

EcoStruxure Machine Expert

PackML

Library Guide

EIO0000002809.01
12/2023

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 Book	11
General Information	17
Presentation of the Library	18
General Information	18
Operation Modes	20
PackTags	22
Diagnostic Concept	24
Diagnostic Concept	24
Common Inputs and Outputs	26
Behavior of Function Blocks with the Input <i>i_xEnable</i>	26
Data Unit Types	27
Enumerations	28
<i>ET_Cmd</i>	28
<i>ET_Modes</i>	29
<i>ET_Result</i>	29
<i>ET_StateModelDefinition</i>	31
<i>ET_States</i>	32
Structures	34
PackTags Main Structures	34
<i>ST_Administration</i>	34
<i>ST_Command</i>	36
<i>ST_Status</i>	37
Other Main Structures	38
<i>ST_InitAlarm</i>	38
<i>ST_UnitModeDefinition</i>	39
<i>ST_VisInterface</i>	40
SubStructures	43
<i>ST_Alarm</i>	43
<i>ST_CountDescrip</i>	44
<i>ST_Cumulative_Times</i>	44
<i>ST_Descriptor</i>	45
<i>ST_EquipmentInterlock</i>	46
<i>ST_Ingredient</i>	46
<i>ST_Ingredients</i>	47
<i>ST_Interface</i>	47
<i>ST_ModeState_Times</i>	48
<i>ST_Parameter_DINT</i>	48
<i>ST_Parameter_LREAL</i>	49
<i>ST_Parameter_REAL</i>	50
<i>ST_Parameter_STRING</i>	50
<i>ST_ProcessVariables</i>	51
<i>ST_Product</i>	52

<i>ST_Product_Data</i>	52
<i>ST_Recipe</i>	53
<i>ST_StateInfo</i>	54
<i>ST_Timestamp</i>	54
Aliases - <i>DateTimeArray</i>	56
<i>DateTimeArray</i>	56
Global Variables	57
Global Constants List	58
GCL	58
Global Parameter List	59
GPL	59
Interfaces	62
<i>IF_UnitMode</i>	63
<i>IF_UnitMode</i> – General Information	63
<i>IF_StateCommands</i>	64
<i>IF_StateCommands</i> – General Information	64
<i>CmdReset</i> (Method)	64
<i>CmdStart</i> (Method)	65
<i>CmdStop</i> (Method)	66
<i>CmdHold</i> (Method)	66
<i>CmdUnHold</i> (Method)	67
<i>CmdSuspend</i> (Method)	68
<i>CmdUnSuspend</i> (Method)	68
<i>CmdAbort</i> (Method)	69
<i>CmdClear</i> (Method)	70
<i>StateComplete</i> (Method)	70
<i>IF_StateModelHandler</i>	72
<i>IF_StateModelHandler</i> – General Information	72
Program Organization Units (POU)	73
Function Blocks	74
<i>FB_DataManagement</i>	74
<i>FB_ModeManager</i>	76
<i>FB_UnitModeManager2</i>	79
<i>FB_UnitModeManager2</i> – General Information	79
PackML Base State Model	80
Properties	81
<i>DefineUnitMode</i> (Method)	81
<i>DefineUnitModeWithHandler</i> (Method)	83
<i>RegisterLoggerPoint</i> (Method)	84
<i>SetApplicationLoggerLogLevel</i> (Method)	85
<i>ExecuteCurrentState</i> (Method)	86
<i>CmdChangeUnitMode</i> (Method)	87
<i>RemoveUnitMode</i> (Method)	88
<i>GetDefinedUnitModes</i> (Method)	88
<i>Command</i> (Method)	89
<i>FB_StateModelHandlerBase</i>	89
<i>FB_StateModelHandlerBase</i> – General Information	89
<i>Clearing</i> (Method)	90
<i>Stopped</i> (Method)	91
<i>Starting</i> (Method)	91

<i>Idle</i> (Method)	92
<i>Suspended</i> (Method)	92
<i>Execute</i> (Method)	93
<i>Stopping</i> (Method)	93
<i>Aborting</i> (Method)	94
<i>Aborted</i> (Method)	94
<i>Holding</i> (Method)	95
<i>Held</i> (Method)	95
<i>Restarting</i> (Method)	96
<i>Suspending</i> (Method)	96
<i>Unsuspending</i> (Method)	97
<i>Resetting</i> (Method)	97
<i>Completing</i> (Method)	98
<i>Complete</i> (Method)	98
<i>FB_VisController</i>	99
Functions	101
<i>FC_CheckCmd</i>	101
<i>FC_CheckCmd2</i>	102
<i>FC_EtResultToString</i>	103
<i>FC_GetDateTimeAsArray</i>	104
<i>FC_InitStateModelChangeStates</i>	104
<i>FC_InitStateModelExistingStates</i>	106
<i>FC_SetAlarm</i>	108
<i>FC_SetWarning</i>	110
Visualization	112
BackgroundFrames	113
<i>FR_<BackgroundFrame></i>	113
States Frames	115
<i>FR_<State></i>	115
Visualization Frames	116
General	116
<i>FR_Alarm</i>	117
<i>FR_AlarmHistory</i>	117
<i>FR_CurrentModeAndStateTime</i>	118
<i>FR_DynStateModel</i>	119
<i>FR_DynStateModel_2022</i>	120
<i>FR_ModeAndStateTime</i>	121
<i>FR_ProdConsumedCount</i>	122
<i>FR_ProdDefectiveCount</i>	123
<i>FR_ProdProcessedCount</i>	124
<i>FR_Warning</i>	125
<i>FR_StopReason</i>	125
<i>FR_StatesDisabled</i>	126
Glossary	129
Index	130

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.

<p>⚠ WARNING</p>
<p>UNGUARDED EQUIPMENT</p> <ul style="list-style-type: none">• 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. <p>Failure to follow these instructions can result in death, serious injury, or equipment damage.</p>

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 Book

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 V2.2.

Available Languages of this Document

This 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 Machine Solutions	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/).

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 POU's 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 POU's 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 POU's.
- 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.


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 POU's 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.

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.

Standard	Description
EN 61496-1:2013	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
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 24

Common Inputs and Outputs..... 26

Presentation of the Library

What’s in This Chapter

General Information..... 18

Operation Modes..... 20

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 EcoStruxure Machine Expert, Functions and Libraries User Guide)
Forward compatible library	Yes (FCL (see EcoStruxure Machine Expert, Functions and Libraries User Guide))

NOTE: For this library, qualified-access-only is set. The POU's, 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.

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

Step	Action	Comment
1	In the EcoStruxure Machine Expert Logic Builder, execute the command New Project .	–
2	In the New Project dialog box, select From Example from the Project type list.	–
3	On the right-hand side of the New Project dialog box, select All from the Controller list.	Result: Available examples are listed in the Matching Examples text box.
4	Select your example from the Matching Examples list.	–
5	Enter a name for the new project, and select the file location.	–
6	Click the OK button.	Result: A new project is created based on the selected example.

General Considerations

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 environment or your machines 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.
- Monitor activities within your systems.
- Prevent subject devices from direct access or direct link by unauthorized parties or unauthenticated actions.
- 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.

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.

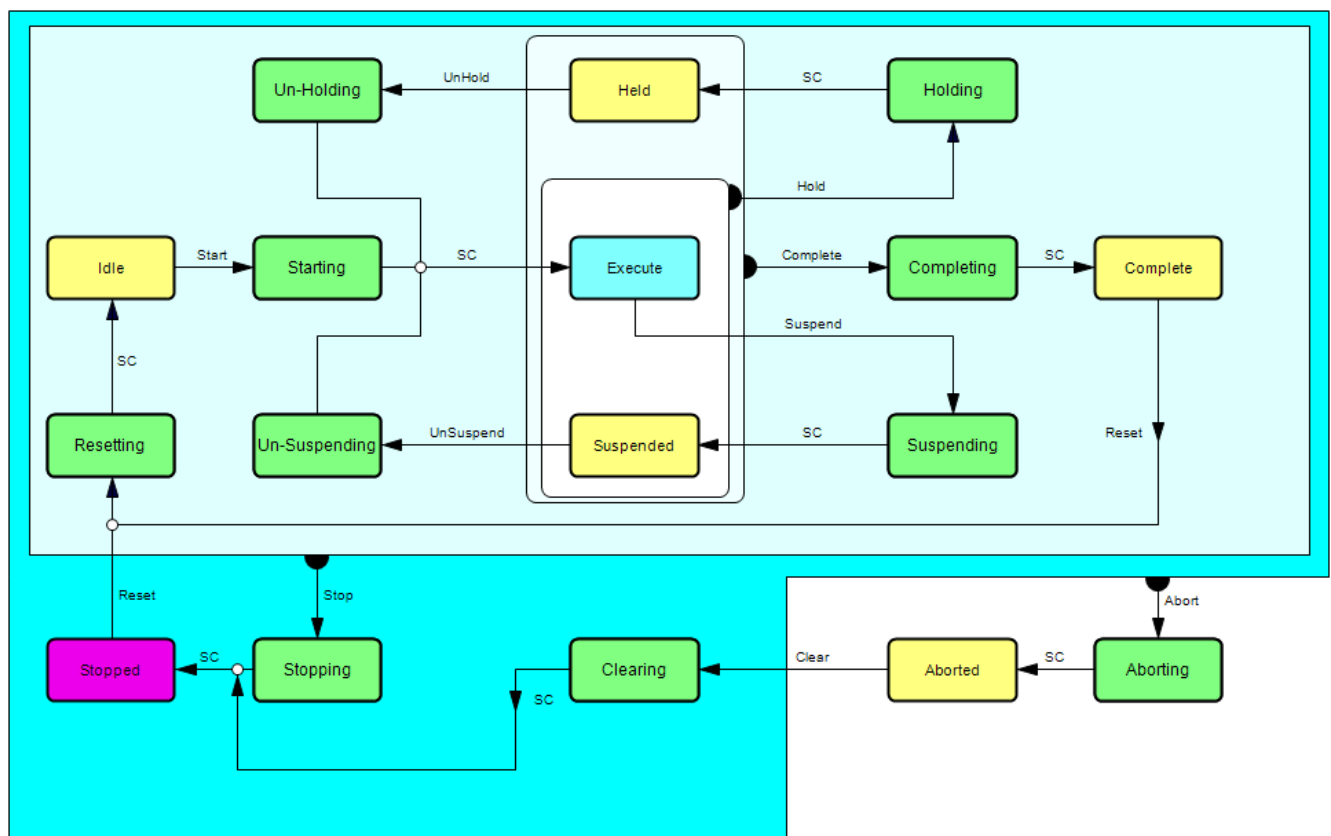
Visualization frames and *FB_VisController* are used to visualize the state model and provide a commissioning interface for it.

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. POU's 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.

