Integrated Architecture™: Foundations of Modular Programming
Disclaimer

Because of the variety of uses for this information, the user of and those responsible for applying this information must satisfy themselves as to the acceptability of each application and use of the program. In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use of or application of this information. The illustrations, charts, and examples shown in this document are intended solely to illustrate the principles of Rockwell Automation products and some of the methods used to apply them. Particularly because of the many requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based upon the illustrative uses and applications.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this text.

Reproduction of the contents of this document, in whole or in part, without written permission of Rockwell Automation, Inc. is prohibited.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Modular Equipment Control Concepts</td>
<td>7</td>
</tr>
<tr>
<td>Process Application Implementations</td>
<td>23</td>
</tr>
<tr>
<td>Discrete Application Implementations</td>
<td>35</td>
</tr>
<tr>
<td>General Naming Conventions</td>
<td>51</td>
</tr>
<tr>
<td>Appendix A—Secondary Content</td>
<td>71</td>
</tr>
<tr>
<td>Appendix B—References</td>
<td>81</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Abbreviations and Acronyms

The following table lists the abbreviations and acronyms that may appear in this document.

**Acronyms and Abbreviations**

<table>
<thead>
<tr>
<th>Acronyms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>AOI</td>
<td>Add-On Instructions in Logix™ (Version 16 and later)</td>
</tr>
<tr>
<td>Auto</td>
<td>Automatic</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer-Aided Drafting (design software)</td>
</tr>
<tr>
<td>CIP</td>
<td>Clean In-Place, Common Industrial Protocol</td>
</tr>
<tr>
<td>CLX</td>
<td>Allen-Bradley® ControlLogix™ series of programmable controllers</td>
</tr>
<tr>
<td>CM</td>
<td>Control Module</td>
</tr>
<tr>
<td>CNB</td>
<td>ControlNet™ Bridge</td>
</tr>
<tr>
<td>CNET</td>
<td>ControlNet™ network</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma-separated Values (.CSV file extension)</td>
</tr>
<tr>
<td>DI</td>
<td>Discrete Input</td>
</tr>
<tr>
<td>DNB</td>
<td>DeviceNet™ Bridge</td>
</tr>
<tr>
<td>DNet</td>
<td>DeviceNet™ network</td>
</tr>
<tr>
<td>DO</td>
<td>Discrete Output</td>
</tr>
<tr>
<td>EM</td>
<td>Equipment Module</td>
</tr>
<tr>
<td>ENet</td>
<td>EtherNet/IP™ network</td>
</tr>
<tr>
<td>EP</td>
<td>Equipment Phase</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning</td>
</tr>
<tr>
<td>FT</td>
<td>Factory Talk™</td>
</tr>
<tr>
<td>GEM</td>
<td>Generic Equipment Mode</td>
</tr>
<tr>
<td>GEM/SECS</td>
<td>Generic Equipment Model / Semiconductor Equipment Communication Standards</td>
</tr>
<tr>
<td>GO</td>
<td>Global Object</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine Interface</td>
</tr>
<tr>
<td>ID</td>
<td>Identification</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IA</td>
<td>Rockwell Automation Integrated Architecture™</td>
</tr>
<tr>
<td>ISA</td>
<td>The International Society of Automation</td>
</tr>
<tr>
<td>MES</td>
<td>Manufacturing Execution System</td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer; a type of Rockwell Automation customer also referred to as a &quot;Machine Builder&quot;</td>
</tr>
<tr>
<td>OMAC</td>
<td>Organization for Machine Automation and Control</td>
</tr>
<tr>
<td>OPW</td>
<td>OMAC Packaging Workgroup</td>
</tr>
<tr>
<td>P&amp;ID</td>
<td>Piping and Instrumentation Diagram (or Drawing)</td>
</tr>
<tr>
<td>PAC</td>
<td>Programmable Automation Controller (such as ControlLogix™)</td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable Logic Controller (such as SLC500™ or PLC5™)</td>
</tr>
</tbody>
</table>
1.2 Intended Audience

This document is intended for use by anyone involved in the design, development, or operation of Rockwell Automation® Integrated Architecture™ products. These practices are based on tried and tested principles collected from subject matter experts (SMEs) within Rockwell Automation and concepts derived from key industry standards.

The intended audience for this document includes the following:

1. The following Rockwell Automation business units that create software solutions such as sample code, HMI faceplates, and engineering libraries for the purpose of product support and/or solution delivery:
   - Rockwell Automation Services & Solutions Business (SSB)
   - Rockwell Automation Drive Systems Business (DSB)
   - Rockwell Automation Customer Support & Maintenance (CSM)
   - Rockwell Software Automation & Software Business (A&S)
   - Rockwell Automation Global OEM (Original Equipment Manufacturers) Solutions Business

2. The following Rockwell Automation Partner Network members:
   - Solution Providers (SPs) or System Integrators (SIs)
   - Machine Builders (OEMs)
   - Encompass partners

3. Manufacturers and Processors (End Users)

1.3 Purpose

The primary purpose of this document is to increase the quality, consistency and re-usability of application software developments associated with Rockwell Automation Integrated Architecture solutions. This goal cannot be achieved by the use of this document alone; other standards and publications that are referenced in this document must also be adequately understood.
Modular programming guidelines are intended to enable the delivery of standardized programming structures, conventions, configurations, and strategies to Rockwell Automation customers. Refer to this document especially when you are developing re-usable application programming libraries.

Your active collaboration with other users of this document is critical to realizing its stated purpose. Sharing your results and feedback are also encouraged for the continuous improvement of this document.

1.4 Scope
This document is mainly focused on the programming of physical manufacturing “unit operations” such as assembly applications, conveyors, mixers, packaging units, process skids, robotic cells, tanks, and valve matrices. From an automation standpoint, the scope can be described as the unit (or machine), or skid level automation controller (PAC or PLC) and human-machine interface (HMI) programming which could be for a single unit or for a complex multi-unit system.

The scope of this document also includes content that establishes the modular programming requirements necessary at the unit level to enable batch and other supervisory software applications to monitor and control these assets in a flexible and consistent manner.

1.5 Benefits of Modular Programming
Modular programming benefits both manufacturers (end users) and developers (machine builders and system integrators). The benefits for both groups are described below.

1.5.1 Benefits for Manufacturers (End Users)
For market segments where modular programming is applicable, the business needs described in the “End User Business Need” column in the following table are typical for end users. Information on how this document can meet these business needs is provided in the included responses.

<table>
<thead>
<tr>
<th>End User Business Needs</th>
<th>Foundations of Modular Programming Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexible Production: Ability to adapt existing assets to new product requirements with minimal time and capital investment.</td>
<td>This document will discuss the separation of procedures and equipment and other concepts that increase modularity, which are key to developing flexible assets.</td>
</tr>
<tr>
<td>Improved Asset and Production Performance: Ability to facilitate continuous improvements through vertical integration of equipment assets to manufacturing information systems.</td>
<td>This document provides a common language to describe the behaviors and status of an asset, which is critical to developing standard information system interfaces and consistent metrics. This consistency also lends itself to improving the efficiency of operator and maintenance personnel.</td>
</tr>
<tr>
<td>Production Management: Ability of integrated systems to enable real time, demand-driven execution.</td>
<td>This document provides examples of standardized means for supervisory systems to acquire control of production assets for batch, continuous and discrete applications, which will in turn facilitate coordination with production management systems.</td>
</tr>
<tr>
<td>Security and Regulatory Compliance: End-to-end supply chain accountability for materials, energy, and the security of infrastructure and assets.</td>
<td>As one example, modular programming makes compliance with regulatory requirements such as software validation easier by reducing the scope of validation efforts to only the modified equipment modules (EMs) and not entire applications.</td>
</tr>
</tbody>
</table>
1.5.2 Benefits for Machine Builders and Solution Providers (Developers)

For the developers who serve market segments where modular programming is applicable, the following business needs are typical. Information on how this document can meet these business needs is provided in the included responses.

### Benefits for Machine Builders and Solution Providers

<table>
<thead>
<tr>
<th>End User Business Need</th>
<th>FOMP Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to Market (New Designs)</td>
<td>Using the standards and methods described in this document enables developers to concentrate on the functionality of machine-specific control modules (CMs) rather than on the mechanics of implementation, thereby reducing overall development time. It will also facilitate concurrent development which can, in turn, further accelerate design cycles.</td>
</tr>
<tr>
<td>Total Cost to Design, Develop, and Deliver Applications</td>
<td>In addition to the time savings described in the previous cell, use of the standards and methods described in this document will ultimately reduce cost during the entire project and product life cycles, which includes design, build, testing, startup, and long term support.</td>
</tr>
<tr>
<td>Machine Innovation, Throughput, and Performance</td>
<td>As one example, modular implementations make it easier to test software objects, as there is significantly less chance that new software modules or modification to existing software modules will adversely affect other unmodified modules. This encourages and facilitates continuous improvements while reducing the risk that changes can present.</td>
</tr>
</tbody>
</table>
2. Modular Equipment Control Concepts

This section introduces concepts that are essential to modular programming.

2.1 Introduction to ANSI/ISA-88.01

In choosing a standard upon which to base Rockwell Automation's modular programming standards, numerous options were considered, ranging from formal industry standards like the International Society of Automation’s (ISA's) Batch Control and the Semiconductor Equipment and Materials Institute’s (SEMI's) Generic Equipment Model / SEMI Equipment Communication Standards (GEM/SECS) to guidelines developed by several different Rockwell Automation business units and customers. While all had their respective advantages and disadvantages, the overwhelming consensus was that ANSI/ISA-88.01-1995 (R2006) had the most recognized and broadly adopted standard. Complementing ISA-88.01 was PackML, the ISA-TR88.00.02 technical report from the Organization for Machine Automation and Control (OMAC), which provided a much needed example of how to apply ISA-88.01 in discrete manufacturing segments. Due to existing demand from the Rockwell Automation customer base to achieve consistent practices across the manufacturing areas where these two standards are prevalent, the decision was made to base this document on both ISA-88.01 and PackML. This decision was based primarily on the desire to leverage a standard that is believed to have the broadest global acceptance and is most applicable to the market segments served by Rockwell Automation. It is also commonly believed that modular programming concepts can be applied in applications and industries beyond where the ISA standards are well known today.

Other factors also contributed to the decision to use the ISA-88 standard. ISA-88 has been a focal point for defining and automating batch process, continuous process, and discrete part manufacturing for many years; its acceptance within and beyond the batch processing community is widespread; and its adoption has had a positive impact on the development of modular programming, vertical integration practices, and diagnostics and debugging practices.

An example of ISA-88’s adaptability is the recognition of the OMAC PackML guidelines as a valid ISA-88 example. In August 2008, OMAC published the technical report ISA-TR88.00.02 entitled “Machine and Unit States: An Implementation Example of ISA-88,” which has been adopted by the ISA-88 standard committee (SP88) and is referenced in this document regarding discrete part manufacturing and other non-batch applications of the ISA-88 standard. While the report has strong influences from the packaging industry, end users and machine builders in segments other than packaging are also starting to adopt this methodology, which is available at [http://www.omac.org](http://www.omac.org) in the Packaging section.

2.1.1 ISA Terminology

- **ISA** refers to the International Society of Automation, an ANSI recognized Standards Development Organization (SDO).
- **ISA-88** refers to a specific set of ISA standards, of which ISA88.01 is a subset part of.
- **SP88** refers to the ISA working member group responsible for the creation and publication of the ISA-88 standards.
- For the remainder of this document, references to **ISA-88.01-1995 (R2006)** will be abbreviated as ISA-88.01.
- For the remainder of this document, references to **PackML** should be understood as references to the **ISA88-TR88.00.02** technical report.
2.2 Separation of Procedural Control and Equipment Control

This section describes one of the most important concepts of the ISA-88 standard and a primary component of modular programming: the separation of procedural control and equipment control.

To understand the control hierarchy, the Purdue Business Model in the following figure shows the hierarchy of a manufacturing system from the Enterprise Resource Planning and Manufacturing Execution Systems levels down to plant floor systems, equipment, and devices.

The separation of procedural control and equipment control typically occurs in between Level 2, where procedural control (such as recipes) resides, and Level 1 or Level 0, where equipment control and intelligent devices reside.

ISA-88.01 states that the logical separation of procedures and equipment provides the ability to separate product-specific definitions, instructions and information from processing equipment entities. In a given implementation there may or may not be a physical separation that matches this logical one. Examples given are as follows:

- A control recipe, or parts of it, may be downloaded to an embedded computing system that was supplied by the physical equipment manufacturer and also contains the equipment entity’s control equipment.
- A control recipe may run in a dedicated computing system that uses a network connection to communicate with the equipment entity and its equipment control.
- Both the control recipe and equipment procedural elements may be implemented in the same control system.
- All equipment interactions are performed manually by a person reading from a written recipe. In all cases, the logical separation exists.


2.3 ISA-88 Models and Terminology
ISA-88 provides three key models and terminology for defining and understanding the automation control requirements for manufacturing plants: the Process Model, the Physical Model and the Procedural Model.

2.3.1 The Process Model
The Process Model describes how to make a product independent of the plant, equipment and controls used to make it. Recipe Management is one of the key elements of the Process Model.

Because Section 2 of this document is concerned primarily with establishing common practices for the equipment and controls used in plant automation, the remainder of this section will focus on the physical model and procedural model.

For additional information about the process model, refer to the ISA-88.01 standard.

2.3.2 The Physical Model
The Physical Model (also known as the Equipment Model) describes a hierarchical organization of equipment and the basic control capabilities associated with that organization. The physical model is a representation of the plant’s equipment used to make the product.

More information about the physical model and terminology that is pertinent to modular programming will be discussed in Section 2.4.

2.3.3 The Procedural Model
The Procedural Model describes a multi-tiered, hierarchical model that defines the process capability and automation control in relation to the Physical Model to perform a task. The Procedural Model is a representation of how to use the equipment (described in the physical model) to make the product.

More information about the procedural model and terminology that is pertinent to modular programming will be discussed in Section 2.5.

2.4 The Physical Model
The Physical Model is used to understand the automation components within a given environment, and thereby determine modular areas and component interactions. Interfaces to the components of the Physical Model provide the means to use all of the capabilities of the equipment. After the Physical Model is created, the Procedural Model is determined. The procedural control laid out in the Procedural Model is used to direct the equipment components, via the component interfaces, to perform the specific tasks out of the available capabilities, needed to produce a given product.
Although the Purdue Business Model displays the hierarchy of a complete manufacturing system, it does not provide enough detail to create application software for the lower levels of the manufacturing system. To clarify these details, ISA-88 created the Physical Model shown in the following figure.

2.4.1 Physical Model Terminology

The terminology defined in this section refers to the physical equipment, the software used for control of the physical equipment, or the equipment and control software together as a single entity.

The hierarchy of the Physical Model, from top to bottom, is as follows:

**Enterprise**: The company that owns the facilities.

**Site**: The location of a single facility.

**Process cell** (or **production line**): A collection of one or more units linked together to perform a single task or multiple tasks of the process for one or more products in a defined sequence. A process cell contains all of units, EMs, and CMs that are required to make one or more batches or lots.
**Unit** (or *machine*): A collection of related CMs and EMs that can carry out one or more processing activities. The *unit* corresponds to the logical grouping of mechanical and electrical assemblies that historically has been called a *machine*. It should be understood that in discrete applications, the term Machine is commonly used to describe a complete piece of equipment as provided by a machine builder, and could technically consist of more than one unit as defined by the standard. A *unit*, according to the ISA-88.01 definition, acts on one batch or lot at any given time on a process application; however, on a discrete machine there is a difference in that multiple batches, typically referred to a products or parts, can be acted upon at the same time.

**Equipment Module (EM):** A functional group of EMs, CMs, or both that can carry out a finite number of activities.

The primary purpose of control in an EM is to coordinate the functions of other EMs and lower-level CMs. An EM may be commanded by a process cell, unit, operator, or by another EM.

