EcoStruxure Machine Expert

PackML

Library Guide

EIO0000002809.03
12/2025

www.se.com

Legal Information

The information provided in this document contains general descriptions, technical characteristics and/or recommendations related to products/solutions.

This document is not intended as a substitute for a detailed study or operational and site-specific development or schematic plan. It is not to be used for determining suitability or reliability of the products/solutions for specific user applications. It is the duty of any such user to perform or have any professional expert of its choice (integrator, specifier or the like) perform the appropriate and comprehensive risk analysis, evaluation and testing of the products/solutions with respect to the relevant specific application or use thereof.

The Schneider Electric brand and any trademarks of Schneider Electric SE and its subsidiaries referred to in this document are the property of Schneider Electric SE or its subsidiaries. All other brands may be trademarks of their respective owner.

This document and its content are protected under applicable copyright laws and provided for informative use only. No part of this document may be reproduced or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), for any purpose, without the prior written permission of Schneider Electric.

Schneider Electric does not grant any right or license for commercial use of the document or its content, except for a non-exclusive and personal license to consult it on an "as is" basis.

Schneider Electric reserves the right to make changes or updates with respect to or in the content of this document or the format thereof, at any time without notice.

To the extent permitted by applicable law, no responsibility or liability is assumed by Schneider Electric and its subsidiaries for any errors or omissions in the informational content of this document, as well as any non-intended use or misuse of the content thereof.

Table of Contents

Safety Information 7
QUALIFICATION OF PERSONNEL 7
PROPER USE 8
Before You Begin 8
Start-up and Test 9
Operation and Adjustments 9
About the Document 11
General Information 17
Presentation of the Library 18
General Information 18
Operation Modes 19
PackTags 22
Diagnostic Concept 23
Diagnostic Concept 23
Common Inputs and Outputs 25
Behavior of Function Blocks with the Input *i_xEnable* 25
Data Unit Types 26
Enumerations 27
ET_Cmd 27
ET_Modes 28
ET_Result 28
ET_StateModelDefinition 30
ET_States 31
Structures 33
PackTags Main Structures 33
ST_Administration 33
ST_Command 35
ST_Status 36
Other Main Structures 37
ST_InitAlarm 37
ST_UnitModeDefinition 38
ST_VisInterface 39
SubStructures 42
ST_Alarm 42
ST_CountDescrip 43
ST_Cumulative_Times 43
ST_Descriptor 44
ST_EquipmentInterlock 45
ST_Ingredient 45
ST_Ingredients 46
ST_Interface 46
ST_ModeState_Times 47
ST_Parameter_DINT 47
ST_Parameter_LREAL 48
ST_Parameter_REAL 49
ST_Parameter_STRING 49
ST_ProcessVariables 50
ST_Product 51

ST_Product_Data 51
ST_Recipe 52
ST_StateInfo 53
ST_Timestamp 53
Aliases - *DateTimeArray* 55
DateTimeArray 55
Global Variables 56
Global Constants List 57
GCL 57
Global Parameter List 58
GPL 58
Global Variable List 61
GVL_VisuParam 61
Interfaces 62
IF_UnitMode 63
IF_UnitMode – General Information 63
IF_StateCommands 64
IF_StateCommands – General Information 64
CmdReset (Method) 64
CmdStart (Method) 65
CmdStop (Method) 66
CmdHold (Method) 66
CmdUnHold (Method) 67
CmdSuspend (Method) 68
CmdUnSuspend (Method) 68
CmdAbort (Method) 69
CmdClear (Method) 70
StateComplete (Method) 70
IF_StateModelHandler 72
IF_StateModelHandler – General Information 72
Program Organization Units (POU) 73
Function Blocks 74
FB_DataManagement 74
FB_ModeManager 76
FB_UnitModeManager2 79
FB_UnitModeManager2 – General Information 79
PackML Base State Model 80
Properties 81
DefineUnitMode (Method) 81
DefineUnitModeWithHandler (Method) 83
RegisterLoggerPoint (Method) 84
SetApplicationLoggerLogLevel (Method) 86
ExecuteCurrentState (Method) 87
CmdChangeUnitMode (Method) 87
RemoveUnitMode (Method) 88
GetDefinedUnitModes (Method) 89
Command (Method) 90
FB_StateModelHandlerBase 90
FB_StateModelHandlerBase – General Information 90
Clearing (Method) 91

Stopped (Method) 91
Starting (Method) 92
Idle (Method) 92
Suspended (Method) 93
Execute (Method) 93
Stopping (Method) 94
Aborting (Method 94
Aborted (Method) 95
Holding (Method) 95
Held (Method) 96
Restarting (Method) 96
Suspending (Method) 97
Unsuspending (Method) 97
Resetting (Method) 98
Completing (Method) 98
Complete (Method) 99
FB_VisController 99
Functions 101
FC_CheckCmd 101
FC_CheckCmd2 102
FC_EtResultToString 103
FC_GetDateTimeAsArray 104
FC_InitStateModelChangeStates 104
FC_InitStateModelExistingStates 106
FC_SetAlarm 108
FC_SetWarning 110
FC_StateSelectionAsDint 111
FC_StateSelectionAsDintFromBool 111
Visualization 113
BackgroundFrames 114
FR_<BackgroundFrame> 114
States Frames 116
FR_<State> 116
Visualization Frames 117
General 117
FR_Alarm 118
FR_AlarmHistory 118
FR_CurrentModeAndStateTime 119
FR_DynStateModel 120
FR_DynStateModel_2022 121
FR_ModeAndStateTime 122
FR_PackMLStateModel_2022_Base 123
FR_PackMLStateModel_2022_UnitModeManager 124
FR_ProdConsumedCount 125
FR_ProdDefectiveCount 126
FR_ProdProcessedCount 127
FR_Warning 128
FR_StopReason 128
FR_StatesDisabled 129
Glossary 131

Index 132

Safety Information

Important Information

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.



The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.



This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

Please Note

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

Before You Begin

Do not use this product on machinery lacking effective point-of-operation guarding. Lack of effective point-of-operation guarding on a machine can result in serious injury to the operator of that machine.

⚠WARNING

UNGUARDED EQUIPMENT

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

This automation equipment and related software is used to control a variety of industrial processes. The type or model of automation equipment suitable for each application will vary depending on factors such as the control function required, degree of protection required, production methods, unusual conditions, government regulations, etc. In some applications, more than one processor may be required, as when backup redundancy is needed.

Only you, the user, machine builder or system integrator can be aware of all the conditions and factors present during setup, operation, and maintenance of the machine and, therefore, can determine the automation equipment and the related safeties and interlocks which can be properly used. When selecting automation and control equipment and related software for a particular application, you should refer to the applicable local and national standards and regulations. The National Safety Council's Accident Prevention Manual (nationally recognized in the United States of America) also provides much useful information.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator's hands and other parts of the body are free to enter the pinch points or other hazardous areas and serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

NOTE: Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the Function Block Library, System User Guide, or other implementation referenced in this documentation.

Start-up and Test

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a check are made and that enough time is allowed to perform complete and satisfactory testing.

⚠WARNING

EQUIPMENT OPERATION HAZARD

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters, and debris from equipment.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Follow all start-up tests recommended in the equipment documentation. Store all equipment documentation for future references.

Software testing must be done in both simulated and real environments.

Verify that the completed system is free from all short circuits and temporary grounds that are not installed according to local regulations (according to the National Electrical Code in the U.S.A, for instance). If high-potential voltage testing is necessary, follow recommendations in equipment documentation to prevent accidental equipment damage.

Before energizing equipment:

- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.
- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

Operation and Adjustments

The following precautions are from the NEMA Standards Publication ICS 7.1-1995:

(In case of divergence or contradiction between any translation and the English original, the original text in the English language will prevail.)

- Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly operated.

- It is sometimes possible to misadjust the equipment and thus produce unsatisfactory or unsafe operation. Always use the manufacturer’s instructions as a guide for functional adjustments. Personnel who have access to these adjustments should be familiar with the equipment manufacturer’s instructions and the machinery used with the electrical equipment.
- Only those operational adjustments required by the operator should be accessible to the operator. Access to other controls should be restricted to prevent unauthorized changes in operating characteristics.

About the Document

Document Scope

This document describes the PackML library.

The library provides functions and function blocks, structures, and visualization frames as tools to help you create applications conforming to PackML and PackTags, as defined by ANSI/ISA TR88.00.02-2022.

The library is not, and cannot be, a full implementation of the ANSI/ISA TR88.00.02-2022 standard, as you are responsible for the application code to support your machine or process. Instead, the library is a tool to help you to deploy the PackML standard for machine application.

Validity Note

This document has been updated for the release of EcoStruxure™ Machine Expert V2.5.

Product Related Information

⚠WARNING

LOSS OF CONTROL

- Perform a Failure Mode and Effects Analysis (FMEA), or equivalent risk analysis, of your application, and apply preventive and detective controls before implementation.
- Provide a fallback state for undesired control events or sequences.
- Provide separate or redundant control paths wherever required.
- Supply appropriate parameters, particularly for limits.
- Review the implications of transmission delays and take actions to mitigate them.
- Review the implications of communication link interruptions and take actions to mitigate them.
- Provide independent paths for control functions (for example, emergency stop, over-limit conditions, and error conditions) according to your risk assessment, and applicable codes and regulations.
- Apply local accident prevention and safety regulations and guidelines.[1]
- Test each implementation of a system for proper operation before placing it into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), *Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control* and to NEMA ICS 7.1 (latest edition), *Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems* or their equivalent governing your particular location.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠WARNING

IMPROPER USE OF PROGRAM ORGANIZATION UNITS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Ensure that the axis is homed and that the homing is valid before usage of absolute movements or POUs using absolute movements.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Incomplete file transfers, such as data files, application files and/or firmware files, may have serious consequences for your machine or controller. If you remove power, or if there is a power outage or communication interruption during a file transfer, your machine may become inoperative, or your application may attempt to operate on a corrupted data file. If an interruption occurs, reattempt the transfer. Be sure to include in your risk analysis the impact of corrupted data files.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION, DATA LOSS, OR FILE CORRUPTION

- Do not interrupt an ongoing data transfer.
- If the transfer is interrupted for any reason, re-initiate the transfer.
- Do not place your machine into service until the file transfer has completed successfully, unless you have accounted for corrupted files in your risk analysis and have taken appropriate steps to prevent any potentially serious consequences due to unsuccessful file transfers.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Care must be taken and provisions made for use of this library for machine control to avoid inadvertent consequences of commanded machine operation, state changes, or alteration of data memory or machine operating elements.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

- Place operator devices of the control system near the machine or in a place where you have full view of the machine.
- Protect operator commands against unauthorized access.
- If remote control is a necessary design aspect of the application, ensure that there is a local, competent, and qualified observer present when operating from a remote location.
- Configure and install the Run/Stop input, if so equipped, or, other external means within the application, so that local control over the starting or stopping of the device can be maintained regardless of the remote commands sent to it.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

The POUs provided with this library use variables of type POINTER TO internally. These pointers are assigned only on start of execution of the respective POU. This is, the pointers are not reassigned while the function block indicates `Busy`.

⚠CAUTION

INVALID POINTER

Do not use the “Online Change” command or the “Log in with online change” option as long as one of the function blocks of this library indicates `Busy` in your running application.

Failure to follow these instructions can result in injury or equipment damage.

General Cybersecurity Information

In recent years, the growing number of networked machines and production plants has seen a corresponding increase in the potential for cyber threats, such as unauthorized access, data breaches, and operational disruptions. You must, therefore, consider all possible cybersecurity measures to help protect assets and systems against such threats.

To help keep your Schneider Electric products secure and protected, it is in your best interest to implement the cybersecurity best practices as described in the Cybersecurity Best Practices document.

Schneider Electric provides additional information and assistance:

- Subscribe to the Schneider Electric security newsletter.
- Visit the Cybersecurity Support Portal web page to:
  - Find Security Notifications.
  - Report vulnerabilities and incidents.
- Visit the Schneider Electric Cybersecurity and Data Protection Posture web page to:
  - Access the cybersecurity posture.
  - Learn more about cybersecurity in the cybersecurity academy.
  - Explore the cybersecurity services from Schneider Electric.

Product Related Cybersecurity Information

Refer to the Cybersecurity Guidelines for EcoStruxure Machine Expert, Modicon and PacDrive Controllers and Associated Equipment User Guide.

NOTE: Schneider Electric adheres to industry best practices in the development and implementation of control systems. This includes a "Defense-in-Depth" approach to secure an Industrial Control System. This approach places the controllers behind one or more firewalls to restrict access to authorized personnel and protocols only.

⚠WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

- Evaluate whether your application environments are connected to your critical infrastructure and, if so, take appropriate steps in terms of prevention, based on Defense-in-Depth, before connecting the automation system to any network.
- Limit the number of devices connected to a network to the minimum necessary.
- Isolate your industrial network from other networks inside your company.
- Protect any network against unintended access by using firewalls, VPN, or other, proven security measures such as an Intrusion Prevention System or Intrusion Detection System.
- Monitor activities within your systems.
- Prevent subject devices from direct access or direct link by unauthorized parties or unauthenticated actions.
- Install certificates that are issued by publicly known Trusted Certificate Authorities.
- Keep your systems up to date and rely only on legitimate sources.
- Prepare a recovery plan including backup of your system and process information.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

For more information on organizational measures and rules covering access to infrastructures, refer to ISO/IEC 27000 series, Common Criteria for Information Technology Security Evaluation, ISO/IEC 15408, IEC 62351, ISA/IEC 62443, NIST Cybersecurity Framework, Information Security Forum - Standard of Good Practice for Information Security and refer to Cybersecurity Guidelines for EcoStruxure Machine Expert, Modicon and PacDrive Controllers and Associated Equipment.

Available Languages of the Document

The document is available in these languages:

- English (EIO0000002809)
- French (EIO0000002810)
- German (EIO0000002811)
- Italian (EIO0000002812)
- Spanish (EIO0000002813)
- Chinese (EIO0000002814)

Related Documents

Document title	Reference
Cybersecurity Best Practices	CS-Best-Practices-2019-340
Cybersecurity Guidelines for EcoStruxure Machine Expert, Modicon and PacDrive Controllers and Associated Equipment	EIO0000004242
EcoStruxure Machine Expert Functions and Libraries User Guide	EIO0000002829 (ENG); EIO0000002830 (FRE); EIO0000002831 (GER); EIO0000002832 (ITA); EIO0000002833 (SPA); EIO0000002834 (CHS);
EcoStruxure Machine Expert Programming Guide	EIO0000002854 (ENG); EIO0000002855 (FRE); EIO0000002856 (GER); EIO0000002857 (ITA); EIO0000002858 (SPA); EIO0000002859 (CHS);

To find documents online, visit the Schneider Electric download center (www.se.com/ww/en/download/).

Information on Non-Inclusive or Insensitive Terminology

As a responsible, inclusive company, Schneider Electric is constantly updating its communications and products that contain non-inclusive or insensitive terminology. However, despite these efforts, our content may still contain terms that are deemed inappropriate by some customers.

Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in the information contained herein, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2023	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2020	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction

Standard	Description
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2015	Safety of machinery - Emergency stop - Principles for design
IEC 62061:2021	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2021	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

NOTE: The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

General Information

What’s in This Part

Presentation of the Library 18
Diagnostic Concept 23
Common Inputs and Outputs 25

Presentation of the Library

What's in This Chapter

General Information....18
Operation Modes....19
PackTags....22

General Information

Library Overview

The library provides components to facilitate the implementation of an application based on the PackML standard. The names and structure of the data types provided in this library correspond to the definitions in ANSI/ISA TR88.00.02-2022.

Characteristics of the Library

Characteristic	Value
Library title	PackML
Company	Schneider Electric
Category	**Application > Solution > Packaging**
Component	**Packaging** (**Application Libraries and Templates > Packaging**)
Default namespace	**PackML**
Language model attribute	qualified-access-only (see Functions and Libraries User Guide)
Forward compatible library	Yes (FCL (see Functions and Libraries User Guide))

NOTE: For this library, qualified-access-only is set. The POUs, data structures, enumerations, and constants must be accessed using the namespace of the library. The default namespace of the library is **PackML**.

Example Project

In conjunction with the library, the example project PackMLExample.project is provided. The example project demonstrates how to implement the components from the PackML library.

Your specific application requirements may be different from those assumed for any related examples, templates or architectures described herein. In that case, you will have to adapt the information provided in this and other related documents to your particular needs. To do so, you will need to consult the specific product documentation of the hardware and/or software components that you may add or substitute for any information specified in this documentation. Pay particular attention and conform to any safety information, different electrical requirements and normative standards that would apply to your adaptation.

⚠WARNING

REGULATORY INCOMPATIBILITY

Be sure that all equipment applied and systems designed comply with all applicable local, regional and national regulations and standards.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

NOTE: The following instructions pertain to EcoStruxure Machine Expert ≤V2.3. In EcoStruxure Machine Expert V2.5, use the EcoStruxure Machine Expert - portal to open a new project. For more information, refer to the Home Page chapter of the EcoStruxure Machine Expert Portal User Guide.

The example project is installed on your PC along with the programming software. To open the project example, proceed as follows:

Step	Action	Result
1	In EcoStruxure Machine Expert ≤V2.3, run the command **File > New Project**.	–
2	In the **New Project** dialog box, select the option **From Example** from the **Project type** list.	–
3	On the right-hand side of the **New Project** dialog box, click **Toggle Filter**.	Available examples are listed in the drop down menu.
4	Select your example from the drop-down menu.	–
5	Select your controller from the **Controllers** list.	–
6	Enter a name for the project, and select the file location.	–
7	Click **OK**.	A project is created based on the selected example.

Operation Modes

Overview

PackML guidelines define standardized state model to demonstrate the fundamental behavior of a packaging machine in a standardized manner.

According to the PackML guidelines each packaging machine may support many operation modes.

Each operation mode can be comprised of up to 17 states (**Aborted**, **Aborting**, **Clearing**, **Complete**, **Completing**, **Execute**, **Held**, **Holding**, **Idle**, **Resetting**,

Starting, **Stopped**, **Stopping**, **Suspended**, **Suspending**, **Un-Holding** and **Un-Suspending**).

To move from one state to another, there are transitions between states. Each transition is triggered by either a state transition command or the SC (State Complete) condition.

The implementation of the PackML library focuses on the unit control mode as described in ANSI/ISA TR88.00.02-2022 standard. In all cases, when the present document refers to operation mode, it is a specific reference to unit control mode.

Using State Models

State models in the PackML library are based on the state model with 17 states, as defined in ANSI/ISA TR88.00.02-2022, refer to the diagram thereafter. A state model must contain these states, or a subset of these states.

A state model is defined by specifying which of these states exist for the operation mode. Transitions between states are created automatically and implicitly based on which states are selected.

ST_UnitModeDefinition is the structure representing the state model for an operation mode. This structure is required for the implementation of the function block *FB_ModeManager*. The structure can be initialized with the functions *FC_InitStateModelExistingStates()* and *FC_InitStateModelChangeStates()*.

When using the function block *FB_UnitModeManager2*, the desired operation modes are defined by means of the methods *DefineUnitMode()* or *DefineUnitModeWithHandler()*. The *ST_UnitModeDefinition* structure is provided by the *GetDefinedUnitModes()* method.

The function blocks *FB_ModeManager* or *FB_UnitModeManager2* verify the state model for consistency and verify whether a change of operation mode is possible in the current state.

FC_CheckCmd2 verifies whether a state transition command is valid and returns the resulting target state.