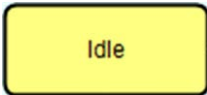

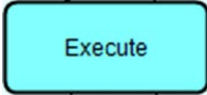
State Model Diagram

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.	<i>ST_Command</i>
status	These variables contain status information (such as which operation mode is active, which state is active, and so on).	<i>ST_Status</i>
administration	These variables contain important OEE data (for example, downtime, reasons for downtime, and so on). Furthermore production information.	<i>ST_Administration</i>

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	24
--------------------------	----

Diagnostic Concept

Overview

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

In principle, the diagnostic information has the following layers:

1. General information on the exception. No specific knowledge about the POU functionality is necessary.
2. POU-specific diagnostic and status messages (part 1): Detailed information on the source triggering the diagnostic or status messages.
3. POU-specific diagnostic and status messages (part 2): 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:

- Online display of the diagnostic messages
- Detailed information on diagnostic events with the availability of the diagnostic codes
- Overview on the status or exceptional condition of a POU
- Pertinent suggestions to help correct the causes for exceptional conditions
- Enumerated diagnostic messages to facilitate multi-language support for HMI displays

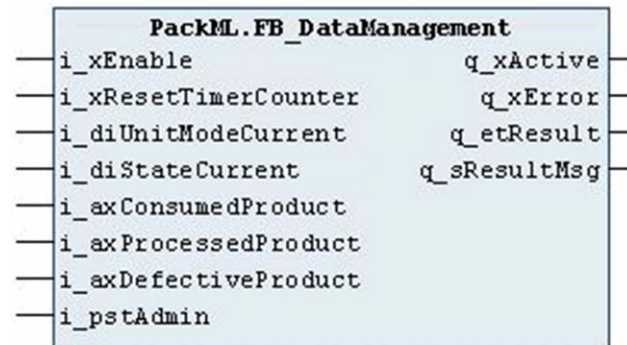
Diagnostic Outputs

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

Output	Data type	Meaning
<i>q_sResultMsg</i>	STRING	Provides additional information about the present status of the POU. If the POU is in error state (<i>q_xError</i> = 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 enumeration <i>ET_Result</i> which contains the sum of the possible <i>q_etResult</i> outputs across the complete library.		

Diagnostic information example:



Common Inputs and Outputs

What’s in This Chapter

Behavior of Function Blocks with the Input *i_xEnable*26

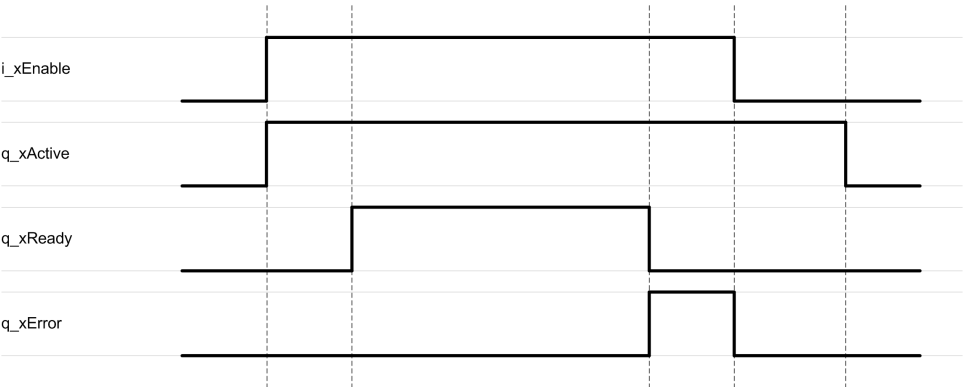
Behavior of Function Blocks with the Input *i_xEnable*

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

Enumerations28

Structures.....34

Aliases - *DateTimeArray*56

Enumerations

What's in This Chapter

<i>ET_Cmd</i>	28
<i>ET_Modes</i>	29
<i>ET_Result</i>	29
<i>ET_StateModelDefinition</i>	31
<i>ET_States</i>	32

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 20 can only be exited with a state transition command (for example *Start*).

Enumeration Elements

Name	Data type	Value	Description
<i>Undefined</i>	DINT	0	Commands to transition from one state to another are described in <i>FC_CheckCmd</i> .
<i>Reset</i>	DINT	1	
<i>Start</i>	DINT	2	
<i>Stop</i>	DINT	3	
<i>Hold</i>	DINT	4	
<i>UnHold</i>	DINT	5	
<i>Suspend</i>	DINT	6	
<i>UnSuspend</i>	DINT	7	
<i>Abort</i>	DINT	8	
<i>Clear</i>	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
<i>Undefined</i>	DINT	0	The operation mode is undefined or invalid.
<i>Producing</i> ⁽¹⁾	DINT	1	Operation mode for production.
<i>Maintenance</i>	DINT	2	Operation mode for maintenance.
<i>Manual</i>	DINT	3	Manual operation mode
(1) With the 2015 revision of the ANSI/ISA TR88.00.02-2015, the control mode <i>Producing</i> was changed to <i>Production</i> . For reasons of compatibility with the previous version of the PackML library, the name <i>Producing</i> 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 <code>q_xError</code> of a function block, page 74 is FALSE, one of the following status messages is shown.			
<i>Ok</i>	DINT	0	The POU is OK, but not running.
<i>Disabled</i>	DINT	1	The POU is disabled.
<i>Initializing</i>	DINT	2	The POU is being initialized.
<i>Running</i>	DINT	4	The POU is being executed.
<i>ResettingTimerAndCounter</i>	DINT	5	The POU is resetting the timer and counter.
If <code>q_xError</code> of a function block, page 74 is TRUE, one of the following error messages is shown.			
<i>UnitModeCurrentRange</i>	DINT	11	The value of the input <code>i_diUnitModeCurrent</code> is out of range. The value of operation mode must be in the range 0 and <code>Gc_uiMaxNumberOfModes</code> .
<i>StateCurrentRange</i>	DINT	12	The value of the input <code>i_diStateCurrent</code> is out of range. The value of state must be in the range 0 and <code>Gc_uiMaxNumberOfStates</code> .
<i>NumberOfModesRange</i>	DINT	13	Number of operation modes is out of range. The number of operation modes must be in the range 0 and <code>Gc_uiMaxNumberOfModes</code> .
<i>PointerInitModeInvalid</i>	DINT	14	The initial operation mode pointer <code>i_pstInitMode</code> is invalid.
<i>PointerAdminInvalid</i>	DINT	15	The admin pointer <code>i_pstAdmin</code> is invalid.
<i>PointerStatusInvalid</i>	DINT	16	The status pointer <code>i_pstStatus</code> is invalid.
<i>UnitModeRange</i>	DINT	17	The operation mode out of range. Range should be: $0 < i_diUnitMode \leq diNumberOfModes$
<i>InvalidStateModelNoExecuteState</i>	DINT	20	The state model of the operation mode is invalid: there is no Execute state.
<i>InvalidStateModelNoStoppedState</i>	DINT	21	The state model of the operation mode is invalid: there is no Stopped state.
<i>InvalidStateModelNoHeldState</i>	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.
<i>InvalidStateModelNoCompleteState</i>	DINT	23	The state model of the operation mode is invalid: if Completing state exists, a Complete state must be present.
<i>InvalidStateModelNoSuspendedState</i>	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.
<i>InvalidStateModelNoAbortedState</i>	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.
<i>InvalidStateModelNoIdleState</i>	DINT	26	The state model of the operation mode is invalid: the Resetting state can be used only in connection with the Idle state.
<i>InvalidStateModelNoResettingState</i>	DINT	27	The state model of the operation mode is invalid: the Idle state can be used only in connection with the Resetting state.
<i>ModeChangeRequestRejected</i>	DINT	28	The request to change the operation mode is rejected.
<i>StateChangeRequestRejected</i>	DINT	29	The triggered state command is not accepted in the present state.
<i>UnknownResult</i>	DINT	30	The value of the input <code>i_etResult</code> of the function <code>FC_EtResultToString</code> is unknown.
<i>InvalidInput</i>	DINT	33	At least one of the specified inputs is invalid.
<i>MaxNumberOfUnitModesExceeded</i>	DINT	34	The maximum number of unit modes is defined. Remove a unit mode before adding a new one.

Name	Data type	Value	Description
<i>NoUnitMode</i>	DINT	35	No unit mode is active. The commanded operation is not allowed.
<i>UnitModeNotFound</i>	DINT	36	The specified unit mode is not defined.
<i>UnitModeActive</i>	DINT	37	Unit mode is active. The commanded operation is not allowed.
<i>NoStateModelHandler</i>	DINT	39	The definition of the unit mode requires an instance of the <i>FB_StateModelHandlerBase</i> .
<i>RegisterLoggerPointFailed</i>	DINT	41	The logger point could not be registered.
<i>AlreadyExists</i>	DINT	42	Unit mode with same value or name is already defined.
<i>NotReady</i>	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
<i>AbortedAndStopped</i>	DINT	000000001000000100 bin	The bits for the states <i>Aborted</i> and <i>Stopped</i> are TRUE. Can be used to define the states where a mode change is allowed.
<i>NoStateDisabled</i>	DINT	0 bin	None of the bits are TRUE, all states are enabled. Can be used to define the states which are disabled.
<i>HoldDisabled</i>	DINT	1110000000000 bin	The bits for the states <i>Holding</i> , <i>Held</i> and <i>UnHolding</i> are TRUE. Can be used to define the states which are disabled.
<i>SuspendDisabled</i>	DINT	110000000100000 bin	The bits for the states <i>Suspending</i> , <i>Suspended</i> and <i>UnSuspending</i> are TRUE. Can be used to define the states which are disabled.
<i>CompleteDisabled</i>	DINT	110000000000000000 bin	The bits for the states <i>Completing</i> and <i>Complete</i> are TRUE. Can be used to define the states which are disabled.
<i>Clearing</i>	DINT	2 hex	Bit 1 represents the state <i>Clearing</i> .
<i>Stopped</i>	DINT	4 hex	Bit 2 represents the state <i>Stopped</i> .
<i>Starting</i>	DINT	8 hex	Bit 3 represents the state <i>Starting</i> .
<i>Idle</i>	DINT	10 hex	Bit 4 represents the state <i>Idle</i> .
<i>Suspended</i>	DINT	20 hex	Bit 5 represents the state <i>Suspended</i> .
<i>Execute</i>	DINT	40 hex	Bit 6 represents the state <i>Execute</i> .
<i>Stopping</i>	DINT	50 hex	Bit 7 represents the state <i>Stopping</i> .
<i>Aborting</i>	DINT	100 hex	Bit 8 represents the state <i>Aborting</i> .
<i>Aborted</i>	DINT	200 hex	Bit 9 represents the state <i>Aborted</i> .
<i>Holding</i>	DINT	400 hex	Bit 10 represents the state <i>Holding</i> .
<i>Held</i>	DINT	800 hex	Bit 11 represents the state <i>Held</i> .
<i>UnHolding</i>	DINT	1000 hex	Bit 12 represents the state <i>UnHolding</i> .
<i>Suspending</i>	DINT	2000 hex	Bit 13 represents the state <i>Suspending</i> .
<i>Unsuspending</i>	DINT	4000 hex	Bit 14 represents the state <i>Unsuspending</i> .
<i>Resetting</i>	DINT	8000 hex	Bit 15 represents the state <i>Resetting</i> .
<i>Completing</i>	DINT	10000 hex	Bit 16 represents the state <i>Completing</i> .
<i>Complete</i>	DINT	20000 hex	Bit 17 represents the state <i>Complete</i> .

