ANSI/ISA-TR88.00.02-2015

A Technical Report prepared by ISA and registered with ANSI

Machine and Unit States: An implementation example of ANSI/ISA-88.00.01

Approved 26 October 2015

ANSI/ISA-TR88.00.02-2015, Machine and Unit States: An implementation example of ANSI/ISA-88.00.01

ISBN: 978-1-941546-65-9

ISA
67 Alexander Drive
P. O. Box 12277
Research Triangle Park, NC 27709 USA

Preface

This preface, as well as all footnotes, is included for information purposes only and is not part of ANSI/ISA-TR88.00.02-2015.

This technical report has been prepared as part of the service of ISA, the International Society of Automation, toward a goal of uniformity in the field of instrumentation. To be of real value, this document should not be static but should be subject to periodic review. Toward this end, the Society welcomes all comments and criticisms and asks that they be addressed to the Secretary, Standards and Practices Board; ISA, 67 Alexander Drive; P.O. Box 12277; Research Triangle Park, NC 277099; Telephone (919) 549-8411; Fax (919) 549-8288; E-mail: standards@isa.org.This ISA Standards and Practices Department is aware of the growing need for attention to the metric system of units in general, and the International System of Units (SI) in particular, in the preparation of instrumentation standards, recommended practices, and technical reports. The Department is further aware of the benefits of USA users of ISA standards of incorporating suitable references to the SI (and the metric system) in their business and professional dealings with other countries. Toward this end, the Department will endeavor to introduce SI and acceptable metric units in all new and revised standards to the greatest extent possible. The Metric Practice Guide, which has been published by the Institute of Electrical and Electronics Engineers (IEEE) as ANSI/IEEE Std. 268-1992, and future revisions, will be the reference guide for definitions, symbols, abbreviations, and conversion factors.

It is the policy of ISA to encourage and welcome the participation of all concerned individuals and interests in the development of ISA standards. Participation in the ISA standards-making process by an individual in no way constitutes endorsement by the employer of that individual, of ISA, or of any of the standards, recommended practices, and technical reports that ISA develops.

This standard is structured to follow the IEC guidelines. Therefore, the first three sections discuss the *Scope* of the standard, *Normative References* and *Definitions*, in that order.

CAUTION — ISA ADHERES TO THE POLICY OF THE AMERICAN NATIONAL STANDARDS INSTITUTE WITH REGARD TO PATENTS. IF ISA IS INFORMED OF AN EXISTING PATENT THAT IS REQUIRED FOR USE OF THE STANDARD, IT WILL REQUIRE THE OWNER OF THE PATENT TO EITHER GRANT A ROYALTY-FREE LICENSE FOR USE OF THE PATENT BY USERS COMPLYING WITH THE STANDARD OR A LICENSE ON REASONABLE TERMS AND CONDITIONS THAT ARE FREE FROM UNFAIR DISCRIMINATION.

EVEN IF ISA IS UNAWARE OF ANY PATENT COVERING THIS STANDARD, THE USER IS CAUTIONED THAT IMPLEMENTATION OF THE STANDARD MAY REQUIRE USE OF TECHNIQUES, PROCESSES, OR MATERIALS COVERED BY PATENT RIGHTS. ISA TAKES NO POSITION ON THE EXISTENCE OR VALIDITY OF ANY PATENT RIGHTS THAT MAY BE INVOLVED IN IMPLEMENTING THE STANDARD. ISA IS NOT RESPONSIBLE FOR IDENTIFYING ALL PATENTS THAT MAY REQUIRE A LICENSE BEFORE IMPLEMENTATION OF THE STANDARD OR FOR INVESTIGATING THE VALIDITY OR SCOPE OF ANY PATENTS BROUGHT TO ITS ATTENTION. THE USER SHOULD CAREFULLY INVESTIGATE RELEVANT PATENTS BEFORE USING THE STANDARD FOR THE USER'S INTENDED APPLICATION.

HOWEVER, ISA ASKS THAT ANYONE REVIEWING THIS STANDARD WHO IS AWARE OF ANY PATENTS THAT MAY IMPACT IMPLEMENTATION OF THE STANDARD NOTIFY THE ISA STANDARDS AND PRACTICES DEPARTMENT OF THE PATENT AND ITS OWNER. ADDITIONALLY, THE USE OF THIS STANDARD MAY INVOLVE HAZARDOUS MATERIALS, OPERATIONS OR EQUIPMENT. THE STANDARD CANNOT ANTICIPATE ALL POSSIBLE APPLICATIONS OR ADDRESS ALL POSSIBLE SAFETY ISSUES ASSOCIATED WITH USE IN HAZARDOUS CONDITIONS. THE USER OF THIS STANDARD MUST EXERCISE SOUND PROFESSIONAL JUDGMENT CONCERNING ITS USE AND APPLICABILITY UNDER THE USER'S PARTICULAR CIRCUMSTANCES. THE USER MUST ALSO CONSIDER THE

APPLICABILITY OF ANY GOVERNMENTAL REGULATORY LIMITATIONS AND ESTABLISHED SAFETY AND HEALTH PRACTICES BEFORE IMPLEMENTING THIS STANDARD.

THE USER OF THIS DOCUMENT SHOULD BE AWARE THAT THIS DOCUMENT MAY BE IMPACTED BY ELECTRONIC SECURITY ISSUES. THE COMMITTEE HAS NOT YET ADDRESSED THE POTENTIAL ISSUES IN THIS VERSION.

The following people served as active members in the revision of this document:

NAME	COMPANY
David Bell	ATR Distributing Co (Wonderware)
Jerry Golden	B&R Automation
John Kowal	B&R Automation
Marc Wolf	B&R Automation
Ryan Stell	B&R Automation
Carl Bostrom	Bosch Rexroth
Dave Chappell	CMAa
Arthur Smith	Corning
James Hoban	Eaton
Kevin Pitts	Eaton
Doug Buschor	E-Technologies
John Spengler	GE
Niels Andersen	Invensys
Kent Knudsen	Krones
Brian Leboeuf	Krones
Uwe Keiter	Lenze
Lee Smith	Mettler Toledo
Steven Abramowski	Miller Coors
Jerry Yen	Mitsubishi Electric
David Kaley	Mitsubishi Electric
Vili Taukolo	Mitsubishi Electric
Fabrice Bertin	Nestlé
Tom Doney	Nestlé
Bryan Griffen	Nestlé
Toshiya Kitagawa	Omron
Tatsuru Hyodo	Omron
Rick Schoonover	Patti Engineering
Jeff Russell	PepsiCo
Murugan Govindasamy	Pfizer
Eelco van der Wal	PLCopen
Tom Egan	PMMI
Mark Ruberg	Pro Mach
Christopher Thomas	Pro Mach (Axon)
Gord Davison	Pro Mach (Edson)
Jason DeBruler	Procter & Gamble
Dan Amundson	Procter & Gamble
John Dart	Rockwell Automation
Dan Seger	Rockwell Automation
Yann Renard	Sidel
Stephane Hacpille	Sidel
Mike Pieper	Siemens Industry
Rana, Ajay S	Siemens Industry
Alexander Stukenkemper	Siemens Industry
Oliver Welle	Siemens AG
Dewayne A Bruschi	SMC
R Johnson	SMC
Doug Meyer	Yaskawa

The following people served as active members in the initial creation of this document:

NAME	COMPANY
Joseph Jablonski	Acumence Inc
Barry Kluener	Alexander & Associates
David Bell	ATR Distribution (Wonderware)
Thomas Hopfgartner	B&R Automation
Uwe Keiter	B&R Automation
Gerd Hoppe	Beckhoff
David Arens	Bosch Rexroth
Dennis Brandl	BR&L Consulting
David Chappell	CMAa-LLC
Joe Faust	Douglas Machine Company
Dominik Gludowatz	Elau
Tom Jensen	Elau
Andre Uhl	Elau
David Bauman	ISA / OMAC
Alex Pereira	KHS
Randy Dwiggins	Maverick Technologies
Ron MacDonald	Nestlé
Eelco VanDerWal	PLCopen
Mike Lamping	Procter & Gamble
Ulrich Arlt	Rockwell Automation
Darren Elliott	Rockwell Automation
Brian Hedges	Rockwell Automation
John Dart	Rockwell Automation
Paul Nowicki	Rockwell Automation
Dan Seger	Rockwell Automation
Garth Basson	SAB Miller
Larry Trunek	SAB Miller
Willie Lotz	SAB Miller Brewing Co.
Fabian Ochoa M.	SAB Miller Brewing Co.
Mario Broucke	Siemens AG, A&D
Roland Heymann	Siemens AG, A&D
Detlef Rausch	Siemens AG, A&D
Robert Freller	Siemens AG, F&B
Dr. Holger Grzonka	Siemens Energy & Automation
Mike Pieper	Siemens Energy & Automation
Mark DeCramer	WAGO
Dr. Tobias Voigt	Weihenstephan University

This document was approved for publication by the ISA Standards and Practices Board on 1 July 2015.

NAME	**AFFILIATION**
N. Sands, Vice President	DuPont
D. Bartusiak	ExxonMobil Research & Engineering
P. Brett	Honeywell Inc.
E. Cosman	OIT Concepts, LLC
K. Demachi	Yokogawa Electric Corp.
B. Dumortier	Schneider Electric
D. Dunn	Consultant
J. Federlein	Federlein & Assoc. Inc.
B. Fitzpatrick	Wood Group Mustang

J. Gilsinn	Kenexis Consulting
J.-P. Hauet	KB Intelligence
E. Icayan	Atkins
J. Jamison	Encana Corp.
K. P. Lindner	Endress + Hauser Process Solutions AG
V. Maggioli	Feltronics Corp.
T. McAvinew	Instrumentation and Control Engineering, LLC
V. Mezzano	Fluor Corporation
C. Monchinski	Automated Control Concepts Inc.
H. Sasajima	Azbil Corp.
T. Schnaare	Rosemount Inc.
J. Tatera	Tatera & Associates Inc.
K. Unger	Stone Technologies Inc.
I. Verhappen	Orbis Engineering Field Services
W. Weidman	WCW Consulting
J. Weiss	Applied Control Solutions LLC
M. Wilkins	Yokogawa IA Global Marketing (USMK)
D. Zetterberg	Chevron Energy Technology Co.

CONTENTS

LIST OF FIGURES AND TABLES

This page intentionally left blank.

Foreword

The ISA88 committee has defined a series of standards addressing the batch industry and providing terminology and a consistent set of concepts and models for batch manufacturing plants and batch control. These standards, however, were not defined in the context of packaging machines, or machines that perform discrete operations. As the ANSI/ISA88 batch standard continues to evolve, the context of the standard models may be extended to include the entire plant, integrating the software definitions of batch, packaging, converting and warehousing. Currently, as noted in this report there is a need to begin consideration of the ANSI/ISA88 standard in the context of differing automated machinery.

This is an informative document. This document contains implementation guidelines in order to establish a common presentation and high level software architecture or layout. The terms and definitions used in this document are harmonized, as much as possible, with ANSI/ISA-88.00.01; the document is not definitive in this respect. The models used, and applied, in this document are an extension of the models presented in ANSI/ISA-88.00.01 and are shown how they are applied to differing machine functionality. Discrete machine functionality is expressed graphically in several situations and described. The intent of this document is proposing specific implementation options and indicates a preference for a specific set of machine types.

In 2013 this document was updated for three reasons: to simplify the document and enable easier adoption, to clarify existing materials and make them easier to apply and to make it more complete, including the best approaches being used to implement. Major changes include addition of a minimum set of PackTags, a minimum set of states and removal of examples and MES definitions not central to the document's purpose. Other changes addressed in this revision are: improved definition of suspending and holding, transition between modes, blocked and starved tags, stop reasons, and warnings.

Publication of this technical report that has been registered with ANSI has been approved by ISA, 67 Alexander Drive, Research Triangle Park, NC 27709. This document is registered as a technical report according to the Procedures for the Registration of Technical Reports with ANSI. This document is not an American National Standard and the material contained herein is not normative in nature. Comments on the content of this document should be sent to ISA, 67 Alexander Drive, Research Triangle Park, NC 27709.

Abstract

The approaches used in programming discrete machines today are generally considered to be solely dependent on the machine and the software engineer, or control systems programmer. This constant change offers little additional value and generally increases the total costs, from the designing and building of the process to operating and maintaining the system by the end user. This technical report on the implementation of ANSI/ISA-88.00.01 in discrete machines breaks this paradigm and demonstrates how to apply the ANSI/ISA-88.00.01 standard concepts to automated machine states and modes. This technical report gives examples of general and specific machine state models and procedural methods. The report cites real control examples as implementations, and provides specific tag naming conventions; it also cites a number of common terms that are consistent with batch processing and ANSI/ISA-88.00.01.

Key words

state machine, state model, mode manager, machine state, unit control mode, PackML, state commands, command tags, status tags, administration tags, base state model, functional programming, modular programming, machine control software, discrete machine software, PackTags, Weihenstephan, Production Data Acquisition, PDA, ISA88, TR88.

Introduction

When the ANSI/ISA-88.00.01 standard is applied to applications across a plant, there is a need to align the terminologies, models and key definitions between different process types: continuous, batch, and discrete processes. Discrete processes involve machines found in the packaging, converting, and material handling applications. The operation of these machines is typically defined by the OEM, system integrator, end user, or is industry specific.

A task group with members from technology providers, OEMs, system integrators, and end users was chartered by the OMAC (Organization for Machine Automation and Control) Packaging Workgroup. The task group generated the PackML guidelines as a method to show how the ANSI/ISA-88.00.01 concepts could be extended into packaging machinery. This technical report is intended to build upon and formalize the concepts of the PackML guidelines and to show application examples.

The purpose of the technical report is to:

- Define a standard state-based model for automated machines.
- Identify definitions for common terminology.
- Explain to practitioners how to use state programming for automated machines.
- Provide references to actual implementation examples and templates from automation and control vendors.
- Identify a common tag structure for automated machines in order to:
  - Provide for “connect & pack” functionality
  - Provide functional interoperability and a consistent look and feel across the plant floor.
  - Provide consistent tag structure for connection to plant MES and enterprise systems.

Machine and Unit States
ANSI/ISA-TR88.00.02-2015

1 Scope

Since its inception, the OMAC Packaging Machine Language (PackML) group has been using a variety of information sources and technical documents to define a common approach, or machine language, for automated machines. The primary goals are to encourage a common "look and feel" across a plant floor, and to enable and encourage industry innovation. The PackML group is recognized globally and consists of control vendors, OEM's, system integrators, universities, and end users, which collaborate on definitions that endeavour to be consistent with the ISA88 standards and consistent with the technology and the changing needs of a majority of automated machinery. The term "machine" used in this report is equivalent to an ISA88 "unit".

This has led to the following:
1. A definition of machine/unit state types
2. A definition of machine/unit control modes
3. A definition of unit control mode management
4. State models, state descriptions, and mode and state transitions
5. A definition of the minimum PackTags required for performance monitoring

2 References

The following documents contain provisions that are referenced in this text. At the time of publication the editions indicated were valid. All documents are subject to revision, and parties to agreements based on this technical report are encouraged to investigate the possibility of applying the most recent editions of the reference documents indicated below.

— ANSI/ISA-88.00.01-2010, Batch Control - Part 1: Models and Terminologies

— ISA-88.00.02-2001, Batch Control - Part 2: Data Structures and Guidelines for Languages

— ANSI/ISA-88.00.03-2003, Batch Control - Part 3: General and Site Recipe Models and Representation

— ANSI/ISA-88.00.04-2006, Batch Control - Part 4: Batch Production Records

— ISA Draft 88.00.05 Batch Control - Part 5: Implementation Models & Terminology for Modular Equipment Control

— IEC 61131-1 Programmable controllers – Part 1: General Information

— IEC 61131-3 Programmable controllers - Part 3: Programming Languages

— IEC TR61131-4 Programmable controllers - Part 4: User Guidelines

— PLCopen TC5 Safety Software

— Weihenstephan Standard – Part 2 Version 2005 www.weihenstephaner-standards.de/index.php?id=2&L=1

— ANSI/ISA-95.00.01-2010 (IEC 62264-1 Mod) Enterprise – Control System Integration - Part 1: Models and Terminologies

— ANSI/ISA-95.00.02-2010 (IEC 62264-2 Mod) Enterprise – Control System Integration - Part 2: Object Model Attributes

— ANSI/ISA-95.00.05-2013, Enterprise - Control System Integration - Part 5: Business-to-Manufacturing Transactions

— DIN 8782, Beverage Packaging Technology; Terminology Associated with Filling Plants and their Constituent Machines

— www.omac.org – Organization for Machine Automation and Control Website

— ISA-TR88.00.02-2008 – The original edition of this technical report.

— ISO 22400 Automation Systems and Integration – Key performance indicators (KPIs) for manufacturing operations management.

3 Overview

3.1 Introduction

Automated machine programming is typically done by software engineers, machine designers, and system integrators. The form and style of the machine software ranges from modular, to monolithic in nature. The objective of this report is to specify the application of a common software methodology that is consistent with the modular programming of automated machinery as described in the draft ISA-88.00.05 standard. The naming of specific software components, or operational aspects, is dependent on the needs of the automated machine. This report shall be interpreted in a general sense to encompass all automated machinery. It is focused on the overall operation and functionality of automated machines. This document enables a consistent method of machine interconnection and operability. The diagrams and examples shown in the report are specific in terms of the functionality they provide but can be implemented in various ways to fit most automated machinery and machine controllers; therefore the figures do not follow ISO/IEC 19501:2005 for depiction of software flow.

If automated machinery is modelled in an ANSI/ISA-88.00.01 physical hierarchy, the example mapping shown in Figure 1 is possible. The example in this document will assume that a machine can represent the unit level in the ISA88 hierarchy.

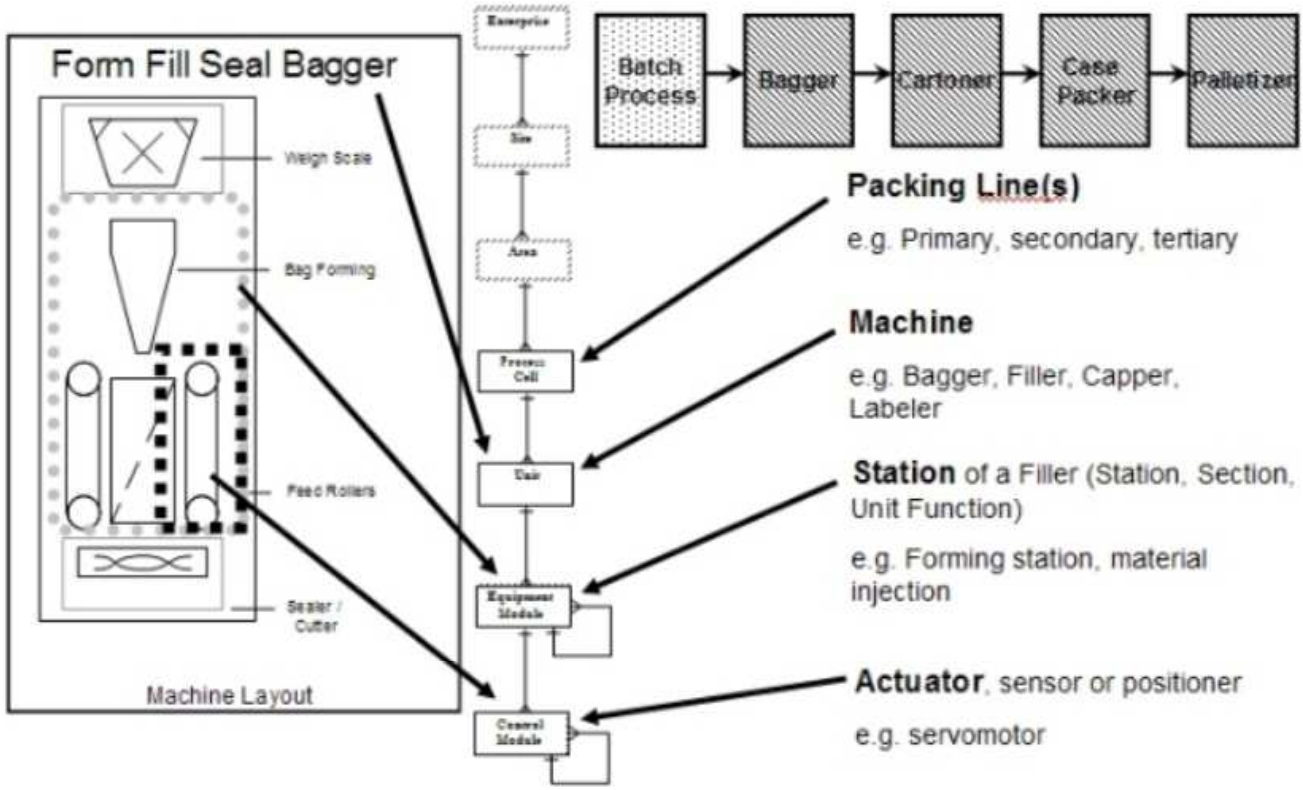


Figure 1: Automated machines applied to ANSI/ISA-88.00.01 physical model

Furthermore, the objective of this document is to provide a definition of machine (unit) modes and states, as well as state models corresponding to the different machine (unit) modes. In this example application of ANIS/ISA-88.00.01, the use of the state model and unit modes are extensible, but the methods governing the way in which the modes and states are used is not. This technical report demonstrates the flexibility and ease in which this method can be implemented in terms of ISA88, as well as how it provides the "common look and feel" desired in automated machines. This model only constrains the standard names and semantics for commonly used high level machine states as per a Base State Model.

The ANSI/ISA-88.00.01 standard describes example modes and states as applied to equipment entities and procedural elements. This report identifies unit/machine modes and states which should be considered an extension of the examples in the ANSI/ISA-88.00.01 standard in order to meet the needs of automated machine processing.