An EM can be part of a unit, or it can be a common resource within a process cell. A common resource can be either exclusive-use, where only one controlling entity can take ownership at a time, or a shared-use resource, where multiple controlling entities can share EM ownership. The EM’s control interface will usually include a method to establish which higher-level entity (or entities in the case of share-use) has ownership of the EM at any given point, in order to prevent conflicting commands from multiple sources. The EM combines all the physical processing and control equipment required to perform the manufacturing process. The scope of the EM is defined by the finite tasks that it is designed to carry out.

Some examples of EMs are as follows:
- A valve matrix used for material transfer between units (shared resource of process cell)
- A level control for a tank (EM within a specific unit)
- A vertical form-fill-seal machine’s “sealing jaws control” (EM within a discrete unit)

**Control module (CM):** A regulating device, state-oriented device or a combination thereof (typically, a collection of sensors, actuators, and other CMs) that is operated as a single device. A CM typically controls or monitors one device state such as a motor running or stopped, or a valve open or closed, or one process variable such as a flow rate, level, pressure, count, or quantity.

Control at this level usually directly manipulates actuators and other CMs. A CM can direct commands to other CMs, or to actuators that have been configured as part of the CM. Equipment-specific manipulation of CMs and actuators control the process. The CM is the lowest level of equipment in the Physical Model that can carry out basic control.

What follows is a list of devices or functions where CMs can be applied:
- An individual sensor or actuator
- A state-oriented device that consists of an on/off automatic block valve with position feedback switches and that is operated via the set point of the device
- A header that contains several on/off automatic block valves that coordinates the valves to direct flow to one or several destinations based upon the set point directed to the header CM
- A servo-controlled electronic gear or cam function (that is, a discrete unit), including its interlock and permissives.

The following list contains some typical CMs that you might find in a programming library:
- Analog output
- Analog input with scaling and alarms
• Reversing motor
• Variable speed drive
• Solenoid-operated 2-state valve
• Motor-operated 2-state valve
• PID with standard modes and deviation alarms

2.4.2 Equipment Control Terminology

ISA-88 defines three types of control that are needed in automated manufacturing: **basic**, **coordination** and **procedural**.

• **Basic control** is dedicated to establishing and maintaining a specific state of equipment and process. It may include regulatory control, monitoring, interlocking, exception handling, repetitive discrete control, or repetitive sequential control.
  
  • Basic control in an EM is generally performed by regulatory control and discrete control in CMs within the EM. CMs contain basic control.
  
  • Basic control is primarily either regulatory- or state-oriented, although in some cases it is both. It may also include conditional logic. An example of basic control is opening a valve when the temperature is within limits and the downstream valve is open. Regulatory control is dedicated to maintaining one or more process variables at or near a desired value.
  
  • Complex control strategies such as multi-variable control, model-based control, and artificial intelligence techniques also fit into this category of regulatory control. State-oriented control refers to setting the state of a piece of equipment as opposed to the state of a process variable or variables. A state-oriented device, which defines a processing sequence that is independent of the product, has a finite number of states. CMs can contain exception handling.

• **Coordination control** directs, initiates, and modifies the execution of procedural control and the utilization of equipment entities. This can include supervising availability of equipment, allocating equipment for batches, and propagating mode and state changes.

  Coordination control in an EM or in a CM can include the following:
  
  • Algorithms for propagating modes and faults
  
  • Algorithms for arbitrating requests from units to acquire and release common resources
  
  • Algorithms for limiting the number of users of a limited capacity shared resource
  
  • Coordination of the EM's component parts

• **Procedural control** (also referred to as **phase control**) directs equipment-oriented processes based on the procedural control model. EMs may execute equipment phases, but they cannot execute higher-level procedural elements. CMs do not perform procedural control.

2.4.3 Process Cell Classification

This section discusses the classification of process cells by the number of different products manufactured in the process cell and by the physical structure of the equipment used in the manufacturing.

2.4.3.1 Classification by Number of Products

A process cell is classified as single-product or multi-product based on the number of products planned for production in that process cell.
A single-product process cell produces the same product in each batch. Variations in procedures and parameters are possible in a single-product process cell. For example, variations may occur in order to compensate for differences in equipment, substituted raw materials, changes in environmental conditions, or simply to optimize the process.

A multi-product process cell produces different products by utilizing one of the following three methods of production:

- The products are produced with the same procedure but using different formula values (that is, by varying materials and/or process parameters). Sometimes products produced in this way are referred to as grades of a product.
- The products are produced by using different procedures.
- The products that are produced are grouped into separate families. Each family uses a different procedure, but products within a family share common procedure and have different formula values.

### 2.4.3.2 Classification by Physical Structure

The order of equipment actually used or expected to be used by a specific batch is called the path. A process cell is classified as single-path, multiple-path, or network based on its physical structure. Regardless of which structure is used, several batches may be in progress at the same time (in different units), multiple input materials may be used, multiple finished materials may be generated, and units may share input material sources and product storage.

The different physical structures are described below. A **single-path structure** is a group of units through which a batch passes sequentially. A single-path structure could be a single unit, such as a reactor, or several units in sequence. The following figure shows an example of a single-path structure.

![Figure 2-3. Single-Path Structure (Source: ISA-88.01 Draft 15)](image)

A **multiple-path structure** consists of multiple single-path structures in parallel with no product transfer between them. Although units within a multi-path structure may be physically similar, it is possible to have paths and units that are of radically different physical design. The following figure shows an example of a multiple-path structure.

![Figure 2-4. Multiple-Path Structure (Source: ISA-88.01 Draft 15)](image)
In a **network structure**, the paths may be either fixed or variable. When the paths are fixed, the same units are used in the same sequence. When the path is variable, the sequence may be determined at the beginning of the batch or as the batch is being produced. Alternatively, the path could be flexible; for example, a batch would not have to start at either Unit 1 or Unit 3, but could instead start with any unit and take multiple paths through the process cell. The following figure shows an example of a network structure.

The following figure shows an example of a network structure.

![Network Structure](image)

**Figure 2-5. Network Structure (Source: ISA-8801 Draft 15)**

### 2.5 The ISA-Procedural Model

The procedural model describes a multi-tiered, hierarchical model that defines the process capability and automation control in relation to the physical model to perform a task. The procedural model is a representation of **how to use** the equipment (described by the physical model) to make the product.
To demonstrate how control of the manufacturing process relates to the physical equipment, ISA-88 created the procedural control model shown in the figure below. The following figure shows the hierarchical relationships of the supervisory procedural control within the manufacturing process.

![Figure 2-6. The Procedural Model (Source: ISA dS88.01 Draft 4A)](image)

ISA-88 then combines the models in Figure 2-2 and Figure 2-6 to create the general model shown in Figure 2-7, which reflects the complete hierarchy of control and equipment, as well as the vertical separation between process controls and procedural controls. It is important to note that not all manufacturing processes require that the procedural control reside in the physical equipment.

![Figure 2-7. Physical and Procedural Models Combined (Source: ISA dS88.01 Draft 4A)](image)
In either a distributed or a flexible process, procedural control can reside outside the equipment in what is called the control recipe. Examples of this type of manufacturing include large batch systems, material handling systems, and automotive assembly systems. Use of the control recipe enables end users with different automation requirements to fully leverage the concept of separating procedural control and equipment control. Distributed and flexible control occurs in the procedural control.

The vertical lines used in many of the ISA-88 diagrams in this document are intended to illustrate where procedural elements are physically located in the automation system. The right side of the line (equipment control area) depicts items that are located in the unit controller, typically an industrial automation controller (PLC or PAC) and human-machine interface (HMI) dedicated to the physical equipment. The left side of the line is typically an external personal computer (PC) or server-based application equipment that is responsible for the functionality of the control recipe.

The following figure shows a batch manufacturing process in which the entire procedural control resides within the control recipe.

![Diagram](Image)

**Figure 2-8. The control recipe (Source: ISA d88.01 Draft 4A)**

The batch manufacturing process shown in this figure creates greater flexibility in the manufacturing process, and lets the end user determine which physical equipment to use at any step in the process.

In the following section, the details of what happens in the Equipment Phase are discussed. In Section 3, manufacturing examples that show how Rockwell Automation implements these concepts in traditional process applications are discussed.
2.5.1 Procedural Model Terminology

The ISA-88 procedural model defines how the controls should function on a piece of production machinery.

The hierarchy of the procedural model, from top to bottom, is as follows:

**Procedure**: The general strategy for production within a process cell. A procedure is made up of Unit Procedures.

**Unit Procedure**: A production sequence. Unit procedures are made up of Operations.

**Operation**: The single sequence necessary for the initiation, organization, and control of Phases. Operations are made up of Phases.

**Phase**: The lowest level of a procedure that can accomplish a process-oriented task. ISA-88 provides the following examples to explain a phase:

- A phase may be subdivided into smaller parts.
- A phase can issue one or more commands or cause one or more of the following actions:
  - Enabling and disabling regulating and state-oriented types of basic control, and specifying their set points and initial output values
  - Setting, clearing, and changing alarm and other limits
  - Setting and changing controller constants, controller modes, and types of algorithms
  - Reading process variables, such as the gas density, gas temperature, and volumetric flow rate from a flow-meter, and calculating the mass flow rate through the flow-meter
  - Conducting operator authorization checks
- The execution of a phase may result in any or all of the following:
  - Commands to EMs or CMs (basic control)
  - Commands to other phases, in either the same or another equipment entity
  - The collection of data
- The intent of a phase is to cause or define a process-oriented action, while the logic or set of steps that make up a phase are equipment-specific. Examples of a phase include the following:
  - Add Ingredient
  - Agitate
  - Heat

The ISA-88 standard also provides an example for implementing consistent procedural control strategies using what are called **modes** and **states**, which are utilized extensively in this document and in the Rockwell Automation implementations described in Sections 3 and 4. For details on other control functions that are defined by the standard, refer to the ISA-88 section on “Control Activities.”

**Modes and States**: In the preceding sections, models describing equipment entities and procedural elements have been defined. In these models, transitions for procedural elements and for equipment entities occur within each hierarchical level. The status of equipment entities and of procedural elements may be described by their modes and states. Modes specify the manner in which these transitions take place; states specify their current status.

In the following sections, the ISA-88 examples for the modes, the state model, and the associated commands that cause transitions in the state model will be discussed.
2.5.2 Procedural Control Modes

Equipment entities and procedural elements may have modes. The example modes below are described by ISA-88 in relation to batch control. The processing behavior of an equipment entity may be determined by evaluating the modes of both the associated procedural elements and the equipment entity itself.

A mode determines how equipment entities and procedural elements respond to commands and how they operate. In procedural elements, the mode determines how the procedure will progress and who can affect that progression. In a CM that contains basic control functions -for example, an automatic block valve-the mode determines both the mechanism used to drive the valve position and who or what (for example, an operator or another device) may manipulate it to change its state.

The ISA-88 standard uses an example of three modes—automatic, semi-automatic and manual—for procedural elements, and two modes—automatic and manual—for equipment entities. A CM contains basic control functions and has automatic and manual modes, whereas a unit running procedural control also has a semi-automatic mode.

The ISA-88 standard neither precludes additional modes nor requires the use of the defined modes. The functionality of the modes presented is considered to be generally useful in most batch applications. By naming the modes and including them in the standard, a defined set of terms that can be used when communicating on process control issues is documented.

The following table describes the ISA-88 procedural control modes.

<table>
<thead>
<tr>
<th>Procedural Control Modes</th>
<th>Behavior</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic (Procedural)</td>
<td>Transitions within a procedure are carried out without interruption as appropriate conditions are met.</td>
<td>Operators may pause the progression, but may not force transitions.</td>
</tr>
<tr>
<td>Automatic (Basic Control)</td>
<td>Equipment entities are manipulated by their control algorithm.</td>
<td>Equipment entities cannot be manipulated directly by the operator.</td>
</tr>
<tr>
<td>Semi-Automatic (Procedural Only)</td>
<td>Transitions within a procedure are carried out on manual commands as appropriate conditions are fulfilled.</td>
<td>Operators may pause the progression or re-direct the execution to an appropriate point. Transitions may not be forced.</td>
</tr>
<tr>
<td>Manual (Procedural)</td>
<td>Procedural elements within a procedure are executed in the order specified by an operator.</td>
<td>Operators may pause the progression or force transitions.</td>
</tr>
<tr>
<td>Manual (Basic Control)</td>
<td>Equipment entities are not manipulated by their control algorithm.</td>
<td>Equipment entities may be manipulated directly by the operator.</td>
</tr>
</tbody>
</table>

2.5.3 Procedural States and Commands

This section introduces the states and commands that are used in the ISA-88 standard. States are used to describe the current status of a machine, and commands are used to cause the machine to transition to a specific state as requested by either a higher-level programmed request or by an operator action.
The following table describes example ISA-88 procedural control states.

### Example Procedural Control States (Source: ISA-88.01 2006)

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>The procedural element is waiting for a START command that will cause a transition to the RUNNING state.</td>
</tr>
<tr>
<td>RUNNING</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>COMPLETE</td>
<td>Normal operation has run to completion. The procedural element is now waiting for a RESET command, which will prompt a transition to IDLE.</td>
</tr>
<tr>
<td>PAUSING</td>
<td>The procedural element or equipment entity has received a PAUSE command. This will cause the procedural element to stop at the next defined safe or stable stop location in its normal RUNNING logic. Once stopped, the state automatically transitions to PAUSED.</td>
</tr>
<tr>
<td>PAUSED</td>
<td>Once the procedural element has paused at the defined stop location, the state changes to PAUSED. This state is usually used for short-term stops. A RESUME command causes transition to the RUNNING state, resuming normal operation immediately following the defined stop location.</td>
</tr>
<tr>
<td>HOLDING</td>
<td>The procedural element has received a HOLD command and is executing its HOLDING logic to put the procedural element or equipment entity into a known state. If no sequencing is required, then the procedural element or equipment entity transitions immediately to the HELD state.</td>
</tr>
<tr>
<td>HELD</td>
<td>The procedural element has completed its HOLDING logic and has been brought to a known or planned state. This state is usually used for a long-term stop. The procedural element or equipment entity is waiting for a further command to proceed.</td>
</tr>
<tr>
<td>RESTARTING</td>
<td>The procedural element has received a RESTART command while in the HELD state. It is executing its restart logic in order to return to the RUNNING state. If no sequencing is required, then the procedural element or equipment entity transitions immediately to the RUNNING state.</td>
</tr>
<tr>
<td>STOPPING</td>
<td>The procedural element has received a STOP command and is executing its STOPPING logic, which facilitates a controlled normal stop. If no sequencing is required, then the procedural element or equipment entity transitions immediately to the STOPPED state.</td>
</tr>
<tr>
<td>STOPPED</td>
<td>The procedural element or equipment entity has completed its STOPPING logic. The procedural element or equipment entity is waiting for a RESET command to transition to IDLE.</td>
</tr>
<tr>
<td>ABORTING</td>
<td>The procedural element has received an ABORT command and is executing its ABORT logic, which is the logic that facilitates a quicker, but not necessarily controlled, abnormal stop. If no sequencing is required, then the procedural element transitions immediately to the ABORTED state.</td>
</tr>
<tr>
<td>ABORTED</td>
<td>The procedural element has completed its ABORTING logic. The procedural element is waiting for a RESET command to transition to IDLE.</td>
</tr>
</tbody>
</table>

The following table describes the example ISA-88 procedural control commands.

