



# **Guidelines for Packaging Machinery Automation Version 3.1**

**Release Date: 5/11/06**

**Open  
Modular  
Architecture  
Controls**



# Open Modular Architecture Controls (OMAC) Users Group

E-mail [omac@ARCweb.com](mailto:omac@ARCweb.com), [www.omac.org](http://www.omac.org)

This document is published by the Packaging Workgroup within the OMAC Users Group. Please submit any comments to:

Jim Ramsey, Hershey Foods

Phone (717) 534-6432 Email: [jramsey@hersheys.com](mailto:jramsey@hersheys.com)

The following individuals and organizations contributed to this document:

Dr. Kenneth Ryan	Alexandria Technical College	Sentiken Can	Nestle – Ralston
Spencer Beckett	Ames Engineering Corp	Juergen Feyerherd	Optima Packaging
Craig Fridley	Anheuser-Busch	Tom Egan	Packaging Machinery Manufacturer's Institute (PMMI)
Bill Krah	ARC Advisory Group	Fred Hayes	PMMI
Dennis Danniels	ARC Advisory Group	Tom Egan	PMMI
Sal Spada	ARC Advisory Group	Scott Bivens	Packaging Technologies
Chuck Padvorac	Beckhoff	David Newcorn	Packaging World
Gerd Hoppe	Beckhoff	Mike Nager	Phoenix Contact
Keith Hopkins	Bosch Packaging	Elco van der Wal	PLCOpen
Doug Davis	Bosch Rexroth Corp	Jay Joyner	Procter & Gamble
Joe Biondo	Bosch Rexroth Corp	Rob Aleksa	Procter & Gamble
Daniel Throne	Bosch Rexroth Corp	Rick Van Dyke	Frito-Lay
Scott Hibbard	Bosch Rexroth Corp	Bob Martell	Pfizer
Brad Weinshenker	Bosch Rexroth Corp	Clifford Still	R.A. Jones
Jeffrey Bock	Bosch Rexroth Corp.	Rick Stachel	R.A. Jones
Don Baechtler	Cleveland Motion Control	Rick Lidington	Lidington Associates
Joe Faust	Douglas Machine	Rick Morse	Rockwell Automation
Thomas Cord	Elau	Dave Whittenton	Rockwell Automation
John Kowal	Elau	Bob Hirschinger	Rockwell Automation
Klaus Weyer	Elau	Dave VanGompel	Rockwell Automation
Tom Jensen	Elau	Mike Wagner	Rockwell Automation
Patrick Hugg	Elau	Bernd Junker	Rovema
Bryan McGovern	Emerson Electric	Jay Clark	ROY-G-BIV Corp.
Gerard Barnhoorn	Heineken	Garth Basson	SAB Miller
Eric Rumi	Zarpac	Larry Trunek	SAB Miller
Wade Latz	Hershey Foods Corp	Richard Covarrubia	Schneider Electric
Joseph Wagner	Hershey Foods Corp	Pete Squires	Schneider Packaging Equipment
Edmund Amanor	Hershey Foods Corp	Klas Hellmann	Siemens
Gary Downey	Hershey Foods Corp	Keith Spiro	Siemens
Herman Rhoads	Hershey Foods Corp	John Ryan	Siemens
Jim Ramsey	Hershey Foods Corp	Hannes Wanner	SIG Pack Systems
Dennis Rose	Klockner Bartelt, Inc	Patrick Dell 'Olivio	SIG Pack Systems
James Conn	Streamfeeder	Shane Loughlin	SL Controls
Steve Ford	M&M Mars	Istvan Ulvros	Tetrapak
David Falcon	M&M Mars	Steven Laycock	Unilever
Fred Putnam	Markem Corp.	Andrew McDonald	Unilever
Chris Sullivan	Markem Corp.	John Downie	Yaskawa Electric
Paul Mills	Markem Corp.	Katherine Voss	Whedco (GE Fanuc)
John Michaloski	National Institute of Standards & Technology (NIST)		

**Copyright © 2006  
OMAC Users Group**

**Disclaimer**

The information contained within this document is intended as a guideline for the control and automation of packaging machinery and systems by end users, equipment builders, and technology suppliers. The guideline is a work-in-progress and will be updated regularly, but no more frequently than 3 times per year following the general meetings of the Workgroup.

While every effort has been made to ensure the accuracy and the completeness of the information presented in this report, the OMAC Users Group accepts no liability whatsoever for consequences of any actions taken upon the findings of the report.

**Permission to Use**

Any OMAC member may use these guidelines for the preparation of specifications. Non members, publishers, press, and others may quote the content of this document with the reference: “used with permission of the OMAC Packaging Workgroup; [www.omac.org](http://www.omac.org)”

# Table of Contents

Executive Overview .....	7
Benefits of Applying Digital Motion Controls to Packaging Machines.....	7
Introduction.....	8
Definitions.....	8
Line Types.....	8
Packaging Line Types.....	9
PackML State Model.....	9
Figure 1: Machine State Model for Automatic Mode Operation.....	11
Table 1: Automatic Operations Machine States .....	12
Table .....	13
PackTags .....	16
System Architecture .....	16
Figure 1: Logical Connectivity of PC-Based System Architecture.....	17
Figure 2: Logical Connectivity of Controller-Based System Architecture.....	17
Figure 3: Logical Connectivity of Drive-Based System Architecture.....	18
Figure 4: Architecture/Connectivity Descriptions.....	18
Figure 5: Architecture/Connectivity of Drive-Based Systems.....	19
Figure 6: Architecture/Connectivity of Controller-Based Systems .....	19
Figure 7: Architecture/Connectivity of PC-Based Systems .....	20
Figure 8: Functionality by Motion Interface Type .....	21
Figure 9: Digital Servo Communication Technology Comparison.....	23
Components.....	24
Programming Language .....	24
Skills Required .....	25
Benefits to be Achieved.....	26
Figure 10: Benefits of Applying Digital Motion Controls to Packaging Machines.....	26
Appendix IA: Glossary of Motion Control Terms v1.0.....	27
Appendix IIA: Line Types v1.1 .....	47
Packaging Line Types.....	48
Figure 2: Typical Packing Line Arrangement .....	49
Figure 3: Line Type 1 Example.....	50
Figure 4: Example of Line Type 2A .....	52
Figure 5: Line Type 2B Example .....	53
Figure 6: Example of Line Type 3 .....	54
Figure 7: Example of Line Type 4 .....	55
Appendix IIB: PackML State Model v2.2: Automatic Mode Machine States Definition .....	58
Figure 1: Machine State Model for Automatic Mode Operation.....	61
Table 1: Automation Operations Machine States.....	62
Table.....	63
Appendix IIC: PackML State Model S88 Software Compatibility .....	67
Comparison of S88 and Pack.....	68
S88 Commands and their Effect in the Pack.....	69
S88 State Model.....	71
Pack .....	71
Appendix IID: PackTags - Tag Naming Guidelines v 2.0.....	72
CONTROL .....	86
Reason Text .....	88
Appendix III: PackAL Packaging Application Function Block Library v1.01 .....	91
Figure 1: Scope of definitions, embracing existing and emerging standards .....	92
Figure 2: Signal Naming for edge- and level triggered input signals .....	94

<b>Table 1: Defined Function Blocks of PackAL</b> .....	<b>97</b>
<b>Figure 3: PS_wind_csv timing diagram</b> .....	<b>99</b>
<b>Figure 4: PS_Wind_ct timing diagram</b> .....	<b>100</b>
<b>Figure 5: PS_Dancer Control timing diagram</b> .....	<b>102</b>
<b>Figure 6: PS_Dancer Control Structure</b> .....	<b>102</b>
<b>Figure 7: PS_Registration timing diagram</b> .....	<b>104</b>
<b>Figure 8: PS_Reg_Correction timing diagram</b> .....	<b>106</b>
<b>Figure 9: PS_Indexing timing diagram</b> .....	<b>108</b>
<b>Table 2: PS_Index Position table</b> .....	<b>108</b>
<b>Table 3: Digital PLS Structure table</b> .....	<b>109</b>
<b>Table 4: Digital PLS Track table</b> .....	<b>110</b>
<b>Table 5: Digital PLS Switch Position table</b> .....	<b>110</b>
<b>Figure 10: Digital PLS timing diagram</b> .....	<b>111</b>
<b>Figure 11: Digital PLS, detailed description of Switch01</b> .....	<b>111</b>
<b>Figure 12: Digital PLS duration switch timing diagram</b> .....	<b>112</b>
<b>Figure 13: PS_SetOverride timing diagram</b> .....	<b>113</b>
<b>Figure 14: PS_Jog Axis timing diagram</b> .....	<b>115</b>
<b>Figure 15: Flying Sync Geometries</b> .....	<b>115</b>
<b>Figure 16: PS_FlyingSync timing diagram for a single synchronization</b> .....	<b>118</b>
<b>Figure 17: PS_GearInDyn structure of gear model</b> .....	<b>119</b>
<b>Figure 18: PS_OrientedStop graphical explanation of behavior</b> .....	<b>121</b>
<b>Figure 19: PackML State Model</b> .....	<b>122</b>
<b>Figure 20: PS_Pack_ML_State_Model FB Names and application adoption</b> .....	<b>124</b>
<b>Appendix V: Reserved</b> .....	<b>133</b>
<b>Appendix VIA: Benefit Examples</b> .....	<b>133</b>

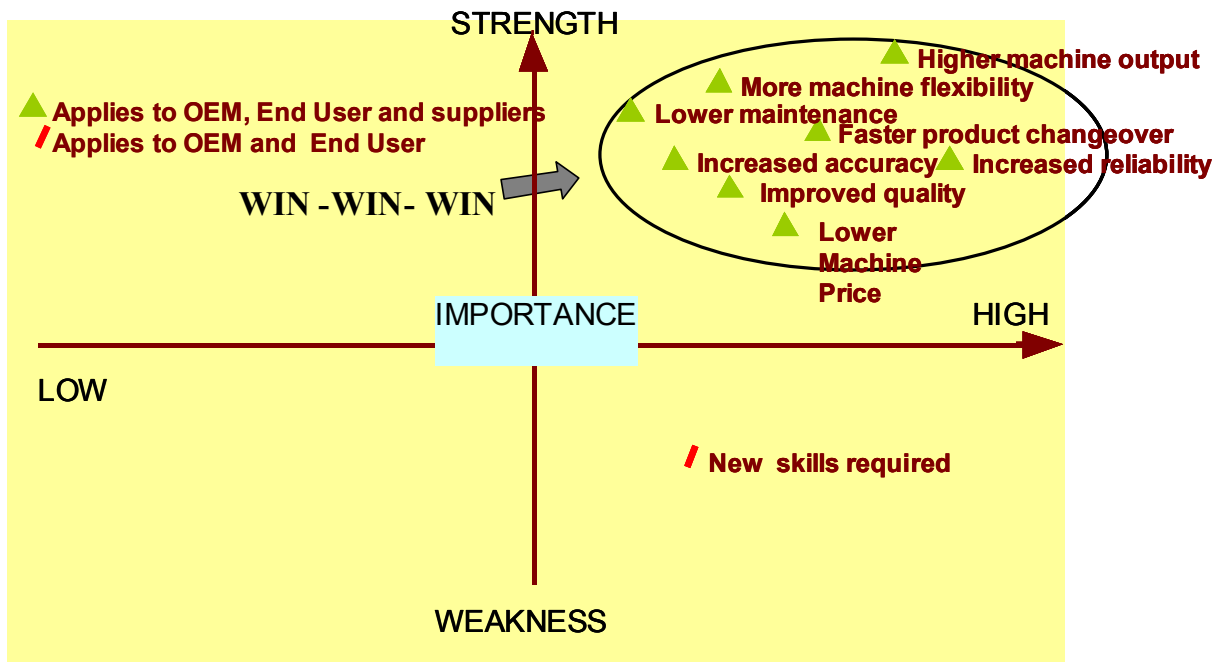


## Executive Overview

These Guidelines for Packaging Machinery Automation represent a point in time in an ongoing quest among manufacturers, OEMs, and control system suppliers to commonize packaging industry vocabulary and provide universal definitions of packaging machine architectures, networks, and line types as enablers behind shared business value.

It is important to understand that this is a work in progress that may be updated as often as three times per year. The first versions are primarily informative without imposing a lot of particular requirements on Users or OEM's. It will be useful for Users, OEMs and Technology Providers to begin to use these guidelines to normalize vocabulary; establish definitions of architectures, networks, and line types; and to set expectations for guidelines that will be forthcoming

The following chart depicts the relative strengths and weaknesses of applying the digital motion controls advocated by the OMAC Packaging Workgroup (OPW) to packaging machines.



**Benefits of Applying Digital Motion Controls to Packaging Machines**

## Introduction

---

The OMAC Packaging Workgroup (OPW), a subset of the Open Modular Architecture Controls Users Group, prepared this document. The information contained herein is intended as a guideline for the control and automation of packaging machinery and systems by end users, equipment builders, and technology suppliers. This document is a work-in-progress and will be updated regularly, but no more frequently than 3 times per year following the general meetings of the Workgroup. Proposed additions to this document will be found at <http://omac.packworld.org/>. Clarifying information that has not been made a part of this guideline may also be found at <http://www.omac.org/>.

Five OPW teams contributed to this document: *PackLearn*, *PackAdvantage*, *PackConnect*, *PackML* and *PackSoft*. These teams are comprised of volunteers representing packagers, packaging machine builders, control manufacturers, systems integrators and organizations affiliated with these industries. To contact a team leader or to contribute to one of these teams, see the group's web site at <http://www.omac.org/>

## Definitions

---

The Glossary of Motion Control Terms, version 1.0, prepared August 2001 by the OMAC Packaging Workgroup PackLearn Team and adopted October 15, 2001 is incorporated as a part of this guideline (Appendix IA). This glossary defines the key terms for control and integration of modern packaging machines and systems.

## Line Types

---

Four distinct packaging line configurations, designated Line Types 1, 2, 3 and 4, are adopted to represent the various methods of packaging line integration. The details of these line types are specified in the Line Types Document, Version 1.1, prepared 7 September 2001 by the OMAC PackML Team and adopted by the Workgroup October 15, 2001 (Appendix IIA).

Line Type 1	Machines are autonomous; they react to plant conditions through their own array of sensors and switches. They do not communicate to the other machines that comprise the line
Line Type 2	<p>Line Type 2 machines differ from Line Type 1 because they have the ability to communicate with other machines</p> <p>Two subtypes have been defined: 2A and 2B. Functionally these are identical, they achieve their functionality through two different methods:</p> <p style="padding-left: 40px;">Machines within Line Type 2A communicate through digital and analog I/O</p> <p style="padding-left: 40px;">Line Type 2B machines communicate across data networks with or without digital and analog I/O.</p>
Line Type 3	<p>An enhanced version of Line Type 2B</p> <p>Contains both, or a combination of, SCADA server and clients and/or line supervisory controller</p> <p>The enhanced functionality of this Line Type provides line performance data, loss analysis, root cause analysis toolkits, maintenance and troubleshooting tools, change parts database, etc.</p> <p>Individual packaging line networks could be bridged to provide data acquisition and display for the whole factory</p>
Line Type 4	<p>Packaging lines are integrated into wider business IT systems via the ERP Bus</p> <p>This provides functionality such as the passing of production orders into the factory and progress against the order to be monitored. It could also allow the automatic reporting of performance, quality and material usage.</p>

### Packaging Line Types

## PackML State Model

---

State names must be defined to create effective line performance metrics. A rigorous definition of state names requires a consistent state model. An industry standard state model will enable development of products and systems to create uniform performance metrics. A summary of the State Model follows with the complete text in Appendix IIB.

### States

A *State* completely defines the current condition of a machine. Transitions between States occur:

- As a result of a *Command*

- As a result of a *Status* change. This is generated by change of state of one or a number of machine conditions, either directly from I/O or completion of a logic routine
- Automatically after completing a *No Command* State

A *No Command* State is one that, after completing its own logic, forces automatic transition to a *Final* State.

A *Final* State represents a safe state, i.e., no moving parts

A *Transient* State is one that 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

A *Quiescent* State is used to identify that a machine has achieved a defined set of conditions. In such a State the machine is holding or maintaining a status until transition to a *Transient* State.

## **Modes**

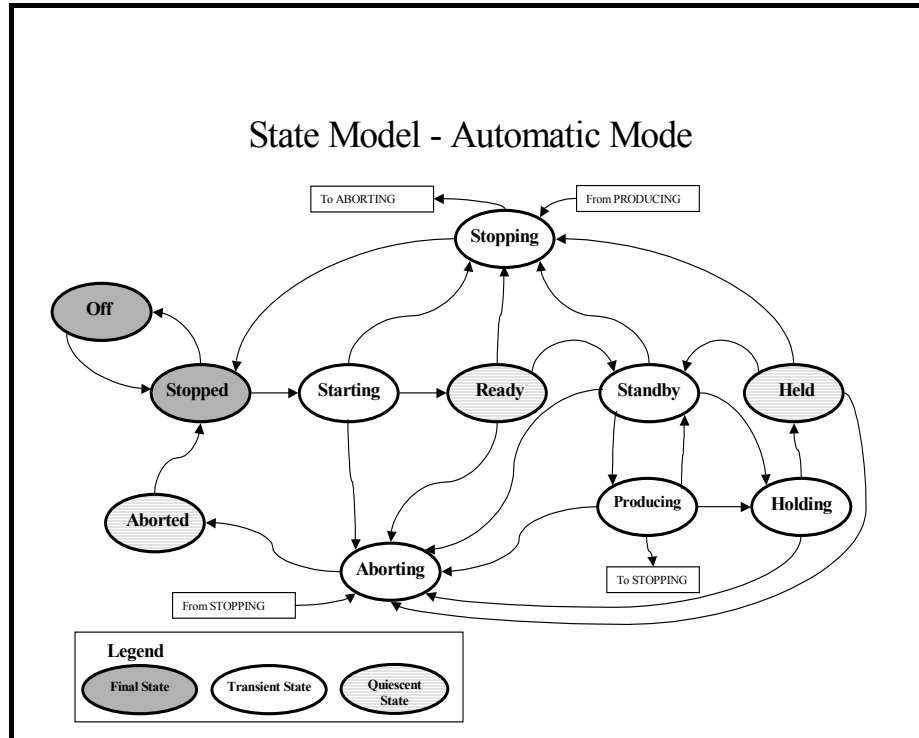
A Mode determines how a machine will operate in response to the commands that are issued to it. This document focuses on one operating mode, Automatic. Other modes of operation do exist across the packing industry, these have various names Semi-automatic, Manual, Maintenance Index etc. Packing operations in modes other than automatic will be the subject of further work.

### **Automatic Mode**

This represents the mode that is utilized for routine production. The machine executes relevant logic in response to commands that are either entered directly by the operator or issued by another supervisory system.

### **Automatic Operation State Model**

The proposed Machine State model for the operation of a machine in Automatic mode is depicted in Figure 1 below.



**Figure 1: Machine State Model for Automatic Mode Operation**

A brief description of the individual machine states appears below. Note that the word “running” was considered as a state name, but was rejected because it is not specific enough as commonly used. In this document, “running” is generally used to refer to the machine in either the STANDBY or PRODUCING state.

State Name	State Type	Description
OFF	Final	<p>All power to machine switched off. This state is assumed if there is no response from the machine. This is an optional Final State.</p> <p>The OFF state may be reached in two ways, first the power to the machine is switched off and second power is removed as in a power utility failure. In the OFF state it must be assured that the machine will go at the least to the initial stopped state when power is restored either by switch or restoration of the utility power source.</p>
STOPPED	Final	The machine is powered and stationary. All communications with other systems are functioning (If applicable).
STARTING	Transient	This state allows the machine to be prepared for running. This could include such processes as heating, self-testing or calibration.
READY	Quiescent	This is a State that indicates that STARTING is complete. This state maintains the machine conditions that were achieved during the STARTING state.
STANDBY	Transient	<p>The machine is running at the relevant setpoint speed, there is no product being produced.</p> <p>This state can be reached either in response to a Start Command from READY or any internal machine logic that would dictate a temporary transition from the PRODUCING state, such as materials runout.</p> <p>The status "materials runout" could refer to lack of materials on the machine's own infeeds or could refer to a stoppage of a downstream machine.</p>
PRODUCING	Transient	Once the machine is processing materials it is deemed to be producing.
STOPPING	No Command	This state executes the logic which brings the machine to a controlled and safe stop
ABORTING	Transient	The ABORTED state can be entered at any time in response to the Abort command or on the occurrence of a machine fault. The aborting logic will bring the machine to a rapid, controlled safe stop. Operation of the Emergency Stop or "E-Stop" will cause the machine to be tripped by its safety system. It will also provide a signal to initiate the ABORTING State.
ABORTED	Quiescent	This state maintains machine status information relevant to the Abort condition. The Stop command will force transition to the Stopped state
HOLDING	Transient	When the machine is STANDBY or PRODUCING the Hold command can be used to start HOLDING logic that brings the machine to a controlled stop.
HELD	Quiescent	The HELD state would typically be used by the operator to temporarily hold the machine's operation while material blockages are cleared, or to stop throughput while a downstream problem is resolved.

**Table 1: Automatic Operations Machine States**

An example State transition matrix for Automatic Mode is shown below. Note that the State Model does not rigidly specify the internal machine state transition logic, which will vary depending on the application.

	Machine Status						Commands				
	Power On	Power Off	Materials Ready	Materials Runout	Machine Fault	State Complete	Prepare	Start	Stop	Hold	Abort
<b>Initial State</b>											
OFF	STOPPED										
STOPPED		OFF					STARTING				
STARTING					ABORTING	READY			STOPPING		ABORTING
READY					ABORTING			STANDBY	STOPPING		ABORTING
STANDBY			PRODUCING		ABORTING				STOPPING	HOLDING	ABORTING
PRODUCING				STANDBY	ABORTING				STOPPING	HOLDING	ABORTING
STOPPING					ABORTING	STOPPED					ABORTING
HOLDING					ABORTING	HELD					ABORTING
HELD					ABORTING			STANDBY	STOPPING		ABORTING
ABORTING						ABORTED					
ABORTED									STOPPED		

**Table 1 : Automatic Mode State Transition Matrix**



## **Automatic Mode Comments**

Some further explanatory notes have been included to further clarify some elements of the State Model.

### **ABORTING, ABORTED, and E-Stop**

The ABORTING State has been included for the following reasons:

- This is the location of the shutdown logic that executes on either an ABORT command or a machine fault. The ABORTING State can be entered from any machine state except STOPPED, ABORTING and ABORTED.
- The ABORTED State is a quiescent state that indicates the ABORTING logic has completed. It holds the machine in a safe state but it also holds any relevant data that will help in diagnosing the fault or helping to reconcile production information after a major event.
- The manufacturer of the specific packing machine develops the logic, which generates the Machine Fault command. This is derived from a number of sensor signals or internal flags as appropriate. For example, if the Emergency Stop or “E-Stop” pushbutton is depressed the hardwired stop system is initiated and the machine is brought to a rapid stop by the operation of the emergency stop system. Generally this achieved by hardwired interlocks which operate independently of the machine's control system. One of the conditions which would result in the setting of the Machine Fault flag would be Emergency Stop System operated. This would ensure that, whatever its current state, the machine would make a transition into ABORTING and the to ABORTED. In this way the machine's software based control system would reflect the condition of the machine which was brought about through the operation of the hardwired system.

### **Notes about Machine Start Up**

In order to start a machine from the STOPPED state the operator needs to issue the Prepare command and once the machine is READY the Start Command will allow the machine to start running. In some lines a single operator may be responsible for the supervision of a number of machines which may all be locally started. On one pass down the line he can prepare all of the machines for operation and once all are confirmed READY he will

then be able to start individual machines in the correct order. For those lines, which have supervisory control, the Prepare command, can be issued at all machines at once and then the correct starting sequence can be initiated once all are confirmed READY.

### **S88 Software Compatibility**

A proposed mapping of the S88 Batch Control state model command set to the OMAC Packaging State Model has been prepared. This should pave the way for S88-based software to be extended from batch control to packaging applications. The proposal in draft form is found in Appendix IIc.

## **PackTags**

---

PackTags are named data elements used for open architecture, interoperable data exchange in packaging machinery. In PackTags version 1.0, an initial set of PackTags sufficient for computing machine performance metrics was defined. In PackTags version 2.0, that initial set was expanded and adapted to support more of the PLCs that are in common use in the packaging industry today. Further information on *PackTags* is available in Appendix IID.

## **PackConnect System Architecture**

---

### **Communications Networks**

System architectures for packaging machines and systems may contain the following types of communications networks: Enterprise bus; Cell bus; Motion bus; Field or I/O bus.

### **Control Architecture**

Control architectures are generally arranged in one of three methods identified as drive-based, controller-based, and PC-based. The physical and logical diagrams for these three methods are shown in Figures 1 through 7.

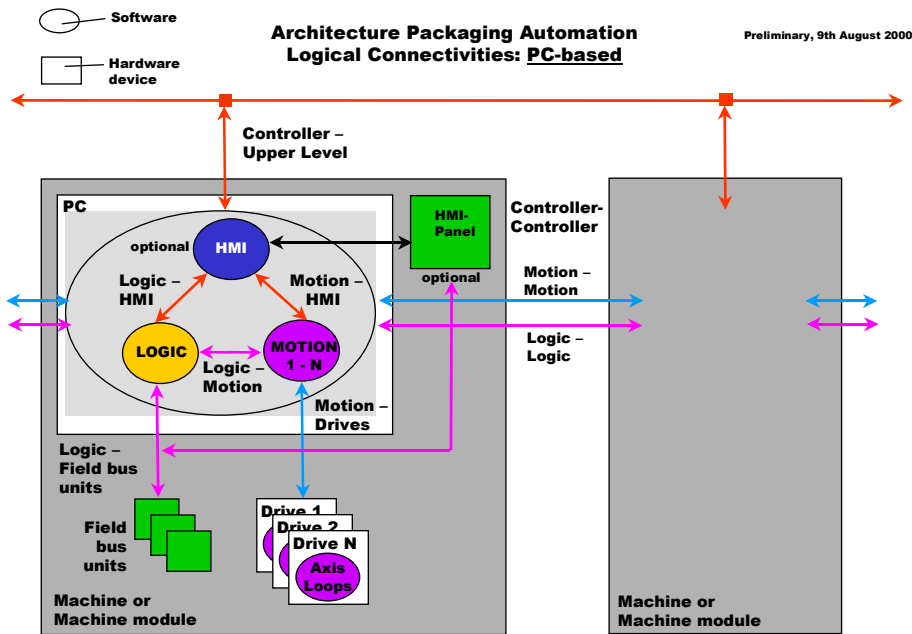


Figure 1: Logical Connectivity of PC-Based System Architecture

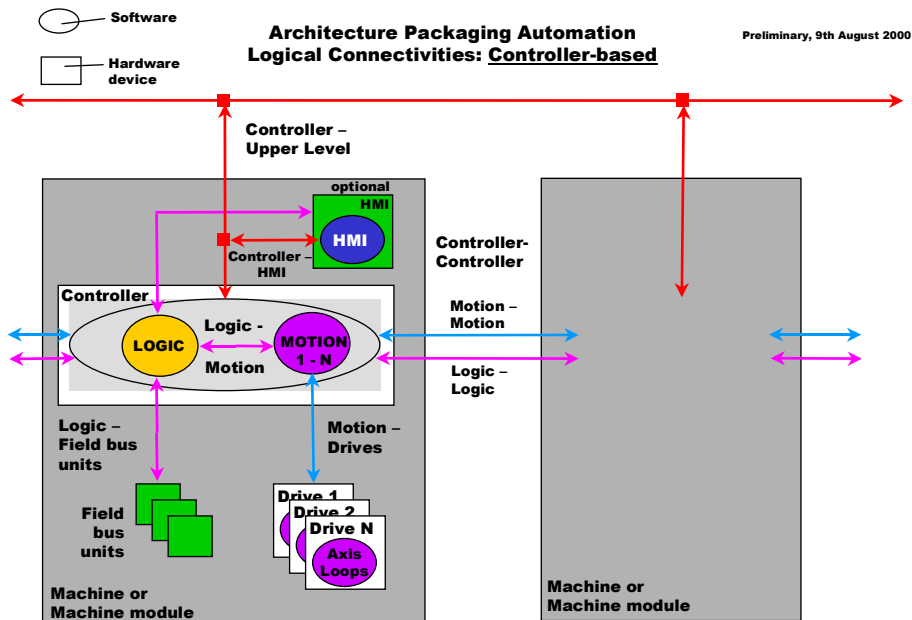


Figure 2: Logical Connectivity of Controller-Based System Architecture

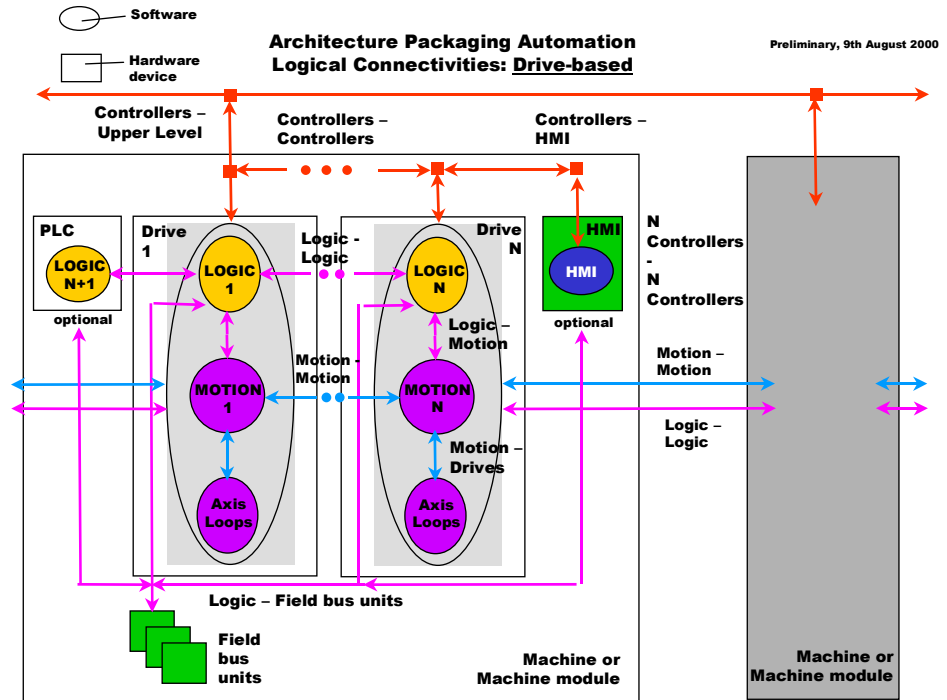


Figure 3: Logical Connectivity of Drive-Based System Architecture

Control for Packaging Automation  
Architecture/Connectivity -Discriptions

The drawings shows kinds of possible Architectures of Packaging machines.  
Every BUS-Level is optional.  
Shown devices are only a subset of possibilities,important is their connectivity tho the various BUS-Levels.  
The table on the right includes the main requirements of the BUS-Systems.