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 TR88.00.02-2022.

Enumeration Elements

NOTE: For a detailed description of each of the individual states, refer to the 2015 revision of the standard ANSI/ISA TR88.00.02-2022.

Name	Data type	Value	Description
<i>Undefined</i>	DINT	0	No state is defined.
<i>Clearing</i>	DINT	1	Acting state Clearing
<i>Stopped</i>	DINT	2	Wait state Stopped
<i>Starting</i>	DINT	3	Acting state Starting
<i>Idle</i>	DINT	4	Wait state Idle
<i>Suspended</i>	DINT	5	Wait state Suspended
<i>Execute</i>	DINT	6	Acting state Execute
<i>Stopping</i>	DINT	7	Acting state Stopping
<i>Aborting</i>	DINT	8	Acting state Aborting
<i>Aborted</i>	DINT	9	Wait state Aborted
<i>Holding</i>	DINT	10	Acting state Holding
<i>Held</i>	DINT	11	Wait state Held
<i>UnHolding</i>	DINT	12	Acting state Un-Holding
<i>Suspending</i>	DINT	13	Acting state Suspending
<i>UnSuspending</i>	DINT	14	Acting state Un-Suspending
<i>Resetting</i>	DINT	15	Acting state Resetting
<i>Completing</i>	DINT	16	Acting state Completing
<i>Complete</i>	DINT	17	Wait state Complete
NOTE: The contents of any given state is subject to your application.			

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	34
Other Main Structures	38
SubStructures	43

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 59.

Variable	Data type	Description
<i>Alarm</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfAlarms</i>] OF <i>ST_Alarm</i>	Array of <i>ST_Alarm</i>
<i>AlarmExtent</i>	DINT	Size of the alarm array
<i>AlarmHistory</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfAlarmHistory</i>] OF <i>ST_Alarm</i>	Array of given size for machine detected error code and messaging, page 43
<i>AlarmHistoryExtent</i>	DINT	Size of the alarm history array
<i>Warning</i>	ARRAY [1.. <i>GPL.Gc_uiMaxNumberOfWarnings</i>] OF <i>ST_Alarm</i>	Array contains the active and/or unacknowledged machine advisories
<i>WarningExtent</i>	DINT	Size of the advisory array
<i>MachDesignSpeed</i>	REAL	Machine design speed
<i>PLCDateTime</i>	<i>DateTimeArray</i>	Present date and time of the programmable logic controller
<i>StopReason</i> ⁽²⁾	ARRAY [1.. <i>GPL.Gc_uiMaxNumberOfStopReasons</i>] OF <i>ST_Alarm</i>	Information about the first event captured during an abort, held, suspended, or stop event
<i>StopReasonExtend</i> ⁽²⁾	DINT	Size of the <i>StopReason</i> array
<i>Parameter</i> ⁽²⁾	ARRAY [1.. <i>GPL.Gc_iNumOfAdminParameter</i>] OF <i>ST_Descriptor</i>	Array of <i>ST_Descriptor</i>
<i>ModeCurrentTime</i> ⁽²⁾	ARRAY [0.. <i>GPL.Gc_uiMaxNumberOfModes</i>] OF UDINT	Time spent in operation mode
<i>ModeCumulativeTime</i> ⁽²⁾	ARRAY [0.. <i>GPL.Gc_uiMaxNumberOfModes</i>] OF UDINT	Cumulative time in each operation mode

Variable	Data type	Description
<i>StateCurrentTime</i> ⁽²⁾	ARRAY [0.. <i>GPL.Gc_uiMaxNumberOfModes</i> ,0.. <i>GVL.Gc_uiMaxNumberOfStates</i>] OF UDINT	Value of last state timer in each operation mode/state
<i>StateCumulativeTime</i> ⁽²⁾	ARRAY [0.. <i>GPL.Gc_uiMaxNumberOfModes</i> ,0.. <i>GVL.Gc_uiMaxNumberOfStates</i>] OF UDINT	Cumulative time in each operation mode/state
<i>ProdConsumedCount</i> ⁽²⁾	ARRAY [1.. <i>GPL.Gc_uiNumberOfConsumedProducts</i>] OF <i>ST_CountDescrip</i>	Array of <i>ST_CountDescrip</i> (of products consumed in the production machine)
<i>ProdProcessedCount</i> ⁽²⁾	ARRAY [1.. <i>GPL.Gc_uiNumberOfProcessedProducts</i>] OF <i>ST_CountDescrip</i>	Array of <i>ST_CountDescrip</i> (of products processed by the production machine)
<i>ProdDefectiveCount</i> ⁽²⁾	ARRAY [1.. <i>GPL.Gc_uiNumberOfDefectiveProducts</i>] OF <i>ST_CountDescrip</i>	Array of <i>ST_CountDescrip</i> (of products that are marked as not acceptable in the production machine)
<i>AccTimeSinceReset</i> ⁽²⁾	DINT	Accumulative time since last <i>Reset</i> command
<i>StatesDisabled</i> ⁽²⁾	DINT	States 1...17 of the PackML State Model can be disabled by turning on the corresponding bit
<i>PACDateTime</i> ^{(1) (2)}	<i>ST_Timestamp</i>	Time stamp (ISO date and time data type, ISO 8601:1988 format), page 54
<i>Parameter_REAL</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Parameter_REAL</i>	Structured array of unit/machine parameter information for values with REAL data type
<i>Parameter_STRING</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Parameter_STRING</i>	Structured array of unit/machine parameter information for values with STRING data type
<i>Parameter_LREAL</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Parameter_LREAL</i>	Structured Array of unit/machine parameter information for values with LREAL data type
<i>Parameter_DINT</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Parameter_DINT</i>	Structured array of unit/machine parameter information for values with DINT data type
<i>StopReason2</i>	<i>ST_Alarm</i>	Information about the first event captured during an abort, held, suspended or stop event
<i>ModeTimeCurrent</i>	DINT	Present mode time
<i>StateTimeCurrent</i>	DINT	Present state time
<i>CumulativeTimes</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfCumulativeTimes</i>] OF <i>ST_Cumulative_Times</i>	Structured array of timer values
<i>ProductData</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfProcessedProducts</i>] OF <i>ST_Product_Data</i>	Structured array of product stream data
<i>DisabledStatesCfg</i>	ARRAY [1.. <i>GPL.Gc_uiMaxNumberOfModes</i>] OF DWORD	Array reflecting the disabled states configuration for defined modes
<i>CurDisabledStates</i>	DWORD	Reflects the disabled states for the present mode (<i>ST_Status.UnitModeCurrent</i>)
<i>EnabledModesCfg</i>	DWORD	Element reflecting the enabled modes configuration for the unit/machine
<i>ModeTransitionCfg</i>	ARRAY [1.. <i>GPL.Gc_uiMaxNumberOfModes</i>] OF DWORD	Array reflecting the mode transition configuration for the modes of the unit/machine
⁽¹⁾ Variable has been replaced by <i>PLCDateTime</i> 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 59.

Variable	Data type	Description
<i>UnitMode</i>	DINT	The target operation mode (see <i>ET_Modes</i>)
<i>UnitModeChangeRequest</i>	BOOL	TRUE = request to change the machine operation mode
<i>MachSpeed</i>	REAL	Machine present speed
<i>CntrlCmd</i>	DINT	Value of the command that provides the state command to drive a state change in the base state model.
<i>CmdChangeRequest</i>	BOOL	TRUE = request to change the machine state
<i>MaterialInterlocks</i> ⁽¹⁾	ARRAY [0..31] OF BOOL	Indicates materials are ready for processing.
<i>RemoteInterface</i> ⁽¹⁾	ARRAY [1.. <i>GPL.Gc_uiNumberOfInterface</i>] OF <i>ST_Interface</i>	Used for coordinating upstream or downstream machines in a cell of multiple unit machines
<i>Parameter</i> ⁽¹⁾	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Descriptor</i>	Array of <i>ST_Descriptor</i>
<i>Product</i> ⁽¹⁾	ARRAY [1.. <i>GPL.Gc_uiNumberOfProducts</i>] OF <i>ST_Product</i>	Array of <i>ST_Product</i>
<i>MaterialInterlock</i>	DWORD	Materials ready
<i>Parameter_REAL</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Parameter_REAL</i>	Structured array of unit/machine parameter information for values with REAL data type
<i>Parameter_STRING</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Parameter_STRING</i>	Structured array of unit/machine parameter information for values with STRING data type
<i>Parameter_LREAL</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Parameter_LREAL</i>	Structured array of unit/machine parameter information for values with LREAL data type
<i>Parameter_DINT</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfParameter</i>] OF <i>ST_Parameter_DINT</i>	Structured array of unit/machine parameter information for values with DINT data type
<i>SelectedRecipe</i>	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
<i>RecipeChangeRequest</i>	BOOL	TRUE = request to change the recipe on the unit/machine
<i>Recipe</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfRecipes</i>] OF <i>ST_Recipe</i>	Structured array of recipe Information
⁽¹⁾ Variable kept for compatibility (according to ANSI/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 59.

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

Variable	Data type	Description
<i>RecipeChangeInProcess</i>	BOOL	An indicator for the user-defined recipe changeover process
<i>Recipe</i>	ARRAY [1.. <i>GPL.Gc_uiNumberOfRecipes</i>] OF <i>ST_Recipe</i>	Structured of recipe information
<i>Stacklight</i>	ARRAY [1..31 OF <i>DINT</i>	Indicator for the stack light status
(1) Variable kept for compatibility (according to ANSI/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
<i>diID</i>	DINT	Event message ID
<i>diValue</i>	DINT	Event message value
<i>sMessage</i>	STRING[<i>GPL.Gc_uiMaxLengthOfAlarmMessage</i>]	Event message
<i>xisInList</i>	BOOL	Event is in list
<i>xisAck</i>	BOOL	Event is acknowledged
<i>Category</i>	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
<i>xModeExistent</i>	BOOL	TRUE: operation mode is defined.
<i>stClearing</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stClearing</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stStopped</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stStopped</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stStarting</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stStarting</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stIdle</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stIdle</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stResetting</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stResetting</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stExecute</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stExecute</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stStopping</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stStopping</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stAborting</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stAborting</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stAborted</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stAborted</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stHolding</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stHolding</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stHeld</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stHeld</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stUnHolding</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stUnHolding</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stSuspended</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stSuspended</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stUnSuspending</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stUnSuspending</i> 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
<i>stSuspending</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stSuspending</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stCompleting</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stCompleting</i> is defined as possible state for the present operating mode and whether a change of the operating mode is allowed while in this state.
<i>stComplete</i>	<i>ST_StateInfo</i>	Indicates whether the state <i>stComplete</i> 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
<i>vq_xBlink</i>	BOOL	Variable used to flash the present Acting state
<i>vq_diNumberOfModes</i>	DINT	Maximum number of operation modes
<i>vq_diUnitModeCurrent</i>	DINT	Operation mode in use
<i>vq_diStateCurrent</i>	DINT	Present state
<i>vq_diModeCurrentTimeActive</i>	DINT	Present time in active operation mode
<i>vq_diModeCumulativeTimeActive</i>	DINT	Cumulative time in active operation mode
<i>vq_diStateCurrentTimeActive</i>	DINT	Present time in active state
<i>vq_diStateCumulativeTimeActive</i>	DINT	Cumulative time in active state
<i>vq_astAlarm</i>	ARRAY [0..10] OF <i>ST_Alarm</i>	Array of given size for active machine error code and messaging
<i>vq_astAlarmHistory</i>	ARRAY [0..10] OF <i>ST_Alarm</i>	Array of given size for machine error code and messaging
<i>vq_diProdConsumedId</i>	DINT	ID of consumed product
<i>vq_sProdConsumedName</i>	STRING [<i>GPL.Gc_uiMaxLengthOfParameter-Name</i>]	Name of consumed product
<i>vq_sProdConsumedUnit</i>	STRING [5]	Unit of consumed product

