] -&gt; InFeed[3]</code></td>
</tr>
<tr>
<td>Success</td>
<td>BOOL Supplies a response stating whether the analysis has been correct.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Analysis executed correctly.</td>
</tr>
<tr>
<td>FALSE</td>
<td>An error has occurred during the analysis</td>
</tr>
</tbody>
</table>
11.2.11 L_EATP_SMAccess

This FB shows the current state of the state machine in a module application (MAP).

- The SMAccess instance of this FB...
  - ...is integrated into the ApplicationTemplate project in each module application.
  - ...is represented graphically in the Statemachine FUP method.

![Diagram]

Outputs

At the outputs, the FB provides the information on the current status of the state machine to the module application.

<table>
<thead>
<tr>
<th>Transfer variables</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>eNomState</td>
<td>Required TARGET status of the state machine</td>
</tr>
<tr>
<td>eActState</td>
<td>Current ACTUAL status of the state machine</td>
</tr>
<tr>
<td>xWarning</td>
<td>Warning is active</td>
</tr>
<tr>
<td>xActStateIsInit</td>
<td>TRUE</td>
</tr>
<tr>
<td>xActStateIsReady</td>
<td>TRUE</td>
</tr>
<tr>
<td>xActStateIsService</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
The L_EATP_ApplicationTemplate library

2_Structs_Types

11.3 2_Structs_Types

11.3.1 L_EATP_ACD_Base (ACD structure)

Die AppChannelData (ACD) structure is the basis for the application data.

- The structure contains the control/status word.
- The structure contains methods for reserving/enabling data areas.

More information on the ACD structure: [The AppChannelData structure(ACD)](75)

**STRUCT view**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Value/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>xActStatesMode(1...2) BOOL</td>
<td>TRUE Current ACTUAL status: “Mode(1...2)”</td>
</tr>
<tr>
<td>xActStatesHoming BOOL</td>
<td>TRUE Current ACTUAL status: “Homing”</td>
</tr>
<tr>
<td>xActStatesQuickStop BOOL</td>
<td>TRUE Current ACTUAL status: “Quick stop”</td>
</tr>
<tr>
<td>xActStatesFault BOOL</td>
<td>TRUE Current ACTUAL status: “Fault”</td>
</tr>
<tr>
<td>xActStatesSystemFault BOOL</td>
<td>TRUE Current ACTUAL status: “Systemfault”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function block</td>
<td>The CS block has to be used for multi-task applications. More information on the block: <a href="123">L_EATP_CriticalSection</a></td>
</tr>
<tr>
<td>CS (Critical Section)</td>
<td><a href="123">L_EATP_CriticalSection</a></td>
</tr>
<tr>
<td>Control word</td>
<td>Start application</td>
</tr>
<tr>
<td>xStartOperation_In BOOL</td>
<td>Start application</td>
</tr>
<tr>
<td>xStopOperation_In BOOL</td>
<td>Stop application</td>
</tr>
<tr>
<td>xPauseOperation_In BOOL</td>
<td>Pause application</td>
</tr>
<tr>
<td>xManualJogNeg_In BOOL</td>
<td>Manual jog in negative direction (CCW rotation)</td>
</tr>
<tr>
<td>xManualJogPos_In BOOL</td>
<td>Manual jog in positive direction (CW rotation)</td>
</tr>
<tr>
<td>Status word</td>
<td>Status: Application active</td>
</tr>
<tr>
<td>xOperationBusy_Out BOOL</td>
<td>Status: Application active</td>
</tr>
<tr>
<td>xOperationDone_Out BOOL</td>
<td>Status: Application stopped</td>
</tr>
<tr>
<td>xOperationPaused_Out BOOL</td>
<td>Status: Application is pausing</td>
</tr>
</tbody>
</table>
11.3.2 **L_EATP_BaseChannel**

This structure implements the communication channel for basic data (BaseChannel) between a master and a slave node, respectively.

- The BaseChannel elements contained the basic functions "State machine" and "Error handling".

**STRUCT view**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Module view</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwCompiD</td>
<td>DWORD</td>
<td>Module ID</td>
</tr>
<tr>
<td>dwVersion</td>
<td>DWORD</td>
<td>Version of the machine module</td>
</tr>
<tr>
<td>eSMActState</td>
<td>L_EATP_SMStates</td>
<td>Current ACTUAL status of the state machine</td>
</tr>
<tr>
<td>xSMWarning</td>
<td>BOOL</td>
<td>The independent &quot;Warning&quot; status is active.</td>
</tr>
<tr>
<td>eEHActReactionType</td>
<td>L_EATP_ErrorReactionType</td>
<td>Error response type</td>
</tr>
<tr>
<td>eSMSetState</td>
<td>L_EATP_SMStates</td>
<td>Required TARGET status of the state machine</td>
</tr>
<tr>
<td>xEHQuitError</td>
<td>BOOL</td>
<td>Error acknowledgement (rising edge)</td>
</tr>
</tbody>
</table>

11.3.3 **L_EATP_ErrorList**

This array type defines the global error list.

- The global variable declaration GVL_L_EATP_GVL defines exactly one instance of this type with the name ErrorList.

```
L_EATP_ErrorList : ARRAY[1.. L_EATP_GVL.ErrorListEntryCount] OF L_EATP_ErrorListEntry;
```

[11-2] **TYPE view**

11.3.4 **L_EATP_ErrorListEntry**

This structure contains the attributes of an error entry in the global error list.

**STRUCT view**

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>sMM_Address</td>
<td>Absolute address of the machine module.</td>
</tr>
<tr>
<td>sModuleName</td>
<td>Version of the machine module.</td>
</tr>
<tr>
<td>wErrCode</td>
<td>Unique error number.</td>
</tr>
<tr>
<td>eReaction</td>
<td>L_EATP_ErrorReactionType</td>
</tr>
</tbody>
</table>
The L_EATP_ApplicationTemplate library
2_Structs_Types

11.3.5 L_EATP_MMD_Base (MMD structure)

The MachineModuleData structure (MMD) of the MMD_scModulename type serves to instance machine module data.

- The ApplicationTemplate contains variables and function blocks for instancing machine module data.
- All module applications (MAPs) of a machine module can access this MMD structure.

![MMD structure in the folder](A65 EmptyModule\MM_EmptyModule\Structs\MMD_scEmptyModule)

<table>
<thead>
<tr>
<th>Identifier/data type</th>
<th>Information/possible settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>sText</td>
<td>STRING Error text (English)</td>
</tr>
<tr>
<td>dwErrDetail</td>
<td>DWORD Detailed error information</td>
</tr>
<tr>
<td>xAckNeeded</td>
<td>BOOL Note: Is ignored for the following error response types:</td>
</tr>
<tr>
<td></td>
<td>• &quot;Information&quot;: Acknowledgement not required</td>
</tr>
<tr>
<td></td>
<td>• &quot;Fault&quot;: Error. Acknowledgement is always required</td>
</tr>
<tr>
<td></td>
<td>• &quot;SystemFault&quot;: System error. No acknowledgement possible</td>
</tr>
<tr>
<td></td>
<td>FALSE Error acknowledgement not required.</td>
</tr>
<tr>
<td></td>
<td>TRUE Error acknowledgement required.</td>
</tr>
</tbody>
</table>

Note!
In contrast to the ACD structure, the higher-level master module cannot access the MMD structure!

The data structure inherits the L_EATP_MMD_Base structure contained in the L_EATP_ApplicationTemplate library.

- The MMD_sc<Modulename> can be individually extended by the data types and function blocks required.
- In the structure, the L_EATP_CriticalSection block which ensures a consistent data exchange is instanced (if the block is used correctly).

![MMD structure in the folder](A65 EmptyModule\MM_EmptyModule\Structs\MMD_scEmptyModule)
Accessing the MMD structure - by means of the module application of the MFB

In each MAP, the "rMMD" is declared which enables the read/write access to the data contents of the MMD structure ("IntelliSense function").