- ERP-BUS for data.exchange between the Packaging machine and the Management Plan insystem
- CELL-BUS for data exchange and interlocking between machines, intelligent devices, controllers,HMI's in a machine
- FIELD-BUS for data exchange and interlocking between controllers and devices with realtime requirements
- MOTION-BUS for data-exchange and synchronization between Motion-Control and Axis-Loops(Amplifier)or between Motion Controllers.
- I/O BUS for Data Exchange between controllers and the I/O-Level

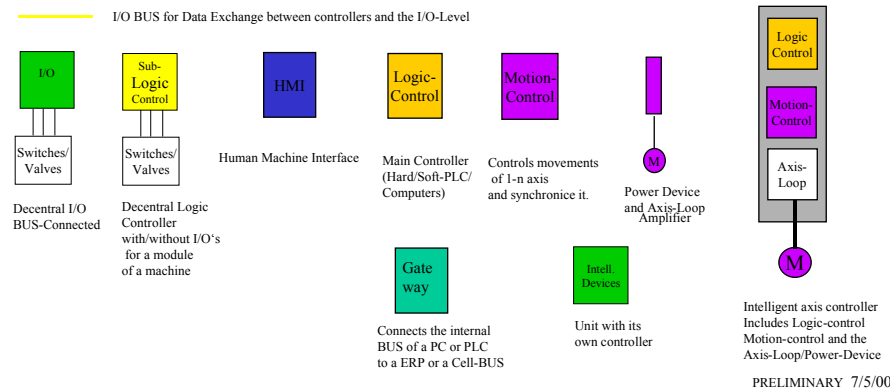
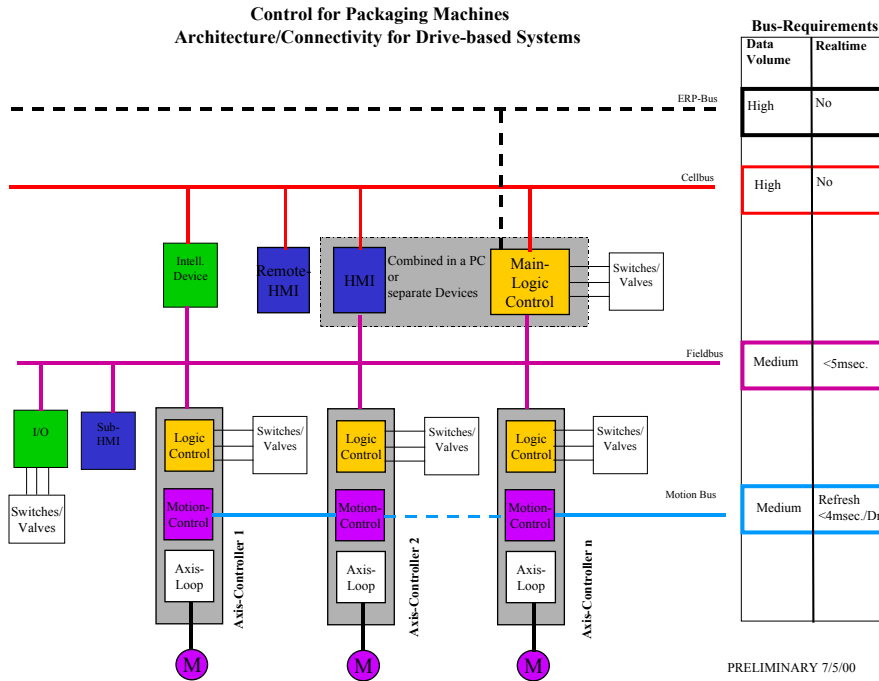
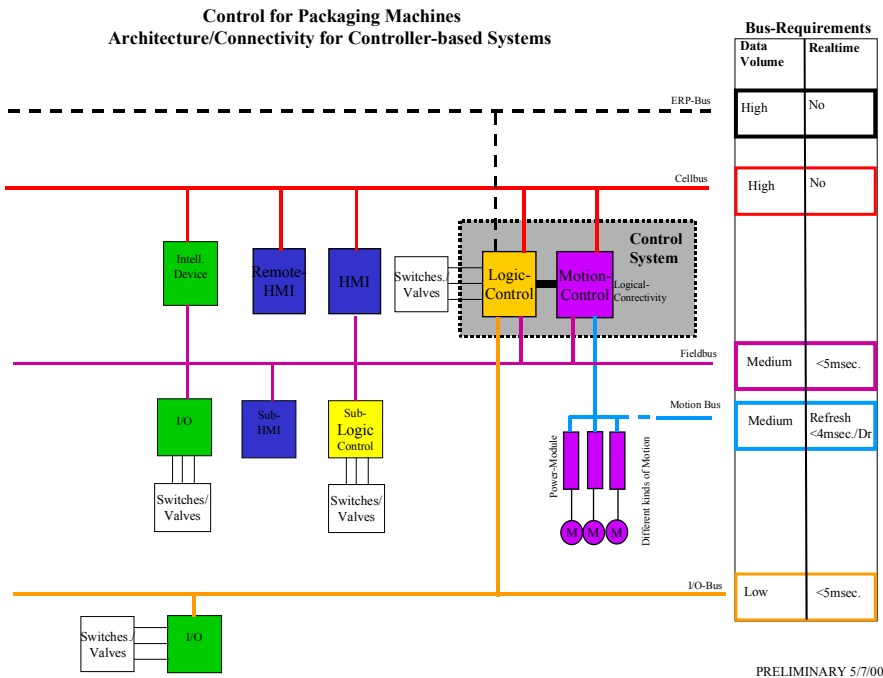


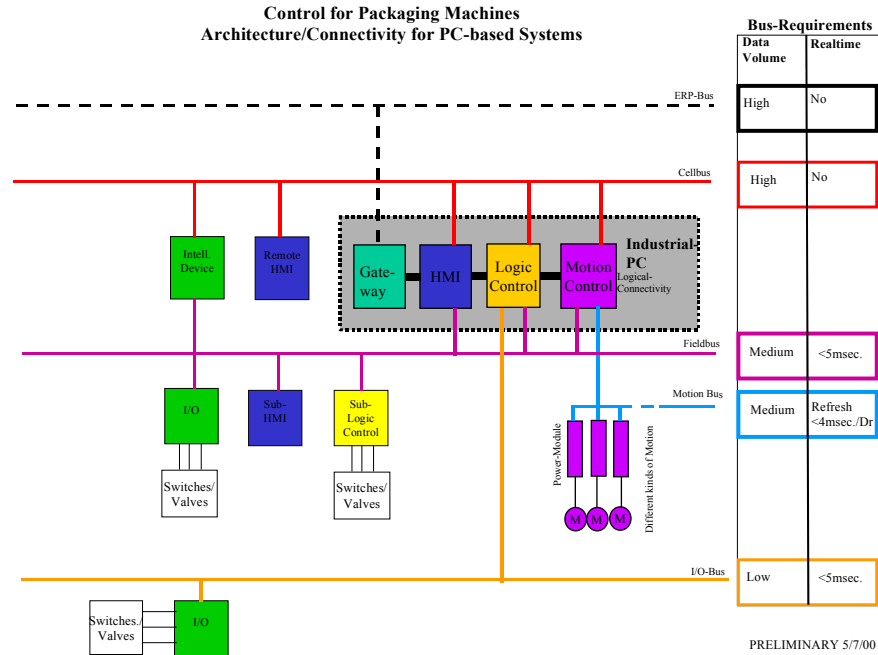
Figure 4: Architecture/Connectivity Descriptions



**Figure 5: Architecture/Connectivity of Drive-Based Systems**



**Figure 6: Architecture/Connectivity of Controller-Based Systems**



**Figure 7: Architecture/Connectivity of PC-Based Systems**

### Motion Interface Types

*Type A* - Single Axis Drive without Synchronization to other drives

*Type B* - Intelligent Single Axis Drive with synchronization to other drives

*Type C* - (Position Mode) Multi-Axis Motion Control

*Type D* - (Velocity Mode) Multi-Axis Motion Control

*Type E* - (Torque Mode) Multi-Axis Motion Control

*Type F* - Multi-Axis Motion Control with Distributed Electronic Gearing in the drive

Figure 8 compares functionality of upper A through F.

	<b>Type A</b>	<b>Type B</b>	<b>Type C</b>	<b>Type D</b>	<b>Type E</b>	<b>Type F</b>
	<b>Single Axis Drive <u>without</u> synchronization to other drives</b>	<b>Intelligent Single Axis Drive <u>with</u> synchronization to other drives</b>	<b>Multi-axis Motion Control (Position mode)</b>	<b>Multi-axis Motion Control (Velocity mode)</b>	<b>Multi-axis Motion Control (Torque mode)</b>	<b>Multi-axis Motion Control with Distributed Electronic Gearing in the Drive</b>
<b>PC-/Controller-based</b>			X	X	X	
<b>Mix between PC-/Controller-based and Drive-based</b>						X
<b>Drive-based</b>	X	X				
<b>Non real time commands. Program, Recipes, Profiles, Parameters, Status</b>	Via cell bus	Via cell bus	Via cell bus	Via cell bus	Via cell bus	Via cell bus and over Motion bus
<b>Realtime I/Os from PLC to Motion. Start, Stop, Cam switch</b>	Via Field bus	Via Field bus	Internal in control	Internal in control	Internal in control	Internal in control and over Motion bus
<b>Real time Motion commands</b>	Internal in the Drive	Internal in the Drive and with external master encoder	Position from control to drives	Velocity from control to drives	Torque from control to drives	Master encoders from control to drives
<b>I/O bus commands</b>	not care	n.c.	n.c.	n.c.	n.c.	n.c.
<b>Location Position loop</b>	In drive	In drive	In drive	In control	In control. Also velocity loop	In drive
<b>Profile location</b>	In drive	In drive	In control	In control	In control	In drive
<b>Profile synchronization</b>	No	Via Motion bus	In control	In control	In control	Via Motion bus
<b>Print register (Profile correction)</b>	In drive	In drive	In control	In control	In control	In drive
<b>Motion bus cycle times for all drives [msec]</b>	n.c.	1	1	0.1	0.05	1
<b>Number of user programs to work together</b>	1, only in PLC	n+1, Number of axes plus in PLC	1, common for PLC and Motion in Control	1, common for PLC and Motion in Control	1, common for PLC and Motion in Control	n+1, Number of axes plus in Control
<b>Motion Control performance depends on number of axes</b>	No	No	Yes, low. Profiles in the control	Yes, middle. Profiles and position-loop in the control	Yes, high. Profiles, position- and velocity-loop in the control	No
<b>Motion functionality: Single axis move</b>	x	x	x	x	x	x
<b>Motion functionality: Camming, Gearing</b>		x	x	x	x	x
<b>Motion functionality: Camming, Gearing, Registration</b>		x	x	x	x	x
<b>Motion functionality: Camming, Gearing, Torque</b>		x	x	x	x	x
<b>Motion functionality: Coordinated robotics moves</b>			x	x	x	

**Figure 8: Functionality by Motion Interface Type**

## **Application Recommendations**

There is an ongoing process that changes the functionality in packaging machines from mechanical to electronic functions. More and more functions are moving from mechanical to servo driven solutions. That means, the numbers of servo axes in most packaging machines are increasing rapidly.

For applications where coordination/synchronization is required between the axes, the use of PC- and Controller-based architectures with multi-axis motion controllers typically perform best.

For applications with a low amount of axes and where no coordination/synchronization is required, drive based controllers can be used quite effectively.

Applications with 1 to 3 axes can be equipped by either Drive-based or with Multi-axis (PC-based and Controller-based) architectures.

## **Network Certification**

Packaging machines and systems should use only motion systems that are specified by IEC in physical (Hardware) and logical (Software) aspects.

## **Recognized Networks**

The following motion networks meet the requirement of being specified by IEC.

IEC 61491      SERCOS Interface

IEC 61158      Profibus-MC

Figure 9 compares features of these networks (see next page).

Feature	SERCOS	Profibus MC (PROFIdrive V3.0)
Type of Communication	Serial	Serial
Network Topology	Ring	Ring,Star,Tree,Bus
Physical Media	Fiber optic	Fiber Optic 2 wire twisted pair copper
Maximum Cable Length	Plastic – 40 m node-to-node; 10,000+ m max. ring length. Glass – 800 m node to node; 200,000+ m max. ring length.	100M Copper at 12Mbps, 2850M Glass FO, 50M Plastic FO Per Bus segment. Up to 7 Segments possible.
Communications Modes	Master-Slave. Cyclic – Deterministic time slice for managing real time communications, speed depending on # of nodes on the ring. Cyclic data can be transmitted at rates as low as 62.5 us. Non-cyclic – Used for non-real time data transfers such as status and diagnostic messages.	Master – Slave Slave -Slave (Peer to Peer) Reserved Space for Acyclic communication and programming
Servo Loop Modes Supported	Block mode (drive internal interpolation), position, velocity, torque	Position, Velocity (PROFIdrive V2.0)
Transmission Speed	2/4/8/16 Mbit/s	1.5/12 Mbps
Update Rates	Selectable from 62.5 usec and up.	Selectable in 250 usec increments.
Axes per cycle time with acyclic datas for each axes / Cycle time / Baudrate / Motion interface type	40 / 1 msec / 16 MBd / Type C	4 / 1 msec / 12 MBd / Type C
Jitter Rate	< 1 uSec	< 1 uSec
Number of Masters	One Master per ring	One Master (class 1) per ring
Maximum Number of Nodes Per Network	254 per ring multiple rings allowed	125 per network
Physical Standard	IEC 61491/-EN61491	IEC 61158 / EN 50 170
Control Standard (Protocol, Message Content, Data Format, Programming)	IEC 61491/- EN61491	IEC 61158 / EN 50 170
Number of Vendors Supporting	More than 75 vendors worldwide manufacturing hardware/software products for Motion Control.	More than 200 Vendors for general ProfiBus worldwide, not just Motion Control. How many vendors use it for Motion Control?
Interoperability across Vendors	Certification ensures compliance with the standard. S- and P-Parameter. S=Standard P=Customer specific	Yes, all vendors must pass Device certification that ensures compliance with the standard (what standard?) and operation with other products (how?).
Installed Base	More than one million SERCOS ASICs shipped worldwide. Used only in Motion Control, not in typical field bus applications.	More than 2 million Profibus ASICs shipped worldwide. How many for Motion Control?
Websites	www.sercos.com, www.sercos.de Vendor sites too numerous (more than 75) to list. See sercos.com and sercos.de for links to SERCOS member's websites.	www.profibus.org; includes links to vendors sites. Detail infos see under www.profibus.org under Technology / Technical Description / PROFIBUS / 5 Application Profiles / 5.3 PROFIdrive

**Figure 9: Digital Servo Communication Technology Comparison**

## **Interoperability Requirements**

An important target is to achieve the interoperability between drives and motion control (for PC- and Controller-based architectures). Interoperability is understood as a possibility to use drives (amplifier & motors) from different suppliers on a supplier-independent motion controller. IEC standards like SERCOS & Profibus-MC currently offer some levels of interoperability, however, the Packaging Industry is challenging SERCOS NA & Profibus-MC and others for a higher level or guarantee of multi-vendor interoperability and for providing Compliance Test procedures.

## **Packaging Specific Parameters**

Necessary features for packaging applications can only be provided by a special optimized Packaging Profile. This Packaging Profile should run with IEC motion bus systems.

## **Components**

---

No hardware, software or communications components are specified or endorsed at this time.

## **PackSoft Programming Language**

---

OMAC derives common solutions collectively for both technical and non-technical issues in the development, implementation, and commercialization of open architecture control technologies to maximize the business value of packaging machinery by developing guidelines that lead to the most appropriate application of advanced automation technology.

The PackSoft Subcommittee recommends that Technology Providers, OEM's and End Users adopt the IEC 61131-3 Standard for Programmable Controllers: Programming Languages for packaging machine automation, including motion control. The IEC 61131-3 standard applies to a wide range of programmable controllers and is commercially available in products from a number of Technology Providers. Use of the standard encourages software design that is hierarchical in nature, re-usable through program organization units; and the standard provides facilities for real-time exchange of information via communication networks.

The PackSoft Subcommittee recommends the use of PLCopen Function Blocks for Motion Control V1.01, the Extensions (Part 2) to this standard, and other coming PLCopen publications as they might be a fit, all based on IEC 61131-3.

The PackSoft Subcommittee develops Packaging Industry related Control Software guidelines that allow to commonly implement as technology, program machinery equipment and controls, maintain, train and learn the use of software on packaging industry devices.

The objectives of the OMAC PWG PackSoft committee are to

- define software modules (based on appropriate standards, e.g. IEC61131-3) which describe basic packaging machinery control elements,
- develop a programming convention and a set of functional software elements which lend themselves to be become common use throughout the Packaging Industry to simplify the structure, understanding and handling of generic machine elements and their representation in software.
- define a set of functions which will serve the majority of packaging user's applications and needs
- promote the adoption of useful existing and emerging standards to own definitions.

*Detailed information on PackAL is available in Appendix III on page 91.*

## **Skills Required**

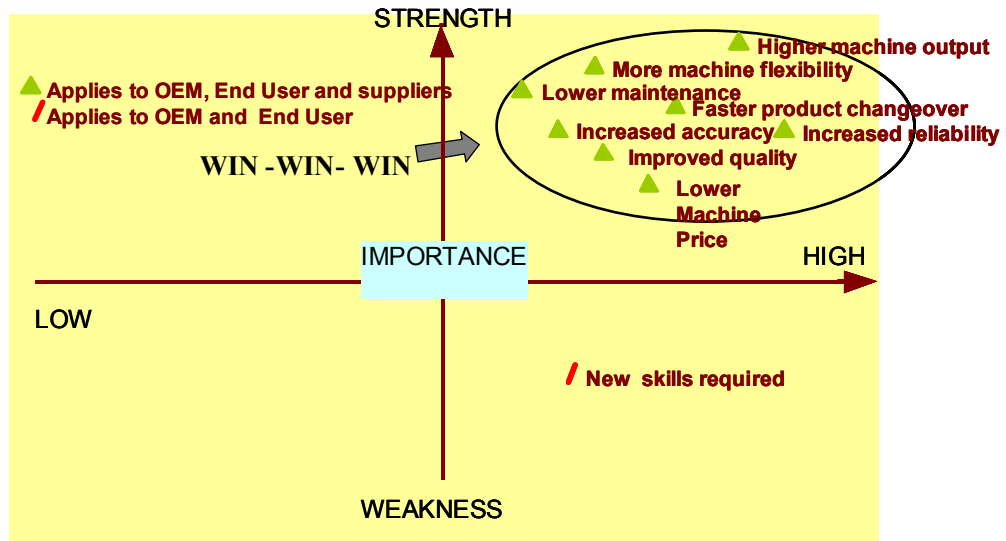
---

The PackLearn Team of the Workgroup has conducted an extensive survey of users, machinery builders and technology providers to determine skills and training needs for these groups. The PackLearn team's survey results are accepted as reference information and may be used or cited by others with the phrase "used with permission of the OMAC Packaging Workgroup; [www.omac.org](http://www.omac.org)". Information on skills and education will be developed.

(Note: Appendix V Reserved)

## Benefits to be Achieved

The following chart depicts the relative strengths and weaknesses of applying digital motion controls to packaging machines. Examples of benefits will be added to Appendix VIA.



**Figure 10:**  
Benefits of Applying Digital Motion Controls to Packaging Machines

## **Appendix IA: Glossary of Motion Control Terms v1.0**

---

This glossary was compiled by the OMAC Packaging Workgroup PackLearn Team and first published in August 2001. Please submit any comments to:

Maria Ferrante  
Director of Technical Services  
PMMI  
4350 North Fairfax Drive  
Suite 600  
Arlington, VA 22203  
703/243-8555 ~ Fax: 703/243-8556  
Email: technical@pmmi.org

### **Motion Control Terms**

#### **Acceleration**

The rate at which something increases its velocity. Acceleration is usually measured in units of velocity change for each unit of time (inches/ second(velocity)/ second (time)) and in this example would be written as in./sec/sec or in./sec<sup>2</sup>.

#### **Accuracy**

The relative status of something compared to its absolute or perfect value. In motion control this will most often be a position description. A command may be sent to move 4.0". The accuracy of the system will be defined by how close to the absolute value of 4.0" the system can effect the move. Accuracy may be defined as a one time incident or the average over a number of cycles or motions. Positioning accuracy will normally be defined in terms of deviation (+/- from theoretical) or limits (acceptable variation from theoretical: ie. 3.8"-4.2" define acceptable limits of variation around a theoretical point of 4.0")

#### **Active front end**

A front end processor which interacts with both upstream and downstream equipment and makes required changes based on the incoming and outgoing parameters and data, without external control.

### **Actual position**

The position of an axis relative to the commanded position. This may be the position at the end of the commanded move or the lag between command position at any point during the move and the actual position of the axis at that point. The later is commonly referred to as following error.

### **AC servo**

A general term referring to a motor drive that generates sinusoidal shaped motor currents.

### **Alarm**

An indication that a monitored parameter is not within the prescribed acceptable range. Usually this is in the form of an output that can be used to initiate an operator warning or advisory, to generate a corrective action, or to cause a cessation of the operation underway.

### **Analog servo**

A servo system that utilizes analog control and feedback systems such as voltage variation, pressure changes etc. Analog servos are most commonly found in hydraulic and similar systems.

### **Analog signal**

A communication within the system that is accomplished by means of a signal that varies in direct relation to the intensity or magnitude of the external quality being measured. Typical examples are a 0-10 volt motor control signal, a hydraulic pilot pressure, a pneumatic control pressure.

### **Axes of motion**

The specific major directions along which controlled movement occurs. Usually the number of these major directions employed in a specific machine. Usually defined as follows:

X: Linear motion in positioning direction

Y: Linear motion perpendicular to positioning direction

Z: Vertical linear motion

A: Angular motion around X (roll)

B: Angular motion around Y (pitch)

C: Angular motion around Z (yaw)

**Axis**

A principal direction along which movement of a tool, component or work-piece occurs.

**Brushless servo**

A servo drive in which commutation of the current is accomplished electronically rather than through the use of mechanical brushes and a commutator. In a brushless motor, the windings are on the stator and the magnetic field is provided by permanent magnets in the armature. This results in lower inertial loadings and increased heat dissipation capability over a standard brush type motor.

**Bus**

One or more conductors used as a path over which information is sent from one of many sources to one of many destinations.

**Centralized control**

A control system in which all of the primary processing is done at a single location rather than at multiple points throughout the system.

**Circular interpolation**

The generation of an apparently circular motion through the coordinated movements of two axes. The actual path is a series of straight line approximations generated by software algorithms.

**Collision detection**

The use of sensors to detect the imminent impact of two or more parts in a system. The signals from the detection sensors can be used to stop motion or to provide a ramped slow down for a “soft” mating of the approaching components.

**Commutation**

A term which refers to the action of steering currents or voltages to the proper motor phases so as to produce optimum motor torque. In brush type motors, commutation is done electromechanically via the brushes and commutator. In brushless motors, commutation is done by the switching

electronics using rotor position information obtained by Hall sensors, a tachometer, or resolver.

### **Converter**

The process of changing AC to DC and back to AC again. This is accomplished through the use of a diode rectifier or thyristor rectifier circuit. The term "converter" may also refer to the process in an adjustable frequency drive, consisting of a rectifier, a DC intermediate circuit, an inverter and a control unit.

### **Coordination**

The integration of the movements of two or more axes of motion so that the resultant motion is a path which none of the axes are capable of independently. Coordination may also involve the use of sensors and other internal or external commands in the integration effort which assist in effecting the movement or work desired.

### **Current controller**

A system that utilizes an electronic method of limiting the maximum current available to a motor. This current is adjustable so that the motor's maximum current can be controlled and normally includes functions that serve as a protective measure to prevent extended overload conditions from damaging the motor or the controller.

### **Cut to length**

A sub routine within a motion control processor or stand alone processor that is designed to feed material being processed a pre-set distance prior to performance of a secondary function such as a cut off. Feed back systems are employed to insure repeatability of the set feed length.

### **DC bus**

A type of circuit or protocol that serves as a common communications pathway shared by several components and which uses a direct current voltage level as a reference. It may also be used to describe a power distribution system shared by multiple components within a machine or power distribution system.

**Deceleration**

The rate at which something decreases its velocity. Deceleration is usually measured in units of velocity change for each unit of time (Inches/ second(velocity)/ second (time)) and in this example would be written as in./sec/sec or in./sec<sup>2</sup>.

**Decentralized control**

A control system in which the logic functions and input/output functions are located at individual pieces of equipment or sub systems and function essentially independent of each other. Normally the independent systems will have some means of communicating vital information with each other.

**Deterministic scan time**

The frequency with which a Programmable Logic Controller (PLC) executes a program, including scanning all the inputs and outputs on the system. This time is usually measured in milliseconds. PLCs will normally read or “scan” the instructions in the logic program in set, sequential manner. The time required to read a specific instruction and all of the following instructions, returning to the initial instruction is generally referred to as scan time.

**Device level network**

A means of putting sensors, actuators and other components on a common network cable that is connected to a PLC. It eliminates point -to-point wiring between the PLC and each device.

**Diagnostic code**

A code (usually alpha numeric or numeric) that is displayed on some type of operator interface or within a program that indicates the status of a monitored component. Usually diagnostic codes are used to indicate the location and nature of a fault or error condition that requires rectification.

**Digital motion control**

A motion control system that utilizes binary coding for all logic and control functions. Analog inputs can normally be used on a digital system, but they must be converted through an analog to digital converter before being processed.

### **Digital servo**

A servo motor (see AC servo and Analog servo) which utilizes binary coding for all parameter generation and feed back.

### **Digital signal**

A signal that communicates information in electrical pulses that represent binary 1s and 0s. It is widely believed that digital signals can transmit more information more reliably in a given unit of time than analog signals.

### **Drive**

An electronic device that translates a given command from a motion controller into the electrical current that turns a motor.

### **Efficiency**

The efficiency of a motor is the ratio of mechanical output to electrical input. It represents the effectiveness with which the motor converts electrical energy into mechanical energy.

### **Electronic cam profiles**

A technique used to perform non-linear motion electronically similar to that achieved with mechanical cams.

### **Electronic clutch**

The process of generating a slave profile based on master position or time periods by enabling and disabling electronic cam or gearing functions.

### **Electronic gearing**

A method that simulates mechanical gears by electrically synchronizing one closed-loop axis to a second axis (open- or closed-loop) through a variable ratio.

### **Electronic line shaft**

A virtual axis that is used as the master axis on a machine to which other axes are synchronized by electronic gearing or camming profiles.

### **Encoder**

A feedback device that translates mechanical motion into electrical signals indicative of actuator position. Incremental and absolute encoders are com-

mon varieties; as the names imply, their output indicates incremental or absolute changes of position.

### **Encoder resolution**

The number of electrically identified positions occurring in 360 degrees of input shaft rotation.

### **EnDat**

An interface that has become a standard for serial data transfer covering position and parameters.

### **EMC/CE**

Testing of electrical devices to check their ability of adhering to conducted/radiated emissions. Set by the European Directive. Applications may need line filters, cabinet bonding, and shielded cabling to assist this effort.

### **E-Stop**

A stop function that has all of the following requirements:

- It shall override all other functions and operations in all modes.
- Power to machine actuators that can cause a hazardous condition (s) shall be removed as quickly as possible without creating other hazards (e.g., by the provision of mechanical means of stopping requiring no external power, by reverse current braking for a Category 1 stop).
- Reset shall not initiate a restart.
- The emergency stop shall function as either a Category 0 or Category 1 stop. The choice of category of emergency stop shall be determined in accordance with the requirements of the application.
- Where a Category 0 stop is used for the emergency stop function, it shall have only hardwired electromechanical components. In addition, its operation shall not depend on electronic logic (hardware or software) or the transmission of commands over a communication network or link.
- Where a Category 1 stop is used for the emergency stop function, final removal of power to the machine actuators shall be ensured and shall be by means of electromechanical components.

### **Ethernet**

An open networking standard, Ethernet is widely used in office automation and is increasingly being used for packaging machine networks. Originally developed for communications speed of 1.5 megabits/sec, newer versions permit speeds up to 100 megabits/sec.

### **Event**

A change-of-state of an input parameter, such as the triggering of a limit switch or proximity sensor.

### **Fault**

The error received when a drive or control has attempted an illegal process and becomes disabled.

### **Feedback**

The signal or signals received from a controlled machine or process to denote its response to the command signal.

### **Feedforward**

A method that "precompensates" a control loop for known errors due to motor, drive, or load characteristics to improve response. It depends only on the command, not the measured error.

### **Fiber optic**

A lightwave system that utilizes a glass or plastic fiber guide to transmit light to a control source where the optic intensity is linearly translated into current or is used to determine the open/close state of a current path.

### **FieldBus**

A process control local area network used for interconnecting sensors, actuators, and control devices to one another, as defined by ISA standard S50.02.

### **Flying restart**

The ability of a drive to restart a spinning motor. This is normally done by sampling the motor speed, encoder input, or back EMF to restart the motor from the speed at which it is coasting.

**Flying virtual master**

The ability of a motion controller to swap virtual encoders instantaneously making advanced synchronizing features possible.

**Following error**

The difference between the commanded position of an axis and its actual position. The amount of Following Error present varies with the speed of the axis. The amount of following error allowed can be adjusted through the KV parameter.

**Frameless motor**

A motor format which consists of only the stator and rotor provided such that a manufacturer can directly incorporate it into the framework of his machine and eliminate the need for couplings, shafts, or mechanical transmissions.

**Gantry**

An overhead framework that is designed to travel linearly in the X, Y, and/or Z axes. Tooling or other devices are generally designed into the framework to perform various functions as it moves from one location to another.

**G code**

A software used for programming machining processes. Typical applications include 3-axis milling and 2-axis wire cutting.

**Hard real-time control**

Refers to the ability of a controller to respond to an event immediately, without delay. While PLCs are inherently designed for this, PCs can be trickier. To be a hard, real-time controller, a PC-based controller's software must be considered the highest priority task, and made independent of the rest of the PCs task.

**Hardware limit switch**

A switch that is operated by some part or motion of a power-driven machine or equipment to alter the electric circuit associated with the machine or equipment.

### **Holding brake**

A positive-action mechanical friction device. Normal configuration is such that when the power is removed, the brake is set.

### **Home position**

A reference position for all absolute positioning movements. Usually defined by a home limit switch and/or encoder marker. Normally set at power-up and retained as long as control system is operational.

### **Homing**

Locating a unique reference position at power-up for axis calibration.

### **Human-machine interface (HMI)**

The console at which an operator or mechanic interacts with the controller of a packaging machine or line. An HMI, or MMI (man-machine interface) or OI (operator interface), often is a computer display with a PC or industrial computer built into or connected to run specialized HMI software.

### **IGBTI- Insulated Gate BiPolar Transistor**

Used in power section of servo drives to invert the PWM signal to the servo motor.

### **Inching**

A means of accomplishing momentary motor movement by repetitive closure of a circuit using a single pushbutton or contact element.

### **Indexer**

An electronic unit that converts high-level commands from a host computer, PLC, or operator panel into step and direction pulses needed by a stepping motor driver.

### **Indexing**

An axis or axes in the process of moving to a pre-programmed position, at a defined velocity and acceleration/deceleration rate.

### **Inertia**

Property of a mass that causes it to resist any change in its direction of motion or speed. In a servo system, the inertia of the mass being moved is reflected back to the servo motor shaft. This is called “reflected inertia.” The

servo system must be able to control this reflected inertia throughout the entire motion profile. Servo drives contain parameters to accomplish this. These parameters are Velocity Loop Proportional Gain (KP), Position Loop Gain (KV), and Velocity Loop Integral Action Time (TN). The definitions of KP, KV, and TN, can be found in this Glossary. See also: Tuning

**In Position Window**

The range of position increments in which the axis is considered by the controller to be at the commanded position point. Can be thought of in terms of +/- N position increments from the commanded position.

**Interpolation**

A coordinated move of two or more axes in a linear and/or circular motion.

**Inverter**

A drive that converts an AC, 60 Hz, power source to DC, then back to a variable frequency AC power source for a 3 phase induction motor.

**Jerk limitation**

Limits the rate of acceleration change during the movement of an axis. Its purpose is to eliminate mechanical jerking when speed changes are made.

**Jitter free synchronization**

In a master/slave configuration, it refers to the slave drive matching the speed of the master at an acceleration/deceleration rate that provides to a smooth transition.

**Jog**

An axis running at a fixed velocity and acceleration/deceleration rate, in a selected direction, with no specific destination.

**KP**

Velocity Loop Proportional Gain. Determines how much velocity error will be allowed by the servo system during a move. See also: Tuning

**KV**

Position Loop Gain. Determines how much positioning error, or following error, will be allowed by the servo system during a move. See also: Tuning

### **Length Units**

The linear units for programming and configuring an axis, typically used in indexing, setting offsets, defining overtravel limits, etc. Length units are often defined in inches, feet, meters, or millimeters.

### **Linear**

A relationship between an input and output in which the output varies in direct proportion to the input.

### **Loop Update Times**

The time interval between updates to calculate the process variable from the following error.

### **Modulo Value**

In a rotary axis, the position increment at which the axis position returns to 0, i.e. 360 degrees.