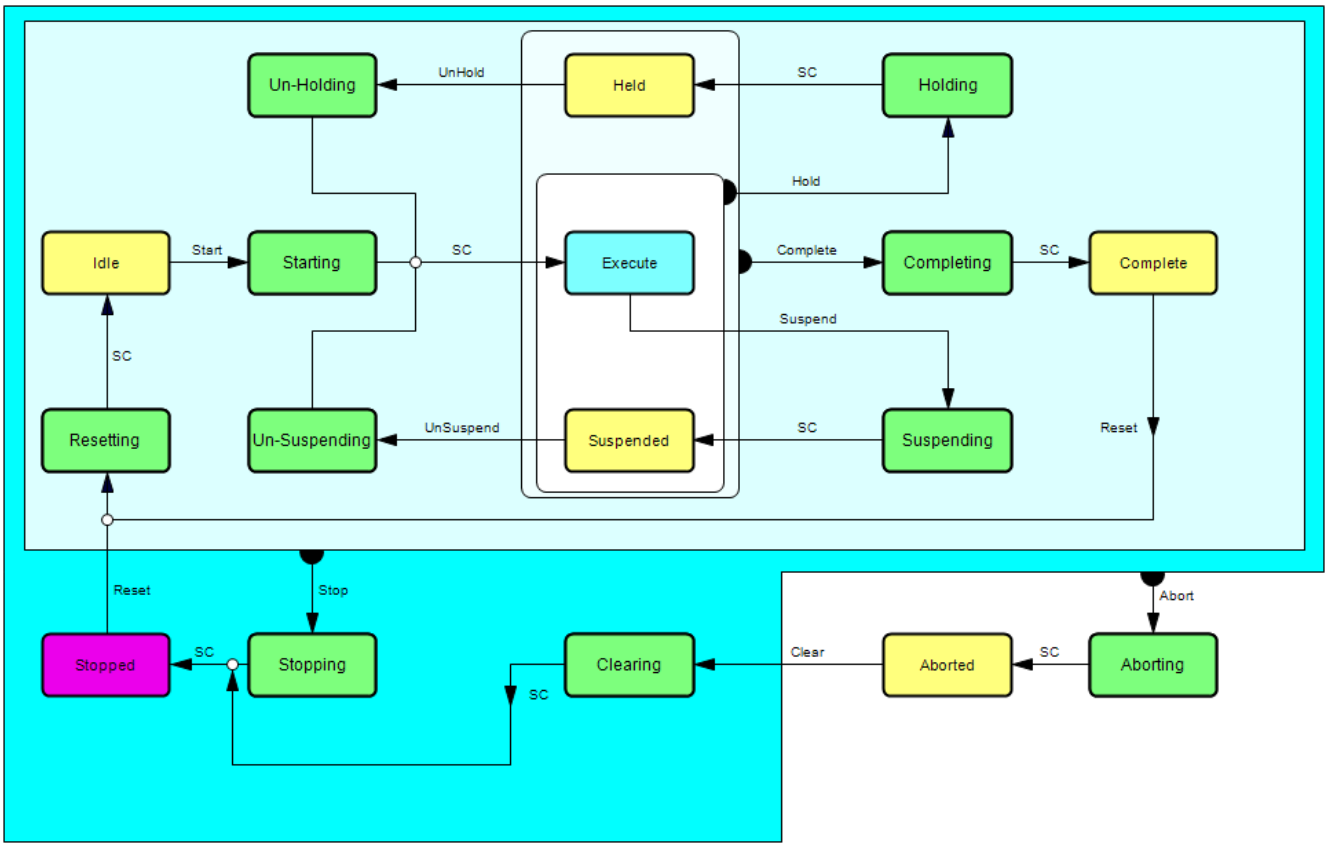
Visualization frames and *FB_VisController* are used to visualize the state model and provide a commissioning interface for it.

You are responsible for implementing the correct flow in the state model according to the transitions defined in the state model, as well as for implementing changes of operation mode. POUs provided in the library can help to implement this logic (*FB_ModeManager* in combination with *FC_CheckCmd* or *FC_CheckCmd2*, or *FB_UnitModeManager2* using its methods).

Implement the tracking of the state and operation mode, as well as for implementing a machine behavior that corresponds to the states, the transitions, and the operation modes.

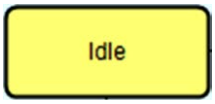
State Model Diagram

The following diagram illustrates the state model for operation modes with the 17 states present:



Type of States

Within the state model, three types of states are differentiated:

Type of state	Description from ANSI/ISA TR88.00.02-2022	Example
Wait state	A state used to identify that a machine has achieved a defined set of conditions. In such a state, the machine is maintaining a status until transitioning to an acting state.	
Acting state	A state which represents some processing activity. It implies the single or repeated execution of processing steps in a logical order, for a finite time or until a specific condition has been reached.	 

Boundaries

The following list describes the boundaries which define operation modes based on the state model:

- Maximum number of states: 17

  The states for the operation mode can be selected from those pre-defined in the state model. Each state can appear only once and whose position is specified by the state model. You cannot add additional states, nor change the order of the states.
- State model is unidirectional.
- The state following a **Wait state** must be an **Acting state**.
- Logic can be executed in each state.

NOTE: More rules apply to build operation modes with the PackML library. Refer to *FB_ModeManager*, page 76 or *FB_UnitModeManager2*, page 79.

PackTags

Overview

PackTags provide a uniform set of naming conventions for data elements used within the procedural elements of the base state model. They are named data elements used for open architecture and interoperable data exchange in automated machinery.

The library provides the PackTags, as defined in ANSI/ISA TR88.00.02-2022.

Variables

The variables are grouped into three types:

PackTags type	Description	In the PackML Library
command	These variables mainly represent the control interface. By using these tags, operation modes can be selected, state transitions can be forced, and recipes as well as formats can be transmitted.	*ST_Command*
status	These variables contain status information (such as which operation mode is active, which state is active, and so on).	*ST_Status*
administration	These variables contain important OEE data (for example, downtime, reasons for downtime, and so on). Furthermore production information.	*ST_Administration*

The individual variables from the three types (command, status, administration) can be provided on the controller within the structures.

Diagnostic Concept

What's in This Chapter

Diagnostic Concept 23

Diagnostic Concept

Overview

EcoStruxure Machine Expert provides a three-layer diagnostic concept for the libraries. This concept is applicable to the Technology/Module libraries (for example, the library CommonToolbox) and uses enumerations for diagnostic coding.

The diagnostic information has the following layers:

1. General information on the exception: Provides basic details about the exception without requiring specific knowledge of the POU functionality.
2. POU-specific diagnostic and status messages (part 1): Offers information on the source triggering the diagnostic or status messages.
3. POU-specific diagnostic and status messages (part 2): Provides detailed and dynamic information on the source triggering the diagnostic or status messages.

   This information changes at runtime (for example, information about the condition of the input parameters). This diagnostic output is optional for the POUs.

The diagnostic concept offers the following advantages:

- Diagnostic messages can be viewed online
- Detailed information on diagnostic events, including diagnostic codes
- Overview of the status or exceptional condition of a POU
- Pertinent suggested solutions to address the causes for exceptional conditions
- Enumerated diagnostic messages to facilitate multi-language support for HMI displays

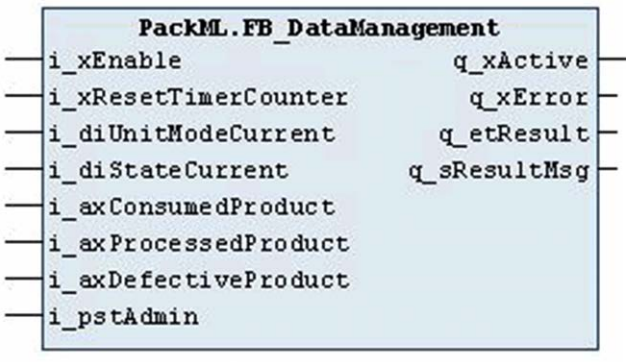
Diagnostic Outputs

Function blocks and functions or methods can have the three diagnostic outputs *q_xError*, *q_etResult*, and *q_sResultMsg*. The outputs are defined in the POU one after another.

Output	Data type	Meaning
q_xError	*BOOL*	The output *q_xError* is set to TRUE when an error has been detected during execution of the function block. The outputs *q_etResult* and *q_sResultMsg* provide the corresponding error code and a plain text information.
q_etResult	*ET_Result*(1)	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information. The enumeration *ET_Result* contains the possible values of the POU operation results.

Output	Data type	Meaning
q_sResultMsg	STRING	Provides additional information about the present status of the POU. If the POU is in error state (*q_xError* = TRUE), the message provides additional information for the error cause and, if meaningful, a possible solution. If the POU is busy the present process or state is issued at this output.
(1) Each library provides its own, specific enumeration *ET_Result* which contains the possible *q_etResult* outputs for that library.		

Diagnostic information example:



Common Inputs and Outputs

What's in This Chapter

Behavior of Function Blocks with the Input *i_xEnable* 25

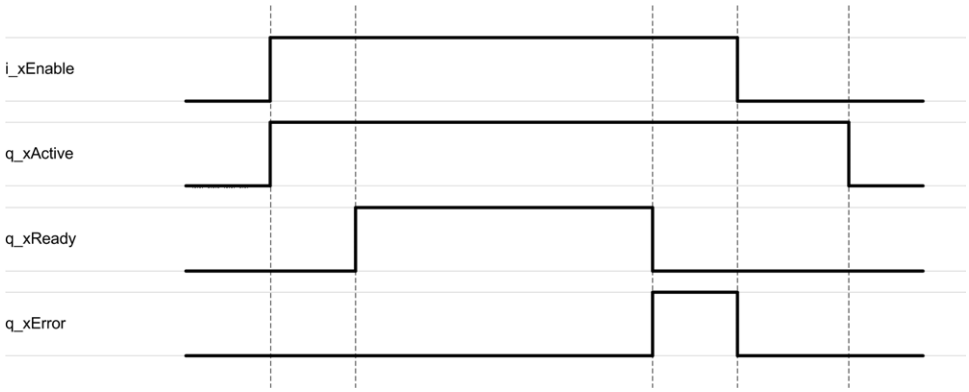
Behavior of Function Blocks with the Input *i_xEnable*

General Information

By setting the input *i_xEnable* to TRUE, the function block starts the enabling process. The function block continues initialization and the output *q_xActive* is set to TRUE. Once the initialization is finished, the output *q_xReady* is set to TRUE.

In case an error is detected, the output *q_xError* remains TRUE until the function block is disabled.

Example



Data Unit Types

What's in This Part

Enumerations 27
Structures 33
Aliases - *DateTimeArray* 55

Enumerations

What's in This Chapter

ET_Cmd 27
ET_Modes 28
ET_Result 28
ET_StateModelDefinition 30
ET_States 31

ET_Cmd

Overview

Type:	Enumeration
Available as of:	V1.0.1.0

Description

The enumeration includes the state commands used to proceed a state transition. A **Wait state**, page 19 can only be exited with a state transition command (for example *Start*).

Enumeration Elements

Name	Data type	Value	Description
Undefined	DINT	0	Commands to transition from one state to another are described in *FC_CheckCmd*.
Reset	DINT	1	
Start	DINT	2	
Stop	DINT	3	
Hold	DINT	4	
UnHold	DINT	5	
Suspend	DINT	6	
UnSuspend	DINT	7	
Abort	DINT	8	
Clear	DINT	9	

Used By

The values of the enumeration can be used as an input value for *i_diCmd* of *FC_CheckCmd* or to fill in the value for the variable *CntrlCmd* of *ST_Command*.

ET_Modes

Overview

Type:	Enumeration
Available as of:	V1.0.1.0

Description

The enumeration includes predefined operation modes (for example, production). An operation mode is an ordered subset of states, commands, and transitions which can be initialized via the structure *ST_UnitModeDefinition*.

Enumeration Elements

Name	Data type	Value	Description
Undefined	DINT	0	The operation mode is undefined or invalid.
Producing (1)	DINT	1	Operation mode for production.
Maintenance	DINT	2	Operation mode for maintenance.
Manual	DINT	3	Manual operation mode
(1) With the 2015 revision of the ANSI/ISA TR88.00.02-2015, the control mode *Producing* was changed to *Production*. For reasons of compatibility with the previous version of the PackML library, the name *Producing* has been maintained.			

Used By

The values of the enumeration can be used to fill values of the variables *UnitModeCurrent* of *ST_Status* and *UnitMode* of *ST_Command*.

ET_Result

Overview

Type:	Enumeration
Available as of:	V1.0.1.0

Description

The enumeration includes the possible values that indicate the result of operations executed by the function block.

Enumeration Elements

Name	Data type	Value	Description
If *q_xError* of a function block, page 74 is FALSE, one of the following status messages is shown.			
Ok	DINT	0	The POU is OK, but not running.
Disabled	DINT	1	The POU is disabled.
Initializing	DINT	2	The POU is being initialized.
Running	DINT	4	The POU is being executed.
ResettingTimerAndCounter	DINT	5	The POU is resetting the timer and counter.
If *q_xError* of a function block, page 74 is TRUE, one of the following error messages is shown.			
UnitModeCurrentRange	DINT	11	The value of the input *i_diUnitModeCurrent* is out of range. The value of operation mode must be in the range 0 and *Gc_uiMaxNumberOfModes*.
StateCurrentRange	DINT	12	The value of the input *i_diStateCurrent* is out of range. The value of state must be in the range 0 and *Gc_uiMaxNumberOfStates*.
NumberOfModesRange	DINT	13	Number of operation modes is out of range. The number of operation modes must be in the range 0 and *Gc_uiMaxNumberOfModes*.
PointerInitModeInvalid	DINT	14	The initial operation mode pointer *i_pstInitMode* is invalid.
PointerAdminInvalid	DINT	15	The admin pointer *i_pstAdmin* is invalid.
PointerStatusInvalid	DINT	16	The status pointer *i_pstStatus* is invalid.
UnitModeRange	DINT	17	The operation mode out of range. Range should be: 0 < *i_diUnitMode* <= *diNumberOfModes*
InvalidStateModelNoExecuteState	DINT	20	The state model of the operation mode is invalid: there is no **Execute** state.
InvalidStateModelNoStoppedState	DINT	21	The state model of the operation mode is invalid: there is no **Stopped** state.
InvalidStateModelNoHeldState	DINT	22	The state model of the operation mode is invalid: a **Held** state must exist if a **Holding** or an **Un-Holding** state is present.
InvalidStateModelNoCompleteState	DINT	23	The state model of the operation mode is invalid: if **Completing** state exists, a **Complete** state must be present.
InvalidStateModelNoSuspendedState	DINT	24	The state model of the operation mode is invalid: a **Suspended** state must exist if a **Suspending** or an **Un-Suspending** state is present.
InvalidStateModelNoAbortedState	DINT	25	The state model of the operation mode is invalid: an **Aborted** state must exist if an **Aborting** or a **Clearing** state is present.
InvalidStateModelNoIdleState	DINT	26	The state model of the operation mode is invalid: the **Resetting** state can be used only in connection with the **Idle** state.
InvalidStateModelNoResettingState	DINT	27	The state model of the operation mode is invalid: the **Idle** state can be used only in connection with the **Resetting** state.
ModeChangeRequestRejected	DINT	28	The request to change the operation mode is rejected.
StateChangeRequestRejected	DINT	29	The triggered state command is not accepted in the present state.
UnknownResult	DINT	30	The value of the input *i_etResult* of the function *FC_EtResultToString* is unknown.
InvalidInput	DINT	33	At least one of the specified inputs is invalid.
MaxNumberOfUnitModesExceeded	DINT	34	The maximum number of unit modes is defined. Remove a unit mode before adding a new one.

Name	Data type	Value	Description
NoUnitMode	DINT	35	No unit mode is active. The commanded operation is not allowed.
UnitModeNotFound	DINT	36	The specified unit mode is not defined.
UnitModeActive	DINT	37	Unit mode is active. The commanded operation is not allowed.
NoStateModelHandler	DINT	39	The definition of the unit mode requires an instance of the *FB_StateModelHandlerBase*.
RegisterLoggerPointFailed	DINT	41	The logger point could not be registered.
AlreadyExists	DINT	42	Unit mode with same value or name is already defined.
NotReady	DINT	43	The POU is blocked by another task and is not ready for execution.

Used By

The enumeration *ET_Result* is used by all POUs of this library.

ET_StateModelDefinition

Overview

Type:	Enumeration
Available as of:	V1.4.2.0

Description

This enumeration is used to define the state model of a unit mode. The states 1...17 of the PackML status model are represented by bits 1...17. State 0 is an undefined state, therefore bit 0 is not used.

Enumeration Elements

Name	Data type	Value	Description
AbortedAndStopped	DINT	000000001000000100 bin	The bits for the states *Aborted* and *Stopped* are TRUE. Can be used to define the states where a mode change is allowed.
NoStateDisabled	DINT	0 bin	None of the bits are TRUE, all states are enabled. Can be used to define the states which are disabled.
HoldDisabled	DINT	1110000000000 bin	The bits for the states *Holding*, *Held* and *UnHolding* are TRUE. Can be used to define the states which are disabled.
SuspendDisabled	DINT	110000000100000 bin	The bits for the states *Suspending*, *Suspended* and *UnSuspending* are TRUE. Can be used to define the states which are disabled.
CompleteDisabled	DINT	1100000000000000000 bin	The bits for the states *Completing* and *Complete* are TRUE. Can be used to define the states which are disabled.
Clearing	DINT	2 hex	Bit 1 represents the state *Clearing*.
Stopped	DINT	4 hex	Bit 2 represents the state *Stopped*.
Starting	DINT	8 hex	Bit 3 represents the state *Starting*.
Idle	DINT	10 hex	Bit 4represents the state *Idle*.
Suspended	DINT	20 hex	Bit 5 represents the state *Suspended*.
Execute	DINT	40 hex	Bit 6 represents the state *Execute*.
Stopping	DINT	50 hex	Bit 7 represents the state *Stopping*.
Aborting	DINT	100 hex	Bit 8 represents the state *Aborting*.
Aborted	DINT	200 hex	Bit 9 represents the state *Aborted*.
Holding	DINT	400 hex	Bit 10 represents the state *Holding*.
Held	DINT	800 hex	Bit 11 represents the state *Held*.
UnHolding	DINT	1000 hex	Bit 12 represents the state *UnHolding*.
Suspending	DINT	2000 hex	Bit 13 represents the state *Suspending*.
Unsuspending	DINT	4000 hex	Bit 14 represents the state *Unsuspending*.
Resetting	DINT	8000 hex	Bit 15 represents the state *Resetting*.
Completing	DINT	10000 hex	Bit 16 represents the state *Completing*.
Complete	DINT	20000 hex	Bit 17 represents the state *Complete*.

Used By

- *FB_UnitModeManager2*

ET_States

Overview

Type:	Enumeration
Available as of:	V1.0.1.0

Description

This enumeration represents the list of possible machine states as defined in ANSI/ISA TR88.00.02.

Enumeration Elements

NOTE: For a detailed description of each of the individual states, refer to the 2015 revision of the standard ANSI/ISA TR88.00.02.

Name	Data type	Value	Description
Undefined	DINT	0	No state is defined.
Clearing	DINT	1	**Acting state Clearing**
Stopped	DINT	2	**Wait state Stopped**
Starting	DINT	3	**Acting state Starting**
Idle	DINT	4	**Wait state Idle**
Suspended	DINT	5	**Wait state Suspended**
Execute	DINT	6	**Acting state Execute**
Stopping	DINT	7	**Acting state Stopping**
Aborting	DINT	8	**Acting state Aborting**
Aborted	DINT	9	**Wait state Aborted**
Holding	DINT	10	**Acting state Holding**
Held	DINT	11	**Wait state Held**
UnHolding	DINT	12	**Acting state Un-Holding**
Suspending	DINT	13	**Acting state Suspending**
UnSuspending	DINT	14	**Acting state Un-Suspending**
Resetting	DINT	15	**Acting state Resetting**
Completing	DINT	16	**Acting state Completing**
Complete (1)	DINT	17	**Wait state Complete**

NOTE: The contents of any given state is subject to your application.

(1) With the 2022 revision of the ANSI/ISA TR88.00.02, the control mode *Complete* was changed to *Completed*. For reasons of compatibility with the previous version of the PackML library, the name *Complete* has been maintained.

Used By

- *FB_ModeManager*
- *FB_UnitModeManager2*
- *FB_VisController*
- *i_diStateCurrent* of *FC_CheckCmd*
- *StateCurrent* of *ST_Status*
- *StateRequested* of *ST_Status*

Structures

What's in This Chapter

PackTags Main Structures 33
Other Main Structures 37
SubStructures 42

PackTags Main Structures

ST_Administration

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_Administration* represents the list of PackTags of type *Administration* as defined in ANSI/ISA TR88.00.02-2022.

Administration tags are used to describe the quality and alarm information of the unit machine.

Structure Elements

For the GPL data types, refer to Global Parameters, page 58.