11.3.6 L_EATP_MM_Address
This structure defines the module address (data type: INT).

11.3.7 L_EATP_MVis
The structure contains data for the external visualization with the Lenze Engineering tool »VisiWinNET« (optional).

11.3.8 L_EATP_scBase
The structure contains the module name (data type: STRING).

11.4 3Enums

11.4.1 L_EATP_ErrorReactionType
This enumeration contains the error response types. The numerical constants are sorted according to the valency and meaning in ascending order.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Numerical constant</th>
<th>Response type</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>No response/not used</td>
</tr>
<tr>
<td>Information</td>
<td>1</td>
<td>Information</td>
</tr>
<tr>
<td>Warning</td>
<td>2</td>
<td>Warning</td>
</tr>
<tr>
<td>Quick stop</td>
<td>3</td>
<td>Quick stop (QSP)</td>
</tr>
</tbody>
</table>
11.4.2 **L_EATP_SMStates**

This enumeration lists the defined states of the basic state machine. The numerical constants are arranged according to the valency and meaning (depending on the critical state) in descending order.

**ENUM view**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Numerical constant</th>
<th>Status in the state machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemFault</td>
<td>0</td>
<td>&quot;SystemFault&quot; status</td>
</tr>
<tr>
<td>Fault</td>
<td>1</td>
<td>&quot;Fault&quot; status</td>
</tr>
<tr>
<td>quick stop</td>
<td>2</td>
<td>&quot;Quickstop&quot; status</td>
</tr>
<tr>
<td>Init</td>
<td>3</td>
<td>&quot;Init&quot; status</td>
</tr>
<tr>
<td>Ready</td>
<td>4</td>
<td>&quot;Ready&quot; status</td>
</tr>
<tr>
<td>Service</td>
<td>5</td>
<td>&quot;Service&quot; status</td>
</tr>
<tr>
<td>Manual</td>
<td>6</td>
<td>&quot;Manual&quot; status</td>
</tr>
<tr>
<td>Homing</td>
<td>7</td>
<td>&quot;Homing&quot; status, the drive is referenced. Either the home position is set directly, or a homing process is started.</td>
</tr>
<tr>
<td>Automatic</td>
<td>8</td>
<td>&quot;Automatic&quot; status</td>
</tr>
<tr>
<td>Mode 1</td>
<td>9</td>
<td>&quot;Mode 1&quot; status</td>
</tr>
<tr>
<td>Mode 2</td>
<td>10</td>
<td>&quot;Mode 2&quot; status</td>
</tr>
</tbody>
</table>
12 Structuring an automation system: Example - flying saw

This chapter describes how to convert the example of a machine structure, the "Flying saw", into the structures of the ApplicationTemplate.

<table>
<thead>
<tr>
<th>Step</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Preparation: Dividing the automation system into subfunctions (138)</td>
</tr>
<tr>
<td></td>
<td>• Identifying subfunctions of the machine structure</td>
</tr>
<tr>
<td></td>
<td>• Representing subfunctions of the machine structure as machine modules</td>
</tr>
<tr>
<td></td>
<td>• Representing machine modules in a tree structure</td>
</tr>
<tr>
<td>2nd</td>
<td>Mapping the actual machine structure in the »PLC Designer« (140)</td>
</tr>
<tr>
<td></td>
<td>• Creating and integrating machine modules</td>
</tr>
<tr>
<td></td>
<td>• Integrating machine modules in the machine structure tree</td>
</tr>
<tr>
<td>3rd</td>
<td>Managing module applications of the machine modules (144)</td>
</tr>
<tr>
<td>4th</td>
<td>Inserting an axis into the module application (145)</td>
</tr>
<tr>
<td>5th</td>
<td>Assigning the module application to the tasks (145)</td>
</tr>
<tr>
<td>6th</td>
<td>Setting up communication between master modules and slave modules (147)</td>
</tr>
<tr>
<td>7th</td>
<td>Application example: Extending the visualisation (148)</td>
</tr>
<tr>
<td>8th</td>
<td>State machine (151)</td>
</tr>
<tr>
<td>9th</td>
<td>Manual jog of the axes (154)</td>
</tr>
<tr>
<td>10th</td>
<td>Error handling: Configuring the error handling (155)</td>
</tr>
</tbody>
</table>

12.1 Preparation: Dividing the automation system into subfunctions

The application example describes how to convert the "Flying saw" machine application into a modularised software structure and map it using the ApplicationTemplate.

How to proceed:

1. Analyse / divide the overall machine functionality (available machine structure / concept) into subfunctions. • Identifying subfunctions of the machine structure (139)

2. Convert the determined subfunctions into machine modules:
   one subfunction = one machine module.

3. Represent the machine modules in the form of a tree structure.
   Then the machine structure is mapped in the machine structure tree (MMT) in the ApplicationTemplate.
Structuring an automation system: Example - flying saw
Preparation: Dividing the automation system into subfunctions

12.1.1 Identifying subfunctions of the machine structure

[Image]

A infeed unit: Infeed
B Saw blade / cutting: Blade
C Saw carriage / synchronous travel: Slide
D Outfeed unit: Outfeed


12.1.2 Representing subfunctions of the machine structure as machine modules

<table>
<thead>
<tr>
<th>Machine module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM 1</td>
<td>Machine Control: Machine module</td>
</tr>
<tr>
<td>MM 2</td>
<td>Infeed</td>
</tr>
<tr>
<td>MM 3</td>
<td>Flying saw</td>
</tr>
<tr>
<td>MM 4</td>
<td>Slide</td>
</tr>
<tr>
<td>MM 5</td>
<td>Blade</td>
</tr>
<tr>
<td>MM 6</td>
<td>Outfeed</td>
</tr>
</tbody>
</table>

12.1.3 Representing machine modules in a tree structure

The subfunctions of the machine structure determined previously must be represented in a tree structure. The tree structure can be seen in the »PLC Designer« as machine structure tree afterwards.

[12-3] Hierarchy of the machine modules as tree structure
12.2 Mapping the actual machine structure in the »PLC Designer«

12.2.1 Device structure: Adding master and slave

How to map the machine structure in the »PLC Designer«:

1. Right-click the controller and select the 1 Add device command to add the 2 "EtherCAT Master":

2. Add the EtherCAT slaves below the EtherCAT Master:

Right-click the EtherCAT Master and execute the 1 Add device command. Select the desired devices from the selection list 2. Repeat the Add device command until all slaves connected to the fieldbus are included in the device view.
12.2.2 Setting the cycle time

The cycle time must be set to 2 ms (input value: 2000 μs).

How to proceed:
1. Double-click the EtherCAT_Master to call the corresponding tab.
2. In the Distributed clock section, assign a value of 2000:

The icon in front of the respective device indicates the successful EtherCAT communication.

12.3 Creating and integrating machine modules

The subfunctions determined previously must be mapped in the form of machine modules. The highest master machine module, MM 1: MachineControl, is already included in the ApplicationTemplate. The other subfunctions must be added.

12.3.1 Creating machine modules

The subfunctions of the machine structure determined previously must be mapped to one machine module each. For this purpose, the following machine modules must be created in the device tree:

<table>
<thead>
<tr>
<th>Machine module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM 1</td>
<td>Machine Control: Machine module</td>
</tr>
<tr>
<td>MM 2</td>
<td>Infeed</td>
</tr>
<tr>
<td>MM 3</td>
<td>Flying saw</td>
</tr>
<tr>
<td>MM 4</td>
<td>Slide</td>
</tr>
<tr>
<td>MM 5</td>
<td>Blade</td>
</tr>
<tr>
<td>MM 6</td>
<td>Outfeed</td>
</tr>
</tbody>
</table>

How to proceed:
1. Copy the MM_EmptyModule machine module.
Structuring an automation system: Example - flying saw

Creating and integrating machine modules

1. Right-click the `A65 EmptyModule\MM_EmptyModule` folder → Copy Empty Module.

2. Insert the previously copied machine module (Empty Module) below the `A70_MachineModuleSources` folder:

   Right-click the `A70_MachineModuleSources` → Insert Empty Module

3. Enter the module name.

   Example: **MM 2**, infeed unit: **Infeed**

4. Click Insert to insert the machine module.

   This is the way the five machine modules **MM 2...MM 6** must be created.