3.2 Personnel and environmental protection

The personnel and environmental protection control activity provides safety for people and the environment. No control activity should intervene between personnel and environmental protection and the field hardware it is designed to operate with. Personnel and environmental protection is, by definition, separate from the higher level control activities in this document. It may map to more than one software level of the equipment as desired.

A complete discussion of personnel and environmental protection, the classification of these types of systems, and the segregation of levels of interlocks within these systems is a topic of its own and beyond the scope of this document.

3.3 Control system compatibility

The minimum PackTag requirements that are identified in this document can be executed on a majority of PLC systems used by packaging machine manufacturers. It can also be implemented on systems such as check weighers, date coders, robotics, and other ancillary devices that may utilize proprietary control platforms. Even though the PackTags naming convention is using logical names, it can be implemented using register memory systems.

4 Unit/Machine states

4.1 Definition

A unit/machine state completely defines the current condition of a machine. A machine state is determined by an ordered procedure, or programming routine, that can consist of one or more commands to other procedural elements[1] or equipment entities, or be affected by the status of a procedural element[1] or equipment entity, or both. In performing the function specified by the state, the machine software will issue a set of commands to the machine procedural elements[1] or equipment entities which in turn can report status.

Only one major processing activity may be active in one machine at any time[2]. The linear sequence of major activities will drive a strictly sequentially ordered flow of control from one state to the next state – no more than one state of the base model is allowed to be active in one machine at the same time.

---

[1] Term "procedural element" defined (ANSI/ISA-88.00.01)

[2] A "major processing activity" corresponds to the term "equipment operation" as defined in ANSI/ISA-88.00.01.

The 2010 revision to this technical report added a minimum set of states in Figure 2. It also clarified the difference between holding and suspending in Table 1.

Note: At a lower level, the minor sub-activities (or control procedures) that are combined to form a major activity at the machine operation level, may indeed be taking place in parallel as well as in sequence as defined in ANSI/ISA-88.00.01 for equipment phases.

4.2 Types of states

For the purposes of understanding, two machine state types are defined:

- Acting state: A state which represents some processing activity. It implies the single or repeated execution of processing steps in a logical order, for a finite time or until a specific condition has been reached. In ANSI/ISA-88.00.01 these are referred to transient states, those states ending in "ING".
- Wait state: A state used to identify that a machine has achieved a defined set of conditions. In such a state, the machine is maintaining a status until transitioning to an acting state. In ANSI/ISA-88.00.01 this was referred to as a "final" or "quiescent" state.

4.3 Defined states

There are a fixed number of states defined in the base state model. This report establishes an example enumerated set of possible unit/machine states illustrated in the figure below, Figure 2. As shown, this set of states is a subset of the ANSI/ISA-88.00.01-2010 example states. Several states defined there are not supported for machine processing. Also see Figure 3.

Note: This table has been updated to show the "minimum required" states for a machine to be considered an appropriate PackML implementation.

ANSI/ISA-88.00.01 Procedural States	ISA-TR88.00.02 Equipment States				
	Value	Unit / Machine States	Wait	Acting	Minimum Required
<not defined>	1	Clearing		x	
Stopped	2	Stopped	x		x
<not defined>	3	Starting		x	
Idle	4	Idle	x		x
Paused	5	Suspended	x		
Running	6	Execute	x	x	x
Stopping	7	Stopping		x	
Aborting	8	Aborting		x	
Aborted	9	Aborted	x		x
Holding	10	Holding		x	
Held	11	Held	x		
Restarting	12	Unholding		x	
Pausing	13	Suspending		x	
<not defined>	14	Unsuspending		x	
<not defined>	15	Resetting		x	
<not defined>	16	Completing		x	
Complete	17	Complete	x		

Figure 2: Example states for automated machines

Formal definitions of these states are given below:

Table 1 : Complete list of machine dtates

State Name	Description
STOPPED	**State Type: Wait** The machine is powered and stationary after completing the STOPPING state. All communications with other systems are functioning (if applicable). A *RESET* command will cause an exit from STOPPED to the RESETTING state.
STARTING	**State Type: Acting** The machine completes the steps needed to start. This state is entered as a result of a *STARTING* command (local or remote). Following this command the machine will begin to "execute".
IDLE	**State Type: Wait** This is the state which indicates that RESETTING is complete. The machine will maintain the conditions which were achieved during the RESETTING state, and perform operations required when the machine is in IDLE.
SUSPENDING	**State Type: Acting** This state shall be used when EXTERNAL (outside this unit machine but usually on the same integrated production line) process conditions do not allow the machine to continue producing, that is, the machine leaves EXECUTE due to upstream or downstream conditions on the line. This is typically due to a blocked or starved event. This condition may be detected by a local machine sensor or based on a supervisory system external command. While in the SUSPENDING state, the machine is typically brought to a controlled stop and then transitions to SUSPENDED upon state complete. To be able to restart production correctly after the SUSPENDED state, all relevant process set-points and return status of the procedures at the time of receiving the *SUSPEND* command must be saved in the machine controller when executing the SUSPENDING procedure.
SUSPENDED	**State Type: Wait** Refer to SUSPENDING for when this state is used. In this state the machine shall not produce product. It will either stop running or continue to cycle without producing until external process conditions return to normal, at which time, the SUSPENDED state will transition to the UNSUSPENDING state, typically without any operator intervention.
UNSUSPENDING	**State Type: Acting** Refer to SUSPENDING for when this state is used. This state is a result of process conditions returning to normal. The UNSUSPENDING state initiates any required actions or sequences necessary to transition the machine from SUSPENDED back to EXECUTE. To be able to restart production correctly after the SUSPENDED state, all relevant process set-points and return status of the procedures at the time of receiving the SUSPEND command must be saved in the machine controller when executing the SUSPENDING procedure.
EXECUTE	**State Type: Acting** Once the machine is processing materials it is in the EXECUTE state. Different machine modes will result in specific types of EXECUTE activities. For example, if the machine is in the "Production" mode, the EXECUTE will result in products being produced, while in "Clean Out" mode the EXECUTE state refers to the action of cleaning the machine.

State Name	Description
STOPPING	**State Type: Acting** This state is entered in response to a *STOP* command. While in this state the machine executes the logic which brings it to a controlled stop as reflected by the STOPPED state. Normal STARTING of the machine cannot be initiated unless RESETTING had taken place.
ABORTING	**State Type: Acting** The ABORTING 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 safe stop.
ABORTED	**State Type: Wait** The machine maintains status information relevant to the ABORT condition. The machine can only exit the ABORTED state after an explicit *CLEAR* command, subsequently to manual intervention to correct and reset the detected machine faults.
HOLDING	**State Type: Acting** This state shall be used when INTERNAL (inside this unit machine and not from another machine on the production line) machine conditions do not allow the machine to continue producing, that is, the machine leaves EXECUTE due to internal conditions. This is typically used for routine machine conditions that requires minor operator servicing to continue production. This state can be initiated automatically or by an operator and can be easily recovered from. An example of this would be a machine that requires an operator to periodically refill a glue dispenser or carton magazine and due to the machine design, these operations cannot be performed while the machine is running. Since these types of tasks are normal production operations, it is not desirable to go through aborting or stopping sequences, and because these functions are integral to the machine they are not considered to be "external". While in the HOLDING state, the machine is typically brought to a controlled stop and then transitions to HELD upon state complete. To be able to restart production correctly after the HELD state, all relevant process set-points and return status of the procedures at the time of receiving the *HOLD* command must be saved in the machine controller when executing the HOLDING procedure.
HELD	**State Type: Wait** Refer to HOLDING for when this state is used. In this state the machine shall not produce product. It will either stop running or continue to dry cycle. A transition to the UNHOLDING state will occur when INTERNAL machine conditions change or an *UNHOLD* command is initiated by an operator.
UNHOLDING	**State Type: Acting** Refer to HOLDING for when this state is used. A machine will typically enter into UNHOLDING automatically when INTERNAL conditions, material levels, for example, return to an acceptable level. If an operator is required to perform minor servicing to replenish materials or make adjustments, then the *UNHOLD* command may be initiated by the operator.
COMPLETING	**State Type: Acting** This state is an automatic response from the EXECUTE state. Normal operation has run to completion, i.e. processing of material at the infeed will stop.
COMPLETE	**State Type: Wait** The machine has finished the COMPLETING state and is now waiting for a *RESET* command before transitioning to the RESETTING state.
RESETTING	**State Type: Acting** This state is the result of a *RESET* command from the STOPPED or COMPLETE state. Faults and stop causes are reset. RESETTING will typically cause safety devices to be energized and place the machine in the IDLE state where it will wait for a *START* command. No hazardous motion should happen in this state.

State Name	Description
CLEARING	**State Type: Acting** Initiated by a state command to clear faults that may have occurred when ABORTING, and are present in the ABORTED state before proceeding to a STOPPED state.

4.4 State transitions and state commands

4.4.1 Definition

A state transition is defined as a passage from one state to another. Transitions between states will occur as a result of a local, remote, or procedural state command. State commands are procedural elements that in effect cause a state transition to occur.

4.4.2 7.5.2 Types of state commands

State commands are comprised of one or a combination of the following types:

- Operator intervention.
- Response to the status of one or more procedural elements.
- Response to machine conditions.
- The completion of an acting state procedure.
- Supervisory or remote system intervention.

4.4.3 Examples of state transitions

An example transition matrix for local or remote state commands generated by an operator is shown in Table 2. After every acting state, as can be seen in Table 2, a procedural element is required that will indicate the acting state is complete, or a command is required to stop or abort the acting state. The state complete indication within the acting state procedure will cause a state transition to occur.

Similarly, an example state command matrix of machine conditions activating a state command is shown in Table 3. The objective of this table is to depict the machine conditions that will cause a state transition using the commands defined in Table 2.

Table 2 : Example transition matrix of local or remote state commands

	State Commands									State Complete
Current State	Start	Reset[1]	Hold	Un-Hold	Suspend	UnSuspend	Clear[1]	Stop[2]	Abort[2]	
IDLE	STARTING							STOPPING	ABORTING	
STARTING								STOPPING	ABORTING	EXECUTE
EXECUTE			HOLDING		SUSPENDING			STOPPING	ABORTING	COMPLETING
COMPLETING								STOPPING	ABORTING	COMPLETE
COMPLETE		RESETTING						STOPPING	ABORTING	
RESETTING								STOPPING	ABORTING	IDLE
HOLDING								STOPPING	ABORTING	HELD
HELD				UNHOLDING				STOPPING	ABORTING	
UNHOLDING								STOPPING	ABORTING	EXECUTE
SUSPENDING								STOPPING	ABORTING	SUSPENDED
SUSPENDED						UNSUSPENDING		STOPPING	ABORTING	
UNSUSPENDING								STOPPING	ABORTING	EXECUTE
STOPPING									ABORTING	STOPPED
STOPPED		RESETTING							ABORTING	
ABORTING										ABORTED
ABORTED							CLEARING			
CLEARING									ABORTING	STOPPED

[1] It is common practice for clearing and resetting COMMANDS to be initiated using the same physical operator interface device.

[2] It is common practice in packaging (but not process) applications to permit use of STOP and ABORT commands while in the IDLE, COMPLETE, STOPPED, and RESETTING states.

Table 3: Example matrix of machine conditions initiating a state command

	Example Machine Conditions										
Current State	**Operator Start**	**Carton Magazine Low**	**Carton Magazine Full**	**Downstream Not Ready**	**Downstream Ready**	**E-Stop or Fault**	**No Product Present**	**Product Present**	**Operator Stop**	**Product Count Reached**	**Clear Faults**
IDLE	Start					Abort			Stop		
STARTING						Abort			Stop		
EXECUTE		Hold		Suspend		Abort	Suspend		Stop	Complete	
COMPLETING						Abort			Stop		
COMPLETE						Abort			Stop		
RESETTING						Abort			Stop		
HOLDING						Abort			Stop		
HELD			UnHold			Abort			Stop		
UNHOLDING						Abort			Stop		
SUSPENDING						Abort			Stop		
SUSPENDED					UnSuspend	Abort		UnSuspending	Stop		
UNSUSPENDING						Abort			Stop		
STOPPING						Abort					
STOPPED						Abort					
ABORTING											
ABORTED											Clear
CLEARING						Abort					

4.5 State model

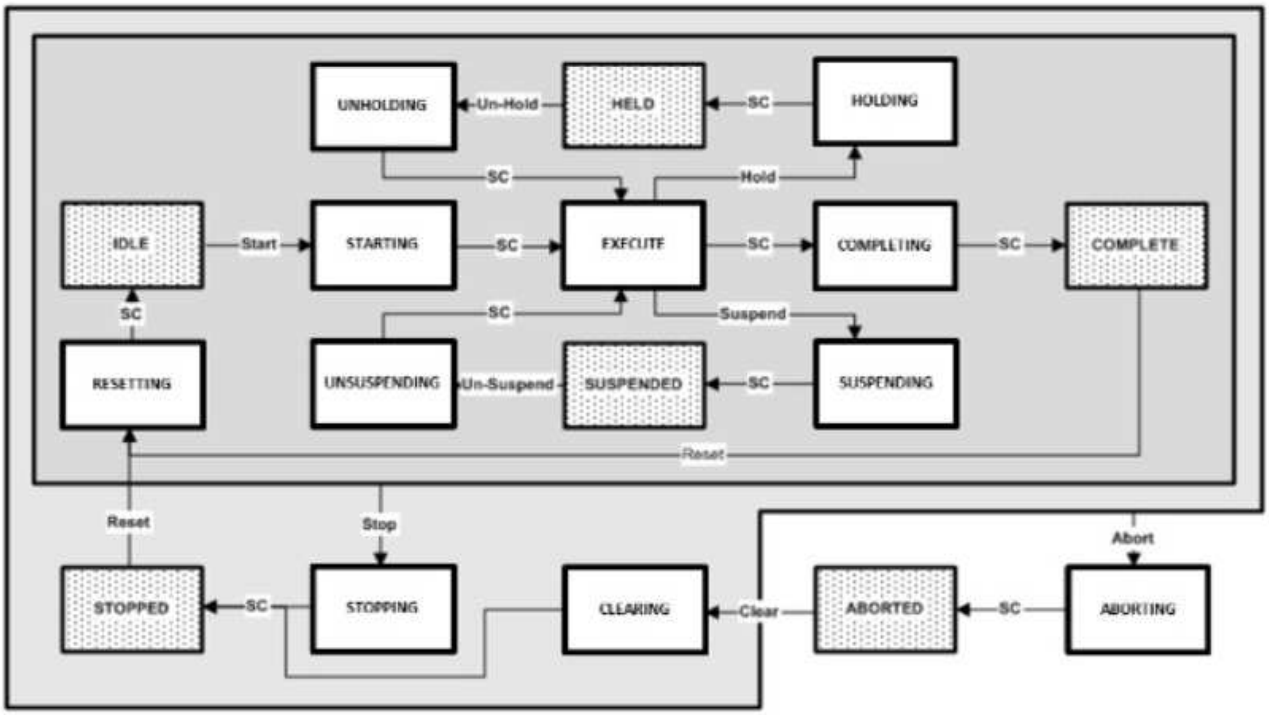
A state model completely defines the behaviour of a machine in one unit control mode. In a state model, states are arranged in an ordered fashion that is consistent with the specified operation of the machine. The specific states required are dependent on the machine operation.

The compilation of all defined functional states and their transitions is called the base state model.

4.5.1 Base state model

The base state model represents the complete set of defined states, state commands, and state transitions. All unit control modes will be defined by subsets of the base state model. The base state model is depicted in Figure 3. This model is a subset of the state model from ANSI/ISA-88.00.01-2010 (Annex D Figure 29) with three distinctions:

a) The paused or pausing states are not used
b) The label "execute" is used for the running state
c) Transition to holding directly from suspending, suspend or unsuspending is not supported



SC = State Complete

IDLE = Wait State

EXECUTE = Acting State

States within the dark gray outline can transition to the STOPPING state or the ABORTING state.
States within the light gray outlined can transition to ABORTING state.
The minimum states are identified in Figure 2.

Figure 3: Base state model visualization

5 Modes

The ANSI/ISA-88.00.01 standard provides a set of modes for equipment entities and procedural elements. This report establishes modes for automated machines that are considered different than the ANSI/ISA-88.00.01 procedural modes (automatic, semi-automatic, and manual). The ANSI/ISA-88.00.01 procedural modes describe the way procedures operate. Procedural modes are not common in automated machinery. They are provided for in this report by noting they may exist, but are not considered in the scope of this work and are not included in the tag table.

For automated machines, the example in this report establishes unit/machine modes in order to allow a machine designer to adjust the set of states, state commands and state transitions a machine may follow given different operating circumstances. The set of defined unit/machine control modes is shown in Figure 4 and described below.

The 2013 revision to this document changed producing mode to production mode.

ANSI/ISA-88.00.01 Example Modes	ISA-TR88.00.02 Control Modes	
	Value	Unit / Machine Control Modes
<not defined>	0	Invalid
<not defined>	1	Production
<not defined>	2	Maintenance
<not defined>	3	Manual
<not defined>	04 – 31	User Definable

Figure 4: Unit/Machine control modes

5.1 Unit/Machine control modes

A unit/mMachine control mode determines the subset of states, state commands, and state transitions that determine the strategy for carrying out a unit/machine's process.

Typical unit/machine control modes are production, maintenance, manual, clean in place, run out, semi-auto, dry cycle, etc. The distinguishing elements between these unit control modes are the selected subset of states, state commands, and state transitions.

The ordered procedures that control the states will be unique for the unit/machine control mode that the unit/machine is in. For example, in a "production" unit/machine control mode the definition of "execute" in a filling machine will mean it is "producing" product. In the "manual" unit/machine control mode the definition of the "executing" state may be jogging or indexing. The "execute" state defines the functional operation of the unit/machine control mode. States of identical names may have different functions in different unit/machine control modes

Examples of unit/machine control modes are:

Production mode - This represents the mode which is utilized 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.

Maintenance mode - This mode may allow suitably authorized personnel the ability to run an individual machine independent of other machines in a production line. This mode would typically be used for fault finding, machine trials or testing operational improvements. This mode would also allow the speed of the machine to be adjusted (where this feature is available).

Manual mode - This provides direct control of individual machine modules. This feature is available depending upon the mechanical constraints of the mechanisms being exercised. This feature may be used for the commissioning of individual drives, verifying the operation of synchronized drives, testing the drive as a result of modifying parameters etc.

5.2 Unit/Machine control mode management

Automated machinery has unit/machine control modes other than “production”, as noted earlier. Each unit/machine control mode has its own state model. In order to manage the change from one mode to the next, a method of mode management must be defined. The mode management method determines how, and in what state a machine may change unit/machine control modes; i.e. the mode management method includes interlocks that prevent the machine changing unit/machine control modes when in inappropriate states.

Unit/machine control mode management enables the machine designer to manage unit/machine control mode transitions. Specification on transitions between unit/machine control modes is left to the user, but typical transition points are at *wait* states. The specification of the unit/machine control mode manager is such that no state or control functions are carried out in this upper level routine. The intent of the mode manager is to logically supervise when a change in mode can be done, command a mode change, and report status of the change request. All considerations of a mode manager must be consistent with prevailing safe practices and standards.

Transitions between unit/machine control modes can occur at only pre-programmed states:

- As a result of a local or remote operator command.
- As a result of a remote request from another automated unit.
- As a result of a state 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 method. For example, if a filling machine has completed its production run in “PRODUCTION” mode of a given number of cases it may change to the “CLEANING” mode to begin a clean cycle.
- When a unit/machine control mode change takes place, a machine state change is NOT allowed to occur at the same time. Mode transitions must take place in a state that is common to both modes. This is necessary to avoid unintended machine sequences from taking place.

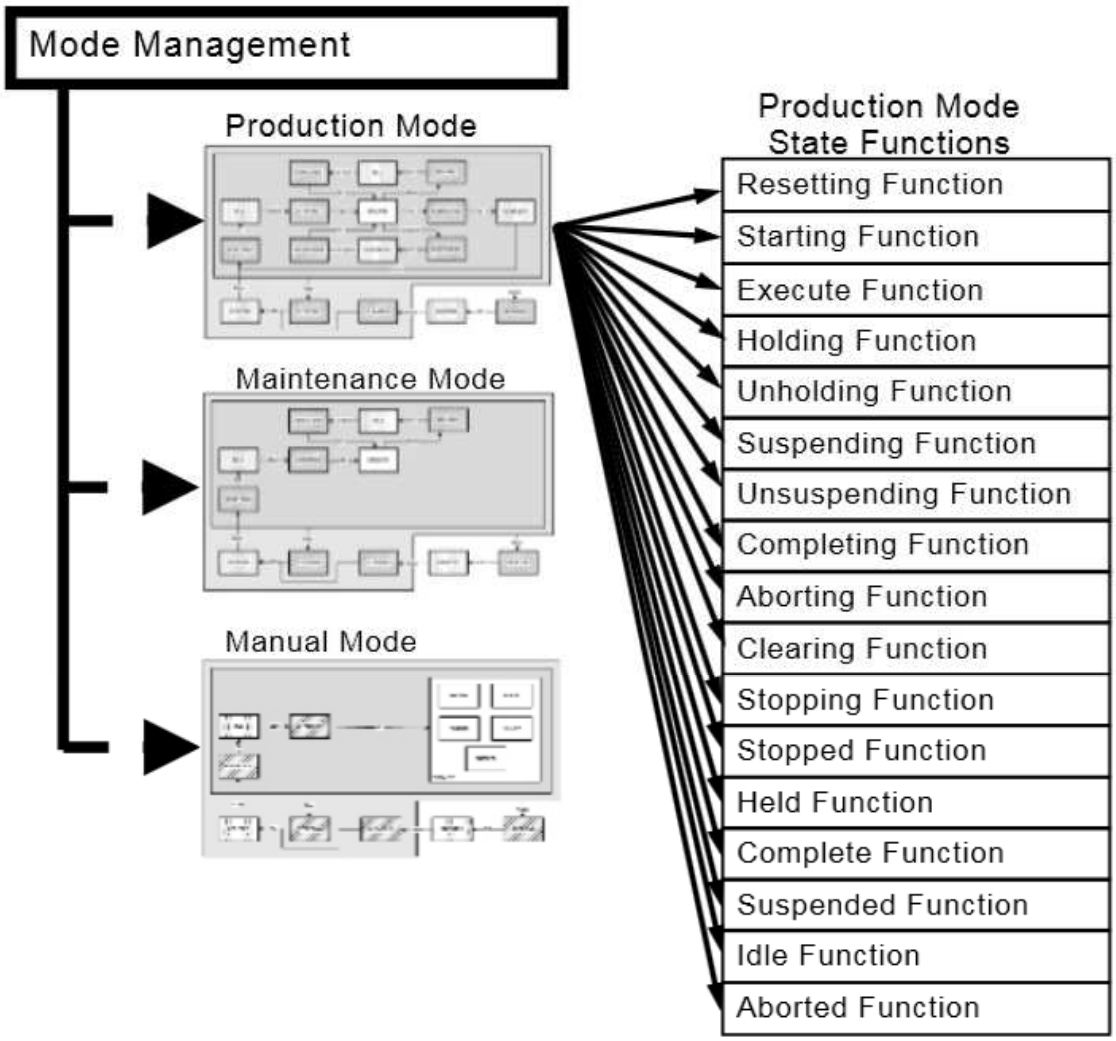


Figure 5: Multi-mode example diagram (production mode states)

6 Common unit/machine mode examples

6.1 Production mode

The base state model above can be used to define a production mode that is used in order to deliver control of routine processing and production. It is recognized that machines also require maintenance, calibration and setting up. To address this requirement two example modes of operation are shown below: maintenance and manual. Because there can be any number of possible modes for an automated machine, a user mode example is also shown. The user mode example is based on the Weihenstephan PDA (production data acquisition) standard.