Variable	Data type	Description
Alarm	ARRAY [1..*GPL.Gc_uiNumberOfAlarms]* OF *ST_Alarm*	Array of *ST_Alarm*
AlarmExtent	DINT	Size of the alarm array
AlarmHistory	ARRAY [1..*GPL.Gc_uiNumberOfAlarmHistory]* OF *ST_Alarm*	Array of given size for machine detected error code and messaging, page 42
AlarmHistoryExtent	DINT	Size of the alarm history array
Warning	ARRAY [1..*GPL.Gc_uiMaxNumberOfWarnings]* OF *ST_Alarm*	Array contains the active and/or unacknowledged machine advisories
WarningExtent	DINT	Size of the advisory array
MachDesignSpeed	REAL	Machine design speed
PLCDateTime	*DateTimeArray*	Present date and time of the programmable logic controller
StopReason(2)	ARRAY [1..*GPL.Gc_uiMaxNumberOfStopReasons]* OF *ST_Alarm*	Information about the first event captured during an abort, held, suspended, or stop event
StopReasonExtend(2)	*DINT*	Size of the *StopReason* array
Parameter(2)	ARRAY [1..*GPL.Gc_iNumOfAdminParameter]* OF *ST_Descriptor*	Array of *ST_Descriptor*
ModeCurrentTime(2)	ARRAY [0..*GPL.Gc_uiMaxNumberOfModes]* OF UDINT	Time spent in operation mode
ModeCumulativeTime(2)	ARRAY [0..*GPL.Gc_uiMaxNumberOfModes]* OF UDINT	Cumulative time in each operation mode

Variable	Data type	Description
StateCurrentTime(2)	ARRAY [0..*GPL.Gc_uiMaxNumberOfModes,0..GVL.Gc_uiMaxNumberOfStates]* OF UDINT	Value of last state timer in each operation mode/state
StateCumulativeTime(2)	ARRAY [0..*GPL.Gc_uiMaxNumberOfModes,0..GVL.Gc_uiMaxNumberOfStates]* OF UDINT	Cumulative time in each operation mode/state
ProdConsumedCount(2)	ARRAY [1..*GPL.Gc_uiNumberOfConsumedProducts]* OF *ST_CountDescrip*	Array of *ST_CountDescrip* (of products consumed in the production machine)
ProdProcessedCount(2)	ARRAY [1..*GPL.Gc_uiNumberOfProcessedProducts]* OF *ST_CountDescrip*	Array of *ST_CountDescrip* (of products processed by the production machine)
ProdDefectiveCount(2)	ARRAY [1..*GPL.Gc_uiNumberOfDefectiveProducts]* OF *ST_CountDescrip*	Array of *ST_CountDescrip* (of products that are marked as not acceptable in the production machine)
AccTimeSinceReset(2)	DINT	Accumulative time since last *Reset* command
StatesDisabled(2)	DINT	States 1...17 of the PackML State Model can be disabled by turning on the corresponding bit
PACDateTime(1) (2)	*ST_Timestamp*	Time stamp (ISO date and time data type, ISO 8601:1988 format), page 53
Parameter_REAL	ARRAY [1.. *GPL.Gc_uiNumberOfParameter]* OF *ST_Parameter_REAL*	Structured array of unit/machine parameter information for values with REAL data type
Parameter_STRING	ARRAY [1.. *GPL.Gc_uiNumberOfParameter]* OF *ST_Parameter_STRING*	Structured array of unit/machine parameter information for values with STRING data type
Parameter_LREAL	ARRAY [1.. *GPL.Gc_uiNumberOfParameter]* OF *ST_Parameter_LREAL*	Structured Array of unit/machine parameter information for values with LREAL data type
Parameter_DINT	ARRAY [1.. *GPL.Gc_uiNumberOfParameter]* OF *ST_Parameter_DINT*	Structured array of unit/machine parameter information for values with DINT data type
StopReason2	*ST_Alarm*	Information about the first event captured during an abort, held, suspended or stop event
ModeTimeCurrent	DINT	Present mode time
StateTimeCurrent	DINT	Present state time
CumulativeTimes	ARRAY [1.. *GPL.Gc_uiNumberOfCumulativeTimes]* OF *ST_Cumulative_Times*	Structured array of timer values
ProductData	ARRAY [1.. *GPL.Gc_uiNumberOfProcessedProducts]* OF *ST_Product_Data*	Structured array of product stream data
DisabledStatesCfg	ARRAY [1.. *GPL.Gc_uiMaxNumberOfModes]* OF DWORD	Array reflecting the disabled states configuration for defined modes
CurDisabledStates	DWORD	Reflects the disabled states for the present mode (*ST_Status.UnitModeCurrent*)
EnabledModesCfg	DWORD	Element reflecting the enabled modes configuration for the unit/machine
ModeTransitionCfg	ARRAY [1.. *GPL.Gc_uiMaxNumberOfModes]* OF DWORD	Array reflecting the mode transition configuration for the modes of the unit/machine

(1) Variable has been replaced by *PLCDateTime* in ANSI/ISA TR88.00.02-2015 but remains in this library for compatibility reasons.

(2) Variable kept for compatibility (according to ANSI/ISA TR88.00.02-2015).

Used By

- *FB_DataManagement*
- *FB_VisController*

ST_Command

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_Command* represents the list of PackTags of type *Command* as defined in ANSI/ISA TR88.00.02-2022.

Command tags are used to control the operation of the unit machine.

Structure Elements

For the GPL data types, refer to Global Parameters, page 58.

Variable	Data type	Description
UnitMode	DINT	The target operation mode (see *ET_Modes*)
UnitModeChangeRequest	BOOL	TRUE = request to change the machine operation mode
MachSpeed	REAL	Machine present speed
CntrlCmd	DINT	Value of the command that provides the state command to drive a state change in the base state model.
CmdChangeRequest	BOOL	TRUE = request to change the machine state
MaterialInterlocks(1)	ARRAY [0..31] OF BOOL	Indicates materials are ready for processing.
RemoteInterface (1)	ARRAY [1..*GPL.Gc_uiNumberOfInterface]* OF *ST_Interface*	Used for coordinating upstream or downstream machines in a cell of multiple unit machines
Parameter (1)	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_Descriptor*	Array of *ST_Descriptor*
Product (1)	ARRAY [1.. *GPL.Gc_uiNumberOfProducts]* OF *ST_Product*	Array of *ST_Product*
MaterialInterlock	DWORD	Materials ready
Parameter_REAL	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_Parameter_ REAL*	Structured array of unit/machine parameter information for values with REAL data type
Parameter_STRING	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_Parameter_ STRING*	Structured array of unit/machine parameter information for values with STRING data type
Parameter_LREAL	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_Parameter_ LREAL*	Structured array of unit/machine parameter information for values with LREAL data type
Parameter_DINT	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_Parameter_ DINT*	Structured array of unit/machine parameter information for values with DINT data type
SelectedRecipe	DINT	Used to designate which recipe should be run on the machine to produce the primary output product, according to a user-design recipe handling procedure
RecipeChangeRequest	BOOL	TRUE = request to change the recipe on the unit/ machine
Recipe	ARRAY [1.. *GPL.Gc_uiNumberOfRecipes]* OF *ST_Recipe*	Structured array of recipe Information
(1) Variable kept for compatibility (according toANSI/ISA TR88.00.02-2015).		

ST_Status

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_Status* represents the list of PackTags of type *Status* as defined in ANSI/ISA TR88.00.02-2022.

Status tags are used to describe the operation of the unit machine.

Structure Elements

For the GPL data types, refer to Global Parameters, page 58.

Variable	Data type	Description
UnitModeCurrent	DINT	Operation mode in use
UnitModeRequested	BOOL	Requested change of operation mode
UnitModeChangeInProcess	BOOL	Requested change of operation mode in process
StateCurrent	DINT	Machine present state
StateRequested	DINT	Requested state / target state
StateChangeInProgress	BOOL	Requested state transition in process
MachSpeed	REAL	PackML machine speed setpoint
CurMachSpeed	REAL	PackML machine present speed
MaterialInterlocks	ARRAY [0..31] OF BOOL;	Describes the status of the materials that are ready for processing.
EquipmentInterlock	*ST_EquipmentInterlock*	Refer to *ST_EquipmentInterlock*.
RemoteInterface(1)	ARRAY [1.. *GPL.Gc_ uiNumberOfInterface]* OF *ST_ Interface*	Used for coordinating upstream or downstream machines in a cell of multiple unit machines
Parameter(1)	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_ Descriptor*	Machine parameter array of *ST_Descriptor*
Product(1)	ARRAY [1.. *GPL.Gc_ uiNumberOfProducts]* OF *ST_ Product*	Array of *ST_Product*.
MaterialInterlock	DWORD	Materials ready
Parameter_REAL	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_ Parameter_REAL*	Structured of unit/machine parameter information for values with REAL data type
Parameter_STRING	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_ Parameter_STRING*	Structured of unit/machine parameter information for values with STRING data type
Parameter_LREAL	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_ Parameter_LREAL*	Structured of unit/machine parameter information for values with LREAL data type
Parameter_DINT	ARRAY [1.. *GPL.Gc_ uiNumberOfParameter]* OF *ST_ Parameter_DINT*	Structured of unit/machine parameter information for values with DINT data type
RecipeCurrent	DINT	The recipe currently running in production
RecipeRequested	DINT	A reflection of the recipe currently selected for production

Variable	Data type	Description
RecipeChangeInProcess	BOOL	An indicator for the user-defined recipe changeover process
Recipe	ARRAY [1.. *GPL.Gc_ uiNumberOfRecipes]* OF *ST_Recipe*	Structured of recipe information
Stacklight	ARRAY [1..31 OF *DINT*	Indicator for the stack light status
(1) Variable kept for compatibility (according toANSI/ISA TR88.00.02-2015).		

Used By

- *FB_VisController*

Other Main Structures

ST_InitAlarm

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_InitAlarm* contains the information about the alarm or advisory which is managed using the functions *FC_SetAlarm* or *FC_SetWarning*.

Structure Elements

Variable	Data type	Description
diID	DINT	Event message ID
diValue	DINT	Event message value
sMessage	STRING*[GPL. Gc_ uiMaxLengthOfAlarmMessage]*	Event message
xIsInList	BOOL	Event is in list
xIsAck	BOOL	Event is acknowledged
Category	DINT	Category of the event

Used By

- *FC_SetAlarm*
- *FC_SetWarning*

ST_UnitModeDefinition

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_UnitModeDefinition* contains the information to initialize the operation modes in the function block *FB_ModeManager*.

Structure Elements

Variable	Data type	Description
xModeExistent	BOOL	TRUE: operation mode is defined.
stClearing	*ST_StateInfo*	Indicates whether the state *stClearing* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stStopped	*ST_StateInfo*	Indicates whether the state *stStopped* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stStarting	*ST_StateInfo*	Indicates whether the state *stStarting* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stIdle	*ST_StateInfo*	Indicates whether the state *stIdle* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stResetting	*ST_StateInfo*	Indicates whether the state *stResetting* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stExecute	*ST_StateInfo*	Indicates whether the state *stExecute* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stStopping	*ST_StateInfo*	Indicates whether the state *stStopping* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stAborting	*ST_StateInfo*	Indicates whether the state *stAborting* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stAborted	*ST_StateInfo*	Indicates whether the state *stAborted* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stHolding	*ST_StateInfo*	Indicates whether the state *stHolding* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stHeld	*ST_StateInfo*	Indicates whether the state *stHeld* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stUnHolding	*ST_StateInfo*	Indicates whether the state *stUnHolding* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stSuspended	*ST_StateInfo*	Indicates whether the state *stSuspended* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stUnSuspending	*ST_StateInfo*	Indicates whether the state *stUnSuspending* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.

Variable	Data type	Description
stSuspending	*ST_StateInfo*	Indicates whether the state *stSuspending* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stCompleting	*ST_StateInfo*	Indicates whether the state *stCompleting* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
stComplete	*ST_StateInfo*	Indicates whether the state *stComplete* is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.

Used By

- *FB_ModeManager*
- *FB_VisController*
- *FC_InitStateModelChangeStates*

ST_VisInterface

Overview

Type:	Data structure
Available as of:	V1.0.1.0

Description

The structure *ST_VisInterface* forms the interface to the visualization screens provided with the PackML library.

Structure Elements

Variable	Data type	Description
vq_xBlink	BOOL	Variable used to flash the present **Acting state**
vq_diNumberOfModes	DINT	Maximum number of operation modes
vq_diUnitModeCurrent	DINT	Operation mode in use
vq_diStateCurrent	DINT	Present state
vq_diModeCurrentTimeActive	DINT	Present time in active operation mode
vq_diModeCumulativeTimeActive	DINT	Cumulative time in active operation mode
vq_diStateCurrentTimeActive	DINT	Present time in active state
vq_diStateCumulativetimeActive	DINT	Cumulative time in active state
vq_astAlarm	ARRAY [0..10] OF *ST_Alarm*	Array of given size for active machine error code and messaging
vq_astAlarmHistory	ARRAY [0..10] OF *ST_Alarm*	Array of given size for machine error code and messaging
vq_diProdConsumedId	DINT	ID of consumed product
vq_sProdConsumedName	STRING *[GPL.Gc_ uiMaxLengthOfParameter-Name]*	Name of consumed product
vq_sProdConsumedUnit	STRING [5]	Unit of consumed product

Variable	Data type	Description
vq_diProdConsumedCount	DINT	Count of consumed products
vq_diProdConsumedAccCount	DINT	Accumulative count of consumed products
vq_diProdProcessedId	DINT	ID of processed product
vq_sProdProcessedName	STRING *[GPL.Gc_ uiMaxLengthOfParameter-Name]*	Name of processed product
vq_sProdProcessedUnit	STRING [80]	Unit of processed product
vq_diProdProcessedCount	DINT	Count of processed products
vq_diProdProcessedAccCount	DINT	Accumulative count of processed products
vq_diProdDefectiveId	DINT	ID of not acceptable product
vq_sProdDefectiveName	STRING *[GPL.Gc_ uiMaxLengthOfParameter-Name]*	Name of not acceptable product
vq_sProdDefectiveUnit	STRING [5]	Unit of not acceptable product
vq_diProdDefectiveCount	DINT	Count of not acceptable products
vq_diProdDefectiveAccCount	DINT	Accumulative count of not acceptable products
vq_diModeCurrentTime	DINT	Present time in each operation mode
vq_diModeCumulativeTime	DINT	Cumulative time in each operation mode
vq_diStateCurrentTime	DINT	Present time in each state
vq_diStateCumulativetime	DINT	Cumulative time in each state
vq_diTimeSinceLastReset	DINT	Time since last *Reset* command
vq_diStateCurrentTimeUndefined	DINT	Present time in undefined state
vq_diStateCurrentTimeClearing	DINT	Present time in **Clearing** state
vq_diStateCurrentTimeStopped	DINT	Present time in **Stopped** state
vq_diStateCurrentTimeStarting	DINT	Present time in **Starting** state
vq_diStateCurrentTimeIdle	DINT	Present time in **Idle** state
vq_diStateCurrentTimeSuspended	DINT	Present time in **Suspended** state
vq_diStateCurrentTimeExecute	DINT	Present time in **Execute** state
vq_diStateCurrentTimeStopping	DINT	Present time in **Stopping** state
vq_diStateCurrentTimeAborting	DINT	Present time in **Aborting** state
vq_diStateCurrentTimeAborted	DINT	Present time in **Aborted** state
vq_diStateCurrentTimeHolding	DINT	Present time in **Holding** state
vq_diStateCurrentTimeHeld	DINT	Present time in **Held** state
vq_diStateCurrentTimeUnholding	DINT	Present time in **Un-Holding** state
vq_diStateCurrentTimeSuspending	DINT	Present time in **Suspending** state
vq_diStateCurrentTimeUnsuspending	DINT	Present time in **Un-Suspending** state
vq_diStateCurrentTimeResetting	DINT	Present time in **Resetting** state
vq_diStateCurrentTimeCompleting	DINT	Present time in **Completing** state
vq_diStateCurrentTimeComplete	DINT	Present time in **Complete** state
vq_diStateCumulativeTimeUndefined	DINT	Cumulative time in undefined state
vq_diStateCumulativeTimeClearing	DINT	Cumulative time in **Clearing** state
vq_diStateCumulativeTimeStopped	DINT	Cumulative time in **Stopped** state

Variable	Data type	Description
vq_diStateCumulativeTimeStarting	DINT	Cumulative time in **Starting** state
vq_diStateCumulativeTimeIdle	DINT	Cumulative time in **Idle** state
vq_diStateCumulativeTimeSuspended	DINT	Cumulative time in **Suspended** state
vq_diStateCumulativeTimeExecute	DINT	Cumulative time in **Execute** state
vq_diStateCumulativeTimeStopping	DINT	Cumulative time in **Stopping** state
vq_diStateCumulativeTimeAborting	DINT	Cumulative time in **Aborting** state
vq_diStateCumulativeTimeAborted	DINT	Cumulative time in **Aborted** state
vq_diStateCumulativeTimeHolding	DINT	Cumulative time in **Holding** state
vq_diStateCumulativeTimeHeld	DINT	Cumulative time in **Held** state
vq_diStateCumulativeTimeUnholding	DINT	Cumulative time in **Un-Holding** state
vq_diStateCumulativeTimeSuspending	DINT	Cumulative time in **Suspending** state
vq_diStateCumulativeTimeUnsuspending	DINT	Cumulative time in **Un-Suspending** state
vq_diStateCumulativeTimeResetting	DINT	Cumulative time in **Resetting** state
vq_diStateCumulativeTimeCompleting	DINT	Cumulative time in **Completing** state
vq_diStateCumulativeTimeComplete	DINT	Cumulative time in **Complete** state
vq_xStateClearingExistent	BOOL	TRUE: **Clearing** state exists
vq_xStateStoppedExistent	BOOL	TRUE: **Stopped** state exists
vq_xStateStartingExistent	BOOL	TRUE: **Starting** state exists
vq_xStateIdleExistent	BOOL	TRUE: **Idle** state exists
vq_xStateSuspendedExistent	BOOL	TRUE: **Suspended** state exists
vq_xStateExecuteExistent	BOOL	TRUE: **Execute** state exists
vq_xStateStoppingExistent	BOOL	TRUE: **Stopping** state exists
vq_xStateAbortingExistent	BOOL	TRUE: **Aborting** state exists
vq_xStateAbortedExistent	BOOL	TRUE: **Aborted** state exists
vq_xStateHoldingExistent	BOOL	TRUE: **Holding** state exists
vq_xStateHeldExistent	BOOL	TRUE: **Held** state exists
vq_xStateUnholdingExistent	BOOL	TRUE: **Un-Holding** state exists
vq_xStateSuspendingExistent	BOOL	TRUE: **Suspending** state exists
vq_xStateUnsuspendingExistent	BOOL	TRUE: **Un-Suspending** state exists
vq_xStateResettingExistent	BOOL	TRUE: **Resetting** state exists
vq_xStateCompletingExistent	BOOL	TRUE: **Completing** state exists
vq_xStateCompleteExistent	BOOL	TRUE: **Complete** state exists
viq_diIndexAlarm	DINT	Index for active alarm table
viq_diIndexAlarmHistory	DINT	Index for active alarm history table
viq_diIndexProdConsumed	DINT	Index of consumed product
viq_diIndexProdProcessed	DINT	Index of processed product
viq_diIndexProdDefective	DINT	Index of not acceptable product
viq_diIndexAdminParameter	DINT	Index of admin parameter
viq_diIndexMode	DINT	Index of operation mode
viq_diIndexWarning	DINT	Index for active advisory table
vq_astWarning	ARRAY [0..9] OF *ST_Alarm*	Array of given size for active machine advisory code and messaging

Variable	Data type	Description
vq_audiBgColorAlarm	ARRAY [0..10] OF UDINT	Variables to control the background color of the single alarms in display *FR_Alarm*.
vq_audiBgColorAlarmHistory	ARRAY [0..10] OF UDINT	Variables to control the background color of the single alarms in display *FR_AlarmHistory*.
vq_audiBgColorAlarmWarning	ARRAY [0..9] OF UDINT	Variables to control the background color of the single advisory in display *FR_Warning*.

Used By

- *FB_VisController*
- Visualization Frames

SubStructures

ST_Alarm

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_Alarm* is a collection of tags needed to describe an event (for example, alarms or advisories).

Structure Elements

Variable	Data type	Description
ID	DINT	Unique identifier of the event
Value	DINT	Value associated to the event for adding details or to break down the event ID to greater detail.
Message	STRING*[GPL.Gc_uiMaxLengthOfAlarmMessage]*	Message associated to the event
TimeEvent(1)	*ST_TimeStamp*	Time stamp (ISO data type) the event occurred
TimeAck(2)	*ST_TimeStamp*	Time stamp the event was acknowledged
Trigger	BOOL	TRUE: Indicates that the event is active.
Category	DINT	Category of the event
DateTime	*DateTimeArray*	Date and time when the event has occurred

Variable	Data type	Description
AckDateTime	*DateTimeArray*	Date and time when the event was acknowledged
(1) Variable has been replaced by *DateTime* in ANSI/ISA TR88.00.02-2015 but remains in this library for compatibility reasons. **(2)** Variable has been replaced by *AckDateTime* in ANSI/ISA TR88.00.02-2015 but remains in this library for compatibility reasons.		

Used By

- *FC_SetAlarm*
- *ST_Administration*
- *ST_VisInterface*

ST_CountDescrip

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_CountDescrip* is a collection of tags used to describe a counter value.

Structure Elements

Variable	Data type	Description
ID	DINT	ID of *CountDescrip*
Name	STRING*[GPL.Gc_ uiMaxLengthOfParameter-Name]*	Name of *CountDescrip*
Unit	STRING[5]	String of *CountDescrip* units
Count	DINT	Count value of *CountDescrip*
AccCount	DINT	Accumulative count value of *CountDescrip*

Used By

- *ST_Administration*

ST_Cumulative_Times

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_Cumulative_Times* is a collection of tags providing information for timer values.

Structure Elements

Variable	Data type	Description
AccTimeSinceReset	DINT	Accumulated time since last reset
ModeStateTimes	ARRAY [1..*Gc_uiMaxNumOfModes*] OF *ST_ModeState_Times*	Structured array of mode and state time values for each cumulative time tracker

Used By

- *ST_Administration*

ST_Descriptor

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_Descriptor* is the data type used for the parameter tags of the PackTags. It serves as descriptor of the respective parameter tag.

Structure Elements

Variable	Data type	Description
ID	DINT	ID of parameter
Name	STRING*[GPL.Gc_uiMaxLengthOfParameter-Name]*	Name of parameter
Unit	STRING[5]	String of parameter units
Value	REAL	Value of parameter

Used By

- *ST_Administration*
- *ST_Commmand*
- *ST_Status*
- *ST_Ingredient*
- *ST_Interface*
- *ST_Product*

ST_EquipmentInterlock

Overview

Type	Data structure
Available as of	V1.1.0.0

Description

The structure *ST_EquipmentInterlock* indicates whether the downstream and upstream system is the reason for the suspended state of a machine.