### **Noise**

An unwanted electrical signal. Typically from RFI or EMI induced onto the drive's components, speed reference or feedback wiring, and can cause the axis to react unexpectedly. Sources of noise are AC power lines, motors, generators, transformers, fluorescent lights, CRT displays, and radio transmitters.

### **Offset**

A preset distance between the actual zero reference point and a programmed zero reference point.

### **Open architecture**

Hardware and/or software designed in a way to provide interchangeability of components and connectivity from multiple vendors.

### **Open loop/close loop**

Open loop control refers to a motion control system with no external sensors to provide position or velocity correction signals. A closed loop control is a motion control system that has position and velocity feedback to generate a correction signal by comparing its position and velocity to desired parameters. Feedback devices are typically encoders, resolvers, LVTDs and/or tachometers.

**Overcurrent**

Any current in excess of the rated current of the drive to maintain or move to a new position at a given velocity and acceleration or deceleration rate.

**Override**

To force an axis to move during a faulted condition. Often required to get an axis to move off of an overtravel limit switch.

**Overshoot**

A system response where the output or result exceeds the desired value.

**Over temperature**

A warning or alarm generated by a motor or drive that indicates the device is too hot. This is generally caused by the demand for too much current through the device. There may be binding at the motor, calling for more torque, or the motor or drive may be undersized.

**PC**

Personal computer typically running under the Intel standard. PC architecture can be adapted for packaging machine control through software.

**Phasing**

Adjusting the position of one axis with respect to others during synchronization or electronic line shafting. This is usually done while the axis are moving, and done to correct for small registration problems.

**PLC**

Programmable Logic Controller is a type of computer that provides hard, real-time control of packaging and other equipment thanks to fast, repeatable deterministic scan times.

**PLS**

A Programmable Limit Switch is a dedicated, high-speed control that converts the rotary motion of a shaft into digital signals. PLS's are typically used to increase the accuracy of material or product positioning or registration.

### **Point-to-point wiring**

A method of wiring each component on a packaging machine directly to the PLC. Hard-wiring eliminates the potential for communication delays found on a network.

### **Position error**

Error caused when the difference between the actual position, and the command position is greater than a set amount.

### **Positioning**

Specifying a move by giving a target position, a velocity and an acceleration. The target position can be an absolute position, or a relative position from the current position.

### **Position loop**

Portion of the command signals that generates the position information based on position feedback.

### **Printmark synchronization**

Feature that captures the position of a passing mark on the product and then compares this position with the expected position based on current speed, and then compensates for this difference.

### **Profile**

Graphical representation of movement. This can be position vs. time, velocity vs. time or torque vs. time.

### **Programmable Limit Switch**

See PLS

### **Programming language**

Interface that allows the user to control the motion system according to the demands of the user.

### **Protocol**

A particular method of encoding either analog or digital information for transmission over a cable. Often used interchangeably with Standard.

**Pulse width frequency**

The rate at which the IGBTs can switch.

**Pulse-width modulation**

A switch-mode control method used in amplifiers and drivers to control motor voltage and current to obtain higher efficiency than linear control. PWM refers to variable on/off times (or width) of the voltage pulses applied to the transistors.

**Quadrature**

A technique that separates signal channels by 90° (electrical) in feedback devices. It is used with encoders and resolvers to detect direction of motion.

**Ramp function generator**

Mathematical model that provides a square wave, triangular wave or sinusoidal wave output.

**Rated speed**

The maximum speed at which the servo motor can rotate.

**Real master**

Physical feedback which provides position information for a synchronized axis to follow.

**Rectifier**

Device that transforms AC power into DC for use by converter drives.

**Referencing**

Procedure to set the feedback device relative to the real world.

**Regen**

Power generated by a motor/drive system during the deceleration phase of movement. In some systems, this regen power can be used by other axis or put back on the network.

**Resolver**

A position transducer that uses magnetic coupling to measure absolute shaft position during one revolution.

### **Rollfeed**

Function that calculates speed of a rotary axis to keep the linear speed of the feed material constant as the diameter of the rotary axis changes.

### **Rotary**

Moving in a circular way, using degrees to indicate position instead of mm, or inches in a linear axis

### **Safe off**

Procedure that ensures that power will not travel from the drive to the motor.

### **SCADA-Supervisory Control & Data Acquisition.**

Refers to software and hardware that (1) permits the control or management of an entire packaging line and (2) automatically collects data on that line's efficiency.

### **S curve**

S curve refers to a control pattern that accelerates and decelerates a motor slowly to reduce mechanical shock. This function is more sophisticated than linear acceleration, but does not have the performance of camming.

### **Sequence of operation**

A series of steps to be executed that then causes an action in a machine.

### **SERCOS**

Serial Real-time Communications Standard. An open communications protocol (adopted as IEC 1491) designed especially for motion-control networks. Defines a method for transmitting digital information over a fiber-optic cable at speeds of 2, and more recently, 4 megabits/sec.

### **Serial communications**

Transmitting digital 1s and 0s in a series over a single cable, the primary method of communication used in and between packaging equipment. Parallel communications use several wires to simultaneously transmit groups of 1s and 0s.

**Servo mechanism**

An automatic, closed-loop motion control system that uses feedback to control a desired output such as position, velocity, or acceleration.

**Servo motor**

A motor that together with its resolver or encoder is capable of being precisely controlled. A resolver or encoder provides constant and highly accurate feedback on the motor's exact position, speed and torque to the drive that powers it.

**Shielded cable**

A cable that has a metallic sleeve wrapped around all of the conductors that comprise its center. The metal sleeve is then grounded to eliminate the effects of electrical noise on the signals being carried by the cable.

**SinCos**

An encoder that outputs both digital and high resolution analog signals used in servo control in packaging machines.

**Software limit switch**

A software function that turns physical outputs on and off, depending on the level of a specified input. Servomotors, resolvers or encoders usually offer the input for software limit switches.

**SSI**

Acronym for Serial Synchronous Interface. This is a type of multi-turn absolute encoder. The position information is sent from the encoder to the device reading the encoder as a serial string in Gray code format.

**Synchronization**

The condition that occurs when several functions of a machine (mechanical, servo or software) follow a common control signal and are in a specific position according to this signal.

**Tachometer**

An electromagnetic feedback transducer providing an analog voltage signal proportional to rotational speed.

**Task**

A software system control that determines the execution rates and priority levels for software modules running in a motion control or PLC.

**TCP/IP**

Transmission Control Protocol/Internet Protocol. A method of encoding data into a series of "packets" for transmission over a network. Designed initially for use on the Internet, TCP/IP is rapidly penetrating non-Internet uses, including the factory floor.

**Teach position**

The position of an axis that is "taught" into the motion control program. The axis is moved, typically by jogging, to the desired position. This "teach position" is then entered into the motion program automatically by the control (using whatever steps are required by the motion control manufacturer) and becomes the new programmed position. The motion control used must have the ability to do this type of program manipulation.

**Telegram**

Communication data packet between controller and device.

**TeleService**

The ability to remotely access a motion control or PLC for service purposes.

**TN**

Velocity Loop Integral Action Time. Associated with KP. When velocity error occurs outside of the tolerance value set in KP, TN determines how quickly the servo drive will bring the velocity back within the specified tolerance. See also: Tuning

**Torque limitation**

A servo function that allows the monitoring and limiting of the current supplied to a servo motor.

**Tuning**

Adjusting the servo drive's internal characteristics to give it the ability to control the reflected inertia and gives the axis a smooth position/velocity profile. The process of Tuning involves setting the Velocity Loop Proportional Gain (KP), Position Loop Gain (KV), and the Velocity Loop Integral

Action Time (TN) values so that the axis has a position/velocity profile allowing only as much position/velocity error as the process will permit.

**Twisted pair**

Two wires twisted together for the purpose of eliminating the effect of electrical noise.

**VxWorks**

VxWorks is a real-time operating system that guarantees an absolutely deterministic response. It is increasingly used in motion control because of its real-time behavior, stability, operating time, and memory efficiency. (Dev. by Wind River Systems)

**Velocity**

The speed at which a motor or mechanical system runs.

**Velocity loop**

A servo control function that sums a velocity command signal with a speed feedback signal from a servo motor, and outputs the difference as a torque command signal.

**Virtual master**

An encoder signal created in the software of a motion control to allow synchronizing multiple servo systems. A typical machine may have several virtual master encoders.

**Warning**

The error condition received from a drive or a controller that a problem, if not remedied within a specified period, will result in a fault. The controller or drive is generally not disabled until a fault condition occurs.

**Wintel**

The defacto industry standard for PCs, referring to Microsoft's Windows™ operating system running on Intel's microprocessors.

**Zero point of feedback**

The point at which a servomotor's encoder position and the machine's physical position line up. If these two points don't agree, the servo axis must be "homed."

## **Resource List**

*Packaging World*, October 1998

*Rockwell Automation Motion Book Version 3.0*, September 4, 1999

*Design News*, June 1999 Semiconductor Manufacturing

*NFPA 70 Electrical Standard for Industrial Machinery*, 1997 Edition

*IEEE Standard Dictionary of Electrical and Electronic Terms*, 1972

*Control Engineering*, Terminology in Motion, January 8, 1998

## **Appendix IIA: Line Types v1.1**

---

This document was compiled by the OMAC Packaging Workgroup PackML Team and first published on September 7, 2001. Please submit any comments to:

Fred A. Putnam, PackML chair  
MARKEM Corporation  
150 Congress Street  
Keene, NH 03431  
USA  
Phone: 978-886-9237  
Email: fputnam@markem.com

### **Executive Summary**

Packing machinery is procured by end-users from a variety of specialist manufacturers from across the world. A major issue arises when machines from different manufacturers, sometimes built according to different standards, are integrated into a single packaging line. The integration process can be time consuming and costly and can introduce significant risk in new product introduction or innovation projects where cutting the time to market has major revenue implications.

A major aid to the improvement of packaging line integration projects is an agreed standard interface between the individual machines which comprise a packaging line. These interfaces can be implemented in a number of ways, utilizing varying degrees of technical sophistication. The first step towards defining a standard inter-machine interface is to agree an outline definition of different packaging line types.

To this end four different line types have been proposed. These are based on configurations which are commonly seen in the packaging field and are intended to be a representative overview for discussion purposes. These proposed Line Type definitions arise from the PackML group's Integration Brief paper V2.0 and the architecture definitions published by the Motion for Packaging subcommittee working on Connectivity and Architecture. The

four typical packaging line configurations or Line Types have been defined below.

Line Type 1	Machines are autonomous; they react to plant conditions through their own array of sensors and switches. They do not communicate to the other machines that comprise the line
Line Type 2	<p>Line Type 2 machines differ from Line Type 1 because they have the ability to communicate with other machines</p> <p>Two subtypes have been defined: 2A and 2B. Functionally these are identical, they achieve their functionality through two different methods:</p> <p style="padding-left: 40px;">Machines within Line Type 2A communicate through digital and analog I/O</p> <p style="padding-left: 40px;">Line Type 2B machines communicate across data networks with or without digital and analog I/O.</p>
Line Type 3	<p>An enhanced version of Line Type 2B</p> <p>Contains both, or a combination of, SCADA server and clients and/or line supervisory controller</p> <p>The enhanced functionality of this Line Type provides line performance data, loss analysis, root cause analysis toolkits, maintenance and troubleshooting tools, change parts database, etc.</p> <p>Individual packaging line networks could be bridged to provide data acquisition and display for the whole factory</p>
Line Type 4	<p>Packaging lines are integrated into wider business IT systems via the ERP Bus</p> <p>This provides functionality such as the passing of production orders into the factory and progress against the order to be monitored. It could also allow the automatic reporting of performance, quality and material usage.</p>

### **Packaging Line Types**

## **Introduction**

The objective of this document is to discuss packaging line arrangements and to propose typical line types, which represent configurations commonly seen within the packaging field. The intention is to provide an overview of typical arrangements; the objective is not to define all possible configurations nor to rigorously specify each component machine within each line type.

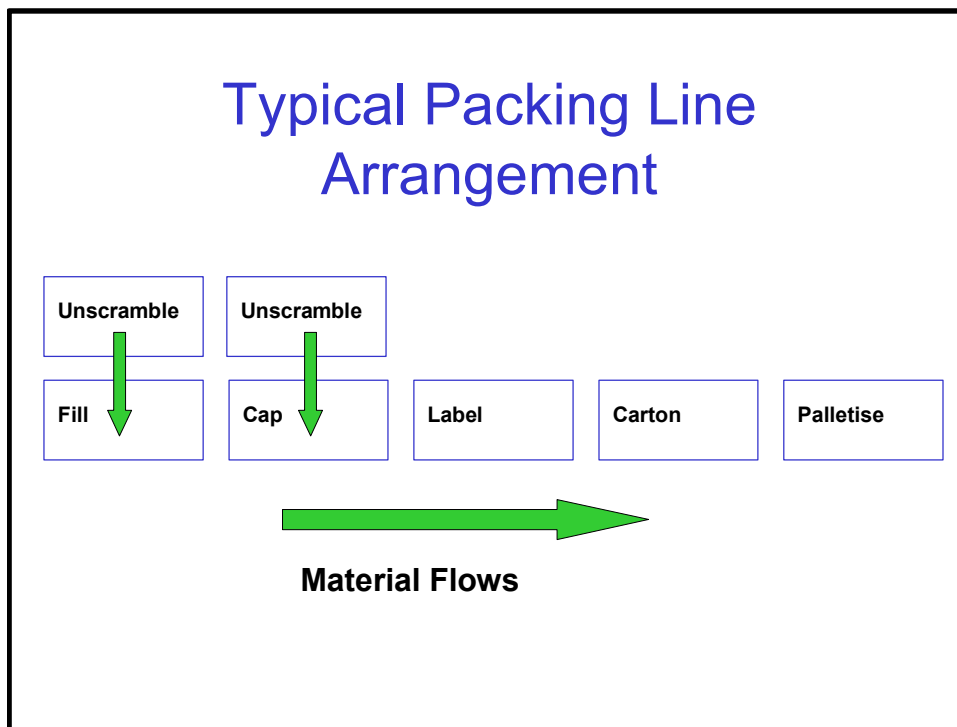
This work builds on the line type definitions stated within the PackML Integration Brief document (V2.0) and references the output of the Connectivity and Architecture subcommittee of the Motion for Packaging Working Group.

Section 3 provides background on the reasoning behind the packaging line Types. Section 0 provides an overview of the four line types being pro-

posed in this document. Sub sections then provide more detail for each particular type. Conclusions resulting from this work appear within Section 0.

## Line Type Overview

A typical packaging line configuration is shown in Figure 2 below. It is comprised of a number of individual machines, which perform one, or a small number of specific tasks. Usually these machines are procured from manufacturers with a high degree of specialization in their field. A major issue for end-users of packing machinery is the installation and integration of individual specialist machines from different manufactures, often from different parts of the world, into a fully functioning integrated packaging line. The process of integration can be time consuming and costly and introduce significant risk into innovation or new product introduction projects where cutting time to market can have major revenue implications.



**Figure 2: Typical Packing Line Arrangement**

A significant aid to the integration of machines into a packaging line is an agreed or standardized interface between individual machines. Such interfaces can be implemented in many different ways using varying degrees of technical sophistication. A first step towards defining inter machine interfaces is an outline definition of line types. This will provide an overview of

the manner in which machines will communicate with each other, the kind of data which will be exchanged as well as an exploration of the potential uses of that data.

## Line Type Definitions

Four packaging line types are proposed and these are discussed in more detail below. These definitions are meant to be indicative of common configurations which are seen throughout the packaging field.

The line type definitions describe an evolutionary process where increased functionality can be incorporated with higher order line types. For consistency it is assumed that where an actual line consists of elements from more than one line definition, the highest order line type is used to describe the line.

### Line Type 1

In this example each of the packing machines and transport systems are not networked together. Instead each component in the line has its own discrete sensors which are used by each of the machines to control their own operation. Figure 3 below shows an example of two adjoining machines within a packaging line Type 1.

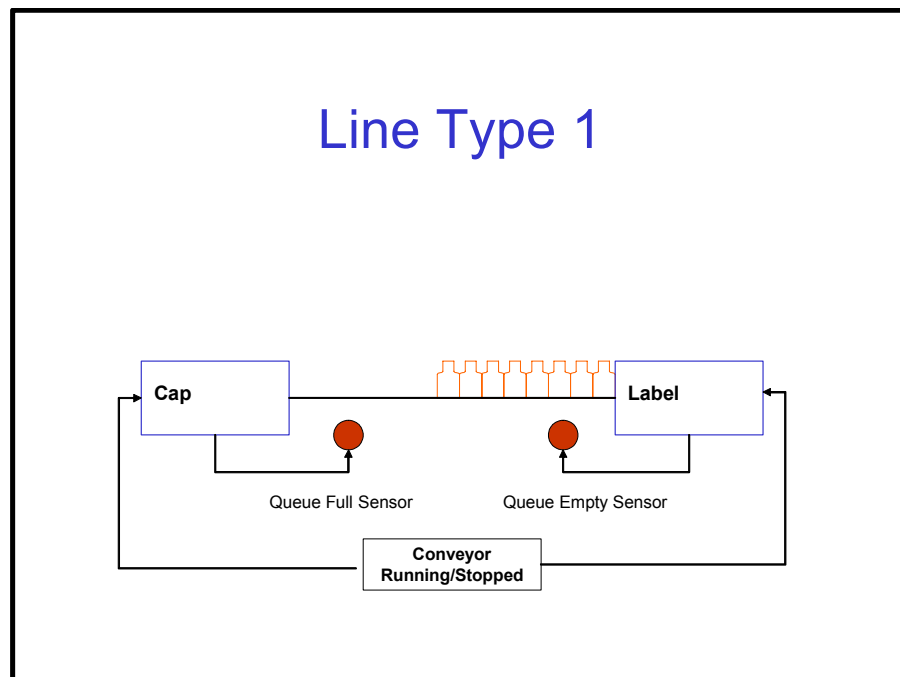


Figure 3: Line Type 1 Example

In the example the conveying system between the capping and labeling machines provides a Running/Stopped status indication. Sensors along the conveyor are used as inputs to the individual packing machines, which respond accordingly. In the example the labeling machine, queue empty sensor is indicating that there are bottles available for filling. In response to NOT Queue empty the machine's speed could be ramped up from zero to its set speed and processing could continue. The presence of Queue Empty could then cause the machine's speed to ramp down to zero.

In the case of the capping machine the Queue Full sensor's signal could cause the capping machine's speed to be ramped down to a low speed setpoint.

In a Ttype 1 line the operational status of each machine is not available to other machines in the line. The performance of an individual machine is adjusted in response to its own sensors. In essence line type 1 machines are autonomous "islands" of operation, they respond to plant conditions through their own array of sensors. There is no communication between machines and they therefore operate reactively to plant conditions.

## **Line Type 2**

Machines within line Type 2 differ from Type 1 because they have the ability to communicate with each other. It is foreseen there is an opportunity to define two sub-types within Line Type 2. Line Type 2A achieves communication through the use of I/O exchange and Line Type 2B does so across a data network.

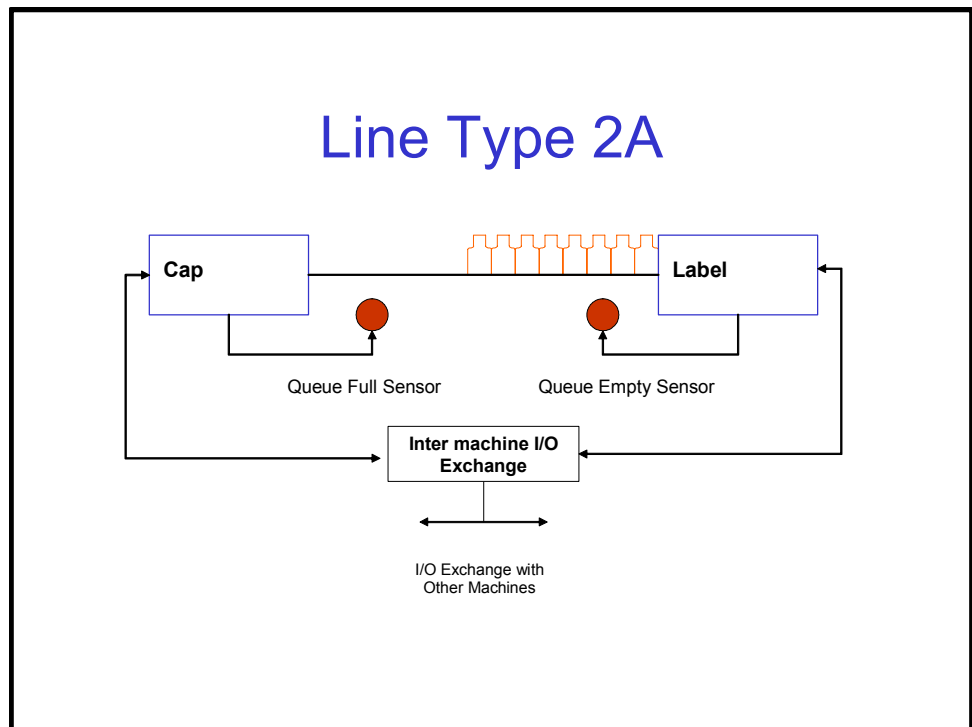
Functionally, Line Types 2A and 2B are identical i.e. they achieve communication with other machines either upstream or downstream. The only difference is the mechanism by which communication is achieved.

The migration path from Line Type 1 to Line Type 2A is relatively simple to achieve, through the addition of extra I/O capacity and some controller software modifications.

The use of a data network (Line Type 2B) provides more communications flexibility. However such line types do require additional engineering effort in their design and, due to the use of network technology, will increase the maintenance and operations overhead to keep them operating.

### Line Type 2A

In this example each machine is connected to its immediate upstream and downstream counterparts by means of discrete I/O exchange. For each machine the interfacing I/O must be defined and spare I/O capacity must be built into the control system of each machine. This type of approach will lead to the need to run a number of multi-core cables along the line to carry the I/O signals and suitable provision must be made for the definition of signal quality, power supplies and electrical isolation. It would also be beneficial to nominate a preferred plug and socket standard.



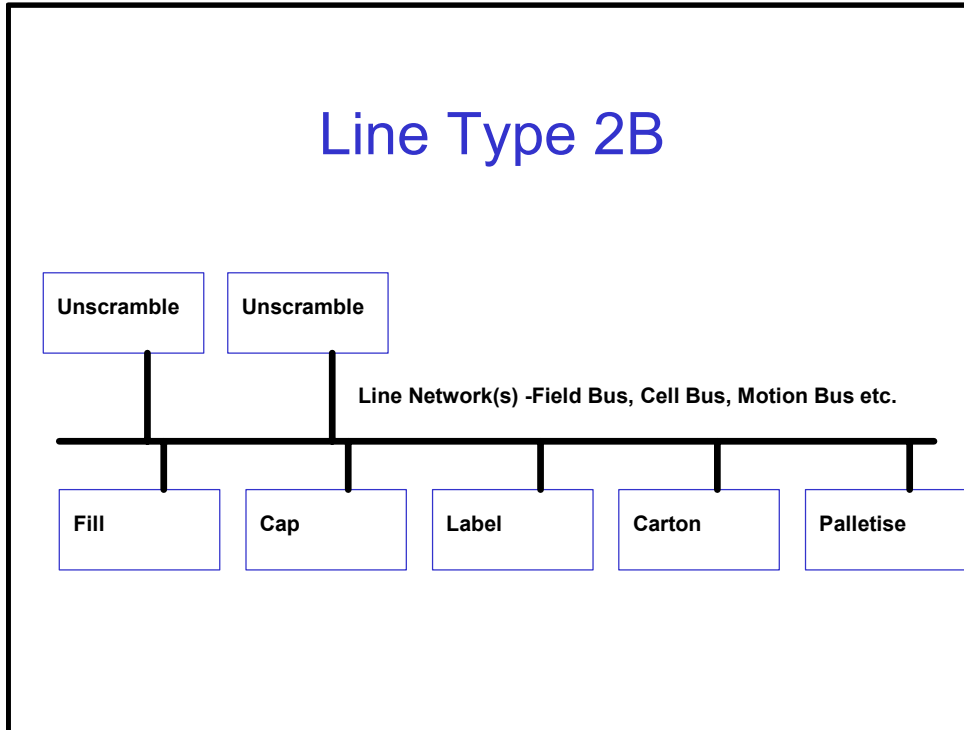
**Figure 4: Example of Line Type 2A**

Whilst this approach is simple, relying as it does on the exchange of I/O signals, it does increase the I/O requirement for each machine, signal marshalling must also be considered as well as the routing of the signal carrying cables. It is envisaged that communication will be achieved using a combination of digital and analogue signals where appropriate.

### Line Type 2B

In this example each machine is connected across a packaging line network and has access to a specified section of data in any machine across the line. Figure 5 below shows the concept of Line Type 2B. In this example, the machines have been shown as being connected to two networks. The line

network(s) provides data exchange and interlocking between controllers and devices with real-time and non real-time requirements. Some applications will have requirements for multiple bus/network types so the application will dictate the actual infrastructure requirement deployed.



**Figure 5: Line Type 2B Example**

To realize this type of configuration communications standards must be agreed as well as the data which will be shared between the machines. Typically available data could be:

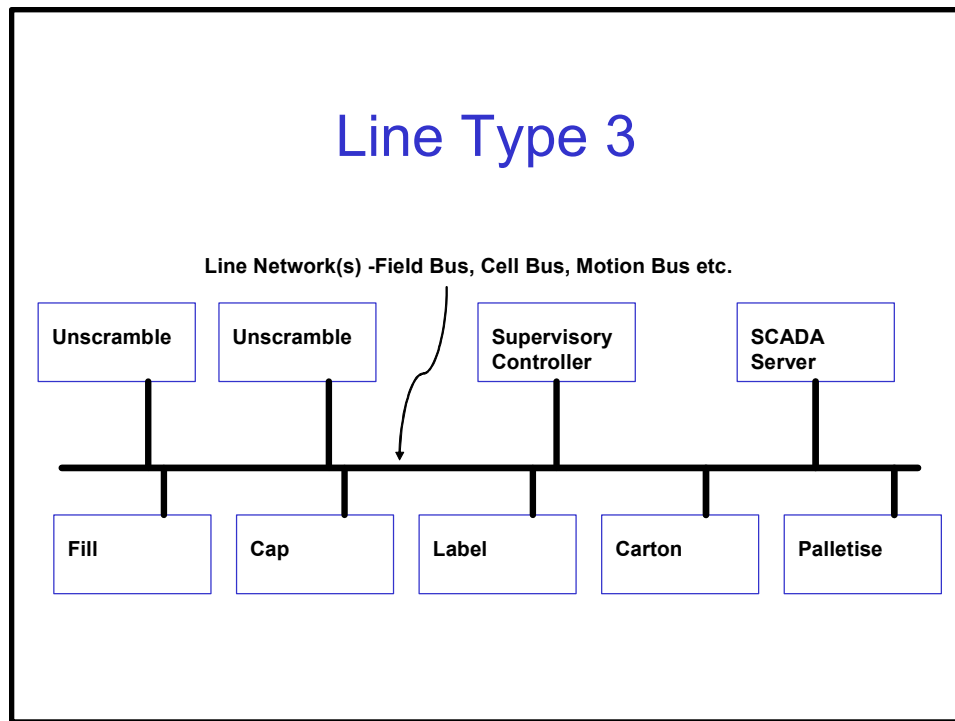
- Machine states and modes
- Operating Speed
- Alarms and alerts

This type of network could also provide remote access to an array of diagnostic and performance data embedded within the drives and controllers of each machine. This could be used to aid maintenance and troubleshooting. Each machine has its own individual HMI, however key status/alerts could be shared across the network and displayed on all HMI terminals, where appropriate. HMI configuration for this line type could consist of pushbuttons, lamps and text display units or could be text or semi graphic HMI panels.

### Line Type 3

This is an enhanced version of line type 2B. The line network has been augmented with a SCADA system which could provide overall line monitoring, throughput, efficiency, quality data etc. It could also be used for monitoring and analysis of losses and breakdowns. This system could also house some maintenance tools which will assist in root cause analysis of breakdown problems and may also house a database of changeover and spare parts. SCADA clients could also be located along the line to give access to information to operators and technicians local to where they are working.

The packaging line networks could also be interconnected to provide a complete factory overview from a central overview SCADA system.



**Figure 6: Example of Line Type 3**

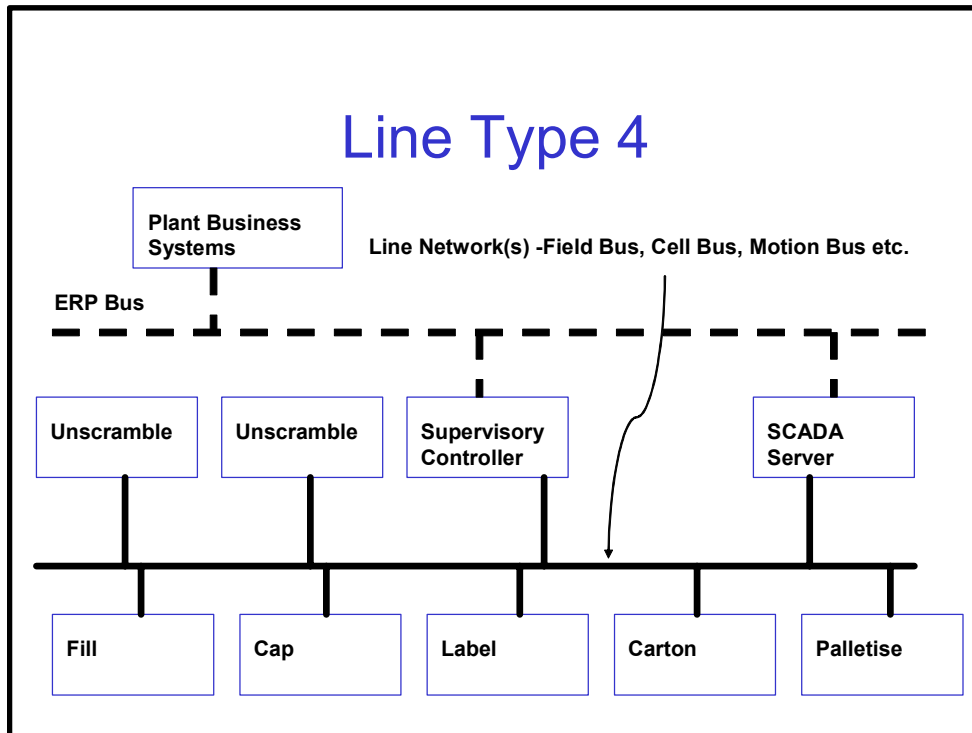
In some lines there could also be another PLC/Soft PLC which provides supervisory control of the complete line, controlling a co-ordinated, automated sequenced start-up for all machines for example.

### Line Type 4

In this example the packaging line systems are integrated into the plant's business systems. This will allow production order data to be passed down to the packing area and progress against the order along with material con-

sumption and quality information being returned automatically. Integration into wider business systems is via the ERP Bus (as defined by Connectivity and Architecture sub committee).

The Supervisory Controller or the SCADA system will facilitate the interaction between the Plant Business System and the packaging line, for the sake of further discussion these systems will generically be referred to as the Line Supervisor. The Line Supervisor will be responsible for acquiring, aggregating and processing all relevant line data and then producing the reports which will be sent to the Plant Business System via the ERP Bus. It will also receive orders or production plan data from the Plant Business System and will enable those plans to be executed.



**Figure 7: Example of Line Type 4**

The actual configuration of the Line Supervisor will depend upon the application. It will bridge the real-time, and the high data volume environment of the Line Networks with the transactional world of the ERP Bus. This will be in addition to its data acquisition, processing and display duties.

## Conclusions

Four distinct typical packaging line configurations of Line Types have been defined

Line Type 1:

- Machines are autonomous; they react to plant conditions through their own array of sensors and switches. They do not communicate to the other machines which comprise the line

Line Type 2:

- Line Type 2 machines differ from Line Type 1 because they have the ability to communicate with other machines
- Two subtypes have been defined, 2A and 2B. Functionally these are identical, they achieve their functionality through two different methods
- Machines within Line Type 2A communicate through digital and analogue I/O
- Line Type 2B machines communicate across data networks with or without digital and analogue I/O.