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

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

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

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

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

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
<i>ID</i>	DINT	ID of <i>CountDescrip</i>
<i>Name</i>	STRING[GPL.Gc_uiMaxLengthOfParameter-Name]	Name of <i>CountDescrip</i>
<i>Unit</i>	STRING[5]	String of <i>CountDescrip</i> units
<i>Count</i>	DINT	Count value of <i>CountDescrip</i>
<i>AccCount</i>	DINT	Accumulative count value of <i>CountDescrip</i>

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
<i>AccTimeSinceReset</i>	DINT	Accumulated time since last reset
<i>ModeStateTimes</i>	ARRAY [1.. <i>Gc_uiMaxNumOfModes</i>] OF <i>ST_ModeState_Times</i>	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
<i>ID</i>	DINT	ID of parameter
<i>Name</i>	STRING[<i>GPL_Gc_uiMaxLengthOfParameter-Name</i>]	Name of parameter
<i>Unit</i>	STRING[5]	String of parameter units
<i>Value</i>	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
<i>Blocked</i>	BOOL	TRUE: Indicates that a downstream system is not able to accept product.
<i>Starved</i>	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
<i>IngredientID</i>	DINT	Arbitrary number associated with the raw material, or ingredient for a particular product number.
<i>Parameter</i>	ARRAY [0.. <i>Gc_uiNumberOfIngredientParameter</i>] OF <i>ST_Descriptor</i>	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
<i>Parameter_REAL</i>	ARRAY [1.. <i>GPL.Gc_uiNumOfParamREALRecipeIngredients</i>] OF <i>ST_Parameter_REAL</i>	Structured array of ingredient information for values with REAL data type
<i>Parameter_STRING</i>	ARRAY [1.. <i>GPL.Gc_uiNumOfParamSTRINGRecipeIngredients</i>] OF <i>ST_Parameter_STRING</i>	Structured array of ingredient information for values with STRING data type
<i>Parameter_LREAL</i>	ARRAY [1.. <i>GPL.Gc_uiNumOfParamLREALRecipeIngredients</i>] OF <i>ST_Parameter_LREAL</i>	Structured array of ingredient information for values with LREAL data type
<i>Parameter_DINT</i>	ARRAY [1.. <i>GPL.Gc_uiNumOfParamDINTRecipeIngredients</i>] OF <i>ST_Parameter_DINT</i>	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
<i>Number</i>	DINT	Identification number of upstream or downstream machine
<i>ControlCmdNumber</i>	DINT	Control command number for upstream or downstream machine
<i>CmdValue</i>	DINT	Command value associated with the control command number.
<i>Parameter</i>	ARRAY [0.. <i>GPL.Gc_uiNumOfParam</i>] OF <i>ST_Descriptor</i>	Array of <i>ST_Descriptor</i> , parameter tags associated to the remote interface

Used By

- *ST_Command*
- *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
<i>Mode</i>	DINT	Represents the cumulative amount of time (in seconds) spent in each mode since the last timer and counter reset was executed
<i>State</i>	ARRAY [1.. <i>GPL.Gc_uiMaxNumberOfModes</i>] 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
<i>ID</i>	DINT	ID of the parameter
<i>Name</i>	STRING(GPL.Gc_uiMaxLengthOfParameter-Name)	Name of the parameter
<i>Unit</i>	STRING[6]	Unit of the parameter
<i>Value</i>	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
<i>ID</i>	DINT	ID of the parameter
<i>Name</i>	STRING(GPL.Gc_uiMaxLengthOfParameter-Name)	Name of the parameter
<i>Unit</i>	STRING[6]	Unit of the parameter
<i>Value</i>	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
<i>ID</i>	DINT	ID of the parameter
<i>Name</i>	STRING(GPL.Gc_uiMaxLengthOfParameter-Name)	Name of the parameter
<i>Unit</i>	STRING[6]	Unit of the parameter
<i>Value</i>	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
<i>ID</i>	DINT	ID of the parameter
<i>Name</i>	STRING(GPL.Gc_uiMaxLengthOfParameter-Name)	Name of the parameter
<i>Unit</i>	STRING[6]	Unit of the parameter
<i>Value</i>	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
<i>Parameter_REAL</i>	ARRAY [1.. GPL.Gc_uiNumOfParamREALRecipeProcessVar] OF ST_Parameter_REAL	Structured array of recipe process variable information for values with REAL data type
<i>Parameter_STRING</i>	ARRAY [1.. GPL.Gc_uiNumOfParamSTRINGRecipeProcessVar] OF ST_Parameter_STRING	Structured array of recipe process variable information for values with STRING data type
<i>Parameter_LREAL</i>	ARRAY [1.. GPL.Gc_uiNumOfParamLREALRecipeProcessVar] OF ST_Parameter_LREAL	Structured array of recipe process variable information for values with LREAL data type
<i>Parameter_DINT</i>	ARRAY [1.. GPL.Gc_uiNumOfParamDINTRecipeProcessVar] 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
<i>ProductID</i>	DINT	ID of the product, used to indicate to the machine which product it is producing.
<i>ProcessVariable</i>	ARRAY [0.. <i>GPL.Gc_uiNumberOfProcessVariable</i>] OF <i>ST_Descriptor</i>	Structured array describes the specific process variables, which are needed or were used by the machine to process the specified product.
<i>Ingredients</i>	ARRAY [0.. <i>GPL.Gc_uiNumberOfIngredients</i>] OF <i>ST_Ingredient</i>	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_Command*
- *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
<i>ID</i>	DINT	ID number for each product stream
<i>Name</i>	STRING	Name of product stream
<i>Unit</i>	STRING[6]	Structured array of ingredients process variable information for values with LREAL data type
<i>PrimaryQty</i>	REAL	Product primary quantity value of each product stream
<i>ConsumedCount</i>	DINT	Number of consumables used of each product stream
<i>ProcessedCount</i>	DINT	Number of produced goods of each product stream
<i>DefectiveCount</i>	DINT	Number of products marked as not acceptable of each product stream
<i>AccConsumedCount</i>	DINT	Accumulated number of consumables used of each product stream since last reset
<i>AccProcessedCount</i>	DINT	Accumulated number of produced goods of each product stream since last reset
<i>AccDefectiveCount</i>	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
<i>ID</i>	DINT	ID of <i>ST_Recipe</i>
<i>Name</i>	STRING	Name of <i>ST_Recipe</i>
<i>Unit</i>	STRING[6]	Unit of <i>ST_Recipe</i>
<i>PrimaryQty</i>	REAL	Primary quantity value of the recipe
<i>ProcessVariables</i>	<i>ST_ProcessVariables</i>	Process variables for the recipe
<i>Ingredients</i>	<i>ST_Ingredients</i>	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
<i>xStateExistent</i>	BOOL	TRUE: Indicates that the state is defined for the given operation mode.
<i>xModeChangePossible</i>	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
<i>AlmDate</i>	DATE	Contains the timespan in seconds since 01.01.1970, 00:00 clock.
<i>AlmTime</i>	TOD	Contains the timespan in milliseconds since 00:00 clock.

Used By

- *ST_Administration*
- *ST_Alarm*

Aliases - *DateTimeArray*

What's in This Chapter

DateTimeArray.....56

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
<i>DateTimeArray</i>	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] = μ s (fraction of seconds)

Global Variables

What’s in This Part

Global Constants List 58

Global Parameter List 59

Global Constants List

What’s in This Chapter

GCL58

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
<i>Gc_sLibraryVersion</i>	STRING[80]	Vx.x.x.0 ¹	Library version
<i>Gc_uiMaxNumberOfStates</i>	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 59

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

Variable	Data type	Value	Description
<i>Gc_uiNumberOfIngredientParameter</i>	UINT[1..65535]	50	Number of parameters per ingredient, determines the array <i>ST_Ingredients.Parameter</i>
<i>Gc_uiNumberOfIngredients</i>	UINT[1..65535]	10	Number of ingredients per product, determines the array <i>ST_Product.Ingredients</i>
<i>Gc_uiMaxNumberOfAlarms</i>	UINT[1..65535]	100	Maximum number of alarms, determines the array <i>ST_Administration.Alarm</i>
<i>Gc_uiNumberOfAlarmHistory</i>	UINT[1..65535]	100	Maximum number of alarms in history, determines the array <i>ST_Administration.AlarmHistory</i>
<i>Gc_uiMaxNumberOfModes</i>	UINT[1..65535]	31	Maximum number of modes
<i>Gc_uiMaxLengthOfParameterName</i>	UINT[1..80]	80	Maximum length of parameter name in <i>ST_Descriptor</i>
<i>Gc_uiMaxNumberOfWarnings</i>	UINT[1..65535]	100	Maximum number of advisories, determines the array <i>ST_Administration.Warning</i>
<i>Gc_uiMaxLengthOfAlarmMessage</i>	UINT[1..255]	80	Maximum length of an alarm message specified with <i>ST_Alarm</i>
<i>Gc_udiBgColorAlarmActive</i>	UDINT	FFCF hex	Used for visualization, RGB color code for active alarms, FFCF hex = light red
<i>Gc_udiBgColorAlarmInactive</i>	UDINT	FFF80 hex	Used for visualization, RGB color code for inactive and not acknowledged alarms, FFF80 hex = light yellow
<i>Gc_udiBgColorAlarmAcknowledged</i>	UDINT	AAD7FF hex	Used for visualization, RGB color code for inactive and acknowledged alarms, AAD7FF hex = light blue
<i>Gc_udiBgColorAlarmHistoryUnacknowledged</i>	UDINT	FFF80 hex	Used for visualization, RGB color code for unacknowledged alarms in history, FFF80 hex = light yellow
<i>Gc_udiBgColorWarningActive</i>	UDINT	FFCF hex	Used for visualization, RGB color code for active advisories, FFCF hex = light red
<i>Gc_udiBgColorWarningInactive</i>	UDINT	FFF80 hex	Used for visualization, RGB color code for inactive and not acknowledged advisories, FFF80 hex = light yellow
<i>Gc_udiBgColorWarningAcknowledged</i>	UDINT	AAD7FF hex	Used for visualization, RGB color code for inactive and acknowledged advisories, AAD7FF hex = light blue
<i>Gc_uiNumberOfParameterREAL</i>	UINT	10	Number of parameters of type REAL
<i>Gc_uiNumberOfParameterSTRING</i>	UINT	10	Number of parameters of type STRING
<i>Gc_uiNumberOfParameterLREAL</i>	UINT	10	Number of parameters of type LREAL
<i>Gc_uiNumberOfParameterDINT</i>	UINT	10	Number of parameters of type DINT
<i>Gc_uiNumberOfRecipe</i>	UINT	3	Number of recipes
<i>Gc_uiNumberOfCumulativeTimes</i>	UINT	10	Number of cumulative times

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

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 Information63

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
<i>sName</i>	STRING	Read	Indicates the name of the present unit mode.
<i>diValue</i>	DINT	Read	Indicates the numeric value of the present unit mode.
<i>diStatesDisabled</i>	DINT	Read	Provides information about the state model. The bits 0...16 of the DINT represent the individual states as defined in <i>ET_StateModelDefinition</i> . The bit value TRUE indicates that the corresponding state is disabled in the state model.
<i>diStatesMode-ChangeAllowed</i>	DINT	Read	Provides information about the state model. The bits 0...16 of the DINT represent the individual states as defined in <i>ET_StateModelDefinition</i> . The bit value TRUE indicates that a mode change in the corresponding state is allowed.
<i>etInitialState</i>	<i>ET_States</i> , page 32	Read	Indicates the initial state for the configured state model.