### Example Procedural Control Commands (Source: ISA-88.01 2006)

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>This command orders the procedural element to begin executing the normal RUNNING logic. This command is only valid when the procedural element is in the IDLE state.</td>
</tr>
<tr>
<td>STOP</td>
<td>This command orders the procedural element to execute the STOPPING logic. This command is valid when the procedural element is in the RUNNING, PAUSING, PAUSED, HOLDING, HELD, OR RESTARTING state.</td>
</tr>
</tbody>
</table>
Example Procedural Control Commands (Source: ISA-88.01 2006)

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOLD</td>
<td>This command orders the procedural element to execute the HOLDING logic. This command is valid when the procedural element is in the RUNNING, PAUSING, PAUSED or RESTARTING state.</td>
</tr>
<tr>
<td>RESTART</td>
<td>This command orders the procedural element to execute the RESTARTING logic to safely return to the RUNNING state. This command is only valid when the procedural element is in the HELD state.</td>
</tr>
<tr>
<td>ABORT</td>
<td>This command orders the procedural element to execute the ABORTING logic. The command is valid in every state except for IDLE, COMPLETED, ABORTING and ABORTED. This command causes a transition to the IDLE state. It is valid from the COMPLETE, ABORTED, and STOPPED states.</td>
</tr>
<tr>
<td>RESET</td>
<td>This command causes a transition to the IDLE state. It is valid from the COMPLETE, ABORTED, and STOPPED states.</td>
</tr>
<tr>
<td>PAUSE</td>
<td>This command orders the procedural element to pause at the next programmed pause transition within its sequencing logic and await a RESUME command before proceeding. This command is only valid in the RUNNING state.</td>
</tr>
<tr>
<td>RESUME</td>
<td>This command orders a procedural element that has PAUSED at a programmed transition as the result of either a PAUSE command or a SINGLE STEP mode to resume execution. This command is only valid when the procedural element is in the PAUSED state.</td>
</tr>
</tbody>
</table>

2.5.4 Procedural State Model and Transitions

There are two ways in which state transitions can occur. First, the state may transition when the control system receives one of the commands listed in the preceding table. Also, as stated in ISA-88, example states ending with ING are transient states. If their logic completes normally, then a state transition to the state listed in the following table under NO COMMAND END STATE occurs. For example, if the RUNNING state completes normally, then that state automatically transitions to the COMPLETE state. Execution of the transient states is governed by the mode.

The following shows the example ISA-88 procedural control state transition matrix.

![State Transition Matrix](image-url)
The following diagram shows an example state transition diagram from ISA-88. This diagram is derived from first three states in the above matrix (Idle, Running and Complete); therefore, only the transitions associated with these three states are visible. For clarity, “State Complete” (SC) has been added to the ISA-88 diagram.

Figure 2-10. Example State Transition Diagram (Source: ISA-88.01 2006)

Sections 3 and 4 show how the concepts discussed in Section 2 are utilized to define procedural control strategies for process and discrete applications, and includes examples of how these strategies can be implemented on the Rockwell Automation Integrated Architecture platform.
3. Process Application Implementations

This section demonstrates how Rockwell Automation has adapted the ISA-88 concepts discussed in Section 2 have been specifically adapted for process applications.

Section 2 described basic concepts derived from ISA-88.01 and the ISA88-TR88.00.02 Technical Report. This section discusses methods for implementing the concepts introduced in Section 2 on the Rockwell Automation Integrated Architecture platform. In this section, RSLogix5000™ application development software, which includes a feature called PhaseManager™, is used as a method of implementing modular programming concepts for process applications.

3.1 Applying ISA-88.01

Section 4 describes adaptations to the examples provided in ISA-88 that were necessary based on Rockwell Automation's experience with the standard, and our knowledge of the best practices for implementing it on applications supported by Rockwell Automation and using the Integrated Architecture platform.

3.2 Physical Model (Process Example)

To see an illustration of the complete ISA-88 physical model, refer to Section 2.4.

The following figure shows the top levels of the physical model, from Enterprise down to Units.
3.2.1 Process Application Physical Model Example

The following figure, a diagram of a typical dairy plant physical layout, shows the top levels of the physical model, from Enterprise down to Units. The parts of this drawing that can be directly associated to the physical model are listed in the table that follows the figure.

![Figure 3-1. Example, Process Application Physical Model](image)

Physical Model Descriptions

<table>
<thead>
<tr>
<th>ENTERPRISE</th>
<th>In this example, the figure shows one of the company’s sites.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SITE</td>
<td>The overall dairy plant is the site.</td>
</tr>
<tr>
<td>AREA</td>
<td>The dairy plant is split into three areas: Receiving, Processing, and Packaging.</td>
</tr>
<tr>
<td>PROCESS CELL</td>
<td>Several process cells are identified with arrows.</td>
</tr>
<tr>
<td>UNIT</td>
<td>Several units are identified with arrows.</td>
</tr>
</tbody>
</table>

For additional details on the ISA-88 physical model, refer to Section 2.4. For purposes of the Rockwell Automation implementation example in this section, no adaptations to ISA-88 are noted.

3.3 Procedural Control Model (Process Example)

For an illustration of the complete ISA-88 procedural model, refer to Section 2.5.
As we begin identifying some of the adaptations to the ISA-88 standard for process applications, it becomes evident that most of the variations from the ISA-88 examples reside in how procedural control is carried out. Procedural control is where procedural modes, states, and commands interface with equipment control.

### 3.3.1 Procedural Control Modes

For complete details on the ISA-88 Mode definitions, refer to Section 2.5. For purposes of the Rockwell Automation implementation example in this section, no adaptations to ISA-88 mode definitions are noted. However, there are some differences between process and discrete applications that require some clarification.

The use of control modes to carry out operations in discrete applications may differ from how control modes are utilized in process applications. In process applications, for example, modes are commonly used at the EM or CM level for things such as a valve Hand-Off-Auto control, in which the mode determines who or what (for example, the operator or the program) has control of the valve.

Control modes for discrete machines are typically implemented at the Equipment Operations level of the procedural model to carry out operations and phases. It is important to qualify the use of certain terminology when adapting the ISA-88 standards to industries that may have different implementations and terminologies. As an example, previously mentioned usages of the word mode should be considered as being ownership modes or operational modes.

### 3.3.2 Procedural Control States and Commands

The procedural state model shown in the following figure was developed from the ISA-88 state transition diagram for batch or process equipment control phases. This model has been slightly modified from the example shown in Section 2.5.

A convention that exists in the ISA-88 model is that when transient states complete, they transition to a wait state; and when a wait state receives a command, it transitions to a transient state. ISA-88 allowed an exception to this convention by having the STOPPED and ABORTED wait states, after a Reset command, transition directly to IDLE, which is another wait state. Many applications supported by Rockwell Automation required some tasks to be performed
when transitioning from the ABORTED or STOPPED states to the IDLE state. Therefore, the RESETTING state was introduced so that these tasks could be performed within a transient state and also to maintain the ISA-88 convention described above.

![Figure 3-2. ISA-88 Procedural Model with RESETTING state](image)

The following table provides an additional state definition that was added to the ISA-88 example used in the implementation example in Section 3. To see the complete list, refer to the table in Section 2.5.3.

<table>
<thead>
<tr>
<th>Procedural Control States</th>
<th>Description</th>
<th>State Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESETTING</td>
<td>The procedural element has received a RESET command and is executing its RESETTING logic to put the procedural element or equipment entity into the IDLE state. If no sequencing or logic is required, the procedural element transitions immediately to IDLE.</td>
<td>Acting</td>
</tr>
</tbody>
</table>

Along with the RESETTING state, two command transitions were added to the state transition matrix. The IDLE state is allowed to transition to the STOPPING or ABORTING states if either of those associated commands are initiated while in the IDLE state.
In a batch system, the recipe procedure typically interacts at the phase level with procedural control in the equipment. (Higher-level procedural control is located in the control recipe.) The state model is intended for interaction between external control-recipe procedural controls and internal equipment procedural controls. However, you can use the state model at whatever level the crossover from control recipe to equipment control occurs.

3.4 ISA-88 State Model PhaseManager Implementation

This section describes a feature called PhaseManager, which is part of the Logix system firmware. PhaseManager represents the ISA-88 procedural control state model, including the adaptations to the ISA-88 model discussed in Section 3.3.

- The term **state machine** (or **state engine**) is commonly used to describe an encapsulated software application that provides a container for each state’s sequencing logic and enforces the state transition rules so that you do not have to write application software to do so.

- The PackML_StateModel_AOI used in Section 4 is considered to be a state machine.

- PhaseManager is an example of an embedded state machine, meaning that the application software is actually written at the firmware level of the control platform. In RSLogix5000, PhaseManager is used to perform Equipment Phases.

- An Equipment Phase in RSLogix5000 software is similar to a program in that it contains routines and a set of isolated tags. The routines in the PhaseManager program type contain an empty routine for each enabled transient state as a container for the user-provided state logic.

PhaseManager gives the developer an easy-to-use interface between the procedural control and the equipment control. It also provides an interface to FactoryTalk Batch™ (FTBatch) to simplify the integration process and batch systems. The state machine provided by PhaseManager can be used without a batch or supervisory controller to provide a state model (that is, a procedural control model) for the manufacturing equipment. PhaseManager helps program the equipment in a structured way, which results in the same behavior in all equipment where it has been applied.
The PhaseManager feature is available on the ControlLogix® 1756 and CompactLogix™ 1768 hardware platforms, including the safety controller versions, and includes the following components:

**Phase program type:** Used to run the state model.

**Equipment phase instructions:** Used to issue procedural control commands from within the logic.

**Phase data type:** Used to link the phase to other equipment and to higher-level systems. A phase data type is a standard tag structure that is automatically created with each phase used to control and monitor the equipment. Phase data types also have “phase-scoped tags,” used to contain data used in implementing the phase but not visible outside it.

**State model:** Provides a procedural control model that follows the ISA-88 standard, including the adaptations discussed in Section 3.3.

The following figure illustrates the PhaseManager state model. This figure is actually a screen capture from RSLogix5000, when the Phase monitor feature is invoked. This feature provides a very valuable user interface for the program developer to monitor and control the Phase during program functional verification activities.

Figure 3-4. The PhaseManager State Machine

For more information about using equipment phases to set up and program in a Logix controller, see the PhaseManagerUser Manual (Rockwell Automation Publication LOGIX-UM001A-EN-P).
The example shown in the following figure illustrates how the PhaseManager feature fits into the control hierarchy of the procedural control model in a typical batch process application.

![Diagram of control hierarchy]

**Figure 3-5. Use of PhaseManager in a Procedural Control Model for a Batch Process**

### 3.5 Equipment Control

In the previous two sections, the equipment procedural elements of an application were discussed. This section discusses the physical layer—that is, the EMs, CMs, and methods used to interface with them.

#### 3.5.1 Procedural Control to Equipment Control Interface

ISA-88 states that the lowest level of procedural control must have the ability to initiate equipment control. ISA-88 users typically refer to this capability as a method or a public interface. In many cases, these methods or public interfaces are accomplished by using simple IEC-61131 programming instructions to map procedural commands directly to EMs and CMs.

Because of the broad range of applications and interfaces required for process applications, an interface example is not explicitly provided in this section. The discrete application example in Section 4 provides a partial equipment interface design.

The interface communicates commands and parameters required to execute equipment phases from the procedural elements to the equipment elements. And conversely, the interface also communicates status and process values from the equipment elements to the procedural elements. The interface may be nothing more than a passive data structure, but may also contain some sequencing and summation logic to optimize communication performance. The primary emphasis of any interface should be to provide the necessary information in an organized manner while maintaining modularity and portability of the application code.

### 3.6 Process Application Solution Example for Logix platforms

This section provides an example of applying the concepts discussed in previous sections in process applications using RSLogix5000 and FTViewME. This example is intended to show how a process unit control is implemented on a Rockwell Automation Logix platform.
3.6.1 Process Unit
The following figure illustrates a sample mix tank with two material feeds. One feed uses a flow meter (CM22) to measure the quantity of material added; the other feed uses a load cell (CM02) on the receiving vessel to measure the total added material.

![Diagram of Material Addition Mix Tank](image)

**Figure 3-6. Example: Material Addition Mix Tank**
For purposes of this example, we assume that each material is fed independently, with the additional process capabilities of mixing the materials and transferring them out of the mix tank.

By using the sample process and instrumentation diagram shown in the preceding figure, its description, and the ISA-88 models, we can define the units, equipment phases, EMs, and CMs that would exist within the controller for the mix tank example.

3.6.2 Procedural and Equipment Control Layout for a Process Unit
Figure 3-5 in section 3.4 is a good reference for what a typical ISA-88 procedural control and equipment layout would look like for this process unit example application. This diagram is helpful when determining the relationship between various areas of control in the process. This also represents how Rockwell Automation recommends that this type of application be implemented in the Logix platform.

3.6.3 Application Program Layout Based on ISA-88
The Logix program structure shown in the following figure is a typical implementation of a batch mix tank application. The controller tasks have been organized in such a way as to define the areas within the process. Each area may contain one or more process cells. Each process cell may contain one or more units. Each unit may contain one or more phases of the PhaseManager feature. In this example, the EM logic is contained within the PhaseManager feature's state routines.
3.6.4 Separating a Process Unit into EMs, CMs, and Phases

To create a modular control program, you must first understand the concepts described in Section 2.2. When creating a modular control program, there is more than one acceptable approach that will provide good results. One way is to start by identifying the CMs in your manufacturing process, then grouping the CMs into EMs that will, in turn, be supervised and coordinated by procedural controls. Another way is to start by determining what the units (which typically are vessels containing a single batch at a time), and then determine the EMs (such as ingredient addition equipment, agitating equipment, thermal jacket temperature control equipment, and transfer out equipment); this is usually done by identifying the related equipment, piping, and instrumentation on a process and instrumentation diagram (P&ID). Then you can more easily determine the CMs that are related to the equipment states that must be controlled, such as motors, valves, or other process control loops. A common convention derived from ISA-88 is that a CM can be contained by at most one EM, and an EM can be contained by at most one unit. This does not preclude CMs and EMs from existing as separate entities independent of other EMs or units.

This section discusses the typical steps involved in breaking down a P&ID into EMs and CMs, and identifying the procedures or phases that they perform.
The following figure is an example process unit P&ID in which the physical device-level equipment control elements are identified. It comprises a vessel, an agitator motor, two material addition valves, a transfer valve, and two valves for controlling the vessel jacket temperature.

First, the physical equipment control elements are identified. The control elements shown in the previous figure are shown here as CMs inside the square boxes.
The physical equipment control element identification continues with identifying key CM groupings that require basic control to coordinate their functions. In the following figure, two EMs, a Liquid Header and Jacket Control, are highlighted in the large rectangles.

Figure 3-10. Example P&ID with EMs Identified

In the following figure, the procedural control elements are identified as equipment phases. As indicated in other sections of this document, the equipment control and process control design tasks can be done in either order; in many cases, these tasks are done by different people and may be done concurrently any time after the P&ID design is complete. For some of the procedures, the phase communicates directly to a CM, such as for Mix or Solid Addition. Because the equipment required to perform the phase is a single control element, an EM is not necessary for these functions. This is acceptable and normal practice as defined in ISA-88.

Figure 3-11. Example P&ID with Phases Identified
3.6.5 Logix Application Layout

In this Logix program example, the unit procedural control is located in the unit controller. This is a common practice for situations in which equipment phases are not initiated from a supervisory controller or an external sequencer such as FTBatch. The upper levels of the unit procedural control is collapsed into a single program entitled Unit01_MasterSequencer. The lowest level of procedural control, that is equipment phases, is contained in the routines that begin with EP_.

In this example, the Unit01_MasterSequencer and the equipment phases are PhaseManager routine types. This structure could be used in many applications; however, the PhaseManager routine can be replaced with simple sequencing logic for smaller applications. The EP_AddMaterialA routine is expanded to show the PhaseManager state routines. The EMs and CMs described in the above P&ID examples are shown in the Logix program layout in programs and routines with names that start with EM_ and CM_.
4. Discrete Application Implementations

This section focuses on demonstrating how Rockwell Automation has specifically adapted the concepts discussed in Section 2 for discrete applications. The OMAC Packaging Workgroup provided an example implementation of ISA-88 for packaging machines, which is documented in the ISA88-TR88.00.02 technical report. The remainder of this section will refer to the ISA88-TR88.00.02 technical report and PackML synonymously.