Structure Elements

Variable	Data type	Description
Blocked	BOOL	TRUE: Indicates that a downstream system is not able to accept product.
Starved	BOOL	TRUE: Indicates that an upstream system is not able to supply product

Used By

- *ST_Status*

ST_Ingredient

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_Ingredient* is a collection of tags used to describe the raw materials that are needed for the product.

Structure Elements

Variable	Data type	Description
IngredientID	DINT	Arbitrary number associated with the raw material, or ingredient for a particular product number.
Parameter	ARRAY [0..*Gc_uiNumberOfIngredientParameter*] OF *ST_Descriptor*	Structured array describes the specific ingredients or raw materials, which are needed or were used by the machine to process the specified product.

Used By

- *ST_Product*

ST_Ingredients

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_Ingredients* is a collection of tags providing information for the recipe ingredients.

Structure Elements

Variable	Data type	Description
Parameter_REAL	ARRAY [1.. *GPL.Gc_uiNumOfParamREALRecipeIngredients*] OF *ST_Parameter_REAL*	Structured array of ingredient information for values with REAL data type
Parameter_STRING	ARRAY [1.. *GPL.Gc_uiNumOfParamSTRINGRecipeIngredients*] OF *ST_Parameter_STRING*	Structured array of ingredient information for values with STRING data type
Parameter_LREAL	ARRAY [1.. *GPL.Gc_uiNumOfParamLREALRecipeIngredients*] OF *ST_Parameter_LREAL*	Structured array of ingredient information for values with LREAL data type
Parameter_DINT	ARRAY [1.. *GPL.Gc_uiNumOfParamDINTRecipeIngredients*] OF *ST_Parameter_DINT*	Structured array of ingredient information for values with DINT data type

Used By

- *ST_Recipe*

ST_Interface

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_Interface* is a collection of tags that are used to describe communication command values between machines using the PackTag structures.

Structure Elements

Variable	Data type	Description
Number	DINT	Identification number of upstream or downstream machine
ControlCmdNumber	DINT	Control command number for upstream or downstream machine
CmdValue	DINT	Command value associated with the control command number.
Parameter	ARRAY [0..*GPL.Gc_uiNumOfParam]* OF *ST_Descriptor*	Array of *ST_Descriptor*, parameter tags associated to the remote interface

Used By

- *ST_Commmand*
- *ST_Status*

ST_ModeState_Times

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_ModeState_Times* is a collection of tags providing information for the cumulative time tracker.

Structure Elements

Variable	Data type	Description
Mode	DINT	Represents the cumulative amount of time (in seconds) spent in each mode since the last timer and counter reset was executed
State	ARRAY [0..*GCL.Gc_uiMaxNumberOfStates*] OF DINT	Represents the cumulative amount of time (in seconds) spent in each state of a particular mode since the last timer and counter reset was executed

Used By

- *ST_Cumulative_Times*

ST_Parameter_DINT

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_Parameter_DINT* provides information for parameters or values with DINT data type.

Structure Elements

Variable	Data type	Description
ID	DINT	ID of the parameter
Name	STRING(GPL.Gc_ uiMaxLengthOfParameter-Name)	Name of the parameter
Unit	STRING[6]	Unit of the parameter
Value	REAL	Value of the parameter

Used By

- *ST_Administration*
- *ST_Command*
- *ST_Status*
- *ST_Ingredients*
- *ST_ProcessVariables*

ST_Parameter_LREAL

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_Parameter_LREAL* provides information for parameters or values with LREAL data type.

Structure Elements

Variable	Data type	Description
ID	DINT	ID of the parameter
Name	STRING(GPL.Gc_ uiMaxLengthOfParameter-Name)	Name of the parameter
Unit	STRING[6]	Unit of the parameter
Value	REAL	Value of the parameter

Used By

- *ST_Administration*

- *ST_Command*
- *ST_Status*
- *ST_Ingredients*
- *ST_ProcessVariables*

ST_Parameter_REAL

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_Parameter_REAL* provides information for parameters or values with REAL data type.

Structure Elements

Variable	Data type	Description
ID	DINT	ID of the parameter
Name	STRING(GPL.Gc_ uiMaxLengthOfParameter-Name)	Name of the parameter
Unit	STRING[6]	Unit of the parameter
Value	REAL	Value of the parameter

Used By

- *ST_Administration*
- *ST_Command*
- *ST_Status*
- *ST_Ingredients*
- *ST_ProcessVariables*

ST_Parameter_STRING

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_Parameter_STRING* provides information for parameters or values with STRING data type.

Structure Elements

Variable	Data type	Description
ID	DINT	ID of the parameter
Name	STRING(GPL.Gc_ uiMaxLengthOfParameter-Name)	Name of the parameter
Unit	STRING[6]	Unit of the parameter
Value	STRING	Value of the parameter

Used By

- *ST_Administration*
- *ST_Command*
- *ST_Status*
- *ST_Ingredients*
- *ST_ProcessVariables*

ST_ProcessVariables

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_ProcessVariables* is a collection of tags providing information for the recipe process variables.

Structure Elements

Variable	Data type	Description
Parameter_REAL	ARRAY [1.. *GPL. Gc_ uiNumOfParamREALReci-peProcessVar*] OF *ST_ Parameter_REAL*	Structured array of recipe process variable information for values with REAL data type
Parameter_STRING	ARRAY [1.. *GPL. Gc_ uiNumOfParamSTRING-RecipeProcessVar*] OF *ST_Parameter_STRING*	Structured array of recipe process variable information for values with STRING data type
Parameter_LREAL	ARRAY [1.. *GPL. Gc_ uiNumOfParamLREALRe-cipeProcessVar*] OF *ST_ Parameter_LREAL*	Structured array of recipe process variable information for values with LREAL data type
Parameter_DINT	ARRAY [1.. *GPL. Gc_ uiNumOfParamDINTReci-peProcessVar*] OF *ST_ Parameter_DINT*	Structured array of recipe process variable information for values with DINT data type

Used By

- *ST_Recipe*

ST_Product

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_Product* is a collection of tags used to describe the product that the machine is making.

Structure Elements

Variable	Data type	Description
ProductID	DINT	ID of the product, used to indicate to the machine which product it is producing.
ProcessVariable	ARRAY [0..*GPL.Gc_uiNumberOfProcessVariable]* OF *ST_Descriptor*	Structured array describes the specific process variables, which are needed or were used by the machine to process the specified product.
Ingredients	ARRAY [0..*GPL.Gc_uiNumberOfIngredients]* OF *ST_Ingredient*	Structured array, holds the information about the raw materials, which are needed or were used by the machine to process the specified product.

Used By

- *ST_Commmand*
- *ST_Status*

ST_Product_Data

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_Product_Data* is a collection of tags providing information for product stream data.

Structure Elements

Variable	Data type	Description
ID	DINT	ID number for each product stream
Name	STRING	Name of product stream
Unit	STRING[6]	Structured array of ingredients process variable information for values with LREAL data type
PrimaryQty	REAL	Product primary quantity value of each product stream
ConsumedCount	DINT	Number of consumables used of each product stream
ProcessedCount	DINT	Number of produced goods of each product stream
DefectiveCount	DINT	Number of products marked as not acceptable of each product stream
AccConsumedCount	DINT	Accumulated number of consumables used of each product stream since last reset
AccProcessedCount	DINT	Accumulated number of produced goods of each product stream since last reset
AccDefectiveCount	DINT	Accumulated number of products marked as not acceptable of each product stream since last reset

Used By

- *ST_Administration*

ST_Recipe

Overview

Type	Data structure
Available as of	V1.4.2.0

Description

The structure *ST_Recipe* is a collection of tags providing recipe information

Structure Elements

Variable	Data type	Description
ID	DINT	ID of *ST_Recipe*
Name	STRING	Name of *ST_Recipe*
Unit	STRING[6]	Unit of *ST_Recipe*
PrimaryQty	REAL	Primary quantity value of the recipe
ProcessVariables	*ST_ProcessVariables*	Process variables for the recipe
Ingredients	*ST_Ingredients*	Ingredient information for the recipe

Used By

- *ST_Command*
- *ST_Status*

ST_StateInfo

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_StateInfo* indicates whether the state is defined for the given operation mode and whether a change of operation mode is possible. The information is used to initialize the operation mode manager.

Structure Elements

Variable	Data type	Description
xStateExistent	BOOL	TRUE: Indicates that the state is defined for the given operation mode.
xModeChangePossible	BOOL	TRUE: indicates that a change of operation mode is possible while in this state.

Used By

- *ST_UnitModeDefinition*

ST_Timestamp

Overview

Type	Data structure
Available as of	V1.0.1.0

Description

The structure *ST_TimeStamp* contains the date and time.

NOTE: The structure has become obsolete with the 2015 revision of ANSI/ISA TR88.00.02. However, it is kept in the library in order to stay compatible with applications which have been created with the earlier versions of this library.

Structure Elements

Variable	Data type	Description
AlmDate	DATE	Contains the timespan in seconds since 01.01.1970, 00:00 clock.
AlmTime	TOD	Contains the timespan in milliseconds since 00:00 clock.

Used By

- *ST_Administration*
- *ST_Alarm*

Aliases - *DateTimeArray*

What's in This Chapter

DateTimeArray 55

DateTimeArray

Overview

Type	Data structure
Available as of	V1.1.0.0

Description

The alias DateTimeArray represents a complex data type which is used in this library.

Structure Elements

Variable	Data type	Data type
DateTimeArray	ARRAY [0..6] OF DINT	The data elements associated with date and time shall be in the format per ISO 8601. [0] = year, [1] = month, [2] = day, [3] = hour, [4] = min, [5] = sec, [6] = µs (fraction of seconds)

Global Variables

What’s in This Part

Global Constants List 57
Global Parameter List 58
Global Variable List 61

Global Constants List

What's in This Chapter

GCL 57

GCL

Overview

Type:	Global constants
Available as of:	V1.0.1.0

Description

The global constants list contains the global constants of the PackML library.

Global Constants

Variable	Data type	Value	Description
Gc_sLibraryVersion	STRING[80]	Vx.x.x.0[1]	Library version
Gc_uiMaxNumberOfStates	UINT	17	Maximum number of states
[1] This value varies to indicates the version of the library.			

Global Parameter List

What's in This Chapter

GPL 58

GPL

Overview

Type:	Global parameters
Available as of:	V1.0.1.0

Description

The global parameter list contains the global parameters of the PackML library which can be overwritten project-specifically in the library manager.

Global Parameters

Variable	Data type	Value	Description
Gc_uiNumberOfInterface	UINT[1..65535]	2	Number of remote interfaces, determines the array *ST_Command.RemoteInterface*
Gc_uiNumOfParam	UINT[1..65535]	10	Number of parameters for an interface, determines the array *ST_Interface.Parameter*
Gc_uiNumOfAdminParameter	UINT[1..65535]	100	Number of parameters for the local interface, determines the array *ST_Administration.Parameter*.
Gc_uiNumberOfParameter	UINT[1..65535]	50	Number of parameters in status and command structure, determines the arrays *ST_Status.Parameter* and *ST_Command.Parameter*
Gc_uiNumberOfProducts	UINT[1..65535]	3	Number of products which can be processed by the machine, determines the arrays *ST_Status.Product* and *ST_Command.Product*
Gc_uiNumberOfConsumedProducts	UINT[1..65535]	10	Number of products consumed by the machine, determines the array *ST_Administration.Prod ConsumedCount*
Gc_uiNumberOfProcessedProducts	UINT[1..65535]	10	Number of products processed by the machine, determines the array *ST_Administration.Prod ProcessedCount*
Gc_uiNumberOfDefectiveProducts	UINT[1..65535]	10	Number of products that are marked as not acceptable in the production machine, determines the array *ST_Administration.Prod DefectiveCount*
Gc_uiNumberOfProcessVariable	UINT[1..65535]	50	Number of process variables per product, determines the array *ST_Product.ProcessVariable*

Variable	Data type	Value	Description
Gc_uiNumberOfIngredientParameter	UINT[1..65535]	50	Number of parameters per ingredient, determines the array *ST_Ingredients. Parameter*
Gc_uiNumberOfIngredients	UINT[1..65535]	10	Number of ingredients per product, determines the array *ST_Product.Ingredients*
Gc_uiMaxNumberOfAlarms	UINT[1..65535]	100	Maximum number of alarms, determines the array *ST_ Administration.Alarm*
Gc_uiNumberOfAlarmHistory	UINT[1..65535]	100	Maximum number of alarms in history, determines the array *ST_Administration. AlarmHistory*
Gc_uiMaxNumberOfModes	UINT[1..65535]	31	Maximum number of modes
Gc_uiMaxLengthOfParameterName	UINT[1..80]	80	Maximum length of parameter name in *ST_Descriptor*
Gc_uiMaxNumberOfWarnings	UINT[1..65535]	100	Maximum number of advisories, determines the array *ST_Administration. Warning*
Gc_uiMaxLengthOfAlarmMessage	UINT[1..255]	80	Maximum length of an alarm message specified with *ST_ Alarm*
Gc_udiBgColorAlarmActive	UDINT	FFCFCF hex	Used for visualization, RGB color code for active alarms, FFCFCF hex = light red
Gc_udiBgColorAlarmInactive	UDINT	FFFF80 hex	Used for visualization, RGB color code for inactive and not acknowledged alarms, FFFF80 hex = light yellow
Gc_udiBgColorAlarmAcknowledged	UDINT	AAD7FF hex	Used for visualization, RGB color code for inactive and acknowledged alarms, AAD7FF hex = light blue
Gc_udiBgColorAlarmHistoryUnacknowledged	UDINT	FFFF80 hex	Used for visualization, RGB color code for unacknowledged alarms in history, FFFF80 hex = light yellow
Gc_udiBgColorWarningActive	UDINT	FFCFCF hex	Used for visualization, RGB color code for active advisories, FFCFCF hex = light red
Gc_udiBgColorWarningInactive	UDINT	FFFF80 hex	Used for visualization, RGB color code for inactive and not acknowledged advisories, FFFF80 hex = light yellow
Gc_udiBgColorWarningAcknowledged	UDINT	AAD7FF hex	Used for visualization, RGB color code for inactive and acknowledged advisories, AAD7FF hex = light blue
Gc_uiNumberOfParameterREAL	UINT	10	Number of parameters of type REAL
Gc_uiNumberOfParameterSTRING	UINT	10	Number of parameters of type STRING
Gc_uiNumberOfParameterLREAL	UINT	10	Number of parameters of type LREAL
Gc_uiNumberOfParameterDINT	UINT	10	Number of parameters of type DINT
Gc_uiNumberOfRecipe	UINT	3	Number of recipes
Gc_uiNumberOfCumulativeTimes	UINT	10	Number of cumulative times

Variable	Data type	Value	Description
Gc_uiNumOfParamREALRecipeProcessVar	UINT	10	Number of parameters of type REAL for recipe process variables
Gc_uiNumOfParamSTRINGRecipeProcessVar	UINT	10	Number of parameters of type STRING for recipe process variables
Gc_uiNumOfParamLREALRecipeProcessVar	UINT	10	Number of parameters of type LREAL for recipe process variables
Gc_uiNumOfParamDINTRecipeProcessVar	UINT	10	Number of parameters of type DINT for recipe process variables
Gc_uiNumOfParamREALRecipeIngredients	UINT	10	Number of parameters of type REAL for recipe ingredients
Gc_uiNumOfParamSTRINGRecipeIngredients	UINT	10	Number of parameters of type STRING for recipe ingredients
Gc_uiNumOfParamLREALRecipeIngredients	UINT	10	Number of parameters of type LREAL for recipe ingredients
Gc_uiNumOfParamDINTRecipeIngredients	UINT	10	Number of parameters of type DINT for recipe ingredients

Global Variable List

What's in This Chapter

GVL_VisuParam 61

GVL_VisuParam

Overview

Type:	Global variables
Available as of:	V1.4.7.0

Description

The global variables list contains the global variables of the library.

Global Variables

Variable	Data Type	Default value	Description
`G_dwFillColorActingStateInactive`	DWORD	FFE1F5E0 hex	Defines the fill color of the boxes representing the acting states in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.
`G_dwTextColorActingStateInactive`	DWORD	FF626469 hex	Defines the text color of the boxes representing the acting states in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.
`G_dwFillColorWaitingStateInactive`	DWORD	FFFFFFCC hex	Defines the fill color of the boxes representing the waiting states in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.
`G_dwTextColorWaitingStateInactive`	DWORD	FF626469 hex	Defines the text color of the boxes representing the waiting states in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.
`G_dwFillColorExecuteStateInactive`	DWORD	FFCCE5FF hex	Defines the fill color of the boxes representing the execute state in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.
`G_dwTextColorExecuteStateInactive`	DWORD	FF626469 hex	Defines the text color of the boxes representing the execute states in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.
`G_dwFillColorStateActive`	DWORD	FFFF00FF hex	Defines the fill color of the boxes representing the active states in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.
`G_dwTextColorStateActive`	DWORD	FFFFFFFF hex	Defines the text color of the boxes representing the active states in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.
`G_dwFrameColorStateActive`	DWORD	FFFF0080 hex	Defines the frame color representing the active state in the visualization frames *FR_PackMLStateModel_2022_Base* and *FR_PackMLStateModel_2022_UnitModeManager*.

Interfaces

What's in This Part

IF_UnitMode 63
IF_StateCommands 64
IF_StateModelHandler 72

IF_UnitMode

What's in This Chapter

IF_UnitMode – General Information 63

IF_UnitMode – General Information

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Task

Interface for retrieving information about the present unit control mode.

Description

The interface provides information about the present unit control mode.

Properties

Name	Data type	Accessing	Description
sName	STRING	Read	Indicates the name of the present unit mode.
diValue	DINT	Read	Indicates the numeric value of the present unit mode.
diStatesDisabled	DINT	Read	Provides information about the state model. The bits 0...16 of the DINT represent the individual states as defined in *ET_StateModelDefinition*. The bit value TRUE indicates that the corresponding state is disabled in the state model.
diStatesMode-ChangeAllowed	DINT	Read	Provides information about the state model. The bits 0...16 of the DINT represent the individual states as defined in *ET_StateModelDefinition*. The bit value TRUE indicates that a mode change in the corresponding state is allowed.
etInitialState	*ET_States*, page 31	Read	Indicates the initial state for the configured state model.

IF_StateCommands

What's in This Chapter

IF_StateCommands – General Information 64
CmdReset (Method) 64
CmdStart (Method) 65
CmdStop (Method) 66
CmdHold (Method) 66
CmdUnHold (Method) 67
CmdSuspend (Method) 68
CmdUnSuspend (Method) 68
CmdAbort (Method) 69
CmdClear (Method) 70
StateComplete (Method) 70

IF_StateCommands – General Information

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Task

Interface for retrieving the state commands and state transitions.

Description

This interface provides the state commands and state transitions for the PackML base state model.

The interface is implemented by the function block *FB_UnitModeManager2*.

CmdReset (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *Reset*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdReset	BOOL	Indicates TRUE if the method is completed successfully.

CmdStart (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *Start*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdStart	BOOL	Indicates TRUE if the method is completed successfully.

CmdStop (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *Stop*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdStop	BOOL	Indicates TRUE if the method is completed successfully.

CmdHold (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *Hold*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdHold	BOOL	Indicates TRUE if the method is completed successfully.

CmdUnHold (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *UnHold*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdUnHold	BOOL	Indicates TRUE if the method is completed successfully.

CmdSuspend (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *Suspend*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdSuspend	BOOL	Indicates TRUE if the method is completed successfully.

CmdUnSuspend (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *UnSuspend*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdUnSuspend	BOOL	Indicates TRUE if the method is completed successfully.

CmdAbort (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *Abort*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdAbort	BOOL	Indicates TRUE if the method is completed successfully.

CmdClear (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to trigger the PackML state command *Clear*. Refer to State Commands, page 79.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdClear	BOOL	Indicates TRUE if the method is completed successfully.

StateComplete (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to perform the state transition form an acting state to the successor state according to the state model.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
StateComplete	BOOL	Indicates TRUE if the method is completed successfully.

IF_StateModelHandler

What's in This Chapter

IF_StateModelHandler – General Information 72

IF_StateModelHandler – General Information

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The interface is implemented by the function block *FB_StateModelHandlerBase*.

Program Organization Units (POU)

What’s in This Part

Function Blocks.....74
Functions.....101

Function Blocks

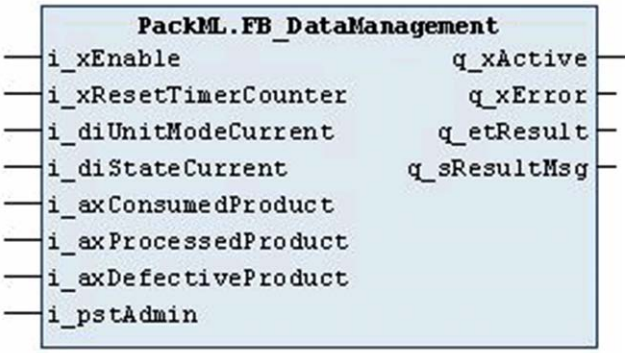
What's in This Chapter

FB_DataManagement 74
FB_ModeManager 76
FB_UnitModeManager2 79
FB_StateModelHandlerBase 90
FB_VisController 99

FB_DataManagement

Overview

Type:	Function block
Available as of:	V1.0.1.0



Functional Description

The function block *FB_DataManagement* is used to record certain machine data and to update the associated tags within the *ST_Administration*. The global structure of type *ST_Administration* is passed to the function block through the input *i_pstAdmin*.