IF_StateCommands

What's in This Chapter

<i>IF_StateCommands</i> – General Information	64
<i>CmdReset</i> (Method)	64
<i>CmdStart</i> (Method)	65
<i>CmdStop</i> (Method)	66
<i>CmdHold</i> (Method)	66
<i>CmdUnHold</i> (Method)	67
<i>CmdSuspend</i> (Method)	68
<i>CmdUnSuspend</i> (Method)	68
<i>CmdAbort</i> (Method)	69
<i>CmdClear</i> (Method)	70
<i>StateComplete</i> (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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdReset</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdStart</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdStop</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdHold</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdUnHold</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdSuspend</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdUnSuspend</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdAbort</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdClear</i>	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 from an acting state to the successor state according to the state model.

Interface

Output	Data type	Description
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>StateComplete</i>	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

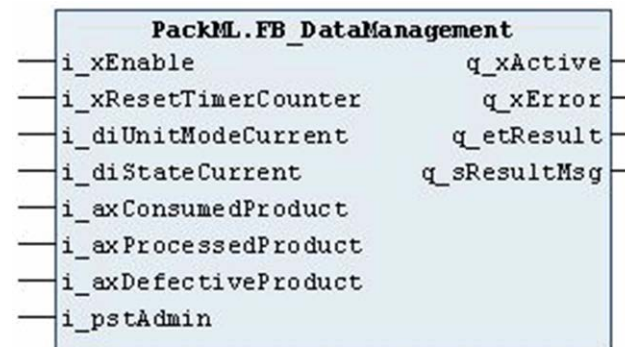
What's in This Chapter

<i>FB_DataManagement</i>	74
<i>FB_ModeManager</i>	76
<i>FB_UnitModeManager2</i>	79
<i>FB_StateModelHandlerBase</i>	89
<i>FB_VisController</i>	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 29 and ET_States, page 32.

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>i_xEnable</i>	BOOL	Activation and initialization of the function block.
<i>i_xResetTimerCounter</i>	BOOL	Upon a rising edge, the timer and counter values are reset.
<i>i_diUnitModeCurrent</i>	DINT	The present operation mode is passed to the function block via this input. <i>Status.UnitModeCurrent</i> should be applied to the input.
<i>i_diStateCurrent</i>	DINT	The present machine state is passed to the function block via this input. <i>Status.StateCurrent</i> should be applied to the input.
<i>i_axConsumedProduct</i>	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>i_axProcessedProduct</i>	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>i_axDefectiveProduct</i>	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>i_pstAdmin</i>	POINTER TO <i>ST_Administration</i>	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.

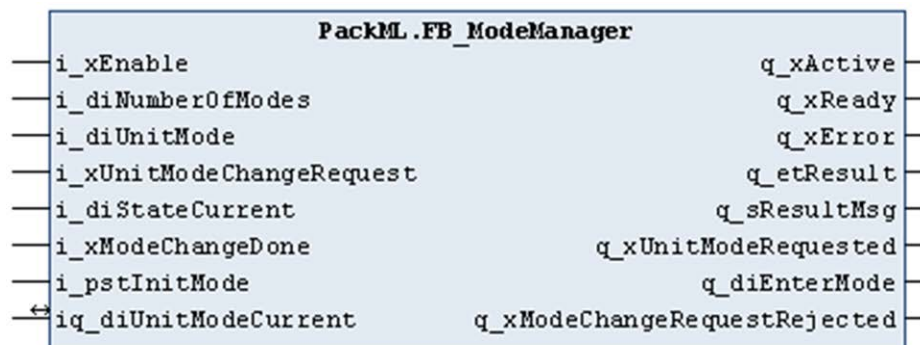
Failure to follow these instructions can result in death, serious injury, or equipment damage.

Output	Data type	Description
<i>q_xActive</i>	BOOL	If this output is set to TRUE, the function block is active.
<i>q_xError</i>	BOOL	If this output is set to TRUE, an error was detected. Refer to <i>ET_Result</i> .
<i>q_etResult</i>	<i>ET_Result</i>	Enumeration with the result.
<i>q_sResultMsg</i>	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>i_xEnable</i>	BOOL	Activation and initialization of the function block.
<i>i_diNumberOfModes</i>	DINT	Number of operation modes If the value is changed, a reinitialization of the function block is required.
<i>i_diUnitMode</i>	DINT	Requested operation mode The PackTag <i>Command.UnitMode</i> should be applied to the input.
<i>i_xUnitModeChangeRequest</i>	BOOL	Upon a rising edge, the function block verifies whether a change of operation mode is possible. The PackTag <i>Command.UnitModeChangeRequest</i> should be applied to the input.
<i>i_diStateCurrent</i>	DINT	Machine present state The PackTag <i>Status.StateCurrent</i> should be applied to the input.

Input	Data type	Description
<i>i_xModeChangeDone</i>	BOOL	Feedback from the application that the requested mode change has been executed. Upon a rising edge, the variable linked to <i>iq_diUnitModeCurrent</i> is updated accordingly.
<i>i_pstInitMode</i>	POINTER TO <i>ST_UnitModeDefinition</i>	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 <i>ST_UnitModeDefinition</i> . The indexes of the array correspond to the numeric value of the available control modes (refer to <i>ET_Modes</i>). Therefore the pointer must point to the index which is associated to the operation mode <i>Producing</i> , which is the first mode.

Input / output	Data type	Description
<i>iq_diUnitModeCurrent</i>	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 <i>Status.UnitModeCurrent</i> should be applied to the input/output.

Output	Data type	Description
<i>q_xActive</i>	BOOL	If this output is set to TRUE, the function block is active.
<i>q_xError</i>	BOOL	If this output is set to TRUE, an error has been detected. Refer to <i>ET_Result</i> .
<i>q_xReady</i>	BOOL	If this output is set to TRUE, the function block is ready for operation.
<i>q_etResult</i>	<i>ET_Result</i>	Enumeration with the result.
<i>q_sResultMsg</i>	STRING[80]	Additional result message.
<i>q_xUnitModeRequested</i>	BOOL	This reflects the input <i>i_xUnitModeChangeRequest</i> . The PackTag <i>Status.UnitModeRequested</i> should be applied to this output.
<i>q_diEnterMode</i>	DINT	This indicates the operation mode which is to be changed to. If the function block is deactivated, the output is set to 0.
<i>q_xModeChangeRequestRejected</i>	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
<i>CmdReset()</i>	Command accepted only in states <i>Stopped</i> and <i>Complete</i> . Reaction: Transition to the state <i>Resetting</i> .
<i>CmdStart()</i>	Command accepted only in states <i>Idle</i> or in state <i>Stopped</i> if the state <i>Idle</i> does not exist. Reaction: Transition to the state <i>Starting</i> .
<i>CmdHold()</i>	Command accepted only in states <i>Execute</i> or <i>Suspended</i> . Reaction: Transition to the state <i>Holding</i> .
<i>CmdUnHold()</i>	Command accepted only in state <i>Held</i> . Reaction: Transition to the state <i>Unholding</i> .
<i>CmdSuspend()</i>	Command accepted only in state <i>Execute</i> . Reaction: Transition to the state <i>Suspending</i> .
<i>CmdUnSuspend()</i>	Command accepted only in state <i>Suspended</i> . Reaction: Transition to the state <i>Unsuspending</i> .
<i>CmdComplete()</i>	Command accepted only in states <i>Execute</i> , <i>Held</i> , or <i>Suspended</i> . Reaction: Transition to the state <i>Completing</i> .
<i>CmdAbort()</i>	Command accepted in all states except <i>Aborting</i> and <i>Aborted</i> . Reaction: Transition to the state <i>Aborting</i> .
<i>CmdClear()</i>	Command accepted only in state <i>Aborted</i> . Reaction: Transition to the state <i>Clearing</i> .
<i>CmdStop()</i>	Command accepted in all states except <i>Aborting</i> , <i>Aborted</i> , <i>Clearing</i> , <i>Stopping</i> and <i>Stopped</i> . Reaction: Transition to the state <i>Stopping</i> .
<i>StateComplete()</i>	Command accepted only in states <i>Resetting</i> , <i>Starting</i> , <i>Holding</i> , <i>Unholding</i> , <i>Suspending</i> , <i>Unsuspending</i> , <i>Aborting</i> , <i>Clearing</i> , or <i>Stopping</i> . Reaction: Transition to the next state in accordance with the state model.