This section discusses methods for implementing the ISA-88.01 and PackML concepts on the Rockwell Automation Integrated Architecture platform. Specifically, RSLogix5000 application development software, using standard IEC-61131 programming languages and a feature called “Add-On Instruction” (AOI), is used in the examples as one method of implementing modular programming concepts for discrete applications.

An AOI-based state model is used in the examples in this section to illustrate how to implement the PackML states on discrete machine applications. AOIs may also be used in the examples for program entities such as EMs and CMs.

4.1 Applying ISA-TR88.00.02-2008 (PackML)

The ISA-TR88.00.02 technical report is entitled Machine and Unit States: An example Implementation of ISA-88. There have been several revisions of PackML released prior to ISA-88’s acceptance of the technical report in 2008. For clarification, the revision that is depicted in the 2008 technical report is PackML version 3.

The following is a list of topics covered in the technical report:

- ISA-88.01 physical model applied to a machine example
- Procedural States, State Transitions, and State Commands
- Control Modes
- PackTags: A data model standard used for integration with information systems
- Use of PackTags for calculation of OEE
- Example software implementations of the preceding items

Section 4.3 of this document provides additional details, not covered in PackML, on how the collapsibility of the ISA-88 procedural model can be utilized to simplify the implementation of the model on a discrete machine. Since this document is focused on modular programming by the automation controller (PAC or PLC), it does not provide any examples related to vertical integration using the PackTags data model. For information on the Rockwell Automation implementation of PackTags, refer to Power Programming at the following website:


4.2 Physical Model (Discrete Example)

For an illustration of the complete ISA-88 physical model, refer to Section 2.4.

The figure in the following section shows the top levels of the physical model, from Enterprise down to Units.
4.2.1 Production Line and Machine
The following figure shows an example of a discrete manufacturing line, which is defined in the ISA-88 physical model as a process cell (or production line). The individual machines are equivalent to a unit; however, in some cases (such as a complex filler), the machine could be comprised of multiple units. As you can see in this figure, the ISA-88 physical model is applicable to discrete manufacturing with only subtle changes in terminology.

![Figure 4-1. Example of Discrete Manufacturing Physical Model](image)

4.2.2 Equipment Modules and Control Modules
The next step is to break down the machines into EMs and CMs. In Section 2.4, definitions and examples of EMs and CMs were provided that were typical for process applications. In this section we will highlight some examples that are typical for discrete applications.

EMs are usually unique to each machine type, and in some cases are unique to a specific machine builder. This is due to the innovation and differentiation that machine builders try to achieve. Even though two machines may perform the same function, a different arrangement of components on each machine will likely dictate correspondingly different arrangements of EMs and CMs. In general, there is much commonality across machine types, but there are also many exceptions.

The following is a list of example EMs that may be defined on a packaging machine:

- Case Packer:
  - Infeed Section EM
  - Case Erector Section EM
  - Product Staging Section EM
  - Product Inserter Section EM
  - Closer Section EM
  - Discharge Section EM

Because discrete applications such as assembly or packaging require precise and consistent mechanical movements, it is common for velocity and position control to be provided on these applications. This type of control is common with servo systems and AC motors controlled by variable frequency drives (VFDs).
With this in mind, the following list includes some example CMs (and the functions they provide) that are typical when servo systems and VFDs are utilized.

- **CM - Servo Axis Object**: Servo functions such as enable, fault reset, home, and jog
- **CM - Virtual Axis Object**: Coordination control for multi-axis systems
- **CM - PCAM Control**: Electronic cam profile calculation, execution, and recovery.
- **CM - VFD Control**: Variable frequency drive control functions
- **CM - Pneumatic Control**: Basic functions for traditional 2-state and 3-state valves
- **CM - Reversing Motor Control**: Basic motor control

Also in Section 4, a detailed example on separating a packaging machine into EMs and CMs will be discussed.

### 4.3 Procedural Control Model (Discrete Example)

For an illustration of the complete ISA-88 procedural model, refer to Section 2.5.

The following figure shows how the combined procedural and physical layer diagram would look when implementing ISA-88 on a discrete machine in which all procedural control, from the unit level down, resides within the physical equipment.

As we begin identifying some of the adaptations to the ISA-88 standard for discrete machines, it becomes evident that most of the variations from the ISA-88 examples reside in how procedural control is carried out. This is where procedural modes, states and commands are utilized to interface with equipment control.

Before reviewing the example implementation of the PackML modes, states and commands, it is important to first understand how the combined procedural model shown in the preceding figure is collapsed to meet discrete application requirements and better fit into the Logix hierarchal program organization.

ISA-88 states that the procedural control model is collapsible. Levels in the procedural control model may be left out in a specific application. ISA-88 further explains that the following considerations must be taken into account when collapsing the procedural model:
• When a procedural element level is taken out, the next higher level must take over its functions and contain the ordering logic that controls the next lower level and any other information that would have been stated in the collapsed level, including equipment requirements and other information.

• The lowest level of equipment procedural control must have the functionality to activate equipment through basic control.

The following figure is a representation of the procedural model that will be utilized in the Rockwell Automation example for a discrete machine implementation of PackML on the Logix platform. In the figure, the Operation and Phase elements have been collapsed to a single level, while the procedure level is not included in the equipment control. In most cases, the control recipe for a discrete machine is optional. A machine can usually run independent of the control recipe unless the machine needs to respond to commands from a supervisory system or upstream or downstream equipment.

![Figure 4-3. Physical and Procedural Models Collapsed - Discrete Example](image)

4.3.1 Procedural Control Modes

Control modes for discrete machines can be implemented in the Operation level of the collapsed procedural model shown in the previous figure, to carry out operations and phases.

The use of control modes to carry out operations in discrete applications may differ from how control modes are used in process applications. In process applications, modes are used at the EM or CM level for things such as a valve “Hand-Off-Auto” control, in which mode determines who or what (for example, the operator or the program) controls the valve.

A control mode contains an ordered subset of states, state commands, and state transitions that determine the strategy for carrying out a machine's process. The processing behavior of an equipment entity may be determined by evaluating the modes of the associated procedural elements and the equipment entity itself.

Typical machine control modes include Production, Manual, Sanitation or CIP, and so on. The distinguishing elements between these operation modes are the selected subset of states, state commands, and state transitions needed to support the mode. The ordered procedures within the states will typically be unique for the control mode that the state resides in.
For example, in a Production mode the EXECUTE state in a filling machine will mean it is producing filled containers in a continuous fashion. In the Manual mode, the EXECUTE state will mean that it is waiting for or performing an operator command such as jog or dry cycle. The EXECUTE state defines the functional operation of the control mode. States that have identical names may have different functions in different control modes.

The following are examples of machine control modes provided by the PackML technical report:

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

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

### 4.3.2 Procedural Control States and Commands

A machine state completely defines the current condition of a machine. A machine state is expressed as an ordered procedure (or programming routine) that can consist of one or more commands to other procedural elements or equipment entities, the status of a procedural element or equipment entity, or both. In performing the function specified by the state, the machine state will issue a set of commands to the machine procedural elements or equipment entities which will, in turn, report status back to the procedural element. The machine state will perform conditional logic which will, in turn, lead either to further execution within the current machine state or cause a transition to another state. The machine state is the result of previous activities that had taken place in the machine to change the previous state. Only one major processing activity may be active in one machine at any time. The linear sequence of major activities will drive a strictly sequentially ordered flow of control from one state to the next. No parallel states operating on the same equipment entity are allowed to be active in one machine at the same time.

There are three machine state types defined by PackML as **acting**, **wait**, and **dual**—and which are described below:

**Acting state:** A state type which represents some processing activity. This state type implies the single or repeated execution of processing steps in a logical order, for a finite amount of time or until a specific condition has been reached. In ISA-88, these state types, which primarily include those states ending in “-ING,” are referred to as “transient” states.

**Wait state:** A state type that is used to identify whether a machine has achieved a defined set of conditions. In this state type, the machine maintains a status until it transitions to an Acting or Dual state type. In the ISA-88 standard, these state types are referred to as “final” or “quiescent” states.

**Dual state:** (Note: This state is a variation from the ISA-88 procedural control standard that warrants additional clarification: when applying the ISA-88 state model to a discrete machine, the ISA-88 standard (which was originally written for batch applications) defines a unit as only being able to act on one batch or lot at a given time. This definition inherently requires that the RUNNING state (which is analogous to the EXECUTE state in the PackML state model) must always come to completion or it will never produce a batch and is therefore is “transient.”) Discrete machines can be continuous operations, which means that they can act on more than one batch at a time, and the EXECUTE state may never complete during normal production operations. Because of this application difference, when OMAC created the PackML state model, they chose to use a different name than the analogous ISA-88 state to indicate the difference in behavior for discrete applications. The unique thing to remember about the
EXECUTE or RUNNING states is that they are responsible for carrying out the primary function of the operational mode or phase. In practice, the behavior of the EXECUTE state is no different than that of the ISA-88 RUNNING state, except that the former states normal usage results in a “state complete” when a configurable number of parts have been produced, (although in most cases this results in a Stop or Hold command when a production run or shift is complete), whereas the latter states normal usage results in “state complete” after a single batch has been produced.

**Note:** This definition of dual state is unique to PackML and does not exist in ISA-88.

### 4.3.3 PackML Base State Model

There are a fixed number of defined states in the PackML Base State Model. As shown in the following table, this set of procedural control states is similar in function to the ISA-88 example states described in Section 2 of this document; however, it has been adapted to be more appropriate discrete manufacturing.

PackML Base State Model (Source ISA-TR88.00.02-2008)

<table>
<thead>
<tr>
<th>PackML States</th>
<th>Description</th>
<th>State Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOPPED</td>
<td>In this state, the machine is powered and stationary. All communications with other systems are functioning (if applicable).</td>
<td>Wait</td>
</tr>
<tr>
<td>STARTING</td>
<td>In this state, the machine is starting as a result of a START type command (local or remote). After completing this command, the machine will begin to EXECUTE.</td>
<td>Acting</td>
</tr>
<tr>
<td>IDLE</td>
<td>This state maintains the machine conditions which were achieved during the RESETTING state.</td>
<td>Wait</td>
</tr>
<tr>
<td>SUSPENDING</td>
<td>This state is a result of a command change from the EXECUTE state. This state is typically required prior to the SUSPENDED wait state, and prepares the machine by stopping active processes prior to the SUSPEND state.</td>
<td>Acting</td>
</tr>
<tr>
<td>SUSPENDED</td>
<td>In this state, the machine may be running at the relevant set-point speed, and no product is being produced. This state is the result of an upstream or downstream machine condition or other external request, and it differs from HELD in that HELD is typically a result of a local operator request.</td>
<td>Wait</td>
</tr>
<tr>
<td>UNSUSPENDING</td>
<td>This state is a result of a request from the SUSPENDED state to return to the EXECUTE state. The actions of this state may include: ramping up speeds, turning on vacuums, or re-engaging clutches. This state prepares the machine for the EXECUTE state.</td>
<td>Acting</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>In this state the machine is processing materials. The action depends on the current mode. If the machine is in the Production mode, then EXECUTE refers to the action of processing discrete parts on a continuous basis.</td>
<td>Dual</td>
</tr>
<tr>
<td>STOPPING</td>
<td>This state executes the logic which brings the machine to a controlled and safe stop.</td>
<td>Acting</td>
</tr>
<tr>
<td>ABORTING</td>
<td>In this state, the machine comes to a rapid, controlled, safe stop. Pressing the Emergency Stop button will cause the safety system to stop the machine, and it provides a signal to initiate the ABORTING state.</td>
<td>Acting</td>
</tr>
<tr>
<td>ABORTED</td>
<td>This state maintains machine status information relevant to the ABORT condition. The STOP command will force transition to the STOPPED state. The ABORTED state can be entered at any time in response to the ABORT command or on the occurrence of a machine fault.</td>
<td>Wait</td>
</tr>
</tbody>
</table>
The following table describes the PackML procedural control state commands.

**PackML Procedural Control State Commands (Source: ISA-TR88.00.02-2008)**

<table>
<thead>
<tr>
<th>PackML Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>This command orders the procedural element to begin executing the RESETTING logic. This command is only valid when the procedural element is in the STOPPED, ABORTED, or COMPLETE state.</td>
</tr>
<tr>
<td>START</td>
<td>This command orders the procedural element to begin executing the EXECUTE logic. This command is only valid when the procedural element is in the IDLE state.</td>
</tr>
<tr>
<td>STOP</td>
<td>This command orders the procedural element to execute the STOPPING logic. This command is valid when the procedural element is in any state except the ABORT/ABORTING or STOP/STOPPING states.</td>
</tr>
<tr>
<td>HOLD</td>
<td>This command orders the procedural element to execute the HOLDING logic. This command is only valid when the procedural element is in the EXECUTE state.</td>
</tr>
<tr>
<td>UNHOLD</td>
<td>This command orders the procedural element to return from the HELD state to the EXECUTE state. This command is only valid when the procedural element is in the HELD state.</td>
</tr>
<tr>
<td>SUSPEND</td>
<td>This command orders the procedural element to execute the SUSPENDING logic. This command is only valid when the procedural element is in the SUSPENDED state.</td>
</tr>
<tr>
<td>UNSUSPEND</td>
<td>This command orders the procedural element to return from the SUSPENDED state to the EXECUTE state. This command is only valid when the procedural element is in the SUSPENDED state.</td>
</tr>
</tbody>
</table>
PackML Procedural Control State Commands (Source: ISA-TR88.00.02-2008)

<table>
<thead>
<tr>
<th>PackML Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT</td>
<td>This command orders the procedural element to execute the ABORTING logic. This command is valid in every state except for ABORTING and ABORTED.</td>
</tr>
<tr>
<td>CLEAR</td>
<td>This command orders the procedural element to execute the CLEARING logic. This command is only valid when the procedural element is in the ABORTED state.</td>
</tr>
</tbody>
</table>

The following table shows an example transition matrix for local or remote state commands generated by an operator. After every ACTING state, a procedural element is required that will indicate that the ACTING state is complete, or a command is required to stop or abort it. The State Complete indication within the ACTING state procedure will cause a state transition to occur.

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

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

PackML State Transition Matrix (Source: ISA-TR88.00.02-2008)

<table>
<thead>
<tr>
<th>Current State</th>
<th>Start</th>
<th>Reset</th>
<th>Hold</th>
<th>Unhold</th>
<th>Suspend</th>
<th>Unsuspend</th>
<th>Clear</th>
<th>Stop</th>
<th>Abort</th>
<th>State Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starting</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resetting</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Held</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unholding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suspending</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suspended</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsuspending</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stopping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stopped</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aborting</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aborted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clearing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4-4. PackML State Transition Matrix (Source: ISA-TR88.00.02-2008)
The following figure shows a procedural control model from the PackML technical report that is recommended for discrete manufacturing equipment procedural control, specifically operations and phases.

Figure 4-5. PackML State Transition Diagram (Source: ISA-TR88.00.02-2008)

States within the dark gray (inner) outlined area can transition to the STOPPING or ABORTING states in addition to the transitions shown in the outlined area. States within the light gray (outer) outlined area can transition to the ABORTING state in addition to the transitions shown in the outlined area. The PackML Base State Model represents the complete set of defined states, state commands, and state transitions. Operational control modes will be defined by complete sets or subsets of the base state model.