The following machine modules must be created:

- **B01_Infeed**
- **B02_Flying Saw**
- **B03_Outfeed**
- **C01_Slide**
- **C02_Blade**

The highest machine control module, **MM_MachineControl**, is already included in the ApplicationTemplate.
12.3.2 Integrating machine modules in the machine structure tree

The machine modules previously created in the device tree must be integrated in the machine structure tree (MMT) to make the individual subfunctions of the machine structure executable in the ApplicationTemplate.

How to proceed:

Integrating a machine module in the MMT

1. Double-click and open in the device tree: A10_MachineModuleTree
2. Double-click MMT (PRG).
3. In the Tools dialog window, click the Block button.
   • Create the new FB via drag-and-drop.
   • Double-click the area of the FB. Click the button.

Use the Input assistance from the Application element...
   • ...to assign the MFB_Infeed FB: Application\A70_MachineModuleSources\MM_B01_Infeed
   • ...to assign the MM_Dcl.B01_Infeed instance: Application\A55_VarLists\MM_Dcl

Note: Go to the Input assistant and select the Insert with namespace prefix when assigning the instance name.

Addressing of the machine modules in the MMT

4. In the Tools dialog window, click the Input button.
   • Add the new input at MM_Address.
   • Assign the relative address (example: 1).

Steps 3 and 4 must be executed for all machine modules M 2...M 6. The modules must be created according to the previously determined tree structure. The representation of the MMT is rotated by 90 degrees to the left. Every module must be assigned the following elements:
   • The corresponding function block,
   • The instance of the module,
   • The relative address.
5. Connecting the machine modules in the MMT
Defining the processing order of the machine modules according to the data flow.
Menu command: CFC→Processing order→Show according to data flow

12.3.3 Managing module applications of the machine modules
Module applications (MAPs) which are not required may be deleted, provided that one MAP per machine module is available.

Note!
If a MAP is deleted, please observe that the corresponding instances in the respective MFB must also be deleted.

Example: Module applications App1...App3 of machine control module MM_MachineControl

How to proceed:
1. Deleting module applications which are not required.
   • Subfolders which are not required must be deleted from the folder of the machine module in the device tree, example: ModApp2, ModApp3.
   • Right-click the subfolder→Delete.
2. Deleting instances which are not required from the declaration part in the MFB.
   • Double-click and open the desired MFB of the machine module.
   • Deleting instances which are not required from the declaration part.
12.3.4 Inserting an axis into the module application

The example shows how to integrate an axis in module application MAP_B01_Infeed_App1.

How to proceed:

1. Double-click and open module application MAP_B01_Infeed_App1.
2. Insert the axis into the declaration part:

   - In the program part, assign the axis to the Axis input:
     
     ```
     Axis: AXIS_REF_SM3;
     ```

   3. For continuous processes, a new action must be created in the MAP_B01_Infeed_App1 module application, example: Continuous:

   ![Image](image.png)

   - Add the Continuous action to the program part of the module application:

     ```
     Continuous[];
     ```

   - Assign the axis to the module in the program part of the Continuous action:

     ```
     Axis(Axis.Axis);
     ```

   In order to integrate all axes, these steps must be executed in the MAPs of MM_B01_Infeed, MM_B02_Outfeed, MM_C01_Slide and MM_C02_Blade.

12.3.5 Assigning the module application to the tasks

How to proceed:

1. Right-click the A11_ModuleAppCalls folder and call the Create Task Call command.
2. In the following dialog box, the module application for calling the task must be selected. Assign the corresponding MAP to the task / reassign with <</>>.

3. Double-click and open:
   - Folder A11_ModuleAppCalls
   - Main program MAC_Task_High of the task with the highest priority.

4. Assign the axis to the module IoConfig_Globals.SM_Drive_ETC_i700:
12.4 Setting up communication between master modules and slave modules

Every machine module has a data container, the ACD structure **MACD_sc<module name>**.
- The ACD structure can be used to exchange data with a higher-level machine module.
- The higher-level master module accesses the ACD structure via POINTER and REFERENCE.

The section below shows how a data exchange between a master module and a slave module works.

How to proceed:

1. Double-click and open:
   - A70_MachineModuleSources\MM_A01_MachineControl\ModApp1
   - Module application MAP_A01_MachineControl_App1

2. In the declaration part, a pointer and a reference to the ACD structure of the slave must be declared which is slave to the **MachineControl** machine control module:

   ```
   proc MAP_A01_MachineControl_App1
   decl
   pA01MM: POINTER TO MACD_scMM_A01_MachineControl;
   rA01MM: REFERENCE TO MACD_scMM_A01_MachineControl;
   pA01In: POINTER TO MACD_scMM_A01_Infeed;
   rA01In: REFERENCE TO MACD_scMM_A01_Infeed;
   pA01FS: POINTER TO MACD_scMM_A01_FlyingSaw;
   rA01FS: REFERENCE TO MACD_scMM_A01_FlyingSaw;
   pA01Out: POINTER TO MACD_scMM_A01_Outfeed;
   rA01Out: REFERENCE TO MACD_scMM_A01_Outfeed;
   ```

   This step must be executed for all direct slave modules of the **MM_A01_MachineControl** master module.

   In this example, pointers and references to the **Infeed**, **FlyingSaw** and **Outfeed** modules must be declared.

3. In all slave modules, access must be provided to the ACD structure.
   - In the program part of the module application of the corresponding slave module, the `AccessACD()` method must be called for accessing the ACD structure.
   - The relative address of the machine module whose ACD structure is to be accessed must be transferred to the **MM_Address** input. The relative addresses of the machine modules can be seen in the machine structure tree:

   Folder A10_MachineModuleTree\MMT (PRG).
Structuring an automation system: Example - flying saw

Application example: Extending the visualisation

- The size of the transferred structure must be assigned to the `dwSize` input.

```cpp
// ------------------- START OF USER SPECIFIC BLOCK -------------------
// Add here the code of your module application.
// Certification of the access to the MACD_scModule1

pACD0m := AccessACD([MM_A01_MachineControl, L_EATP_CONST.ONMID], dwSize := SIZEOF(MACD_scM01_MacMachineControl));
 lạcD0m REP := pACD0m;

pACD01 := AccessACD([MM_B02_FlyingSaw], dwSize := SIZEOF(MACD_scB01_Infeed));
 lạcD01 REP := pACD01;

pACD02 := AccessACD([MM_B02_FlyingSaw], dwSize := SIZEOF(MACD_scB02_FlyingSaw));
 lạcD02 REP := pACD02;

pACD03 := AccessACD([MM_B02_FlyingSaw], dwSize := SIZEOF(MACD_scB03_Outfeed));
 lạcD03 REP := pACD03;
```

This step is required for all slave modules directly below the `MM_A01_MachineControl` master module.
- In this example, the `MM_B02_FlyingSaw` machine module is the higher-level master module of the `MM_C01_Slide` and `MM_C02_Blade` slave modules.
- Steps 1 to 3 must be executed for the `MM_C01_Slide` and `MM_C02_Blade` slave modules in the `MAP_B02_FlyingSaw_App1` module application.

12.5 Application example: Extending the visualisation

The ApplicationTemplate contains a visualization of the machine modules. The following section describes how to adapt the visualization individually. Extending the L_Main main visualisation:

Visualisation of a machine module

![Structure of the visualization in the ApplicationTemplate](image)
Structuring an automation system: Example - flying saw

Application example: Extending the visualisation

In order to insert the visualization of a machine module into the L_Main main visualization and make it visible, it must be extended in two ways.

1. In the L_Main main visualization: On the right, the visualization pages display the following information in switchable area A: module list, error list, visualization of the selected machine module.