6.2 Maintenance mode

Maintenance mode allows suitably authorized personnel the ability to run an individual machine independent of other machines in a production line. This would typically be used for fault finding, machine trials or testing operational improvements. It is expected that, because the machine will generally operate in its usual manner, it will need to undergo some or all of its routine starting up procedures. Maintenance mode will follow a recognized unit state model.

By way of example, one possible maintenance mode state model is shown in Figure 6 below. It is recognized that individual machine manufacturers may have good reason to develop other versions of maintenance mode state models. Typically modes, such as maintenance mode are developed as containing a subset of the unit states in the base state model. The unit state names remain consistent but the function of the states has been modified to be consistent with the mode function.

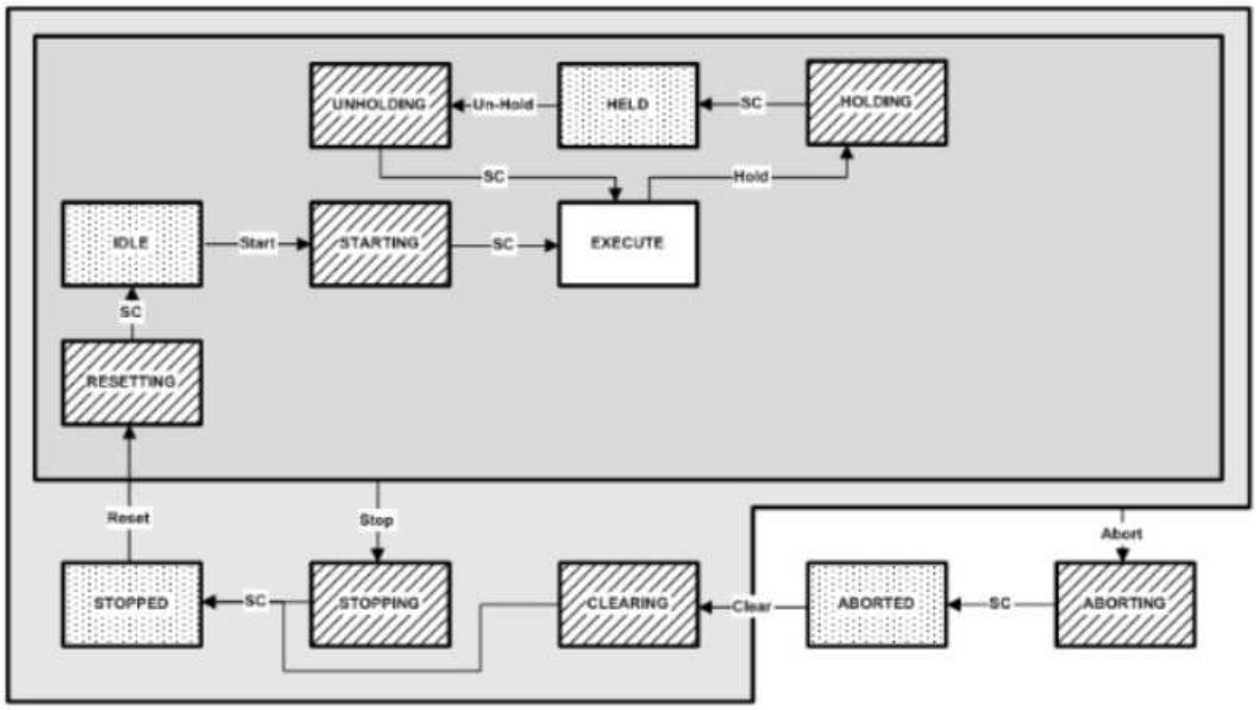


Figure 6: Maintenance mode state model

As can be seen above, the unit state model proposed for maintenance operations is a subset of the previously defined base state model. The essential difference between the base atate model and maintenance mode state model is the absence of a SUSPENDED state in maintenance mode. It is envisaged for certain line types that the SUSPENDED state is not required, as its function is to provide for a wait state for incoming material. In this example maintenance mode is not designed for routine production and hence no SUSPENDED state is available. The function of EXECUTE

state has also taken on new meaning, in that EXECUTING production may not require the same logic as EXECUTING in maintenance.

In the figure below, the maintenance mode execution model is shown. Only unit state functions represented in the state model for the maintenance mode will be executed. The maintenance mode state functions are not necessarily the same functions as those in other modes, even though the unit states are named the same – they may be uniquely referenced for the mode they are associated with. Programmatically, the functions for each unit control mode are executed only when the respective unit control mode has been chosen by the "mode management" routine.

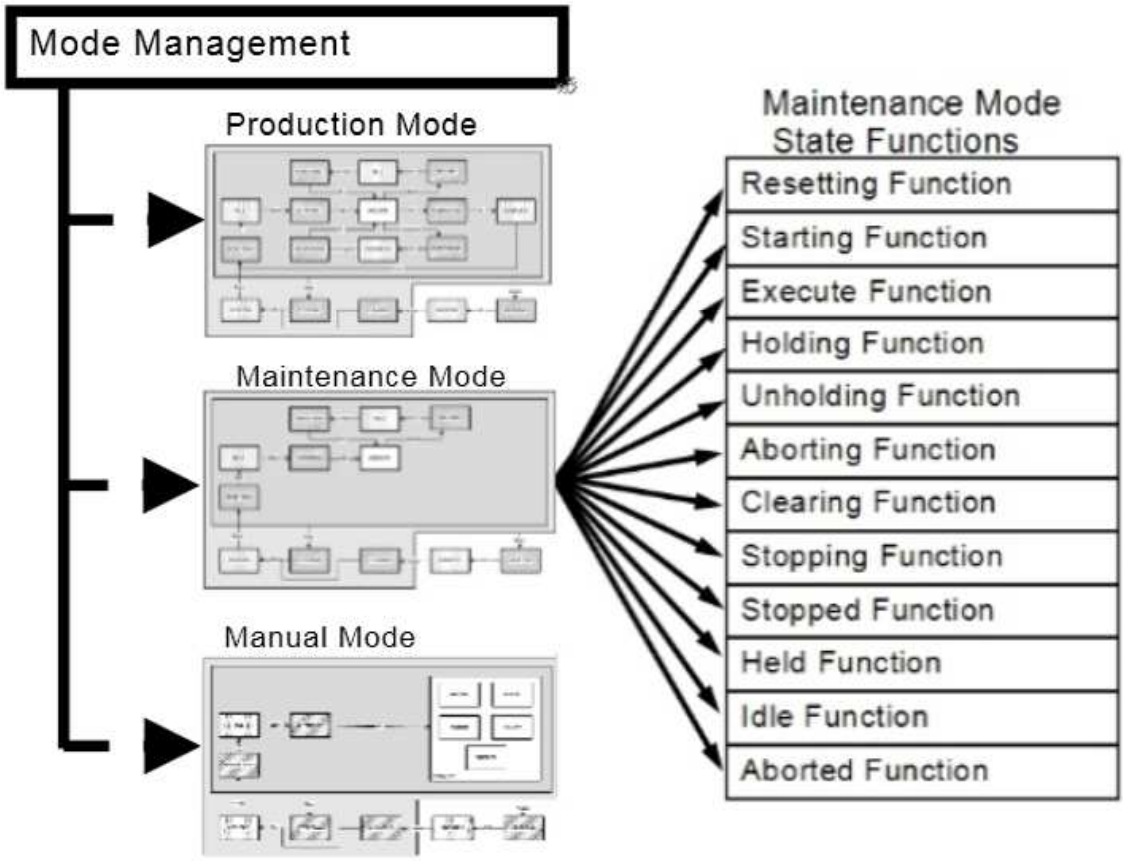


Figure 7: Maintenance mode execution model

6.3 Manual mode

Manual mode provides suitably authorized personnel the ability to operate individual subordinate equipment controls (such as drive logic) within the machine under manual pushbutton control. Such controls in this mode may be on a "hold-to-run" basis such that removal of the run signal will cause the drive to be stopped. The ability to perform specific functions will be dependent upon mechanical constraints and interlocks. Manual mode will be of particular use for setting up the machine to work.

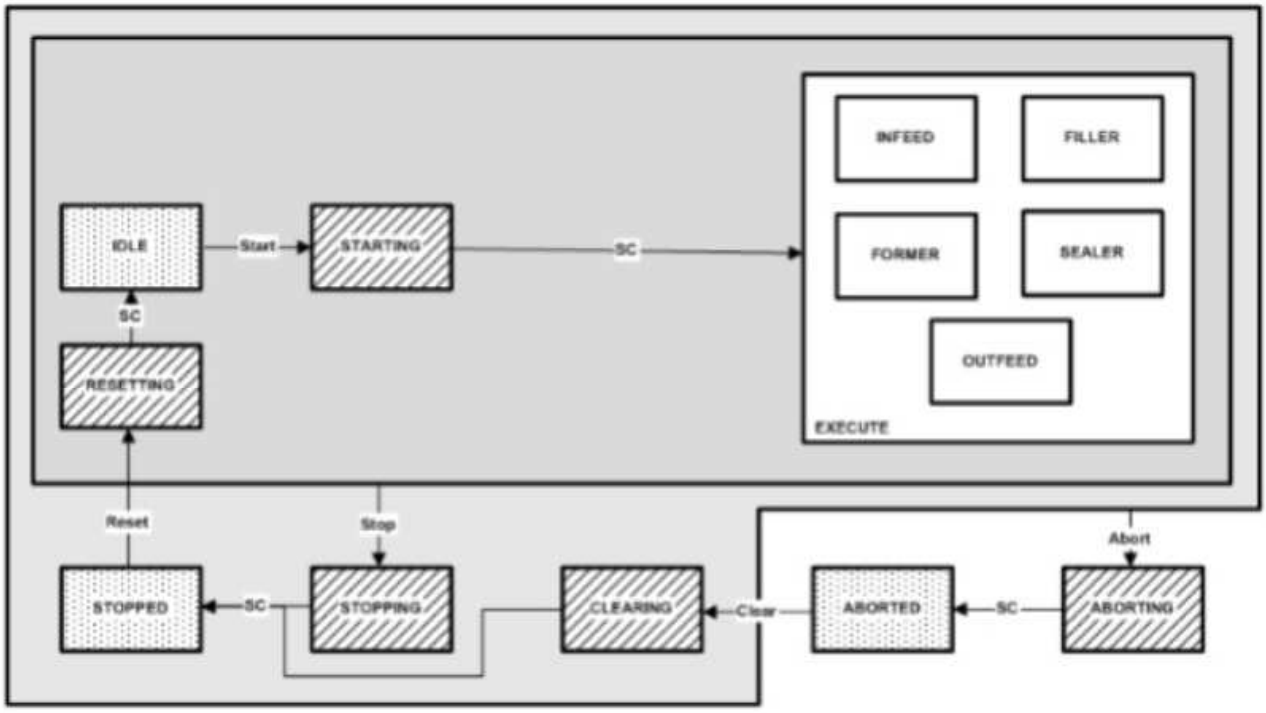


Figure 8: Manual mode state model

The predefined unit state model associated with manual mode can again be defined as a subset from the base state model. Common synonyms for this unit control mode are inch, jog, or index. Figure 8 illustrates a manual mode in which the EXECUTE states of subordinate equipment controls (such as drive logic) are shown within the EXECUTE state.

In Figure 9, the manual mode execution model is shown. Only unit state functions represented in the state model for the manual mode will be executed. The manual mode state functions are not necessarily the same functions as those in other modes, even though the unit states are named the same – they may be uniquely referenced for the mode they are associated with. Programmatically, the functions for each mode are executed only when the respective unit control mode has been chosen by the “mode management” routine.

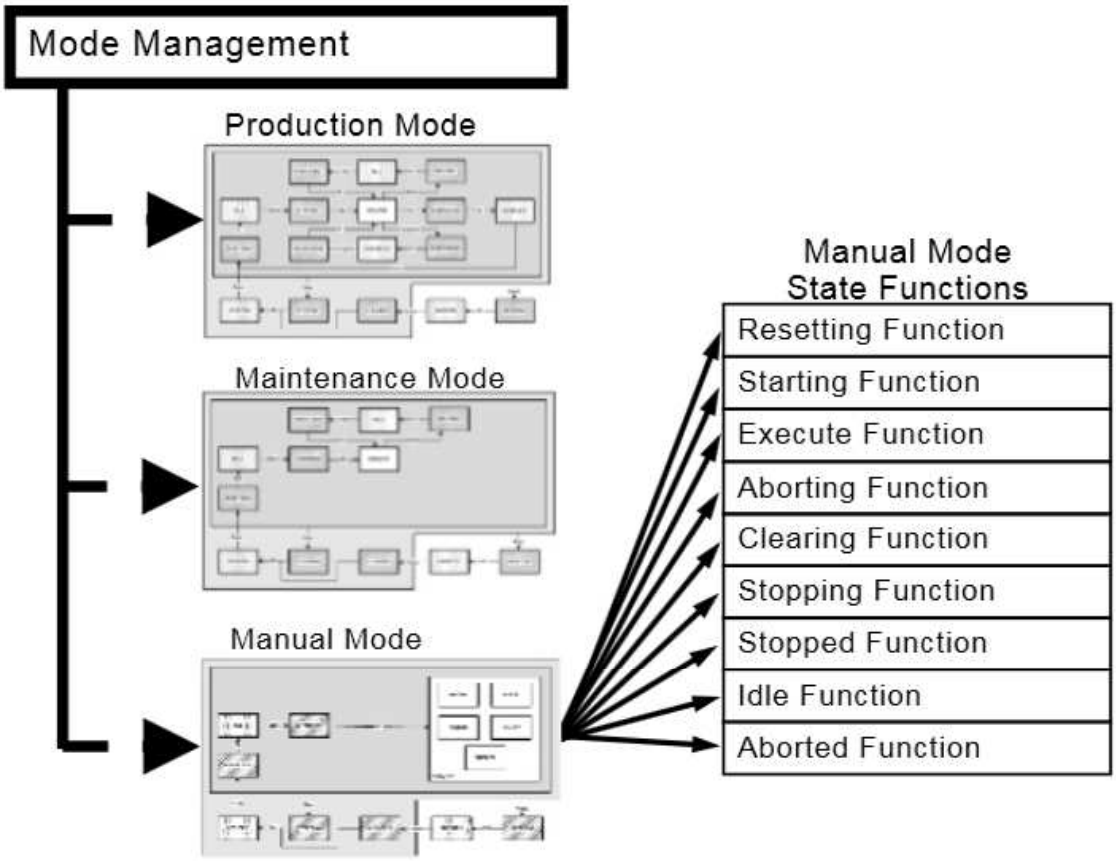
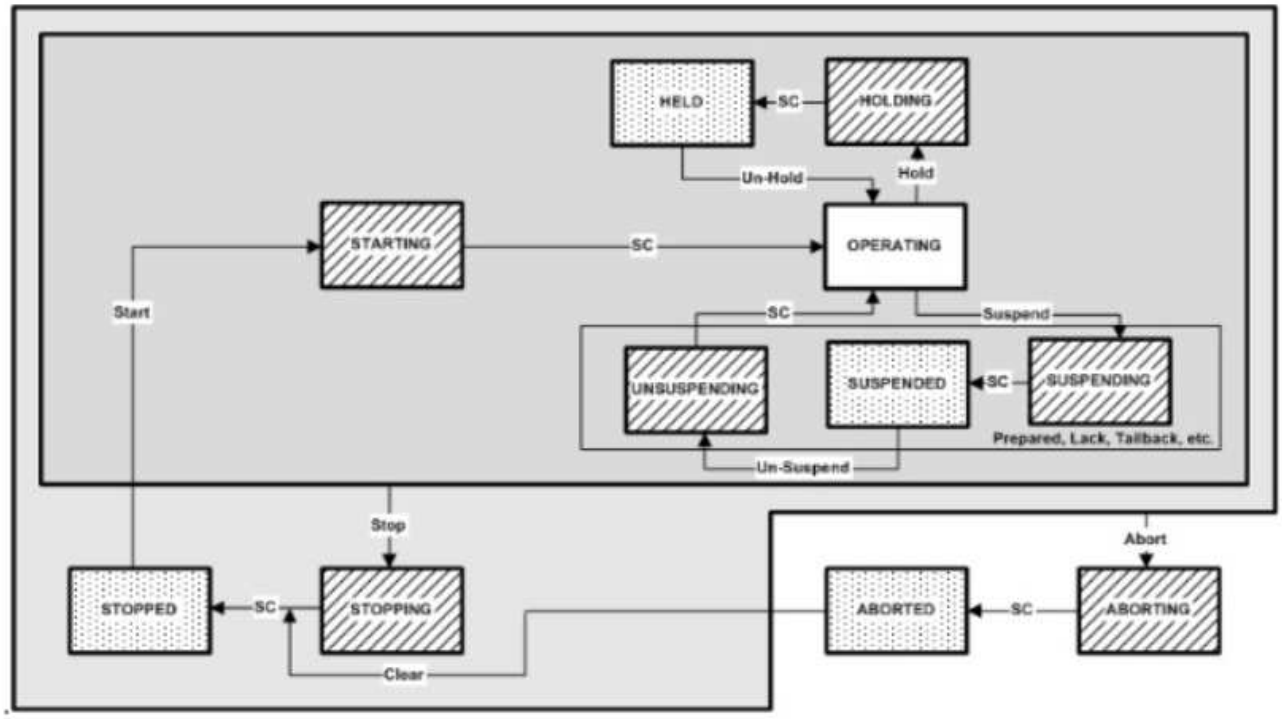


Figure 9: Manual mode execution model

6.4 User definable modes

Any unit control mode can be defined which provides a required function for the machine. The unit control mode provides for suitably authorized personnel operating the machine under pushbutton control, or for a remote system operating the machine as part of an integrated work center. This report recommends the approach in which all unit control modes are based on a fixed set of enumerated machine states. The "name" of the state(s) may be customized to provide the operator with an intuitive or descriptive name for the state(s), but the function of the state(s) is consistent with, and a subset of, the general definition of the base state model.

Below is a depiction of the Weihenstephan standard harmonized to the base state model in this report. In this user mode example the EXECUTE state is renamed the OPERATING state to be consistent with the terminology used in the Weihenstephan model. As can be seen, the base state model included states that were collapsed to be consistent with the Weihenstephan standard. The state commands are not defined as causing individual or undefined states; they are commands to implementation logic that describes the corresponding state transitions. There may be multiple conditions that cause a state transition but they do not cause "unique" machine states, they cause implementation specifics within the given framework of given states.



Note: OPERATING state is equivalent to an EXECUTE state; "prepared", "lack", and "tailback" conditions provide for the machine to go into the SUSPENDING state by activating the suspend command.

Figure 10: Automatic Weihenstephan state model

7 Automated machine functional tag description

7.1 Introduction to PackTags

PackTags provide a uniform set of naming conventions for data elements used within the procedural elements of the base state model. As seen earlier in the document the base state model provides a uniform set of machine states, so that all automated machinery can be looked at in a common way. PackTags are named data elements used for open architecture, interoperable data exchange in automated machinery. This document includes the fundamental names of the data elements as well as the data type, values, ranges and where necessary, data structures. PackTags are useful for machine-to-machine (intermachine) communications; for example between a filler and a capper. PackTags can also be used for data exchange between machines and higher-level information systems like manufacturing operations management and enterprise information systems.

This report defines all the PackTags necessary to navigate through a state model, as well as those that are required to define and manipulate the unit control mode. This report also defines a list of PackTags that will provide necessary information that might be available from a machine. The use of all PackTags is needed to be consistent with the principles for integrated connectivity with systems using this same implementation method. [3]

The 2013 revision to this document added a minimum set of PackTags, added blocked and starved tags, specified date and time format, and added stop reason and warnings.