4.4 PackML State Model AOI Implementation

In Section 3, the PhaseManager feature was used to implement the states and transition rules of the ISA-88 state model. PhaseManager currently does not support all the necessary states that are defined by the PackML technical report; therefore, Rockwell recommends that you use an AOI to implement and enforce the PackML State Model. An AOI-based state model lets you build all the necessary logic into a user-defined instruction so that only the necessary inputs and outputs are exposed. This provides a structured and organized program and facilitates reusability.

- The term state machine (or state engine) is commonly used to describe an encapsulated software application that provides a container for each state's sequencing logic and enforces the state transition rules so that you do not have to write application software that do so.

- The PackML_StateModel_AOI used in the examples of this section is considered to be a state machine.

Note: See Appendix B—References, for information on how to obtain a copy of the Rockwell Automation PackML_StateModel_AOI and associated FTVIEWME faceplates and documentation.
PhaseManager is an example of an embedded state machine, meaning that the application software is actually written at the firmware level of the control platform. In RSLogix5000, PhaseManager is a “Program” type; program contains an empty routine for each enabled transient state as a container for the user-provided state logic.

The PackML_StateModel_AOI example gives the developer an easy-to-use interface between the procedural control and the equipment control. It also can provide an interface to a supervisory controller or other machines. The state machine provided by the PackML_StateModel_AOI can be used without a supervisory controller to provide a state model (that is, a procedural control model) for the manufacturing equipment. The PackML_StateModel_AOI helps program the equipment in a structured way, which results in the same behavior in all equipment where it has been applied.

The following figure is an illustration of the PackML_StateModel_AOI. The figure shows the configuration tags (Cfg_s) and command tags (Cmd_s) on the left side of the instruction, and all of the visible status tags (Sts_s) on the right side of the instruction. See Section 5 for best practices on creating AOIs.

![Figure 4-6. The PackML State Model AOI](image)

The following figure is the PackML HMI faceplate that directly interfaces with the PackML_StateModel_AOI. In this figure, only nine of the basic states are enabled, which is typical for many discrete machine operational modes.

This figure shows three primary functions of the state model faceplate:

- The status of each enabled state.
- The eligibility of a state to execute a state or mode change request.
• The available operator commands. Some of these buttons may become invisible or grayed out if the state machine is not able to accept a specific command at any given time.

Figure 4-7. The PackML State Model HMI Faceplate

4.5 Equipment Control
The previous two sections discussed the equipment procedural elements of an application. This section discusses the physical layer—that is, the EMs, CMs, and the methods used to interface with them.

4.5.1 Procedural Control to Equipment Control Interface
ISA-88 states that the lowest level of procedural control must have the ability to initiate equipment control. ISA-88 users typically refer to this capability as a method or a public interface. In many cases, these methods are accomplished by using IEC-61131 programming instructions to map procedural commands directly to EMs and CMs.

Because of the broad range of applications and interfaces required for process applications, an interface example is not explicitly provided in this section. The discrete application example in this section provides a partial equipment interface design.

The interface communicates commands and parameters required to execute equipment phases from the procedural elements to the equipment elements. Conversely, the interface communicates status and process values from the equipment elements to the procedural elements. The interface may be nothing more than a passive data structure, but may also contain some sequencing and summation logic to optimize communication performance. The primary emphasis of any interface should be to provide the necessary information in an organized matter while maintaining modularity and portability of the application code.

4.6 Discrete Application Solution Example for Logix platforms
This section discusses an example of applying the concepts discussed in previous sections in discrete applications that use RSLogix5000 and FTViewME. This example is intended to show how a machine can be represented in the ISA-S88 / PackML hierarchy, and how this hierarchy can then be implemented on a Rockwell Automation Logix platform.
4.6.1 Discrete Machine
Implementing the ISA-88/PackML concepts in RSLogix5000 is easy. The following figure shows a Vertical Form Fill Seal (VFFS) packaging machine. VFFS machines are used to package products such as potato chips or candy in pillow-shaped bags.

![VFFS Packaging Machine](image)

4.6.2 Procedural and Equipment Control Layout for a VFFS Machine
Figure 4-3 in Section 4.3 is a good reference for what a typical ISA-88 procedural control and equipment control layout would look like for this discrete machine example application. Figures like the following are usually helpful when determining the relationships between various areas of control in the process. This also represents how Rockwell Automation recommends that this type of application be implemented in the Logix platform.

In this scenario, both the procedural control and equipment control are located in the machine/unit controller. The lines connecting the different elements of control indicate the communications (or interfaces) between the elements. The recommendations in this document adhere to this convention by only allowing communications to take place as indicated; that is, unit procedures are performed by Operations, which are implemented in the controller as Operations; Operation steps are performed on EMs; and EMs direct CMs, which control the equipment. Under no circumstances should EMs communicate with other peer EMs or CMs communicate with other peer CMs. It is acceptable, however, for EMs to communicate with subordinate EMs, and CMs to communicate with subordinate CMs. This convention preserves modularity by not allowing an element to have dependencies on other elements at the same level.
4.6.3 Application Program Layout Based on ISA-88.01 and PackML

The following process is an example that can be followed to design a software application based on the concepts and standards described or references in this document. Information from each step in this process is documented in the figure immediately following the list of steps.

1. Identify operational modes (unit procedures).
   a. Production, Dry Cycle, Manual, are examples of some typical operational modes that a machine will to perform.

2. Identify machine states.
   a. The machine states needed for each operational mode that you identify in Step 1 needs to be identified in this step.
   b. Begin with the full state model and identify states that are not required.
   c. States that are not required will be disabled in your software application.

3. Identify the operations to be performed in each state.
   a. Define all operations for each state. Repeat this process for each mode. For example, Home Machine or Home Infeed Section.
   b. Determine whether each operation is required to act independently, or operations can be grouped for issuing commands to multiple EM’s with a single command bit.
   c. If there are multiple steps to be performed in each state, identify their proper sequence.

4. Identify equipment commands from operation steps.
   a. Based on the operations identified in Step 3, identify the commands that will be required by the EMs. This will be the basis of the equipment interface.
   b. Find commonalities between the operations identified in Step 3 to build an equipment interface that serves all necessary operations down to the EM level with as few commands as possible.

   For example, determine whether a single command bit can be acted upon by multiple EMs so that individual commands do not have to be created for common EM functions. Enable, Disable, Home, and Fault Reset would be examples of commands that many EMs may have in common. In most cases, all EMs can perform these functions at the same time.

5. Identify equipment modules (EMs).
   a. Identify the EMs required for each operational mode that you identified in Step 1.
   b. Relate physical equipment sections to EMs.
   c. Identify CMs that work together to perform a specific machine function and group them together into separate EMs.
   d. Individual motors, actuators, and other output devices will usually be controlled by one for more CMs.
   e. Additional examples of this process are illustrated later in this section.

6. Identify equipment module steps.
   a. List all the steps that each EM will need to perform in order to carry out each operation step and associated command identified in Steps 3 and 4.

   For example, “Move Sealing Jaws to start position,” where “Move to start position” is the action, and “Sealing Jaws” is the physical machine device.
   c. Summarize equipment steps as a list of commands that will be performed by CMs.

7. Identify control modules (CMs).
   a. Identify the physical devices and associated CMs that will perform the equipment steps listed in Step 6.
b. Identify existing CMs in your CM library that can be re-used.

c. Create new CMs or obtain them from other libraries if required functionality is not available in your CM library. This step should be performed such that the new functionality is added to your library if you believe it can be re-used in the future.

The following figure shows the information that would be captured in Steps 1 though 7 above:

<table>
<thead>
<tr>
<th>Modes</th>
<th>States</th>
<th>EM's</th>
<th>Operation Steps</th>
<th>Seq.</th>
<th>Equipment Interface Cmd's</th>
<th>Equipment Steps</th>
<th>CMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production</td>
<td>Resetting</td>
<td>Film Control</td>
<td>Reset Faults</td>
<td>1</td>
<td>FaultReset</td>
<td>Reset VFD 1</td>
<td>CM03_VFDControl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OutFeed</td>
<td>Reset Faults</td>
<td>1</td>
<td>M101.1</td>
<td>Reset Servo Drive 1</td>
<td>CM03_ServoAxis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sealing Jaws</td>
<td>Reset Faults</td>
<td>1</td>
<td>FaultReset</td>
<td>Reset Servo Drive 2</td>
<td>CM03_ServoAxis</td>
</tr>
</tbody>
</table>

| Starting | Film Control | Enable Drive | Enable Drives | 1 | Enable Drives | Enable VFD 1 | CM03_VFDControl |
|          | OutFeed      | Move to Start   | Home          | 2 | Move to Start   | Move Serve 1 | CM03_ServoAxis   |
|          |              | Start Camning    | PrepareExecution | 3 | PrepareExecution | Calculate Cam | CM03_PCMStart     |

| Sealing Jaws | Enable Drive | Enable Drives | Move to Start | 1 | Move to Start | Move Serve 2 | CM03_ServoAxis |
|              | Start Camning | PrepareExecution | Home          | 2 | PrepareExecution | Calculate Cam | CM03_PCMStart     |

| Execute | Virtual Master | Start VM | StartExecution | 1 | StartExecution | Jog Virtual Master | CM04_VirtualAxis |
|         | Film Control   | Register Unwind |                  |    |                |                   |                  |

### Figure 4-9. Example Application Layout documentation

#### 4.6.4 Separating a Machine into Equipment Modules

The following figure illustrates the core processes of a VFFS machine.

![Core Processes of a VFFS Machine](Figure_4-10.png)

**Figure 4-10. Core Processes of a VFFS Machine**
In the following figure, the VFFS machine is shown with shaded areas representing its functional areas. These areas have been separated into logical control elements—that is, EMs.

![Figure 4-11. Breakdown of Core VFFS Machine Processes into Equipment Modules](image)

### 4.6.5 Logix Application Layout

The following figure shows where the procedural and equipment control elements reside within the Rockwell Automation Logix implementation example of a VFFS machine. At this stage of the application planning, the unit procedures and unit operational modes are listed out. In the figure, the unit operational modes are shown as PackML state diagrams, which is what is actually implemented in the controller to align with the ISA88-TR88.00.02 (PackML) standard.

![Figure 4-12. Typical Procedural and Equipment Control Layout for VFFS](image)
The preceding figure shows the EMs that were established in the previous step; in addition, a Virtual Master EM is required to coordinate movements across multiple EMs. This is done through the capability of the physical multi-axis servo system and should not be perceived as the Virtual Master EM communicating directly to the other EMs. This example maintains modularity of the control elements as discussed in previous sections.

The following figure is a screen shot of the VFFS RSLogix5000 program hierarchy view. In the figure, all pertinent elements of the ISA-88 and PackML standard are represented as a Logix task, program, routine, AOI, UDT, or standard IEC 61131 programming language.

![Figure 4-13. Sample Discrete Control - Logix Program Structure](image)

This modularity enables copying and moving CMs from one EM to another, and copying and moving EMs from one unit procedure to another, without reprogramming interlocks.
5. General Naming Conventions

This section provides suggested naming conventions that will help facilitate making your engineering library more re-usable by other developers who follow the guidelines described in this document. It will also help the resulting applications have a more consistent look and feel.

The following are the areas that will be addressed with naming conventions in this section:

- RSLogix5000 program entities (section 5.1)
  - Projects
  - Tasks
  - Programs
  - Phases
  - Routines
- Tags (Appendix A3)
- Tags RSLogix5000 enforced rules (section 5.2)
- Entities of ISA-88 physical model (section 5.3, section Appendix A2)
- I/O and Network Modules (section 5.4)
- Exported Projects - Text (.L5K), XML (.L5x) (section 5.5)
- User Defined Data Types (UDTs) and members (section 5.6)
- Add-On Instructions (AOI) (section 5.6)
- Faceplates and Global Objects with Add-On Instructions (section 5.7)
- Graphic for Displays, Faceplates, Navigation Icons (section 5.8)

5.1 General Naming Guidelines for RSLogix 5000 Program Entities

The following general naming guidelines are not enforced by software; however, they are good practice when naming RSLogix5000 entities, files, or anything else in your project:

- Names should be meaningful (and readable) to persons who will use the named entity at a later date.
- RSLogix5000 entity names use controller memory and have limited length; therefore, keep names short by using abbreviations and acronyms. Since names cannot contain spaces, names with multiple words should use mixed case to indicate the words. This keeps the name shorter than changing spaces to underscores. The following is an example:
  SlurryMixTank
- When using acronyms, stick to those in common use or those provided by industry standards.
  - GPS (Global Positioning System)
  - FIC (Flow Indicating Controller, from ANSI/ISA 5.1-1984 (R1992), a commonly used standard for identifying a field device
• When using abbreviations, be consistent, clear, and unambiguous. Use enough of the word being abbreviates so that what you are abbreviating is obvious to another reader. The following are examples:
  - EP is a suitable abbreviation for Equipment Phase
  - Fdbck is a suitable abbreviation for Feedback (instead of “FB,” which could be misinterpreted as referring to Function Block)
  - OP for Operation (procedural)
  - EP is used for Equipment Phase
  - UP is a suitable abbreviation used for Unit Procedure
  - PR is used for Procedure (in ISA-00-based systems)

5.2 Naming Rules Enforced by RSLogix 5000 Software
Formal names of entities (listed above) within RSLogix5000 projects must adhere to the following rules, which are enforced by the software:

• A name may not exceed 40 characters.
  **Note:** An operand of an instruction may be longer than 40 characters because it contains multiple items with formal names — or example, a tag (which cannot exceed 40 characters), a dot separator (“.”), and a member data element (which cannot exceed 40 characters).

• A name can contain only upper-case and lower-case letters, numbers and underscores.
  - AB_Cd2 is valid.
  - AB%Cd2 is not valid (because “%” is not a letter, number, or underscore).
  - AB CD is not valid (because names may not contain spaces).

• A name cannot begin with a number.
  - AB_C2 is valid.
  - _AB_C2 is valid.
  - _2A_BC is valid (Names may start with an underscore.)
  - 2A_BC is not valid.

• A name cannot have adjacent underscores or end with an underscore.
  - AB_CD is valid.
  - AB___CD is not valid.
  - AB_CD_ is not valid.

• Underscores are significant.
  - AB_CD and A_BCD are different names.

Case is not significant (lower-case letters match upper-case letters), and names are displayed with the case entered when first created. If the name **MySuperTag** is given to a tag when created, entering **mysupertag** for an operand will use **MySuperTag**. When the display of the operand is opened in a new window, the tag will display as **MySuperTag**, the case with which it was originally created. This is important to keep in mind, because the naming conventions described in the following sections use mixed-case names.

5.3 Naming Conventions for Physical Entities
Physical entities are items in the physical model, as defined in ANSI/ISA-88.01.
If the Site name is used in software, it should be a short abbreviation of the formal name of the site, spelled in all upper-case letters. Because an area often corresponds to a building, the area name may be entered as a building number prefixed by a site name, with no separating underscore.

If a cell is used in software, it should be entered as a two- to three-character abbreviation of the formal name of the cell. Prefixing a cell name with an area name and a single underscore is optional.

For example: MSN3_Line5 would represent Line 5 (cell) in Building 3 (area) of the Madison facility (site).

Keep in mind that using the site, area, and cell names within RSLogix5000 may limit modularity. However, using these names in FactoryTalk-enabled products can provide the desired hierarchical effect (for example: area names within FT FactoryTalk® View Site Edition (FTViewSE) and FTBatch).

5.4 Naming Conventions for I/O and Network Modules

Input/output (I/O) modules and network modules, also known as cards, are given names in the software that reflect the function of the card and the relative path from the controller to the card.

Conventions for network modules should be developed during the detailed design process.

The I/O module name should consist of the following elements, in the order listed:

1. The location of the chassis (rack) relative to the controller.
   
   Examples:
   
   R00 (Local Rack is always 00)
   
   R01 (Remote Rack or DIO Adapter #01)
   
   Rnn (Remote Racks or DIO Adapter #02 through nn as required)

2. A single underscore.

3. The letter S, followed by the slot number in the chassis.
   
   - Two-digit slot numbers are preferred to maintain sort order.
   