Depending on the present unit mode (*i_diUnitModeCurrent*) and the machine state (*i_diStateCurrent*), the corresponding timer values are continuously updated. The values for control mode and machine state correspond to the definitions in the enumerations ET_Modes, page 28 and ET_States, page 31.

The different product counters are triggered by the corresponding signals within the arrays which are provided by the inputs *i_ax…Product*.

A global reset of the timer and counter values can be triggered through the input *i_xResetTimerCounter*.

> **NOTE:** The counter values provided by the tag *AccCount* within the structure *ST_CountDescrip* are given as non-resetting counters, hence they are not reset by this function block.

The retaining of data across external events, for example, a warm start of the controller, is not covered by this function block.

The following administration tags are processed by the function block:

- *ST_Administration.ModeCurrentTime[#]*
- *ST_Administration.ModeCummulativeTime[#]*
- *ST_Administration.StateCurrentTime[#,#]*
- *ST_Administration.StateCummulativeTime[#,#]*

- *ST_Administration.ProdConsumedCount[#]*
- *ST_Administration.ProdProcessedCount[#]*
- *ST_Administration.ProdDefectiveCount[#]*
- *ST_Administration.AccTimeSinceReset*
- *ST_Administration.AlarmExtent*
- *ST_Administration.AlarmHistoryExtent*
- *ST_Administration.WarningExtent*
- *ST_Administration.PLCDateTime*
- *ST_Administration.PACDateTime*
- *ST_Adminsitration.ModeTimeCurrent*
- *ST_Adminsitration.StateTimeCurrent*

Interface

Input	Data type	Description
i_xEnable	BOOL	Activation and initialization of the function block.
i_xResetTimerCounter	BOOL	Upon a rising edge, the timer and counter values are reset.
i_diUnitModeCurrent	DINT	The present operation mode is passed to the function block via this input. *Status.UnitModeCurrent* should be applied to the input.
i_diStateCurrent	DINT	The present machine state is passed to the function block via this input. *Status.StateCurrent* should be applied to the input.
i_axConsumedProduct	ARRAY [1..*GPL.Gc_uiNumberOfConsumedProducts]* OF BOOL	Upon a rising edge of a boolean variable in the array, the corresponding counter value in the administration tag is increased.
i_axProcessedProduct	ARRAY [1..*GPL.Gc_uiNumberOfConsumedProducts]* OF BOOL	Upon a rising edge of a boolean variable in the array, the corresponding counter value in the administration tag is increased.
i_axDefectiveProduct	ARRAY [1..*GPL.Gc_uiNumberOfDefectiveProducts]* OF BOOL	Upon a rising edge of a boolean variable in the array, the corresponding counter value in the administration tag is increased.
i_pstAdmin	POINTER TO *ST_Administration*	Through this input, the pointer address to the administration tags is passed to the function block.

To prevent access violation eventually caused by pointer access to the memory, make sure that the pointer points to a variable of type *ST_Administration*.

Executing the command **Online Change** can move variables to another place in the memory. The shift of variables may have the effect that `POINTER` variables point to invalid memory. So, ensure that a pointer is not kept between cycles, but is reassigned in each cycle.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

Assign the value of any `POINTER TO` type variable(s) prior to the first use of it within a function block, and at every subsequent cycle.

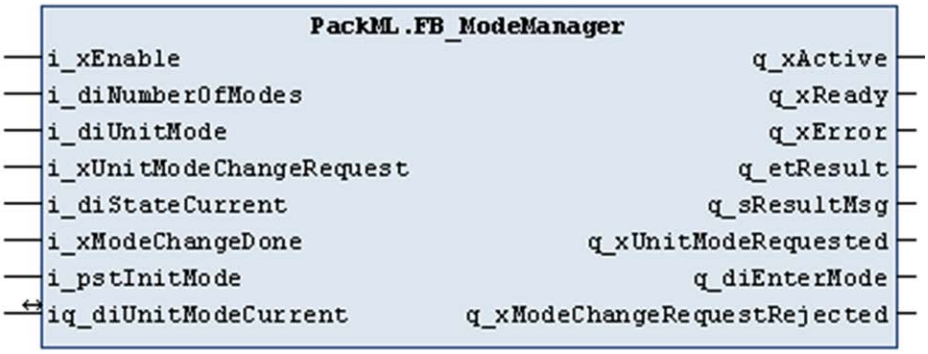
Failure to follow these instructions can result in death, serious injury, or equipment damage.

Output	Data type	Description
q_xActive	BOOL	If this output is set to TRUE, the function block is active.
q_xError	BOOL	If this output is set to TRUE, an error was detected. Refer to *ET_Result*.
q_etResult	*ET_Result*	Enumeration with the result.
q_sResultMsg	STRING[80]	Additional result message.

FB_ModeManager

Overview

Type:	Function block
Available as of:	V1.0.1.0



Functional Description

The function block *FB_ModeManager* can be used to manage mode change requests in a machine application, which is based on the PackML state model, page 20. Based on the unit mode definition the function block verifies whether the requested mode change is possible or not.

The unit mode definition comprises the definition of the possible states for each available control mode and the definition, from which state a mode change is permitted. For executing the unit mode definition, the auxiliary functions *FC_InitStateModelStateChange* and *FC_InitStateModelExistingStates* can be used.

Finally, the unit mode definition must be provided in an array of type *ST_UnitModeDefinition*. The indexes of the array correspond to the numeric value of the available control modes (refer to *ET_Modes*).

The unit mode definition must be made at start (initialization phase) of an application, prior to the first call with *i_xEnable* = *TRUE* of the instance of the function block *FB_ModeManager*.

Upon a rising edge of the input *i_xEnable*, the initialization phase of the function block *FB_ModeManager* is executed. During this phase, the unit state definitions for each available control mode are verified whether a PackML compliant state model can be created. If the initialization is completed successfully, the function block is ready for operation, which is indicated by the output *q_xReady*.

During initialization, the following conditions are verified:

- The maximum number of states is 17.
- The states **Execute** and **Stopped** are obligatory states for each control mode.
- The state **Completing** requires the state **Complete** too.
- The states **Holding** and **Unholding** require in each case the state **Held** too.
- The states **Suspending** and **UnSuspending** require in each case the state **Suspended** too.
- The states **Aborting** and **Clearing** require in each case the state **Aborted** too.
- The states **Resetting** and **Idle** can be used only in combination.

During normal operation, the function block verifies the mode change request, which is indicated by the inputs *i_diUnitMode* and *i_xUnitModeChange*, in relation to the present control mode (*iq_diUnitModeCurrent*) and the present state (*i_diStateCurrent*). Exclusively a rising edge on *i_sUnitModeChange* triggers an execution of the function block. The result is indicated by the outputs *q_diEnterMode*, and *q_xModeChangeRequestRejected*. If the mode change is permitted, the output *q_xModeChangeRequestRejected* indicates FALSE and the output *q_diEnterMode* is updated with the value of the requested mode. If so, the required steps to change the control mode must be executed depending on your application. Afterwards the information that the mode change was executed must be passed to the function block through the input *i_xModeChangeDone*. If this input indicates TRUE, the variable linked to the input / output *iq_diUnitModeCurrent* is updated with the previously requested control mode.

Interface

Input	Data type	Description
i_xEnable	BOOL	Activation and initialization of the function block.
i_diNumberOfModes	DINT	Number of operation modes If the value is changed, a reinitialization of the function block is required.
i_diUnitMode	DINT	Requested operation mode The PackTag *Command.UnitMode* should be applied to the input.
i_xUnitModeChangeRequest	BOOL	Upon a rising edge, the function block verifies whether a change of operation mode is possible. The PackTag *Command.UnitModeChangeRequest* should be applied to the input.
i_diStateCurrent	DINT	Machine present state The PackTag *Status.StateCurrent* should be applied to the input.

Input	Data type	Description
i_xModeChangeDone	BOOL	Feedback from the application that the requested mode change has been executed. Upon a rising edge, the variable linked to *iq_diUnitModeCurrent* is updated accordingly.
i_pstInitMode	POINTER TO *ST_UnitModeDefinition*	Through this input, the pointer address to the unit mode definitions is passed to the function block.* * The unit mode definition provides the state definitions for each available operation mode and must be provided in an array of *ST_UnitModeDefinition*. The indexes of the array correspond to the numeric value of the available control modes (refer to *ET_Modes*). Therefore the pointer must point to the index which is associated to the operation mode *Producing*, which is the first mode.

Input / output	Data type	Description
iq_diUnitModeCurrent	DINT	Through the variable linked to this input / output, the function block obtains the present operation mode on a change request and it writes the new operation mode after the change has been performed. The PackTag *Status.UnitModeCurrent* should be applied to the input/output.

Output	Data type	Description
q_xActive	BOOL	If this output is set to TRUE, the function block is active.
q_xError	BOOL	If this output is set to TRUE, an error has been detected. Refer to *ET_Result*.
q_xReady	BOOL	If this output is set to TRUE, the function block is ready for operation.
q_etResult	*ET_Result*	Enumeration with the result.
q_sResultMsg	STRING[80]	Additional result message.
q_xUnitModeRequested	BOOL	This reflects the input *i_xUnitModeChangeRequest*. The PackTag *Status.UnitModeRequested* should be applied to this output.
q_diEnterMode	DINT	This indicates the operation mode which is to be changed to. If the function block is deactivated, the output is set to 0.
q_xModeChangeRequestRejected	BOOL	As long as this output indicates TRUE, a mode change request is not authorized by this function block and should not be performed.

FB_UnitModeManager2

FB_UnitModeManager2 – General Information

Overview

Type:	Function block
Available as of:	V1.4.2.0

Functional Description

The *FB_UnitModeManager2* function block is used to implement various unit control modes in your application. The function block processes the commands for unit mode switching and the commands for controlling the state model of the present unit control mode.

The commands are triggered by the methods provided by the block.

When a mode switch is requested, it is verified whether switching of the unit control mode is allowed in the present state. If mode switching is allowed, the requested unit control mode is set, otherwise it remains in the present mode and a corresponding error message is issued.

The state commands are used to perform the state transitions of the active state model. If a command is triggered, it is verified whether the command is permissible in the present state according to the configured state model. If the command is permissible, it is accepted and the resulting state is set. If the command is not permissible, a corresponding error message is issued.

The individual unit control modes must be specified using the *DefineUnitMode()* method. The *FB_UnitModeManager2* function block supports up to 31 unit modes.

Mode Change

The following preconditions must be fulfilled to change the unit control mode:

- The desired unit control mode is known to the *FB_UnitModeManager2*. A unit control mode is identified by its numeric value.
- Exiting the present unit mode is allowed in the present state.
- The present state also exists in the status model of the target unit control mode. Unit mode transitions takes place only in a state that is common in both unit modes, the present and the target mode.

State Commands

The provided methods for triggering the state commands or the state transitions correspond to the PackML base state model as defined in ANSI/ISA TR88.00.02 – 2022. Refer to PackML Base State Model, page 80.

The *StateComplete()* method is used to carry out the state transition from the present acting state to the follow state according to the defined state model. Refer to *StateComplete* (Method), page 70.

The methods *Cmd<command name>()* are used to trigger the corresponding state command. Depending on the present state and the configured state model the state transition is carried out or the command is rejected. Refer to *IF_StateCommands* – General Information, page 64.

Supported state commands and their reactions

Command method	Description
CmdReset()	Command accepted only in states *Stopped* and *Complete*. Reaction: Transition to the state *Resetting*.
CmdStart()	Command accepted only in states *Idle* or in state *Stopped* if the state *Idle* does not exist. Reaction: Transition to the state *Starting*.
CmdHold()	Command accepted only in states *Execute* or *Suspended*. Reaction: Transition to the state *Holding*.
CmdUnHold()	Command accepted only in state *Held*. Reaction: Transition to the state *Unholding*.
CmdSuspend()	Command accepted only in state *Execute*. Reaction: Transition to the state *Suspending*.
CmdUnSuspend()	Command accepted only in state *Suspended*. Reaction: Transition to the state *Unsuspending*.
CmdComplete()	Command accepted only in states *Execute*, *Held*, or *Suspended*. Reaction: Transition to the state *Completing*.
CmdAbort()	Command accepted in all states except *Aborting* and *Aborted*. Reaction: Transition to the state *Aborting*.
CmdClear()	Command accepted only in state *Aborted*. Reaction: Transition to the state *Clearing*.
CmdStop()	Command accepted in all states except *Aborting*, *Aborted*, *Clearing*, *Stopping* and *Stopped*. Reaction: Transition to the state *Stopping*.
StateComplete()	Command accepted only in states *Resetting*, *Starting*, *Holding*, *Unholding*, *Suspending*, *Unsuspending*, *Aborting*, *Clearing*, or *Stopping*. Reaction: Transition to the next state in accordance with the state model.

Multitask Applications

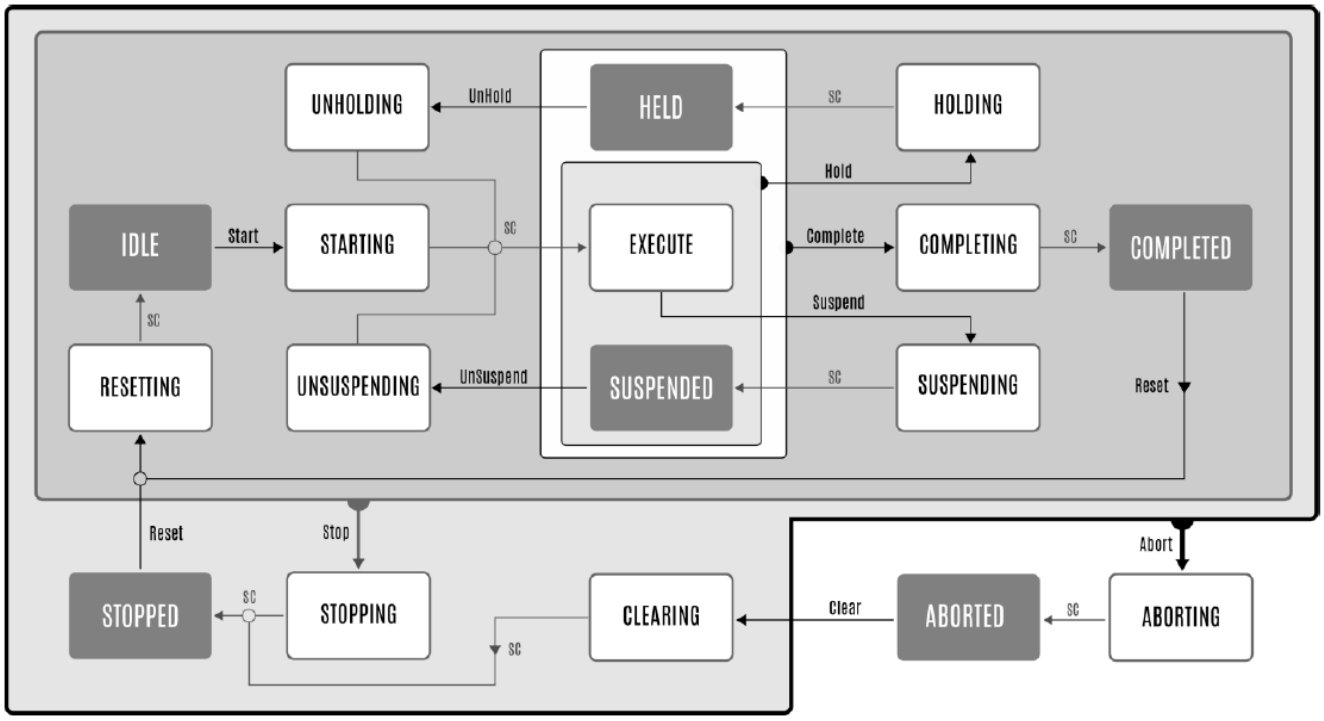
Due to the implementation using methods and properties it is possible that the individual methods are called by different tasks. To ensure complete processing of a command, the corresponding methods are implemented in such a way that a command is only accepted if no other command is in progress. If a command is rejected due to active processing, a corresponding diagnostic message is returned by the method.

However, you must consider the behavior of multitask applications when implementing your application.

PackML Base State Model

Overview

The following diagram illustrates the PackML base state model as per ANSI/ISA TR88.00.02 – 2022. The base state model represents the complete set of defined states, state commands, and state transitions.



SC: State Completed (state transition from an acting state, in white background, to the succeeding state, in grey background)

Properties

Overview

Name	Data type	Accessing	Description
ifUnitModeCurrent	IF_UnitMode, page 63	Read	Interface of the present unit mode. This interface provides status information of the present unit control mode.
etStateCurrent	*ET_States*, page 31	Read	Indicates the present state.
timUnitModeCurrentTime	TIME	Read	Indicates the time elapsed in the present mode.
timStateCurrentTime	TIME	Read	Indicates the time elapsed in the present state.

DefineUnitMode (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to define a unit control mode in the *FB_UnitModeManager2*.

For each unit control mode, an individual state model can be defined using the parameter *i_diStatesDisabled*. The method verifies whether the defined state model is plausible according to the dependency rules.

Interface

Input	Data type	Description
i_sName	STRING	Specifies the name of the unit mode. Refer to Unit Control Modes, page 82.
i_diValue	UDINT	Numeric value of the unit mode. Refer to the table below. Refer to Unit Control Modes, page 82. Valid range: 1...31
i_diStatesDisabled	DINT	This parameter is used to determine the state model for the unit mode. The states 1...17 of the PackML state model can be disabled by turning on the corresponding bit in the DINT. The bits 0...16 represent the individual states as defined in *ET_StateModelDefinition*, page 30. Refer to the rules for state model definition (Dependency Rules, page 83).
i_diStatesModeChangeAllowed	DINT	This parameter is used to determine in which state a unit mode change is allowed. The states 1...17 of the PackML state model where a mode change is allowed are defined by turning on the corresponding bit in the DINT. The bits 0...16 represent the individual states as defined in *ET_StateModelDefinition*, page 30.
i_etInitialState	*ET_States*, page 31	This parameter is used to determine the initial state for the unit mode. This input is optional. If it is unassigned the initial state is set to *ET_States.Stopped*.

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
DefineUnitMode	BOOL	Indicates TRUE if the method is completed successfully.

Unit Control Modes

The ANSI/ISA TR88.00.02-2015 standard defines a set of unit control modes. These are listed in the table below. The names and the corresponding values are reserved in the *FB_UnitModeManager2*.

Unit Control Mode	Value	Description
Invalid	0	Invalid value for unit control mode.
Production (1)	1	This mode represents the mode which is used for routine production.
Maintenance	2	This mode may allow authorized personnel to run an individual machine independent of other machines in a production line.

Unit Control Mode	Value	Description
Manual	3	This mode provides direct control of individual machine modules.
User definable	4...31	User definable states
(1) With the 2015 revision of the ANSI/ISA TR88.00.02-2015, the control mode *Producing* was changed to *Production*. For reasons of compatibility with the previous version of the PackML library, the name *Producing* has been maintained.		

Dependency Rules

Dependency rules for defining a state model as per ANSI/ISA TR88.00.02-2022:

- The states *Stopped*, *Aborted*, and *Execute* are mandatory.
- If the state *Resetting* exists, the state *Idle* must exist. However, the state *Idle* does not necessarily require the state *Resetting*.
- If the state *Completing* exists, the state *Complete* must exist.
- If the states *Holding* or *UnHolding* exist, the state *Held* must exist.
- If the states *Suspending* or *UnSuspending* exist, the state *Suspended* must exist.

DefineUnitModeWithHandler (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method *DefineUnitModeWithHandler* has the same functionality as the method *DefineUnitMode()*. In addition, however, the method *DefineUnitModeWithHandler* offers the possibility of passing an instance of the function block *FB_StateModelHandlerBase*. This gives the *FB_UnitModeManager2* access to this instance and using the *ExecuteCurrentState()* method, the corresponding state method of the instance is invoked.

Interface

Input	Data type	Description
i_sName	STRING	Refer to the interface of *DefineUnitMode* (Method), page 81.
i_diValue	UDINT	
i_diStatesDisabled	DINT	
i_diStatesModeChangeAllowed	DINT	
i_etInitialState	*ET_States*, page 31	
i_ifStateModelHandler	*IF_StateModelHandler*, page 72	This parameter is used to pass the interface to the instance of the *FB_StateModelHandlerBase*.

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
DefineUnitModeWithHandler	BOOL	Indicates TRUE if the method is completed successfully.

RegisterLoggerPoint (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

With the method *RegisterLoggerPoint*, the function block *FB_UnitModeManager2* is registered as a logger point to the global application logger.

The name of the logger point in the application logger is defined by the input *i_sName*.