Line Type 3:

- This is an enhanced version of Line Type 2B
- It contains both (or a combination of) SCADA server and clients and/or line supervisory controller
- The enhanced functionality of this Line Type provides line performance data, loss analysis, root cause analysis toolkits, maintenance and troubleshooting tools, change parts database etc
- Individual packaging line networks could be bridged to provide data acquisition and display for the whole factory

Line Type 4:

- Via the ERP Bus packaging lines are integrated into wider business IT systems

- This provides functionality such as the passing of production orders into the factory and progress against the order to be monitored. It could also allow the automatic reporting of performance, quality and material usage

## **References**

PackML Integration Brief V2.0

Connectivity and Architecture Subteam Overview

## **Appendix IIB: PackML State Model v2.2: Automatic Mode Machine States Definition**

---

This document was compiled by the OMAC Packaging Workgroup PackML Team and first published on March 11, 2002. Please submit any comments to:

Fred A. Putnam, PackML chair  
MARKEM Corporation  
150 Congress Street  
Keene, NH 03431  
USA  
Phone: 978-886-9237  
Email: fputnam@markem.com

### **Executive Summary**

Since its inception, the PackML group has been using a variety of information sources and technical documents to define standard terminology to be applied in the integration of packing machines.

Until now, there has not been agreed definition of the following:

1. Packing machine state types
2. Packing machine operating modes
3. State models, State descriptions and transitions

### **Machine State Types**

A State completely defines the current condition of a machine. Four machine state types are proposed:

- No Command State is one which, after completing its own logic, forces automatic transition to a Final State
- Final State represents a safe state, i.e. no moving parts
- A Transient State is one which represents some processing activity
- A Quiescent State is used to identify that a machine has achieved a defined set of conditions

## Machine Operating Modes

A Mode determines how a machine will operate in response to the commands which are issued to it.

- For packing machines the principal operating mode is Automatic. This is utilised for routine production
- Other machine operating modes do exist examples being Semi-automatic, Manual, Maintenance etc

## State Models

A State model has been proposed for Automatic mode; details of the states and the conditions that force transition between states have been provided.

Work done to date concentrates on Automatic mode, its State model and the commands and machine status which force transition between modes. Further work to define other modes of operation and state models, where appropriate; will be carried out in the future.

The concepts proposed uses previous PackML work as its foundation and sets out to further enhance the work done to date.

## Introduction

The objective of this document is to provide a definition of a state model corresponding to packing machine operations in Automatic mode. It builds on previous machine state definition work carried out by the PackML group. Other machine modes such as Semi-automatic, Manual and Maintenance etc are not considered here. Definition of machine modes and state models, where appropriate, will be the subject of future work.

Section 0 provides a series of definitions which are used later in this document. Four different state types are described and named.

A state model for routine, automatic, machine operation is proposed in Section 0. A diagrammatic representation of the state model is provided along with a more detailed description of each state and the conditions which force transition between states.

Conclusions are summarised in section 0.

## Definitions

This section provides descriptions of some of the terms which are used throughout the document. Section 0 provides definitions of machine states; this illustrates the different types of states and how transitions between states are forced. In Section 0 automatic operating mode is discussed.

## **States**

A State completely defines the current condition of a machine

Transitions between States occur:

- As a result of a Command
- As a result of a Status change. This is generated by change of state of one or a number of machine conditions, either directly from I/O or completion of a logic routine
- Automatically after completing a No Command State
- A No Command State is one which, after completing its own logic, forces automatic transition to a Final State
- A Final State represents a safe state, i.e. no moving parts
- A Transient State is one 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
- A Quiescent State is used to identify that a machine has achieved a defined set of conditions. In such a State the machine is holding or maintaining a status until transition to a Transient State.

## **Modes**

A Mode determines how a machine will operate in response to the commands which are issued to it. This document focuses on one operating mode, Automatic. Other modes of operation do exist across the packing industry, these have various names: Semi-automatic, Manual, Maintenance Index etc. Packing operations in modes other than automatic will be the subject of further work.

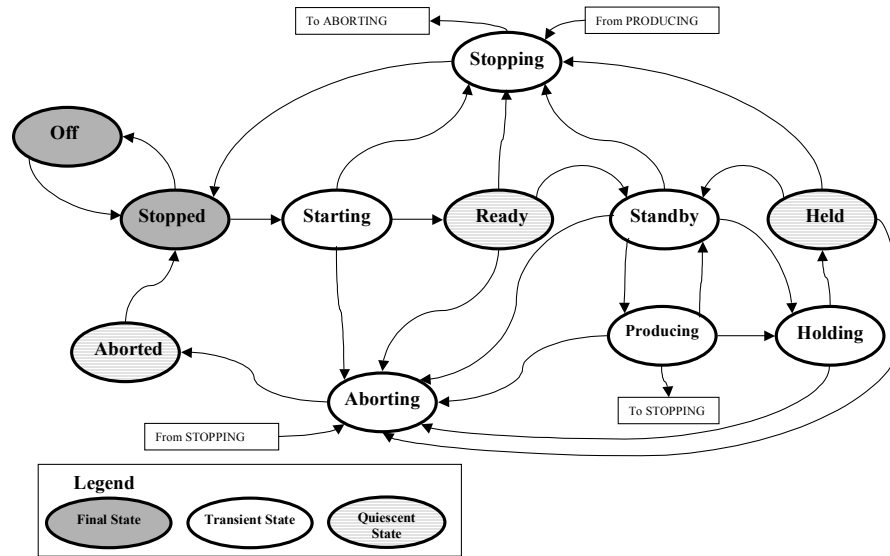
### **Automatic Mode**

This represents the mode which is utilised for routine production. The machine executes relevant logic in response to commands which are either entered directly by the operator or issued by another supervisory system.

## **Automatic Operation State Model**

The proposed Machine State model for the operation of a machine in Automatic mode is depicted in Figure 1 below.

### State Model - Automatic Mode



**Figure 1: Machine State Model for Automatic Mode Operation**

A brief description of the individual machine states appears in Table 1 below:

State Name	State Type	Description
OFF	Final	<p>All power to machine switched off. This state is assumed if there is no response from the machine. This is an optional Final State.</p> <p>The OFF state may be reached in two ways, first the power to the machine is switched off and second power is removed as in a power utility failure. In the OFF state it must be assured that the machine will go at the least to the initial stopped state when power is restored either by switch or restoration of the utility power source.</p>
STOPPED	Final	The machine is powered and stationary. All communications with other systems are functioning (If applicable).
STARTING	Transient	This state allows the machine to be prepared for running. This could include such processes as heating, self-testing or calibration.
READY	Quiescent	This is a State that indicates that STARTING is complete. This state maintains the machine conditions that were achieved during the STARTING state.
STANDBY	Transient	<p>The machine is running at the relevant setpoint speed, there is no product being produced.</p> <p>This state can be reached either in response to a Start Command from READY or any internal machine logic that would dictate a temporary transition from the PRODUCING state, such as materials runout.</p> <p>The status "materials runout" could refer to lack of materials on the machine's own infeeds or could refer to a stoppage of a downstream machine.</p>
PRODUCING	Transient	Once the machine is processing materials it is deemed to be producing.
STOPPING	No Command	This state executes the logic which brings the machine to a controlled and safe stop
ABORTING	Transient	The ABORTED state can be entered at any time in response to the Abort command or on the occurrence of a machine fault. The aborting logic will bring the machine to a rapid, controlled safe stop. Operation of the Emergency Stop or "E-Stop" will cause the machine to be tripped by its safety system. It will also provide a signal to initiate the ABORTING State.
ABORTED	Quiescent	This state maintains machine status information relevant to the Abort condition. The Stop command will force transition to the Stopped state
HOLDING	Transient	When the machine is STANDBY or PRODUCING the Hold command can be used to start HOLDING logic that brings the machine to a controlled stop.
HELD	Quiescent	The HELD state would typically be used by the operator to temporarily hold the machine's operation while material blockages are cleared, or to stop throughput while a downstream problem is resolved.

**Table 1: Automation Operations Machine States**

An example State transition matrix for Automatic Mode is shown below. Note that the State Model does not rigidly specify the internal machine state transition logic, which will vary depending on the application.

	Machine Status						Commands				
	Power On	Power Off	Materials Ready	Materials Runout	Machine Fault	State Complete	Prepare	Start	Stop	Hold	Abort
<b>Initial State</b>											
OFF	STOPPED										
STOPPED		OFF					STARTING				
STARTING					ABORTING	READY			STOPPING		ABORTING
READY					ABORTING			STANDBY	STOPPING		ABORTING
STANDBY			PRODUCING		ABORTING				STOPPING	HOLDING	ABORTING
PRODUCING				STANDBY	ABORTING				STOPPING	HOLDING	ABORTING
STOPPING					ABORTING	STOPPED					ABORTING
HOLDING					ABORTING	HELD					ABORTING
HELD					ABORTING			STANDBY	STOPPING		ABORTING
ABORTING						ABORTED					
ABORTED									STOPPED		

**Table 2 : Automatic Mode State Transition Matrix**

## **Automatic Mode Comments**

Some further explanatory notes have been included to further clarify some elements of the State Model.

### **ABORTING, ABORTED, and E-Stop**

The ABORTING State has been included for the following reasons:

- This is the location of the shutdown logic which executes on either an ABORT command or a machine fault. The ABORTING State can be entered from any machine state (except STOPPED, ABORTING and ABORTED)
- The ABORTED State is a quiescent state which indicates that the ABORTING logic has completed. It holds the machine in a safe state but it also holds any relevant data which will help in diagnosing the fault or helping to reconcile production information after a major event.
- The manufacturer of the specific packing machine develops the logic, which generates the Machine Fault command. This is derived from a number of sensor signals or internal flags as appropriate. For example, if the Emergency Stop or “E-Stop” pushbutton is depressed the hardwired stop system is initiated and the machine is brought to a rapid stop by the operation of the emergency stop system. Generally this is achieved by hardwired interlocks which operate independently of the machine's control system. One of the conditions which would result in the setting of the Machine Fault flag would be Emergency Stop System operated. This would ensure that, whatever its current state, the machine would make a transition into ABORTING and then to ABORTED. In this way the machine's software based control system would reflect the condition of the machine which was brought about through the operation of the hardwired system.

### **Notes about Machine Start Up**

In order to start a machine from the STOPPED state the operator needs to issue the Prepare command and once the machine is READY the Start Command will allow the machine to start running. In some lines a single operator may be responsible for the supervision of a number of machines which may all be locally started. On one pass down the line he can prepare all of the machines for operation and once all are confirmed READY he will

then be able to start individual machines in the correct order. For those lines, which have supervisory control, the Prepare command, can be issued at all machines at once and then the correct starting sequence can be initiated once all are confirmed READY.

### **State Model**

A Quiescent State, READY, has been introduced after completing the STARTING state. This state maintains the operating conditions of the machine until the Start command is issued and essentially it represents that the STARTING State has been completed.

The ABORTING state, in this document, is entered on occurrence of a machine fault or in response to the Abort command. Running out of packing materials will not force transition to ABORTING. It is proposed that ABORTING will cause the machine to be brought to a rapid controlled stop, the completion of this will result in the newly added quiescent state ABORTED.

Table 2 defines the commands and machine status, which force transitions between machine states. Individual machine designers have complete freedom to define the logic and conditions which drive those transitions. Table 2 shows that in order for a machine to enter the STANDBY State from STOPPED, two commands must be issued. Firstly the Prepare command will force transition to STARTING, the machine will run its initialisation logic and once complete will enter the transient state READY. It will remain in that state until it receives the Start command which will force transition into STANDBY.

In some cases the machine designer may not want to wait in the Ready State before entering STANDBY. In this case the Start command may automatically be generated, by internal logic, on detection of the machine's current state being Ready. This approach means that the state model remains standard, the command and status names remain standard but machine designers can engineer flexibility by logical manipulation of machine conditions.

### **Conclusions**

Four state types were defined to cover differences in the functionality of machine states. These have been called Final, Transient, Quiescent and No Command.

A state model for Automatic mode has been provided. It is recognised that other machine modes are commonly used throughout the industry. Further work will identify these standard modes and will provide state models where appropriate.

The ABORTED State has been included in order to allow all internal diagnostic data to be held available for analysis before the machine is returns to the STOPPED State

The HOLDING and HELD states have been added to allow a machine's operations to be temporarily halted

Required operational flexibility can be ensured by using standard machine states, commands and status and allowing designers to use their own logic to drive machine state transitions.

This work uses previous *PackML* work as its foundation and sets out to further enhance the work done to date

## **References**

*PackML* Integration Brief V2.0

*PackML* State Definition Drafts

## **Appendix IIC: PackML State Model S88 Software Compatibility**

---

This document was compiled by the following OMAC Packaging Workgroup PackML subteam and first published on March 22, 2002:

Andrew McDonald, Unilever

Rick Morse, Rockwell Automation

Fred Putnam, Markem

### **Summary**

PackML has developed a proposed mapping of the S88 Batch Control state model command set to the PackML state model. This should pave the way for S88-based software to be extended from batch control to packaging applications.

### **Background**

PackML is a subteam of the OMAC Packaging Workgroup. At the February PackML meeting, Rick Morse presented to PackML a compelling argument that PackML should find a way that existing S88-based software could be used with the PackML state model.

At the March PackML meeting, this was discussed, and it was decided that Rick, Andrew McDonald, and Fred Putnam would meet by telephone on Wed. March 20th to the feasibility of this in detail. This is a report of the results of that meeting.

### **Comparison of State Models**

Comparing the S88 and PackML state models, the following state mapping between the two is apparent:

<b>S88 states</b>	<b>PackML states</b>
Running	Producing
{Pausing -> Paused}	Standby
Idle	{Stopped -> Starting -> Ready}
-	Off
Complete	Stopped
Restarting	{part of Starting}

#### **Comparison of S88 and PackML States**

This comparison is not really critical to the objective of this work, which is mapping the S88 commands to the PackML state model. It's utility is that it enables us to see what how the commands should be mapped.

#### **Command Mapping**

The S88 command set is as follows:

{Start, Hold, Restart, Pause, Resume, Stop, Abort, Reset}

and the PackML command set is as follows:

{Prepare, Start, Stop, Hold, Abort}

The commands these have in common are Start, Stop, Hold, and Abort. Three of these, Hold, Stop, and Abort, are trivially mapped between the two models, as they have essentially the same effect in each. The other S88 commands are effectively mapped to the PackML state model in the following table, where the two command sets are prefixed by S88\_ and PackML\_ to make it clear which model they belong to.

<b>S88 Commands</b>	<b>Effect in the PackML State Model</b>
S88_Start	PackML_Prepare, then PackML_Start
S88_Hold	PackML_Hold
S88_Restart	Similar to the S88 description, as follows: Orders the machine to execute its Restarting logic to safely return to the Producing state. Only valid while the machine is in the Held or Aborted state.
S88_Pause	transition from PackML_Producing to PackML_Standby
S88_Resume	transition from PackML_Standby to PackML_Producing
S88_Stop	PackML_Stop
S88_Abort	PackML_Abort
S88_Reset	Transition to PackML_Stopped

#### **S88 Commands and their Effect in the PackML State Model**

Note that commands are defined such that they can be issued from logic that is internal to the machine controller, from the machine operator interface, or from a supervisory control system.

### **Implementation**

The mapping shown above will have greatest utility when it enables packaging machinery that incorporates the PackML state model to be used in conjunction with (and be controlled by) preexisting software that is based on the S88 state model. Therefore, to achieve S88 compatibility, PackML state model implementers will need to add the S88\_xxx commands as shown above.

### **Rationale**

Rationale for PackML State Model S88 Software Compatibility, from the minutes of the PackML February meeting:

- Several companies are offering very sophisticated batch software packages that are based on the S88 specification. This includes Rockwell as well as Emerson, Intellution, and others.
- S88, and the batch software, provides a separation of the machine control from the recipe. It's also used in materials tracking.

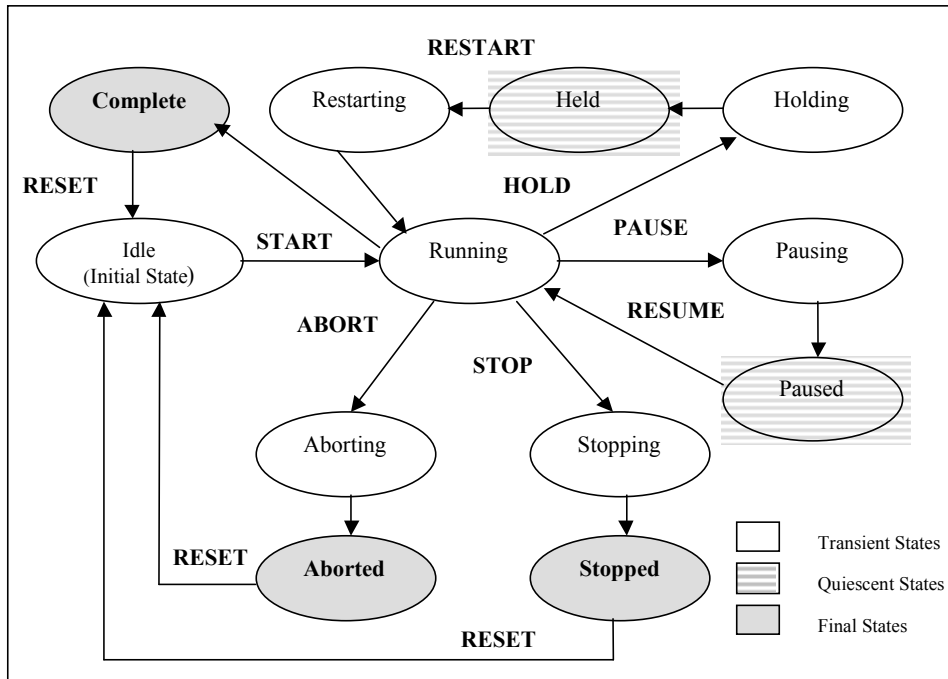
- It would be very attractive if this batch software could be extended from the batch manufacturing operations to the packaging lines for the same products. This would be attractive in part because one could carry all the information about the product down to the packing line.
- Some of the S88 based batch software has features that implement the new FDA 21 CFR Part 11 regulations, including electronic signatures, data tracking, and safety issues.
- These batch software products make use of the S88 state model, which is close to but not identical to our state model. Consequently, our state model wouldn't fit under an S88 sequencer.
- Rick would like us to work on, and has joined our team to help us work on, finding a way to enable our state model to be compatible with this batch software.
- At present, the best way to do this seems to be to define a mapping of the S88 command interface to our state model.

### **Reference List**

*Applying S88* by Jim Parshall and Larry Lamb, ISA Press

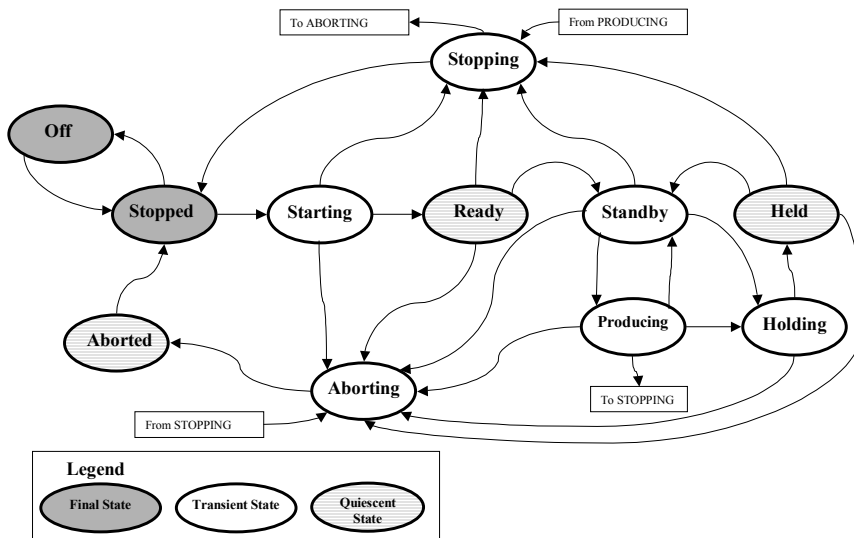
PackML Machine State Model Guidelines V 2.2:

[www.packml.com/PackML\\_Machine\\_State\\_Model\\_Guidelines\\_V2.2.pdf](http://www.packml.com/PackML_Machine_State_Model_Guidelines_V2.2.pdf)



**S88 State Model**

State Model - Automatic Mode



**PackML State Model**

## **Appendix IID: PackTags – Tag Naming Guidelines v 2.0**

---

These guidelines were compiled by the Tag Naming Subteam (TST) within the PackML committee of the OMAC Packaging Workgroup and published on December 26, 2003. Please submit any comments to:

Larry Trunek, PackML tags subteam chair  
Miller Brewing Company  
3939 W. Highland Blvd  
Milwaukee, WI USA  
414/931-2953 ~ Fax: 414/931-6061  
Email: [ltrunek@mbco.com](mailto:ltrunek@mbco.com)

### **EXECUTIVE SUMMARY**

PackTags are named data elements used for open architecture, interoperable data exchange in packaging machinery. In PackTags version 1.0, an initial set of PackTags sufficient for computing machine performance metrics was defined. In PackTags version 2.0, that initial set was expanded and adapted to support more of the PLCs that are in common use in the packaging industry today.

### **INTRODUCTION**

PackML's mission is to develop open architecture naming convention guidelines for communications between production machinery within the packaging industry. Three guidelines have been released to date, and are available on the OMAC.org and PackML.org websites. The PackML Line Type Definitions define the different levels of machine integration in common use today. The PackML State Model provides a uniform set of machine states, so that all packaging machinery can be looked at in a common way. PackML Machine Modes Definition document, defined automatic, semi-automatic and manual operations. This work addresses the third major objective of PackML, tag names and definitions of data sets. These include the fundamental names of the data elements (or "tags") as well as the data types, values, ranges and where necessary data structures – PackTags.

Consistent meanings of the data elements will be defined. Examples from the PackML State Model are the state names, such as "Producing", "Stopping", and "Aborted". Other examples are "total run time", "total productive time", "total downtime", "wastage", "efficiency", etc. From the existing work, and new PackTags that are suggested, we will arrive at a com-

mon set of well-defined naming conventions that the OMAC Packaging Workgroup (OPW) can recommend as guidelines for wide adoption in the packaging industry.

### **Plug-and-Pack**

Tag name guidelines are useful for several reasons. Many of these fall under the broad set of goals that we call Plug-and-Pack™. This name is meant to suggest that it is to packaging machines what “plug and play” is to computing machines. Packaging users want their equipment to be easy to integrate, and Plug-and-Pack represents the ideal of machinery that is extraordinarily easy to integrate. Plug-and-Pack also implies multivendor interoperability.

There is much more to Plug-and-Pack than tag naming. Other OPW teams, such as Pack-Connect™ and PackSoft™ are working on some of the other aspects, such as network architecture, performance, and languages. PackTags “ride above” these aspects – they are on a different level. Our intention is to define PackTags such that they can be used with any underlying network and communication protocol. However, in their implementation, network protocols will need to be used.

Packaging users cite the following as key business drivers for achieving Plug-and-Pack™:

- Lowering the cost and reducing time for machine integration
- Allowing automatic startup and shutdown of packaging lines
- Increasing machine uptime via more rapid troubleshooting and facilitating root cause analysis of problems that cause downtime.
- Reducing operator and engineering training costs by providing a more uniform interface to machines for operators and engineers
- Lowering the cost of validation of production systems for Good Manufacturing Practices and other FDA regulations

### **Name Spaces**

PackTags address the “name space” problem – in order to communicate, different entities of any type need to have a common vocabulary. In the case of packaging machinery that is enabled with computers and networks, the name space problem is a matter of defining a specific, unambiguous, and easy to use set of names and data types – PackTags.

Another way to look at name spaces is via databases. If all the information in each machine is stored in a database, data in each machine’s database can be readily compared only if the field names are consistent – i.e., they use the same PackTags.

## **Domain of use**

As noted above, PackTags are useful for machine-to-machine (intermachine) communications; for example between a filler and a capper. They will also be useful for intramachine communications; for example between motion controllers and PLCs on a single machine. In addition, PackTags will be used to exchange information between machines and higher-level information systems like SCADA systems, plant databases, and enterprise information systems.

## **Performance Metrics**

Our subteam wanted to tackle a small area that would have immediate value to end-users, could be released for comment quickly and allowed us to work on our process. We chose the broad area of machine performance metrics such as machine efficiency, availability, etc. However, within this broad area, we chose to concentrate on information that represented the raw data necessary to calculate machine efficiency and other performance metrics without actually specifying or endorsing a particular calculation for these metrics.

The goal is to define a set of PackTags that would allow users to derive (or compute) most of the performance metrics that are in wide use in the industry. These are extremely useful, for

- equipment maintenance, including predictive maintenance and troubleshooting
- machine performance evaluations, in relation to business objectives such as productivity, profitability, quality, etc.
- performance comparisons between different machines
- deciding if and when to replace or update equipment
- new equipment specification and acceptance criteria
- work load scheduling and management
- asset management
- provide for more systematic logging of machine operations

There are a wide variety of different machinery performance metrics in use in industry. Often, companies and even different plants within companies will use different performance metrics, and there are cases where they are proprietary. Therefore, PackML could not provide for the computation of all the performance metrics users might need. For this reason, as stated above, the initial work focused not on calculating specific performance metrics, but rather on providing sufficient raw data to allow the calculation of virtually any performance metric in use. Later work may provide for tags which correspond to some of the well-

known standard performance metrics, such as those in DIN standard 8743: Concepts associated with packaging machines; definitions for time, output and their indication.

## **PACKTAGS**

### **Methods of Supplying Raw Data**

For basic machine availability (efficiency) computations, the time spent in each of the machine modes and states is sufficient. Two methods, elapsed time or event driven, can be used to supply the raw data for this. Some users prefer one of these, and some prefer the other, so this guideline supports both.

**Elapsed time:** The “elapsed time” method is to have the machine keep track of the elapsed time spent in all the machine modes and states. Upon request, these times can be supplied to another system, which can then compute performance metrics. For example, a basic measure of availability might be the time spent in the Producing state divided by the total elapsed time spent in all states. Advantages of this method are (1) the requisite information can be obtained by requests only (the requesting system does not have to be set up to receive asynchronous messages), and (2) less message traffic is needed than the “event” method. Disadvantages are that it requires a timekeeping function in the machine, and the stored elapsed times can be lost if the power to the machine is cut off. The cumulative time tags in Appendix “A” are sufficient for this method.

**Event:** The “event” method is to have the machine record each change of state and mode whenever they occur. The other system receives this message, notes the time, and keeps track of the time spent in each state. Its advantages and disadvantages are opposite to those given above. In addition, this method can record supplemental information with each state transition, such as detailed information on the reason for the state transition. Note that a primitive hard-wired version of this method without any network is possible. The State\_Transition tag in Appendix “A” is the only tag needed for this method.

### **Communication Format**

This guideline has been generated to support machines operating from all OMAC PackML line types. Communication methods of control platform manufacturers vary widely, but must be capable of communicating with other machines on the line as well as SCADA systems. Advanced data types such as structured data types, arrays, and text strings are optionally supported by systems that can support the structure.

Pack Tags are broken out into two types; Control and Information. Control data is defined as data required to interface between machines and line control for coordination. Information data is described as data collected by higher level systems for machine performance

analysis. Each grouping of data should be in a contiguous grouping of registers to optimize communications.

### **PackTag prefix**

PackTags will often be used in information systems that have a wealth of other named tags, so each PackTag should be prefixed with an identifying string to distinguish them from other tags that may have the same name. Previous work on S88 compatibility shows how necessary this is, and established a precedent of using “PML\_” as the prefix string. This makes it easy for a user who is unfamiliar with PackTags to look up their definitions.

### **PackTags name strings**

Many factory information systems do not allow for spaces in tag names, the guideline uses the common practice of substituting underline characters for spaces between words. The first letter of each word is capitalized for readability. While IEC 61131 is not case sensitive, to ensure inter-operability with all systems it is recommended that the mixed case format be adhered to. The total string shall not exceed 20 characters. Tag names are kept to a maximum length of 20 characters to allow us to use them in some of the older processors. This 20 characters limit includes the “PML\_” prefix. Thus, the exact text strings that should be used as tag names should be as follows:

PML\_Xxxxxx\_Yyyyyyyyy

For example:

PML\_Cur\_Mode,

PML\_Mode\_Time,

PML\_Cum\_Time\_Modes, etc.

### **Data Types, Units, and Ranges**

PackTags Appendix A shows the data type for each tag. They are as follows:

- Byte – 8 bit, or 0 to 255 in unsigned decimal format
- Integer – 32 bit, or 0 to 4294967295 in unsigned decimal format
- Real – 32-bit IEEE 754 standard floating point format (maximum value of 16,777,215 without introducing error in the integer portion of the number)
- Binary – Bit pattern within registers

- Time – seconds - Integer
- String – null-terminated ASCII
- Structure – a collection of data types. (This data type is typically reserved for higher level processors.)

Note that in future work, we may need to use a more formal method of defining tags data types, structures, units, ranges, etc. XML has often been suggested as a good method for this, but we have avoided it to date for simplicity’s sake and to avoid using any method which would be protocol or language specific.

Date/Time stamps are not supported within this guideline due to problems with synchronization between controllers for accurate SCADA data collection. Date/Time stamping will be provided by higher level systems, and not maintained within the control platform.

Where applicable, the units and ranges should be as indicated in the PackTags Appendix A table.

### Tag Details

The following is a detailed description of each tag.

#### Current Mode

**Tag Name:** PML\_Cur\_Mode

**Data Type:** Byte

**Tag Descriptor:** Current Mode

**Comments:** The response is one of a restricted set of values that define the current mode. Values 1-3 are identical to S88 modes, for compatibility reasons. The modes are:

0	Undefined
1	Automatic
2	Semi-Automatic
3	Manual
4	Idle

#### Mode Time

**Tag Name:** PML\_Mode\_Time

**Data Type:** Time

**Tag Descriptor:** Time in Current Mode

**Comments:** The response is the elapsed time the machine has spent in that mode since it entered that mode.

#### Cumulative Time In Each Mode

**Tag Name:** PML\_Cum\_Time\_Modes  
**Data Type:** Time  
**Tag Descriptor:** Cumulative Time In Each Mode

**Comments:** This item represents numerous tags, one for each mode. The response is the cumulative elapsed time the machine has spent in each mode since it's timers and counters were reset. For controllers that do support arrays, an array of times is supplied, the length of the array being the number of possible modes. For controllers that do not support arrays, individual tags will need to be configured. Examples of the tags would be:

PML\_Cum\_Time\_Auto  
PML\_Cum\_Time\_Man

### Current State

**Tag Name:** PML\_Cur\_State  
**Data Type:** Byte  
**Tag Descriptor:** Current State

**Comments:** The response is one of a restricted set of integers that define the current state. These are one of the following:

0	undefined
1	"Off"
2	"Stopped"
3	"Starting"
4	"Ready"
5	"Standby"
6	"Producing"
7	"Stopping"
8	"Aborting"
9	"Aborted"
10	"Holding"
11	"Held"

### State Time

**Tag Name:** PML\_State\_Time  
**Data Type:** Time  
**Tag Descriptor:** State Time

**Comments:** The response is the cumulative elapsed time the machine has spent in that state since it entered that state.

### Cumulative Time in Each State

**Tag Name:** PML\_Cum\_Time\_States  
**Data Type:** Time  
**Tag Descriptor:** Cumulative Time In Each State

**Comments:** This item consists of numerous tags, one for each state. The response is the cumulative elapsed time the machine has spent in each state since it's timers and counters were reset. For controllers that do support arrays, an array of times is supplied, the length of the array being the number of

possible modes. For controllers that do not support arrays, individual tags will need to be configured. Examples of the tags would be;

PML\_Cum\_Time\_Stop  
PML\_Cum\_Time\_Off

### Sequence Number

**Tag Name:** PML\_Seq\_Number

**Data Type:** Integer

**Tag Descriptor:** Sequence Number

**Comments:** Sequence Number is a free running number used to help in the verification of state transitions. It should increment by one for each successive state transition message.