   In order to be able to display the visualization of a module, it must be added to the frame selection of the visualization page / switchable area A.

2. In the Keys_Main visualization: In order to call the module visualization, a button must be created.

How to proceed:

1. Call the visualization in the device tree:
   
   Double-click and open: 20_Visualisation → L_Main

2. Open the visualization element list:
   
   • Call <Alt>+<F6> or the menu command Visualisation → Element list.
   
   • Go to the Element list tab and select the #0 Frame visualization: 1

   • Select the Visualization → Frame selection menu command: 2

   • From the available visualisations, select the visualisations of the previously created machine modules that are to be referenced in the visualisation frame:

   The sequence visible here of the selected visualisations then corresponds to the order of the buttons/control elements.
3. Go to the **Properties** dialog box and enter the corresponding variable for every single visualization in the **References** area.

4. Go to the **A71_LocalSources** folder and extend the **Vis_eFrame** enumeration data type by the additional module visualizations.

   **Note:** The sequence of the visualizations in **Vis_eFrame** must correspond to the sequence in the frame selection (cp. step 2)!

5. Add the button for calling the module visualization.
   - Double-click and open: **A11_Visualisation\SubVisu～** **Keys_Main**
   - Add another button below the **Machine Control** button.
   - Name the new button according to the previously assigned module names, example of first button: **Infeed**.

**Configuring the button: Change of colour on mouse-up / mouse-down**

### How to proceed:

1. Select the **Infeed** button by clicking the visualisation page.
2. Go to the **Properties** dialog box and click **Configure** in the **Input configuration** area to configure the behaviour for **OnMouseUp**:
3. Go to the Input configuration dialog box and insert Execute ST code above the arrow buttons.

4. Enter the structured text (ST) in the text field which is to be executed for OnMouseUp. Enter the following value:

   \[ \text{MM\_Vis\_scVisuIntern\_diFrame := Vis\_eFrame\_MVis\_Infeed;} \]

5. Go to the Properties area and enter below the Color Variables of the button the following value for toggle color:

   \[ \text{MM\_Vis\_scVisuIntern\_diFrame = Vis\_eFrame\_MVis\_Infeed;} \]

12.6 State machine

12.6.1 Controlling the state transition in the machine control module

Example: It should be possible to set all lower-level machine modules in the "Ready" or "Manual" state via two buttons in the visualization of the Machine Control machine control module.

How to proceed:

1. Declare the elements for the buttons in the MVis_ * structure.

   Double-click and open:

   - A70_MachineModuleSources\ MM_A01_MachineControl\Structs

   - Structure MVis_scA01_MachineControl

   In the program part, the variables of the BOOL data type for the state change must be declared, example: xReady and xManual.

2. Insert the buttons into the visualization.
Double-click and open:

- 📦 A70_MachineModuleSources\MM_A01_MachineControl\Visu
- Visualization 📚 MVis_A01_MachineControl
- Go to the Tools dialog box and select the Common Controls section. Insert the buttons into the visualization interface:

3. Assign the variables.

Go to the Properties dialog box and select the Input configuration\Tab\Variable. Assign the previously created xReady variable.

- Structure 📚 MVis_scA01_MachineControl
- Location: Folder 📀 A55_VarLists\MM_Vis

4. Go to the Properties dialog box and click Configure in the Input configuration area to configure the behaviour for OnMouseUp:

5. Go to the Properties area under Color Variables and assign the variable from the MVis_scA01_MachineControl structure to the button for toggle color:

6. Supplement the program line in the S02_Ready action in the Machine Control control module:

Steps 3 to 6 must be performed correspondingly for the Manual button.
12.6.2 Controlling state transitions in the slave modules

The previously created buttons serve to switch between the Ready and Manual states.

If the respective state is requested via the Manual button, the lower-level machine modules are supposed to enable the connected axes automatically. The lower-level modules must not change to the Manual state before the axes are enabled.

If the respective state is requested via the Ready button, the lower-level modules are supposed to inhibit the axes automatically. The lower-level modules must not change to the Ready state before the axes are inhibited.

The example below shows how a state transition can be controlled with the MM_B01_Infeed machine module.

How to proceed:

1. Trigger the "Ready" state.
   
   Double-click and open:
   
   - \A70_MachineModuleSources\MM_B01_Infeed\ModApp1
   
   - Subfolder with the state actions: Map_B01_Infeed_App1 (FB)\States.
   
   - S02_READY.

2. Supplement the program lines in the S02_READY action:

   ```
   SNDisableReadyToManual (xValue:= NOT Axis1.xAxisEnabled);
   IF baseChannelGetNominalState (MM_Address:=0) = L_EATF_SMStates.Manual THEN
   Axis1.xRegulatorOn:=TRUE;
   Axis1.xDriveStart:=TRUE;
   END_IF
   ```

3. Trigger the "Manual" state.

   - Double-click and open S03_MANUAL
   
   - Supplement the program lines in the S03_MANUAL action:

   ```
   IF baseChannelGetNominalState (MM_Address:=0) = L_EATF_SMStates.Ready THEN
   Axis1.xRegulatorOn:=FALSE;
   Axis1.xDriveStart:=FALSE;
   END_IF
   ```
12.7 Manual jog of the axes

The visualization of the **MM_Infeed** machine module is supposed to enable the manual jog of the axis. The example below shows how to insert two buttons for the positive and negative manual jog into the visualization of the **Infeed** machine module.

How to proceed:

1. Declare the elements for the `MVis_xxx` structure.
   
   Double-click and open:
   
   - `A70_MachineModuleSources\MM_B01_MachineModuleSources\MM_B01_Infeed\Structs`  
   - Structure `MVis_scB01_Infeed`

   In the program part, declare two variables for the positive / negative manual jog:

   ```
   ' TYPE MVis_scB01_Infeed :
   STRUCT
   &scaleModuleName: STRING;
   &xLogPos  : BOOL ;
   &xLogNeg : BOOL ;
   ```

2. Insert the buttons into the visualization.
   
   Double-click and open:
   
   - `A70_MachineModuleSources\MM_B01_Infeed\Visu`  
   - Visualization `MVis_B01_Infeed`

   Go to the Tools dialog box and select the Common Controls section. Insert the buttons into the visualization interface:

   ![Common Controls](image)

3. Assign the variables.
   
   - Go to the Properties dialog box and assign the variable from the `MVis_scB01_Infeed` structure in the Input configuration\Tab\Variable section.
   - Go to the Properties area under Color Variables and assign the variable from the `MVis_scB01_Infeed` structure for `toggle color`.

   ![Property Assignments](image)
Assign the variables to the **AxisBasicControl** block.

- **Location:**
  
  Module application `MAP_B01_Infeed_App1\Action` **S00_CYCLIC**

- The variables must be interconnected to the `xJoggingPos` and `xJoggingNeg` inputs of **AxisBasicControl**.

- Assign the values for speed, acceleration, deceleration to the `L_SMCAxisBasicControl` function block:

  ```
  Axis1.xJoggingPos = Vis.xJogPos;
  Axis1.xJoggingNeg = Vis.xJogNeg;
  Axis1.IxJoggingSetVel = 10;
  Axis1.IxJoggingSetAcc = 10;
  Axis1.IxJoggingSetDec = 10;
  ```

  The assignments for the variables must be positioned in front of the call of the block instance. If the assignments are positioned behind, they will not be effective before the subsequent cycle.

---

### 12.8 Error handling: Configuring the error handling

In this example, the enable of the axes must be monitored during a state change. The error handling is supposed to trigger an error if the axes are not enabled within 200 ms.

- For this purpose, the axes must be inhibited via the hardware input.
- It should be possible to detect and control the error and the error response in the visualization.
- It should be possible to acknowledge the error via a button in the visualization.

The example below shows how the desired error response can be configured in the **MM_B01_Infeed** machine module.