   - Most racks use slot number 0 for the left-most slot.
   
   Examples:
   
   S00 (Slot or Module 0 is the far left slot or closest module to the DIO adapter if not in the local rack)
   
   S01 (Slot or Module 1)
   
   SNN (Slot or Module 2 through nn as required)

4. A single underscore.

5. A function abbreviation.

The following table shows several commonly used function abbreviations.

**Note:** Rockwell recommends that you do not use the module part number, such as ENBT or EN2TR, as the abbreviation because if the module is upgraded in the future then the tag names would also have to be changed in order to preclude their becoming misleading.

<table>
<thead>
<tr>
<th>Function</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Input</td>
<td>AI</td>
</tr>
<tr>
<td>Analog Output</td>
<td>AO</td>
</tr>
</tbody>
</table>

Commonly Used Abbreviations
Commonly Used Abbreviations

<table>
<thead>
<tr>
<th>Function</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete Input</td>
<td>DI</td>
</tr>
<tr>
<td>Discrete Output</td>
<td>DO</td>
</tr>
<tr>
<td>Analog Input/Output combo</td>
<td>AIO</td>
</tr>
<tr>
<td>Discrete Input/Output combo</td>
<td>DIO</td>
</tr>
<tr>
<td>Analog/Discrete Input/Output combo</td>
<td>ADIO</td>
</tr>
<tr>
<td>Remote I/O</td>
<td>RIO</td>
</tr>
<tr>
<td>Serial data I/O</td>
<td>SIO</td>
</tr>
<tr>
<td>Motion I/O</td>
<td>MIO</td>
</tr>
<tr>
<td>DeviceNet</td>
<td>DNET</td>
</tr>
<tr>
<td>ControlNet</td>
<td>CNET</td>
</tr>
<tr>
<td>EtherNet/IP</td>
<td>ENET</td>
</tr>
<tr>
<td>HighSpeedCounter</td>
<td>HSC</td>
</tr>
<tr>
<td>ProgrammableLimitSwitch</td>
<td>PLS</td>
</tr>
<tr>
<td>Sequence Of Events</td>
<td>SOE</td>
</tr>
</tbody>
</table>

For functions that are not listed, use the same pattern to create an abbreviation.

The following table shows example I/O module names.

Example I/O Module Names

<table>
<thead>
<tr>
<th>Chassis Location, Slot Number, Module Type</th>
<th>Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local chassis, Slot 4, Analog Output Module</td>
<td>R00_S04_AO</td>
</tr>
<tr>
<td>Local chassis, Slot 12, Discrete Output Module</td>
<td>R00_S12_DO</td>
</tr>
<tr>
<td>Remote chassis #1, Slot 1, Analog Input Module</td>
<td>R01_S01_AI</td>
</tr>
<tr>
<td>Remote chassis #2, Slot 6, Discrete Output Module</td>
<td>R02_S06_DO</td>
</tr>
<tr>
<td>Local chassis, Slot 16, EtherNet/IP Bridge Module</td>
<td>R00_S16_ENET</td>
</tr>
<tr>
<td>Local chassis, Slot 15, Remote I/O Scanner Module</td>
<td>R00_S15_RIO</td>
</tr>
<tr>
<td>Remote chassis #4, Slot 2, DeviceNet Scanner Module</td>
<td>R04_S02_DNET</td>
</tr>
</tbody>
</table>

Note: In this example, 00 is always reserved for the local rack—that is, the primary rack that contains the CPU which “owns” the I/O being addressed. This is how the local rack is identified. If all of the I/O in the system is located remotely, the I/O addressing would start with R01_.

5.5 Naming Suggestions for Exported Project Components

Although it is possible to export and save .L5x files, the exporting software does not differentiate the contents of these files. This section discusses suggestions for naming .L5x files so that developers can more easily determine a file’s component and version prior to importing it.

As functionalities change, more components may become exportable, so this list is not all-inclusive. In the example, all items are listed as revision 1.0.0. Developers should adopt whatever revision tracking meets their requirements.

The following table shows examples of file naming suggestions for exported project components.

Suggested File Naming for Exported Project Components

<table>
<thead>
<tr>
<th>Exported Component</th>
<th>Suggested Filename Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add-On Instruction</td>
<td>_AOI_1.0-00. L5X</td>
</tr>
<tr>
<td>User-Defined Data Type</td>
<td>_UDT_1.0-00. L5X</td>
</tr>
<tr>
<td>Rung</td>
<td>_RUNG_1.0-00. L5X</td>
</tr>
</tbody>
</table>
5.6 Best Practices for Designing UDTs

This section focuses on the best practices to use when developing User-Defined Data Types (UDTs). As with any feature discussed in this document, a certain level of experience is expected of the reader. To learn more about UDTs and how to use them, read the Logix5000 Controllers Common Procedures Programming Manual. The manual is available for downloading from the following Rockwell Automation Online literature website:


5.6.1 User-Defined Data Types

A UDT within the Logix platform lets you create your own special device block. You can mix data types such as real or floating-point values, counters, timers, arrays, Booleans, and even other UDTs.

UDTs are re-usable, and you can copy a UDT from one project to another or from one Logix controller type to another. Modifying a UDT is easy, and doing so enables you to organize your data to mimic your machine, device, or process. A well-developed UDT is self-documenting and provides a logical representation of parts or sub-systems. Within a UDT, you can include the tag names that you assign in your code, so that all of the data associated with a device (such as a pressure transmitter or variable-frequency drive) can be grouped together.

For example, a tank might have fill control and discharge control, each consisting of a Fill Rate value, a Fill Control structure, and a discrete valve control output associated with it. In a traditional programmable controller, you would need three different data tables, which would in turn necessitate multiple tanks that you would have to map into each data table. By contrast, using a UDT enables you to define the tank by grouping together different data types.

In the example shown in Figure 5-2, the real values, control structures, and Booleans are grouped into a UDT named **UDT_TankControl**.

**Note:** Once you have used a UDT, you need to create only one tag per tank, rather than six individual tags for each tank. See the next two figures for examples of the two approaches.

---

<table>
<thead>
<tr>
<th>Exported Component</th>
<th>Suggested Filename Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routine</td>
<td>_RTN_1.0-00.L5X</td>
</tr>
<tr>
<td>Program</td>
<td>_PROG_1.0-00.L5X</td>
</tr>
<tr>
<td>Controller Project (XML)</td>
<td>_CNTL_1.0-00.L5X</td>
</tr>
<tr>
<td>Controller Project (L5K)</td>
<td>_CNTL_1.0-00.L5K</td>
</tr>
</tbody>
</table>

---

![Figure 5-1. Control Tank Tags if UDT Is Not Used](image-url)
Implementing user-defined sub-routines and associated UDTs for interface tags enables you to standardize code and makes maintaining software and managing revisions easier. In software development, each UDT and associated user-defined data sub-routine can relate directly to a physical piece of equipment such as a valve, motor, or pump. Each piece of equipment in a plant is represented in wiring diagrams, PAC logic, HMI graphic systems, and historical data systems. The common thread in all of these components is the UDT or data structure that stores attributes of the physical device, thereby making them easily accessible to other parts of the control system.

### 5.6.2 Naming Conventions for User-Defined Data Types

Because UDTs are mostly self-documenting, determining a consistent naming convention for them is fairly simple. The following convention and associated details apply only to UDT names:

- **Data Type Name:** `UDT_UserSpecificName`

  The `UserSpecificName`, which helps describe the UDT’s purpose, is always preceded by `UDT_`. There are no spaces between words.

  All words in the UDT name start with a capital letter to indicate a break point (for an example of this, see the following figure).

In re-usable component classes that have the same UDT, do not create a new UDT for each instance. For example, all valve CM instances will be of the same type: `UDT_ValveDiscrete2State`. Use the exact instance-style naming for UDTs only when the data type is unique to that instance.

*UDT_WaterSupply* could be a general UDT for several different instances of water addition. Even if the code is unique, there is no need to create a unique UDT if the code uses the generic data type.

### 5.6.3 UDT Tag Structure Order

The order in which elements are listed in the UDT can have a significant impact on its size if several BOOL, INT, or SINT elements are defined. Bear in mind that memory is allocated in 4-byte (that is, 32-bit) increments, and every DINT, REAL, STRING, or sub-UDT element will always start at the beginning of a 4-byte boundary. For example, if the first element defined is a BOOL, it uses the first 4 bytes allocated to the UDT. Other BOOLs can be assigned immediately following without consuming any more memory, until the first 4 bytes are consumed. However, if the next element is a DINT, the DINT element allocates another 4 bytes even though the BOOL occupies only a single bit in the first 4 bytes. Thus, in this example, the 31 bits of memory between the BOOL and the start of the DINT are allocated but are not accessible.

Here are some good rules to remember:

- UDT memory is allocated in 4-byte increments.
• Elements that occupy 4 bytes or more will always start at a 4-byte boundary. These will include DINT, REAL, STRING, any UDT, or any other complex data structure.

• Elements of smaller data types (for example, BOOL, SINT, or INT) will start on the next byte boundary that matches its size, so that all the data types in the UDT are fully contained in their respective 4-byte increments. For example, INT elements will start on 2-byte boundaries, SINT elements will follow at the next byte, and BOOL elements in succession will occupy consecutive bits within a byte.

When creating a UDT, keep the following in mind:

Consider how the UDT will be used. UDTs are usually created to simplify the implementation of a control function. Consider not only what data is needed but how that data will be implemented in the application. Consider assigning data into nested UDTs when substructures must be copied or when the data naturally fall into a multi-tiered structure. You will need to create any contained substructure UDTs before you can create the containing overall UDT.

Consider which program and how much memory the UDT will require. Often, ease of use conflicts with performance; both should be taken into account when determining the order in which to place UDT structure members. One method is to order the members by usage, which renders the data type of each member irrelevant.

In the following figure, the example UDT on the left, UDT_Tank, has members arranged by function without regard for memory usage. This makes sense in the context of implementation, because members toward the top are ordinarily used in the software code.

However, the disjointed listing of data types in UDT_Tank consumes 25% more memory than the example UDT on the right, UDT_TankPacked. In UDT_TankPacked, the BOOL members are grouped according to their functionality, with the input BOOLs grouped at the top and the output BOOLs grouped at the bottom. As a result, the data type size is reduced from 80 bytes to 64 bytes. (Remember that the placement of small data types affects the amount of memory used by the UDT, so 16 bytes was gained by grouping the BOOLs together.) Reordering larger elements will have no impact on memory usage. If your UDT has only a few small data types or is already fairly well-grouped, the memory gained by reordering those data types may be outweighed by a loss of usability of the UDT. However, reordering larger UDTs that have many BOOLs interspersed among larger elements can have significant impact on the UDT’s size.
In an application in which you will use only a few UDTs, the UDTs have only a few smaller elements types, or in which memory limitations are not an issue, arrange the UDT members for ease of use. In an application in which you will create many tags from UDTs or in which memory limitations will be an issue, group smaller data types together to minimize the UDT’s use of memory.

5.7 Best Practices for Designing AOIs

This section focuses on the best practices when developing AOIs. As with any of the features discussed in this document, a certain level of experience is expected of the reader. To learn more about AOIs and how to use them, read the Logix5000 Controllers Common Procedures Programming Manual. The manual is available for downloading from the following Rockwell Automation Online literature website:


5.7.1 Add-On Instructions

An AOI is used to encapsulate commonly used functions or device controls. It is not intended for use as a high-level hierarchical design tool. Programs with routines are better suited than an AOI to contain code for your application’s area or unit levels.

The benefits of using an AOI are as follows:
Reusable code: Using an AOI promotes consistency among projects by re-using commonly used control algorithms. For example, if you have an algorithm that will be used multiple times in the same project or in multiple projects, you can incorporate it into an AOI to make the AOI algorithm more modular and easier to re-use.

An easy-to-understand interface: Placing a complicated algorithm inside an AOI creates an easy-to-understand interface (by making only essential parameters visible or required) and reduces the time necessary for documentation development (by automatically generating instruction help). The AOI also keeps internal data, such as intermediate calculation results that have no meaning to the AOI user, in hidden Local Tags, keeping irrelevant information from complicating the formal interface.

Protection of intellectual property: Placing proprietary code inside an AOI and then using source protection prevents others from viewing or changing your code.

Simplified code maintenance: Because AOI logic animates for a single instance, code maintenance is simplified.

Overall re-usability: AOIs can be used across multiple projects. You can define the instructions for an AOI, get them from someone else, or copy them from another project.

Once the AOI is defined in a project, the AOI behaves similarly to the built-in instructions that are already available in the RSLogix5000 software. Like the built-in RSLogix5000 software instructions, the AOI is easily accessible on the instruction toolbar and in the instruction browser.

5.7.2 Naming Conventions for Add-On Instructions
Like UDTs, AOIs are partially self-documenting—which, as with UDTs, makes creating consistent naming conventions for them easy. The following naming convention and associated detail apply only to AOI names. For an example of an AOI name, see the following figure.

AOI Name: ApplicationName_VariantName(_AOI)

The Application Name helps describe the AOI’s general application purpose.

The _VariantName helps describe its detailed application purpose.

Do not insert spaces between the words in an AOI name.

Capitalize the first letter in all words in an AOI, to indicate a break point.

Ending with _AOI is optional. This may be a good practice if you need to quickly identify an instruction as an AOI and not a native RSLogix5000 instruction; however, there is also the need to keep the names as short as possible to avoid name-length issues.
When exporting an AOI, it should be named as described in Section 5.5.

Figure 5-4. Example AOI

### 5.7.3 AOI Parameter Name Prefixes

The number of parameters in an AOI can quickly multiply. This likelihood, combined with the capability of an AOI to control the visibility of parameters, may result in cryptic descriptions of how an AOI’s parameters are used.

The convention for the prefixes is to abbreviate the parameter’s base function to three letters and an underscore, followed by additional text to better clarify the specific parameter’s function. By using these prefixes, you will increase the developer’s understanding of how to properly use the AOI. For more information, see the following table.