7.2 Tag types

PackTags are broken out into three groups: command, status and administration. Command and status tags contain data required for interfacing between machines and line control for coordination, or for recipe/parameter download. Command tags are "written" to and consumed by the machine program, as the “information receiver”, while status tags are produced by and read from the machine program. Administration tags contain data collected by higher level systems for machine performance analysis, or operator information.[4] Generally informational data is passed using OPC on an Ethernet-based communication network.

- Command tags are prefixed by “Command”.
- Status tags are prefixed by “Status”.
- Administration tags are prefixed by “Admin”.

---

[3] Required tags are those necessary for the function of the automated machine or the connectivity to supervisory or remote systems.

[4] Each grouping of data should be in a contiguous grouping of registers to optimise communications.

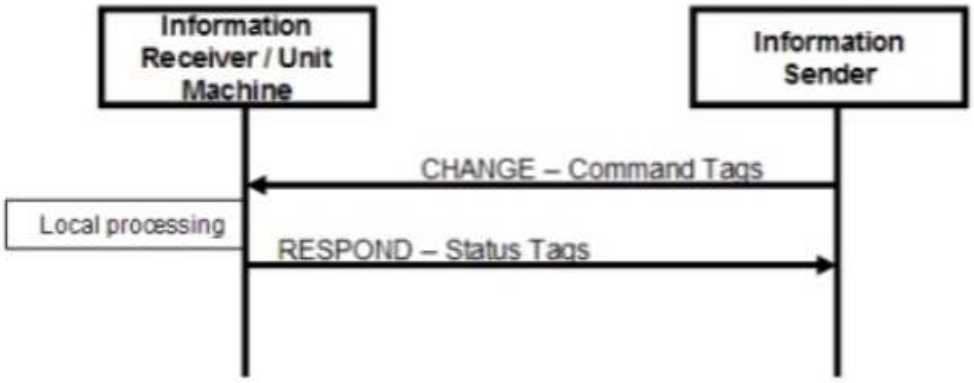


Figure 11: Tag information flow

7.3 PackTags names

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. Optionally, underscores may also be used in place of the "dot" notation for legacy systems that do not support structured tagnames.

Thus, the exact text strings that should be used as tag names should be as follows:

Status.StateCurrent
Status.UnitModeCurrent

7.4 Data types, units, and ranges

The following are the typical data types used for the tags.

- Integer – 32 bit, signed 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)
- Boolean structure – Bit pattern
- String – null-terminated ASCII, any size but not to exceed 80 characters
- Date and time – Data type: INT (32 bit) array
    - Array element 0 = Year
    - Array element 1 = Month
    - Array element 2 = Day
    - Array element 3 = Hour (24hr format)
    - Array element 4 = Min
    - Array element 5 = Sec
    - Array element 6 = USec (1/1,000,000 sec)
- Time display format – ISO 8601:1988 24hr time data type, beginning at 00:00:00.
- Date display format – ISO 8601:1988 date data type YYYY-MM-DD

7.4.1 Structured data types

- PACKMLV30 – is a placeholder for the machine unit name, and is the top level in the PackTag structure.
- PMLc – is the collection of all command tags in the PackTag structure.
- PMLs – is the collection of all status tags in the PackTag structure.
- PMLa – is the collection of all administration tags in the PackTag structure.

- Interface – is a collection of tags that are used to describe communication command values between machines using the PackTag structure.
- Descriptor structure – is a collection of tags that are used to describe parameters in the machine unit.
- Product structure – is a collection of tags used to describe the product that the machine is making.
- Ingredient – is a collection of tags used to describe the raw materials that are needed for the product.
- Alarm – is the collection tags needed to describe alarm events.

7.5 Tag details

The following section is a summary listing of the tags. Tables 4, 5 and 6 list the command, admin and status PackTags. Tables 7 and 8 capture the minimum set of tags to be consistent with the technical report. Tag definitions are detailed below:

Table 4: Command tags (complete listing)

MIN REQ						TAGNAME	DATA TYPE
UnitName						UnitName	PackMLv30
	Command					UnitName.Command	PMLc
x		UnitMode				UnitName.Command.UnitMode	Int (32bit)
x		UnitModeChangeRequest				UnitName.Command.UnitModeChangeRequest	Bool
x		MachSpeed				UnitName.Command.MachSpeed	Real
		MaterialInterlock				UnitName.Command.MaterialInterlock	Bool Structure
x		CntrlCmd				UnitName.Command.CntrlCmd	Int (32bit)
x		CmdChangeRequest				UnitName.Command.CmdChangeRequest	Bool
		RemoteInterface[#]				UnitName.Command.RemoteInterface[#]	Interface
			Number			UnitName.Command.RemoteInterface[#].Number	Int (32bit)
			ControlCmdNumber			UnitName.Command.RemoteInterface[#].ControlCmdNumber	Int (32bit)
			CmdValue			UnitName.Command.RemoteInterface[#].CmdValue	Int (32bit)
			Parameter[#]			UnitName.Command.RemoteInterface[#].Parameter[#]	Descriptor Structure
				ID		UnitName.Command.RemoteInterface[#].Parameter[#].ID	Int (32bit)
				Name		UnitName.Command.RemoteInterface[#].Parameter[#].Name	String
				Unit		UnitName.Command.RemoteInterface[#].Parameter[#].Unit	String
				Value		UnitName.Command.RemoteInterface[#].Parameter[#].Value	Real
		Parameter[#]				UnitName.Command.Parameter[#]	Descriptor Structure
			ID			UnitName.Command.Parameter[#].ID	Int (32bit)
			Name			UnitName.Command.Parameter[#].Name	String
			Unit			UnitName.Command.Parameter[#].Unit	String
			Value			UnitName.Command.Parameter[#].Value	User Defined
		Product[#]				UnitName.Command.Product[#]	Product Structure
			ProductID			UnitName.Command.Product[#].ProductID	Int (32bit)
			ProcessVariables[#]			UnitName.Command.Product[#].ProcessVariables[#]	Descriptor Structure

MIN REQ						TAGNAME	DATA TYPE
				ID		UnitName.Command.Product[#].ProcessVariables[#].ID	Int (32bit)
				Name		UnitName.Command.Product[#].ProcessVariables[#].Name	String
				Unit		UnitName.Command.Product[#].ProcessVariables[#].Unit	String
				Value		UnitName.Command.Product[#].ProcessVariables[#].Value	Real
			Ingredients[#]			UnitName.Command.Product[#].Ingredients[#]	Ingredient
				IngredientID		UnitName.Command.Product[#].Ingredients[#].IngredientID	Int (32bit)
				Parameter[#]		UnitName.Command.Product[#].Ingredients[#].Parameter[#]	Descriptor Structure
					ID	UnitName.Command.Product[#].Ingredients[#].Parameter[#].ID	Int (32bit)
					Name	UnitName.Command.Product[#].Ingredients[#].Parameter[#].Name	String
					Unit	UnitName.Command.Product[#].Ingredients[#].Parameter[#].Unit	String
					Value	UnitName.Command.Product[#].Ingredients[#].Parameter[#].Value	Real

Table 5: Status tags (complete listing)

MIN REQ						TAGNAME	DATATYPE
UnitName						UnitName	PackMLv30
	Status					UnitName.Status	PMLs
x		UnitModeCurrent				UnitName.Status.UnitModeCurrent	Int (32bit)
		UnitModeRequested				UnitName.Status.UnitModeRequested	Bool
		UnitModeChangeInProcess				UnitName.Status.UnitModeChangeInProcess	Bool
x		StateCurrent				UnitName.Status.StateCurrent	Int (32bit)
		StateRequested				UnitName.Status.StateRequested	Int (32bit)
		StateChangeInProcess				UnitName.Status.StateChangeInProcess	Bool
x		MachSpeed				UnitName.Status.MachSpeed	Real
x		CurMachSpeed				UnitName.Status.CurMachSpeed	Real
		MaterialInterlock[#]				UnitName.Status.MaterialInterlock	Bool Array [32]
		EquipmentInterlock				UnitName.Status.EquipmentInterlock	Bool Structure [2]
x			Blocked			UnitName.Status.EquipmentInterlock.Blocked	Bool
x			Starved			UnitName.Status.EquipmentInterlock.Starved	Bool
		RemoteInterface[#]				UnitName.Status.RemoteInterface[#]	Interface
			Number			UnitName.Status.RemoteInterface[#].Number	Int (32bit)
			ControlCmdNumber			UnitName.Status.RemoteInterface[#].ControlCmdNumber	Int (32bit)
			CmdValue			UnitName.Status.RemoteInterface[#].CmdValue	Int (32bit)
			Parameter[#]			UnitName.Status.RemoteInterface[#].Parameter[#]	Descriptor Structure
				ID		UnitName.Status.RemoteInterface[#].Parameter[#].ID	Int (32bit)
				Name		UnitName.Status.RemoteInterface[#].Parameter[#].Name.	String
				Unit		UnitName.Status.RemoteInterface[#].Parameter[#].Unit	String
				Value		UnitName.Status.RemoteInterface[#].Parameter[#].Value	Real
		Parameter[#]				UnitName.Status.Parameter[#]	Descriptor Structure
			ID			UnitName.Status.Parameter[#].ID	Int (32bit)
			Name			UnitName.Status.Parameter[#].Name	String

MIN REQ						**TAGNAME**	**DATATYPE**
			Unit			UnitName.Status.Parameter[#].Unit	String
			Value			UnitName.Status.Parameter[#].Value	User Defined
		Product[#]				UnitName.Status.Product[#]	Product Structure
			ProductID			UnitName.Status.Product[#].ProductID	Int (32bit)
			ProcessVariables[#]			UnitName.Status.Product[#].ProcessVariables[#]	Descriptor Structure
				ID		UnitName.Status.Product[#].ProcessVariables[#].ID	Int (32bit)
				Name		UnitName.Status.Product[#].ProcessVariables[#].Name	String
				Unit		UnitName.Status.Product[#].ProcessVariables[#].Unit	String
				Value		UnitName.Status.Product[#].ProcessVariables[#].Value	Real
			Ingredients[#]			UnitName.Status.Product[#].Ingredients[#]	Ingredient
				IngredientID		UnitName.Status.Product[#].Ingredients[#].IngredientID	Int (32bit)
				Parameter[#]		UnitName.Status.Product[#].Ingredients[#].Parameter[#]	Descriptor Structure
					ID	UnitName.Status.Product[#].Ingredients[#].Parameter[#].ID	Int (32bit)
					Name	UnitName.Status.Product[#].Ingredients[#].Parameter[#].Name	String
					Unit	UnitName.Status.Product[#].Ingredients[#].Parameter[#].Unit	String
					Value	UnitName.Status.Product[#].Ingredients[#].Parameter[#].Value	Real

Table 6: Admin tags (complete listing)

MIN REQ						TAGNAME	DATATYPE
UnitName						UnitName	PackMLv30
	Admin					UnitName.Admin	PMLa
		Parameter[#]				UnitName.Admin.Parameter[#]	Descriptor Structure
			ID			UnitName.Admin.Parameter[#].ID	Int (32bit)
			Name			UnitName.Admin.Parameter[#].Name	String
			Unit			UnitName.Admin.Parameter[#].Unit	String
			Value			UnitName.Admin.Parameter[#].Value	Real
		Alarm[#]				UnitName.Admin.Alarm[#]	Alarm Structure
			Trigger			UnitName.Admin.Alarm[#].Trigger	Bool
			ID			UnitName.Admin.Alarm[#].ID	Int (32bit)
			Value			UnitName.Admin.Alarm[#].Value	Int (32bit)
			Message			UnitName.Admin.Alarm[#].Message	String
			Category			UnitName.Admin.Alarm[#].Category (Event Grouping)	Int (32bit)
			DateTime			UnitName.Admin.Alarm[#]	Date-Time Array
				[0] (year)		UnitName.Admin.Alarm[#].DateTime[0]	Int (32bit)
				[1] (month)		UnitName.Admin.Alarm[#].DateTime[1]	Int (32bit)
				[2] (day)		UnitName.Admin.Alarm[#].DateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.Alarm[#].DateTime[3]	Int (32bit)
				[4] (min)		UnitName.Admin.Alarm[#].DateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.Alarm[#].DateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.Alarm[#].DateTime[6]	Int (32bit)
			AckDateTime			UnitName.Admin.Alarm[#]	Date-Time Array
				[0] (year)		UnitName.Admin.Alarm[#].AckDateTime[0]	Int (32bit)
				[1] (month)		UnitName.Admin.Alarm[#].AckDateTime[1]	Int (32bit)
				[2] (day)		UnitName.Admin.Alarm[#].AckDateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.Alarm[#].AckDateTime[3]	Int (32bit)

MIN REQ						TAGNAME	DATATYPE
				[4] (min)		UnitName.Admin.Alarm[#].AckDateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.Alarm[#].AckDateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.Alarm[#].AckDateTime[6]	Int (32bit)
		AlarmExtent				UnitName.Admin.AlarmExtent	Int(32bit)
		AlarmHistory[#]				UnitName.Admin.AlarmHistory[#]	Alarm Structure
			Trigger			UnitName.Admin.AlarmHistory[#].Trigger	Bool
			ID			UnitName.Admin.AlarmHistory[#].ID	Int (32bit)
			Value			UnitName.Admin.AlarmHistory[#].Value	Int (32bit)
			Message			UnitName.Admin.AlarmHistory[#].Message	String
			Category			UnitName.Admin.AlarmHistory[#].Category (Event Grouping)	Int (32bit)
			DateTime			UnitName.Admin.AlarmHistory[#]	Date-Time Array
				[0] (year)		UnitName.Admin.AlarmHistory[#].DateTime[0]	Int (32bit)
				[1] (month)		UnitName.Admin.AlarmHistory[#].DateTime[1]	Int (32bit)
				[2] (day)		UnitName.Admin.AlarmHistory[#].DateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.AlarmHistory[#].DateTime[3]	Int (32bit)
				[4] (min)		UnitName.Admin.AlarmHistory[#].DateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.AlarmHistory[#].DateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.AlarmHistory[#].DateTime[6]	Int (32bit)
			AckDateTime			UnitName.Admin.AlarmHistory[#]	Date-Time Array
				[0] (year)		UnitName.Admin.AlarmHistory[#].AckDateTime[0]	Int (32bit)
				[1] (month)		UnitName.Admin.AlarmHistory[#].AckDateTime[1]	Int (32bit)
				[2] (day)		UnitName.Admin.AlarmHistory[#].AckDateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.AlarmHistory[#].AckDateTime[3]	Int (32bit)
				[4] (min)		UnitName.Admin.AlarmHistory[#].AckDateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.AlarmHistory[#].AckDateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.AlarmHistory[#].AckDateTime[6]	Int (32bit)
		AlarmHistoryExtent				UnitName.Admin.AlarmHistoryExtent	Int (32bit)
		StopReason				UnitName.Admin.StopReason	Alarm Structure

MIN REQ						TAGNAME	DATATYPE
			Trigger			UnitName.Admin.StopReason.Trigger	Bool
x			ID			UnitName.Admin.StopReason.ID	Int (32bit)
			Value			UnitName.Admin.StopReason.Value	Int (32bit)
			Message			UnitName.Admin.StopReason.Message	String
			Category			UnitName.Admin.StopReason.Category (Event Grouping)	Int (32bit)
			DateTime			UnitName.Admin.StopReason[#]	Date-Time Array
				[0] (year)		UnitName.Admin.StopReason[#].DateTime[0]	Int (32bit)
				[1] (month)		UnitName.Admin.StopReason[#].DateTime[1]	Int (32bit)
				[2] (day)		UnitName.Admin.StopReason[#].DateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.StopReason[#].DateTime[3]	Int (32bit)
				[4] (min)		UnitName.Admin.StopReason[#].DateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.StopReason[#].DateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.StopReason[#].DateTime[6]	Int (32bit)
			AckDateTime			UnitName.Admin.StopReason[#]	Date-Time Array
				[0] (year)		UnitName.Admin.StopReason[#].AckDateTime[0]	Int (32bit)
				[1] (month)		UnitName.Admin.StopReason[#].AckDateTime[1]	Int (32bit)
				[2] (day)		UnitName.Admin.StopReason[#].AckDateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.StopReason[#].AckDateTime[3]	Int (32bit)
				[4] (min)		UnitName.Admin.StopReason[#].AckDateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.StopReason[#].AckDateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.StopReason[#].AckDateTime[6]	Int (32bit)
		StopReasonExtent				UnitName.Admin.StopReasonExtent	Int (32bit)
		Warning[#]				UnitName.Admin.Warning[#]	Alarm Structure
			Trigger			UnitName.Admin.Warning[#].Trigger	Bool
			ID			UnitName.Admin.Warning[#].ID	Int (32bit)
			Value			UnitName.Admin.Warning[#].Value	Int (32bit)
			Message			UnitName.Admin.Warning[#].Message	String
			Category			UnitName.Admin.Warning[#].Category (Event Grouping)	Int (32bit)

MIN REQ						TAGNAME	DATATYPE
			DateTime			UnitName.Admin.Warning[#]	Date-Time Array
				[0] (year)		UnitName.Admin.Warning[#].DateTime[0]	Int (32bit)
				[2] (day)		UnitName.Admin.Warning[#].DateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.Warning[#].DateTime[3]	Int (32bit)
				[4] (min)		UnitName.Admin.Warning[#].DateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.Warning[#].DateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.Warning[#].DateTime[6]	Int (32bit)
			AckDateTime			UnitName.Admin.Warning[#]	Date-Time Array
				[0] (year)		UnitName.Admin.Warning[#].AckDateTime[0]	Int (32bit)
				[1] (month)		UnitName.Admin.Warning[#].AckDateTime[1]	Int (32bit)
				[2] (day)		UnitName.Admin.Warning[#].AckDateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.Warning[#].AckDateTime[3]	Int (32bit)
				[4] (min)		UnitName.Admin.Warning[#].AckDateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.Warning[#].AckDateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.Warning[#].AckDateTime[6]	Int (32bit)
		WarningExtent				UnitName.Admin.WarningExtent	Int (32bit)
		ModeCurrentTime[#]				UnitName.Admin.ModeCurrentTime[#]	Int (32bit)
		ModeCumulativeTime[#]				UnitName.Admin.ModeCumulativeTime[#]	Int (32bit)
		StateCurrentTime[#,#]	(Mode,State)			UnitName.Admin.StateCurrentTime[#,#] (Mode,State)	Int (32bit)
		StateCumulativeTime[#,#]	(Mode,State)			UnitName.Admin.StateCumulativeTime[#,#] (Mode,State)	Int (32bit)
		ProdConsumedCount[#]				UnitName.Admin.ProdConsumedCount[#]	Count Structure
			ID			UnitName.Admin.ProdConsumedCount[#].ID	Int(32bit)
			Name			UnitName.Admin.ProdConsumedCount[#].Name	String
			Unit			UnitName.Admin.ProdConsumedCount[#].Unit	String
			Count			UnitName.Admin.ProdConsumedCount[#].Count	Int(32bit)
			AccCount			UnitName.Admin.ProdConsumedCount[#].AccCount	Int(32bit)
		ProdProcessedCount[#]				UnitName.Admin.ProdProcessedCount[#]	Count Structure
			ID			UnitName.Admin.ProdProcessedCount[#].ID	Int(32bit)

MIN REQ						TAGNAME	DATATYPE
			Name			UnitName.Admin.ProdProcessedCount[#].Name	String
			Unit			UnitName.Admin.ProdProcessedCount[#].Unit	String
x			Count			UnitName.Admin.ProdProcessedCount[#].Count	Int(32bit)
			AccCount			UnitName.Admin.ProdProcessedCount[#].AccCount	Int(32bit)
		ProdDefectiveCount[#]				UnitName.Admin.ProdDefectiveCount[#]	Count Structure
			ID			UnitName.Admin.ProdDefectiveCount[#].ID	Int(32bit)
			Name			UnitName.Admin.ProdDefectiveCount[#].Name	String
			Unit			UnitName.Admin.ProdDefectiveCount[#].Unit	String
x			Count			UnitName.Admin.ProdDefectiveCount[#].Count	Int(32bit)
			AccCount			UnitName.Admin.ProdDefectiveCount[#].AccCount	Int(32bit)
		AccTimeSinceReset	AccTimeSinceReset			UnitName.Admin.AccTimeSinceReset	Int(32bit)
		MachDesignSpeed				UnitName.Admin.MachDesignSpeed	Real
		StatesDisabled				UnitName.Admin.StatesDisabled	Int(32bit)
		PLCDateTime				UnitName.Admin.PLCDateTime	Date-Time Array
				[0] (year)		UnitName.Admin.PLCDateTime[0]	Int (32bit)
				[1] (month)		UnitName.Admin.PLCDateTime[1]	Int (32bit)
				[2] (day)		UnitName.Admin.PLCDateTime[2]	Int (32bit)
				[3] (hour)		UnitName.Admin.PLCDateTime[3]	Int (32bit)
				[4] (min)		UnitName.Admin.PLCDateTime[4]	Int (32bit)
				[5] (sec)		UnitName.Admin.PLCDateTime[5]	Int (32bit)
				[6] (usec)		UnitName.Admin.PLCDateTime[6]	Int (32bit)

Table 7: PackTags: Minimum required for information/machine monitoring

STATUS TAGS						TAGNAME	DATATYPE
UnitName						UnitName	PackMLv30
	Status					UnitName.Status	PMLs
		UnitModeCurrent				UnitName.Status.UnitModeCurrent	Int (32bit)
		StateCurrent				UnitName.Status.StateCurrent	Int (32bit)
		MachSpeed				UnitName.Status.MachSpeed	Real
		CurMachSpeed				UnitName.Status.CurMachSpeed	Real
		EquipmentInterlock				UnitName.Status.EquipmentInterlock	Bool Structure [2]
			Blocked			UnitName.Status.EquipmentInterlock.Blocked	Bool
			Starved			UnitName.Status.EquipmentInterlock.Starved	Bool
ADMIN TAGS						TAGNAME	DATATYPE
UnitName						UnitName	PackMLv30
	Admin					UnitName.Admin	PMLa
		ProdProcessedCount[#]	Count			UnitName.Admin.ProdProcessedCount[#].Count	Int(32bit)
		ProdDefectiveCount[#]	Count			UnitName.Admin.ProdDefectiveCount[#].Count	Int(32bit)
		StopReason	ID			UnitName.Admin.StopReason.ID	Int (32bit)

Table 8: PackTags: Minimum required for supervisory control

COMMAND TAGS						TAGNAME	DATATYPE
UnitName						UnitName	PackMLv30
	Command					UnitName.Command	PMLc
		UnitMode				UnitName.Command.UnitMode	Int (32bit)
		UnitModeChangeRequest				UnitName.Command.UnitModeChangeRequest	Bool
		MachSpeed				UnitName.Command.MachSpeed	Real
		CntrlCmd				UnitName.Command.CntrlCmd	Int (32bit)
		CmdChangeRequest				UnitName.Command.CmdChangeRequest	Bool

7.5.1 Command tags

Command tags are used to control the operation of the unit machine. Command tags include unit state commands which control the state transitions in the base state model. The command tags also include parameters and process variables which control how the machine operates. Command tags generally originate from the machine user or a remote system. The originator of the command in this report is defined as the "requestor" or "information sender". The unit machine in this report is known as the "execution system".