### Reason Code

**Tag Name:** PML\_Reason\_Code

**Data Type:** Integer

**Tag Descriptor:** Reason Code

**Comments:** The PML\_Reason\_Code is an integer, which will correspond to a user/machine builder defined list of reasons for the state transition to have been made. This could include error codes that have been propagated from other sub systems. Once the state transition has been initiated, this code must be locked in until the next state transition.

A standardized reason code will be available in Appendix “B”.

### Reason Code Index

**Tag Name:** PML\_Reason\_Index

**Data Type:** Integer

**Tag Descriptor:** Reason Code Index

**Comments:** The PML\_Reason\_Index is an integer, which will correspond to a specific PML\_Reason\_Code, and represents multiple instances of the same error on a given machine. An example would be an open guard door on a machine that has multiple guard doors. The PML\_Reason\_Code would be a value of 13. The PML\_Reason\_Index for that fault would be an integer value of one through the number of doors on the machine, representing the door number that was open.

As with the PML\_Reason\_Code, once the state transition has been initiated, this code must be locked in until the next state transition.

### Reason Text (Optional)

**Tag Name:** PML\_Reason\_Text

**Data Type:** String

**Tag Descriptor:** Reason Text

**Comments:** The Reason Text is an optional text string that states the reason for the state transition. For example, a transition from the Producing state to the Standby state may have as a reason “Materials Runout” or “Downstream Queue Full”. Also, the Reason might optionally indicate the initiator of the transition, as for example an “Operator Pushbutton” or “Supervisory System Command”.

In the case of a transition to the Abort state or any other fault or problem, the Reason should contain sufficient information to construct OEE, a Loss Tree, and to perform Root Cause Analysis<sup>1</sup>.

There must be a direct correlation between PML\_Reason\_Code and PML\_Reason\_Text.

### Supplemental Reason Code

**Tag Name:** PML\_Reason\_Code\_Supp

**Data Type:** Binary

**Tag Descriptor:** Supplemental Reason Code

**Comments:** This code supplies additional information while in a state. Unlike the PML\_Reason\_Code, which identifies, and locks in, the reason for the transition, the supplementary reason code provides additional information while in the state. An example may be a machine stops for a downstream backup. While the machine is stopped, the operator opens a guard door. The downstream backup is removed. The supplemental reason code should identify the guard door, as well as any other reason for the machine not being allowed to transition states. Every bit should have a corresponding reason in Appendix “B”. These reasons shall be different than the PML\_Mat\_Ready tag. An index such as PML\_Reason\_Index is not provided.

### Current Machine Speed

**Tag Name:** PML\_Cur\_Mach\_Spd

**Data Type:** Real

**Units:** Primary packages/minute

**Tag Descriptor:** Current Machine Speed

**Comments:** The response is the current speed of the machine in primary packages per minute. Keeping units in primary packages, allows for easier control integration. The following example is for a bottle line running at balance line speed doing a change over on the fly with flexible packaging.

Machine	Using Pack Counts	Using Primary packages
Bulk Depalletizer	41.6666 (24 pack equiv)	1,000
Filler	1,000	1,000
Labeler	1,000	1,000
Packer	66.666 (15 packs)	1,000

### Current Speed Selected

**Tag Name:** PML\_Cur\_Spd\_Sel

**Data Type:** Integer

**Tag Descriptor:** Current Speed Selected

**Comments:** The response is the current selected discrete speed of the machine. This allows monitoring of machine speed set points and frequency of speed changes. Additional speeds may be added between 9 and 99.

0	Undefined
1	Jog
2	Prime

<sup>1</sup> OEE (Overall Equipment Efficiency) Definitions, PackML web site, slides 6 and 26, [http://www.packml.org/OEE\\_Definitions.ppt](http://www.packml.org/OEE_Definitions.ppt)

3	Pre-Lube
4	Maintenance
5	Slow
6	Medium
7	High
8	Surge
9	Tracking
99	Analog control only

### Machine Design Speed

**Tag Name:** PML\_Mach\_Design\_Spd

**Data Type:** Real

**Units:** Primary Packages/minute

**Tag Descriptor:** Machine Design Speed

**Comments:** The response is the maximum design speed of the machine in primary packages per minute for the package configuration being run. This speed is NOT the maximum speed as specified by the manufacturer, but rather the speed the machine is designed to run in it's installed environment. Note that in practice the maximum speed of the machine as used for efficiency calculations will be a function of how it is set up and what products it is producing.

### Number of Products Processed

**Tag Name:** PML\_Prod\_Processed

**Data Type:** Integer

**Units:** Primary Packages

**Tag Descriptor:** Number Products Processed

**Comments:** The response is the cumulative number of primary packages processed since the machine's counters and timers have been reset. This number represents machine through put, which includes good, defective and re-workable product.

### Number of Defective Products

**Tag Name:** PML\_Defect\_Prod

**Data Type:** Integer

**Units:** Primary packages

**Tag Descriptor:** Number Defective Products

**Comments:** The response is the cumulative number of defective (non re-workable) primary packages processed since the machine's counters and timers have been reset.

### Number of Re-workable Products

**Tag Name:** PML\_Rework\_Prod

**Data Type:** Integer

**Units:** Primary packages

**Tag Descriptor:** Number Re-workable Products

**Comments:** The response is the cumulative number of re-workable primary packages processed since the machine's counters and timers have been reset. It is assumed that these products will be reprocessed through the machine.

## Machine Cycle Count

**Tag Name:** PML\_Mach\_Cycle

**Data Type:** Integer

**Units:** none

**Tag Descriptor:** Machine Cycle Count

**Comments:** Indicates the number of complete cycles of the machine with or without product being processed. This is derived from some sensor and then transmitted across the communication network.

## Materials Ready

**Tag Name:** PML\_Mat\_Ready

**Data Type:** Binary

**Units:** none

**Tag Descriptor:** Materials Ready

**Comments:** Indicates materials are ready for processing. Comprised of a series of bits within a register(s) with 1 equaling ready, 0 equaling not ready. Each bit represents a different material. Materials are defined as all consumables such as product, cartons, labels, utilities, and glue.

## Materials Low

**Tag Name:** PML\_Mat\_Low

**Data Type:** Binary

**Units:** none

**Tag Descriptor:** Materials Low

**Comments:** Indicates materials are running low. Comprised of a series of bits within a register(s) with 1 equaling material at full supply, 0 equaling material is running low. Each bit represents different material. Materials are defined as all consumables such as product, cartons, labels, utilities, and glue. This tag would indicate when one of the “*Material Ready*” supplies was becoming low. De-activation of one of the Binary components would not shut the machine down, but simply be an alarm to the operator. Continued operation of the machine in a “*Material Low*” may eventually result in the machine continuing to consume material and the corresponding bit in the “*Material Ready*” tag to go low, which would shut down the machine. Certain “*Material Low*” bits may cause the machine to change speed awaiting an operator in an attempt to preserve constant motion. If one of the materials doesn’t have a low level warning, the bit for that material within the “*Material Low*” would be set high manually and left. There should never be a condition where a bit within “*Materials Low*” is high (1) while the corresponding bit within “*Material Ready*” is low (0). For example:

<b>MACHINE READY TO RUN OR RUNNING, BUT RAW MATERIAL AND CO2 SUPPLY IS RUNNING LOW</b>	Raw Material	Container	CO2	Compressed Air	Lubrication Water	Container Closures	Undefined / Unused	Undefined / Unused	Undefined / Unused	Undefined / Unused
Materials Ready	1	1	1	1	1	1	1	1	1	1
Materials Low	0	1	0	1	1	1	1	1	1	1

### Transition Trigger Report

**Tag Name:** PML\_Trans\_Trigger

**Data Type:** Structure

**Units:** none

**Tag Descriptor:** Transition Trigger Report

**Comments:** Transmission of each trigger. This will be time and date stamp entry to a new state. The format is <Date and Time>, <Event Name>.

External triggers have significant importance to the other systems, yet may not immediately cause a state transition.

The <Event Name> is one of a restricted set of strings that define the current state. These are one of the following:

“Materials\_Ready\_True”  
 “Materials\_Ready\_False”  
 and others yet to be defined.

### Package Ratio

**Tag Name:** PML\_Prod\_Ratio

**Data Type:** Integer

**Units:** Primary packages / Package

**Tag Descriptor:** Count of primary packages per secondary package

**Comments:** Contains the quantity of primary packages per current package being produced.

### Package Reset

**Tag Name:** PML\_Reset

**Data Type:** Boolean – Input into control device

**Tag Descriptor:** Reset timers and counters

**Comments:** Resets all timers and counters to zero. Refer to section 4.

## Control Command

**Tag Name:** PML\_Cntrl\_Cmd

**Data Type:** Integer

**Tag Descriptor:** Control Command

**Comments:** Instruction to drive State change.

0	Undefined
1	Prepare
2	Start
3	Stop
4	Hold
5	Abort
6	Home
7	E-Stop

## COMMANDS

In addition to tags, there is at least one command that appears to be necessary, a command to reset timers and counters associated with the performance tags. (Depending on the method of communication, commands could be more difficult to implement, so their incorporation in the guideline should be considered carefully.) Refer to section 3.7.23.

## MACHINE IDENTIFICATION

The above tags provide a standardized method for communicating machine status and performance. It does not provide a standardized naming convention for associating a specific tag to a specific machine. Due to the diversity of available control architecture naming conventions, combined with the variability of how the end user sub-divides their facilities, along with user preference, no standard architecture could be recommended. It is highly recommend that each end user identify “their” standard for machine extensions for PackTags and then communicate those extensions to the technology provider or vendor. The machine location extensions may be prefixes or suffixes depending upon what the end users platform supports, and user preference. Possible options may be:

Plant.line.machine.PML\_tag

PML\_tag.Plant.line.machine

PML\_tag.plant.area.line.machine

## **CONCLUSIONS**

We have defined the second release of tags for PackML. This set is adequate to begin implementations that will provide real value to users and machine builders, by providing an open and interoperable way for packaging machinery to communicate with other machinery on packaging lines, with supervisory systems, and with enterprise information systems.

## **FUTURE WORK / ISSUES**

Key areas for future work are:

- Publication of a set of performance metrics expressed in terms of PackML tags. Some examples are machine efficiency, production efficiency, availability etc.
- Review of the methods for the retrieval of tag information across different networks
- Internationalization of terminology
- Definition of recipes for packaging

## **REFERENCES**

PackML Integration Brief V2.0

PackML Line Types Document

PackML State Definition Document

PackML State Model S88 Software Compatibility document, PackML, 3/22/2002

OEE (Overall Equipment Efficiency) Definitions, PackML web site, slides 6 and 26, [http://www.packml.org/OEE\\_Definitions.ppt](http://www.packml.org/OEE_Definitions.ppt)

Document

## PackTags Appendix A

### CONTROL

Spec #	Prefix	Tag Name <sup>2</sup>	Tag Descriptor	Data Type	Units
3.7.1	PML_	Cur_Mode	Current Mode	Byte	
3.7.4	PML_	Cur_State	Current State	Byte	
3.7.12	PML_	Cur_Mach_Spd	Current Machine Speed	Real	Primary packages/Min
3.7.19	PML_	Mat_Ready	Materials Ready	Binary	
3.7.20	PML_	Mat_Low	Materials Low	Binary	
3.7.21	PML_	Trans_Trigger	Transition Trigger	Structure	
3.7.23	PML_	Reset	Package Reset	Boolean	
3.7.24	PML_	Cntrl_Cmd	Control Command	Integer	

### INFORMATION

Spec #	Prefix	Tag Name	Tag Descriptor	Data Type	Units
3.7.2	PML_	Mode_Time	Time in Current Mode	Time	Seconds
3.7.3	PML_	Cum_Time_Modes	Cumulative Time In All Modes	Time	Seconds
3.7.5	PML_	State_Time	State Time	Time	Seconds
3.7.6	PML_	Cum_Time_States	Cumulative Time In All States	Time	Seconds
3.7.7	PML_	Seq_Number	Sequence Number	Integer	
3.7.8	PML_	Reason_Code	Reason Code	Integer	
3.7.9	PML_	Reason_Index	Reason Index	Integer	
3.7.10	PML_	Reason_Text	Reason Text	String	
3.7.11	PML_	Reason_Code_Supp	Supplemental Reason Code	Binary	
3.7.13	PML_	Cur_Spd_Sel	Current Speed Selected	Integer	
3.7.14	PML_	Mach_Design_Spd	Machine Design Speed	Real	Primary packages/Min
3.7.15	PML_	Prod_Processed	Number Products Processed	Integer	Primary packages
3.7.16	PML_	Defect_Prod	Number Defective Products	Integer	Primary packages
3.7.17	PML_	Rework_Prod	Number Re-workable Products	Integer	Primary packages
3.7.18	PML_	Mach_Cycle	Machine Cycle	Integer	

<sup>2</sup> Tag names are kept here to a maximum length of 20 characters to allow us to use them in some of the older processors. This 20 character limit includes the "PML\_" prefix.

---

3.7.22	PML	Prod_Ratio	Product Ratio	Integer	Primary packages/Pack
--------	-----	------------	---------------	---------	-----------------------

## PackTags Appendix B - Reason Codes

### Grouping

Reason #	Reason Text
1 – 32	Machine Internal Reason - Safeties - OMAC Defined
33 – 64	Machine Internal Reason – Operator Actions - OMAC Defined
65 – 256	Machine Internal Reason – Internal Machine faults – Product related - OMAC Defined
257 - 512	Machine Internal Reason – Internal Machine faults – Machine related - OMAC Defined
513 – 999	Machine Internal Reason – General Information - OMAC Defined
1000 – 1999	Machine Internal Reason – Vendor Defined
2000 - 2499	Machine Upstream Process Reason – OMAC Defined
2500 - 2999	Machine Upstream Process Reason – Vendor Defined
3000 – 3499	Machine Downstream Process Reason – OMAC Defined
3500 - 3999	Machine Downstream Process Reason – Vendor Defined
4000 – 4499	Out Of Service – OMAC Defined
4500 - 4999	Out Of Service – Vendor Defined

### Detail (From above grouping)

Reason #	Reason Text
0	Undefined
1	E-Stop pushed
2	Perimeter Protection Fault
3	Mains turned off
4	Safety Gate/Guard Door Open
5 - 32	Reserved for future OMAC defined safety codes
33	Cycle Stop Button Pushed
34	Start Button Pushed
35	Reset Button Pushed
36	Jog Mode Selected
37	Automatic Mode Selected
38	Manual Mode Selected
39	Semi-Automatic Mode Selected
40 - 64	Reserved for future OMAC defined operator action codes
65	Material Jam
66 - 256	Reserved for future OMAC defined internal material related codes
257	Machine Jam
258	Electrical Overload
259	Mechanical Overload
260	Drive Fault
261	Drive Failure
262	Servo Axis Fault
263	Servo Axis Failure

264	Communication Error
265	PLC Error Code
266	Vacuum
267	Air Pressure
268	Voltage
269	Temperature
270	Hydraulic Pressure
271	Hydraulic Level
272	Hydraulic Temperature
273 - 512	Reserved for future OMAC defined internal machine related codes
513	Counter Preset Reached
514	Product Selected
515	Local Slow Speed Requested
516	Local Medium Speed Requested
517	Local High Speed Requested
518	Local Surge Speed Requested
519	Remote Speed Requested
520	Drive Warning
521	Servo Warning
522 - 998	Reserved for future OMAC defined general information related codes
999	Catch All - Unidentified internal reason
<b>1000 - 1999</b>	<b>Vendor defined area for machine internal items</b>
2000	Infeed Not turned On
2001	Infeed Overload
2002	Low Prime Material
2003	High Prime Material
2004 - 2498	Reserved for future OMAC defined upstream related codes
2499	Catch All - Unidentified upstream reason
<b>2500 - 2999</b>	<b>Vendor defined area for upstream items</b>
3000	Discharge Not Turned On
3001	Discharge Overload
3002	Discharge Blocked reason
3003	Discharge Cycle Stop reason
3004	Discharge Immediate Stop reason
3005 – 3498	Reserved for future OMAC defined downstream related codes
3499	Catch All - Unidentified downstream reason
<b>3500 - 3999</b>	<b>Vendor defined area for downstream items</b>
4000	Line Not Scheduled
4001	Planned Maintenance

4002	Meals and Rest
4003	Meetings
4004	Training
4005	No Materials
4006	Remote Stop Requested
4007	Machine Not Selected
4008	Changeover
4009	Lubrication
4010	Product count preset reached
4011	Setup Selected
4012 – 4499	Reserved for future OMAC defined Out of Service related codes
4500 - 4999	Vendor defined area for Out of Service items

## **Appendix III: PackAL Packaging Application Function Block Library v1.01**

---

These guidelines were compiled by the PackSoft committee of the OMAC Packaging Workgroup and published on March 29, 2005. Please submit any comments to:

Gerd Hoppe, PackSoft committee chair  
Beckhoff Automation GmbH  
Eiserstraße 5  
33415 Verl  
Germany  
+49 5246 963-0; Fax: +49 5246 963-9130

Email: [g.hoppe@beckhoff.com](mailto:g.hoppe@beckhoff.com)

### **EXECUTIVE SUMMARY**

OMAC derives common solutions collectively for both technical and non-technical issues in the development, implementation, and commercialization of open architecture control technologies to maximize the business value of packaging machinery by developing guidelines that lead to the most appropriate application of advanced automation technology.

The PackSoft Subcommittee recommends that Technology Providers, OEM's and End Users adopt the IEC 61131-3 Standard for Programmable Controllers: Programming Languages for packaging machine automation, including motion control. The IEC 61131-3 standard applies to a wide range of programmable controllers and is commercially available in products from a number of Technology Providers. Use of the standard encourages software design that is hierarchical in nature, re-usable through program organization units; and the standard provides facilities for real-time exchange of information via communication networks.

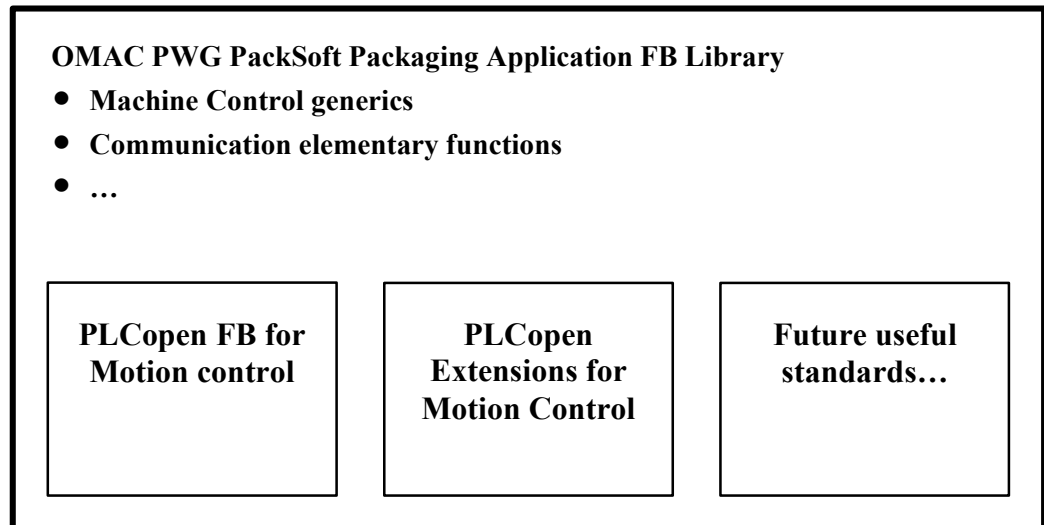
The PackSoft Subcommittee recommends the use of PLCopen Function Blocks for Motion Control V1.0, the Extensions (Part 2) to this standard, and other coming PLCopen publications as they might be a fit, all based on

IEC 61131-3.

The PackSoft Subcommittee develops Packaging Industry related Control Software guidelines that allow to commonly implement as technology, program machinery equipment and controls, maintain, train and learn the use of software on packaging industry devices.

## INTRODUCTION

The objectives of the OMAC PWG PackSoft committee are to define software modules (based on appropriate standards, e.g. IEC61131-3) which describe basic packaging machinery control elements; develop a programming convention and a set of functional software elements which lend themselves to be become common use throughout the Packaging Industry to simplify the structure, understanding and handling of generic machine elements and their representation in software; define a set of functions which will serve the majority of packaging user’s applications and needs; and promote the adoption of useful existing and emerging standards to own definitions.



**Figure 1: Scope of definitions, embracing existing and emerging standards**

## GENERAL OBJECTIVES

- Machine related as first priority, process related as second
- Providing an easy-to-use interface to the Packaging Functionality

- Related to existing packaging standards
- Re-usable parts, usable in a wide range of applications.
- Application program should be implementable on any platforms
- Accepted/User-tested Functionality/Concepts providing the basis for FBs
- Providing a common basis, terminology, references
- Providing a 'style guide' for additional / future FBs
- Providing user guidelines / examples
- Standard Function Blocks Library for standard Packaging related Functionality
- Combining these FB's to an application program needs an environment that is suitable for Packaging related applications. Requirements and restrictions for such an environment are partly dealt with in this standard.

### **Language context goals**

- Focus on definition of Function block interfaces and behavior, and Data Types according to the IEC 61131-3.
- These FBs and data types can be used in all IEC61131-3 languages.
- The examples in this draft are just given informatively in textual (IL, ST) and graphical (LD, FBD, SFC) IEC61131-3 languages.
- The contents of the function blocks can be implemented in any programming language (e.g. IEC61131-3 ST, C) or even in firmware or hardware. Therefore the content should not be expected to be portable.
- Reusable applications composed from these Function Blocks and data types can be achieved by PLCopen Conformity Level and Reusability Level if IEC61131-3 languages and future PLCopen certification and exchange standards.
- This specification shall be seen as an open framework without hardware dependencies. It provides openness in the implementation on different platforms such as fully integrated, centralized or distributed systems. The actual

implementation of the Function Blocks themselves is out of the scope of this Packaging FB Library.

## PACKAGING APPLICATION LIBRARY

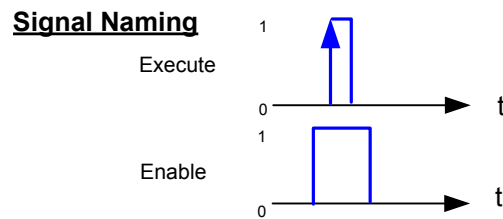
### Technical Units

The only specification is made on the length unit (noted as [u]) that is to be coherent with its derivatives i.e. (velocity [u/s]; acceleration [u/s<sup>2</sup>]; jerk [u/s<sup>3</sup>]). Nevertheless, the unit [u] is not specified (manufacturer dependent). Only its relations with others is specified.

### Naming Conventions

All naming conventions shall adhere to common rules developed within OMAC PWG. All standardization conventions, e.g. for definition of technical values, general coding rules, etc. shall adhere to the conventions developed by PLCopen Motion Task Force as described in the Function Blocks for Motion Control V1.0 standard.

The name “Enable” refers to a level-sensitive input signal, whereas the name “Execute” refers to an edge-trigger input signal.



**Figure 2: Signal Naming for edge- and level triggered input signals**

### Axis\_REF Data type

The Axis\_REF is a structure that contains information on the corresponding axis. It is used as a VAR\_IN\_OUT in all Function Blocks defined in this document. The content of this structure is implementation dependent and can ultimately be empty. If there are elements in this structure, the supplier

shall support the access to it but this is outside the scope of this document. The refresh rate of this structure is also implementation dependent.

Axis\_REF data type declaration:

```

TYPE AXIS_REF : STRUCT

- (Content is implementation dependent)

END_STRUCT
    
```

Example:

```

TYPE AXIS_REF : STRUCT

- AxisNo: UINT;

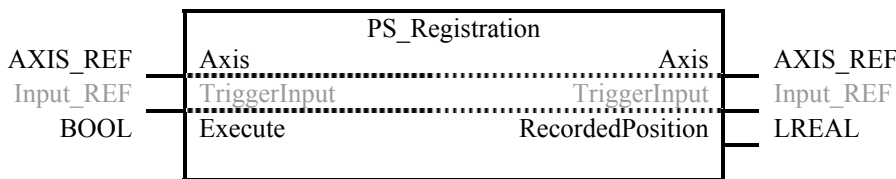
- AxisName: STRING(255);

END_STRUCT
    
```

### Documentation Conventions

Variables and types of the implementation interface are listed in tables and a FB graphical representation. (The code implementation and the execution of the FB may be implemented in any IEC61131-3 language, see above). Optional (extended) Inputs and Outputs are represented in grey font. VAR\_IN\_OUT are represented dashed underlined inside the body of the function block as represented in this document.

Example:



### Compliance and Portability

The objective of this work is to achieve a level of portability of Function Blocks acting on a base level device, and providing the same functionality to the user as described within this paper, with respect to user interface, input / output variables, parameters and units used.

The possibility of combining several MC libraries from different suppliers within one application is left open to be solved by the systems integrator.

An implementation which claims compliance with this specification shall offer a set of (meaning one or more) Function Blocks with at least the basic interface variables marked as “B” in the defined tables in the definition of the Function blocks in this document. For higher level systems and future extensions any subset of the extended interface variables marked as “E” in the tables can be implemented. Supplier specific additions should be marked “S”

Three sets of interface variables are defined:

1. Basic Set                      Shown in the tables with the letter “B”
2. Extended Set                 Shown in the tables with the letter “E”
3. Supplier Set                 Shown in the tables with the letter “S”

The basic set is focused to the minimal requirements to create a functional control system. The extended version defines the options for higher level systems, and future extensions. Compliance and usage of the OMAC PWG logos is described in Appendix A.

## **Certification**

In order to fulfill the requirements set, different levels of certification are applicable:

1. Certification / Declaration of Conformity of the software tool supplier, often part of the control supplier
2. Certification / Conformity of the application at the user and/or machine builder

Ad 1: Certification /Conformity Declaration of the software tool supplier, often part of the control supplier

The development environment, including the Packaging related function blocks, need to be certified by the supplier according to rules set forth in IEC61131-3. These requirements are beyond the scope of this document. Typically, a supplier will self-certify the level of compliance of his product with the technical paper provided. A certification matrix is provided in the appendix to this document.

Ad 2: Certification / Conformity of the application at the user and/or machine builder

Within an application, a certification includes the Packaging related software combined with the infrastructure, like sensors, switches and actuators, connection schemes, etc. Certification or Conformity Declarations for these applications are beyond the scope of this document, and have to be dealt with by external involved entities.

The use of the OMAC or Packaging Workgroup logos does not give any guarantee about any compliance or fulfillment. The use of the logo just refers to the inclusion of the ideas and guidelines as described in this document, within the relevant software environment, and the availability of this information in more detail on the relevant section of OMAC website [www.OMAC.org](http://www.OMAC.org).

### Overview of the defined Packaging Application Function Blocks

Packaging Process Functions	Machine Communication
Wind / Unwind Axis (constant surface velocity, csv mode)	PS_Send,
Wind / Unwind Axis (constant torque, ct mode)	PS_Receive
Dancer Control	PS_CommRequest
Registration	PS_Indicate
Registration_Correction	PS_CommRespond
Indexing	PS_CommConfirm
Batch Counter	PS_CommNotify
Digital PLS (Digital CAM Switch)	PS_CommIndicate
SetOverride	PS_Publish
JogAxis	PS_Subscribe
Flying Sync	
Gear_In with Dynamic Gear Factor	
Motion Command Stop with oriented halt	
<b>Machine Behavior Organization</b>	
PackML Machine State Model FB	

**Table 1: Defined Function Blocks of PackAL**

## Packaging Application Function Blocks

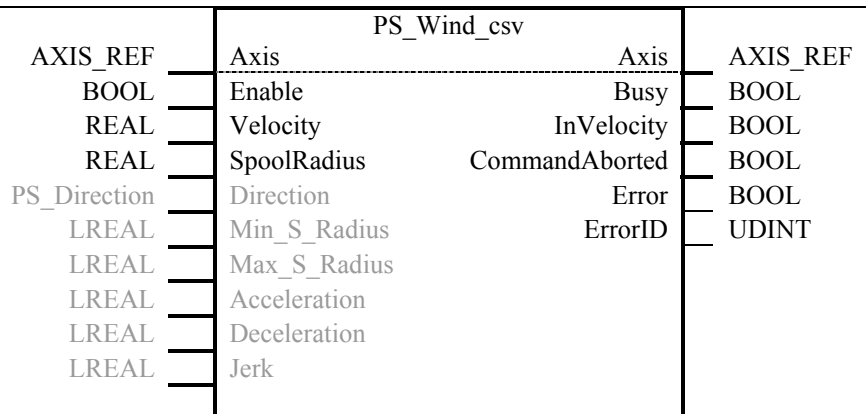
### Wind / Unwind Axis (constant surface velocity, csv mode)

FB-Name		PS_Wind_csv	
This function block commands a controlled motion at a specified circumflex velocity for a wind / unwind axis. The circumflex is calculated from the radius of a wind/unwind spool, measured with a sensor.			
VAR IN OUT			
B	Axis	AXIS_REF	
VAR INPUT			
B	Enable	BOOL	Start the motion at high level, stop at low level
B	Velocity	LREAL	Value of the maximum Surface velocity [u/s]
B	SpoolRadius	LREAL	Actual value of the radius of the spool [u]
B	Min_S_Radius	LREAL	Value of the Minimal Spool Radius, initial value [u]
B	Max_S_Radius	LREAL	Value of the Maximal Spool Radius, initial value [u]
E	Direction	PS_Direction	Enum type (pos, neg)
E	Acceleration	LREAL	Value of the acceleration (increasing energy of the motor) [u/s <sup>2</sup> ]
E	Deceleration	LREAL	Value of the deceleration (decreasing energy of the motor) [u/s <sup>2</sup> ]
E	Jerk	LREAL	Value of the Jerk [u/s <sup>3</sup> ]
VAR OUTPUT			
B	Busy	BOOL	Executing status
B	InVelocity	BOOL	command circumflex velocity phase active
B	CommandAborted	BOOL	True once command is aborted
B	Error	BOOL	Signals that error has occurred within Function block
B	ErrorID	UDINT	Error Number

Note : This FB contains the mathematical calculation for coupling between a rotary slave axis and a translatory master axis. Its purpose is to generate and re-adjust a constant surface velocity (peripheral speed), relative to the master axis, for the rotary calibrated and controlled slave axis, depending on its spool diameter. Via a signal proportional to the spool radius, the spool radius of this slave axis is automatically evaluated and used for the calculation. This radius must never have the value 0.0 mm, since otherwise a calculation is no longer possible.

- The FB decelerates the axis to stop while winding above the Max\_S\_Radius
- The FB decelerates the axis to stop while unwinding below the Min\_S\_Radius
- SpoolRadius needs to satisfy Min\_S\_Radius < SpoolRadius < Max\_S\_Radius to execute the action, otherwise Error = True, ErrorID set
- Error ID is implementation specific

SpoolRadius is scanned in every cycle of the Function Block execution, other inputs in first cycle only.