Multitask Applications

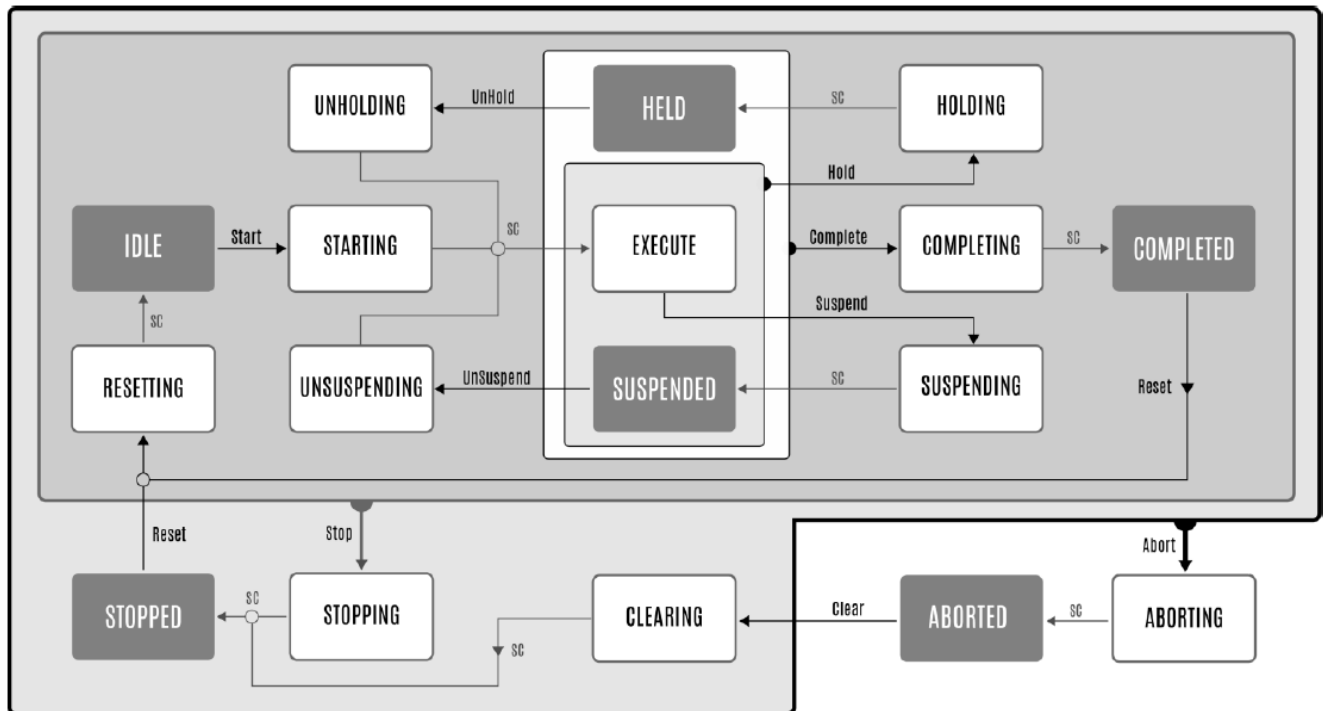
Due to the implementation using methods and properties it is possible that the individual methods are called by different tasks. To ensure complete processing of a command, the corresponding methods are implemented in such a way that a command is only accepted if no other command is in progress. If a 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
<i>ifUnitModeCurrent</i>	IF_UnitMode, page 63	Read	Interface of the present unit mode. This interface provides status information of the present unit control mode.
<i>etStateCurrent</i>	ET_States, page 32	Read	Indicates the present state.
<i>timUnitModeCurrentTime</i>	TIME	Read	Indicates the time elapsed in the present mode.
<i>timStateCurrentTime</i>	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>i_sName</i>	STRING	Specifies the name of the unit mode. Refer to Unit Control Modes, page 82.
<i>i_diValue</i>	UDINT	Numeric value of the unit mode. Refer to the table below. Refer to Unit Control Modes, page 82. Valid range: 1...31
<i>i_diStatesDisabled</i>	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 <i>ET_StateModelDefinition</i> , page 31. Refer to the rules for state model definition (Dependency Rules, page 83).
<i>i_diStatesModeChangeAllowed</i>	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 <i>ET_StateModelDefinition</i> , page 31.
<i>i_etInitialState</i>	<i>ET_States</i> , page 32	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 <i>ET_States.Stopped</i> .

Output	Data type	Description
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>DefineUnitMode</i>	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 <i>Producing</i> was changed to <i>Production</i> . For reasons of compatibility with the previous version of the PackML library, the name <i>Producing</i> 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>i_sName</i>	STRING	Refer to the interface of <i>DefineUnitMode (Method)</i> , page 81.
<i>i_diValue</i>	UDINT	
<i>i_diStatesDisabled</i>	DINT	
<i>i_diStatesModeChangeAllowed</i>	DINT	
<i>i_etInitialState</i>	<i>ET_States</i> , page 32	
<i>i_ifStateModelHandler</i>	<i>IF_StateModelHandler</i> , page 72	This parameter is used to pass the interface to the instance of the <i>FB_StateModelHandlerBase</i> .

Output	Data type	Description
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>DefineUnitModeWithHandler</i>	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 EcoStruxure Machine Expert, 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 <i>DefineUnitMode()</i> method is unsuccessful.	<i>APL.ET_LogLevel.Warning</i>
A state command is triggered.	<i>APL.ET_LogLevel.DebugMessage</i>
A state command is performed.	<i>APL.ET_LogLevel.StatusMessage</i>
A state command is rejected.	<i>APL.ET_LogLevel.Warning</i>
A mode change command is triggered.	<i>APL.ET_LogLevel.DebugMessage</i>
A mode change is performed.	<i>APL.ET_LogLevel.StatusMessage</i>
A mode change command is rejected.	<i>APL.ET_LogLevel.Warning</i>

Interface

Input	Data type	Description
<i>i_ifParent</i>	<i>APL.IF_LoggerPoint</i>	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 <i>APL.GVL.G_ifApplicationLogger</i> is defined as parent.
<i>i_sName</i>	STRING	The name of the logger point that is shown in the application logger.

Output	Data type	Description
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. If <i>q_xError</i> = FALSE, <i>q_etResult</i> provides status information. If <i>q_xError</i> = TRUE, <i>q_etResult</i> provides diagnostic/error information.
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>RegisterLoggerPoint</i>	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 EcoStruxure Machine Expert, ApplicationLogger Library Guide).

Interface

Input	Data type	Description
<i>i_etLogLevel</i>	<i>APL.ET_LogLevel</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. If <i>q_xError</i> = FALSE, <i>q_etResult</i> provides status information. If <i>q_xError</i> = TRUE, <i>q_etResult</i> provides diagnostic/error information.
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>SetApplicationLoggerLogLevel</i>	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>ExecuteCurrentState</i>	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>i_diValue</i>	<i>DINT</i>	Numeric value of the target unit control mode

Output	Data type	Description
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>CmdChangeUnit-Mode</i>	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>i_diValue</i>	<i>DINT</i>	Numeric value of the unit control mode

Output	Data type	Description
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>RemoveUnitMode</i>	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
<i>iq_</i> <i>astUnitModeDefinitions</i>	ARRAY [1..GPL.Gc_	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
<i>q_etResult</i>	<i>ET_Result</i> , page 29	Provides diagnostic and status information as a numeric value. <i>If q_xError = FALSE, q_etResult provides status information.</i> <i>If q_xError = TRUE, q_etResult provides diagnostic/error information.</i>
<i>q_sResultMsg</i>	STRING	Provides additional diagnostic and status information as a text message.

Return Value

Output	Data type	Description
<i>GetDefinedUnitModes</i>	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