The input *i_ifParent* specifies the parent logger point under which the logger point for the function block *FB_UnitModeManager2* is registered in the logger point tree.

Following successful execution of the method, the return value of the method is TRUE.

For more general information on the Application Logger, refer to Using ApplicationLogger library (see ApplicationLogger Library Guide).

NOTE: As a prerequisite for using the *RegisterLoggerPoint()* method, the **Application Logger** plug-in must be added to the project and the **Application Logger** service must be registered.

After the logger point was registered successfully the desired logger level must be set using the *SetApplicationLoggerLogLevel()* method.

The *FB_UnitModeManager2* supports the creation of log entries at the following events:

Event	Logger level
The execution of the *DefineUnitMode()* method is unsuccessful.	*APL.ET_LogLevel.Warning*
A state command is triggered.	*APL.ET_LogLevel.DebugMessage*
A state command is performed.	*APL.ET_LogLevel.StatusMessage*
A state command is rejected.	*APL.ET_LogLevel.Warning*
A mode change command is triggered.	*APL.ET_LogLevel.DebugMessage*
A mode change is performed.	*APL.ET_LogLevel.StatusMessage*
A mode change command is rejected.	*APL.ET_LogLevel.Warning*

Interface

Input	Data type	Description
i_ifParent	*APL.IF_LoggerPoint*	Parent logger point under which the logger point of the function block is registered. If the input is set to 0, the global logger point *APL.GVL.G_ifApplicationLogger* is defined as parent.
i_sName	STRING	The name of the logger point that is shown in the application logger.

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
RegisterLoggerPoint	BOOL	Indicates TRUE if the method is completed successfully.

SetApplicationLoggerLogLevel (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to specify the logger level of the logger point of the *FB_UnitModeManager2*. By the logger level you can control which kind of log entries are created by the function block.

For more information on the logger levels, refer to ApplicationLogger library (see ApplicationLogger Library Guide).

Interface

Input	Data type	Description
i_etLogLevel	*APL.ET_LogLevel*	The logger level of the application logger. The level specifies which kind of information is shown in the messages of the application logger.

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
SetApplicationLog-gerLogLevel	BOOL	Indicates TRUE if the method is completed successfully.

ExecuteCurrentState (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

If an instance of the *FB_StateModelHandlerBase* is assigned to the present unit control mode, this method is used to invoke the state method of the present state. For assigning an instance of the *FB_StateModelHandlerBase* to a unit control mode, the unit mode must be defined using the *DefineUnitModeWithHandler()* method.

Interface

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
ExecuteCurrentState	BOOL	Indicates TRUE if the method is completed successfully.

CmdChangeUnitMode (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to change the unit control mode.

Interface

Input	Data type	Description
i_diValue	*DINT*	Numeric value of the target unit control mode

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
CmdChangeUnit-Mode	BOOL	Indicates TRUE if the method is completed successfully.

RemoveUnitMode (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to remove a defined unit control mode from the function block.

Interface

Input	Data type	Description
i_diValue	*DINT*	Numeric value of the unit control mode

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
RemoveUnitMode	BOOL	Indicates TRUE if the method is completed successfully.

GetDefinedUnitModes (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method is used to obtain the definitions of the unit modes. The information about the defined unit modes is stored in the assigned array. The array can be processed by other functions of this library, for example by the function block *FB_VisController*.

Interface

Input/Output	Data type	Description
iq_astUnitModeDefinitions	ARRAY *[1..GPL.Gc_uiMaxNumberOfModes] OF ST_UnitModeDefinition*	Array where the information about the defined unit modes is stored. The modes are stored in the array according to their value. For example: Unit mode with value 7 is stored in array element [7].

Output	Data type	Description
q_etResult	*ET_Result*, page 28	Provides diagnostic and status information as a numeric value. *If q_xError* = FALSE, *q_etResult* provides status information. *If q_xError* = TRUE, *q_etResult* provides diagnostic/error information.
q_sResultMsg	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
GetDefinedUnitModes	BOOL	Indicates TRUE if the method is completed successfully.

Command (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The function block *FB_UnitModeManager2* implements the interface *IF_StateCommands*. Consequently, it implements the methods of this interface.

For information about the methods implemented by the interface, refer to *IF_StateCommands*, page 64.

FB_StateModelHandlerBase

FB_StateModelHandlerBase – General Information

Overview

Type:	Function block
Available as of:	V1.4.2.0

Functional Description

The *FB_StateModelHandlerBase* function block is used to implement the control logic for each state of a state model.

The function block must not be instantiated directly, but must be extended by a custom function block. Inside your own state model handler function block, use the keyword *EXTENDS* in combination with the function block name. In this way, your function block inherits the methods, properties, and local variables of the base function block.

FB_StateModelManual

```
1  FUNCTION_BLOCK FB_StateModelManual EXTENDS PackML.FB_StateModelHandlerBase
2  VAR_INPUT
```

The function block provides the methods according to the states 1...17 of the PackML base state model. The methods are called in the application via the *ExecuteCurrentState()* method of the block *FB_UnitModeManager2*. To do this, the instance of the function block that extends the *FB_StateModelHandlerBase* must be assigned to a unit mode managed by the *FB_UnitModeManager2*. The unit mode definition must be done using the method *FB_UnitModeManager2.DefineUnitModeWithHandler()*.

The state methods of the *FB_StateModelHandlerBase* do not contain the custom logic. The custom logic is added by overriding the methods under the extending block.

The inputs of the state methods provide access to the state commands and the present unit mode.

With the input *i_ifCommands*, it is possible to issue a state command within the methods by calling the*FB_UnitModeManager2*.

The *i_ifUnitMode* input gives you access to information about the present unit mode. This makes it possible to perform certain actions within the method depending on the unit mode. This allows the use of the same state model handler for different unit modes.

Another option is to create multiple custom state model handlers which have access to a common implementation through extension or inheritance. With this option, you can implement only the differences by overriding the respective method.

Clearing (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Clearing* (1) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Stopped (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Stopped* (2) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Starting (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Starting* (3) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Idle (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Idle* (4) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Suspended (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Suspended* (5) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Execute (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Execute* (6) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Stopping (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Stopping* (7) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Aborting (Method

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Aborting* (8) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Aborted (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Aborted* (9) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Holding (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Holding* (10) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Held (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Held* (11) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Restarting (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Restarting* (12) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Suspending (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Suspending* (13) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Unsuspending (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Unsuspending* (14) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Resetting (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Resetting* (15) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Completing (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Completing* (16) from the PackML state model.

Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

Complete (Method)

Overview

Type:	Interface
Available as of:	V1.4.2.0
Inherits from:	–

Description

The method represents the state *Complete* (17) from the PackML state model.

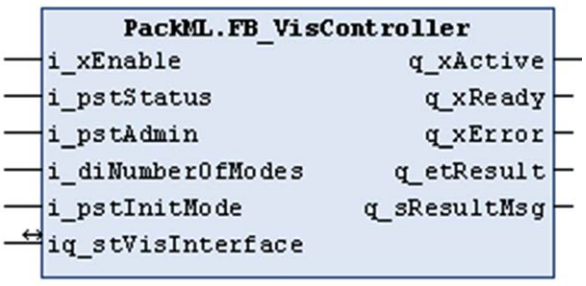
Interface

Intput	Data type	Description
i_xIsFirstCycle	BOOL	Indicates if this is the first time the state method is invoked.
i_ifCommands	*IF_StateCommands*, page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
i_ifUnitMode	*IF_UnitMode*, page 63	Provides information about the unit control mode.

FB_VisController

Overview

Type:	Function block
Available as of:	V1.0.1.0



Functional Description

The *FB_VisController* is used to transfer the data (PackTags) from the application into the data structure of type *ST_VisInterface* which is used by the visualization frames provided with the PackML library. Further, the function block implements functions to control the appearance of the visualization. For example: the scrolling through the alarm list and the control of state-dependent background colors for alarms.

Interface

Input	Data type	Description
i_xEnable	BOOL	Activation and initialization of the function block.
i_pstStatus	POINTER TO *ST_Status*	Through this input, the pointer address to the PackTags of type *ST_Status* is passed to the function block.
i_pstAdmin	POINTER TO *ST_Administration*	Through this input, the pointer address to the PackTags of type *ST_Administration* is passed to the function block.
i_diNumberOfModes	DINT	Number of operation modes. If the value is changed, a reinitialization of the function block is required.
i_pstInitMode	POINTER TO *ST_UnitModeDefinition*	Through this input, the pointer address to the unit mode definitions is passed to the function block.* * The unit mode definition provides the state definitions for each available operation mode and must be provided in an array of *ST_UnitModeDefinition*. The indexes of the array correspond to the numeric value of the available control modes (refer to *ET_Modes*). Therefore, the pointer must point to the index which is associated to the operation mode Producing, which is the first mode.

Input / Output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Through this input / output, the variable is passed to the function block which is linked to the visualizations provided in the PackML library.

Output	Data type	Description
q_xActive	BOOL	If this output is set to TRUE, the function block is active.
q_xReady	BOOL	If this output is set to TRUE, the function block is ready for operation.
q_xError	BOOL	If this output is set to TRUE, an error was detected.
q_etResult	*ET_Result*	Result. Refer to *ET_Result*.
q_sResultMsg	STRING[80]	Additional result message.

Functions

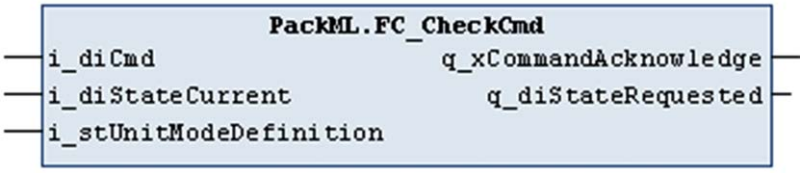
What's in This Chapter

FC_CheckCmd 101
FC_CheckCmd2 102
FC_EtResultToString 103
FC_GetDateTimeAsArray 104
FC_InitStateModelChangeStates 104
FC_InitStateModelExistingStates 106
FC_SetAlarm 108
FC_SetWarning 110
FC_StateSelectionAsDint 111
FC_StateSelectionAsDintFromBool 111

FC_CheckCmd

Overview

Type:	Function
Available as of:	V1.0.4.0



Functional Description

The function *FC_CheckCmd* is used to verify the plausibility of a state transition command (*i_diCmd*) in the present state (*i_diStateCurrent*) of the machine. The plausibility verification is performed based on the unit mode definition (*i_stUnitModeDefinition*), which is provided in an array of type *ST_UnitModeDefinition*. It is assumed that the unit mode definition has been validated (and deemed correct) by the function block *FB_ModeManager*.

If the command is deemed as plausible, the output *q_xCommandAcknowledge* indicates TRUE and the target state is provided on the output *q_diStateRequested*.

FC_CheckCmd supports the PackML base state model as defined in ANSI/ISA TR88.00.02-2015.

The following table indicates which commands are supported in which state and what is the resulting target state, which was derived from the respective unit state definition.

Initial state	**Transition command**	**Target state**
-	*Undefined*	Undefined, the output *q_xCommandAcknowledge* is FALSE.
Execute	*Reset*	**Resetting** if **Complete** and **Completing** do not exist.
Complete	*Reset*	**Resetting**, or **Starting**, if **Resetting** does not exist, or **Execute**, if **Resetting** and **Starting** does not exist.
Stopped		
Idle	*Start*	**Starting**, or **Execute**, if **Starting** does not exist.

Initial state	Transition command	Target state
Stopped	*Start* (command is only processed if **Idle** does not exist)	**Starting**, or **Execute**, if **Starting** does not exist.
Any state except **Aborted**, **Aborting**, **Clearing**, **Stopping**, and **Stopped**	*Stop*	**Stopping**, or **Stopped**, if **Stopping** does not exist.
Execute	*Hold*	**Holding**, or **Held** if **Holding** does not exist.
Held	*UnHold*	**Un-Holding**, or **Execute** if **Un-Holding** does not exist.
Execute	*Suspend*	**Suspending**, or **Suspended** if **Suspending** does not exist.
Suspended	*UnSuspend*	**Un-Suspending**, or **Execute** if **Un-Suspending** does not exist.
Any state except **Aborted** and **Aborting**	*Abort*	**Aborting**, or **Aborted** if **Aborting** does not exist.
Aborted	*Clear*	**Clearing**, or **Stopped** if **Clearing** does not exist.

Interface

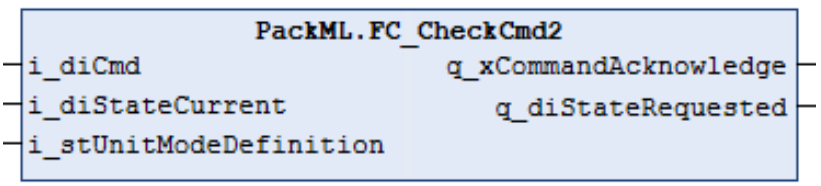
Inputs	Data type	Description
i_diCmd	DINT	The state transition command; the PackTag *ST_Command.CntrlCmd* should be applied to the input.
i_diStateCurrent	DINT	Machine present state; the PackTag *ST_Status.StateCurrent* should be applied to the input.
i_stUnitModeDefinition	*ST_UnitModeDefinition*	The operation mode definition of the present operation mode; the PackTag *ST_Status.UnitModeCurrent* should be used as index variable of array of *ST_UnitModeDefinition*.

Output	Data type	Description
q_xCommandAcknowledge	BOOL	Indicates TRUE if the given command is plausible.
q_diStateRequested	DINT	Provides the target state if the given command is plausible.

FC_CheckCmd2

Overview

Type:	Function
Available as of:	V1.4.2.0



Functional Description

The *FC_CheckCmd2* function is a further development of the *FC_CheckCmd* function.

Compared to the *FC_CheckCmd* function, the *FC_CheckCmd2* supports the PackML base state model as defined in ANSI/ISA TR88.00.02-2022.

For the functional description and the documentation of the interface, refer to function *FC_CheckCmd*, page 101.

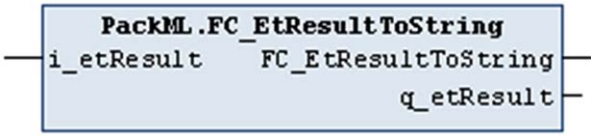
The following table indicates the commands supported in each state and what is the resulting target state which is derived from the respective unit state definition.

Initial state	Transition command	Target state
Any state, or **Undefined**	*Undefined*	**Undefined**, the output *q_xCommandAcknowledge* is FALSE.
Stopped or **Complete**	*Reset*	**Resetting**, or **Idle** if **Resetting** does not exist.
Stopped or **Idle**	*Start*	Command *Start* in state **Stopped** is supported only if **Idle** does not exist. **Starting**, or **Execute** if **Starting** does not exist.
Execute or **Suspended**	*Hold*	**Holding**, or **Held** if **Holding** does not exist.
Execute	*Suspend*	**Suspending**, or **Suspended** if **Suspending** does not exist.
Execute, **Held** or **Suspended**	*Complete*	**Completing**, or **Completed** if **Completing** does not exist.
Any state except **Aborting**, **Aborted**, **Clearing**, **Stopping**, and **Stopped**	*Stop*	**Stopping**, or **Stopped** if **Stopping** does not exist.
Any state except **Aborting** or **Aborted**	*Abort*	**Aborting**, or **Aborted** if **Aborting** does not exist.
Aborted	*Clear*	**Clearing**, or **Stopped** if **Clearing** does not exist.

FC_EtResultToString

Overview

Type:	Function
Available as of:	V1.0.1.0



Task

Convert an enumeration element of type *ET_Result* to a string value.

Functional Description

Using the function *FC_EtResultToString*, you can convert an enumeration element of type *ET_Result* to a string value.

Interface

Input	Data type	Description
i_etResult	*ET_Result*	Enumeration with the result.

Output	Data type	Description
q_etResult	*ET_Result*	Enumeration with the result.

Return Value

Data type	Description
STRING(80)	The *ET_Result* converted to a string value.

FC_GetDateTimeAsArray

Overview

Type:	Function
Available as of:	V1.1.0.0



Functional Description

The function *FC_GetDateTimeAsArray* is used to obtain the RTC of the controller and to convert it to the data type *DateTimeArray*.

The return value of the function is of type *DateTimeArray* and represents the value of the RTC of the controller.

FC_InitStateModelChangeStates

Overview

Type:	Function
Available as of:	V1.0.1.0



Task

Use the function *FC_InitStateModelChangeStates* to define, for an operation mode, from which state a change of operation mode is possible.

Functional Description

Using the input/output *iq_stStateModel*, a structure of type *ST_UnitModeDefinition* is passed to the function.

Use the 17 inputs of type BOOL, which reflect the states of an operation mode, to define from which states a change of operation mode is possible.

TRUE indicates that a change of operation mode is possible; FALSE indicates that a change of operation mode must not be attempted from this state.

Following successful initialization of the operation mode, the function provides a TRUE.

Interface

Input	Data type	Description
i_xStopping	BOOL	If this input is set to TRUE, the **Stopping** state is allowed to change the operation mode.
i_xStopped	BOOL	If this input is set to TRUE, the **Stopped** state is allowed to change the operation mode.
i_xResetting	BOOL	If this input is set to TRUE, the **Resetting** state is allowed to change the operation mode.
i_xIdle	BOOL	If this input is set to TRUE, the **Idle** state is allowed to change the operation mode.

Input	Data type	Description
i_xStarting	BOOL	If this input is set to TRUE, the **Starting** state is allowed to change the operation mode.
i_xExecute	BOOL	If this input is set to TRUE, the **Execute** state is allowed to change the operation mode.
i_xHolding	BOOL	If this input is set to TRUE, the **Holding** state is allowed to change the operation mode.
i_xHeld	BOOL	If this input is set to TRUE, the **Held** state is allowed to change the operation mode.
i_xUnHolding	BOOL	If this input is set to TRUE, the **UnHolding** state is allowed to change the operation mode.
i_xSuspending	BOOL	If this input is set to TRUE, the **Suspending** state is allowed to change the operation mode.
i_xSuspended	BOOL	If this input is set to TRUE, the **Suspended** state is allowed to change the operation mode.
i_xUnSuspending	BOOL	If this input is set to TRUE, the **UnSuspending** state is allowed to change the operation mode.
i_xCompleting	BOOL	If this input is set to TRUE, the **Completing** state is allowed to change the operation mode.
i_xComplete	BOOL	If this input is set to TRUE, the **Complete** state is allowed to change the operation mode.
i_xAborting	BOOL	If this input is set to TRUE, the **Aborting** state is allowed to change the operation mode.
i_xAborted	BOOL	If this input is set to TRUE, the **Aborted** state is allowed to change the operation mode.
i_xClearing	BOOL	If this input is set to TRUE, the **Clearing** state is allowed to change the operation mode.

Input / output	Data type	Description
iq_stStateModel	*ST_UnitModeDefinition*	This structure reflects the operation mode which is to be initialized.

Return Value

Data type	Description
BOOL	TRUE if initialization of the operation mode was successful.

FC_InitStateModelExistingStates

Overview

Type:	Function
Available as of:	V1.0.1.0



Task

Use the *FC_InitStateModelExistingStates* function to initialize the state model and define an operation mode. By presenting TRUE or FALSE at the inputs, you are selecting those states to be part of your state model.

Functional Description

Using the input/output *iq_stStateModel*, a structure of type *ST_UnitModeDefinition* is passed to the function.

This structure reflects an operation mode. Using the 18 inputs of type BOOL which reflect the existence of an operation mode and its states, it is possible to define which states make up the relevant operation mode.

TRUE signifies that the state exists; FALSE signifies that the state does not exist.

Following successful initialization of the operation mode, the function provides a TRUE.

Interface

Input	Data type	Description
i_xModeExistent	BOOL	If this input is set to TRUE, the operation mode is available.
i_xStopping	BOOL	If this input is set to TRUE, the **Stopping** state is available.
i_xStopped	BOOL	If this input is set to TRUE, the **Stopped** state is available.
i_xResetting	BOOL	If this input is set to TRUE, the **Resetting** state is available.
i_xIdle	BOOL	If this input is set to TRUE, the **Idle** state is available.

Input	Data type	Description
i_xStarting	BOOL	If this input is set to TRUE, the **Starting** state is available.
i_xExecute	BOOL	If this input is set to TRUE, the **Execute** state is available.
i_xHolding	BOOL	If this input is set to TRUE, the **Holding** state is available.
i_xHeld	BOOL	If this input is set to TRUE, the **Held** state is available.
i_xUnHolding	BOOL	If this input is set to TRUE, the **Un-Holding** state is available.
i_xSuspending	BOOL	If this input is set to TRUE, the **Suspending** state is available.
i_xSuspended	BOOL	If this input is set to TRUE, the **Suspended** state is available.
i_xUnSuspending	BOOL	If this input is set to TRUE, the **Un-Suspending** state is available.
i_xCompleting	BOOL	If this input is set to TRUE, the **Completing** state is available.
i_xComplete	BOOL	If this input is set to TRUE, the **Complete** state is available.
i_xAborting	BOOL	If this input is set to TRUE, the **Aborting** state is available.
i_xAborted	BOOL	If this input is set to TRUE, the **Aborted** state is available.
i_xClearing	BOOL	If this input is set to TRUE, the **Clearing** state is available.