7.5.1.1 Command.UnitMode

Data Type: INT (32bit)
Tag Descriptor: Unit Mode Target

This value is predefined by the user/OEM, and are the desired unit modes of the machine. The UnitMode tag is a numerical representation of the commanded mode. There can be any number of unit modes, and for each unit mode there is an accompanying state model. Example unit modes are production, maintenance, manual, clean out, dry run, setup, etc.

See Figure 4 in Clause 5 for default mode definitions.

7.5.1.2 Command.UnitModeChangeRequest

Data Type: Bool
Tag Descriptor: Request Unit Mode Change

When a unit mode request takes place, a numerical value must be present in the Command.UnitMode tag to change the unit mode. Local processing and conditioning of the requested mode change is necessary in order to accept, reject, or condition the timing of the change request.

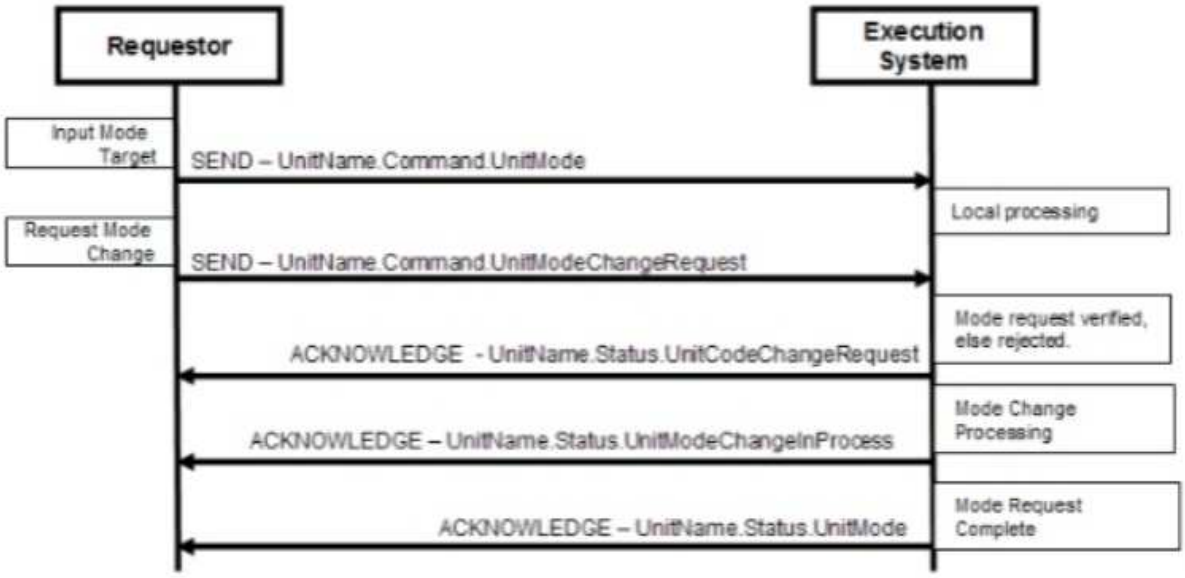


Figure 12: Unit mode change example sequence

7.5.1.3 Command.MachSpeed

Data Type: REAL
Unit of Measure: Primary Packages/Minute
Tag Descriptor: Current Machine Speed

This defines the set point for the current speed of the machine in primary packages per minute. Keeping speed in a primary package unit of measure (UOM) allows for easier control integration. The primary package UOM is the normalized rate for the machine, normalized to a value chosen on the line. The following example is for a bottle line running at balance line speed of 1000 packages/minute. The UOM chosen is equivalent to be the actual count of the filler, or labeler.

Machine	Actual Pack Counts	Primary packages (UOM)
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

7.5.1.4 Command.MaterialInterlock

Data Type: Bool Structure of 32 Bits in Length
Tag Descriptor: Materials Ready

Indicates materials are ready for processing. It is comprised of a series of bits with 1 equaling ready or not low, 0 equaling not ready, or low. Each bit represents a different user material. Materials are defined as all consumables such as product, cartons, labels, utilities, and glue. The word contains bits that indicate when a critical material or process parameter is ready for use. It can also be used for production, and/or indication of low condition. This information may be sent to the unit machine at any time as the interlock information changes.

MATERIALINTERLOCK EXAMPLE	Raw Material #1 – Not Low	Raw Material #1 - Ready	Air Pressure - Ready	Compressed Air - Ready	Lubrication Water - Ready	Container Caps – Not Low	Container Caps – Ready	Undefined / Unused	Undefined / Unused	Undefined / Unused
MaterialInterlock.[Value]	1	1	1	1	1	1	1	1	1	1
MaterialInterlock.[Bit #]	0	1	2	3	4	5	6	..	30	31

7.5.1.5 Command.CntrlCmd

Data Type: INT (32bit)
Tag Descriptor: Control Command

The tag holds the value of the command that provides the state command to drive a state change in the base state model, this tag is typically manipulated locally. Local processing of this tag, local machine conditions and local commands drive changes between machine states. This tag can be set by a local or remote source. All values in the table below are reserved.

0	Undefined
1	Reset
2	Start
3	Stop
4	Hold
5	Unhold
6	Suspend
7	Unsuspend
8	Abort
9	Clear

7.5.1.6 Command.CmdChangeRequest

Data Type: Bool
Tag Descriptor: State Change Request

This CmdChangeRequest bit will command the machine to proceed to change the state to the target state. The tag can be used to condition when a change of state can occur. The target state will be one of the states in the base state model.

7.5.1.7 Command.RemoteInterface[#]

Data Type: Structured Array of DataType Interface
Tag Descriptor: Upstream or Downstream Machine

This structured array is use for coordinating upstream or downstream machines in a cell of multiple unit machines. The array is chosen to be of a length that is equal to the number of machines that will be sending commands. This could be expanded if a machine is capable of receiving material from multiple upstream and/or downstream machines, thereby receiving multiple commands and parameters. This can be used for machine to machine coordination without supervisory control, or for tightly controlled units under supervisory control. These tags are typically used for consumption within the unit machine procedure. Specifically, if a remote controller was issuing commands, the commands would be read by this tag and used in the unit machine.

7.5.1.7.1 Command.RemoteInterface[#].Number

Data Type: INT (32bit)
Tag Descriptor: Identification Number of Upstream or Downstream Unit Machine

This is the unique number for the downstream/upstream unit machine using a common tag structure as the unit machine. The number should correspond to a number on the communication network, such as network ID, or IP address identifier. This number corresponds to the "information sender" that is setting the command data in the RemoteInterface[#] structure of the unit machine.

7.5.1.7.2 Command.RemoteInterface[#].ControlCmdNumber

Data Type: INT (32bit)
Tag Descriptor: Control Command for Upstream or Downstream Machine

A user defined command number associated with coded value from a remote unit. This number is a coded value sent from one node on the network to another. The value can be associated with a unit mode change request, speed change request, a state change request, etc.

7.5.1.7.3 Command.RemoteInterface[#].CmdValue

Data Type: INT (32bit)
Tag Descriptor: Control Command Value Associated ControlCmdNumber

This is the command value associated with the ControlCmdNumber above. The command value may be the speed requested, state change, etc.

Example:
For an upstream machine designated as #2 a control command number of 5 may be related to the speed setting value for the machine. A value of 400 can be used to modify the remote machine setpoint.
Command.RemoteInterface[1].Number = 2
Command.RemoteInterface[1].ControlCmdNumber = 5
Command.RemoteInterface[1].CmdValue = 400

For a downstream machine designated as #4 the control command number of 0 can be used to remotely command a state transition for the machine. The value of 2 is the command value for start.
Command.RemoteInterface[1].Number = 4
Command.RemoteInterface[1].ControlCmdNumber = 0
Command.RemoteInterface[1].CmdValue = 2

7.5.1.7.4 Command.RemoteInterface[#].Parameter[#]

Data Type: Structured Array of Data Type Descriptor

The parameter tags associated to commanded remote interface are typically used for command parameters that are given to the unit machine from remote machines. The parameters are typically needed for coordinating the unit machine or production with other machines. The parameter value may be anything from machine limit parameters to temperatures and counter presets. The parameters are typically limited to machine parameters as product and process parameters are described in later tags.

7.5.1.7.4.1 Command.RemoteInterface[#].Parameter[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) ID value assigned to the parameter. This is a non-descript value that can be used for user tag requirements.

7.5.1.7.4.2 Command.RemoteInterface[#].Parameter[#].Name

Data Type: String
Tag Descriptor: String Value Assigned to Parameter

The literal parameter name is used to describe the parameter variable number, and its associated value from the remote interface. An example parameter name may be GLUE TEMP, BEARING TEMP, OVERLOAD TIME, etc.

7.5.1.7.4.3 Command.RemoteInterface[#].Parameter[#].Unit

Data Type: String[5]
Tag Descriptor: String Value of Parameter Unit of Measure

Unit is a string that describes the unit of measure associated with the parameter's value, i.e secs, deg, rpm, ppm, etc. This tag describes the unit of measure associated with the following tag value sent from the remote interface.

7.5.1.7.4.4 Command.RemoteInterface[#].Parameter[#].Value

Data Type: REAL
Tag Descriptor: Numeric Value of Parameter

This is the numeric value of the parameter. The value is described by the Parameter[#].ID, Parameter[#].Name, and the unit of measure is described by the Parameter[#].Unit sent by the remote interface as a command to the unit machine.

7.5.1.8 Command.Parameter[#]

Data Type: Array of Data Type Descriptor

The Parameter tags are associated to the end user supervisory interface and are typically used for command parameters that are given to the unit by the end user's supervisory HMI. The parameters are typically needed for running the unit machine. The parameter value may be anything from machine limit parameters to temperatures and counter presets. The parameters are typically limited to machine parameters as product and process parameters are described in later tags.

7.5.1.8.1 Command.Parameter[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) ID value of the parameter. This is a non-descript value that can be used for any user tag requirements.

7.5.1.8.2 Command.Parameter[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Parameter Names for the Machine Unit.

The literal parameter name is used to describe the parameter number, and its associated value. An example parameter name may be GLUE TEMP, BEARING TEMP, OVERLOAD TIME, etc.

7.5.1.8.3 Command.Parameter[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Parameter Unit of Measure for the Machine Unit.

The parameter unit is used to describe the unit of measure of the parameter, and its associated value. An example parameter unit of measure may be DegF, secs, PPM, revs, mm, etc.

7.5.1.8.4 Command.Parameter[#].Value

Data Type: User Defined
Tag Descriptor: Structured Array of Parameter Values or Strings

This is the numeric or string value of the parameter. The value is described by the Parameter[#].ID, Parameter[#].Name, and the unit of measure is described by the Parameter[#].Unit commanded by the local interface, or local processor, as a command to the unit machine. An example is the following:

Command.Parameter[1].Value = 22.5
Command.Parameter[2].Value = 12

An example of a machine unit process variable:

Command.Parameter[1].Name = BEARING_1_OVERTEMP
Command.Parameter[1].Unit = DegC
Command.Parameter[1].Value = 350.00

This defines the temperature of a Bearing Overtemp alarm of the #1 bearing and is to be set at 350.0 Degrees C for all products.

7.5.1.9 Command.Product[#]

Data Type: Array of Data Type Product

The product data type can be used for defining product and product processing parameter variables. The command tags can come from either a local HMI or remote systems and are used to process the product on the unit machine. The array is typically needed for machines that run multiple products.

7.5.1.9.1 Command.Product[#].ProductID

Data Type: INT (32bit)
Tag Descriptor: Structured Array of Product ID#

This product ID is used to indicate to the machine which product it is producing (i.e. SKU or UPC). The array can be used for machines that run multiple products.

7.5.1.9.2 Command.Product[#].ProcessVariables[#]

Data Type: Array of Data Type Descriptor

The ProcessVariables structured array can be used for specific process variables needed by the unit machine for the processing of a specific product. Process variables include set points, limits, quality parameters, etc., that are needed to produce a particular product on a unit machine. The number of tags for this array will be the maximum number of needed process variables for any particular product defined on the unit machine.

7.5.1.9.2.1 Command.Product[#].ProcessVariables[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) process variable ID value assigned to the process variable.

7.5.1.9.2.2 Command.Product[#].ProcessVariables[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Process Variable Names for Multiple Product ID#s

The process variable literal name is used to describe the process variable number, and its associated value. An example process variable name may be GLUE TEMP, MaxTimeInMachine, MixingTime, KnifeSpeed, ChillRollPhaseOffset, etc.

7.5.1.9.2.3 Command.Product[#].ProcessVariables[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Process Variable Unit of Measure

The process variable unit of measure is a string data type used to describe the unit of measure of the process variable number, and its associated value. An example process unit of measure may be DegF, secs, PPM, revs, mm, etc. .

7.5.1.9.2.4 Command.Product[#].ProcessVariables[#].Value

Data Type: Real
Tag Descriptor: Structured Array of Process Variable Values

The process variable value is used to specify a process variable for the Product[#] specified by the ProcessVariable[#].ID, ProcessVariables[#].Name, and ProcessVariables[#].Unit.

Command.Product[1].ProcessVariables[1].Value = 22.5
Command.Product[1].ProcessVariables[2].Value = 12

Combined with other examples of Product[#]

Command.Product[1].ProcessVariables[1].Name = Glue1Temp
Command.Product[1].ProcessVariables[1].Unit = DegF
Command.Product[1].ProcessVariables[1].Value = 356.4

Meaning the temperature of Glue station 1 is to be controlled at 356.4 Degrees F for product number 1.

Other Examples of Process Variable Name For Products:

Name = ProductMaxTimeInMachine
Name = ProductMinTimeInMachine
Name = ByproductID
Name = ByproductsMaxTimeInMachine

7.5.1.9.3 Command.Product[#].Ingredients[#]

Data Type: Array of Data Type Ingredient

This array serves to hold the information needed for the raw materials that are used by the unit machine in the processing of a particular product. The extent of this array will be the maximum number of ingredients used in the processing of any particular product.

7.5.1.9.3.1 Command.Product[#].Ingredients[#].IngredientID

Data Type: INT (32bit)
Tag Descriptor: Structured Array of Ingredient IDs

The IngredientID is an arbitrary number associated with the raw material, or ingredient for a particular product number. The user will define the value associated to the ingredient IDs that are used in the operation of the machine for a particular product. Each ingredient should have a distinct ID (SKU or UPC).

7.5.1.9.3.2 Command.Product[#].Ingredients[#].Parameter[#]

Data Type: Array of Data Type Descriptor

This array or structure is used for parameters associated with a particular ingredient or raw material used for the processing of a particular product number. This command tag is typically set by an "information sender" to the unit machine controller. The extent of this array is the maximum number of parameters associated with any ingredient in any product that is defined on the unit machine.

7.5.1.9.3.2.1 Command.Product[#].Ingredients[#].Parameter[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) ID value assigned to one of the parameters for the ingredient or raw material, needed for the processing of the product defined by the product number.

7.5.1.9.3.2.2 Command.Product[#].Ingredients[#].Parameter[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Ingredient Parameter Names

The parameter variable name in the parameter array is used to describe the parameter names associated with a specific ingredient number in a specific product number. An example parameter name may be SETUP TIME, TEMP, TAB POSITION, etc. The array is typically needed for machines that run multiple ingredients with multiple parameters with multiple products.

7.5.1.9.3.2.3 Command.Product[#].Ingredients[#].Parameter[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Unit of Measure

The parameter unit tag is used to describe the parameter names associated with a specific parameter in a specific ingredient in a specific product. An example unit of measure name may be DegF, secs, PPM, revs, mm, etc. The array is typically needed for unit machines that run multiple products with multiple ingredients with multiple processing parameters.

7.5.1.9.3.2.4 Command.Product[#].Ingredients[#].Parameter[#].Value

Data Type: Real
Tag Descriptor: Structured Array of Values

The ingredient parameter value is used to specify a parameter variable for the control of the process. The array is typically needed for unit machines that run multiple products with multiple ingredients with multiple processing parameters. As an example with Ingredient number 1 in Product number 1:

Command.Product[1].Ingredients[1].Parameter[1].Value = 225
Command.Product[1].Ingredients[1].Parameter[2].Value = 12

Combined with other examples of Product[#]

Command.Product[1].Ingredients[1].Parameter[1].Name = KNIFE_OFFSET
Command.Product[1].Ingredients[1].Parameter[1].Unit = degrees
Command.Product[1].Ingredients[1].Parameter[1].Value = 3.564

Meaning the offset of the knife is to be controlled at 3.564 Degrees for Parameter variable number 1, on ingredient 1 for product number 1.

7.5.2 Status tags

Status tags are used to describe the operation of the unit machine. Status tags include state commands which describe the state transitions in the base state model. The status tags also include parameters and process variables which describe how the machine operates. Status tags generally originate from the unit machine user and can be used on a remote system. The originator of the status tags in this report is defined as the "execution system".

7.5.2.1 Status.UnitModeCurrent

Data Type: INT (32bit)
Tag Descriptor: Unit Mode in Current Use

This value is predefined by the user/OEM of the available unit modes of the machine allowing a possible different set of states for the base state model and could provide completely different functionality in the same machinery such as Cleanout, Production, etc.

0	Invalid
1	Production
2	Maintenance
3	Manual
04 - 31	User Definable

7.5.2.2 Status.UnitModeRequested

Data Type: Bool
Tag Descriptor: Requested Unit Mode Change

When a unit mode request takes place a numerical value must be present in the unit mode target to change the unit mode. Local processing and conditioning of the requested mode change is necessary in order to accept, reject, or condition the timing of the change request.

7.5.2.3 Status.UnitModeChangeInProcess

Data Type: Bool
Tag Descriptor: Requested Unit Mode Change In Process

When a unit mode request takes place, this tag reflects the status of the state model. If the state of the machine required time to change mode this bit would track the request and reset when the change was completed.

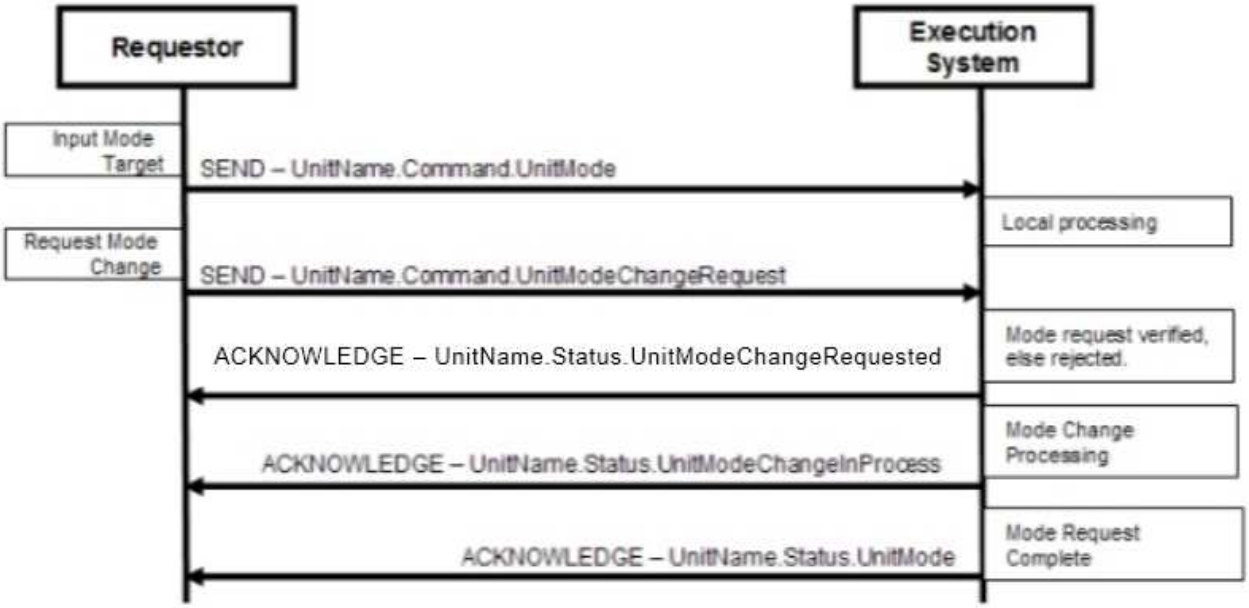


Figure 13: Unit mode change example sequence

7.5.2.4 Status.StateCurrent

Data Type: INT (32bit)
Tag Descriptor: Current State Number

The StateCurrent status tag specifies the current state in the current unit mode of the unit machine. The numerical values are in the table below are reserved.