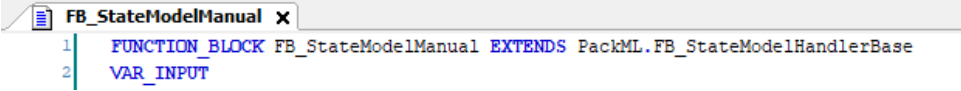
Overview

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.



```

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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xlsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xlsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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.

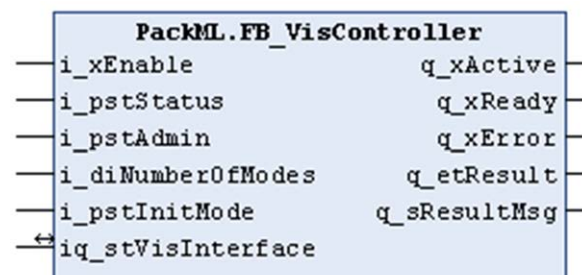
Interface

Input	Data type	Description
<i>i_xIsFirstCycle</i>	BOOL	Indicates if this is the first time the state method is invoked.
<i>i_ifCommands</i>	<i>IF_StateCommands</i> , page 64	Provides access to the state commands. This allows you to trigger a command from inside the state method.
<i>i_ifUnitMode</i>	<i>IF_UnitMode</i> , 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>i_xEnable</i>	BOOL	Activation and initialization of the function block.
<i>i_pstStatus</i>	POINTER TO <i>ST_Status</i>	Through this input, the pointer address to the PackTags of type <i>ST_Status</i> is passed to the function block.
<i>i_pstAdmin</i>	POINTER TO <i>ST_Administration</i>	Through this input, the pointer address to the PackTags of type <i>ST_Administration</i> is passed to the function block.

Input	Data type	Description
<i>i_diNumberOfModes</i>	DINT	Number of operation modes. If the value is changed, a reinitialization of the function block is required.
<i>i_pstInitMode</i>	POINTER TO <i>ST_UnitModeDefinition</i>	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 <i>ST_UnitModeDefinition</i> . The indexes of the array correspond to the numeric value of the available control modes (refer to <i>ET_Modes</i>). 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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	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
<i>q_xActive</i>	BOOL	If this output is set to TRUE, the function block is active.
<i>q_xReady</i>	BOOL	If this output is set to TRUE, the function block is ready for operation.
<i>q_xError</i>	BOOL	If this output is set to TRUE, an error was detected.
<i>q_etResult</i>	<i>ET_Result</i>	Result. Refer to <i>ET_Result</i> .
<i>q_sResultMsg</i>	STRING[80]	Additional result message.

Functions

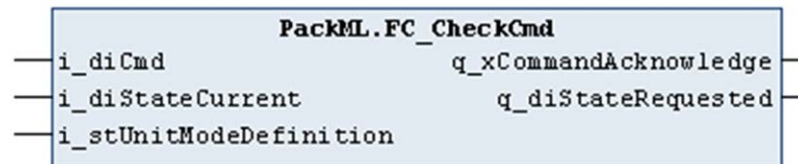
What's in This Chapter

<i>FC_CheckCmd</i>	101
<i>FC_CheckCmd2</i>	102
<i>FC_EtResultToString</i>	103
<i>FC_GetDateTimeAsArray</i>	104
<i>FC_InitStateModelChangeStates</i>	104
<i>FC_InitStateModelExistingStates</i>	106
<i>FC_SetAlarm</i>	108
<i>FC_SetWarning</i>	110

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
-	<i>Undefined</i>	Undefined, the output <i>q_xCommandAcknowledge</i> is FALSE.
Execute	<i>Reset</i>	Resetting if Complete and Completing do not exist.
Complete	<i>Reset</i>	Resetting , or Starting , if Resetting does not exist, or Execute , if Resetting and Starting does not exist.
Stopped		
Idle	<i>Start</i>	Starting , or Execute , if Starting does not exist.

Initial state	Transition command	Target state
Stopped	<i>Start</i> (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	<i>Stop</i>	Stopping , or Stopped , if Stopping does not exist.
Execute	<i>Hold</i>	Holding , or Held if Holding does not exist.
Held	<i>UnHold</i>	Un-Holding , or Execute if Un-Holding does not exist.
Execute	<i>Suspend</i>	Suspending , or Suspended if Suspending does not exist.
Suspended	<i>UnSuspend</i>	Un-Suspending , or Execute if Un-Suspending does not exist.
Any state except Aborted and Aborting	<i>Abort</i>	Aborting , or Aborted if Aborting does not exist.
Aborted	<i>Clear</i>	Clearing , or Stopped if Clearing does not exist.

Interface

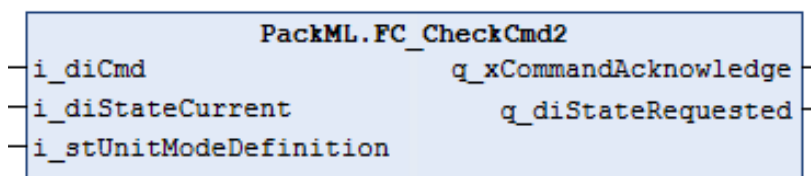
Inputs	Data type	Description
<i>i_diCmd</i>	DINT	The state transition command; the PackTag <i>ST_Command.CntrlCmd</i> should be applied to the input.
<i>i_diStateCurrent</i>	DINT	Machine present state; the PackTag <i>ST_Status.StateCurrent</i> should be applied to the input.
<i>i_stUnitModeDefinition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

Output	Data type	Description
<i>q_xCommandAcknowledge</i>	BOOL	Indicates TRUE if the given command is plausible.
<i>q_diStateRequested</i>	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	<i>Undefined</i>	Undefined , the output <i>q_xCommandAcknowledge</i> is FALSE.
Stopped or Complete	<i>Reset</i>	Resetting , or Idle if Resetting does not exist.
Stopped or Idle	<i>Start</i>	Command <i>Start</i> in state Stopped is supported only if Idle does not exist. Starting , or Execute if Starting does not exist.
Execute or Suspended	<i>Hold</i>	Holding , or Held if Holding does not exist.
Execute	<i>Suspend</i>	Suspending , or Suspended if Suspending does not exist.
Execute , Held or Suspended	<i>Complete</i>	Completing , or Completed if Completing does not exist.
Any state except Aborting , Aborted , Clearing , Stopping , and Stopped	<i>Stop</i>	Stopping , or Stopped if Stopping does not exist.
Any state except Aborting or Aborted	<i>Abort</i>	Aborting , or Aborted if Aborting does not exist.
Aborted	<i>Clear</i>	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>i_etResult</i>	<i>ET_Result</i>	Enumeration with the result.

Output	Data type	Description
<i>q_etResult</i>	<i>ET_Result</i>	Enumeration with the result.

Return Value

Data type	Description
STRING(80)	The <i>ET_Result</i> converted to a string value.

FC_GetDateTimeAsArray

Overview

Type:	Function
Available as of:	V1.1.0.0

PackML.FC_GetDateTimeAsArray
FC_GetDateTimeAsArray

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>i_xStopping</i>	BOOL	If this input is set to TRUE, the Stopping state is allowed to change the operation mode.
<i>i_xStopped</i>	BOOL	If this input is set to TRUE, the Stopped state is allowed to change the operation mode.
<i>i_xResetting</i>	BOOL	If this input is set to TRUE, the Resetting state is allowed to change the operation mode.
<i>i_xIdle</i>	BOOL	If this input is set to TRUE, the Idle state is allowed to change the operation mode.

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

Input / output	Data type	Description
<i>iq_stStateModel</i>	<i>ST_UnitModeDefinition</i>	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>i_xModeExistent</i>	BOOL	If this input is set to TRUE, the operation mode is available.
<i>i_xStopping</i>	BOOL	If this input is set to TRUE, the Stopping state is available.
<i>i_xStopped</i>	BOOL	If this input is set to TRUE, the Stopped state is available.
<i>i_xResetting</i>	BOOL	If this input is set to TRUE, the Resetting state is available.
<i>i_xIdle</i>	BOOL	If this input is set to TRUE, the Idle state is available.

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

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

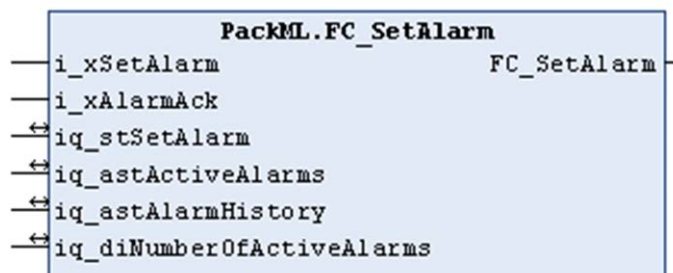
Return Value

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>i_xSetAlarm</i>	BOOL	<ul style="list-style-type: none"> <i>TRUE</i>: the reason code or machine-specific alarm message passed to the input/output <i>iq_stSetAlarm</i> is written into the tag at the input/output <i>iq_stActiveAlarm</i>. <i>FALSE</i>: the message is deleted from <i>iq_stActiveAlarms</i> and entered <i>iq_stAlarmHistory</i>.
<i>i_xAlarmAck</i>	BOOL	<i>TRUE</i> : the message has been detected. The appropriate time stamp for <i>Admin.Alarm[#].TimeAck</i> is determined.

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

Input/Output	Data type	Description
<i>iq_astAlarmHistory</i>	ARRAY [1..Gc_ uiNumberOfAlarmHis- tory] OF ST_Alarm	The administration tag <i>Admin.AlarmHistory[#]</i> should be applied to this input/output.
<i>iq_diNumberOfActiveAlarms</i>	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>i_xSetWarning</i>	BOOL	Request to write (TRUE) or remove (FALSE) the specified message into/ from the list linked to <i>iq_astWarnings</i> .
<i>i_xWarningAck</i>	BOOL	Request to acknowledge an active advisory.

Input/Output	Data type	Description
<i>iq_stSetWarning</i>	<i>ST_InitAlarm</i>	Specifies the advisory message to be treated by the function.
<i>iq_astWarnings</i>	ARRAY [1.. <i>Gc_uiMaxNumberOfWarnings</i>] OF <i>ST_Alarm</i>	The administration tag <i>Admin.Warning [#]</i> must be linked to this input/output.
<i>iq_diNumberOfActiveWarnings</i>	DINT	Provides the number of advisories in the list.

Visualization

What's in This Part

BackgroundFrames	113
States Frames	115
Visualization Frames	116

BackgroundFrames

What's in This Chapter

FR_<BackgroundFrame> 113

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
<i>FR_Control1024x768</i>	Control frame, size 1024x768 pixels
<i>FR_Nav1024x768</i>	Navigation frame, size 1024x768 pixels
<i>FR_NavControl1024x768</i>	Navigation and control frame, size 1024x768 pixels
<i>FR_Control800x600</i>	Control frame, size 800x600 pixels
<i>FR_Nav800x600</i>	Navigation frame, size 800x600 pixels
<i>FR_NavControl800x600</i>	Navigation and control frame, size 800x600 pixels

Interface

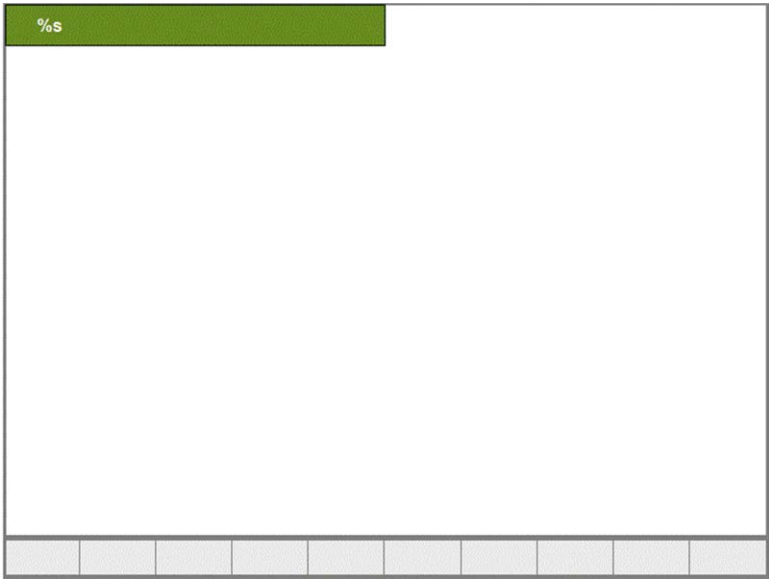
Input	Data type	Description