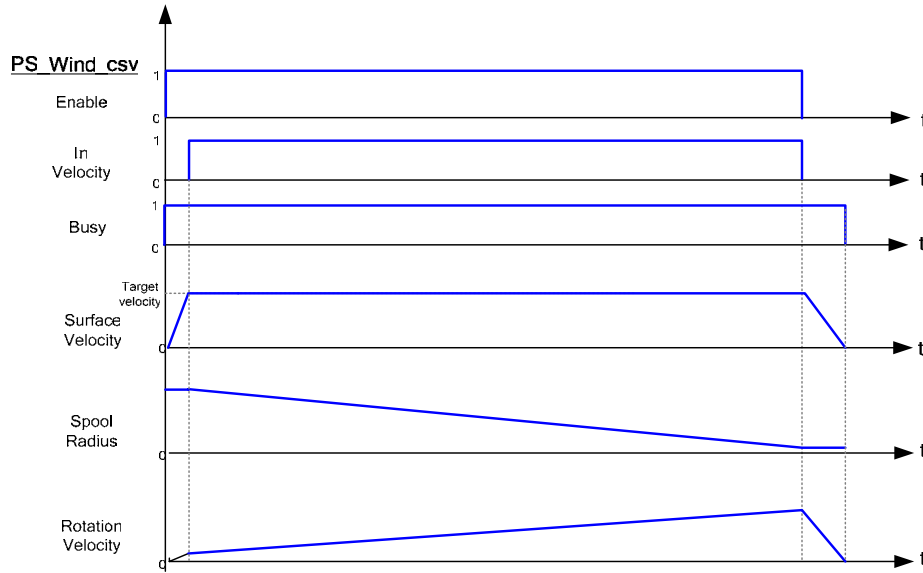


Figure 3: PS\_wind\_csv timing diagram

### Wind / Unwind Axis (constant torque, ct mode)

FB-Name		PS Wind ct		
This function block commands a simpler controlled motion at a specified torque for a wind / unwind axis. The circumflex is calculated from the radius of a wind/unwind spool, measured with a sensor.				
VAR IN OUT				
B	Axis	AXIS_REF	Identifies the axis to work upon	
VAR INPUT				
B	Enable	BOOL	Start the motion at high level, stop at low level	
B	Torque	LREAL	Value of the torque (Torque or force in t.u.)	
B	SpoolRadius	LREAL	Actual value of the radius of the spool [u]	
B	Min_S_Radius	LREAL	Value of the Minimal Spool Radius, initial value [u]	
B	Max_S_Radius	LREAL	Value of the Maximal Spool Radius, initial value [u]	
E	TorqueRamp	LREAL	The maximum time derivative of the torque (Torque or force in t.u. per sec)	
E	Direction	PS_Direction	Enum type (pos, neg)	
VAR OUTPUT				
B	InTorque	BOOL	Set value of torque reached	
B	Busy	BOOL	Executing status	
B	CommandAborted	BOOL	True once command is aborted	
B	Error	BOOL	Signals that an error has occurred within Function Block	
E	ErrorID	UDINT	Error identification	

Notes: : This FB contains the mathematical calculation for coupling between a rotary slave axis and a translatory master axis. Its purpose is to generate and re-adjust a constant surface velocity (peripheral speed), relative to the master axis, for the rotary calibrated and controlled slave axis, depending on its spool diameter. Via a signal proportional to the spool radius, the spool radius of this slave axis is automatically evaluated and used for the calculation. This radius must never have the value 0.0 mm, since otherwise a calculation is no longer possible. In this mode, the FB will act in a simpler way with constant torque based solutions to the wind / unwind problem, e.g. for a FI based implementation.

- Max\_S\_Radius and Min:S\_Radius are assumed to be initial values for first calculation of gear between master and slave axis at start of FB.
- The FB decelerates the axis to stop while winding above the Max\_S\_Radius
- The FB decelerates the axis to stop while unwinding below the Min\_S\_Radius
- SpoolRadius needs to satisfy  $Min\_S\_Radius < SpoolRadius < Max\_S\_Radius$  to execute the action, otherwise Error = True, ErrorID set
- Error ID is implementation specific

SpoolRadius is scanned in every cycle of the Function Block execution, other inputs in first cycle only.

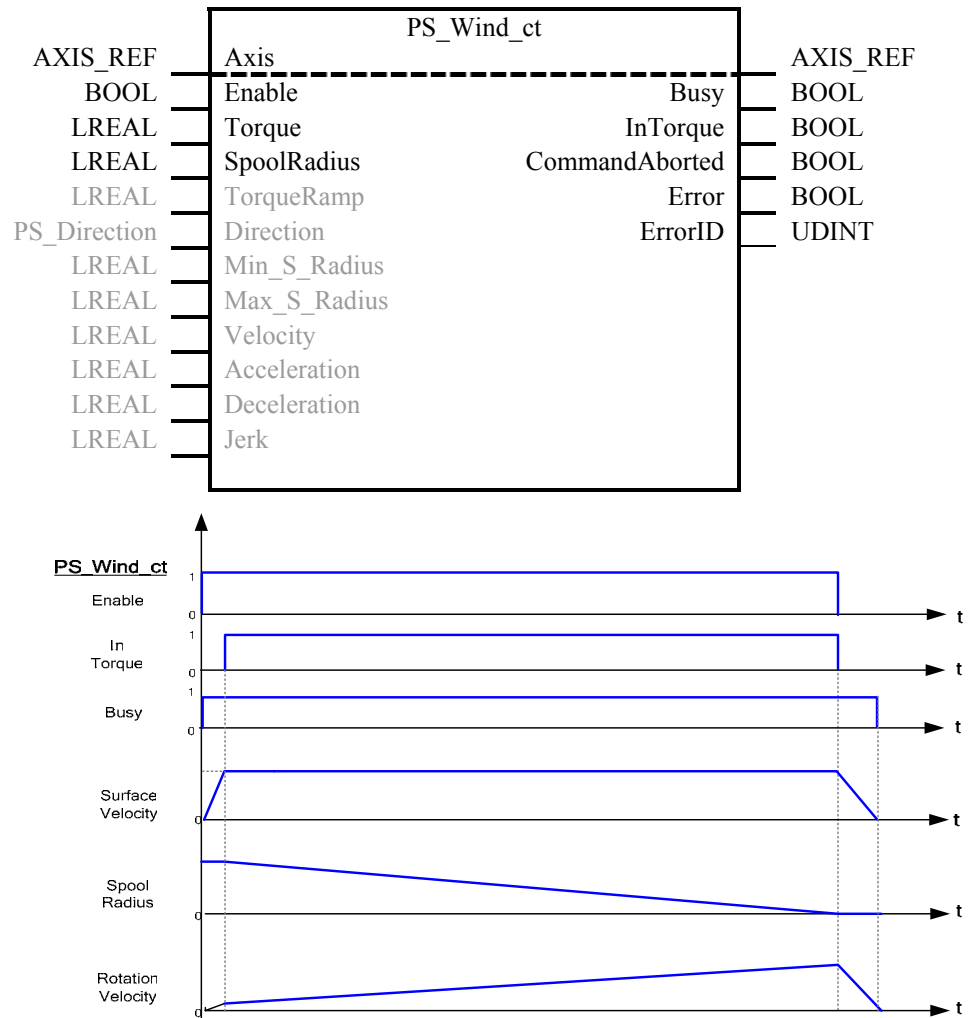
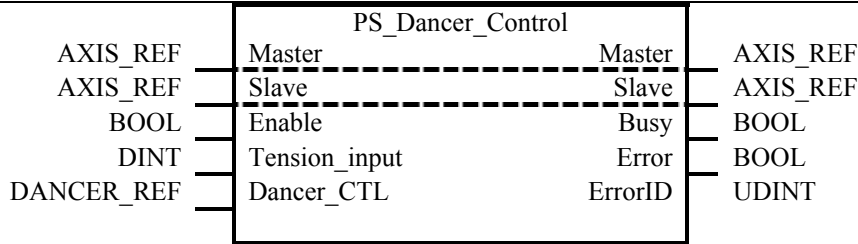


Figure 4: PS\_Wind\_ct timing diagram

**Dancer Control**

FB-Name		<b>PS Dancer Control</b>	
This function block commands a controlled motion of an axis as slave of a dancer-coupled master axis. The Master axis is a physical or virtual axis. This FB provides the coupling between a slave axis (typically the infeed to the dancer) and a master axis (the outfeed of the dancer) via a dancer-PID controlled variable gearing factor.			
VAR IN OUT			
B	Master	AXIS_REF	
B	Slave	AXIS_REF	
VAR INPUT			
B	Enable	BOOL	Start at high level, stop at low level
B	Tension_input	REAL	Input signal for tension of the Web, usually represented by the relative actual dancer position sensor
B	Dancer_CTL	DANCER_REF	Structure with Dancer Controller Parameters
VAR OUTPUT			
B	Busy	BOOL	Executing status
B	Error	BOOL	Signals that an error has occurred within Function Block
E	ErrorID	UDINT	Error identification
<p>Notes : The FB’s purpose is to generate and re-adjust a constant surface velocity (peripheral speed), relative to the master axis, for the rotary calibrated and controlled slave axis, depending on a dancer position. Via a position signal (dancer position signal), the tension between master (web) and slave (e.g. spool) is represented. For general use, the raw dancer position is often aligned to a “balance position” by an offset (and optional multiplier) in a first dancer scaling algorithm. The PID calculates the control value depending on the difference to a scaled command dancer position. This PID control value output then tunes the gear ratio between the master (web) and the slave (e.g. spool). A multiplier limits the distortion of the gear, offset adjusts to gear setpoint. Scaling algorithm, dancer balance position, PID factors, gear factor, delta and offset to the gear are application specific.</p> <p>Other possible implementations for Dancer Control, especially those with simpler two-switch control compared to PID control, may be represented by additional Function Blocks defined in the future.</p> <p>Tension_input is scanned in every cycle of the Function Block execution, other inputs in first cycle only.</p>			



Elements within the array structure of Structure **Dancer\_CTL**:  
**DANCER\_REF**:

B/E	Parameter	Type	Description
B	Tension_ctl	DINT	Target value for web tension, typically the target position for the dancer in balanced condition
B	GearRatio	REAL	Ratio of gear factor g(t) between master (web) and Slave (spool) to balance the dancer. Default is 1.0.
B	deltaGear	REAL	Delta scaling multiplier of PID output (-1.0 ... +1.0) to GearRatio – must be smaller than 1 but never 0. deltagear would be e.g. 0.8 or 0.9 to limit the gear distortion
B	Gearoffset	REAL	scaling offset to fit GearRatio distortion by satisfying equation $g(t) = \text{deltaGear} * \text{PIDout} + \text{Gearoffset}$

B	fKp	REAL	PID Control Proportional gain (P)
B	fTn	REAL	PID Control Integral gain Tn (I) [s]
B	fTv	REAL	PID Control Derivative gain Tv (D-T1) [s]
B	fTd	REAL	PID Control Derivative damping time Td (D-T1) [s]
B	Accel_limit	REAL	corresponds indirectly, i.e. relative to a maximum master velocity, to a maximum permitted acceleration ( $pa = aSlaveMax / v MasterMax$ ). The limit parameter pa corresponds to the reciprocal value of the run-up time $tH = 1 / pa$

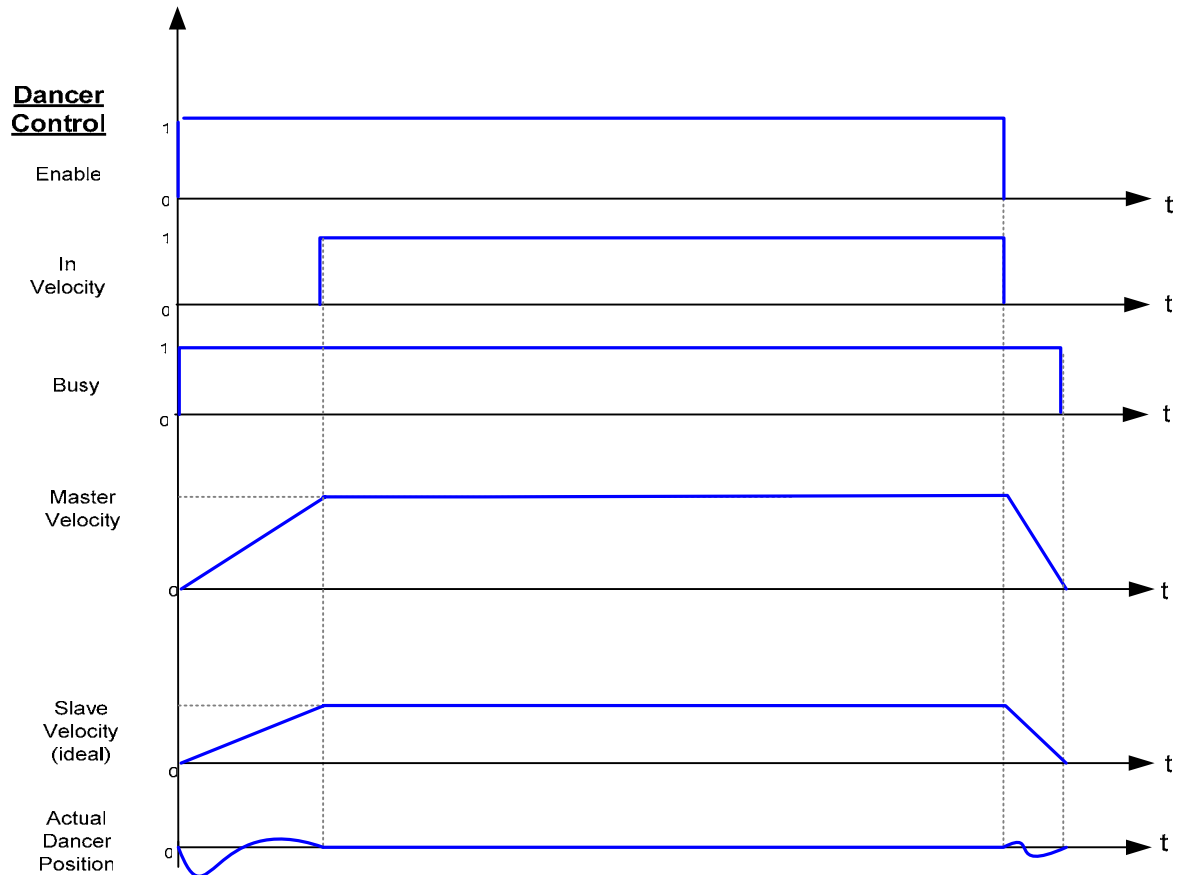


Figure 5: PS\_Dancer Control timing diagram

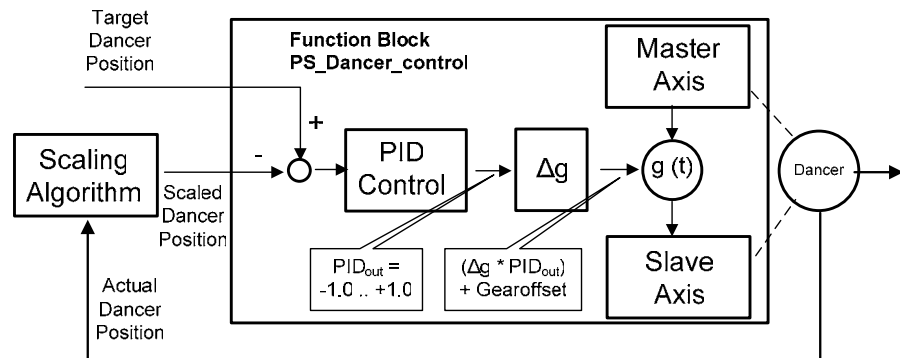
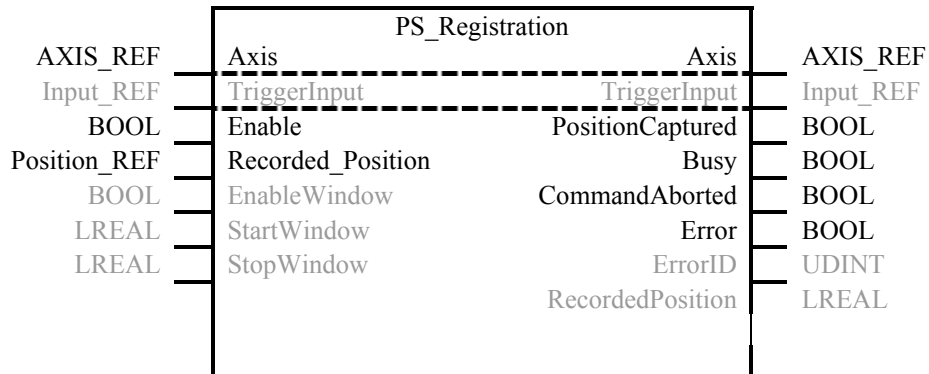


Figure 6: PS\_Dancer Control Structure

### Registration

FB-Name		<b>PS Registration</b>	
This function block captures an axis position on receiving a rising edge execute command. The captured position is often referred to as “Latch Position”.			
VAR IN OUT			
B	Axis	AXIS_REF	Identifies the axis which position shall be latched at the trigger event
E	TriggerInput	Input_REF	Reference to the trigger signal source. Trigger signal may be specified by the axis reference
VAR INPUT			
B	Enable	BOOL	Start the Position Latch at high level
B	Recorded_Position	Position_REF	Actual Position of the Axis defined by Axis.
E	EnableWindow	BOOL	Enable an “capture input” window, in which input trigger events are captured
E	StartWindow	LREAL	Start position of capture window zone in technical units [u]
E	StopWindow	LREAL	Stop position of capture window zone in technical units [u]
VAR OUTPUT			
B	PositionCaptured	BOOL	Trigger event recorded
B	Busy	BOOL	FB is active waiting for trigger event, capture pending on EnableWindow
E	CommandAborted	BOOL	Command aborted
B	Error	BOOL	Signals that an error has occurred within Function Block
E	ErrorID	UDINT	Error identification
B	RecordedPosition	LREAL	Latch Position, where the trigger event occurred (in t.u. [u])
<p>Notes : This FB is a subset of the Registration_correction FB. Registration is often referred to as “Probing”. Some Motion Devices have dedicated “Registration” or “Probe” Inputs to quickly latch the position on the fly.</p> <p>Enable activates the Function Block, Recorded_Position is scanned in every cycle of the Function Block execution, other inputs in first cycle only.</p>			



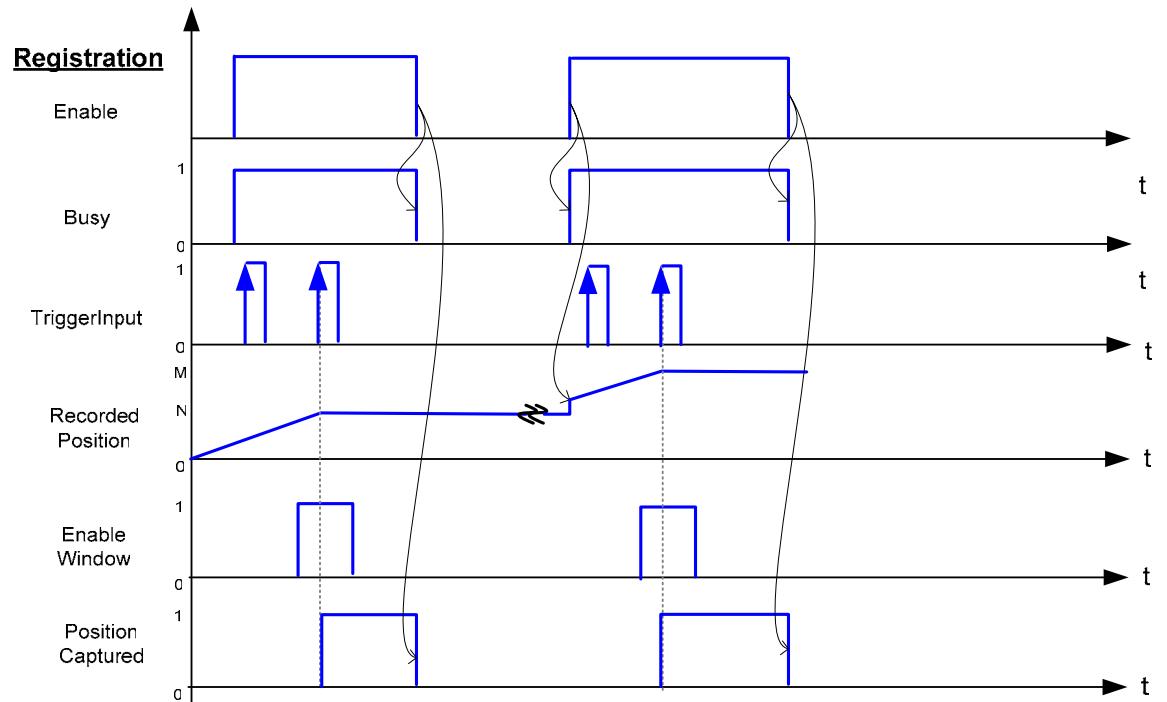


Figure 7: PS\_Registration timing diagram

**Registration\_Correction**

FB-Name		<b>PS_Reg_Correction</b>		
This function block corrects a distance offset, calculated after capturing an axis position, e.g. with the “Registration” FB, on receiving a rising edge execute command. The captured position is often referred to as “Latch Position”. The difference to the ideal position is the input “delta correction distance” to this FB.				
VAR_IN_OUT				
B	Master	AXIS_REF		
B	Slave	AXIS_REF		
VAR_INPUT				
B	Enable	BOOL	Start the Correction FB, level triggered	
E	Mode	BYTE	Correction method selector input. Two methods defined, other subject to supplier implementation	
B	EnableCorr	BOOL	Level sensitive, TRUE during correction motion	
B	DeltaCorr	LREAL	Correction distance (relative position) for correction move	
B	CompZone	LREAL	Distance range of master in which correction move is to complete	
E	DelayDistance	LREAL	Distance of master to move before correction starts	
E	AccComp	LREAL	Max. acceleration for compensation [u/s <sup>2</sup> ]	
E	DecComp	LREAL	Max. deceleration for compensation [u/s <sup>2</sup> ]	
E	VdeltaComp	LREAL	Max. velocity for compensation move [u/s]	
E	Vprocess	LREAL	Actual process (master) velocity [u/s]	
E	TimeOut	TIME	Timeout supervision of the compensation move	

VAR_OUTPUT			
B	Done	BOOL	This output changes from FALSE to TRUE once the correction motion is correctly done. It resets with the rising edge of EnableCorr or with Enable = FALSE.
B	Busy	BOOL	This output remains TRUE as long as Function Block executes
B	Command-Aborted	BOOL	True once command is aborted
B	Error	BOOL	Signals that an error has occurred within Function Block
E	ErrorID	UDINT	Error identification

Notes : This correction FB executes after e.g. a registration FB captures apposition and the delta to a desired position has been determined by this or other methods. The mode selector allows the user to choose different ways of correction. Partial implementation of the solutions is possible; in this case, selection of the other methods results in an error.

Mode: 0 (Offset correction)

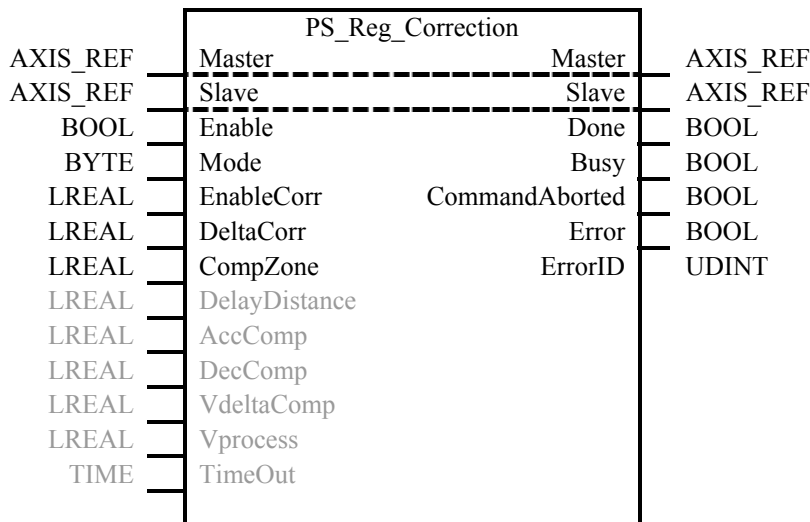
In this case, the DeltaCorr distance is directly applied to the Master Axis by means of a relative offset of the zero position, as used for CAM axes.

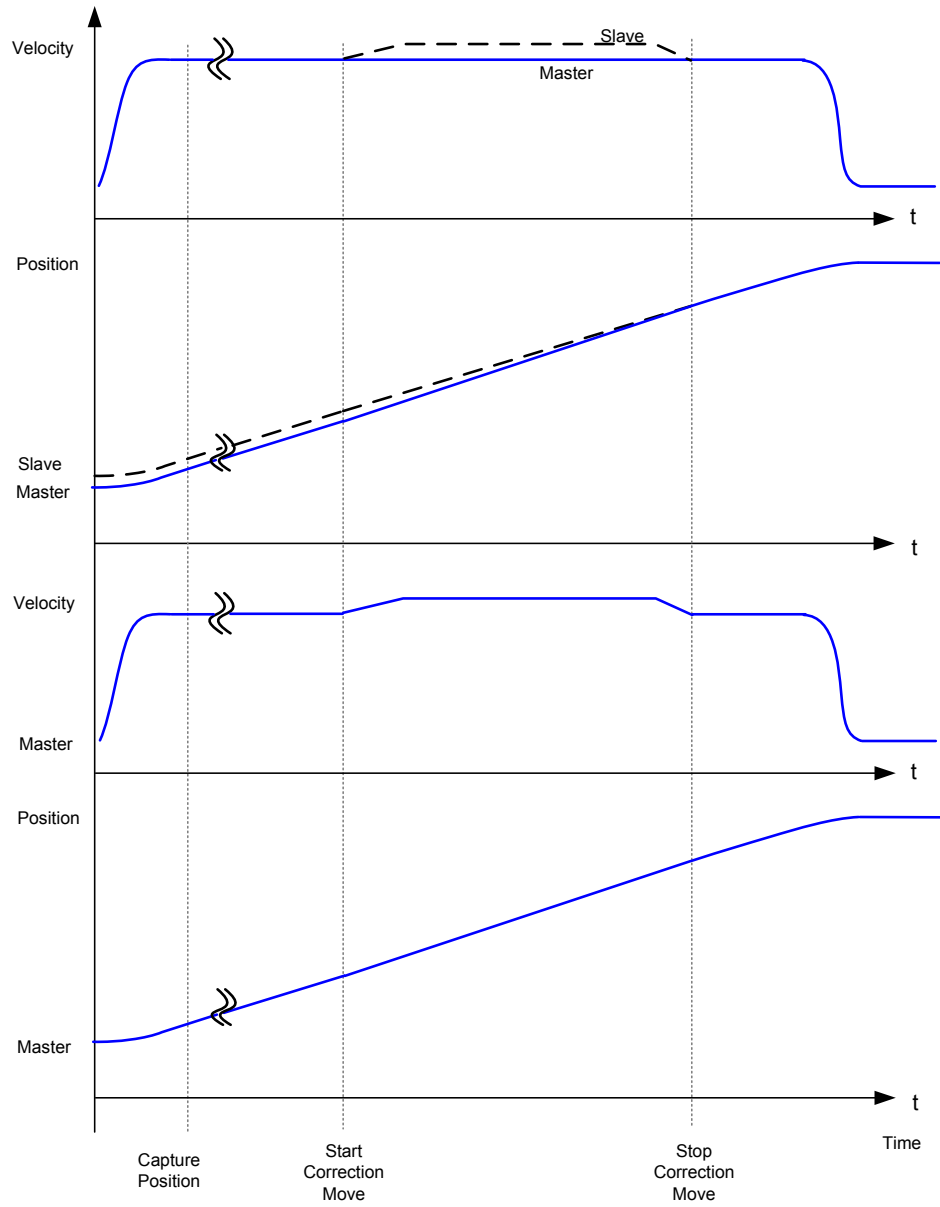
Mode: 1 (Distance correction)

The DeltaCorr distance is used to calculate a single super-positioned relative move of the slave axis to correct the delta correction distance.

Other correction methods may apply, e.g. by dynamic gear factor, and could be implemented as supplier specific enhancements.

Vprocess, the actual master of process velocity, is scanned in every cycle of the execution of the Function Block. Other inputs are scanned once at start of FB.



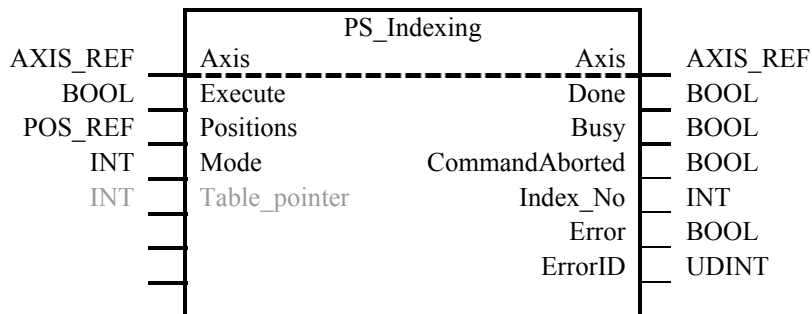


**Figure 8: PS\_Reg\_Correction timing diagram**

Top: Distance correction mode  
 Bottom: Offset correction mode  
 blue: Master, black: Slave

## Indexing

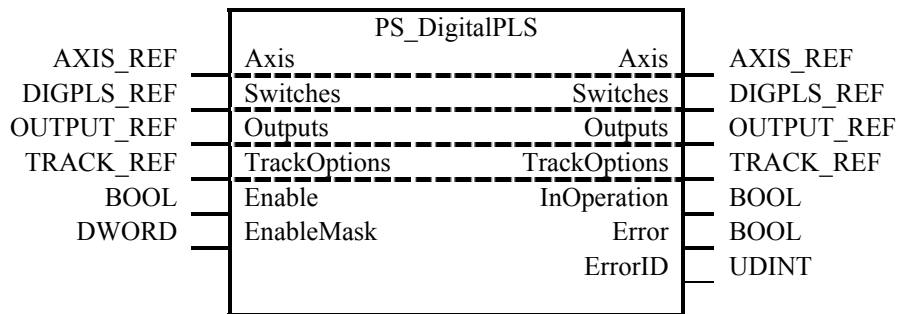
FB-Name		<b>PS_Indexing</b>	
This function block will – upon R_Trigger on Execute – do a number of relative moves, listed in a table (Struct or Array). The FB will position the axis to a complete stop at target position and continue with the next move from the table upon next R_Trigger signal.			
VAR IN OUT			
B	Axis	AXIS_REF	
VAR INPUT			
B	Execute	BOOL	Start the next relative positioning by a rising edge.
B	Positions	POS_REF	Reference to Structure or Array with relative move distances listed. Typical Type of Positions: LREAL
B	Mode	INT	Mode of Indexing, see notes
E	Table_pointer	INT	Pointer to element n of Table
VAR OUTPUT			
B	Done	BOOL	True once Indexing is done
B	Busy	BOOL	This output remains TRUE until the block has executed a command
B	CommandAborted	BOOL	True once command is aborted
B	Index_No	INT	Index executing or last index completed
B	Error	BOOL	Signals that an error has occurred within Function Block
E	ErrorID	UDINT	Error identification
<p>Notes : This function block will – upon R_Trigger on Execute – do a number of relative moves 1 - N, listed in a table (Struct or Array). The FB will position the axis to a complete stop at target position n and continue with the next move from the table upon next Execute signal.</p> <p>Modes of operation are</p> <p>Mode: 0            pass through table elements once until finding zero distance entry,  Mode: 1            cycle through table, starting at first element on finding a zero distance entry,  Mode: 2            position relative for distance of table element N (Table_pointer)</p> <p>The FB will complete a singular move if possible. In case a second Execute rising edge triggers the next move, the FB will not decelerate to stop but continue to new target position immediately.</p> <p>Inputs are updated on rising edge of Execute.</p>			





### Digital PLS (Phase Limit Switch or CAM Switch)

FB-Name		<b>PS_DigitalPLS</b>	
This function block is the analogy to switches on a motor shaft: it commands a group of discrete output bits to switch in analogy to a set of mechanical cam controlled switches connected to an axis. Forward and backward movements are allowed. Note that it is intended to keep this FB compatible with PLCopen MC FB Extensions (Part 2) MC_CAMSWITCH function Block.			
VAR_IN_OUT			
B	Axis	AXIS_REF	Reference to the axis to which the switches are connected to
B	Switches	DIGPLS_REF	Reference to the switching actions.
B	Outputs	OUTPUT_REF	Reference to the signal outputs, directly related to the referenced tracks. (max. 32 per function block) (First output = first TrackNumber)
E	TrackOptions	TRACK_REF	Reference to structure containing track related properties, e.g. the ON and OFF compensations per output/track.
VAR_INPUT			
B	Enable	BOOL	Enables the Switches outputs
E	EnableMask	DWORD	32 bits of BOOL. Enables the different tracks. Least significant data is related to the lowest TrackNumber. With data SET (to '1' resp. TRUE) the related TrackNumber is enabled.
VAR_OUTPUT			
B	InOperation	BOOL	The commanded tracks are enabled
B	Error	BOOL	Signals that error has occurred within Function block
E	ErrorID	UDINT	Error Identification
Notes:			
<ul style="list-style-type: none"> <li>• DIGPLS_REF is a supplier specific reference to the pattern data.</li> <li>• OUTPUT_REF is a supplier specific structure linked to the (physical) outputs</li> <li>• TRACK_REF is supplier specific structure containing the track properties, e.g. the compensation per track (A track is a set of switches related to one output).</li> </ul>			



Basic elements within the structure of DIGPLS\_REF

B/E	Parameter	Type	Description
B	TrackNumber	INT	TrackNumber is the reference to the track
B	FirstOnPosition [u]	REAL	Lower boundary where the switch is ON
B	LastOnPosition [u]	REAL	Upper boundary where the switch is ON
E	AxisDirection	INT	Both (=0; Default); Positive (1); Negative (2)
E	CamSwitchMode	INT	Position based (=0; Default); Time based (=1)
E	Duration	TIME	Coupled to time based CamSwitchMode

**Table 3: Digital PLS Structure table**

Basic elements within the array structure of TRACK\_REF

B/E	Parameter	Type	Description
E	OnCompensation	TIME	Compensation time with which the switching on is advanced or delayed in time per track.
E	OffCompensation	TIME	Time compensation the switching off is delayed per track.
E	Hysteresis [u]	REAL	Switching is shifted to a later reached position in order to avoid multiple switching around the switching point.