Input / output	Data type	Description
iq_stStateModel	*ST_UnitModeDefinition*	This structure reflects the operation mode which is to be initialized.

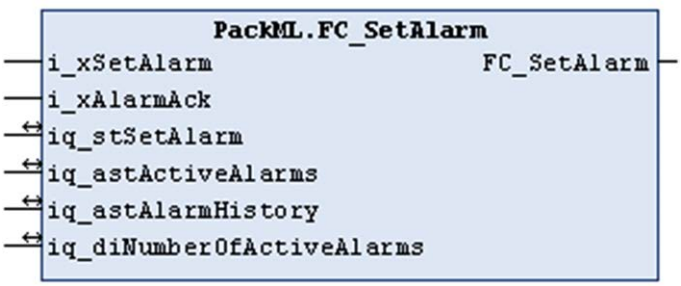
Return Value

Data type	Description
BOOL	TRUE if initialization of the operation mode was successful.

FC_SetAlarm

Overview

Type:	Function
Available as of:	V1.0.1.0



Functional Description

The function *FC_SetAlarm* is an auxiliary function to write alarm messages into the appropriate administration tag and also delete them again from the same. An alarm is identified exclusively by its unique identifier and the associated value. The identifier and its value are specified by the input/output parameter *iq_stSetAlarm* which corresponds with the input/output parameter *iq_astAlarms[#]*.

This function cannot be used simultaneously in several tasks.

Through the input *iq_stSetAlarm*, a machine-specific alarm message can be passed to the function.

Through the input/output parameter *iq_astAlarms*, the administration tag *Admin.Alarm[#]* must be passed to the function. It represents the list of active alarms in the machine unit in chronological order starting with the first occurred and still active alarm.

Through the input/output parameter *iq_stAlarmHistory*, the administration tag *Admin.AlarmHistory[#]* must be passed to the function. It represents the alarm history of the machine unit containing the alarms in chronological order starting with the last reset alarm.

If the function is called with input *i_xSetAlarm* = *TRUE*, the function verifies whether the specified alarm message is already in the list of active alarms. If not, the function obtains the RTC of the controller and writes it together with the specified alarm message into the list of active alarms. If the maximum number of messages in the list is reached, no new message is added.

If the function is called with *i_xSetAlarm* = *FALSE*, the specified alarm message is removed from the list of active alarms. Then it is written at the first position of the alarm history while the existing entries are moved one position down in the list. If the list is full, the oldest reset alarm is removed.

If the function is called with inputs *i_xAckAlarm* and *i_xSetAlarm* = *TRUE*, the function obtains the RTC of the controller and updates the parameter (tag) *AckDateTime* for the specified alarm message in the list of active alarms. If an alarm has been acknowledged once, a new request to acknowledge has no effect.

The return value of the function indicates TRUE if the specified alarm has been either written into the list or deleted from the list, or has been set to acknowledged. If the function returns FALSE, either no action was requested or the maximum number of messages in the list is reached.

Interface

Input	Data type	Description
i_xSetAlarm	BOOL	• TRUE: the reason code or machine-specific alarm message passed to the input/output *iq_stSetAlarm* is written into the tag at the input/output *iq_stActiveAlarm*. • FALSE: the message is deleted from *iq_stActiveAlarms* and entered *iq_stAlarmHistory*.
i_xAlarmAck	BOOL	TRUE: the message has been detected. The appropriate time stamp for *Admin.Alarm[#].TimeAck* is determined.

Input/Output	Data type	Description
iq_stSetAlarm	*ST_InitAlarm*	Reason code or machine-specific alarm message is passed to this input/output.
iq_astActiveAlarms	ARRAY [1..*Gc_uiMaxNumberOfAlarms]* OF *ST_Alarm*	The administration tag *Admin.Alarm[#]* should be applied to this input/output.

Input/Output	Data type	Description
iq_astAlarmHistory	ARRAY [1..*Gc_uiNumberOfAlarmHistory]* OF *ST_Alarm*	The administration tag *Admin.AlarmHistory[#]* should be applied to this input/output:
iq_diNumberOfActiveAlarms	DINT	Provides the number of active alarm messages.

FC_SetWarning

Overview

Type:	Function
Available as of:	V1.1.0.0



Functional Description

The function *FC_SetWarning* is an auxiliary function to write advisory messages into the appropriate administration tag and also delete them again from the same. An advisory is identified exclusively by its unique identifier and the associated value. The identifier and its value are specified by the input/output parameter *iq_stSetWarning* which corresponds with the input/output parameter *iq_astWarnings[#]*.

This function cannot be used simultaneously in several tasks.

Through the input *iq_stSetWarning*, a machine-specific advisory message can be passed to the function.

Through the input/output parameter *iq_astWarnings*, the administration tag *Admin.Warning[#]* must be passed to the function. It represents the list of active advisories in the machine unit in chronological order starting with the first occurred and still active advisory.

If the function is called with input *i_xSetWarning* = *TRUE*, the function verifies whether the specified advisory message is already in the list of active advisories. If not, the function obtains the RTC of the controller and writes it together with the specified advisory message into the list of active advisories. If the maximum number of messages in the list is reached, no new messages are added.

If the function is called with *i_xSetWarning* = *FALSE*, the specified advisory message is removed from the list of active advisories.

If the function is called with inputs *i_xAckWarning* and *i_xSetWarning* = *TRUE*, the function obtains the RTC of the controller and updates the parameter (tag) *AckDateTime* for the specified advisory message in the list of active advisories. If an advisory has been acknowledged once, a new request to acknowledge has no effect.

The return value of the function indicates TRUE if the specified advisory has been either written into the list or deleted from the list, or has been set to acknowledged. If the function returns FALSE, either no action was requested or the maximum number of messages in the list is reached.

Interface

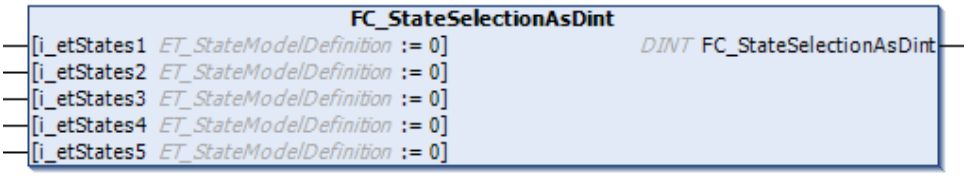
Input	Data type	Description
i_xSetWarning	BOOL	Request to write (TRUE) or remove (FALSE) the specified message into/ from the list linked to *iq_astWarnings*.
i_xWarningAck	BOOL	Request to acknowledge an active advisory.

Input/Output	Data type	Description
iq_stSetWarning	*ST_InitAlarm*	Specifies the advisory message to be treated by the function.
iq_astWarnings	ARRAY [1..*Gc_ uiMaxNumberOf-Warnings]* OF *ST_ Alarm*	The administration tag *Admin.Warning [#]* must be linked to this input/output.
iq_diNumberOfActiveWarnings	DINT	Provides the number of advisories in the list.

FC_StateSelectionAsDint

Overview

Type:	Function
Available as of:	V1.4.7.0



Functional Description

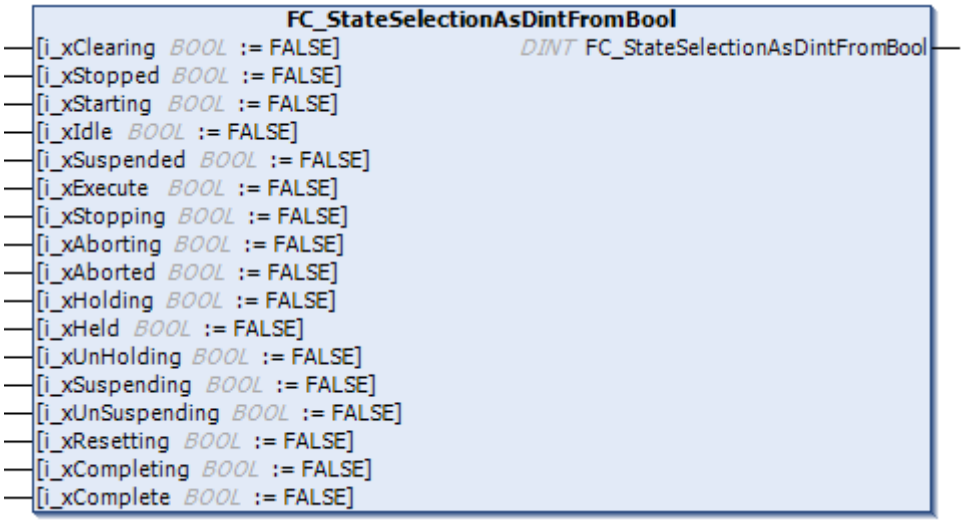
The *FC_StateSelectionAsDint* function is a utility function for definition of the PackTags *StatesDisabled* and *StatesModeChangeAllowed* that allows you to assign the corresponding parameter to the method *DefineUnitMode* of the *FB_ UnitModeManager2*, page 81. The states can be configured by using the input of the function of type *ET_StateModelDefinition*.

The inputs of type ET_StateModelDefinition, page 30 are combined using the OR expression to the return value of type DINT.

FC_StateSelectionAsDintFromBool

Overview

Type:	Function
Available as of:	V1.4.7.0



Functional Description

The *FC_StateSelectionAsDintFromBool* function is a utility function for definition of the PackTags *StatesDisabled* and *StatesModeChangeAllowed* that allows you to assign the corresponding parameter to the method *DefineUnitMode* of the *FB_UnitModeManager2*, page 81.

The Boolean inputs representing the PackML states are combined using the OR expression to the return value of type DINT.

Visualization

What’s in This Part

BackgroundFrames 114
States Frames 116
Visualization Frames 117

BackgroundFrames

What's in This Chapter

FR_<BackgroundFrame> 114

FR_<BackgroundFrame>

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display a frame in the background.

Functional Description

The *FR_<BackgroundFrame>* visualization frames (for example, *FR_Control1024x768*) are frames in the background to group the other frames, and to realize control and navigation functionalities. See examples below.

The following background frames are available:

Frame	Description
FR_Control1024x768	Control frame, size 1024x768 pixels
FR_Nav1024x768	Navigation frame, size 1024x768 pixels
FR_NavControl1024x768	Navigation and control frame, size 1024x768 pixels
FR_Control800x600	Control frame, size 800x600 pixels
FR_Nav800x600	Navigation frame, size 800x600 pixels
FR_NavControl800x600	Navigation and control frame, size 800x600 pixels

Interface

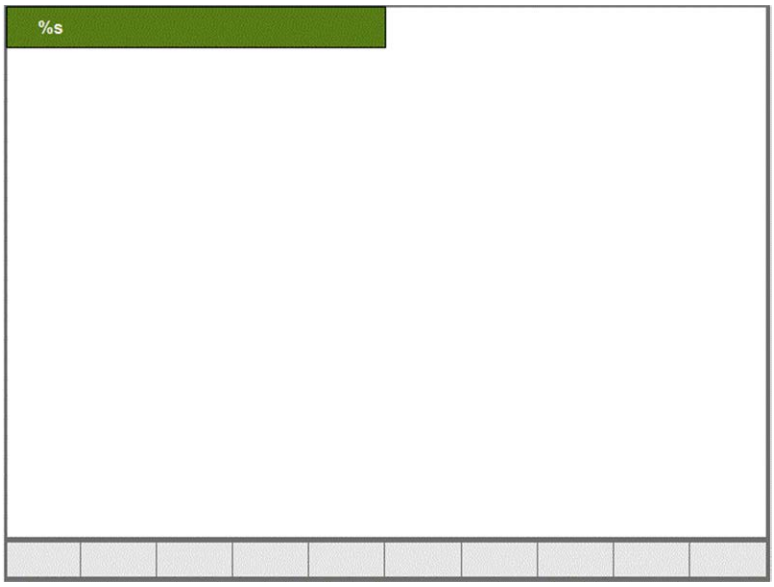
Input	Data type	Description
i_sTitle	STRING(80)	Title of the background frame

Examples

FR_Control1024x768 with controls on the left side



FR_Nav1024x768 with navigation buttons at the bottom



States Frames

What's in This Chapter

FR_<State> 116

FR_<State>

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the single states of the state model for the operation mode.

Functional Description

For each specified state in the state model (refer to *ET_States*), a visualization is provided.

The frames are used by the *FR_DynStateModel* to display the single states of the state model. See following example.

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example



Visualization Frames

What's in This Chapter

General 117
FR_Alarm 118
FR_AlarmHistory 118
FR_CurrentModeAndStateTime 119
FR_DynStateModel 120
FR_DynStateModel_2022 121
FR_ModeAndStateTime 122
FR_PackMLStateModel_2022_Base 123
FR_PackMLStateModel_2022_UnitModeManager 124
FR_ProdConsumedCount 125
FR_ProdDefectiveCount 126
FR_ProdProcessedCount 127
FR_Warning 128
FR_StopReason 128
FR_StatesDisabled 129

General

Overview

The library provides several visualization frames.

You can use these frames to display information about:

- Alarms
- Machine operation modes and states
- Production data

FB_VisController Function Block

The main task of the function block *FB_VisController* is to process data provided by the administration tags and status tags and to provide these to the visualization frames via the visualization interface input/output *iq_stVisInterface* of type *ST_VisInterface*.

Visualization Frames

- *FR_Alarm*
- *FR_AlarmHistory*
- *FR_CurrentModeAndStateTime*
- *FR_DynStateModel*
- *FR_ModeAndStateTime*
- *FR_ProdConsumedCount*
- *FR_ProdDefectiveCount*
- *FR_ProdProcessedCount*
- *FR_Warning*
- *FR_StopReason*
- *FR_StatesDisabled*

FR_Alarm

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display an alarm list of the active alarms.

Functional Description

FR_Alarm is a visualization frame to display the alarm list of the active alarms (*Admin.Alarm[#]*). See example below.

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example

#	ID	Value	Cat.	Message	Trigger	DateTime	AckDateTime
1	39	2	2	This alarm is active.	TRUE	2017 - 9 - 14 - 14:15:59.619000	2017 - 9 - 14 - 14:15:59.619000
2	1234	1	1	This alarm is inactive but not acknowledged.	FALSE	2017 - 9 - 14 - 14:36:23.2365	0 - 0 - 0 - 0 : 0 : 0 .0
3	456	63	4	This alarm is inactive and acknowledged.	FALSE	2017 - 9 - 14 - 14:52:26.56987	2017 - 9 - 14 - 14:53:11.369
4	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
5	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
6	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
7	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
8	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
9	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
v	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0

FR_AlarmHistory

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the alarm history.

Functional Description

FR_AlarmHistory is a visualization frame to display the alarm history (*Admin.AlarmHistory[#]*). See example below.

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example

#	ID	Value	Cat.	Message	DateTime	AckDateTime
1	38	1	19	This alarm is not yet acknowledged.	2017 - 9 - 14 - 14:15:57.549000	0 - 0 - 0 - 0:0:0.0
2	37	1	65	This alarm is already acknowledged.	2017 - 9 - 14 - 14:14:30.540000	2017 - 9 - 14 - 14:14:41.233
3	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
4	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
5	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
6	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
7	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
8	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
9	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0
v	0	0	0		0 - 0 - 0 - 0:0:0.0	0 - 0 - 0 - 0:0:0.0

FR_CurrentModeAndStateTime

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the time the machine has been running in the present operation mode/state.

Functional Description

FR_CurrentModeAndStateTime is a visualization frame to display the time the machine has been running in the present operation mode (*Admin.ModeCurrentTime[#] & Admin.ModeCumulativeTime[#]*) and state (*Admin.StateCurrentTime[#,#] & Admin.StateCumulativeTime[#,#]*). See example below.

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example

Current Unit Mode:	1
Time in Current Mode:	58
Cummulative Time in Current Mode:	4885
Current State:	6
Time in Current State:	40
Cummulative Time in Current State:	40

NOTE: This example only demonstrates the PackTag information, which, according to the standards, is presented as seconds.

FR_DynStateModel

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Dynamically display the visualization of the state model for the operation mode during online operation.

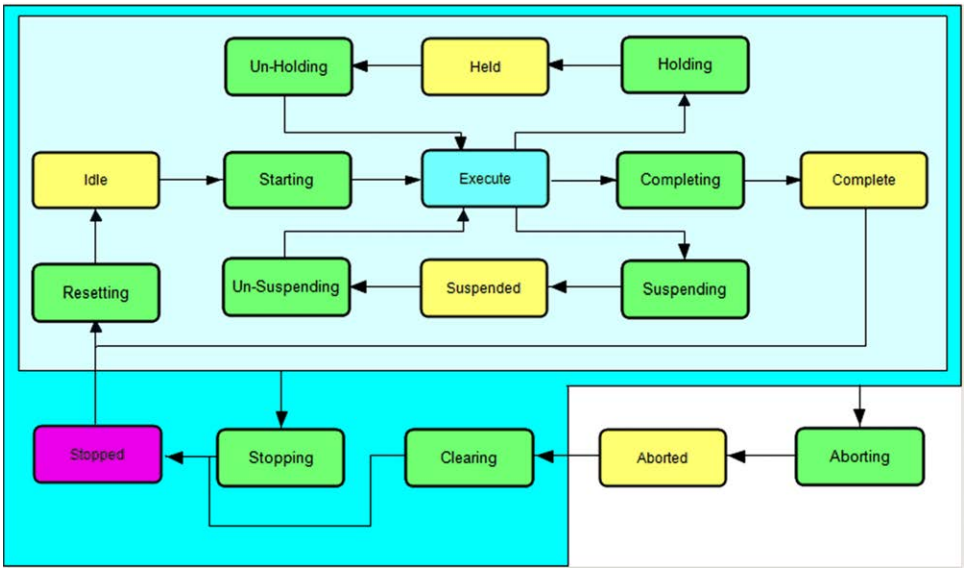
Functional Description

FR_DynStateModel provides a visualization frame to display the state models. This frame dynamically generates the visualization of the corresponding state model during online operation.

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example



FR_DynStateModel_2022

Overview

Type:	Visualization frame
Available as of:	V1.4.2.0

Task

Dynamically display the visualization of the state model of the present unit mode.

Functional Description

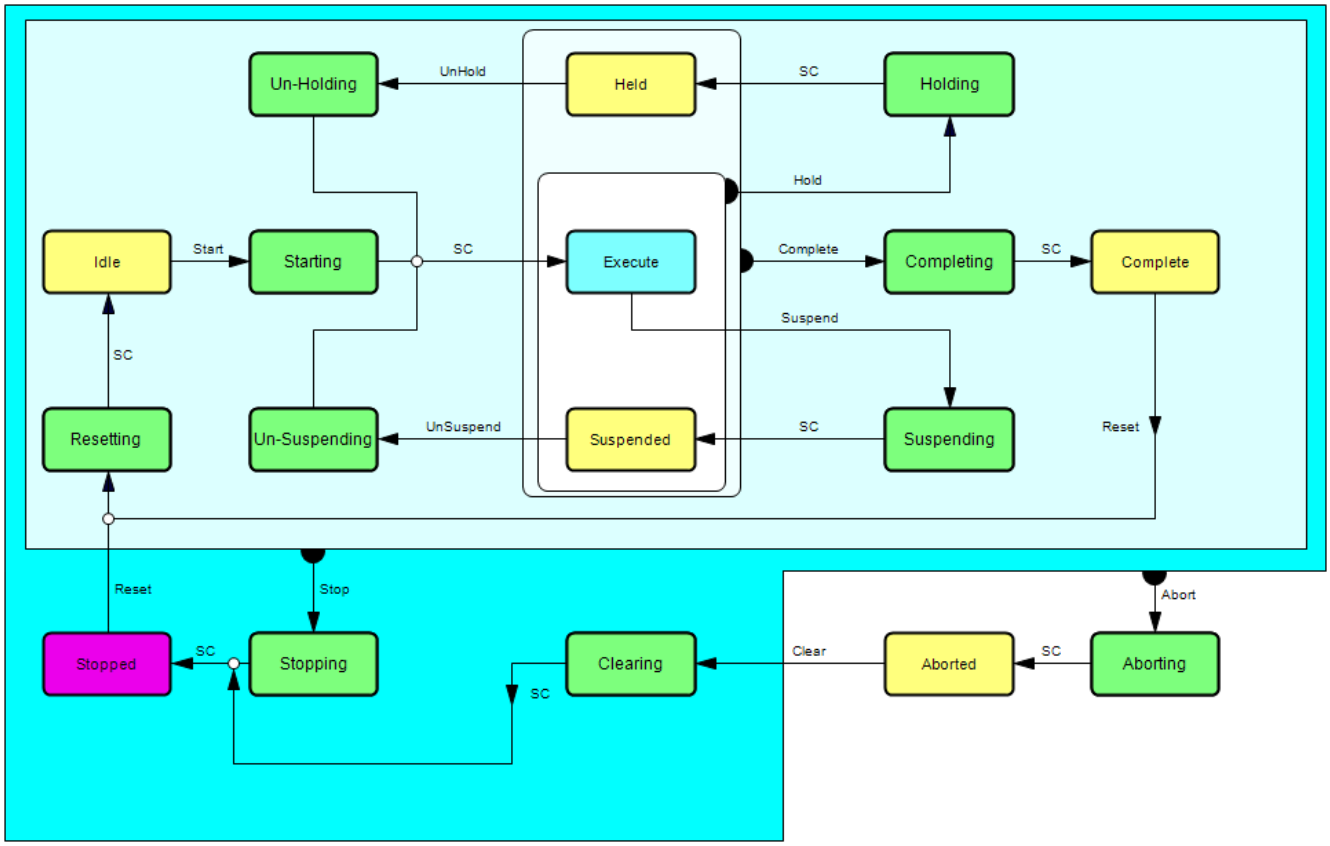
The *FR_DynStateModel_2022* dynamically displays the visualization of the state model during online operation. The state model is compliant to the PackML base state model as defined in ANSI/ISA TR88.00.02-2022.