<table>
<thead>
<tr>
<th>Parameter Base Function</th>
<th>Prefix</th>
<th>Description of Use</th>
</tr>
</thead>
</table>
| Command                 | Cmd_    | Generally used to as a command input, either from the operator via the HMI or from the program. Examples: 
  - Cmd_Reset: Clear faults and reset the process.
  - Cmd_JogServo: Jog a servo axis.
  - Cmd_FillTank: Fill a tank with a liquid. |
| Configuration           | Cfg_    | Generally used to designate a value used in configuring how the process within the AOI functions. This is only occasionally changed. It can be entered from the HMI or as part of a recipe. Examples: 
  - Cfg_JogDirection: Selects in which direction a servo will jog: 0=Positive, 1=Negative.
  - Cfg_BulkFill: Selects which fill rate to use: 0=Slow Rate, 1=Fast Rate.
  - Cfg_UserUnits: Selects what measure of volume to use: 0=mm³, 1=m³, 2=gal.
  - * Cfg_EnableInterlocks: Enable interlock functionality.
  - * Cfg_EnablePermissive: Enable permissive functionality |
### Parameter Base Functions

<table>
<thead>
<tr>
<th>Parameter Base Function</th>
<th>Prefix</th>
<th>Description of Use</th>
</tr>
</thead>
</table>
| **Status**              | Sts_   | Status of the process within the AOI instruction.  
Examples:  
- * Sts_Alm: An alarm condition (such as a HI/LOW alarm) exists within the process. See the section Alarm for how to deal with alarm enumerations.  
- * Sts_ER: An error with an instruction execution within the process has been detected. See the Error row for information about dealing with Error enumerations.  
- Sts_IndexComplete: The servo index move within the process has completed.  
- Sts_FillInProcess: The tank filling process is underway. |
| **Error**               | Err_   | If the Sts_ER bit is on, the Err_ parameter will indicate which actual error is occurring within the process. This can be either a bit level or value level indication. Value-based error annunciation allows for a large quantity of errors to be supported within a single indicator. However, this approach requires that errors are annunciated one at a time. Bit level error recording can support multiple errors simultaneously, but can require a large number of indicators to support all error states.  
Examples:  
- Err_Value: A non-zero value indicates an error condition. Different values indicate different errors.  
- Err_PCamCalcFault: Indicates that an error has occurred in an M CCP. |
| **Alarm**               | Alm_   | If the Sts_Alm bit is on, the Alm_ parameter will indicate which alarm is occurring within the process. This can be either a bit-level or value-level indication. Value-based alarm annunciation allows for a large quantity of alarms to be supported within a single indicator. However, this approach requires alarms to be annunciated one at a time. Bit-level alarming can support multiple alarms simultaneously, but can require a large number of indicators to support all alarm states.  
Examples:  
- Alm_Value: A non-zero value indicates an alarm condition. Different values indicate different alarms.  
- Alm_TankHI: Indicates that a HI level condition has been detected within a tank. |
| **Input**               | Inp_   | Real-time data used to drive the process within an AOI. Generally used to designate a connection either to a real input point, a control device, or to data received from other calculation processes.  
Examples:  
- Inp_ServoPosition: Variable providing the input value for a servo’s position.  
- Inp_ServoRegistrationPosition: Input of a servo’s registration position.  
- * Inp_InterlockOK: Input indicating external interlocks are met.  
- Inp_TankLevel: Variable providing the analog input for a tank’ level.  
- Inp_TankLevelFillRate: |
| **Output**              | Out_   | Real-time data driven from the process within an AOI. Generally used to designate a connection to a real output point, a control device, or to data sent to other calculation processes.  
Examples:  
- Out_GlueGun1: Output signal to turn of Glue Gun 1  
- Out_ServoCorrectionDistance: Output of a servo registration correction distance.  
- Out_OverflowValve: Output signal to open the Overflow Valve.  
- Out_TankLevelError: Output of a difference between target and actual fill level of a tank. |
Parameter Base Functions

<table>
<thead>
<tr>
<th>Parameter Base Function</th>
<th>Prefix</th>
<th>Description of Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>Ref_</td>
<td>Reference data are complex data structures made up of a combination of input and output data. These structures are used to pass data into an AOI, where some process is performed. The results are then loaded back into the structure to be passed out of AOI for use in a subsequent step. Examples: - Ref_PositionCamRecovery: Provides the data set for calculating a Position Cam with all offsets factored in, as well as the resulting Position Cam Profile to run in an MAPC instruction. - Ref_TankFillControl</td>
</tr>
<tr>
<td>Parameter</td>
<td>Par_</td>
<td>Parameters are variables that are received from an external source, such as a batch or a recipe management system that can be internal or external to the program. Examples: - Par_MachineSpeed: Provides a machine’s running speed. - Par_TankFillLevel: Provides a tank’s target fill level.</td>
</tr>
<tr>
<td>Set point</td>
<td>Set_</td>
<td>Set points are variables received from an operator or HMI and are not part of external source such as a batch or recipe management system. Examples: - Set_MachineMaxSpeed: Provides the setting for a machine’s maximum allowed speed. - Set_TankHILevel: Provides the setting for a tank’s HI alarm limit.</td>
</tr>
<tr>
<td>Value</td>
<td>Val_</td>
<td>Designates a value (calculated inside the instruction) that might not be the primary output of the instruction.</td>
</tr>
<tr>
<td>Report</td>
<td>Rpt_</td>
<td>Designates a value (calculated inside the instruction) that is typically used for batch reporting.</td>
</tr>
<tr>
<td>Information</td>
<td>Inf_</td>
<td>Non-functional data such as an AOI revision level, AOI name, and so forth, for displaying a faceplate.</td>
</tr>
<tr>
<td>Ready</td>
<td>Rdy_</td>
<td>Command-ready bits. Typically, they are Booleans calculated inside the control routines to reflect whether the routine will allow state change commands. Used with the HMI faceplates to enable or disable command buttons.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>Prefix</th>
<th>Description of User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Command</td>
<td>PCmd_</td>
<td>Used as a command input for commands typically issued by the application program. Examples: - PCmd_ProgReq - Request for Program Mode made by the application (as opposed to Cmd_ProgProgReq) - PCmd_AutoReq - Request for Auto Mode made by the application (as opposed to Cmd_ProgAutoReq)</td>
</tr>
<tr>
<td>Operator Command</td>
<td>OCmd</td>
<td>Used as a command input for commands typically issued by the operator via the HMI. Examples: - OCmd_ProgReq - Request for Program Mode made by the operator (as opposed to Cmd_OperProgReq) - OCmd_AutoReq - Request for Auto Mode made by the operator (as opposed to Cmd_OperAutoReq)</td>
</tr>
</tbody>
</table>

Note: Parameters in this table that have an asterisk (*) beside them have a special role in faceplate animation; for further explanation of this role, refer to Section 5.10. Note that if one of the animated navigation buttons described is used without the corresponding AOI parameters present, the navigation button will not function properly.
5.7.4 Examples of AOI Parameters

The following figures show the parameter list from an example AOI, using recommended naming prefixes. Unlike in UDTs, there is no memory penalty in AOIs for interspersing smaller element data types among larger ones; therefore, BOOLs, DINTs, and REALs are placed where they make the most functional sense rather than giving consideration to data type grouping.

In the following figure, three parameter types are shown as indicated in the second column: **Input**, **Output**, and **InOut**.

**Input parameters** generate data input points to allow data values to be transferred from the application program into the AOI, much like a subroutine’s input parameters. Data is actually moved from the AOI tag’s Input parameter to the reference inside the AOI for execution. Only atomic data types (that is, BOOL, INT, SINT, DINT, REAL) can be Input parameters; data structures, including UDTs, cannot be Input parameters.

**Output parameters** are values returned from AOI execution, and they hold values to be passed back into the application program, much like subroutine return parameters. Like the Input parameters, data is actually moved from the AOI reference back to the AOI tag parameter after execution. Only atomic data types can be Output parameters; data structures, including UDTs, cannot be Output parameters.

- **InOut parameters**, such as Ref_Title at the bottom of the list in the following figure, function differently from Input and Output parameters in that no data is actually moved. If an InOut parameter is altered during the AOI’s execution, the value in the referenced tag is updated directly without the need to transfer data through an Input or Output parameter. Because of this, InOut parameters are considered “passed-by-reference” parameters. Complex data types, including UDTs, can be assigned as InOut parameters.
The **Default** column indicates the values placed in each new instance of the AOI when it is created. If the **Req** column is checked, that parameter requires a link to an external tag of the same data type. Use the **Vis** column to select which parameters are visible in ladder diagrams, and which are visible in function-block diagrams. The **Descriptions** column holds text that will be visible when an AOI’s configuration dialog is opened in RLogix5000.

![Figure 5-5. Example AOI Parameters for a Regulator](image-url)
Figure 5-6. Example AOI for a Discrete Device

The following figure shows an example of an AOI integrated into a program.

![Figure 5-6](image)

**Figure 5-7. Simple DeviceControl AOI as a Function Block**
5.7.5 AOI Design Concepts

To be sure that certain data is passed into or out of the AOI, use a required parameter. A required parameter must be passed as an argument in order for a call to the instruction for verification.

To pass a required parameter in ladder diagrams and in structured text, specify an argument tag for the parameter.

In a function block diagram, required Input parameters and Output parameters must be wired.

In a ladder diagram, InOut parameters must have an argument tag.

If a required parameter lacks an associated argument, the routine that contains the call to the AOI will not verify.

Make an Output parameter visible if any of the following conditions apply:

- You don’t need to pass the parameter’s value out to an argument, but you do want to prominently display that value for troubleshooting purposes. Required parameters are always visible, and InOut parameters are always required and visible. Every input and output parameter, regardless of whether it is marked as Required or Visible, can be programmatically accessed as a member of the instruction’s tag.

- You have a parameter for which the user must specify a tag as either its source for input or its destination as output, and you don’t want this to be optional. In this case, set the parameter as required. Any required parameter is automatically set to Visible.

- The Visible setting is always set for InOut parameters. All InOut parameters are required.

- An Output parameter of type BOOL that is not required but is visible will show as a status flag on the right side of the block in the ladder. This can be used for status flags such as DN or ER.

5.8 Using AOIs with HMI Faceplates and Global Objects

The goal of conventions for user interfaces is to provide a consistent look and feel. This section describes the methodologies and standards used for developing ControlLogix member naming conventions and popup faceplates for use within FTViewSE and FactoryTalk® View (Machine Edition) (FTViewME).

In addition to the information provided in this document, two standard FactoryTalk® View (FTView) display files are available. Each contains navigation icons.

5.8.1 AOIs and FTView Global Objects

AOIs are frequently used in conjunction with Global Objects (GOs), which are their equivalent feature in the FTView domain. GOs are screen objects that can be created and then re-used. When you create a complex AOI, it uses a GO to act as an interface for configuration, control, and diagnostic purposes. When a complex AOI uses a GO in this manner, the AOI is referred to as a faceplate. Make sure that the AOI name that you select also references the AOI’s faceplate interface.

5.8.2 Global Naming Conventions for Faceplates

The naming convention for the faceplate is as follows:

Faceplate Name: Faceplate: AOI Name

Tag Name: Tag: AOI Instance Name

This label can either be hard-coded on the HMI or created as a String Display field and changed dynamically, as shown in the following figure.

The name for the AOI to which the faceplate connects should be in the upper-left corner. The AOI Name is the name given to the AOI in the RSLogix5000 project.
5.9 Images For Faceplate Navigation Buttons

An icon can be used on a display or button as an indicator. The following table lists and describes the FTViewSE functions and graphics.

*Note:* xxx is the base name of the screenset.

<table>
<thead>
<tr>
<th>FTViewSE Function</th>
<th>FTViewSE Graphic</th>
<th>FTViewSE Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status and Alarms</td>
<td></td>
<td>Click Action: Launches display xxx_Status and passes a tag.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Animation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>—If Sts_Alar, the icon blinks red.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>—If Sts_ER, the icon is a solid yellow.</td>
</tr>
<tr>
<td>Close Display</td>
<td></td>
<td>Click Action: Closes this display.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Animation: None.</td>
</tr>
<tr>
<td>Configuration Display</td>
<td></td>
<td>Click Action: Launches display xxx_Config and passes a tag.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Animation: None.</td>
</tr>
</tbody>
</table>
### Faceplate Navigation Buttons

<table>
<thead>
<tr>
<th><strong>FTViewSE Function</strong></th>
<th><strong>FTViewSE Graphic</strong></th>
<th><strong>FTViewSE Description</strong></th>
</tr>
</thead>
</table>
| Trend Display         | ![Trend Display Icon](image) | Click Action: Launches display xxx_Trend and passes a tag.  
Animation: None. |
| Interlocks            | ![Interlocks Icon](image) | Click Action: Launches display xxx_tmpInterlock and passes two tags.  
Animation:  
If not inp_InterlockOK, the icon is yellow.  
Visible only if Cfg_EnableInterlocks is true. |
| Permissives           | ![Permissives Icon](image) | Click Action: Launches display xxx_Perm and passes two tags.  
Animation: Visible only if Cfg_EnablePermissives is true. |
| Help Display          | ![Help Display Icon](image) | Click Action: Launches display xxx_Help and passes no tags.  
Animation: None. |
| Manual Tuning Display | ![Manual Tuning Display Icon](image) | Click Action: Launches display xxx_Tune and passes a tag.  
Animation: None. |
| Auto Tuning Display   | ![Auto Tuning Display Icon](image) | Click Action: Launches display xxx_Autotune and passes a tag.  
Animation: None |

The following table lists and describes the FTViewME faceplate navigation buttons.

*Note: The icons pictured in the FTViewME Graphic column in this table were the current icons at the time of publication of this document. The latest versions of ME and SE navigation icons can be found at: http://www.rockwellautomation.com/solutions/oem/modular.html.*

### FTViewME Faceplate Navigation Buttons

<table>
<thead>
<tr>
<th><strong>FTViewME Function</strong></th>
<th><strong>FTViewME Icon</strong></th>
<th><strong>FTViewME Graphic</strong></th>
<th><strong>FTViewME Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Status and Alarms</td>
<td><img src="image" alt="Status and Alarms Icon" /></td>
<td><img src="image" alt="Status and Alarms Graphic" /></td>
<td>Click Action: Opens the Status and Alarms screen</td>
</tr>
<tr>
<td>Close Display</td>
<td><img src="image" alt="Close Display Icon" /></td>
<td><img src="image" alt="Close Display Graphic" /></td>
<td>Click Action: Closes an open display.</td>
</tr>
<tr>
<td>Configuration Display</td>
<td><img src="image" alt="Configuration Display Icon" /></td>
<td><img src="image" alt="Configuration Display Graphic" /></td>
<td>Click Action: Opens the Configuration display.</td>
</tr>
</tbody>
</table>
## FTViewME Faceplate Navigation Buttons

<table>
<thead>
<tr>
<th>FTViewME Function</th>
<th>FTViewME Icon</th>
<th>FTViewME Graphic</th>
<th>FTViewME Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trend Display</td>
<td><img src="image1" alt="Icon" /></td>
<td><img src="image2" alt="Graphic" /></td>
<td>Click Action: Opens the Trend display.</td>
</tr>
<tr>
<td>Interlocks</td>
<td><img src="image3" alt="Icon" /></td>
<td><img src="image4" alt="Graphic" /></td>
<td>Click Action: Opens the Interlocks display.</td>
</tr>
<tr>
<td>Permissives</td>
<td><img src="image5" alt="Icon" /></td>
<td><img src="image6" alt="Graphic" /></td>
<td>Click Action: Opens the Permissives display.</td>
</tr>
<tr>
<td>Help Display</td>
<td><img src="image7" alt="Icon" /></td>
<td><img src="image8" alt="Graphic" /></td>
<td>Click Action: Opens the Help display.</td>
</tr>
<tr>
<td>Manual Tuning Display</td>
<td><img src="image9" alt="Icon" /></td>
<td><img src="image10" alt="Graphic" /></td>
<td>Click Action: Opens the Manual Tuning display.</td>
</tr>
<tr>
<td>Auto Tuning Display</td>
<td><img src="image11" alt="Icon" /></td>
<td><img src="image12" alt="Graphic" /></td>
<td>Click Action: Opens the Auto Tuning display.</td>
</tr>
<tr>
<td>Home</td>
<td>N/A</td>
<td><img src="image13" alt="Graphic" /></td>
<td>Click Action: Opens the Home display.</td>
</tr>
<tr>
<td>Recipe</td>
<td>N/A</td>
<td><img src="image14" alt="Graphic" /></td>
<td>Click Action: Opens the Recipe or Parameter display.</td>
</tr>
<tr>
<td>Performance</td>
<td>N/A</td>
<td><img src="image15" alt="Graphic" /></td>
<td>Click Action: Opens the Performance display.</td>
</tr>
<tr>
<td>Warnings</td>
<td>N/A</td>
<td><img src="image16" alt="Graphic" /></td>
<td>Click Action: Opens the Warnings display.</td>
</tr>
<tr>
<td>Realtime Data</td>
<td>N/A</td>
<td><img src="image17" alt="Graphic" /></td>
<td>Click Action: Opens the Real Time Data display.</td>
</tr>
</tbody>
</table>
### FTViewME Faceplate Navigation Buttons

<table>
<thead>
<tr>
<th>FTViewME Function</th>
<th>FTViewME Icon</th>
<th>FTViewME Graphic</th>
<th>FTViewME Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Settings</td>
<td>N/A</td>
<td><img src="image" alt="Settings Icon" /></td>
<td>Click Action: Opens the <strong>Settings</strong> display.</td>
</tr>
</tbody>
</table>
Appendix A—Secondary Content

This document includes primary content and secondary content, which are defined as follows:

- Primary content directly facilitates the purpose and intent of this document. Practices discussed in these sections should be adhered to whenever possible to maximize the quality, consistency, and re-usability of your developments. All primary content is contained in the main sections of this document.