**How to proceed:**

1. Declare the variables in the `Mvis_` structure.

   Double-click and open:
   
   - `A70_MachineModuleSources\MM_B01_Infeed`
   - Structure `MVis_scB01_Infeed`

   In the structure, declare a variable of the BOOL data type for acknowledging the error, example: `xQuit`.

   ```
   TYPE MVis_scB01_Infeed:
   STHRC
   MModName: STRING;
   xJogPos: BOOL; //Variable for manual jog CW
   xJogNeg: BOOL; //Variable for manual jog CCW
   xQuit: BOOL; //Variable to quit the error
   ```

2. Insert the buttons into the visualization.
Double-click and open:

- [A70_MachineModuleSources\ MM_B01_Infeed\Visu](#)
- [MVis_B01_Infeed](#)

Go to the Tools dialog box and select the Common Controls section. Insert the Error Quit button into the visualization interface:

3. Assign the variables: Visualization and module application
   - Click the previously created button.
   - Go to the Properties dialog box and select the Input configuration Tab Variable section to assign the Infeed.xQuit variable from the MVis_scB01_Infeed structure. The variable is declared in the A55_VarLists\MM_Vis folder.
   - Go to the Properties area under Color Variables and assign the same variable for toggle color.

Double-click and open:

- Folder [A70_MachineModuleSources\ MM_B01_Infeed\ModApp1](#)
- Module application [MAP_B01_Infeed_App1](#), action [Continuous](#)

The Infeed.xQuit variable must be assigned to the L_SMC_AxisBasicControl function block.

4. Define the error.

Double-click and open:

- Folder [A70_MachineModuleSources\ MM_B01_Infeed\ModApp1\BasicFunctions](#)
- Module application [MAP_B01_Infeed_App1](#), method [SetErrors](#)

The desired properties of the error must be defined on the ErrorsA block:

5. Declare and call the timer in the module application.

Double-click and open:

- Folder [A70_MachineModuleSources\ MM_B01_Infeed\ModApp1](#)
- Module application [MAP_B01_Infeed_App1](#)
Insert the timer into the declaration section of the MAP:

```
Axis1: L_SMC_AxisBasicControl;
```

Double-click and open:
- Folder `A70_MachineModuleSources\MM_B01_Infeed\ModApp1`
- Module application `MAP_B01_Infeed_App1`

Call the timer, deceleration time: 200 ms

```
Axis1.XResetError:=Vis.xOut;
Timer[FT:="T#200MS];
```

6. Program the error handling.

Add the program line:

```
ErrorsB.xSetError3:= NOT Axis1.xAxisEnabled AND Timer.0;
```

Double-click and open:
- Folder `A70_MachineModuleSources\MM_B01_Infeed\ModApp1\Map_B01_Infeed_App1 (FB)\States`
- Action `S02_Ready`

Insert the program line into the **Continuous** action:

```
SMDisableReadyToManual(xValue:= NOT Axis1.xAxisEnabled);
Axis1.xRegulatorOn:=FALSE;
Axis1.xDriveStart:=FALSE;
IF baseChannelGetNominalState (MM_Address:=0) = L_EATP_SMStates.Manual THEN
  Axis1.xRegulatorOn:=TRUE;
  Axis1.xDriveStart:=TRUE;
  Timer.IM:=TRUE.
END_IF
```

Double-click and open:
- Folder `A70_MachineModuleSources\MM_B01_Infeed\ModApp1\Map_B01_Infeed_App1 (FB)\States`
- Action `S03_Manual`

Insert the program line into the **Continuous** action:

```
IF baseChannelGetNominalState (MM_Address:=0) = L_EATP_SMStates.Ready THEN
  Axis1.xRegulatorOn:=FALSE;
  Axis1.xDriveStart:=FALSE;
  Timer.IM:=FALSE.
END_IF
```

Double-click and open:
- Folder `A70_MachineModuleSources\MM_B01_Infeed\ModApp1\Map_B01_Infeed_App1 (FB)\States`
- Action `S10_Fault`
Insert the program line into the **Continuous** action:

```plaintext
3  Tint.IN := FALSE;
4  baseChannelSetErrorQuit (MM_Address := 0, xValue:=Vis.xQuit);
```
13 The "ApplicationTemplateCounter" sample project

The ApplicationTemplateCounter is a sample project based on the ApplicationTemplate with two implemented machine modules. The visualization demonstrates the communication of the machine modules.

The user interface of the visualization is divided into the following areas:

- Area F visualises communication between the individual machine modules.
- In this area, the demo visualization of the sample programs is shown.

What does the sample program in the ApplicationTemplateCounter achieve?

- The counter example demonstrates that the machine modules (MM_Module1/MM_Module2) work depending on the specified state.
- The visualization illustrates communication between the machine modules by the ACD structure.

<table>
<thead>
<tr>
<th>Machine module/state</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM_Module1</td>
<td></td>
</tr>
<tr>
<td>Automatic</td>
<td>Incrementing (50 units per second)</td>
</tr>
<tr>
<td>Mode1</td>
<td>Decrementing (10 units per second)</td>
</tr>
<tr>
<td>Service</td>
<td>Load value 123</td>
</tr>
<tr>
<td>Fault</td>
<td>Set current value to zero.</td>
</tr>
<tr>
<td>Systemfault</td>
<td>Set current value to zero.</td>
</tr>
<tr>
<td>MM_Module2</td>
<td></td>
</tr>
<tr>
<td>Automatic</td>
<td>Incrementing (20 units per second)</td>
</tr>
<tr>
<td>Mode1</td>
<td>Incrementing (single steps up to 800000)</td>
</tr>
<tr>
<td>Mode2</td>
<td>Decrementing (5 units per second)</td>
</tr>
<tr>
<td></td>
<td>• In &quot;Mode2&quot; of module 2, the data can be transferred in a locked/unlocked manner (data consistency).</td>
</tr>
<tr>
<td></td>
<td>✓ L_EATP_CriticalSection (L 123)</td>
</tr>
<tr>
<td></td>
<td>✓ L_EATP_ErrorListHandler (L 124)</td>
</tr>
<tr>
<td>Service</td>
<td>Load value 456</td>
</tr>
<tr>
<td>Fault</td>
<td>Set current value to zero.</td>
</tr>
<tr>
<td>Systemfault</td>
<td>Set current value to zero.</td>
</tr>
<tr>
<td>MM_Machine</td>
<td></td>
</tr>
<tr>
<td>Add the value of module 1 to the value of module 2:</td>
<td>&lt;value module1&gt; + &lt;value module2&gt;</td>
</tr>
<tr>
<td>Subtract the value of module 2 from the value of module 1:</td>
<td>&lt;value module1&gt; - &lt;value module2&gt;</td>
</tr>
</tbody>
</table>
13.1 Sample programs in the machine module MM_Module1/MM_Module2

The respective module application calls the state machine.

- **Example:** In the case of the MM_Module1 machine module, the MAP_Module1_App1 module application calls the state machine.

13.1.1 "Automatic" status

If the "Automatic" status is active, the SMDispatcher method calls the S05_AUTOMATIC action:

- **MM_Module1:** "Incrementing by 50 units per second". The "tonParam.Q" timer in the corresponding MAP_Module1_App1 machine module is activated.
The "ApplicationTemplateCounter" sample project
Sample programs in the machine module MM_Module1/MM_Module2

• MM_Module2: "Incrementing by 20 units per second". The "ton20Hz" timer in the corresponding MAP_Module2_App1 machine module is activated.

13.1.2 "Mode1" status

If the "Mode1" status is active, the SMDispatcher method calls the S06_MODE1 action: "Decrementing by 10 units per second". The "ton10Hz" timer in the corresponding MAP_Module1_App1 machine module is activated.

• MM_Module2: The "ton10Hz" timer in the corresponding MAP_Module2_App1 machine module is activated.

13.1.3 "Mode2" status

If "Mode2" is active, the S07_MODE2 action calls "Decrementing by 5 units per second".