Interface

Input / output	Data type	Description
i_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example

The displayed state model is dynamically adapted based on the state model defined.



FR_ModeAndStateTime

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the operation time of all operation modes and their respective states.

Functional Description

FR_ModeAndStateTime is a visualization frame to display the operation time of all operation modes (*Admin.ModeCurrentTime[#] & Admin.ModeCumulativeTime[#]*) and their respective states (*Admin.StateCurrentTime[#,#] & Admin. StateCumulativeTime[#,#]*).

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example

Unit Mode [0]			>
Current Time	0		
Cumulative Time	18		
States			
Current Time		Cumulative Time	
Undefined	0	Undefined	18
Execute	0	Execute	0
Stopping	0	Stopping	0
Stopped	0	Stopped	0
Resetting	0	Resetting	0
Idle	0	Idle	0
Starting	0	Starting	0
Holding	0	Holding	0
Held	0	Held	0
Unholding	0	Unholding	0
Suspending	0	Suspending	0
Suspended	0	Suspended	0
Unsuspending	0	Unsuspending	0
Completing	0	Completing	0
Complete	0	Complete	0
Aborting	0	Aborting	0
Aborted	0	Aborted	0
Clearing	0	Clearing	0

FR_PackMLStateModel_2022_Base

Overview

Type:	Visualization frame
Available as of:	V1.4.7.0

Task

Dynamically display the visualization of the state model and the present state.

Functional Description

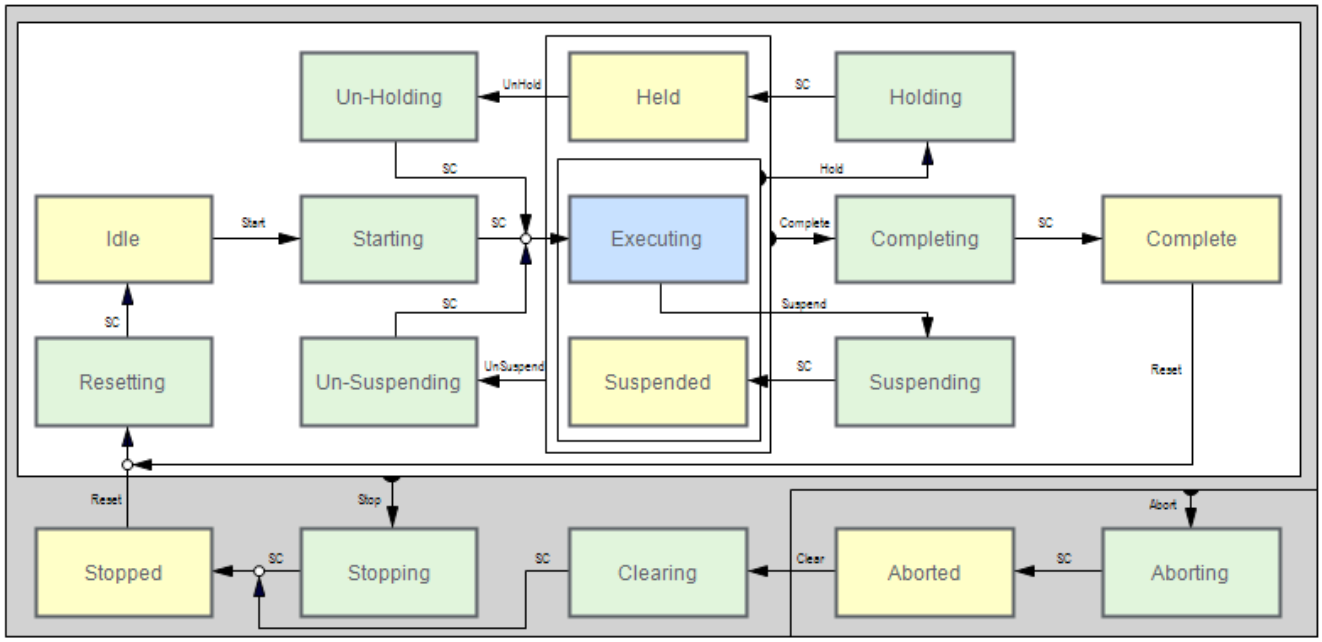
The *FR_PackMLStateModel_2022_Base* dynamically displays the state model and the present state. The state model is compliant to the PackML base state model as defined in ANSI/ISA TR88.00.02-2022.

Interface

Input / output	Data type	Description
iq_diStateCurrent	DINT	Indicates the present state.
iq_dwStatesDisabled	DWORD	Indicates the definition of the state model.

Example

The displayed state model is dynamically adapted based on the state model defined.



FR_PackMLStateModel_2022_UnitModeManager

Overview

Type:	Visualization frame
Available as of:	V1.4.7.0

Task

Dynamically display the state model of the present unit mode and the present state.

Functional Description

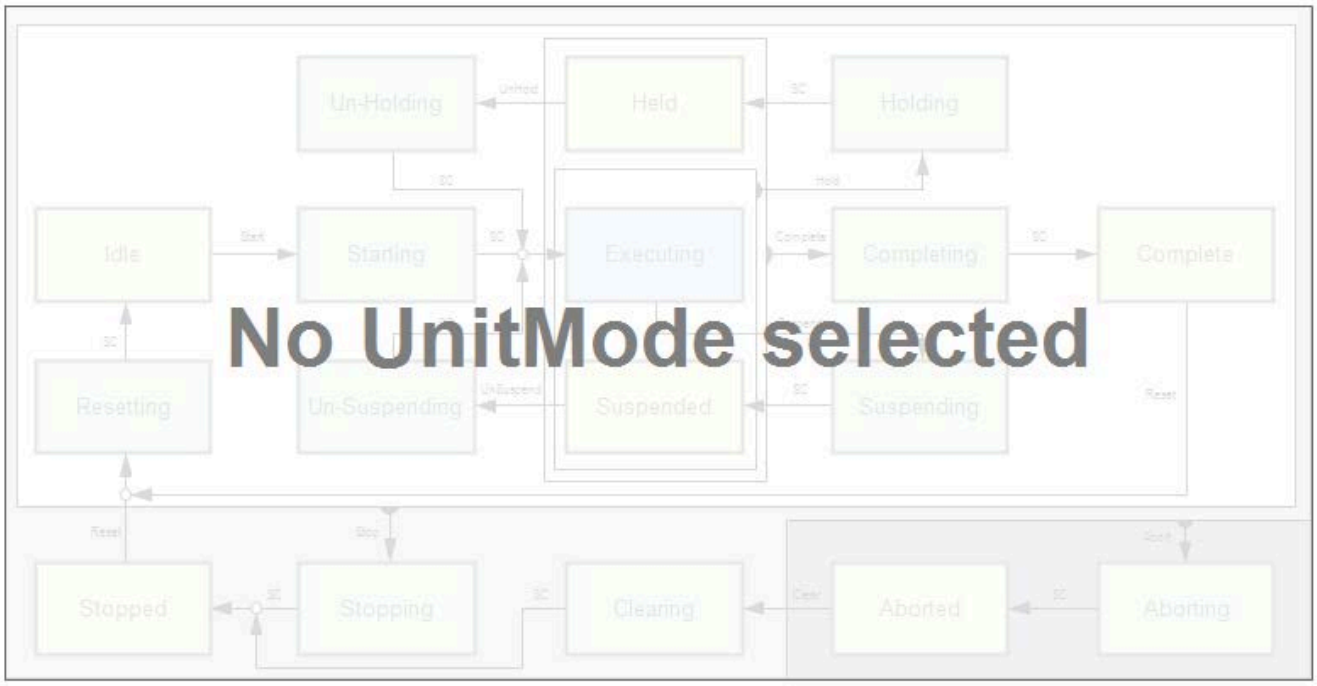
The *FR_PackMLStateModel_2022_UnitModeManager* in conjunction with the *FB_UnitModeManager2*, page 79 dynamically displays the state model of the present unit mode and the present state. The state model is compliant to the PackML base state model as defined in ANSI/ISA TR88.00.02-2022.

Interface

Input / output	Data type	Description
iq_fbUnitModeManager	*FB_UnitModeManager2*, page 79	Indicates the state model of the present unit mode and the present state.

Example

The displayed state model is dynamically adapted based on the state model defined.



FR_ProdConsumedCount

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the product counters of the consumables.

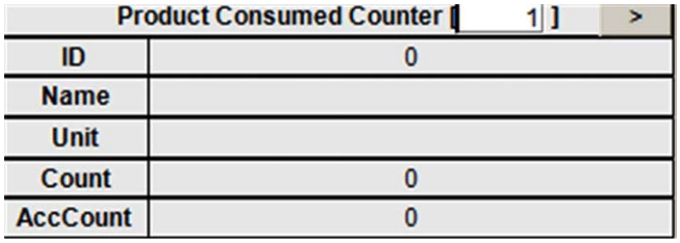
Functional Description

FR_ProdConsumedCount is a visualization frame to display the product counters of the consumables (*Admin.ProdConsumedCount[#]*).

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example



FR_ProdDefectiveCount

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the product counters of the goods detected as not acceptable.

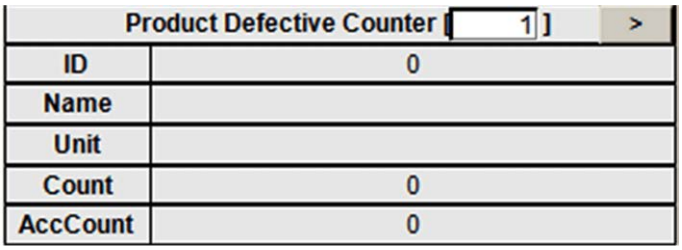
Functional Description

FR_ProdDefectiveCount is a visualization frame to display the product counters of the goods detected as not acceptable (*Admin.ProdDefectiveCount[#]*)

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example



FR_ProdProcessedCount

Overview

Type:	Visualization frame
Available as of:	V1.0.1.0
Implements:	VisuElems.IVisualization

Task

Display the product counters of the produced goods.

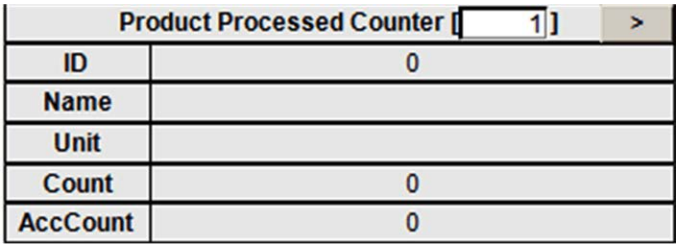
Functional Description

FR_ProdProcessedCount is a visualization frame to display the product counters of the produced goods (*Admin.ProdProcessedCount[#]*).

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example



FR_Warning

Overview

Type:	Visualization frame
Available as of:	V1.1.0.0
Implements:	VisuElems.IVisualization

Task

Display a list of the active advisories.

Functional Description

FR_Warning is a visualization frame to display the list of the active advisories (*Admin.Warning[#]*). See example below.

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example

#	ID	Value	Cat.	Message	Trigger	DateTime	AckDateTime
1	123	1	1	This Warning is active.	TRUE	2017 - 9 - 14 - 14 : 58 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
2	456	2	2	This warning is inactive but not acknowledged.	FALSE	2017 - 14 - 9 - 14 : 58 : 1 .123	0 - 0 - 0 - 0 : 0 : 0 .0
3	789	3	3	This warning is inactive and acknowledged.	FALSE	2017 - 9 - 14 - 14 : 58 : 32 .1258	2017 - 9 - 14 - 14 : 59 : 1 .0
4	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
5	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
6	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
7	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
8	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
9	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0
v	0	0	0		FALSE	0 - 0 - 0 - 0 : 0 : 0 .0	0 - 0 - 0 - 0 : 0 : 0 .0

FR_StopReason

Overview

Type:	Visualization frame
Available as of:	V1.1.0.0
Implements:	VisuElems.IVisualization

Task

Display the PackTag *Admin.StopReason*.

Functional Description

FR_StopReason is a visualization frame to display the PackTag *Admin.StopReason*.

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example

ID	Value	Cat.	Message	DateTime
10	8	21	Abort command detetcted.	2017 - 10 - 23 - 13:15:26.532000

FR_StatesDisabled

Overview

Type:	Visualization frame
Available as of:	V1.1.0.0
Implements:	VisuElems.IVisualization

Task

Monitor and control the PackTag *Admin.StatesDisabled*.

Functional Description

FR_StatesDisabled is a visualization frame to monitor and control the PackTag *Admin.StatesDisabled*.

Interface

Input / output	Data type	Description
iq_stVisInterface	*ST_VisInterface*	Interface to the *FB_VisController*

Example


StatesDisabled
Clearing
Stopped
Starting
Idle
Suspended
Execute
Stopping
Aborting
Aborted
Holding
Held
UnHolding
Suspending
UnSuspending
Resetting
Completing
Complete

Glossary

P

POU:

(*program organization unit*) A variable declaration in source code and a corresponding instruction set. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

R

RTC:

(*real-time clock*) A battery-backed time-of-day and calender clock that operates continuously, even when the controller is not powered for the life of the battery.

Index

A

Aborted 95
Aborting 94

B

BackgroundFrames 114
base state model PackML 80

C

Clearing 91
CmdAbort 69
CmdChangeUnitMode 87
CmdClear 70
CmdHold 66
CmdReset 64
CmdStart 65
CmdStop 66
CmdSuspend 68
CmdUnHold 67
CmdUnSuspend 68
Command 90
common inputs and outputs
  behavior of function blocks with the input *i_xEnable* 25
Complete 99
Completing 98

D

DateTimeArray 55
DefineUnitMode 81
DefineUnitModeWithHandler 83
diagnostic concept 23

E

ET_Cmd 27
  abort 102
  Abort 27
  clear 102
  Clear 27
  hold 102
  Hold 27
  reset 101
  Reset 27
  start 101
  Start 27
  stop 102
  Stop 27
  suspend 102
  Suspend 27
  undefined 101
  Undefined 27
  UnHold 27, 102
  UnSuspend 27, 102
ET_Modes 28
  Maintenance 28
  Manual 28
  Producting 28
  Undefined 28
ET_Result 28
  AlreadyExists 30
  Disabled 29
  Initializing 29
  InvalidInput 29
  InvalidStateModelNoAbortedState 29
  InvalidStateModelNoCompleteState 29
  InvalidStateModelNoExecuteState 29
  InvalidStateModelNoHeldState 29
  InvalidStateModelNoIdleState 29
  InvalidStateModelNoResettingState 29
  InvalidStateModelNoStoppedState 29
  InvalidStateModelNoSuspendedState 29
  MaxNumberOfUnitModesExceeded 29
  ModeChangeRequestRejected 29
  NoStateModelHandler 30
  NotReady 30
  NoUnitMode 30
  NumberOfModesRange 29
  Ok 29
  PointerAdminInvalid 29
  PointerInitModeInvalid 29
  PointerStatusInvalid 29
  RegisterLoggerPointFailed 30
  ResettingTimerAndCounter 29
  Running 29
  StateChangeRequestRejected 29
  StateCurrentRange 29
  UnitModeActive 30
  UnitModeCurrentRange 29
  UnitModeNotFound 30
  UnitModeRange 29
  UnknownResult 29
ET_StateModelDefinition 30
  Aborted 31
  AbortedAndStopped 31
  Aborting 31
  Clearing 31
  Complete 31
  CompleteDisabled 31
  Completing 31
  Execute 31
  Held 31
  HoldDisabled 31
  Holding 31
  Idle 31
  NoStateDisabled 31
  Resetting 31
  Starting 31
  Stopped 31
  Stopping 31
  SuspendDisabled 31
  Suspended 31
  Suspending 31
  UnHolding 31
  Unsuspending 31
ET_States 31
  Aborted 32
  Aborting 32
  Clearing 32
  Complete 32
  Completing 32
  Execute 32
  Held 32
  Holding 32
  Idle 32
  Resetting 32
  Starting 32
  Stopped 32
  Stopping 32

Suspended 32
Suspending 32
Undefined 32
UnHolding 32
UnSuspending 32
Execute 93
ExecuteCurrentState 87

F

FB_DataManagement 74
FB_ModeManager 76
FB_StateModelHandlerBase 90
Aborted 95
Aborting 94
Clearing 91
Complete 99
Completing 98
Execute 93
Held 96
Holding 95
Idle 92
Resetting 98
Restarting 96
Starting 92
Stopped 91
Stopping 94
Suspended 93
Suspending 97
Unsuspending 97
FB_UnitModeManager2 79
CmdChangeUnitMode 87
Command 90
DefineUnitMode 81
DefineUnitModeWithHandler 83
ExecuteCurrentState 87
GetDefinedUnitModes 89
properties 81
RegisterLoggerPoint 84
RemoveUnitMode 88
SetApplicationLoggerLogLevel 86
FB_VisController 99
FC_CheckCmd 101
FC_CheckCmd2 102
FC_EtResultToString 103
FC_GetDateTimeAsArray 104
FC_InitStateModelChangeStates 104
FC_InitStateModelExistingStates 106
FC_SetAlarm 108
FC_SetWarning 110
FC_StateSelectionAsDint 111
FC_StateSelectionAsDintFromBool 111
FR_<BackgroundFrame> 114
FR_<State> 116
FR_Aborted 116
FR_Aborting 116
FR_Alarm 118
FR_AlarmHistory 118
FR_Clearing 116
FR_Complete 116
FR_Completing 116
FR_Control1024x768 114
FR_Control800x600 114
FR_CurrentModeAndStateTime 119
FR_DynStateModel 120
FR_DynStateModel_2022 121
FR_Execute 116
FR_Held 116
FR_Holding 116
FR_Idle 116
FR_ModeAndStateTime 122
FR_Nav1024x768 114
FR_Nav800x600 114
FR_NavControl1024x768 114
FR_NavControl800x600 114
FR_PackMLStateModel_2022_Base 123
FR_PackMLStateModel_2022_UnitModeManager 124
FR_ProdConsumedCount 125
FR_ProdDefectiveCount 126
FR_ProdProcessedCount 127
FR_Resetting 116
FR_Starting 116
FR_StatesDisabled 129
FR_Stopped 116
FR_Stopping 116
FR_StopReason 128
FR_Suspended 116
FR_Suspending 116
FR_Undefined 116
FR_UnHolding 116
FR_UnSuspending 116
FR_Warning 128

G

GCL 57
PackML 57
GetDefinedUnitModes 89
GPL 58
PackML 58
GVL (Global Variables List) 61

H

Held 96
Holding 95

I

Idle 92
IF_StateCommands 64
CmdAbort 69
CmdClear 70
CmdHold 66
CmdReset 64
CmdStart 65
CmdStop 66
CmdSuspend 68
CmdUnHold 67
CmdUnSuspend 68
StateComplete 70
IF_StateModelHandler 72
IF_UnitMode 63

P

PackML
GCL 57
GPL 58
PackML base state model 80
properties
FB_UnitModeManager2 81

Q

q_etResult 23
q_sResultMsg 23
q_xError 23

R

RegisterLoggerPoint 84
RemoveUnitMode 88
Resetting 98
Restarting 96

S

SetApplicationLoggerLogLevel 86
ST_Administration 33
ST_Alarm 42
Starting 92
StateComplete 70
States frames 116
ST_Command 35
ST_CountDescrip 43
ST_Cumulative_Times 43
ST_Descriptor 44
ST_EquipmentInterlock 45
ST_Ingredient 45
ST_Ingredients 46
ST_InitAlarm 37
ST_Interface 46
ST_ModeState_Times 47
Stopped 91
Stopping 94
ST_Parameter_DINT 47
ST_Parameter_LREAL 48
ST_Parameter_REAL 49
ST_Parameter_STRING 49
ST_ProcessVariables 50
ST_Product 51
ST_Product_Data 51
ST_Recipe 52
ST_StateInfo 53
ST_Status 36
ST_Timestamp 53
ST_UnitModeDefinition 38
ST_VisInterface 39
Suspended 93
Suspending 97

U

Unsuspending 97

V

Visualization Frames 117

Schneider Electric
35 rue Joseph Monier
92500 Rueil Malmaison
France

www.se.com

As standards, specifications, and design change from time to time, please ask for confirmation of the information given in this publication.

© 2025 Schneider Electric. All rights reserved.

EIO0000002809.03