0	Undefined
1	“Clearing”
2	“Stopped”
3	“Starting”
4	“Idle”
5	“Suspended”
6	“Execute”
7	“Stopping”
8	“Aborting”
9	“Aborted”
10	“Holding”
11	“Held”
12	“UnHolding”
13	“Suspending”
14	“Unsuspending”
15	“Resetting”
16	“Completing”
17	“Complete”

7.5.2.5 Status.StateRequested

Data Type: INT (32bit)
Tag Descriptor: Target State

This value is used for state transition checking, to ensure that transition to a target state can be achieved. The target state, StateRequested, is a numerical value corresponding to a state in the base state model (shown above).

7.5.2.6 Status.StateChangeInProcess

Data Type: Bool
Tag Descriptor: State Change in Process

This bit indicates that a change in state is in progress following a state change request command.

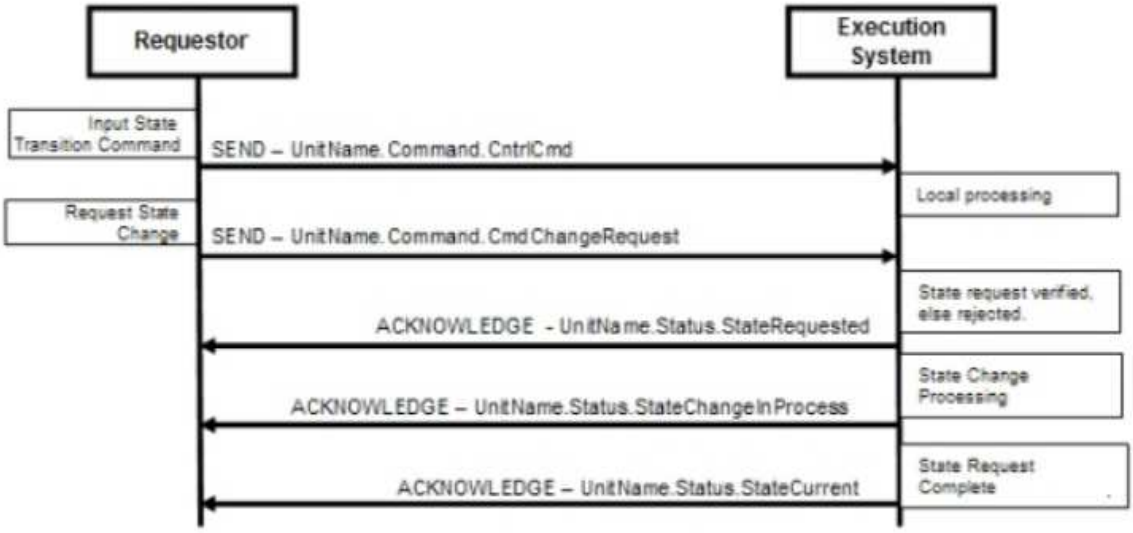


Figure 14: State change example sequence

7.5.2.7 Status.MachSpeed

Data Type: REAL
Units: Primary Packages/Minute
Tag Descriptor: Current Machine Speed

This describes the set point for the current speed of the machine in primary packages per minute. Keeping speed in a primary package unit of measure (UOM) allows for easier control integration. The primary package UOM is the normalized rate for the machine, normalized to a value chosen on the line. The following example is for a bottle line running at balance line speed of 1000 packages/minute. The UOM chosen is equivalent to be the actual count of the filler, or labeler.

Machine	Actual Pack Counts	Primary packages (UOM)
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

7.5.2.8 Status.CurMachSpeed

Data Type: Real
Tag Descriptor: Current Machine Speed in Primary Packages/Minute

This the actual value of the machine speed. Keeping units in primary package unit of measure (UOM), allows for easier control integration. The primary package UOM is the normalized rate for the machine, normalized to a value chosen on the line. Pack counts are parameters stored in the administration tags or downloaded parameters stored in command tags parameters.

7.5.2.9 Status.MaterialInterlock

Data Type: Structure of 32 Bits in Length
Tag Descriptor: Materials Ready

MaterialInterlock describes the status of the materials that are ready for processing. It is comprised of a series of bits with 1 equaling ready or not low, 0 equaling not ready, or low. Each bit represents a different user material. Materials are defined as all consumables such as product, cartons, labels, utilities, and glue. The word contains bits that indicate when a critical material or process parameter is ready for use; it can also be used for production, and/or indication of low condition. This information is set by the unit machine at any time as the interlock information changes.

MATERIALINTERLOCK EXAMPLE	Raw Material #1 – Not Low	Raw Material #1 - Ready	Air Pressure - Ready	Compressed Air - Ready	Lubrication Water - Ready	Container Caps – Not Low	Container Caps – Ready	Undefined / Unused	Undefined / Unused	Undefined / Unused
MaterialInterlock.[Value]	1	1	1	1	1	1	1	1	1	1
MaterialInterlock.[Bit #]	0	1	2	3	4	5	6	7	8	31

7.5.2.10 Status.EquipmentInterlock.Blocked

Data Type: Bool

This bit, when set to 1, indicates that a downstream system is not able to accept product. In this condition, the equipment is capable of producing product but is in a suspended state due to a downstream system. This tag is necessary for external equipment monitoring so that the reason for the machine being in a suspended state can be identified.

7.5.2.11 Status.EquipmentInterlock.Starved

Data Type: Bool

This bit, when set to 1, indicates that an upstream system is not able to supply product. In this condition, the equipment is capable of producing product but is in a suspended state due to an upstream system. This tag is necessary for external equipment monitoring so that the reason for the machine being in a suspended state can be identified.

7.5.2.12.4 Status.RemoteInterface[#].Parameter[#]

Data Type: Structured Array of Data Type Descriptor

The status parameter tags associated to the remote interface are typically used for parameters that are sent to the remote machine(s) from the unit machine. The parameters are typically needed for coordinating the unit with other machines. The parameter value may be anything from machine limit parameters to temperatures and counter presets. The parameters are typically limited to machine parameters as product and process parameters are described in later tags.

7.5.2.12.4.1 Status.RemoteInterface[#].Parameter[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) ID value assigned to the parameter. This is a non-descript value that can be used for user tag requirements.

7.5.2.12.4.2 Status.RemoteInterface[#].Parameter[#].Name

Data Type: String
Tag Descriptor: String Value Assigned to Parameter

The literal parameter name is used to describe the parameter variable number, and its associated value from the unit machine. An example parameter name may be GLUE TEMP, BEARING TEMP, OVERLOAD TIME, etc. This also could be displayed on HMI screens.

7.5.2.12.4.3 Status.RemoteInterface[#].Parameter[#].Unit

Data Type: String[5]
Tag Descriptor: String Value of Parameter Unit of Measure

Unit is a string that describes the unit of measure associated with the parameter's value, i.e. secs, deg, rpm, ppm, etc. This tag describes the unit of measure associated with the following tag value that is sent from the unit machine.

7.5.2.12.4.4 Status.RemoteInterface[#].Parameter[#].Value

Data Type: REAL
Tag Descriptor: Numeric Value of Parameter

This is the numeric value of the parameter. The value is described by the Parameter[#].ID, Parameter[#].Name, and is a unit of measure that is parameterized by the Parameter[#].Unit. The value can sent by the unit machine as information to a remote machine.

7.5.2.13 Status.Parameter[#]

Data Type: Array of Data Type Descriptor

The parameter tags are associated to the end user supervisory interface and are typically used for parameters that are displayed by the end user's supervisory HMI. The parameters are typically needed for running the unit machine. The parameter value may be anything from machine limit parameters to temperatures and counter presets. The parameters are typically limited to machine parameters as product and process parameters are described in later tags. The extent of the array is dependent on the number of parameters needed.

7.5.2.13.1 Status.Parameter[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) ID value of the parameter. This is a non-descript value that can be used for any user tag requirements.

7.5.2.13.2 Status.Parameter[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Parameter Variable Names for the Machine Unit

The literal parameter name is used to describe the parameter number, and its associated value. An example parameter name may be GLUE TEMP, BEARING TEMP, OVERLOAD TIME, etc. This also could be displayed on HMI screens.

7.5.2.13.3 Status.Parameter#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Parameter Unit of Measure for the Machine Unit

The parameter unit is used to describe the unit of measure of the parameter, and its associated value. An example parameter unit of measure may be DegF, secs, PPM, revs, mm, etc. This also could be displayed on HMI screens.

7.5.2.13.4 Status.Parameter[#].Value

Data Type: User Defined
Tag Descriptor: Structured Array of Parameter Values or Strings

This is the numeric or string value of the parameter. The value is described by the Parameter[#].ID, Parameter[#].Name, and the unit of measure is described by the Parameter[#].Unit on the local machine. An example is the following:

Status.Parameter[1].Value = 22.5
Status.Parameter[2].Value = 12

An example of a machine unit process variable:

Status.Parameter[1].Name = BEARING_1_OVERTEMP
Status.Parameter[1].Unit = DegC
Status.Parameter[1].Value = 350.00

This defines the temperature of a bearing overtemp alarm of the #1 bearing is to be set at 350.0 Degrees C for all products.

7.5.2.14 Status.Product[#]

Data Type: Array of Data Type Product

The product data type can be used for displaying product and product processing parameter variables. The status tags can come from either a local HMI or remote systems and are used to display or send information on the product(s) on the unit machine. The array is typically needed for machines that run multiple products.

7.5.2.14.1 Status.Product[#].ProductID

Data Type: INT (32bit)
Tag Descriptor: Structured Array of Product ID#

This product ID is used to indicate to the machine which product it is producing (i.e. SKU or UPC). This can also be displayed on all HMI screens. The array can be used for machines that run multiple products.

7.5.2.14.2 Status.Product[#].ProcessVariables[#]

Data Type: Array of Data Type Descriptor

The ProcessVariables structured array can be used to display specific process variables used by the unit machine for the processing of a specific product. Process variables include set points, limits, quality parameters, etc., that are displayed to produce a particular product on a unit machine. The number of tags for this array will be the maximum number of needed process variables for any particular product defined on the unit machine.

7.5.2.14.3 Status.Product[#].ProcessVariables[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) process variable ID value assigned to the process variable.

7.5.2.14.4 Status.Product[#].ProcessVariables[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Process Variable Names for Multiple Product ID#s

The process variable literal name is used to describe the process variable number, and its associated value. An example process variable name may be GLUE TEMP, MaxTimeInMachine, MixingTime, KnifeSpeed, ChillRollPhaseOffset, etc. This also could be displayed on HMI screens.

7.5.2.14.5 Status.Product[#].ProcessVariables[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Process Variable Unit of Measure for Multiple Product ID#s

The process variable unit is used to describe the units of the process variable number, and its associated value. An example process unit name may be DegF, secs, PPM, revs, mm, etc. This also could be displayed on HMI screens. The array is typically needed for machines that run multiple products.

7.5.2.14.6 Status.Product[#].ProcessVariables[#].Value

Data Type: Real
Tag Descriptor: Structured Array of Process Variable Values

The process variable value is used to specify a process variable for the Product[#] specified by the ProcessVariable[#].ID, ProcessVariables[#].Name, and ProcessVariables[#].Unit. This also could be displayed on HMI screens.

Status.Product[1].ProcessVariables[1].Value = 22.5
Status.Product[1].ProcessVariables[2].Value = 12

Combined with other examples of Product[#]:

Status.Product[1].ProcessVariables[1].Name = Glue1Temp
Status.Product[1].ProcessVariables[1].Unit = DegF
Status.Product[1].ProcessVariables[1].Value = 356.4

Meaning the temperature of Glue station 1 is to be controlled at 356.4 Degrees F for Product number 1.

Other Examples of Process Variable Name For Products:

Name = ProductMaxTimeInMachine
Name = ProductMinTimeInMachine
Name = ByproductID
Name = ByproductsMaxTimeInMachine

7.5.2.14.7 Status.Product[#].Ingredients[#]

Data Type: Array of Data Type Ingredient

This array serves to hold the information needed for the raw materials that are used by the unit machine in the processing of a particular product. The extent of this array will be the maximum number of ingredients used in the processing of any particular product.

7.5.2.14.7.1 Status.Product[#].Ingredients[#].IngredientID

Data Type: INT (32bit)
Tag Descriptor: Structured Array of Ingredient IDs

The IngredientID is an arbitrary number associated with the raw material, or ingredient for a particular product number. The user will define the value associated to the ingredient IDs that are used in the operation of the machine for a particular product. Each ingredient should have a distinct ID (SKU or UPC).

7.5.2.14.7.2 Status.Product[#].Ingredients[#].Parameter[#]

Data Type: Array of Data Type Descriptor

This array of structures is used for parameters associated with a particular ingredient or raw material used for the processing of a particular product number. This status tag is typically set by the unit machine controller. The extent of this array is the maximum number of parameters associated with any ingredient in any product that is defined on the unit machine.

7.5.2.14.7.2.1 Status.Product[#].Ingredients[#].Parameter[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) ID value assigned to one of the parameters for the ingredient or raw material, needed for the processing of the product defined by the product number.

7.5.2.14.7.2.2 Status.Product[#].Ingredients[#].Parameter[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Ingredient Parameter Names

The parameter variable name in the parameter array is used to describe the parameter names associated with a specific ingredient number in a specific product number. An example parameter name may be SETUP TIME, TEMP, TAB POSITION etc. This also could be displayed on HMI screens. The array is typically needed for machines that run multiple ingredients with multiple parameters with multiple products.

7.5.2.14.7.2.3 Status.Product[#].Ingredients[#].Parameter[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Unit of Measure

The parameter unit tag is used to describe the parameter names associated with a specific parameter in a specific ingredient in a specific product. An example process unit of measure name may be DegF, secs, PPM, revs, mm, etc. This also could be displayed on HMI screens. The array is typically needed for unit machines that run multiple products with multiple ingredients with multiple processing parameters.

7.5.2.14.7.2.4 Status.Product[#].Ingredients[#].Parameter[#].Value

Data Type: Real
Tag Descriptor: Structured Array of Values

The ingredient parameter value is used to specify a parameter variable for displaying information about the process. This also could be displayed on HMI screens. The array is typically needed for unit machines that run multiple products with multiple ingredients with multiple processing parameters. As an example with Ingredient number 1 in Product number 1:

Status.Product[1].Ingredients[1].Parameter[1].Value = 225
Status.Product[1].Ingredients[1].Parameter[2].Value = 12

Combined with other examples of Product[#]

Status.Product[1].Ingredients[1].Parameter[1].Name = KNIFE_OFFSET
Status.Product[1].Ingredients[1].Parameter[1].Unit = degrees
Status.Product[1].Ingredients[1].Parameter[1].Value = 3.564

Meaning the offset of the knife is to be controlled at 3.564 Degrees for Parameter variable number 1, on Ingredient 1 for product number 1.

7.5.3 Administration tags

Administration tags are used to describe the quality and alarm information of the unit machine. Administration tags include alarm parameters which describe the conditions within the base state model typically for production data acquisition (PDA) systems. The administration tags also include parameters which can describe how well the machine operates, or specific information on the product quality produced by the machine. Administration tags generally originate from the unit machine and can be used on the HMI or a remote system.

Some administration tags support transfer of data for OEE calculations. Refer to ISO 22400 for information concerning measurement of key performance indicators (KPIs), including OEE.

7.5.3.1 Admin.Parameter[#]

Data Type: Array of Data type Descriptor

The parameter tags associated to the local interface are typically used for as parameters that are displayed or used on the unit locally, for example from an HMI. These parameters can be used to display any quality, alarm, or machine downtime parameter. The parameters are typically limited to parameters related the unit. The extent of the array is the maximum number of parameters needed.

7.5.3.1.1 Admin.Parameter[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of Parameter

This is the arbitrary (user defined) ID value of the parameter. This is a non-descript value that can be used for any user tag requirements.

7.5.3.1.2 Admin.Parameter[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Parameter Names for the Machine Unit

The parameter name is used to describe the parameter number, and its associated value. An example parameter name may be CASES MADE, OPERATOR SHIFT, REJECTED PRODUCTS, etc. This ialso could be displayed on HMI screens. The array is typically needed for machines that have quality reporting or PDA (Production Data Acquisition) needs.

7.5.3.1.3 Admin.Parameter[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Parameter Variable Unit of Measure for the Machine Unit

The administration parameter unit of measure is used to describe the unit of measure of the parameter, and its associated value. An example parameter unit of measure may be CASES, PROD, PPM, etc. This also could be displayed on HMI screens.

7.5.3.1.4 Admin.Parameter[#].Value

Data Type: Real
Tag Descriptor: Structured Array of Parameter Values

The parameter value is used to specify a unit machine variable for use in the PDA or to be sent to an "information receiver". This also could be displayed on HMI screens. As an example:

Admin.Parameter[1].Value = 50019

An example of a machine unit process variable:

Admin.Parameter[1].Name = TOTAL PRODUCTION
Admin.Parameter[1].Unit = STAT
Admin.Parameter[1].Value = 50010.

7.5.3.2 Admin.Alarm[#]

Data Type: Alarm Structure
Descriptor: Array of Given Size for Machine Alarms

The alarm tags associated to the local interface are typically used as parameters that are displayed or used on the unit locally, for example from an HMI. These alarm parameters can be used to display any alarm, or machine downtime cause that is currently occurring in the system. The alarms

are typically limited to the machine unit. The extent of the array is the maximum number of alarms needed to be enunciated.

7.5.3.2.1 Admin.Alarm[#].Trigger

Data Type: Bool
Tag Descriptor: Alarm Message Trigger

Alarm trigger should be turned on only when the alarm is currently active.

7.5.3.2.2 Admin.Alarm[#].ID

Data Type: INT (32bit)
Tag Descriptor: Alarm Message Identification Number

The alarm ID number is a unique value assigned to each alarm.

7.5.3.2.3 Admin.Alarm[#].Value

Data Type: INT (32bit)
Tag Descriptor: Alarm Message Number

The alarm message number is a value that is associated to the alarm allowing for user specific detail or to break down the Alarm.ID to greater detail.

7.5.3.2.4 Admin.Alarm[#].Message

Data Type: String
Tag Descriptor: Alarm Message

The alarm message is the actual text of the alarm for those machines capable of providing string information.

7.5.3.2.5 Admin.Alarm[#].Category

Data Type: INT (32bit)
Tag Descriptor: Alarm Category

Alarm category is used to identity what type of alarm has occurred (e.g. electrical, mechanical, materials, utilities, etc.).

7.5.3.2.6 Admin.Alarm[#].DateTime[#]

Data Type: Date-Time Array
Tag Descriptor: Date and Time the Alarm Occurred

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

7.5.3.2.7 Admin.Alarm[#].AckDateTime[#]

Data Type: Date-Time Array
Tag Descriptor: Date and Time the Alarm was Acknowledged

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

The AlarmHistory array is reserved for alarms that have occurred and can be sorted in chronological order with the most recently occurring alarmed indexed as Admin.AlarmHistory[0].

Any unused elements should be set to zero (null) values

7.5.3.3 Admin.AlarmExtent

Data Type: INT (32bit)
Tag Descriptor: Extent of Alarm Array

The alarm extent is associated with the maximum number of alarms needed for the machine annunciation or reporting. This tag can be used by a remote machine to understand the extent of the alarm array, or locally to manage the use of the array.

7.5.3.4 Admin.AlarmHistory[#]

Data Type: Alarm Structure

Note: Alarm history uses the same data structure as Alarm. Tags such as the trigger bit are not typically used for history functionality.

Descriptor: Array of Given Size for Machine Fault Number and Messaging History

The alarm tags associated to the local interface are typically used for parameters that are displayed or used on the unit locally, for example for a HMI. These alarm history parameters can be used to display any alarm history, or machine downtime cause. The historical alarms are typically limited to the machine unit. The extent of the array is the maximum number of historical alarms needed.

7.5.3.4.1 Admin.AlarmHistory[#].Trigger

Data Type: Bool
Tag Descriptor: Alarm History Message Trigger

Alarm trigger should be turned on only when the alarm is currently active.

7.5.3.4.2 Admin.AlarmHistory[#].ID

Data Type: INT (32bit)
Tag Descriptor: Alarm Message Identification Number

The alarm ID number is a unique value assigned to each alarm.

7.5.3.4.3 Admin.AlarmHistory[#].Value

Data Type: INT (32bit)
Tag Descriptor: Alarm Message Number

The Alarm message number is a value that is associated to the alarm allowing for user specific detail or to break down the Alarm.ID to greater detail.

7.5.3.4.4 Admin.AlarmHistory[#].Message

Data Type: String
Tag Descriptor: Alarm Message

The alarm message is the actual text of the alarm for those machines capable of providing string information.

7.5.3.4.5 Admin.AlarmHistory[#].Category

Data Type: INT (32bit)
Tag Descriptor: Alarm Category

Alarm category is used to identity what type of alarm has occurred (e.g. electrical, mechanical, materials, utilities, etc.).

7.5.3.4.6 Admin.AlarmHistory[#].DateTime[#]

Data Type: Date-Time Array
Tag Descriptor: Date and Time the Alarm Occurred

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

7.5.3.4.7 Admin.AlarmHistory[#].AckDateTime[#]

Data Type: Date-Time Array
Tag Descriptor: Date and Time the Alarm was Acknowledged

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

The AlarmHistory array is reserved for alarms that have occurred and can be sorted in chronological order with the most recently occurring alarm indexed as Admin.AlarmHistory[0].

Any unused elements should be set to zero (null) values

7.5.3.5 Admin.AlarmHistoryExtent

Data Type: INT (32bit)
Tag Descriptor: Extent of Alarm History Array

The alarm history extent is associated with the maximum number of alarms needed to be archived or tagged as alarm history for the machine. This tag can be used by a remote machine to understand the extent of the alarm array, or locally to manage the array.

7.5.3.6 Admin.StopReason[#]

Data Type: Alarm Structure

Descriptor: Machine Stop Reason is typically used for "First Out Fault" Reporting and Other Stoppage Events. The stop reason is the first event captured during an abort, held, suspended or stop event.