• MM_Module2: The "ton5Hz" timer in the corresponding MAP_Module2_App1 machine module is activated.

13.1.4 "Service" status

If the "Service" status is active, the SMDispatcher method calls the S04_SERVICE action:

• MM_Module1: "Load value 123".
• MM_Module2: "Load value 456".
13.1.5 "Fault"/"System fault" status

If the "Fault"/"System fault" status is active, the \texttt{SMDispatcher} method calls the \texttt{S10_FAULT}/\texttt{S11_SYSTEMFAULT} action, respectively.

- The actions \texttt{S10_FAULT}/\texttt{S10_SYSTEMFAULT} contain the "Set current value to zero" function.
13.2 Sample program MM_Machine machine module

The MAP_Machine_App1 module application of the MM_Machine machine module contains the following functions:

- Add the value of module 1 to the value of module 2:
  \( <\text{value module1}> + <\text{value module2}> \)

- Subtract the value of module 2 from the value of module 1:
  \( <\text{value module1}> - <\text{value module2}> \)
13.3 Simulation in the ApplicationTemplateCounter

Note!

In order to be able to use the simulation mode in the "ApplicationTemplateCounter", the "Watchdog" monitoring must be deactivated (corresponds to the standard setting in the ApplicationTemplate). An activated "Watchdog" monitoring results in an error message so that the simulation mode cannot be activated.

How to deactivate the "Watchdog" monitoring:

1. Double-click the A90_Resources folder in the device view.
   - Double-click Task configuration → TaskHighPriority.
   - On the Configuration tab, deactivate the Watchdog option by removing the checkmark:
     ![Watchdog option]

   - Double-click Task configuration → TaskMidPriority.
   - On the Configuration tab, deactivate the Watchdog option by removing the checkmark:
     ![Watchdog option]
## Appendix

### Method overview - ApplicationTemplate

#### 14.1 Accessing the state machine - the methods of the BaseChannel

<table>
<thead>
<tr>
<th>Identifier/function</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reading values</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| `baseChannelGetActualState` | `MM_Address` `L_EATP_MM_Address` | • Address of the module the current state of which is to be requested.  
• The value “0” and the `L_EATP_CONST.OWNID` constant address the module's intrinsic ID.  
• Address of the module the current state of which is to be requested.  
• The value “0” and the `L_EATP_CONST.OWNID` constant address the module's intrinsic ID. |
| **Return value/data type** | `L_EATP_SMStates` | • Current status of the machine module requested |
| `baseChannelGetErrorInformation` | `Input` | • Address of the module the current state of which is to be requested.  
• The value “0” and the `L_EATP_CONST.OWNID` constant address the module's intrinsic ID. |
| **Return values / data type** | | Current error information:  
• Priority,  
• Error number,  
• Error response type,  
• Acknowledgement response,  
• Error texts,  
• Error details. |
| `baseChannelGetNominalState` | `MM_Address` `L_EATP_MM_Address` | • Relative address of the module the current state of which is to be requested.  
• The value “0” and the `L_EATP_CONST.OWNID` constant address the module's intrinsic address. |
| **Return value/data type** | `L_EATP_SMStates` | • Desired status of the machine module requested |
### Method overview - ApplicationTemplate

<table>
<thead>
<tr>
<th>Identifier/function</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>baseChannelGetNominalStateFromMaster</strong></td>
<td>-</td>
<td><strong>Return value/data type</strong></td>
</tr>
<tr>
<td>▪ Determines the target status of the user's own module that is requested by the master module</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> It is only advisable to use this method when the DisableDefaultCouplingSlave function is active.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ In all other cases the method supplies the same result as the baseChannelGetNominalState call (L_EATP_CONST.OWNID).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM_Address</td>
<td>L_EATP_SMStates</td>
<td>▪ Desired status of the machine module requested</td>
</tr>
<tr>
<td><strong>baseChannelGetWarning</strong></td>
<td>MM_Address</td>
<td>BOOL</td>
</tr>
<tr>
<td>Determines whether a warning for a slave machine module/the user’s own module is active.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Writing values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>baseChannelSetNominalState</strong></td>
<td>MM_Address</td>
<td>L_EATP_MM_Address</td>
</tr>
<tr>
<td>▪ Sets the target status of the state machine for a slave module/the user’s own module.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> This method is only active when the DisableDefaultCouplingSlave function (target status for the user’s own module) or DisableDefaultCouplingMaster (target status for a slave module) is activated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eState</td>
<td>L_EATP_SMStates</td>
<td>▪ Setpoint status to be set.</td>
</tr>
<tr>
<td><strong>baseChannelSetErrorQuit</strong></td>
<td>MM_Address</td>
<td>L_EATP_MM_Address</td>
</tr>
<tr>
<td>▪ Sets the error acknowledgement bit for a slave module/the user’s own module.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> This method only works if the error acknowledgement of the user’s own module/the master master (DisableDefaultCouplingSlave/DisableDefaultCouplingMaster is activated).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xValue</td>
<td>BOOL</td>
<td>▪ TRUE: Error acknowledgement is active</td>
</tr>
<tr>
<td>▪ FALSE: Error acknowledgement is not active</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Inhibiting/enabling module-specific state transitions

<table>
<thead>
<tr>
<th>Identifier/function</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| SMDisableAutomaticToReady | xValue | • Controls the "Automatic"→"Ready" state transition  
  • TRUE: Inhibit state transition.  
  • FALSE: Enable state transition. |
| SMDisableReadyToAutomatic | xValue | • Controls the "Ready"→"Automatic" state transition |
| SMDisableReadyToManual | xValue | • Controls the "Ready"→"Manual" state transition |
| SMDisableReadyToMode1 | xValue | • Controls the "Ready"→"Mode1" state transition |
| SMDisableReadyToMode2 | xValue | • Controls the "Ready"→"Mode2" state transition |
| SMDisableReadyToHoming | xValue | • Controls the "Ready"→"Homing" state transition |
| SMDisableReadyToService | xValue | • Controls the "Ready"→"Service" state transition |
| SMDisableManualToReady | xValue | • Controls the "Manual"→"Ready" state transition |
| SMDisableMode1ToReady | xValue | • Controls the "Mode1"→"Ready" state transition |
| SMDisableMode2ToReady | xValue | • Controls the "Mode2"→"Ready" state transition |
| SMDisableHomingToReady | xValue | • Controls the "Homing"→"Ready" state transition |
| SMDisableServiceToReady | xValue | • Controls the "Service"→"Ready" state transition |
| SMEnableInitToReady | xValue | • Controls the "Init"→"Ready" state transition |

### Activate / deactivate quick stop response of the state machine

<table>
<thead>
<tr>
<th>Identifier/function</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| baseChannelDisableQSPFollowing | Input, MM_Address, L_EATP_MM_Address, xValue | • TRUE: Activate automatic quick stop response of the state machine.  
  • FALSE: Deactivate automatic quick stop response of the state machine. |
14.1.4 Activating/deactivating default coupling

<table>
<thead>
<tr>
<th>Identifier/function</th>
<th>Transfer value/data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate / deactivate decoupling from the state machine: machine control module / master module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMDisableDefaultCouplingMaster</td>
<td>xValue</td>
<td>BOOL</td>
</tr>
<tr>
<td>Decoupling of the machine control module from the state machine.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TRUE: Deactivate decoupling of the master.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• FALSE: Activate decoupling of the master.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

14.2 Tips&tricks

14.2.1 Renaming a machine module

For changing the module name the following steps have to be observed.