<i>i_sTitle</i>	STRING(80)	Title of the background frame

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> 115

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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

Example



Visualization Frames

What's in This Chapter

General	116
<i>FR_Alarm</i>	117
<i>FR_AlarmHistory</i>	117
<i>FR_CurrentModeAndStateTime</i>	118
<i>FR_DynStateModel</i>	119
<i>FR_DynStateModel_2022</i>	120
<i>FR_ModeAndStateTime</i>	121
<i>FR_ProdConsumedCount</i>	122
<i>FR_ProdDefectiveCount</i>	123
<i>FR_ProdProcessedCount</i>	124
<i>FR_Warning</i>	125
<i>FR_StopReason</i>	125
<i>FR_StatesDisabled</i>	126

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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

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
▼	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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

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
▼	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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

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.

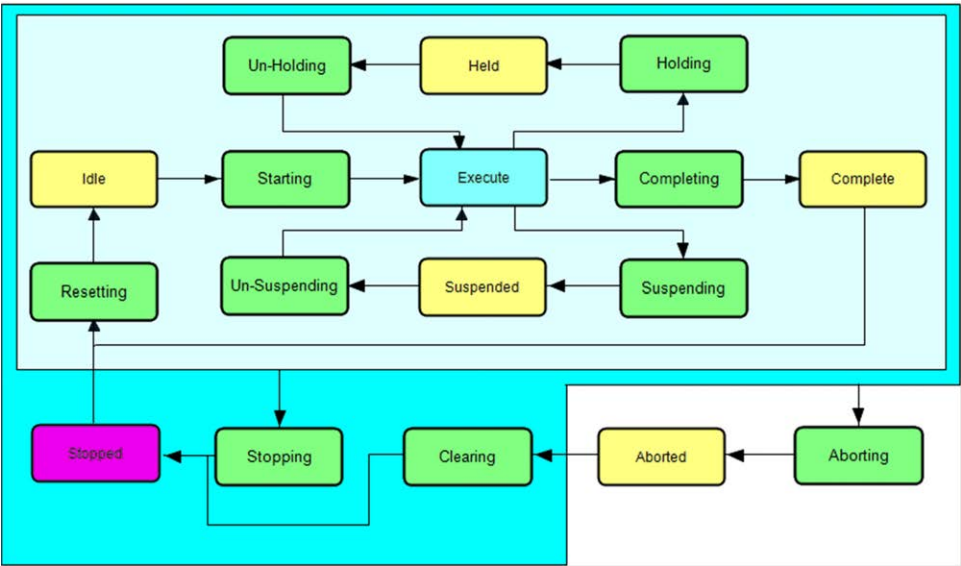
Functional Description

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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

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

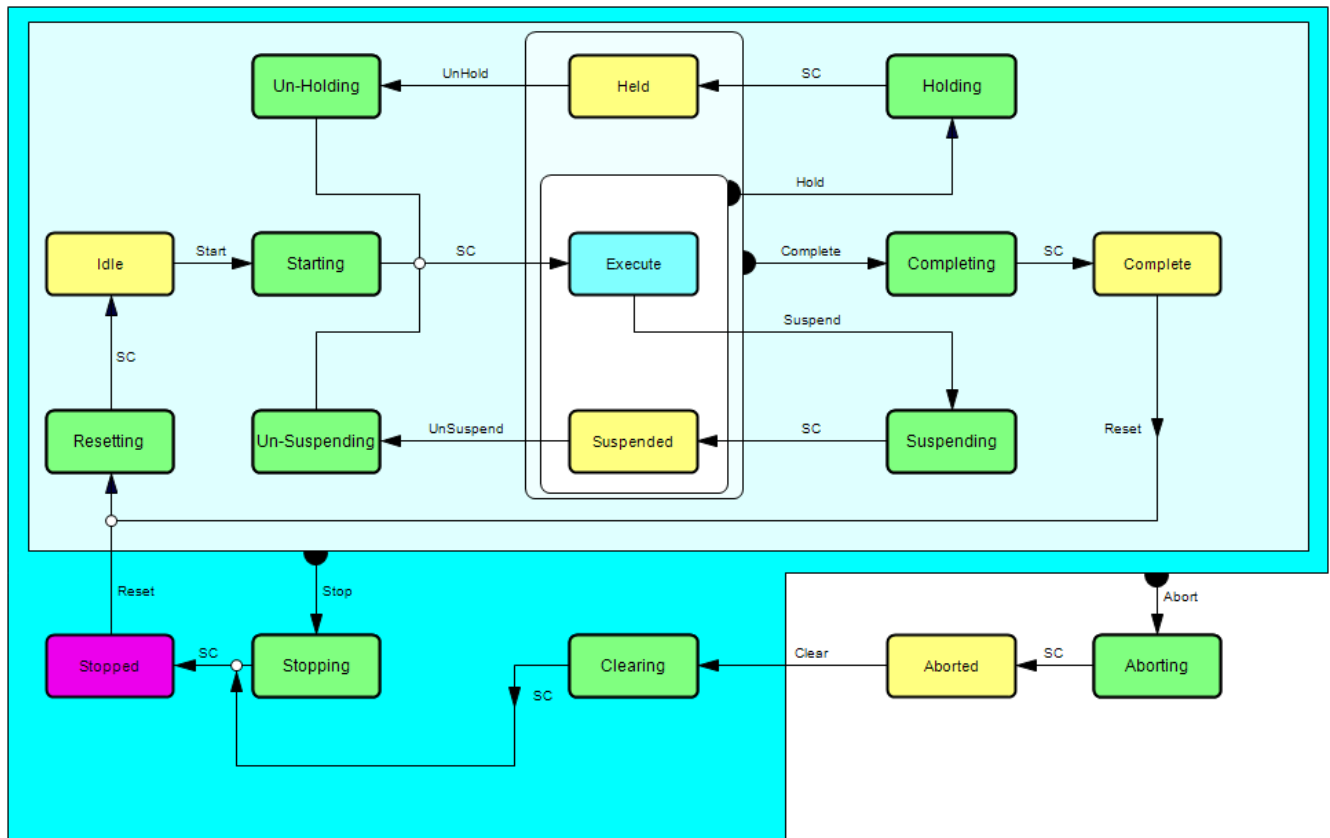
The *FR_DynStateModel_2022* dynamically displays the visualization of the state model during online operation. The state model is compliant to the PackML base state model as defined in ANSI/ISA TR88.00.02-2022.

Interface

Input / output	Data type	Description
<i>i_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

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:	VisuElem.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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

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_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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

Example

Product Consumed Counter [1] >	
ID	0
Name	
Unit	
Count	0
AccCount	0

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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

Example

Product Defective Counter [<input type="text" value="1"/>]	
ID	0
Name	
Unit	
Count	0
AccCount	0

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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

Example

Product Processed Counter [<input type="text" value="1"/>]	
ID	0
Name	
Unit	
Count	0
AccCount	0

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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

Example

#	ID	Value	Cat.	Message	Trigger	DateTime	AckDateTime
1	123	1	1	This Warning is active.	TRUE	2017 - 9 - 14 - 14:58:00.0	0 - 0 - 0 - 0:0:0.0
2	456	2	2	This warning is inactive but not acknowledged.	FALSE	2017 - 14 - 9 - 14:58:11.123	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 - 14:59:11.0
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
▼	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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

Example

ID	Value	Cat.	Message	DateTime
10	8	21	Abort command detected.	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
<i>iq_stVisInterface</i>	<i>ST_VisInterface</i>	Interface to the <i>FB_VisController</i>

Example

StatesDisabled	
<input type="checkbox"/>	Clearing
<input type="checkbox"/>	Stopped
<input type="checkbox"/>	Starting
<input type="checkbox"/>	Idle
<input type="checkbox"/>	Suspended
<input type="checkbox"/>	Execute
<input type="checkbox"/>	Stopping
<input type="checkbox"/>	Aborting
<input type="checkbox"/>	Aborted
<input type="checkbox"/>	Holding
<input type="checkbox"/>	Held
<input type="checkbox"/>	UnHolding
<input type="checkbox"/>	Suspending
<input type="checkbox"/>	UnSuspending
<input type="checkbox"/>	Resetting
<input type="checkbox"/>	Completing
<input type="checkbox"/>	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 calendar clock that operates continuously, even when the controller is not powered for the life of the battery.

Index

A		
Aborted	94	
Aborting.....	94	
B		
BackgroundFrames	113	
base state model PackML	80	
C		
Clearing.....	90	
CmdAbort	69	
CmdChangeUnitMode	87	
CmdClear	70	
CmdHold	66	
CmdReset	64	
CmdStart	65	
CmdStop	66	
CmdSuspend	68	
CmdUnHold	67	
CmdUnSuspend.....	68	
Command	89	
common inputs and outputs		
behavior of function blocks with the input <i>i_</i>		
<i>xEnable</i>	26	
Complete.....	98	
Completing	98	
D		
DateTimeArray	56	
DefineUnitMode	81	
DefineUnitModeWithHandler	83	
diagnostic concept.....	24	
E		
ET_Cmd	28	
abort	102	
Abort	28	
clear	102	
Clear	28	
hold	102	
Hold	28	
reset	101	
Reset	28	
start	101	
Start	28	
stop	102	
Stop	28	
suspend	102	
Suspend	28	
undefined	101	
Undefined.....	28	
UnHold.....	28, 102	
UnSuspend	28, 102	
ET_Modes	29	
Maintenance.....	29	
Manual.....	29	
Producing	29	
Undefined.....	29	
ET_Result	29	
AlreadyExists.....	31	
Disabled	30	
Initializing	30	
InvalidInput	30	
InvalidStateModelNoAbortedState	30	
InvalidStateModelNoCompleteState	30	
InvalidStateModelNoExecuteState	30	
InvalidStateModelNoHeldState	30	
InvalidStateModelNoIdleState	30	
InvalidStateModelNoResettingState	30	
InvalidStateModelNoStoppedState	30	
InvalidStateModelNoSuspendedState	30	
MaxNumberOfUnitModesExceeded	30	
ModeChangeRequestRejected	30	
NoStateModelHandler	31	
NotReady	31	
NoUnitMode	31	
NumberOfModesRange	30	
Ok	30	
PointerAdminInvalid	30	
PointerInitModelInvalid	30	
PointerStatusInvalid	30	
RegisterLoggerPointFailed	31	
ResettingTimerAndCounter	30	
Running	30	
StateChangeRequestRejected	30	
StateCurrentRange	30	
UnitModeActive	31	
UnitModeCurrentRange	30	
UnitModeNotFound	31	
UnitModeRange.....	30	
UnknownResult	30	
ET_StateModelDefinition	31	
Aborted	32	
AbortedAndStopped.....	32	
Aborting	32	
Clearing	32	
Complete	32	
CompleteDisabled	32	
Completing	32	
Execute.....	32	
Held	32	
HoldDisabled	32	
Holding	32	
Idle	32	
NoStateDisabled.....	32	
Resetting.....	32	
Starting	32	
Stopped	32	
Stopping.....	32	
SuspendDisabled.....	32	
Suspended	32	
Suspending	32	
UnHolding	32	
Unsuspending	32	
ET_States	32	
Aborted	33	
Aborting	33	
Clearing	33	
Complete	33	
Completing	33	
Execute.....	33	
Held	33	
Holding	33	
Idle	33	
Resetting.....	33	
Starting	33	
Stopped	33	
Stopping.....	33	

Suspended	33
Suspending	33
Undefined	33
UnHolding	33
UnSuspending	33
Execute	93
ExecuteCurrentState	86

F

FB_DataManagement	74
FB_ModeManager	76
FB_StateModelHandlerBase	89
Aborted	94
Aborting	94
Clearing	90
Complete	98
Completing	98
Execute	93
Held	95
Holding	95
Idle	92
Resetting	97
Restarting	96
Starting	91
Stopped	91
Stopping	93
Suspended	92
Suspending	96
Unsuspending	97
FB_UnitModeManager2	79
CmdChangeUnitMode	87
Command	89
DefineUnitMode	81
DefineUnitModeWithHandler	83
ExecuteCurrentState	86
GetDefinedUnitModes	88
properties	81
RegisterLoggerPoint	84
RemoveUnitMode	88
SetApplicationLoggerLogLevel	85
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
FR_<BackgroundFrame>	113
FR_<State>	115
FR_Aborted	115
FR_Aborting	115
FR_Alarm	117
FR_AlarmHistory	117
FR_Clearing	115
FR_Complete	115
FR_Completing	115
FR_Control1024x768	113
FR_Control800x600	113
FR_CurrentModeAndStateTime	118
FR_DynStateModel	119
FR_DynStateModel_2022	120
FR_Execute	115
FR_Held	115
FR_Holding	115
FR_Idle	115
FR_ModeAndStateTime	121

FR_Nav1024x768	113
FR_Nav800x600	113
FR_NavControl1024x768	113
FR_NavControl800x600	113
FR_ProdConsumedCount	122
FR_ProdDefectiveCount	123
FR_ProdProcessedCount	124
FR_Resetting	115
FR_Starting	115
FR_StatesDisabled	126
FR_Stopped	115
FR_Stopping	115
FR_StopReason	125
FR_Suspended	115
FR_Suspending	115
FR_Undefined	115
FR_UnHolding	115
FR_UnSuspending	115
FR_Warning	125

G

GCL	58
PackML	58
GetDefinedUnitModes	88
GPL	59
PackML	59

H

Held	95
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	58
GPL	59
PackML base state model	80
properties	
FB_UnitModeManager2	81

Q

q_etResult	24
q_sResultMsg	24
q_xError	24

R

RegisterLoggerPoint.....	84
RemoveUnitMode	88
Resetting	97
Restarting	96

S

SetApplicationLoggerLogLevel	85
ST_Administration	34
ST_Alarm	43
Starting	91
StateComplete	70
States frames	115
ST_Command	36
ST_CountDescrip	44
ST_Cumulative_Times	44
ST_Descriptor	45
ST_EquipmentInterlock	46
ST_Ingredient	46
ST_Ingredients	47
ST_InitAlarm	38
ST_Interface	47
ST_ModeState_Times	48
Stopped	91
Stopping	93
ST_Parameter_DINT	48
ST_Parameter_LREAL	49
ST_Parameter_REAL	50
ST_Parameter_STRING	50
ST_ProcessVariables	51
ST_Product	52
ST_Product_Data	52
ST_Recipe	53
ST_StateInfo	54
ST_Status	37
ST_Timestamp	54
ST_UnitModeDefinition	39
ST_VisInterface	40
Suspended	92
Suspending	96

U

Unsuspending	97
--------------------	----

V

Visualization Frames	116
----------------------------	-----

Schneider Electric
35 rue Joseph Monier
92500 Rueil Malmaison
France

+ 33 (0) 1 41 29 70 00

www.se.com

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

© 2023 Schneider Electric. All rights reserved.

EIO0000002809.01