**Table 4: Digital PLS Track table**

This definition of cams has a start and an end position. So the user can define each single cam, that has an **FirstOnPosition** and an **LastOnPosition** (or time). In addition to a mechanical cam it has the possibilities to set it for a certain time and to give it a time compensation and a hysteresis. If (FirstOnPosition > LastOnPosition) it gives an inverse cam switch, which is off only within these positions.

**CamSwitchMode** can be Position, Time or other additional supplier specific types.

**Duration:** Time, the output of a time cam is ON

The time compensation (**OnCompensation** and **OffCompensation**) can be positive or negative. Negative means the output changes before the switching position is reached.

**Hysteresis:** This parameter avoids that the output is switching very often, if the axis is near the switching point and the actual position is jittering around the switching position. Hysteresis is part of TRACK\_REF, which means that for each track a different hysteresis is possible.

Example of CAMSWITCHREF

Parameter	Type	Switch01	Switch02	Switch03	Switch04	...	SwitchN
TrackNumber	INTEGER	<i>1</i>	<i>1</i>	<i>1</i>	<i>2</i>		
FirstOnPosition [u]	REAL	<i>2000</i>	<i>2500</i>	<i>4000</i>	<i>3000</i>		
LastOnPosition [u]	REAL	<i>3000</i>	<i>3000</i>	<i>1000</i>	<i>--</i>		
AxisDirection	INTEGER	<i>1=Pos</i>	<i>2=Neg</i>	<i>0=Both</i>	<i>0=Both</i>		
CamSwitchMode	INTEGER	<i>0=Position</i>	<i>0=Position</i>	<i>0=Position</i>	<i>1=Time</i>		
Duration	TIME	<i>--</i>	<i>--</i>	<i>--</i>	<i>1350</i>		

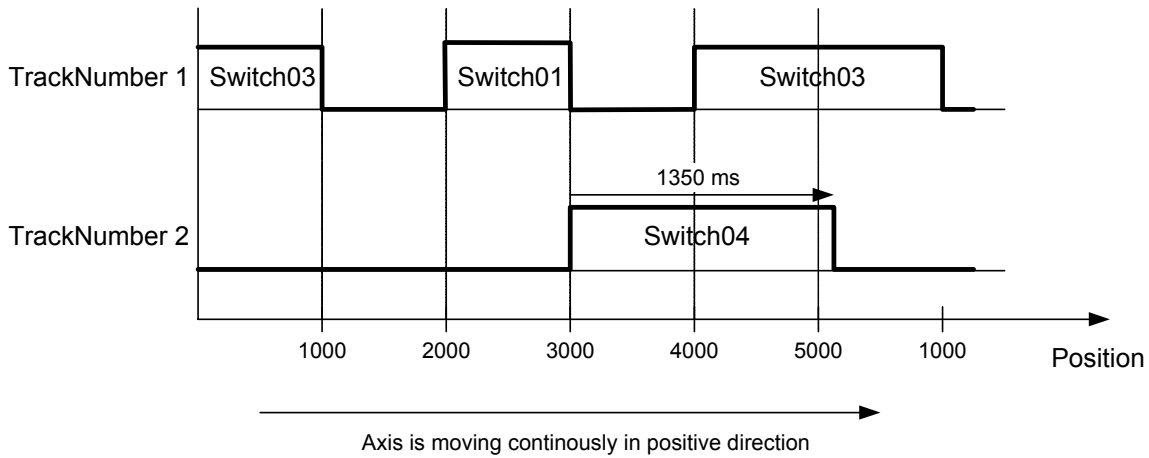
Note: Values are Examples

**Table 5: Digital PLS Switch Position table**

Example.

This example uses the values from the example for DIGPLS\_REF above. It uses no On/OffCompensation, nor hysteresis.

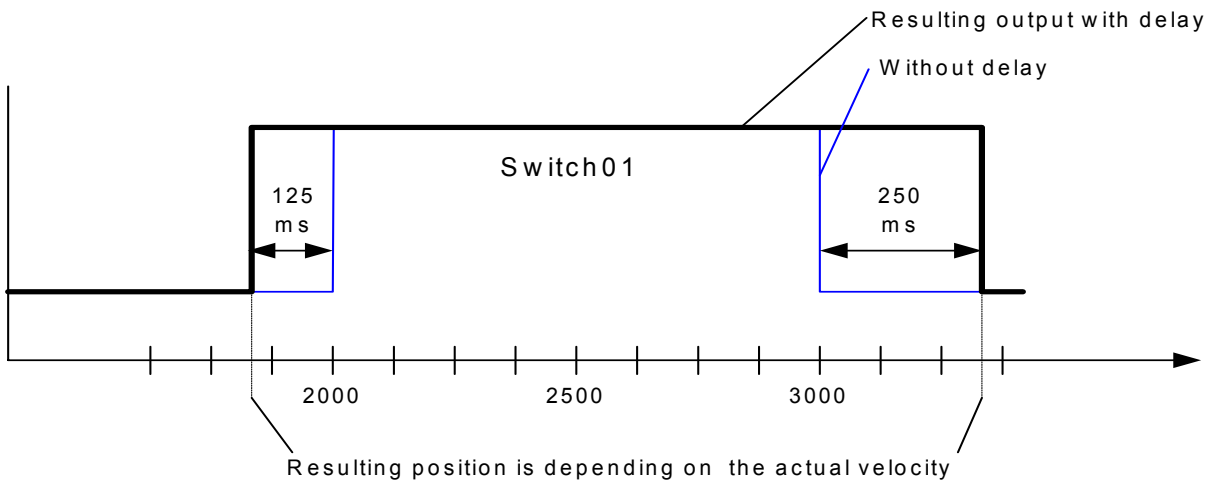
This is the behavior of the outputs, when the axis is moving continuously in positive direction. The axis is a modulo axis with a modulo length of 5000 u.



**Figure 10: Digital PLS timing diagram**

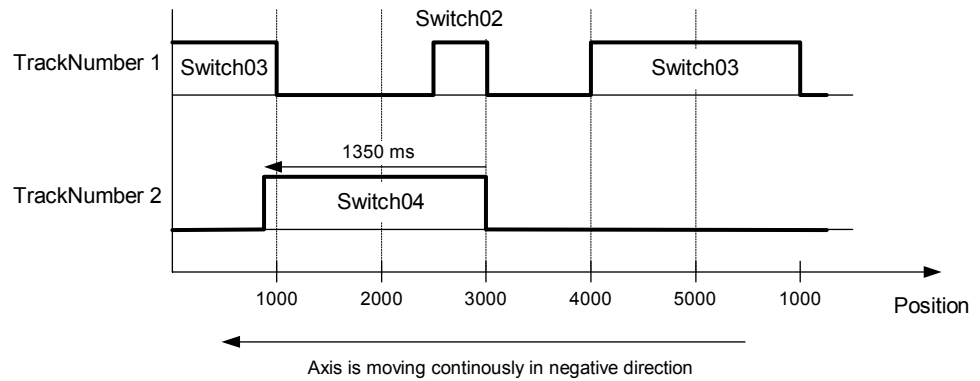
Detailed description of Switch01.

This example additionally uses OnCompensation -125ms and OffCompensation +250ms.



**Figure 11: Digital PLS, detailed description of Switch01.**

This is the behavior of the outputs, when the axis is moving continuously in negative direction without On- or OffCompensation and without Hysteresis.



**Figure 12: Digital PLS duration switch timing diagram**

**Override**

FB-Name		<b>PS_SetOverride</b>		
This function block sets the values of override. The override parameters act as a factor that is multiplied to the commanded velocity, acceleration, deceleration and jerk of the move function block. Note that it is intended to keep this FB compatible with PLCopen MC FB Extensions (Part 2) MC_SetOverride function Block.				
VAR IN OUT				
B	Axis	AXIS_REF	Identifies the axis to work upon	
VAR INPUT				
B	Enable	BOOL	Write the value of the override factor at rising edge	
B	VelFactor	LREAL	New override factor for the velocity (e.g. 0...100 %)	
E	AccFactor	LREAL	New override factor for the acceleration/deceleration	
E	JerkFactor	LREAL	New override factor for the jerk	
E	Timeout	TIME	Timeout	
VAR OUTPUT				
B	Done	BOOL	Signals that the override factor(s) is (are) set successfully	
B	Error	BOOL	Signals that error has occurred within Function block	
E	ErrorID	UDINT	Error identification	

Notes:

1. The Input AccFactor acts on positive and negative acceleration (deceleration).
2. This FB sets the factor. The override factor is valid until a new override is set.
3. The default values of the override factor are 1.0.
4. The value of the overrides can be between 0.0 and 1.0. The behavior of values > 1.0 is supplier specific. Values < 0.0 are not allowed. The value 0.0 is not allowed for AccFactor and JerkFactor.
5. The value 0.0 set to the VelFactor stops the axis without bringing it to the state standstill.
6. Override does not act on slave axes. (Axes in the state synchronized motion).
7. The FB does not influence the state diagram of the axis.
8. Override can be changed at any time and acts directly on the ongoing motion.
9. **Override overrides all other motion commands active in regards to velocity applied**

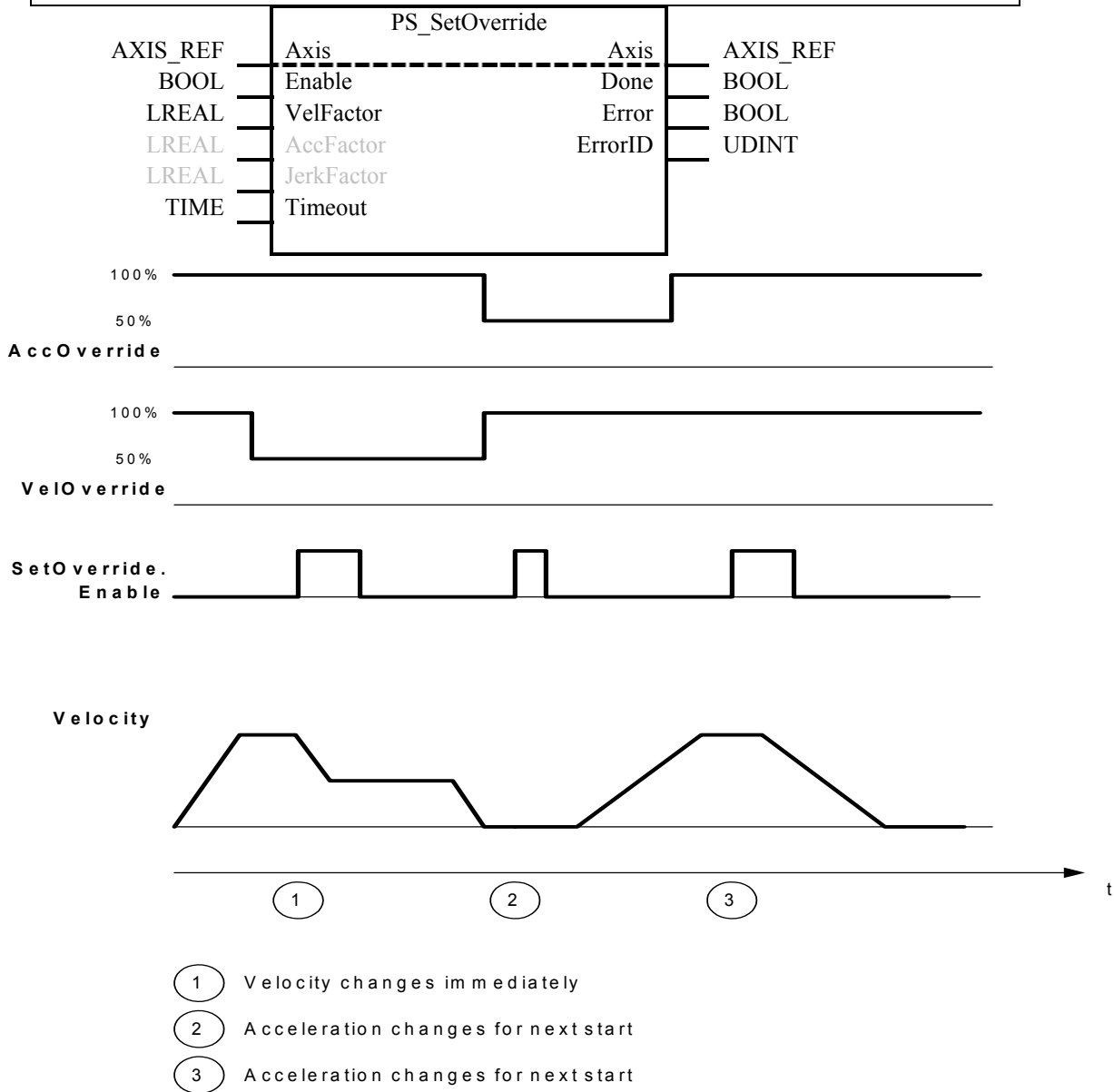


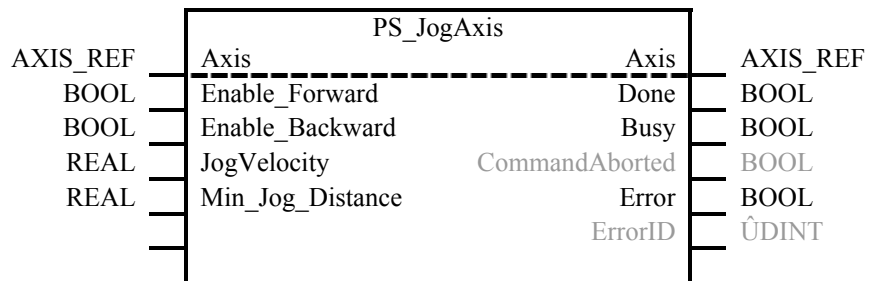
Figure 13: PS\_SetOverride timing diagram

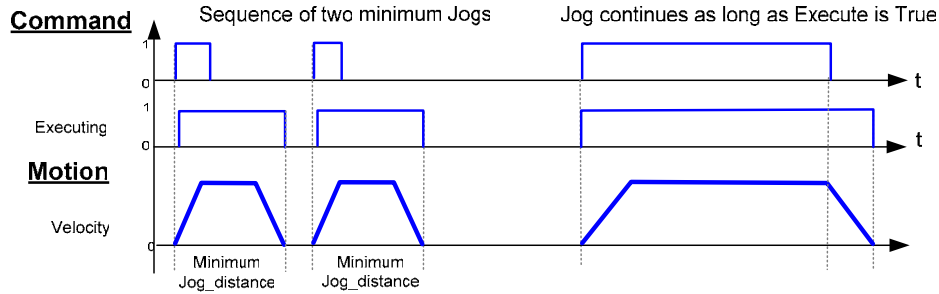
### JogAxis

FB-Name		<b>PS JogAxis</b>	
<ul style="list-style-type: none"> <li>This function block jogs an axis for N increments (represented in t.u.) forward or backward with the selected (manual operation) jog velocity and acceleration. N must satisfy certain minimum count limitation, otherwise no movement is executed.</li> </ul>			
VAR IN OUT			
B	Axis	AXIS_REF	Reference to axis to be jogged
VAR INPUT			
B	Enable_Forward	BOOL	Executes a sequence of a relative move Forward during high and a stop at signal = low
B	Enable_Backward	BOOL	Executes a sequence of a relative move Backward during high and a stop at signal = low
B	JogVelocity	LREAL	Velocity to jog (in t.u. [u])
B	Min_Jog_Distance	LREAL	Minimum distance to jog (in t.u. [u])
VAR OUTPUT			
E	Done	BOOL	True if jogging is done
B	Busy	BOOL	True if jogging is being performed
E	CommandAborted	BOOL	True if Jog command is aborted
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new jog command while still executing
E	ErrorID	UDINT	Error identification

Notes : This function block will – after rising edge on Enable\_Forward or Enable\_Backward - start a continuous move (at least for the minimum distance) and continue upon high-level on these inputs with a continuous motion – until they are FALSE – then, on their falling edge, the axis is regularly decelerated to stop. The movement is carried out on Jog velocity for the minimum distance or longer.

- In case both Enable signals are high, the FB will assume the result to be low as in an EXOR conjunction.



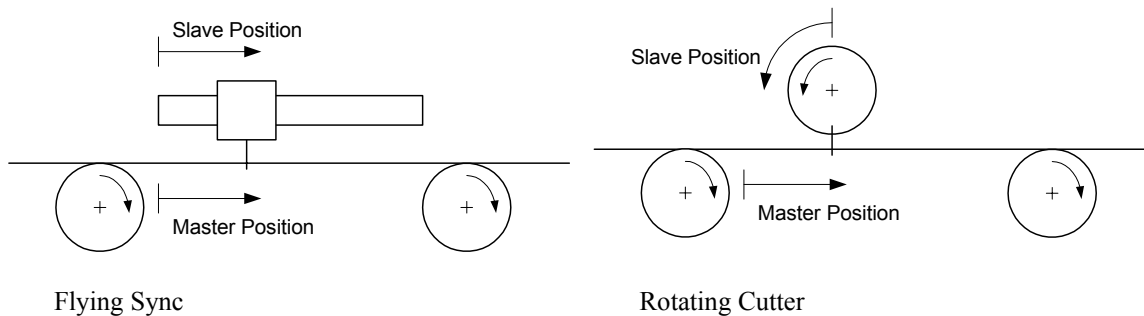


**Figure 14: PS\_Jog Axis timing diagram**

**Flying Sync**

The Flying Sync function provides a slave axis that can be synchronized to a moving master axis comparable to a “flying shears” or alternatively “flying cutter” functionality. The slave axis moves in synchronism with the master axis to perform machining processes. This kind of movement, synchronized to the master axis, allows to machine a work piece even while it is being transported.

The "Flying Sync" function is able to start synchronization of the slave even when the slave has already started, and is therefore no longer stationary. The "Flying Sync" function also calculates improved set value profiles, and these can be influenced by the user through a wide range of boundary conditions.



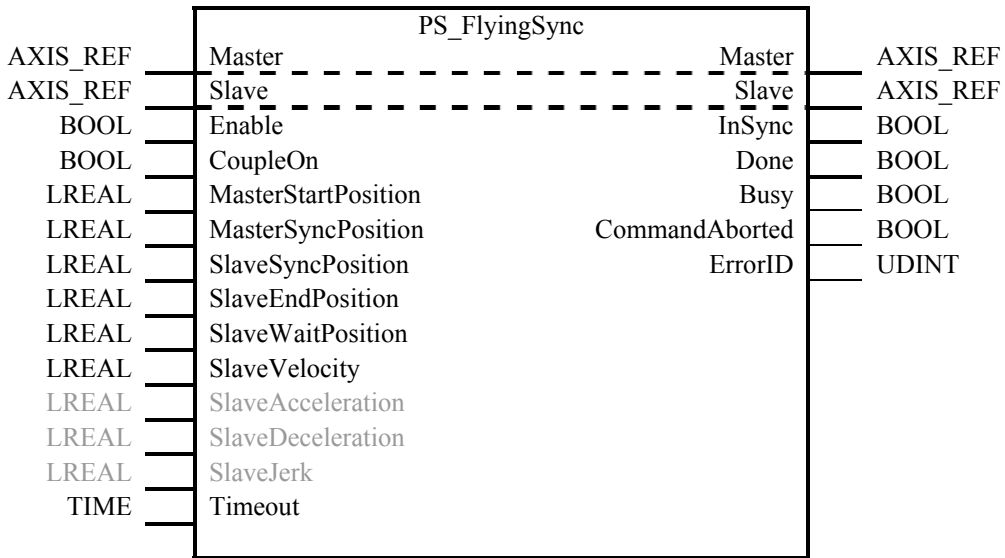
**Figure 15: Flying Sync Geometries**

FB-Name		<b>PS FlyingSync</b>	
<ul style="list-style-type: none"> <li><b>PS_FlyingSync</b> couples a slave axis to a master axis similar to a flying shear or flying saw. The synchronization speed is achieved precisely at the synchronization position and kept for the distance to the SyncEndposition. The FB allows to either provide a</li> </ul>			
VAR_IN_OUT			
B	Axis_Slave	AXIS_REF	Reference to axis to be coupled as a flying saw
B	Axis_Master	AXIS_REF	Reference to axis to be coupled to as Master for a flying saw
VAR_INPUT			
B	Enable	BOOL	Initiates the function block and moves slave initially to SlaveWaitPosition
B	CoupleOn	BOOL	Couple initiation, level sensitive -> loads Master-SyncPosition and SlaveSyncPosition for execution. On FALSE, coupling is disabled
B	MasterStartPosition	LREAL	Master position that determines the phase relation between master and slave axis [u]
B	MasterSyncPosition	LREAL	Master position where synchronized motion starts [u]
B	SlaveSyncPosition	LREAL	Corresponding slave position [u]
B	SlaveEndPosition	LREAL	Slave position where synchronized part of motion ends [u]
B	SlaveWaitPosition	LREAL	Slave position where slave axis waits [u]
B	SlaveVelocity	LREAL	Value of the maximum slave velocity (always positive) (not necessarily reached) [u/s].
E	SlaveAcceleration	LREAL	Value of the acceleration (always positive) (increasing energy of the motor) [u/s <sup>2</sup> ]
E	SlaveDeceleration	LREAL	Value of the deceleration (always positive) (decreasing energy of the motor) [u/s <sup>2</sup> ]
E	SlaveJerk	LREAL	Value of the Jerk [u/s <sup>3</sup> ]. (always positive)
B	Timeout	TIME	Supervisory timeout

VAR_OUTPUT			
B	InSync	BOOL	True if slave is in sync with Master
B	Done	BOOL	True once Slave is in wait position
B	Busy	BOOL	True once Slave is synchronizing
B	CommandAborted	BOOL	True once Command is aborted
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new command while still executing
B	ErrorID	UDINT	Error identification

Notes :

- Slave retracts after Slave reaches SyncEnd position to Wait Position
- Slave might either work in finite travel space (for flying shears function) by retracting to wait position
- Slave might work in infinite travel space (for cutter function) by traveling forward only
- If Slave Accel, Decel and Jerk are not supported, default values will apply.



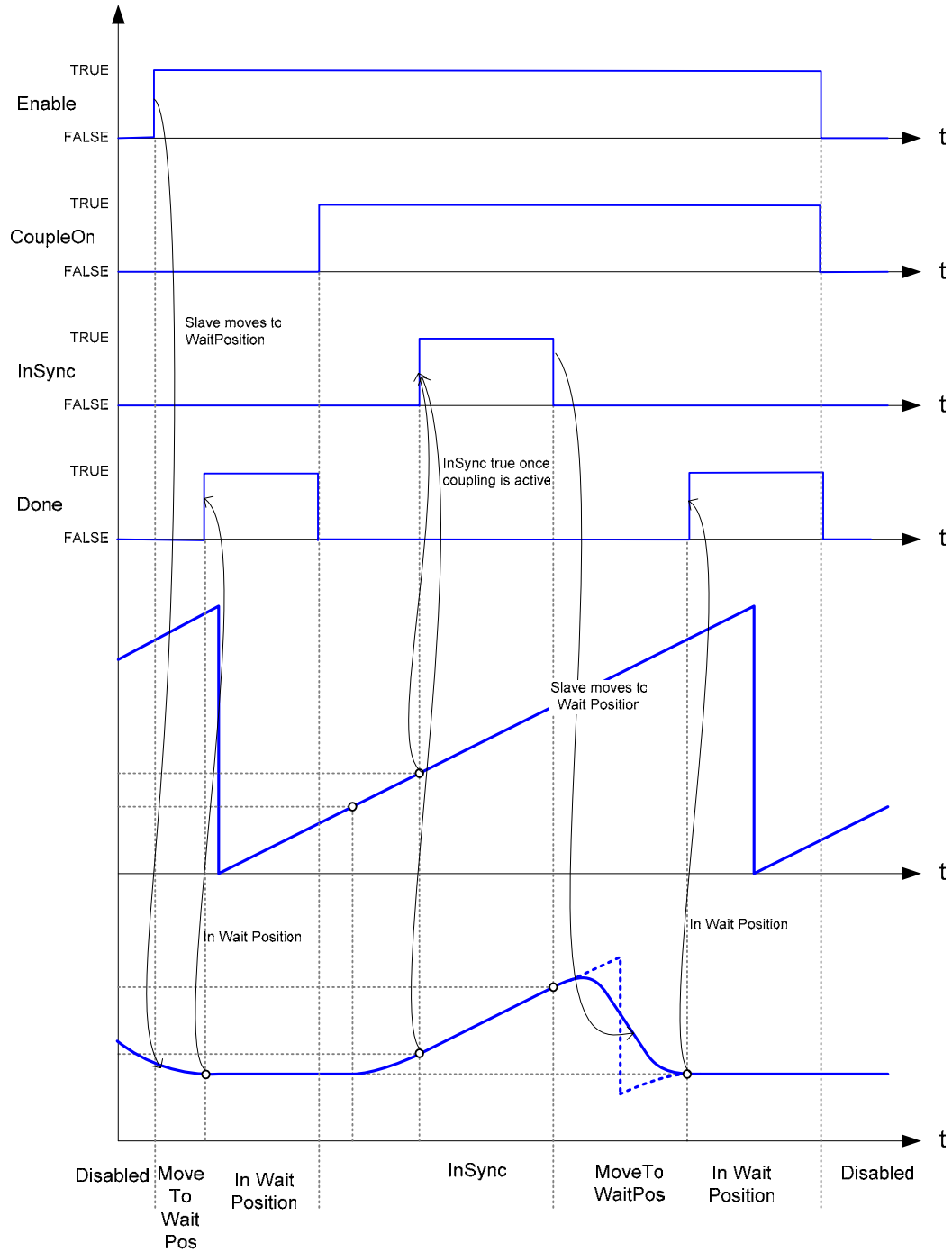
The figures above give a sketch that shows the function to be solved with this function block. The primary function is to cut a passing continuous flow of material at specified positions into discrete pieces (i.e. products). Optionally, the FB allows to apply a Slave Wait Position larger than Slave End Position to create the function of a rotating cutter.

Both functions are quite similar, with the difference that for the Flying Shear the slave axis features a finite range of motion and for the rotating cutter the slave axis continues to move in one direction.

The motion can be split into two parts:

1. Synchronization between master and slave axis and synchronous motion of both axes (where the actual task is performed)
2. Slave axis moves to a defined waiting position and waits for next action request

For the first part, the master and slave axes have to operate at synchronous speeds while maintaining a specified phase relationship (e.g. to assure an accurate cutting point). With this function block, the sequence of actions can be easily generated. Figure 2 shows a timing diagram of this process.



**Figure 16: PS\_FlyingSync timing diagram for a single synchronization**

### Gear\_In with dynamic Gear Factor

FB-Name		<b>PS_GearInDyn</b>	
<ul style="list-style-type: none"> <li><b>PS_GearInDyn</b> activates a linear master-slave coupling (gear coupling). The gear ratio can be modified in every control cycle. The block is therefore suitable for velocity controlled master slave couplings. The acceleration parameter limits the acceleration of the slave in case of high gear ratio changes.</li> </ul>			
VAR IN OUT			
B	Axis_Master	AXIS_REF	Reference to axis to be coupled to as Master
B	Axis_Slave	AXIS_REF	Reference to axis to be coupled
VAR INPUT			
B	Enable	BOOL	Coupling initiation, level sensitive. Initiates coupling and maintains during high, aborts on low signal
B	GearRatio	LREAL	Gearing factor. The gear ratio may be changed in every PLC cycle. 0.0 is invalid, default is 1.0
B	Acceleration	LREAL	Limit to acceleration of slave due to gear ration changes
B	Deceleration	LREAL	Limit to deceleration of slave due to gear ration changes
VAR OUTPUT			
B	Busy	BOOL	Becomes TRUE, as soon as the block is active. Busy becomes TRUE with a rising edge at Enable and becomes FALSE again after the block finished or aborted its operation. A high level at Enable will then be accepted.
B	InGear	BOOL	True if coupling is being successfully performed
B	CommandAborted	BOOL	True if slave is decoupled during Enable is TRUE
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new command while still executing
E	ErrorID	UDINT	Error identification
Notes :			

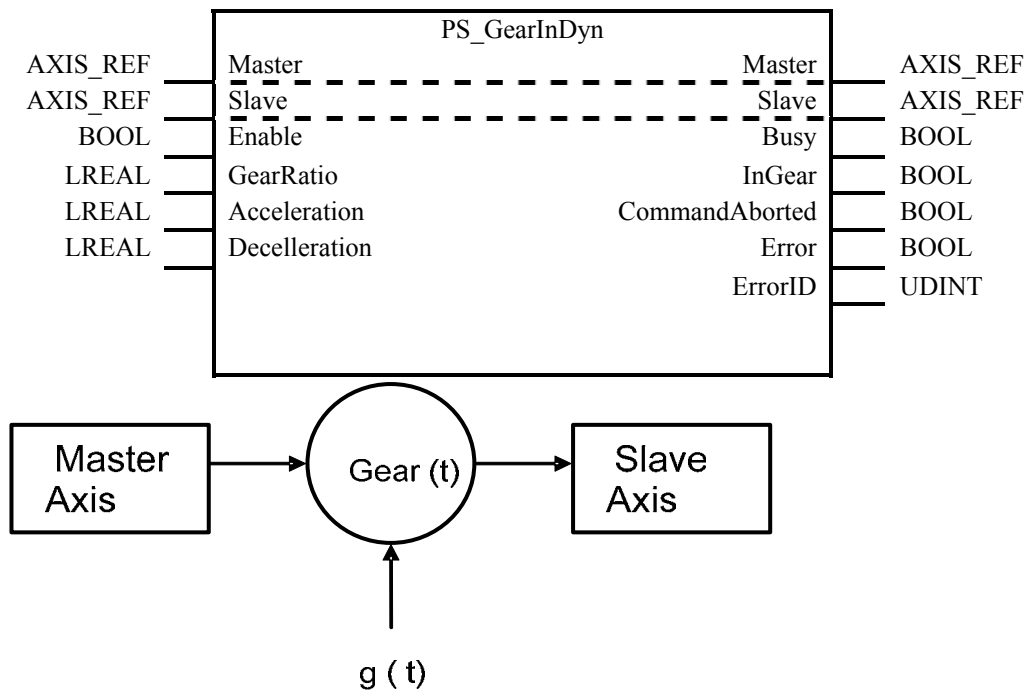
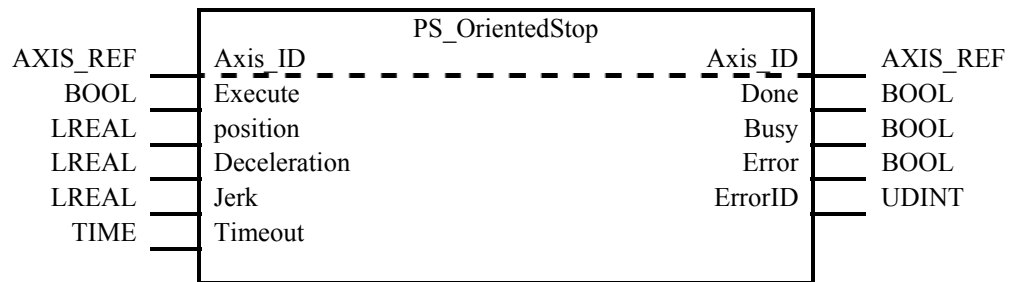
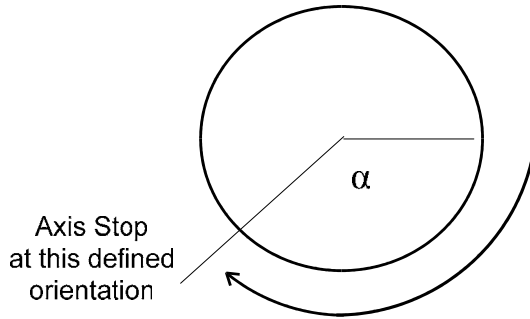


Figure 17: PS\_GearInDyn structure of gear model

### Motion Command Stop with oriented Halt

FB-Name		<b>PS_OrientedStop</b>		
<ul style="list-style-type: none"> <li><b>PS_OrientedStop</b> is used to affect an "oriented" axis stop at the specified modulo position. Depending on the speed and the dynamic parameters of the axis, it will stop at the next possible oriented position.</li> </ul> <p>For convenience in packaging, a stop with a resulting orientation of tool or product is often utilized. Therefore, this stop command calculates the deceleration ramp to final stop but will end always in the desired modulo position.</p>				
VAR_IN_OUT				
B	Axis_ID	AXIS_REF	Reference to axis to be stopped	
VAR_INPUT				
B	Execute	BOOL	Stop initiation, rising edge trigger	
B	fposition	LREAL	Absolute modulo destination position at which the axis should stop	
E	fDeceleration	LREAL	Limit to deceleration of axis during decel ramp. This input is optional. The default deceleration specified at the start of the axis is used for stopping.	
B	fJerk	LREAL	Limit to jerk of of axis during decel ramp. This input is optional. The default jerk specified at the start of the axis is used for stopping.	
B	Timeout	TIME	Supervisory timeout	
VAR_OUTPUT				
B	Done	BOOL	True if stop was successfully performed and the axis has come to a complete stop	
B	Busy	BOOL	True if coupling FB is being executed	
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new command while still executing	
E	ErrorID	UDINT	Error identification	
<p>Notes : The error output must be analyzed, because under certain conditions an oriented stop is not possible. For example, a standard stop may have been triggered just before, or an oriented stop would cause an active software limit switch to be exceeded. For all fault conditions, the axis is stopped safely, but it may subsequently not be at the required oriented position.</p> <p>The position must be in the range <math>[0 \leq \text{position} &lt; \text{modulo period}]</math>. With a modulo period of 360 degrees, for example, 360 is therefore not a valid position. A value of 0 has to be specified instead.</p>				





**Figure 18: PS\_OrientedStop graphical explanation of behavior**

### Packaging Machine State Blocks

Packaging Machine State Function Blocks provide a common interface to the existing PackML Machine State Model implementations available. It is expected that the application specific logic including the transitions between states is programmed in external function blocks, but the central logic of the state machine and the status representation should be handled by the Pack\_ML\_State\_Machine Function block. Therefore, this FB comes with a recommendation how to combine with other logic.