7.5.3.6.1 Admin.StopReason.Trigger

Data Type: Bool
Tag Descriptor: Stop Reason Message Trigger

The stop reason trigger should be turned on only when the stop reason is currently active.

7.5.3.6.2 Admin.StopReason.ID

Data Type: INT (32bit)
Tag Descriptor: Stop Reason Message Identification Number

The stop reason ID number is a unique value assigned to each stop reason.

7.5.3.6.3 Admin.StopReason.Value

Data Type: INT (32bit)
Tag Descriptor: Stop Reason Value

The stop reason value number is a value that is associated to the stop reason allowing for user specific detail or to break down the StopReason.ID to greater detail.

7.5.3.6.4 Admin.StopReason.Message

Data Type: String
Tag Descriptor: Stop Reason Message

The stop reason message is the actual text of the stop reason for those machines capable of providing string information.

7.5.3.6.5 Admin.StopReason.Category

Data Type: INT (32bit)
Tag Descriptor: Stop Reason Category

Stop reason category is used to identity what type of stop reason has occurred (e.g. electrical, mechanical, materials, utilities, etc.).

7.5.3.6.6 Admin.StopReason.DateTime[#]

Data Type: Date-Time Array
Tag Descriptor: Date and Time the Stop Reason Occurred

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

7.5.3.6.7 Admin. StopReason.AckDateTime[#]

Data Type: Date-Time Array
Tag Descriptor: Date and Time the Stop Reason was Acknowledged

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

7.5.3.7 Admin.StopReasonExtent

Data Type: INT (32bit)
Tag Descriptor: Extent of Stop Reason Array

The stop reason extent is associated with the maximum number of stop reasons needed to be archived or tagged as stop reasons for the machine. This tag can be used by a remote machine to understand the extent of the alarm array, or locally to manage the array.

7.5.3.8 Admin.Warning[#]

Data Type: Alarm Structure
Descriptor: Machine warnings are for general events that do not cause the machine to stop, but may require operator action as a stoppage may be imminent.

7.5.3.8.1 Admin.Warning.Trigger

Data Type: Bool
Tag Descriptor: Warning Message Trigger

Warning trigger should be turned on only when the warning is currently active.

7.5.3.8.2 Admin.Warning.ID

Data Type: INT (32bit)
Tag Descriptor: Warning Message Identification Number

The warning ID number is a unique value assigned to each warning.

7.5.3.8.3 Admin.Warning.Value

Data Type: INT (32bit)
Tag Descriptor: Warning Value

The warning value number is a value that is associated to the warning allowing for user specific detail or to break down the Warning.ID to greater detail.

7.5.3.8.4 Admin.Warning.Message

Data Type: String
Tag Descriptor: Warning Message

The warning message is the actual text of the warning for those machines capable of providing string information.

7.5.3.8.5 Admin.Warning.Category

Data Type: INT (32bit)
Tag Descriptor: Warning Category

Warning category is used to identity what type of warning has occurred (e.g. electrical, mechanical, materials, utilities, etc.).

7.5.3.8.6 Admin.Warning.DateTime[#]

Data Type: Date-Time Array
Tag Descriptor: Date and Time the Warning Occurred

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

7.5.3.8.7 Admin.Warning.AckDateTime[#]

Data Type: Date-Time Array
Tag Descriptor: Date and Time the Warning was Acknowledged

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

7.5.3.9 Admin.WarningExtent

Data Type: INT (32bit)
Tag Descriptor: Extent of Warning Array

The warning extent is associated with the maximum number of warnings needed to be archived or tagged as warnings for the machine. This tag can be used by a remote machine to understand the extent of the warning array, or locally to manage the array.

7.5.3.10 Admin.ModeCurrentTime[#]

Data Type: INT (32bit)
Unit of Measure: sec
Tag Descriptor: Array of Timer Values

This tag represents the current amount of time (in sec) in any defined unit mode. The array index is equal to the designation of the unit machine mode values – defined in Status.UnitModeCurrent. The values roll over to 0 at 2,147,483,647.

7.5.3.11 Admin.ModeCumulativeTime[#]

Data Type: INT (32bit)
Unit of Measure: sec
Tag Descriptor: Array of Timer Values

This tag represents the cumulative amount of time (in sec) in any defined unit mode. The array index is equal to the designation of the unit machine mode values – defined in Status.UnitModeCurrent. The value is the cumulative elapsed time the machine has spent in each mode since its timers and counters were reset. The values roll over to 0 at 2,147,483,647.

7.5.3.12 Admin.StateCurrentTime[#,#]

Data Type: INT (32bit)
Unit of Measure: sec
Tag Descriptor: Array of Timer Values

This tag represents the current amount of time (in sec) in any defined state in any particular mode. The array index is equal to the designation of the unit machine mode values defined in Status.UnitModeCurrent, and the state values defined in Status.StateCurrent; such that the array index is [Status.UnitModeCurrent, Status.StateCurrent]. The values roll over to 0 at 2,147,483,647.

7.5.3.13 Admin.StateCumulativeTime[#,#]

Data Type: INT (32bit)
Unit of Measure: sec

Tag Descriptor: Array of Timer Values

This tag represents the cumulative amount of time (in sec) in any defined state in any particular mode since the last timer and counter reset was executed. The array index is equal to the designation of the unit machine mode values defined in Status.UnitModeCurrent, and the state values defined in Status.StateCurrent; such that the array index is [Status.UnitModeCurrent, Status.StateCurrent]. The values roll over to 0 at 2,147,483,647.

7.5.3.14 Admin.ProdConsumedCount[#]

Data Type: Array of Data Type ProdCount

This tag represents the material used/consumed in the production machine. An example of tag usage would be the number of bags consumed in a filler, or bagger packaging machine, or the amount of linear length used, or the number caps used. This tag can be used locally or remotely if needed. The extent of the array is typically limited to the number of raw materials needed to be counted. The array is typically used for unit machines that run multiple raw materials. This array may also be used to track the number of unfinished products entering a machine for processing, but typically Admin.ProdProcessedCount[#] is used for this.

7.5.3.14.1 Admin.ProdConsumedCount[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of ProdConsumedCount

This is the arbitrary (user defined) ID value of the consumed production material. This is a non-descript value that can be used for any user tag requirements. The ID value can be SKU or a user specific material identifier.

7.5.3.14.2 Admin.ProdConsumedCount[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Names in ProdConsumedCount

The name is used to literally describe the material ID, and its associated material. An example parameter name may be PRODUCT A BAGS, XYZ CAPS FOR ABC PRODUCT, etc. This also could be displayed on HMI screens. The array is typically needed for machines that have quality reporting or PDA (Production Data Acquisition) needs.

7.5.3.14.3 Admin.ProdConsumedCount[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Units in ProdConsumedCount

The unit tag is used to describe the names associated with a specific material used by the machine. An example process unit of measure name may be FT, CNT, KG, etc. This also could be displayed on HMI screens.

7.5.3.14.4 Admin.ProdConsumedCount[#].Count

Data Type: Int(32bit)
Tag Descriptor: Structured Array of Values

The count value is used as a variable for displaying information about the amount of consumed production material. The value is indexed upon the machine consuming a unit of the material defined by the ID and NAME. This could be displayed on HMI screens or higher level PDA systems. The counter rolls over to 0 at 2,147,483,648.

7.5.3.14.5 Admin.ProdConsumedCount[#].AccCount

Data Type: Int(32bit)
Tag Descriptor: Structured Array of Values

The accumulative count value is used as a variable for displaying information about the total amount of consumed production material. The value is indexed upon the machine consuming a unit of the material defined by the ID and NAME. This could be displayed on HMI screens or higher level PDA systems. This counter gives the user a non-resetting counter that may be used for OEE calculations. The counter rolls over to 0 at 2,147,483,648.

An example of the production consumption counter is the following:

Admin.ProdConsumedCount[1].ID = 546732
Admin.ProdConsumedCount[1].Name = LABELS FOR XYZ
Admin.ProdConsumedCount[1].Units = CNT
Admin.ProdConsumedCount[1].Count = 2305
Admin.ProdConsumedCount[1].AccCount = 14,995,100

The above describes consumed labels used by the machine for product XYZ as being 2305 since the last operator reset and 14,995,100 since the last accumulative counter reset.

7.5.3.15 Admin.ProdProcessedCount[#]

Data Type: Array of Data Type ProdCount

This tag represents the number of products processed by the production machine. An example of tag usage would be the number of products that were made, including all good and defective products. This tag can be used locally or remotely if needed. The extent of the array is typically limited to the number of products needed to be counted. The number of products processed minus the defective count is the number of products made by the machine. The array index of # = 0 can be reserved for the count of the number of units from the primary production stream.

7.5.3.15.1 Admin.ProdProcessedCount[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of ProdProcessedCount

This is the arbitrary (user defined) ID value of the processed products. This is a non-descript value that can be used for any user tag requirements. The ID value can be SKU or a user specific product identifier.

7.5.3.15.2 Admin.ProdProcessedCount[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Names in ProdProcessedCount

The name is used to literally describe the product ID. An example parameter name may be PRODUCT A, ABC PRODUCT, etc. This also could be displayed on HMI screens. The array is typically needed for machines that have quality reporting or PDA (Production Data Acquisition) needs.

7.5.3.15.3 Admin.ProdProcessedCount[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Units in ProdProcessedCount

The unit tag is used to describe the names associated with a specific product used by the machine. An example process unit of measure name may be FT, CNT, KG, etc. This also could be displayed on HMI screens.

7.5.3.15.4 Admin.ProdProcessedCount[#].Count

Data Type: Int(32bit)
Tag Descriptor: Structured Array of Values

The count value is used as a variable for displaying information about the amount of processed product. The value is indexed upon the machine processing a unit of the product defined by the ID and NAME. This could be displayed on HMI screens or higher level PDA systems. The counter rolls over to 0 at 2,147,483,648.

7.5.3.15.5 Admin.ProdProcessedCount[#].AccCount

Data Type: Int(32bit)
Tag Descriptor: Structured Array of Values

The accumulative count value is used as a variable for displaying information about the amount of processed product. The value is indexed upon the machine processing a unit of the product defined by the ID and NAME. This could be displayed on HMI screens or higher level PDA systems. This counter gives the user a non-resetting counter that may be used for OEE calculations. The counter rolls over to 0 at 2,147,483,648.

An example of the production processed counter is the following:

Admin.ProdProcessedCount[1].ID = 546732
Admin.ProdProcessedCount[1].Name = XYZ Product
Admin.ProdProcessedCount[1].Units = CNT
Admin.ProdProcessedCount[1].Count = 2305
Admin.ProdProcessedCount[1].AccCount = 2305

This describes the number of processed products the machine has made for product XYZ, i.e. 2305 products were processed by the machine.

7.5.3.16 Admin.ProdDefectiveCount[#]

Data Type: Array of Data Type ProdCount

This tag represents the product that is marked as defective in the production machine, to be used if applicable. An example of tag usage would be the number of products rejected or products that are termed defective. This tag can be used locally or remotely if needed. The extent of the array is typically limited to the number of products needed to be counted as defective. When this tag is used with Admin.ProdProcessedCount[#] the number of good products/well formed cycles made by the machine can be calculated. The array index of # = 0 can be reserved for the total cumulative rejected units from the primary production stream.

7.5.3.16.1 Admin.ProdDefectiveCount[#].ID

Data Type: INT (32bit)
Tag Descriptor: ID Value of ProdDefectiveCount

This is the arbitrary (user defined) ID value of the defective production material. This is a non-descript value that can be used for any user tag requirements. The ID value can be SKU or a user specific material identifier.

7.5.3.16.2 Admin.ProdDefectiveCount[#].Name

Data Type: STRING
Tag Descriptor: Structured Array of Names in ProdDefectiveCount

The name is used to literally describe the product ID, and its associated material. An example parameter name may be PRODUCT A, XYZ PRODUCT, etc. This also could be displayed on HMI screens. The array is typically needed for machines that have quality reporting or PDA (Production Data Acquisition) needs.

7.5.3.16.3 Admin.ProdDefectiveCount[#].Unit

Data Type: STRING[5]
Tag Descriptor: Structured Array of Units in ProdDefectiveCount

The unit tag is used to describe the names associated with a specific product processed by the machine. An example process unit of measure name may be FT, CNT, KG, etc. This also could be displayed on HMI screens. The array is typically used for unit machines that run multiple products.

7.5.3.16.4 Admin.ProdDefectiveCount[#].Count

Data Type: Int(32bit)
Tag Descriptor: Structured Array of Values

The count value is used as a variable for displaying information about the amount of defective product. The value is indexed upon the machine using a unit of the material defined by the ID and NAME. This could be displayed on HMI screens or higher level PDA systems. The counter rolls over to 0 at 2,147,483,648.

7.5.3.16.5 Admin.ProdDefectiveCount[#].AccCount

Data Type: Int(32bit)
Tag Descriptor: Structured Array of Values

The accumulative count value is used as a variable for displaying information about the amount of defective product. The value is indexed upon the machine using a unit of the material defined by the ID and NAME. This could be displayed on HMI screens or higher level PDA systems. This counter gives the user a non-resetting counter that may be used for OEE calculations. The counter rolls over to 0 at 2,147,483,648.

An example of the production defective counter is the following:

Admin.ProdDefectiveCount[1].ID = 546732
Admin.ProdDefectiveCount[1].Name = XYZ Product
Admin.ProdDefectiveCount[1].Units = CNT
Admin.ProdDefectiveCount[1].Count = 1005
Admin.ProdDefectiveCount[1].AccCount = 3001

which describes defective products made by the machine, i.e. 1005 defective products made since the last reset of count and 3001 defective products since the last reset of AccCount.

7.5.3.17 Admin.AccTimeSinceReset

Data Type: Int(32bit)
Unit of Measure: sec
Tag Descriptor: Accumulative Time since Last Reset

The tag represents the amount of time since the reset has been triggered. When a reset is triggered, all resettable tags are reset which can include:

UnitName.Admin.ModeCurrentTime[#]
UnitName.Admin.ModeCumulativeTime[#]
UnitName.Admin.StateCurrentTime[#,#]
UnitName.Admin.StateCumulativeTime[#,#]
UnitName.Admin.ProdConsumedCount[#].Count
UnitName.Admin.ProdProcessedCount[#].Count
UnitName.Admin.ProdDefectiveCount[#].Count
UnitName.Admin.AccTimeSinceReset

This value rolls over at 2,147,483,648 to 0. The tag can be used for simple OEE calculations as the definition of "scheduled production time". The simple OEE calculation is total amount of good products divided by total amount of good products that can be produced with the unit time, with the unit of time being scheduled production time.

7.5.3.18 Admin.MachDesignSpeed

Data Type: Real
Unit of Measure: Primary Packages/Minute
Tag Descriptor: Machine Design Speed

This tag represents 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 of the machine is designed to run in its 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.

7.5.3.19 Admin.StatesDisabled

Data Type: INT (32 Bit)
Tag Descriptor: States 1 to 17 of the PackML State Model can be disabled by turning on the corresponding bit in the INT. Note that some state transition rules may over-ride the disable bits. For example, if HOLDING is not disabled, then HELD cannot be disabled.

7.5.3.20 Admin.PLCDateTime

Data Type: Date-Time Array
Tag Descriptor: Current Date and Time of the Programmable Logic Controller

The data elements associated with time and date shall be in the format of year:month:day and hour:min:sec:usec per ISO 8601. The data type shall be an array of 32 bit integers.

Annex A — Weihenstephan harmonization

The Weihenstephan standards serve as a basis for an application standard for Production Data Acquisition Systems (PDAS) for primarily beverage filling systems. The standard describes how PDAS should be constructed in the filling area and how to configure the system in terms of hardware and software. Weihenstephan discusses a standard interface connection for the PDAS as well as control systems. The standard also elaborates on what information will be communicated from the machines to the PDAS. For more information, http://www.wzw.tum.de/lvt/englisch/Weihenstephaner_Standards_GB.html.

The Weihenstephan standard defines a functional programming method consistent with the definition of principles and methods in this report. Due to the nature of the standard, harmonization with the tag details called out in the report can be accomplished as follows:

Table A1 - Weihenstephan parameterized control tags

#	Command. Parameter[#].ID	Command. Parameter[#].Name	Command. Parameter[#].Unit	Command. Parameter[#].Value
1	30001	WS_Pallet_Type		
2	30002	WS_Crate_Type		
3	30003	WS_Bottle_Type		
4	30004	WS_Beer_Type		
5	30005	WS_Outfit_Type		
6	30011	WS_Pallet_Pattern		
7	30021	WS_Bot_Tank_No		
8	00061	Fillingbatch ID Set Low		
9	00062	Fillingbatch ID Set High		
10	00063	Fillingbatch-ID Current Low		
11	00064	Fillingbatch-ID Current High		
12	00065	Fillingorder-ID Set Low		
13	00066	Fillingorder-ID Set High		
14	00067	Fillingorder-ID Current Low		
15	00068	Fillingorder-ID Current High		
16	00069	SSCC Low		
17	00070	SSCC High		

Table A2 - Weihenstephan parameterized status tags

#	Status. Parameter[#].ID	Status. Parameter[#].Name	Status. Parameter[#].Unit	Status. Parameter [#].Value
1	40001	WS_Pressure		
2	40002	WS_Temperature		
3	40003	WS_Vol_Flow		
4	40004	WS_PU		
5	40005	WS_Conductance		
6	40011			
7	40012	WS_Redox_Hot_Water		
8	40021	WS_Temp_Main_Caustic		
9	40022	WS_Cond_Main_Caustic		
10	40031	WS_Temp_Caus_Spray		
11	40032	WS_Caus_Caus_Spray		
12	40041	WS_Press_Jet_Zone_XX		
13	40042	WS_Press_Jet_Zone_XX		
14	40043	WS_Press_Jet_Zone_XX		
15	40044	WS_Press_Jet_Zone_XX		
16	40045	WS_Press_Jet_Zone_XX		
17	40046	WS_Press_Jet_Zone_XX		
18	40047	WS_Temp_Hot_Water		
19	40048	WS_Press_Jet_Zone_XX		
20	40049	WS_Press_Jet_Zone_XX		
21	40050	WS_Press_Jet_Zone_XX		
22	40051	WS_Press_Bottling		
23	40052	WS_Temp_Bottling		
24	40061	WS_Press_HPI		
25	40062	WS_Temp_HPI		
26	40071	WS_Temp_Flod_Water		
27	40081	WS_Temp_Product		
28	40082	WS_Cond_Product		
29	40083	WS_PH_Product		
30	40084	WS_O2_Product		
31	40085	WS_CO2_Product		
32	40086	WS_Extr_Product		
33	40091	WS_Temp_Runback		
34	40092	WS_Cond_Runback		
35	40101	WS_Temp_Steril		
36	40111	WS_Vol_Flow_Det		
37	40121	WS_O_Press_Cleanroom		
38	40131	WS_Fill_Factor		
39	40141	WS_Temp_Glue		
40	40151	WS_Temp_Past_Zone_XX		
41	40152	WS_Temp_Past_Zone_XX		
42	40153	WS_Temp_Past_Zone_XX		
43	40154	WS_Temp_Past_Zone_XX		
44	40155	WS_Temp_Past_Zone_XX		
45	40156	WS_Temp_Past_Zone_XX		
46	40157	WS_Temp_Past_Zone_XX		
47	40158	WS_Temp_Past_Zone_XX		
48	40159	WS_Temp_Past_Zone_XX		
49	40160	WS_Temp_Past_Zone_XX		
50	40161	WS_Temp_Past_Zone_XX		

51	40162	WS_Temp_Past_Zone_XX
52	40163	WS_Temp_Past_Zone_XX
53	40164	WS_Temp_Past_Zone_XX
54	40165	WS_Temp_Past_Zone_XX
55	40166	WS_Temp_Past_Zone_XX
56	40167	WS_Temp_Past_Zone_XX
57	40168	WS_Temp_Past_Zone_XX
58	40169	WS_Temp_Past_Zone_XX
59	40170	WS_Temp_Past_Zone_XX
60	40171	WS_Spd_Conveyor
61	40181	WS_Freq_Freq_Conv_XX
62	40182	WS_Freq_Freq_Conv_XX
63	40183	WS_Freq_Freq_Conv_XX
64	40184	WS_Freq_Freq_Conv_XX
65	40185	WS_Freq_Freq_Conv_XX
66	40186	WS_Freq_Freq_Conv_XX
67	40187	WS_Freq_Freq_Conv_XX
68	40188	WS_Freq_Freq_Conv_XX
69	40189	WS_Freq_Freq_Conv_XX
70	40190	WS_Freq_Freq_Conv_XX
71	40191	WS_Freq_Freq_Conv_XX
72	40192	WS_Freq_Freq_Conv_XX
73	40193	WS_Freq_Freq_Conv_XX
74	40194	WS_Freq_Freq_Conv_XX
75	40195	WS_Freq_Freq_Conv_XX
76	40196	WS_Freq_Freq_Conv_XX
77	40197	WS_Freq_Freq_Conv_XX
78	40198	WS_Freq_Freq_Conv_XX
79	40199	WS_Freq_Freq_Conv_XX
80	40200	WS_Freq_Freq_Conv_XX
81	40201	WS_Freq_Freq_Conv_XX
82	40202	WS_Freq_Freq_Conv_XX
83	40203	WS_Freq_Freq_Conv_XX
84	40204	WS_Freq_Freq_Conv_XX
85	40205	WS_Freq_Freq_Conv_XX
86	40206	WS_Freq_Freq_Conv_XX
87	40207	WS_Freq_Freq_Conv_XX
88	40208	WS_Freq_Freq_Conv_XX
89	40209	WS_Freq_Freq_Conv_XX
90	40210	WS_Freq_Freq_Conv_XX
91	40211	WS_Freq_Freq_Conv_XX
92	40212	WS_Freq_Freq_Conv_XX
93	40213	WS_Freq_Freq_Conv_XX
94	40214	WS_Freq_Freq_Conv_XX
95	40215	WS_Freq_Freq_Conv_XX
96	40216	WS_Freq_Freq_Conv_XX
97	40217	WS_Freq_Freq_Conv_XX
98	40218	WS_Freq_Freq_Conv_XX
99	40219	WS_Freq_Freq_Conv_XX
100	40220	WS_Freq_Freq_Conv_XX
101	40221	WS_Freq_Freq_Conv_XX
102	40222	WS_Freq_Freq_Conv_XX
103	40223	WS_Freq_Freq_Conv_XX
104	40224	WS_Freq_Freq_Conv_XX
105	40225	WS_Freq_Freq_Conv_XX
106	40226	WS_Freq_Freq_Conv_XX