• Example: The following module names are to be changed:
  1. "Module1"→"VirtualMaster"
  2. "Module2"→"Transport2"

How to proceed:

1. Call the global variable list in the A55_VarLists folder. The illustration shows the textual representation of the variable list.
   • Instance machine modules (MFB_xxx).
   • Rename the instance names:
     "Module1"→"VirtualMaster"
     "Module2"→"Transport2"

2. The instance names in the A10_MachineModule folder of the machine module tree (MMT) must be renamed correspondingly (example: ApplicationTemplateCounter):
3. The instance names have to be renamed in every task call in the `A11_ModuleAppCalls` folder.
• Example: [MAC_Task_high](#)

4. The changed module names are then visible in the **Machine Module List**.

- [A20_Visualisation](#) → [L_Main](#) folder, click **M.-Modules** button

<table>
<thead>
<tr>
<th>Machine Modules</th>
<th>Machine Modules Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine 1</td>
<td>Machine Module Details</td>
</tr>
<tr>
<td>Module 1</td>
<td>Machine Address</td>
</tr>
<tr>
<td>Module 2</td>
<td>Machine Name</td>
</tr>
<tr>
<td>Module 3</td>
<td>MM CompID</td>
</tr>
<tr>
<td>Module 4</td>
<td>MM Version</td>
</tr>
<tr>
<td>Module 5</td>
<td>Master MM Address</td>
</tr>
<tr>
<td>Module 6</td>
<td>Master MM Name</td>
</tr>
<tr>
<td>Module 7</td>
<td>Number of Slaves</td>
</tr>
</tbody>
</table>

14.2.2 **Next steps: Renaming the visualization buttons**

To rename the visualization buttons, execute the following steps.

[How to proceed:](#)

1. Call the following visualization: [A20_Visualisation](#) → [Keys_Main](#) folder

- When the instance names are altered, the variable assignments have to be updated appropriately. In the [Keys_Main](#) visualization, click the **MM_Machine** area.
2. Under **View → Element properties**, assign the instance names to the visualizations to be referenced:

   ![Element properties screenshot]

3. Rename buttons:

   - Double-click the desired button, allocate the desired name.
   - Alternatively, the names of the buttons can be edited under **View → Element properties → Texts**.

4. In order to...
   - ...clean the project: **Build → Clean**
   - ...compile the project: **Build → Build (F11)**

5. When the project has been compiled successfully, the visualization is functional.
14.2.3 Recipe manager

General information on the recipe manager is provided in the »PLC Designer«. This section contains further information about...

- ...the creation of recipes
- ...the management of recipes via commands: RecipeManCommands

The recipe manager provides for the creation and management of recipes.

- Recipe definitions are variable lists that can be defined by the user.
- Recipes can be read by the controller/archived on the control.

Archiving recipes

All recipes available in the recipe definition are saved in separate files in each case (volutarily on the controller).

- The recipe files available on the controller can for instance be transferred to the hard disk of the Engineering PC.

How to proceed:

1. Click the desired axis in the Device view.
2. Click the associated Files tab:
3. Click the button to display the controller data.
   - Select the desired recipe file (*.txtrecipe)
   - Transfer the highlighted file to the target location by clicking the arrow key (to transfer the recipe file to the Engineering PC, you have to navigate to the local hard disk of the Engineering PC in the area).
14.2.4 Condition

• Every machine module has an `MPar_sc<M-identifier>` interface, which contains the parameters for the corresponding machine module.

• Every machine module interface must be instanced globally under `MM_Par` in the `A55_VarLists` folder:

```
.Module : MM_Par
* Summary : In that list one pod
declared for every a
* Note: The definition
* within the set
* History :
* Date Author
* -----------------------------------------------
* yyyy-mm-dd John Q. Public
14.0
14.0
14.0
14.0
14.0
14.0
14.0
14.0
14.0
14.0
```

14.2.5 Creating the recipe definition

By means of the recipe manager, different recipe definitions can be actively used within a project.

How to proceed:

1. In the `A90_Resources` folder, click the `Recipe manager`:

   ![Recipe Manager](image)

   - Use this dialog window to create/save a recipe definition.

2. By right-clicking the `Recipe manager`, execute the following command in the `A90_Resources` folder:

   ![Add object](image) `Recipe definition`
Appendix
Tips&tricks

14.2.6 Recipe definition - assigning variables

Use the input assistance to assign the corresponding variables to a newly created recipe definition.

How to proceed:

1. Double-click the desired recipe definition (example: MM_Par_Modul1)

2. On the tab of the recipe definition (example: MM_Par_Modul1), double-click in the Variable column:

- Click the button to start the Input assistance.

3. Use the Input assistance to assign the variables of the machine module to the recipe definition:

- Confirm the entry by clicking OK
• The variables assigned can be viewed on the tab of the recipe definition:

• After the "log in", the data of the PLC variables are available under 1.

14.2.7 Inserting recipes into a recipe definition

How to proceed:

1. Insert recipe into the recipe definition (example: MM_Par_Modul1): Right-click  Add recipe:

   • Allocate desired name.
   • Under "Copying of:" the recipes that are already available can be selected.
   • Click OK to insert the recipe.
   • Result: The recipe inserted (example: Recipe1) is visible as a further column in the recipe definition. Assign the desired values in the column to the recipe.

Via RecipeManCommands, further recipes can be managed from the PLC program.
14.2.8 Managing recipes with RecipeManCommands

This section contains information for the creation of recipes by means of the RecipeManCommands. To create a recipe, the name of the recipe definition available ([RecipeDef]) has to be specified.

- When the CreateRecipe is called, the applicable name of the recipe definition ([RecipeDef]) has to be specified, example: MM_Par_Modul1

[14-1] Example: Recipe definition MM_Par_Modul1

14.2.8.1 Create Recipe

- ...creates a new (empty) recipe.

```pascal
VAR
  _RecipeManCommands : RecipeManCommands;
  _stRecipeDef       : STRING := 'MM_Par_Modul1';  [Name of the recipe definition]
  _stRecipe3         : STRING := 'Recipe1';
END_VAR

  _RecipeManCommands.CreateRecipe(_stRecipeDef,_stRecipe3);
```

[14-2] Program example: Create the "Recipe1" recipe in the MM_Par_Modul1 recipe definition

Result: Afterwards the following files are available on the controller:

- The recipe manager has created the "Recipe1" recipe (file: Recipe1.MM_Par_Modul1.txtrecipe).
14.2.8.2 ReadAndSaveRecipeAs

...transfers the current PLC values to the recipe/file specified.

```
VAR
  _RecipeManCommands  :  RecipeManCommands;
  _stRecipeDef        :  STRING := 'MM_Par_Modul1';  [name of the recipe definition]
  _stRecipe2          :  STRING := 'Recipe2';  [name of the recipe]
  _stFileName1       :  STRING := 'Recipe_2011_10_01.txtrecipe';
END_VAR

MM_Par.scModule1.lrVel  :=  22;
MM_Par.scModule1.lrAcc  :=  12;
MM_Par.scModule1.lrDec  :=  13;
_RecipeManCommands.ReadAndSaveRecipeAs(_stRecipeDef, _stRecipe2, _stFileName1);
```

Program example: Transfer the current PLC values to the recipe/file "Recipe_2011_10_01.txtrecipe"/"Recipe2".

Initial situation - MM_Par_Modul1 recipe definition:

- Column 1 shows the current PLC values.
- Column 2 shows the recipe ("Recipe1")

Result:

- The recipe/file "Recipe_2011_10_01.txtrecipe" is created on the controller.
- The current PLC values are visible in the recipe/file.
14.2.8.3 LoadFromAndWriteRecipe

...loads the recipe from the file specified to "_stRecipe2" and 
...writes the recipe to the current PLC values

VAR
  _RecipeManCommands : RecipeManCommands;
  _stRecipeDef        : STRING := 'MM_Par_Modul1'; [name of the recipe
  definition]
  _stRecipe1: STRING := 'Recipe1';
  _stRecipe2: STRING := 'Recipe2';
  _stRecipe3: STRING := 'Recipe3';
  _stFileName1: STRING := 'Recipe2011_11_01.txtrecipe';
  _stFileName3: STRING := 'Recipe3. MM_Par_Modul1.txtrecipe';
END_VAR

MM_Par.Par.Printer.lrPar1:=325.68;
MM_Par.Par.Printer.wPar4:= 375;
  _RecipeManCommands. LoadFromAndWriteRecipe(_stRecipeDef, _stRecipe2, _stFileName1);

[14-5] Example: Program view

Initial situation:

Result:
The values of the "Recipe_2011_10_01.txtrecipe" recipe (file) have been...