### Generic Machine State Function Block

#### PS\_PackML\_State\_Machine

FB-Name		<b>PS PackML State Machine</b>	
<ul style="list-style-type: none"> <li><b>PS_PackML_State_Machine</b> executes the PackML State Machine. Transitions need be programmed depending on machine application; however, the machine status and sequence of transitions shall satisfy the State Model listed for reference.</li> </ul>			
VAR IN OUT			
E	Machine_ID	Machine_REF	Reference to machine State Model runs on
VAR INPUT			
B	Execute	BOOL	Execute State Machine from rising edge
B	Abort	BOOL	to Aborting or to Aborted
B	Stop	BOOL	to Stopping or to Stopped
B	Start	BOOL	to Starting
B	Ready	BOOL	to Ready
B	Standby	BOOL	to Standby
B	Produce	BOOL	to Producing
B	Hold	BOOL	to Holding or to Held
B	Off	BOOL	to Off
E	Auto_Mode	BOOL	True if automatic mode is selected, otherwise FALSE

VAR OUTPUT			
B	Status	WORD	Status Word representing the status of the State Machine
B	ST_Aborting	BOOL	True if State Machine is in state Aborting
B	ST_Stopping	BOOL	True if State Machine is in state Stopping
B	ST_Stopped	BOOL	True if State Machine is in state Stopped
B	ST_Off	BOOL	True if State Machine is in state Off
B	ST_Starting	BOOL	True if State Machine is in state Starting
B	ST_Ready	BOOL	True if State Machine is in state Ready
B	ST_Standby	BOOL	True if State Machine is in state Standby
B	ST_Producing	BOOL	True if State Machine is in state Producing
B	STj_Holding	BOOL	True if State Machine is in state Holding
B	ST_Held	BOOL	True if State Machine is in state Held
E	ST_Manual	BOOL	True if Manual mode is selected
E	Error	BOOL	Signals that an error has occurred within Function Block
E	ErrorID	UDINT	Error identification

Notes :

For enhancement of the use for this state model, some variables were added as Extension Types:

To identify the machine executed by the state model, a Machine ID is implemented for convenient reference. Also, to switch from automatic mode to manual mode, a selector input is given. Error indications will be displayed by an Error ID. All inputs are scanned continuously except for Machine\_ID, which is latched at rising edge of Execute.

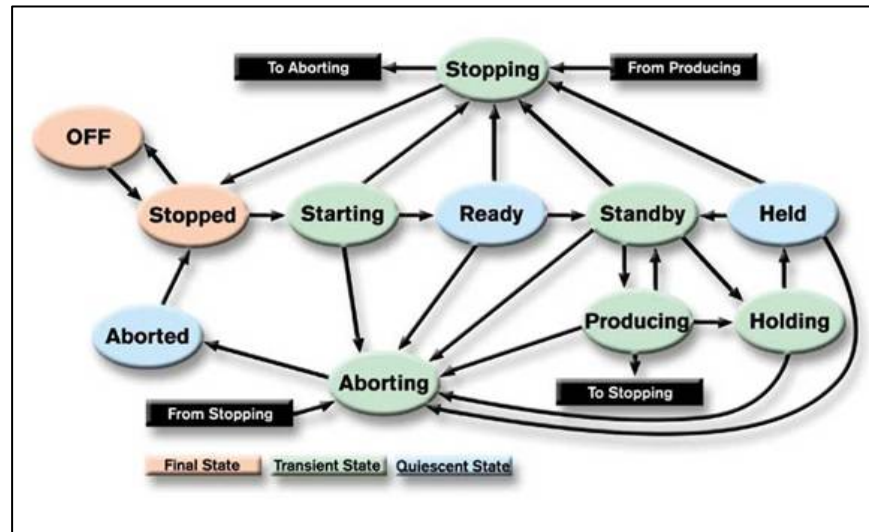
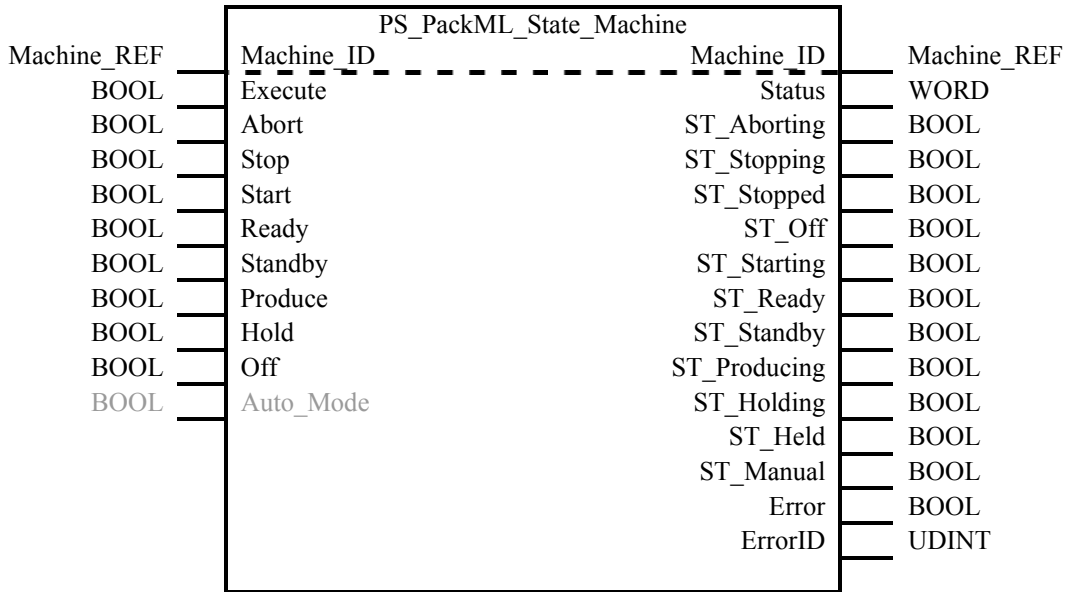


Figure 19: PackML State Model

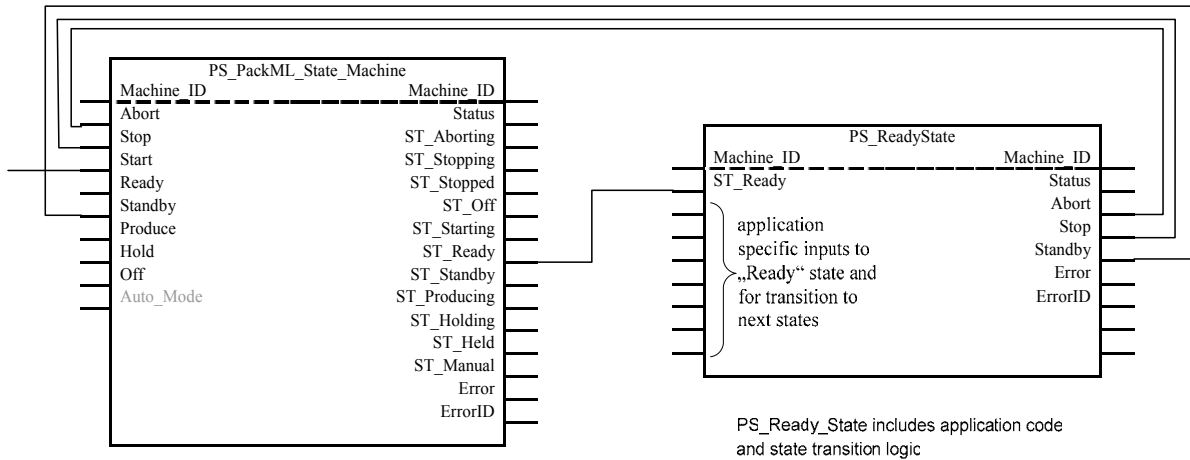


The state transitions to a machine application are always application specific. Therefore, to facilitate standardization, it should be best practice to form state function blocks that are connected to the PS\_PackML\_State\_Machine. The state function blocks collect application specific signals and form the transition logic to the adjacent states as displayed in the PackML state model. All state FB feed back into PS\_PackML\_State\_Machine, offering a standard state machine and state reporting. The state FBs will contain the machine execution code next to the application specific transition logic.

State Function Blocks are listed in Figure N and will be programmed by application programmer accordingly to maintain integrity and function of the PackML State Machine:

PS\_Pack\_\_ML\_State\_Model Function Block Names:

PS_AbortingState	PS_StandbyState
PS_StoppingState	PS_ProducingState
PS_StoppedState	PS_HoldingState
PS_OffState	PS_HeldState
PS_StartingState	PS_ManualState
PS_ReadyState	



**Figure 20: PS\_Pack\_ML\_State\_Model FB Names and application adoption**

PackML\_State\_Machine and application specific state Function Blocks (e.g., but not limited to, PS\_ReadyState) containing state logic and machine execution code. All other states require according Function Blocks to contain the application adoption. Therefore, above listed Function Block names are reserved for this state model:

### Packaging Communication Function Blocks

Packaging communication Function blocks provide basic methods to communicate between cell controllers. Three types of functions are defined:

- Send/ Receive Data FB for basic (serial) Point-to-Point communications,
- A confirmed handshake for Telegram based communications, as known with many protocols, to cover multi-Point communication,
- Publish /Subscribe for variable publishing based communications as alternative multi-point communication.

Intention is to standardize these three basic communications to be used independently of network standards to abstract from certain specific network technologies to allow software to run regardless of underlying network offered.

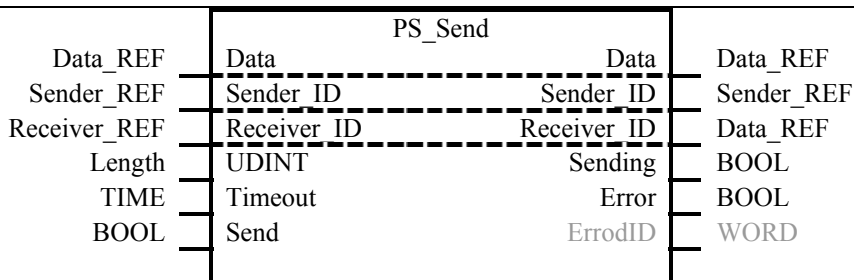
### Point to Point Communication by Function Block

Provides references to data elements e.g. variables or structures to communicate the content of these data elements. PackTags can be handed from control to control or to devices by means of these higher level function blocks.

PS\_Send, PS\_Receive - Provide simple methods of communication over cell bus or other media. Sender and Receiver ID must be known, the communication is a point to point communication.

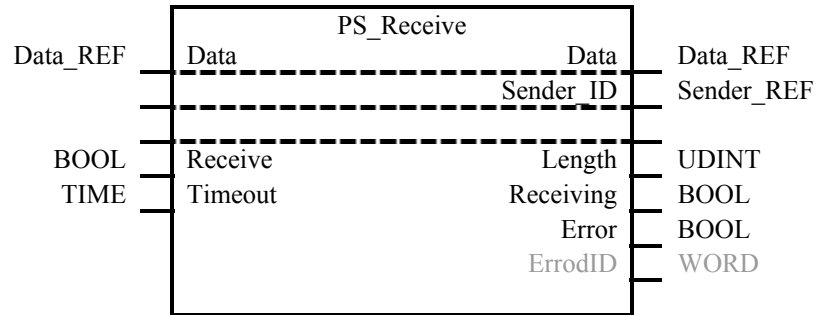
**PS\_Send**

FB-Name		<b>PS_Send</b>	
<ul style="list-style-type: none"> <li>This function block sends the data structure referenced to the receiver. This may take more than one cycle of control or network of communication.</li> </ul>			
VAR IN OUT			
B	Data_REF	Data_REF	Reference to data to be sent
B	Sender_ID	Sender_REF	Reference to Sender_ID
B	Receiver_REF	Receiver_REF	Reference to Receiver_ID
VAR INPUT			
B	Length	UDINT	Length of data set in bytes
B	Send	BOOL	Executes a send sequence
B	Timeout	TIME	Supervisory timeout
VAR OUTPUT			
B	Busy	BOOL	True if sending is being performed
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new send command while still executing
E	ErrorID	WORD	Error identification
Notes :			



**PS\_Receive**

FB-Name		<b>PS_Receive</b>	
<ul style="list-style-type: none"> <li>This function block receives data structures referenced to the receiver. This may take more than one cycle of control or network of communication.</li> <li>The FB is executed in every control cycle</li> </ul>			
VAR IN OUT			
B	Data_REF	Data_REF	Reference to data to be stored after reception
B	Sender_ID	Sender_REF	Reference to Sender_ID retrieved from transmission
VAR INPUT			
B	Receive	BOOL	Awaiting reception of data
B	Timeout	TIME	Supervisory timeout
VAR OUTPUT			
B	Length	UDINT	Length of data set in bytes retrieved from Message
B	Receiving	BOOL	True if receiving data
B	Error	BOOL	Signals that an error has occurred within Function Block
E	ErrorID	WORD	Error identification
Notes :			



### Message or Telegram based Communication

Establishes a confirmed connection between devices. The communication is message or telegram based, transferring data content in a transparent way. All information, e.g. sender and receiver header information, is encapsulated in the data field of implemented telegram in a transparent way. These FBs will not handle protocol specific content but manage raw telegram exchange, thus tunneling any specific information between sender and receiver.

The progress of this communication begins with an CommRequest, which arrives at the server where it invokes a CommIndication. The server replies with a CommResponse, which in turn is received by the client and creates a CommConfirmation flag.

Client	Telegram Transmission	Server
Request ->	-> Request telegram	-> Indication
Confirmation <-	<- Response telegram	<- Response

The sender creates a Communication Request, triggering the initiation of the communication by a first telegram. The Server receives the telegram, which creates an indication, and calculates a reply, transmitted by the response FB. The receipt of the answer telegram creates a confirmation at the client side. The communication is carried out by two telegrams in a confirmed way.

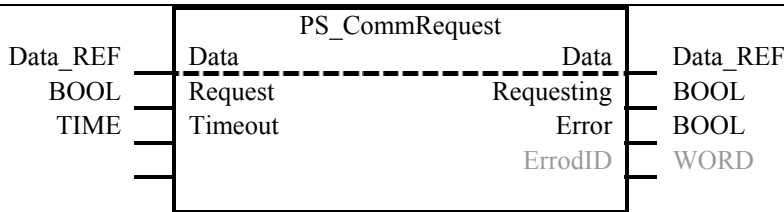
Messages which are sent autonomously by a server (e.g. error or other status messages) are registered by the client as a notification indication:

Client	Telegram Transmission	Server
Notification Indication <-	<- Notification telegram	<- Notification

The sender creates a Notification, which arrives at the Client without a previous Request handle. Therefore, it is noted as a Notification, forming an unconfirmed communication of data.

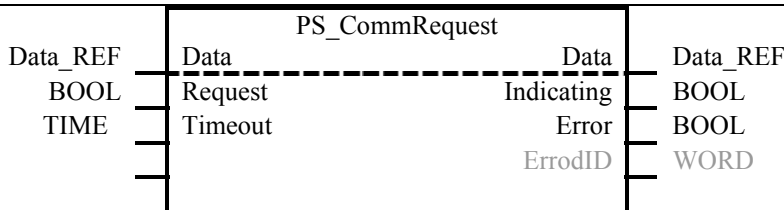
### PS\_CommRequest

FB-Name		<b>PS_CommRequest</b>	
<ul style="list-style-type: none"> <li>This function block is used by a client and initiates a confirmed connection via a PS_CommunicationRequest The FB is executed once and triggers the request telegram.</li> </ul>			
VAR IN OUT			
B	Data_ID	Data_REF	Reference to Data_ID
VAR INPUT			
B	Request	BOOL	Executes a Request sequence to establish a connectin
B	Timeout	TIME	Supervisory timeout
VAR OUTPUT			
B	Requesting	BOOL	True after FB Executed; cleared by Termination or Loss of connection (Timeout)
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new request command while still executing
E	ErrorID	WORD	Error identification
Notes :			



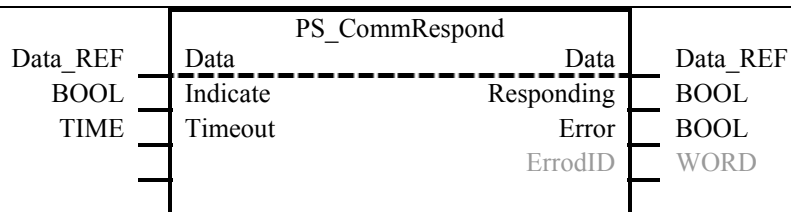
### PS\_Indicate

FB-Name		<b>PS_CommIndicate</b>	
<ul style="list-style-type: none"> <li>This function block is executed by a server and responds to a CommRequest by a client, indicating availability to communicate to establish a confirmed connection. The FB is executed once.</li> </ul>			
VAR IN OUT			
B	Data_ID	Data_REF	Reference to Data_ID
VAR INPUT			
B	Request	BOOL	Executes a Request sequence to establish a connectin
B	Timeout	TIME	Supervisory timeout
VAR OUTPUT			
B	Indicate	BOOL	True after FB Executed; cleared by Termination or Loss of connection (Timeout)
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new request command while still executing
E	ErrorID	WORD	Error identification
Notes :			



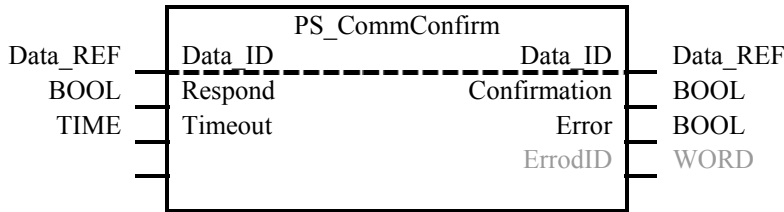
### PS\_CommRespond

FB-Name		<b>PS_CommRespond</b>		
<ul style="list-style-type: none"> <li>This function block is executed by a server and responds back to a CommIndication by transmission of a telegram, confirming the establishing of a confirmed connection by a response. The FB is executed once and triggers the response telegram.</li> </ul>				
VAR_IN_OUT				
B	Data_ID	Data_REF	Reference to Data_ID	
VAR_INPUT				
B	Indicate	BOOL	Triggers a Respond sequence to establish a connection, edge-trigger	
B	Timeout	TIME	Supervisory timeout	
VAR_OUTPUT				
B	Respond	BOOL	True after FB Executed; cleared by Termination or Loss of connection (Timeout)	
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new request command while still executing	
E	ErrorID	WORD	Error identification	
Notes :				



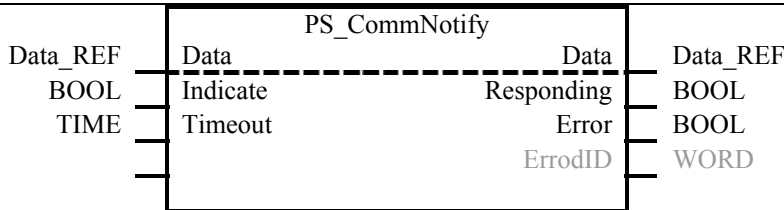
### PS\_CommConfirm

FB-Name		<b>PS_CommConfirm</b>		
<ul style="list-style-type: none"> <li>This function block is executed by the client and terminates the communication from a CommResponse, handling the received data. The FB is executed once.</li> </ul>				
VAR_IN_OUT				
B	Data_ID	Data_REF	Reference to Data_ID	
VAR_INPUT				
B	Respond	BOOL	Triggers a Confirmation sequence to establish a connection, edge-trigger	
B	Timeout	TIME	Supervisory timeout	
VAR_OUTPUT				
B	Confirmation	BOOL	True after FB Executed; cleared by Termination or Loss of connection (Timeout)	
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new request command while still executing	
E	ErrorID	WORD	Error identification	
Notes :				



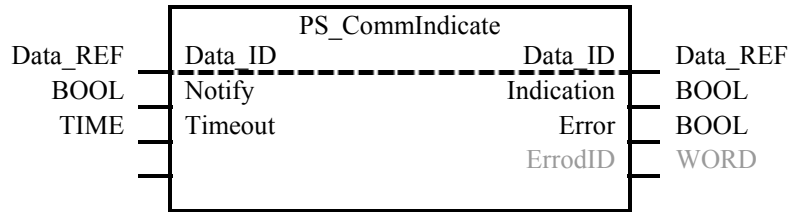
### PS\_CommNotify

FB-Name		<b>PS_CommNotify</b>	
<ul style="list-style-type: none"> <li>This function block is executed by a server and creates an immediate notification by telegram transmission, creating an unconfirmed and unrequested message to a client. The FB is executed once and does not trigger a response telegram.</li> </ul>			
VAR IN OUT			
B	Data_ID	Data_REF	Reference to Data_ID
VAR INPUT			
B	Notify	BOOL	Triggers a notification telegram, edge-trigger
B	Timeout	TIME	Supervisory timeout
VAR OUTPUT			
B	Notified	BOOL	True after FB Executed; cleared by Termination or Loss of connection (Timeout)
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new request command while still executing
E	ErrorID	WORD	Error identification
Notes :			



### PS\_CommIndicate

FB-Name		<b>PS_CommIndicate</b>	
<ul style="list-style-type: none"> <li>This function block is executed by the client and terminates the communication from a CommResponse, handling the received data. The FB is executed once.</li> </ul>			
VAR IN OUT			
B	Data_ID	Data_REF	Reference to Data_ID
VAR INPUT			
B	Notify	BOOL	Enables the FB to receive Notification during execution of block, level triggered
B	Timeout	TIME	Supervisory timeout
VAR OUTPUT			
B	Indication	BOOL	True after receipt of Notification; cleared by Termination of FB execution or Loss of connection (Timeout)
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new request command while still executing
E	ErrorID	WORD	Error identification
Notes :			



### Communication by Publisher/Subscriber

The Publisher/subscriber model allows to exchange data on a network by means of publishing and subscribing to variables. It offers widest flexibility and asynchronous as well as synchronous connection methods independent from network technologies.

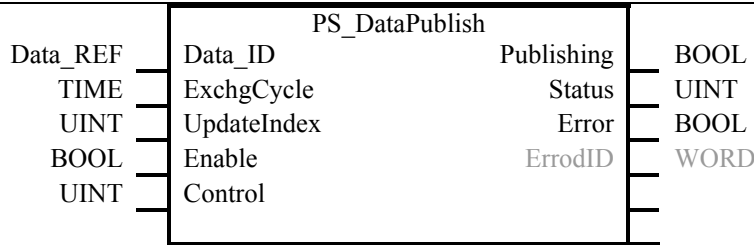
The sender publishes variables (data structures) at a certain cycle time without confirmed connections. Subscribers (one or many) may consume the published information at a different cycle time, synchronous or not. A quality variable informs about the age of information in sender cycle counts.

### PS\_Publish

Publishes a variable or structure on a network. Certain additional “quality of information” meta-data is published additionally to provide information to the subscriber about age and timeliness of data retrieved by subscription.

FB-Name		<b>PS_DataPublish</b>	
<ul style="list-style-type: none"> <li>• This function block publishes variables (a data structure referenced).</li> <li>• User defines a network update time period, the Exchange Cycle Time</li> <li>• User increments the cycle index on each data update provided to FB</li> </ul>			
VAR INPUT			
B	Data_ID	Data_REF	Input Reference to Data to be published
B	ExchgCycle	TIME	Input The cycle time of publishing (transmission) cycle
B	UpdateIndex	UINT	Input by user: Counts cyclically from 0 to 65535 continuously as a publishing cycle counter for every update of Data the user provides to the FB for publication. Not a counter to accrued network cycles.
B	Enable	BOOL	Triggers a publishing of a data structure
B	Control	UINT	Control Word for Publication purposes, implementation specific
VAR OUTPUT			
B	Publishing	BOOL	True while FB executes
B	Status	UINT	Status information
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new command while still executing
E	ErrorID	WORD	Error identification

Notes : Status information includes at minimum:  
 01h:                0: no error, +  
                          1: transmission link status error,  
 02h:                0: no error,  
                          1: cycle time violation, the transmission rate, protocol and size of data structure do not allow transmission of data within selected Exchange cycle time



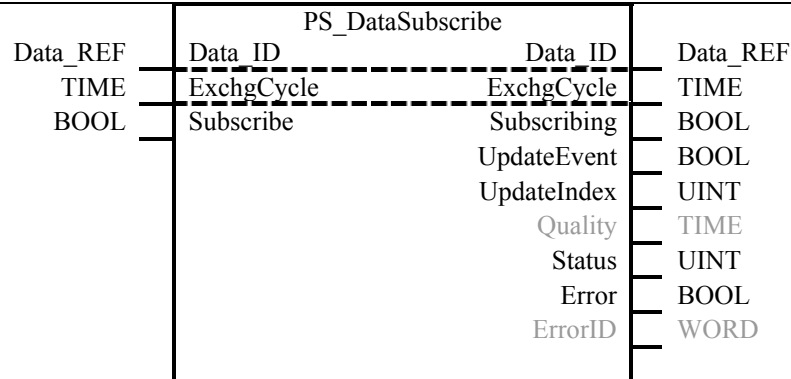
### PS\_Subscribe

Subscribes to a variable or structure on a network. Certain additional quality of information data is to be retrieved from the transmitted data, e.g. the CycleIndex.

FB-Name		<b>PS DataSubscribe</b>	
<ul style="list-style-type: none"> <li>This function block subscribes to variables (a data structure referenced).</li> </ul>			
VAR IN OUT			
B	Data_ID	Data_REF	Reference to Data to be published
B	ExchgCycle	TIME	The cycle time of publishing (transmission) cycle
VAR INPUT			
B	Subscribe	BOOL	Triggers a subscription of a data structure, level sensitive
VAR OUTPUT			
B	Subscribing	BOOL	True while FB executes
B	UpdateEvent	BOOL	Rising edge on every increment of UpdateIndex as notification to receiver
B	UpdateIndex	UINT	Output from subscribed Data: counter cycles from 0 to 65535 continuously as a publishing cycle counter for every Data update containing new information the user provides.
E	Quality	TIME	This variable shows how "old" the value of this subscriber network variable is by a timer implementation in the subscriber. As long as Data is not updated on the network, this value gets increased by 100 μs increments. This could occur e.g. by asynchronous nature of transmission or when the link of the network connection is lost. After re-establishing the connection, the value is set back to 0 again. Max. value of <b>Quality</b> is 65535 sec, then starting at 0.
B	Status	UINT	Status information if the Data subscription
B	Error	BOOL	Signals that an error has occurred within Function Block, e.g. new request command while still executing
E	ErrorID	WORD	Error identification

Notes : Status information includes at minimum:

- 01h: 0: no error, +  
1: transmission link status error,
- 02h: 0: no error,  
1: cycle time violation, the transmission rate, protocol and size of data structure do not allow reception of data within selected Exchange cycle time
- 04h: Type or Size mismatch between Data\_REF and received data
- 08h: Cycle Time Violation Flag, data transmission is slower than configured publication cycle time.



## CONCLUSIONS

PackAL packaging application function block library version 1.01 user benefits include a common look and feel in packaging software; more efficient training; easier maintenance and troubleshooting; more time to focus on process issues; an open, user and vendor expandable library; common platform-independent software elements; utilization in any controls or drives; common device behavior; and a platform independent software standard. PackAL defines the interfaces and behavior, not the implementations.

## REFERENCES

References to other standards:

IEC 61131-3 2nd edition, 2003

Machine Directive 98/37/EC, clause 1.2.5. - and related US and Asia directives

Function Blocks for Motion Control, V. 1.0, PLCopen, 2001

Function Blocks for Motion Control, Part 2, Extensions, PLCopen, 2005

## **Appendix V: Reserved**

---

## **Appendix VIA: Benefit Examples**

---

(To Follow)

## Acronym Reference

For a complete list of industry acronyms, refer to the ARC Advisory Group web page at [www.arcweb.com/arcweb/Community/terms/indterms.htm](http://www.arcweb.com/arcweb/Community/terms/indterms.htm)

<b>API</b>	Application Program Interface
<b>COTS</b>	Commercial Off-the-Shelf
<b>ERP</b>	Enterprise Resource Planning
<b>HMI</b>	Human Machine Interface
<b>IT</b>	Information Technology
<b>LAN</b>	Local Area Network
<b>OMAC</b>	Open Modular Architecture Control
<b>OIT</b>	Operator Interface Terminal
<b>OPC</b>	OLE for Process Control
<b>PLC</b>	Programmable logic controller
<b>TCP</b>	Transmission Control Protocol
<b>IP</b>	Internet Protocol
<b>ROI</b>	Return on Investment
<b>SPC</b>	Statistical Process Control

The OMAC Users Group was formed in 1997 to create an organization through which companies could work together to promote development and adoption of open automation controls. OMAC Users Group membership is open to all manufacturing automation users, suppliers, system integrators, and technology providers. Further information is available at <http://www.omac.org>



**Open  
Modular  
Architecture  
Controls**



<http://www.omac.com>