107	40227	WS_Freq_Freq_Conv_XX
108	40228	WS_Freq_Freq_Conv_XX
109	40229	WS_Freq_Freq_Conv_XX
110	40230	WS_Freq_Freq_Conv_XX
111	40231	WS_Spd_Freq_Conv_XX
112	40232	WS_Spd_Freq_Conv_XX
113	40233	WS_Spd_Freq_Conv_XX
114	40234	WS_Spd_Freq_Conv_XX
115	40235	WS_Spd_Freq_Conv_XX
116	40236	WS_Spd_Freq_Conv_XX
117	40237	WS_Spd_Freq_Conv_XX
118	40238	WS_Spd_Freq_Conv_XX
119	40239	WS_Spd_Freq_Conv_XX
120	40240	WS_Spd_Freq_Conv_XX
121	40241	WS_Spd_Freq_Conv_XX
122	40242	WS_Spd_Freq_Conv_XX
123	40243	WS_Spd_Freq_Conv_XX
124	40244	WS_Spd_Freq_Conv_XX
125	40245	WS_Spd_Freq_Conv_XX
126	40246	WS_Spd_Freq_Conv_XX
127	40247	WS_Spd_Freq_Conv_XX
128	40248	WS_Spd_Freq_Conv_XX
129	40249	WS_Spd_Freq_Conv_XX
130	40250	WS_Spd_Freq_Conv_XX
131	40251	WS_Spd_Freq_Conv_XX
132	40252	WS_Spd_Freq_Conv_XX
133	40253	WS_Spd_Freq_Conv_XX
134	40254	WS_Spd_Freq_Conv_XX
135	40255	WS_Spd_Freq_Conv_XX
136	40256	WS_Spd_Freq_Conv_XX
137	40257	WS_Spd_Freq_Conv_XX
138	40258	WS_Spd_Freq_Conv_XX
139	40259	WS_Spd_Freq_Conv_XX
140	40260	WS_Spd_Freq_Conv_XX
141	40261	WS_Spd_Freq_Conv_XX
142	40262	WS_Spd_Freq_Conv_XX
143	40263	WS_Spd_Freq_Conv_XX
144	40264	WS_Spd_Freq_Conv_XX
145	40265	WS_Spd_Freq_Conv_XX
146	40266	WS_Spd_Freq_Conv_XX
147	40267	WS_Spd_Freq_Conv_XX
148	40268	WS_Spd_Freq_Conv_XX
149	40269	WS_Spd_Freq_Conv_XX
150	40270	WS_Spd_Freq_Conv_XX
151	40271	WS_Spd_Freq_Conv_XX
152	40272	WS_Spd_Freq_Conv_XX
153	40273	WS_Spd_Freq_Conv_XX
154	40274	WS_Spd_Freq_Conv_XX
155	40275	WS_Spd_Freq_Conv_XX
156	40276	WS_Spd_Freq_Conv_XX
157	40277	WS_Spd_Freq_Conv_XX
158	40278	WS_Spd_Freq_Conv_XX
159	40279	WS_Spd_Freq_Conv_XX
160	40280	WS_Spd_Freq_Conv_XX

Table A3 - Weihenstephan parameterized administration tags

#	Admin. Parameter[#].ID	Admin. Parameter[#].Name	Admin. Parameter[#].Unit	Admin. Parameter[#].Value
1	50001	WS_Tot_Pallets		
2	50002	WS_Tot_Crates		
3	50003	WS_Tot_Crates_Full		
4	50004	WS_Tot_Crates_Empty		
5	50005	WS_Tot_Bottles		
6	50006	WS_Good_Bottles		
7	50007	WS_Dis_Bott_Cont		
8	50008	WS_Dis_Bott_Return		
9	50009	WS_Burst_Bottles		
10	50010	WS_Label		
11	50011	WS_Tot_Rej		
12	50012	WS_Rej_Wrong_Bottle		
13	50013	WS_Rej_Bottle_High		
14	50014	WS_Rej_Bottle_Low		
15	50015	WS_Rej_Bottle_Colour		
16	50016	WS_Rej_Def_Opening		
17	50017	WS_Rej_Def_Botton		
18	50018	WS_Rej_Scuffing		
19	50019	WS_Rej_Bottle_Closed		
20	50020	WS_Rej_Caustic		
21	50021	WS_Rej_Foreign_Obj		
22	50022	WS_Rej_Bottle_Under		
23	50023	WS_Rej_Bottle_Over		
24	50024	WS_Rej_Bottle_Clos		
25	50025	WS_Rej_Date_Coding		
26	50026	WS_Rej_Label_Fault		
27	50027	WS_Rej_Crate_Defect		
28	50028	WS_Rej_Crate_Colour		
29	50029	WS_Rej_Crate_Logo		
30	50030	WS_Rej_Comp_Empty		
31-100	50031-50100	OTHER REJECT CAUSES		
101	50101	WS_Cons_Clean_Water		
102	50102	WS_Cons_Hot_Water		
103	50103	WS_Cons_Steam		
104	50104	WS_Cons_Sterile_Air		
105	50105	WS_Cons_CO2		
106	50106	WS_Cons_Detergents		
107	50107	WS_Cons_Additives		
108	50108	WS_Cons_Lubricant		
109-150	50109-50150	OTHER CONSUMPTIONS		
151	50151	WS_Tot_Crates_Bad		
152	50152	WS_Bad_Cr_Miss_Bott		
153	50153	WS_Bad_Cr_High_Bott		
154	50154	WS_Bad_Cr_Low_Bott		
155	50155	WS_Bad_Cr_Colour		
156	50156	WS_Bad_Cr_Def_Handle		
157-200	50157-50200	OTHER REASONS BAD CRATES		
201	50201	WS_Pallets_Not_Comp		
202	50202	WS_Def_Pallets		
203	50203	WS_Not_Def_Pallets		
204	50211	WS_Quantity_Product		
205	50212	WS_Prod_Flow_Rate		
206-905	50301-50999	WS_Fill_Valve_XXX_YYY		

This page intentionally left blank.

Annex B (informative) – Overview of 2015 Technical Report changes

Contents

- ISA 88 History
- Summary of changes
- Overview of changes in definitions
- Overview in changes in models
  - What has been updated
  - What has remained unchanged
  - What has been added
  - What has been removed
- Other changes
- Summary

The technical report is being updated to simplify, clarify and make more complete

ISA 88 Brief History

- ISA 88.01 Batch Control Part 1: Models and Terminology
  - Originally complete (1995)
  - Updated (2010)
- Other sections approved
  - ISA 88.02 – Data structures and guidelines for languages (2001)
  - ISA 88.03 – General and site recipes (2003)
  - ISA 88.04 – Batch production records (2006)
- Other sections still in Draft
  - ISA 88.05 Implementation Models & Terminology for Modular Equipment Control
- Technical Reports Issued (latest revision)
  - TR88.95.01 Using ISA-88 and ISA-95 together (2008)
  - TR88.00.02 Machine and unit states: An implementation example of ISA-88 (2008)
  - TR88.0.03 Possible recipe procedure presentation formats (1996)

Summary of Changes

- Simplify – To enable easier adoption and application.
  - Identify Minimum Set of PackTags
  - Identify Minimum Set of States
  - Remove sections that are not central to machine design (sections 8, 9 and Appendix A)

- Clarify – To remove or improve confusing or unclear portions.
  - Clarify definition of Suspending & Holding
  - Clarify how the transition is made between Modes
  - Clarify what is done in Resetting and how Clear & Reset is initiated
  - Clarify how to communicate Blocked & Starved in PackTags
  - Add specific data structure for Date & Time tags
  - Eliminate contradictions between use of PackTags in Local HMI vs. Supervisory HMI
  - Change "Producing mode" to "Production" & expand User Defined Modes

- Make More Complete – Fill in gaps identified by those applying the standard.
  - Add Stop Reason to PackTags
  - Add Warnings to PackTags
  - Add PackML States Disabled PackTag
  - Add execution in memory based systems

The committee that assembled the changes believes these changes will speed adoption across industry, increase standardization and ultimately reduce costs for suppliers, original equipment manufacturers and end users of packing equipment.

Simplify - Minimum Set of PackTags

Table 7 : PackTags: Minimum Required for Information / Machine Monitoring

STATUS TAGS					TAGNAME	DATATYPE
UnitName					UnitName	PackMLxx.x
	Status				UnitName.Status	Status
		UnitModeCurrent			UnitName.Status.UnitModeCurrent	int (32bit)
		StateCurrent			UnitName.Status.StateCurrent	int (32bit)
		MachSpeed			UnitName.Status.MachSpeed	Real
		CurMachSpeed			UnitName.Status.CurMachSpeed	Real
		Equipment Interlock			UnitName.Status.EquipmentInterlock	Bool Structure [2]
			Blocked		UnitName.Status.EquipmentInterlock.Blocked	Bool
			Starved		UnitName.Status.EquipmentInterlock.Starved	Bool
ADMIN TAGS						
	Admin					
		ProdProcessed				
		ProdDefects				
		StopReason				

Table II: PackTags: Minimum Required for Supervisory Control

COMMAND TAGS					TAGNAME	DATATYPE
UnitName					UnitName	PackMLxx.x
	Command				UnitName.Command	Command
		UnitMode			UnitName.Command.UnitMode	int (32bit)
		UnitModeChangeRequest			UnitName.Command.UnitModeChangeRequest	Bool
		MachSpeed			UnitName.Command.MachSpeed	Real
		CmdCmd			UnitName.Command.CmdCmd	int (32bit)
		CmdChangeRequest			UnitName.Command.CmdChangeRequest	Bool

- Added Tables 7 & 8 to clearly identify the minimum set of PackTags required
- Added reference to tables 7&8 in section 7.5

Today PackTags is very comprehensive but in practice nobody is using the same set. The addition of a minimum set allows further standardization and clear compliance criteria in contrast to the original huge original list.

Simplify - Minimum Set of States

ISA-88.01 Procedural States	ISA-TR88.00.02 Equipment States				
	Value	Unit / Machine States	Wait	Acting	Minimum Required
<not defined>	1	Clearing		x	
Stopped	2	Stopped	x		x
<not defined>	3	Starting		x	
Idle	4	Idle	x		x
Paused	5	Suspended	x		
Running	6	Execute	x	x	x
Stopping	7	Stopping		x	
Aborting	8	Aborting		x	
Aborted	9	Aborted	x		x
Holding	10	Holding		x	
Held	11	Held	x		
Restarting	12	Unholding		x	
Pausing	13	Suspending		x	
<not defined>	14	Unsuspending		x	
<not defined>	15	Resetting		x	
<not defined>	16	Completing		x	
Complete	17	Complete	x		

Figure 2: Example States for Automated Machines

- Added column to figure 2 to identify the minimum set of states required
- Added reference to figure 2 to the bottom text of figure 3.

Simplify - Remove Sections Not Central

- Deleted
  - Section 8 "Software Implementation Examples"
  - Section 9 "OEE Implementation Examples"
  - Appendix A.1 "Alarm Codes"
- Examples should not be included in this document. Examples provided were not effective.
- The alarm codes examples have not been agreed to by member companies and have confused readers

Part of the issue with the original version was it was so large it intimidated those trying to understand it. This is part of an effort to reduce and eliminate things not central to what TR88 is.

Clarify - Definition of Suspending and Holding

- **Updated language describing Suspending to be due to External conditions**
  - This state shall be used when EXTERNAL process conditions do not allow the machine to continue producing, that is, the machine leaves EXECUTE due to upstream or downstream conditions on the line. This is typically due to a blocked or starved event. This condition may be detected by a local machine sensor or based on a supervisory system external command. While in the SUPENDING state, the machine is typically brought to a controlled stop and then transitions to SUSPENDED upon state complete. To be able to restart production correctly after the SUSPENDED state, all relevant process set-points and return status of the procedures at the time of receiving the Suspend command must be saved in the machine controller when executing the SUSPENDING procedure.
- **Updated language describing Holding to be due to Internal conditions –**
  - This state shall be used when INTERNAL machine conditions do not allow the machine to continue producing, that is, the machine leaves EXECUTE due to INTERNAL conditions. This is typically used for routine machine conditions that requires minor operator servicing to continue production. This state can be initiated automatically or by an operator and can be easily recovered from. An example of this would be a machine that requires an operator to periodically refill a glue dispenser or carton magazine and due to the machine design, these operations cannot be performed while the machine is running. Since these types of tasks are normal production operations, it is not desirable to go through aborting or stopping sequences, and because these functions are integral to the machine they are not considered to be "external". While in the HOLDING state, the machine is typically brought to a controlled stop and then transitions to HELD upon state complete. To be able to restart production correctly after the HELD state, all relevant process set-points and return status of the procedures at the time of receiving the Hold command must be saved in the machine controller when executing the HOLDING procedure.

Users, in practice, have been confused between these states and the original document did not specify clearly how they were different.
These definitions position Holding to be due to internal conditions and Suspending to be due to external conditions.

Clarify - How Transition is Made Between Modes

- Bullet 4 under section 5.1 was added to clarify how to transitions between modes must be done:
  - When a Unit Control Mode change takes place, a Machine State change is NOT allowed to occur at the same time. Mode Transitions must take place in a state that is common to both modes. This is necessary to avoid unintended machine sequences from taking place.

In applying the existing standards users were often confused about how to handle transition between modes. This clarifies how that is done.

Clarify - What Is Done In Resetting and How Reset and Clear are Initiated

- Resetting – Language clarified to only indicate safety devices would be energized. Removed reference to horn.
  - This state is the result of a RESET command from the STOPPED or complete state. Faults and stop causes are reset. RESETTING will typically cause safety devices to be energized and place the machine in the IDLE state where it will wait for a START command. No hazardous motion should happen in this state.
- Added foot note to Table 2 to identify it is common practice to use one button or device to initiate clearing and resetting.
  - It is common practice for Clearing and Resetting COMMANDS to be initiated using the same physical operator interface device.

In application of the original standard many users applied it this way to simplify the execution. This clarifies to drive more common implementations.

This was the biggest outage with the original version. Blocked and starved was not clearly called out. Nearly all packaging machines have this status and it had to be added to make it complete.

Clarify - Add Specific Data Structure for Date and Time

- Section 7.4 now provides detail on how to communicate date and time:
  - Date and Time – Data Type: INT (32 bit) Array
    - Array element 0 = Year
    - Array element 1 = Month
    - Array element 2 = Day
    - Array element 3 = Hour (24hr format)
    - Array element 4 = Min
    - Array element 5 = Sec
    - Array element 6 = USec (1/1,000,000 sec)
  - Time Display Format – ISO 8601:1988 24hr Time data type, beginning at 00:00:00.
  - Date Display Format – ISO 8601:1988 Date data type YYYY-MM-DD

- Other references updated to be consistent

In applying the standard the original text was not specific enough to execute consistently. This addresses that outage.

Complete - Add Stop Reason to PackTags

- This is the first out fault for the machine.
- Users have been applying this in practice
- Standard updated to reflect what's the best practice
  - Descriptor: Machine Stop Reason is typically used for "First Out Fault" reporting and other stoppage events. The stop reason is the event captured during an abort, held, suspended or stop event.

With packing equipment MES systems require a first out fault in order to record stop reason. That was missing from the original standard.

Complete - Add Warnings to PackTags

- This is for general events that require attention of the operator but do not stop the machine
  - Descriptor: Machine Warnings are for general events that do not cause the machine to stop, but may require operator action as a stoppage may be imminent.

The original standard did not have a provision for warnings. We need this so a line level operator can be warned when materials are expiring for example.

Other Items

- Clarify - Eliminate contradictions between use of PackTags in Local HMI vs. Supervisory HMI(7)
- Clarify - Change "Producing mode" to "Production" & expand User Defined Modes (31)
  - Figure 4 updated as shown
  - Update references to "Producing" to "Production"

ISA-88.01 Example Modes	ISA-TR88.00.02 Control Modes	
	Value	Unit / Machine Control Modes
<not defined>	0	Invalid
<not defined>	1	Production
<not defined>	2	Maintenance
<not defined>	3	Manual
<not defined>	04 – 31	User Definable

Figure 4: Unit/Machine Control Modes

- Add execution in memory based systems (22) – Added section 3.3 to enable execution on lower cost memory based PLC systems:
  - The minimum PackTag requirements that are identified in this document can be executed on a majority of PLC systems used by packaging machine manufacturers. It can also be implemented on systems such as check weighers, date coders, robotics, and other ancillary devices that may utilize proprietary control platforms. Even though the PackTags naming convention is using logical names, it can be implemented using register memory systems.
- Clarify - Eliminated one sentence from 4.1:
  - The Machine state is the result of previous activities that had taken place in the machine to change the previous state.
- Complete - References updated to current ISA-88.00.01-2010 and PLCopen TC5, added OMAC website

Other Items

- Simplified language describing Execute in Table 1 to be more clear –
  - Once the machine is processing materials it in the EXECUTE state. Different machine modes will result in specific types of EXECUTE activities. For example, if the machine is in the "Production" mode, the EXECUTE will result in products being produced, while in "Clean Out" mode the EXECUTE state refers to the action of cleaning the machine . #19 Edit Complete
- Clarify - Command and Status – Parameter. Value updated to be User Defined data time to provide more flexibility in application.
- Complete - Add PackML States Disabled PackTag – This tag will allow HMI's to dynamically update the state model to only show states that are active.
- Complete - Added .Trigger tag to Alarms, Warnings and Stop Reason to be Boolean that is true when the alarm or warning is active.
- Complete - Added .Category to Alarms, Warnings and Stop Reason so they could be grouped at the supervisory HMI or PLC

Other Items

- Clarify - Minor wordsmithing to make Forward, Abstract, Introduction more clear.
  - Foreword – First sentence clarified to state ISA SP88 defined a "series of standards"
  - Foreword – Added 3rd paragraph to describe why the revision was made and what was included.
  - The word "concepts" was added to the Abstract in the 3rd sentence after the word "standard"
- Clarify – 4.3 and 4.5.1 State model remained unchanged. Added text to describe how the state model is a sub-set of that shown in 88.00.01-2010
- Clarify – 5.1 struck the word "routine" with "method" to avoid vendor specific connotation.
- Clarify – 4.1 added paragraph capturing was changed in the states with this revision.
- Clarify 5 – Added paragraph capturing what was changed in the modes with this revision.

Other Items

- Clarify - A.2 was change to be Appendix 1 to avoid confusion.
- Clarify - 7.4 changed 4th bullet "string" to be any size but not to exceed 80 characters from always being 80 characters because this is how it is implemented in practice.
- Clarify - Tables 4 and 5 – Changed Data Type for Product tags to be "Product Structure" type instead of "Product" type to be more consistent with other data types. Same for Interface, Product, Ingredient and Alarm.
- Clarify - 7.4 changed 3 bullet to be Boolean Structure instead of Binary to be consistent with tables 4, 5 and 6
- Clarify – Table 6 and Section 7.5.3.18 changed Admin.PACDateTime to Admin.PLCDateTime to be consistent with other reference in the document
- Clarify – Clean up of minor misplaced spaces and minor typos.
- Clarify – Tables 4, 5, 6, 7 & 8 – Remove parameter data type and replace with existing descriptor type.

Other Items

- Clarify – Section 7.4.1 remove TimeStamp data type as it is no longer used.
- Clarify – Table 4 and 5 materialinterlocks change to be singular for consistency.
- Clarify – Table 6 added UnitName to AlarmHistoryExtent tag name.
- Clarify – Tables 2 & 3 move label to the top to be consistent with other tables
- Clarify – Tables 4 & 5 change structure types that were "descriptor" to "descriptor structure" to be consistent with other data type labels.
- Clarify – Changed several references to this document as a "standard" to say "technical report" instead.
- Clarify – Fixed formatting problem in middle of table A1
- Clarify – Fixed to typos in section 7.5.2.5

Other Items

- Clarify – included bit polarity for blocked and starved bits 7.5.2.10 & 7.5.2.11
- Complete – harmonized DateTime tag naming for alarm.datetime, warning.datetime, stopreason.datetime, alarmhistory.datetime
- Clarify – removed all references to dual states for consistency with S88
- Clarify – removed language suggesting states do actions, states describe behavior
- Clarify – deleted "state engine" reference from keywords
- Clarify – changed introduction wording from "explain functional state programming" to "define a standard state-based model for automated machines"
- Clarify – Removed OEE state descriptions from Figure 3. Also deleted last line of ABORTING description.

Summary

- TR88.02 has been improved to be simpler, more clear and more complete
- We have incorporated the current best approaches from users of how they are applying the standard in practice

Developing and promulgating sound consensus standards, recommended practices, and technical reports is one of ISA's primary goals. To achieve this goal the Standards and Practices Department relies on the technical expertise and efforts of volunteer committee members, chairmen and reviewers.

ISA is an American National Standards Institute (ANSI) accredited organization. ISA administers United States Technical Advisory Groups (USTAGs) and provides secretariat support for International Electrotechnical Commission (IEC) and International Organization for Standardization (ISO) committees that develop process measurement and control standards. To obtain additional information on the Society's standards program, please write:

ISA
Attn: Standards Department
67 Alexander Drive
P.O. Box 12277
Research Triangle Park, NC 27709

ISBN: 978-1-941546-65-9