• ...transferred to the current PLC values 
  (column 1).

• ...transferred to the "Recipe3.MM_Par_Modul1.txtrecipe" recipe (file).
14.2.8.4 WriteRecipe

...writes the values of a recipe to the PLC variables.

```plaintext
VAR
  _RecipeManCommands : RecipeManCommands;
  _stRecipeDef : STRING := 'MM_Par_Modul1';  [name of the recipe definition]
  _stRecipe2 : STRING := 'Recipe2';
END_VAR

 ריקריפט(_stRecipeDef,_stRecipe2);
```

[14-6] Example: Program view

Initial situation:

- The values (to be transferred) of the "Recipe2.MM_Par_Modul1.txt" recipe file.

Result:

- ...transferred to the values of MFB_Module1.

- ...transferred to the current PLC values (column 1).
14.2.8.5 ReadRecipe

...transfers the current PLC variable values to a selected recipe.

```delphi
VAR
  _RecipeManCommands : RecipeManCommands;
  _stRecipeDef : STRING := 'MM_Par_Modul1'; // name of the recipe definition
  _stRecipe2 : STRING := 'Recipe2';
END_VAR

_endRecipe(_stRecipeDef, _stRecipe2);
```

- The current PLC values are...
  - ...visible in column "1".
  - ...visible in the "Recipe.MM_Par_Modul1.txtrecipe" recipe (file).
14.2.8.6 GetRecipeCount

...outputs the number of recipes within the recipe definition.

```
VAR
  _RecipeManCommands : RecipeManCommands;
  _stRecipeDef       : STRING := 'MM_Par_Modul1'; [name of the recipe definition]
  _RecipeCount      : INT;
END_VAR

_RecipeCount := _RecipeManCommands.GetRecipeCount(_stRecipeDef);
```

[14-8] Example: Program view
Index

Zahlen
1_POUs 121
2_Structs_Types 133
3Enums 136

A
Access from machine modules with several module applications 91
AccessACD 121
Accessing structure variables of machine modules 73
Accessing structures of other machine modules 74
Accessing the ACD structure 76
Accessing the module's intrinsic structure variables 73
Accessing the state machine - the methods of the BaseChannel 165
ACD structure 75
Acknowledging errors 95, 126
Acknowledging errors - response in the machine module 96
Adding further devices - EtherCAT bus system 41
Addressing the machine modules 21
appChannelLock 105
appChannelUnlock 105, 124
Application notes (representation) 12
ApplicationTemplate 17
AT_ACTION_ADD_MODULE 55
AT_ACTION_ADD_MODULEAPPLICATION 68
AT_ACTION_CREATE_NEW_MODULE 52
AT_ACTION_CREATE_NEW_MODULEAPPLICATION 64
Automatically generated functions 117

B
Base 122
Base (method) 127
BaseChannel 22
BaseChannel methods 165
baseChannelSetNominalState 84

C
CANbus 45
CANopen I/O image 50
CANopen Manager 46
CANopen Remote Device 49
checkSlave 129
Communication between the machine modules 22
Compiling the project data 35
Configuring an error response 95
Consistent data transfer 105
Conventions used 11
Creating a machine module 52
Creating a project - standard procedure 25
Creating instances of machine modules 53
Creating module applications 64
Creating the recipe definition 173

D
Declaring/recording the ACD structure in the MFB 75
DefaultCoupling 85
Defining errors - standard procedure 96
DisableDefaultCouplingMaster 87
Document history 10
Downloading and starting the PLC program 35
Dynamic memory 34

E
Elements of the ApplicationTemplate 19
E-mail to Lenze 185
Error handling 24, 94
Defining errors 94
Triggering errors 97
Error List 98
Export CSV file 100
Error response types 94
ErrorList Handler 98
Establishing communication with the controller 15
Extending the visualization 110

F
FB_L_EATP_SMAccess 91
Feedback to Lenze 185

G
GetBooleanProperty 117
GetCompany 118
GetNumberProperty 118
GetRecipeCount 181
GetTextProperty 119
Getting started - operating the ApplicationTemplate 36
GetTitle 119
GetVersion 120
GetVersionProperty 120

I
Influencing state transitions 90
Inserting an axis 60
Inserting recipes into a recipe definition 175
Integrating a created machine module 55
Integrating a module application 68
Integrating I/O modules 62
Integrating machine modules in the machine structure tree 143
Internal Control 107

L
L_EATP_ACD_Base 133
L_EATP_Application
checkSlave 122
Local variables 121
L_EATP_ApplicationTemplate 116
Index

L_EATP_BaseChannel 134
L_EATP_CONST.OWNID 84, 88, 89
L_EATP_CriticalSection 123
L_EATP_ErrorListEntry 134
L_EATP_ErrorListHandler 124, 125
L_EATP_ErrorReactionType 136
L_EATP_ErrorSet 125
L_EATP_MM_Address 136
L_EATP_MMD_Base 135
L_EATP_Module 127
L_EATP_ModuleDiag 129
L_EATP_ModuleErrorHandler 130
L_EATP_ModulRelations 131
L_EATP_MVis 136
L_EATP_SMAccess 132
L_EATP_SMStates 137
LoadFromAndWriteRecipe 178
Lock 123
LockState 123
Logbook 101
Logging in 35

M
MAC_Task_Free 104
MAC_Task_High 104
MAC_Task_Mid 104
Machine module 20
Machine module (MM) 20
Machine module tree 19, 20
MachineModuleTree 19
MAP 21
Method overview - ApplicationTemplate 165
MFB_Module1 66
MM 20
MM_Dcl 31
MM_EmptyModule 52
MM_IO 31
MM_Par 31
MM_PD 31
MM_Vis 31
MMT 19
Modularising the automation system 27
Module address, absolute 21
Module address, relative 21
Module application 21
Module diagnostics 103
Module ID 58
ModuleAppCalls 21, 28, 31, 104
Multitasking 104

N
Notes used 12

O
Online change 70
Online Change not possible 71

P
PDO mapping 50
Programming with the module application 67

Q
Query example
  Enabling a state transition 91
  Enabling/inhibiting a state transition 92
  Querying the state machine (IF condition) 91
  Querying the TARGET status of the state machine of the higher-level master 91
Query examples
  State machine 91

R
ReadAndSaveRecipeAs 177
ReadRecipe 180
Recipe manager 172
RecipeManCommands
  Create Recipe 176
Renaming a module 55

S
Safety 14
Safety instructions (representation) 12
Sample program
  ApplicationTemplateCounter 159
Sample program MM_Machine module 163
Screenshots 9
setCompIDAndVersion 128
SetErrors 65
SetErrors (method) 126
Setting of CAN parameters (Logic devices) 49
Setting of PDO mapping (Logic devices) 49
Setting sync generation 46
Simulation 164
SMDispatcherCaller 122
Standard response 85
Starting the ApplicationTemplate 32
State entry 93
State exit 93
State machine 23, 79
State machine - program examples 91
State transitions 81
State transitions (initial values) 80
State transitions and conditions - overview 23
States 84
Index

Structure of a machine module 20
Structuring within a machine module 28

T
Target group 9
Target of the ApplicationTemplate 17
Transferring the functionality of the machine structure to machine modules 27
Transferring the project to the control system (logging in) 35

V
Validity of the documentation 10
Visualisation of the machine modules 38

W
Working with the ApplicationTemplate 40
WriteRecipe 179
Your opinion is important to us

We have created these instructions to the best of our knowledge with the objective of providing you with the best possible support when handling our product.

Perhaps we have not succeeded in achieving this objective in every respect. If you notice this, please send your suggestions and points of criticism in a short e-mail to:

feedback-docu@Lenze.de

Thank you for your support.

Your Lenze documentation team