- Secondary content describes a known good practice that is related to, or is an extension of, the primary content. Secondary content typically stems from the absence of a single best practice that the review committee can agree to follow. This situation typically occurs when several different acceptable methods are already prevalent. If the content is not critical to the primary intent of the document, then the most prevalent method is chosen as the default, and other methods are eliminated or provided as secondary content. Secondary content will be provided at the end of the document in this appendix (A). A reference should be provided from the primary content location to the appendix where the secondary content is located and the reverse.

This appendix contains sections that discuss the following topics:

- Conventions for revision numbering
- Naming conventions within the RSLogix5000 application
- Naming conventions for tags
- Double aliasing/mapping of I/O
- Suggestions for HMI display versioning
Appendix A1 Conventions for Revision Numbering

(Secondary Content)

Revision numbers are assigned to all software and documentation components, such as the following:
- ControlLogix Add-On Instructions, Routines and Equipment Phases
- FTView graphic objects (global objects) and screens (faceplates)
- FTBizware Batch Recipe Procedures and Area Models

Whenever a change is made to a component, a new revision number must be assigned to that component. Where components in multiple subsystems work together as one object, a change may require a new revision number for all the related components.

The AOI feature in release 16 and later of RSLogix5000 provides a revision number format that consists of the following elements:

- A major revision number, with no leading zeroes, expressed as an uppercase letter: M
- A minor revision number, with no leading zeroes, expressed as a lowercase letter: n
- Extended text (xxxxxxxxxx), which is used for additional revision information.

Example Revision Number Format: M.n. xxxxxxxxxxxxx

The following conventions are suggested for revision numbers for all components and are made compatible with the preceding AOI format by using the extended text:

- A “Tweak” 2-digit (minimum) revision number, expressed as two uppercase letter “Ts”: TT
- A plant-specific 2-digit (minimum) revision number, expressed as two uppercase letter “Ps”: PP

Example “Extended” Revision Number Format: M.n-TT.PP

Example Revision Number Format: 1.3-02.05

In this example, the major revision is 1; the minor revision, 3; the tweak revision,.02; and the plant-specific revision,.05.

The tweak and plant-specific revision numbers are placed in the revision extended text for Add-On Instructions.

<table>
<thead>
<tr>
<th>Revision</th>
<th>When to Change</th>
<th>Format and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major revision</td>
<td>Change when a library or other major set of components is released as a set.</td>
<td>No leading zeroes</td>
</tr>
<tr>
<td>Minor revision</td>
<td>Change when the (public) interface or outwardly visible behavior of a component changes (in other words, any change that can affect how other components work with the revised component).</td>
<td>Use 0 (zero) for unreleased components. Use 1 on first release draft (Revision 0.1-00). Use 0 when the major revision changes and increment from there. (Revision 1.0-00, Revision 1.1-00, Revision 2.0-00, and so on)</td>
</tr>
</tbody>
</table>
When a set of components works together as one object, their major and minor revision numbers should be kept in lockstep. Doing this enables a user to know that a given FTView faceplate works with a corresponding ControlLogix AOI. When a change is made to the Logix code that changes the AOI parameters (that is, the public interface), the Faceplate is also revised. Tweaks (such as bug fixes) can be made to each component individually. Therefore, faceplate revision 1.2-03 will work compatibly with AOI revision 1.2-01, because the M.n. numbers (that is, the "major minor" numbers) are the same. However, faceplate revision 1.2-03 is not guaranteed to work with AOI revision 1.3-03, because the M.n. numbers (in this case, the minor numbers) are not the same.

<table>
<thead>
<tr>
<th>Revision</th>
<th>When to Change</th>
<th>Format and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tweak revision</td>
<td>Change when any change at all is made to a component provided in a Rockwell Automation library. Such a change is a bug fix or a change of the internal logic and Local Tags of an AOI.</td>
<td>No leading zeroes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use 1 for the first draft (Revision 0.1-00).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use 0 when the major revision changes and increment from there. (Revision 1.0-00, Revision 1.1-00, 1.10.00, Revision 2.0-00, ..., 10.0-00, and so on)</td>
</tr>
<tr>
<td>Plant-specific revision</td>
<td>Add and maintain a Plant-specific revision when a Rockwell Automation library item is modified for a particular project, customer or plant. Increment on any revision to the component.</td>
<td>Two digits with leading zero if required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use 00 for the first release change from the Rockwell Automation library component, and increment from there. Reset to 00 when the minor revision is incremented.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Two digits with leading zero if required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use 01 for the first change from the Rockwell library component, and increment from there.</td>
</tr>
</tbody>
</table>

When a set of components works together as one object, their major and minor revision numbers should be kept in lockstep. Doing this enables a user to know that a given FTView faceplate works with a corresponding ControlLogix AOI. When a change is made to the Logix code that changes the AOI parameters (that is, the public interface), the Faceplate is also revised. Tweaks (such as bug fixes) can be made to each component individually. Therefore, faceplate revision 1.2-03 will work compatibly with AOI revision 1.2-01, because the M.n. numbers (that is, the “major minor” numbers) are the same. However, faceplate revision 1.2-03 is not guaranteed to work with AOI revision 1.3-03, because the M.n. numbers (in this case, the minor numbers) are not the same.
Appendix A2 Naming Conventions within the RSLogix5000 Application

(Secondary Content)

Use the information in this section when naming the software entities in a Logix application.

A2.1 Naming Conventions for ISA-88 Related Entities (Secondary Content)

The following table lists the standard prefixes for ISA-88 related entities in RSLogix5000 software programs. Common prefixes are needed for items that have no other way to be differentiated within the software. Unique identifiers (such as sequential numbers or customer asset numbers) can be used for instances of class objects where differentiation is required.

<table>
<thead>
<tr>
<th>SA-88 Entity</th>
<th>Prefix</th>
<th>Process Class Example</th>
<th>Process Instance Example</th>
<th>Discrete Class Example</th>
<th>Discrete Instance Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Procedure</td>
<td>UPxx_</td>
<td>UP_MIXTank</td>
<td>UP01_MixTank_MT01</td>
<td>UN_VFFS</td>
<td>UN01_VFFS</td>
</tr>
<tr>
<td>Operation Procedure</td>
<td>OPxx_</td>
<td>OP_Calib</td>
<td>OP01_SpeedCalib</td>
<td>OP_Mode</td>
<td>OP01_Production</td>
</tr>
<tr>
<td>Equipment Module</td>
<td>EMxx_</td>
<td>EM_Agitator</td>
<td>EM01_Agitator</td>
<td>EM_SealJaw</td>
<td>EM01_SealJaw</td>
</tr>
<tr>
<td>Control Module</td>
<td>CMxx_</td>
<td>CM_Pump</td>
<td>CM01_Pump</td>
<td>CM_Enable</td>
<td>CM01_Enable</td>
</tr>
</tbody>
</table>
Appendix A3 Naming Conventions for Tags

(Secondary Content)

When naming tags, remember the following guidelines:

The naming convention for tags should be kept consistent throughout an application.

ControlLogix tag names are sorted alphabetically; therefore, these tags should be named according to the first characters in the entity name. For examples of common entity names, refer to the list in the following table.

### Naming Conventions for Tags

<table>
<thead>
<tr>
<th>Physical or Procedural Logic Entity Name</th>
<th>Prefixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Input Tags</td>
<td>I_</td>
</tr>
<tr>
<td>I/O Output Tags</td>
<td>O_</td>
</tr>
<tr>
<td>Remote I/O Tags</td>
<td>RIO_</td>
</tr>
<tr>
<td>Control Module Class Tags</td>
<td>CMDevice ID_</td>
</tr>
<tr>
<td>Equipment Module Class Tags</td>
<td>EM_</td>
</tr>
<tr>
<td>Equipment Phase Class Tags</td>
<td>EP_</td>
</tr>
<tr>
<td>Inflight Class Tags</td>
<td>Inflight_</td>
</tr>
<tr>
<td>Queue Class Tags</td>
<td>Queue_</td>
</tr>
</tbody>
</table>
Appendix A4 Aliasing, Double Aliasing, and Mapping of I/O Modules

The previous section describes the naming convention for I/O modules. The physical naming conventions for I/O points within a module are provided by RSLogix5000 in controller tags.

The following figure shows an example for local I/O in controller tags.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local2D</td>
<td>(...) AB:1756D_014D0</td>
<td></td>
</tr>
<tr>
<td>Local2I</td>
<td>(...) AB:1756D_014I0</td>
<td></td>
</tr>
<tr>
<td>Local2I Fault</td>
<td>2#0000_0000_0000_0000_0000</td>
<td>DINT</td>
</tr>
<tr>
<td>Local2I Data</td>
<td>2#0000_0000_0000_0000_0000</td>
<td>DINT</td>
</tr>
<tr>
<td>Local2I Data</td>
<td>0 BOOL</td>
<td></td>
</tr>
<tr>
<td>Local2I Data</td>
<td>0 BOOL</td>
<td></td>
</tr>
<tr>
<td>Local2I Data</td>
<td>0 BOOL</td>
<td></td>
</tr>
<tr>
<td>Local2I Data</td>
<td>0 BOOL</td>
<td></td>
</tr>
</tbody>
</table>

Figure Appendix-1. Figure Local I/O

Access and change management is convenient because all access points will be updated when changing the location of the module.

However, the following two general requirements for I/O naming are not covered by module and Logix 5000 I/O point names:

1. Use of a functional name that describes the function of the specific I/O point, such as Pusher_Valve. This name should be re-used in different instances of the same EM or CM class, so it is not unique within a controller.

2. The link of an I/O point to electrical schematics with their nameplates and I/O numbering, such as 50Y11. This name is unique in a controller in the same way as the module name is with the I/O point.

While there is no commonly used standard in the naming of I/O modules, and there is no real specification for I/O points, Rockwell Automation recommends that you use either double aliasing or I/O mapping are used to support these requirements.
Appendix A4.1 Aliasing

As a good practice for modularity, the functional name should be an alias in program scope. For example, when copying the program to create another instance of an EM or CM class, the aliases can be used to change the I/O point without touching the code. By changing the alias reference, the I/O point will be changed.

The following figure shows an example of aliasing:

Figure Appendix-2. Figure Aliasing Example 1
**Appendix A4.2 Double Aliasing**

In addition to the naming of I/O modules, the electrical schematics nomenclature for I/O modules is useful to track hardware signals or perform I/O tests. Aliases can be used to provide these names in a Logix project. As described, these names are unique within a controller and should reside in controller scope.

The following figure shows an example of how to create an alias in the controller scope to match the naming conventions for electrical schematics:

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias For</th>
<th>Base Tag</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>_1CB5</td>
<td>Local:21 Data 0</td>
<td>Local:21 Data 0</td>
<td>BOOL</td>
</tr>
<tr>
<td>_B0Y11</td>
<td>Local:30 Data 0</td>
<td>Local:30 Data 0</td>
<td>BOOL</td>
</tr>
<tr>
<td>_B0Y12</td>
<td>Local:30 Data 1</td>
<td>Local:30 Data 1</td>
<td>BOOL</td>
</tr>
<tr>
<td>± Local:2C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>± Local:2C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>± Local:3C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>± Local:3C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>± Local:3C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following figure shows how to use double aliasing to combine electrical schematics and function names.

**Note:** The electrical schematics can be used to alias the functional name.

The following figure illustrates how the program would activate two pusher valves in sequence when the button is pressed.

**Figure Appendix-4. Double Aliasing Example**

As a functional description, select names like Push Button, Pusher Valve_1, and Pusher Valve_2. These names can be used generically throughout the CM.

The advantages of aliasing and double aliasing are as follows:

Testing of I/O modules can be done according to hardware plans without a running program.

The only requirement is the list of aliased controller tags that can be generated as a CSV file (Comma Separated Variable) from the electrical CAD (Computer Aided Drafting) system.

No code is required, which saves memory and scan time.
Appendix A4.3 Mapping of I/O Modules

The other method that uses several names for the same I/O point is the mapping of I/O modules. The following figure shows an example of this mapping.

![Diagram of I/O Module Mapping Programming Example]

**Figure Appendix-6. I/O Module Mapping Programming Example**

**Advantage of mapping I/O modules:**

- I/O module naming can be mapped several times without risk of losing it when uploading the program.
- In the event of an I/O channel failure, the mapping can be online-edited to change the input or output to a working spare channel. The alias tag cannot be re-assigned online.

**Disadvantage:**

- Certain data types, such as axes, cannot be mapped and need to be aliased instead.
- Requires scan time and additional program code that aliases do not require.
Appendix A5 Suggestions for HM1 Display Versioning

(Secondary Content)

Each developer must supply a version number for the screen. Versioning is not currently ideal; the following list presents some suggested workarounds.

**Visual Basic** (FTViewSE only). You can use Microsoft Visual Basic (VBA) to add comments about the version number and the record of revisions to a display. However, if you paste the display with your comments onto a new display, your comments will be transported onto the new display.

**Tool tips.** These can be used in FTViewSE to identify the version.

**Text.** Text works well for an easy version identifier, although it occupies some screen space. Place the text in the bottom-right corner of the faceplate.
Appendix B—References

Note: At the time of publication of this document, the reference documents listed here were valid. Referenced documents are subject to revision or discontinuation of availability without notice. Users are encouraged to look for and review the latest version(s) of this document and the referenced documents before determining the documents and standards that are best suited for your purposes.

Rockwell Automation has developed application libraries for process and discrete applications that have developed according to this document. These libraries are available for download by Rockwell Automation customers from the Rockwell Automation Knowledgebase.

- The process library is a comprehensive set of simple and advanced control modules. This library is called Plant PAX Process Objects. The CMs in this library are AOIs with FTViewSE faceplates.
  
  Go to: http://www.rockwellautomation.com/knowledgebase

  Search for Plant PAX Process Objects or for Answer ID #62682.

- The discrete library is a completely documented modular application template. This library includes commonly used controls modules, including complex motion used in discrete market segments. This library is called Power Programming. The CMs in this library are AOIs with FTViewME faceplates.

  Go to: http://www.rockwellautomation.com/knowledgebase

  Search for Power Programming. (Version 4.1 is the appropriate version at the time of this publication.)

The following documents contain provisions that are referenced in this document.

All of the documents are subject to revision; 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.

- ISA-TR88.00.02-2008 Machine and Unit States: An Implementation Example of ISA-88
- ISA-88.00.01-1995 Batch Control Part 1: Models and Terminologies
- ANSI/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 Symbols and Identification
- ANSI/ISA-5.1-1984 (R1992) Instrumentation Symbols and Identification

Modular Equipment Control

- IEC 61131-1 Standard for programmable logic controllers (PLCs), General Information
- IEC 61131-3 Standard for programmable logic controllers (PLCs), Programming Languages
- IEC 61131-4 Standard for programmable logic controllers (PLCs), User Guidelines
- PLCopen TC5 Safety Certification
- Weihenstephan Standard - Part 2 Version 2005
  
  http://www.wzw.tum.de/lvt/englisch/Weihenstephaner_Standards_GB.html

- ANSI/ISA-95.00.01-2000 Enterprise-Control System Integration Part 1: Models and Terminologies
- ANSI/ISA-95.00.02-2001 Enterprise-Control System Integration Part 2: Object Model Attributes
- ANSI/ISA-95.00.05, Enterprise-Control System Integration Part 5: Business-to-Manufacturing Transactions
- DIN 8782, Beverage Packaging Technology: Terminology Associated with Filling Plants and their Constituent Machines