PLC Programming Standards

Las Virgenes Standards
6518021-STD-00

Document Revision History

Revision	Date	Description	Author
00	10/31/19	Issued for release.	M. Gassaway

Table of Contents

1 Purpose

The purpose of this standards (STD) document is to offer a description of how each controller within the plant should be programmed and communicate with each other and associated HMI/SCADA applications. The primary function of this STD is to provide guidance for the PLC logic, overall architecture of the software and the algorithms used to implement the requirements imposed on. It is not intended to provide holistic details such as those that are mechanical in nature or specific equipment/process operation typically found in the standard operating procedures (SOP). This document is not a maintenance or operations manual; however it does supply the important design information necessary for a thorough understanding of how each PLC program is organized and the common features.

1.1 Scope

This document is intended to be a general overview of programming guidelines and implementation. It should be used in conjunction with other standards documents for HMI/SCADA, Tag Naming Conventions, etc.

1.2 Acronyms & Abbreviations

Term	Definition
HMI	Human Machine Interface
SCADA	Supervisory Control and Data Acquisition
LVMWD	Las Virgenes Municipal Water District
PC	Personal Computer
PLC	Programmable Logic Controller
SCS	System Controller Supervisor
SSC	Subsystem Controller
VFD	Variable Frequency Drive

1.3 Notation and Nomenclature

1.3.1 State Machines/Diagrams

Functional control of the various processes within the facility shall be controlled utilizing a state machine. This state machine shall be able to intelligently determine the state the equipment/process is in and systematically move through the various states of operation for said piece of equipment/process based upon a variety of

preset and predetermined conditions including operator input, normal sequence of equipment operation, abnormal conditions, interlocks and error handling. The state machine utilized by the Contractor shall be flexible and scalable to be able to be utilized for control scenarios that have minimal states and/ or inputs as well as control scenarios that have several states and/ or multiple inputs.

This state machine shall reside in the PLC and corresponding graphics/user interface shall be developed for the corresponding HMI and/or SCADA system which will allow Operations to monitor and control the status of various equipment/processes including the current state it is in and actionable data (Fault Status, Alarm State, Next State, etc.) that will better help Operations make informed decisions on how to operate the system, including giving them the ability to take over the system manually by taking the state machine out of Auto/ Remote mode.

State diagrams are often used to illustrate logic. They are generally based on UML conventions for such diagrams.

State "entry" and "exit" actions, and state transitions, are all implemented as one-shot actions in the PLC. State "do" actions are continuously executed while the logic is in that state. The action itself can be a typical PLC instruction, like Latch; if the action is Set, or no command is given, it is assumed that the PLC instruction is an OTE().

All timers in the state diagrams should be treated as local timers (one for each instance of the state diagram) unless specified otherwise.

Documentation including all equipment/ process relationship map, control strategy, and exception handling strategy for each state machine and the associated equipment/ process it controls shall be provided as part of this project. Documentation shall also include an automated tool that allows for ease of configuration and installation as well as for ongoing maintenance of the system/ state machines.

1.4 General Considerations and Assumptions

1.4.1 Inputs and Outputs

It is assumed for all state machines that inputs do not change within the update/evaluation period of the logic (the design presumes that the logic is executed instantaneously). Therefore, in implementation, the software may have to internally buffer inputs on a per-scan or per-state machine level in order to achieve this data consistency requirement. This may be an issue in a Standard task because inputs can change as soon as a new packet arrives from the input card, even during the program scan and thus needs to be accounted for.

1.4.2 Timers

Timers used in this document have the following properties:

- All times (unless otherwise noted) are specified in seconds.
- Elapsed Time or Accumulated Time: The total time which the timer has been "run".
- Expiration/Expired: When the elapsed time of the timer equals or exceeds a preset value, the timer is said to have expired. It is assumed that the timer does not continue to run (elapsed time does not continue to increase) after it has reached its expiration.

Timers have the following interfaces to control their operation:

- Reset: A timer's accumulated time is set to zero. The timer continues to time up from zero if it is also being asked to Run when the reset occurs.
- Run: The timer accumulates elapsed time when the timer is instructed to run.
- Pause: The timer stops accumulating time when it is instructed to pause, but the elapsed time is retained. A subsequent Run command would add time to the retained value.

Note that in a typical PLC implementation (using a Timer On instruction), removing the Run request also resets the timer to zero, so it is not necessary to explicitly reset the timer in most cases. Also, a Pause is typically implemented by disabling a retentive timer (which allows the timer accumulated value to remain intact).

2 Control System Architecture

The control system implements a distributed architecture with dedicated control systems for different parts of the plant, as applicable. The distributed control systems are as follows:

- SCS-XXX: The System Controller Supervisor (SCS) is the overall coordinator and manager for the plant. The SCS sets the operating mode and interfaces with the other control systems to coordinate the overall plant as required. The SCS also is responsible for the Falling Tower, Water Cannon, Bomb Run, Dock Hits, and Tipping Drums effects.
- SSC-XXX: This Subsystem Controller (SSC) is responsible for the individual equipment/process in which is it is controlling.

Elements of the distributed control system communicate with each other via Ethernet I/P, using direct messaging between controllers to exchange control and status information.

The following diagram represents the control systems that work together to make up the overall plant control system and shows how each SSC could potentially communicate with each other.

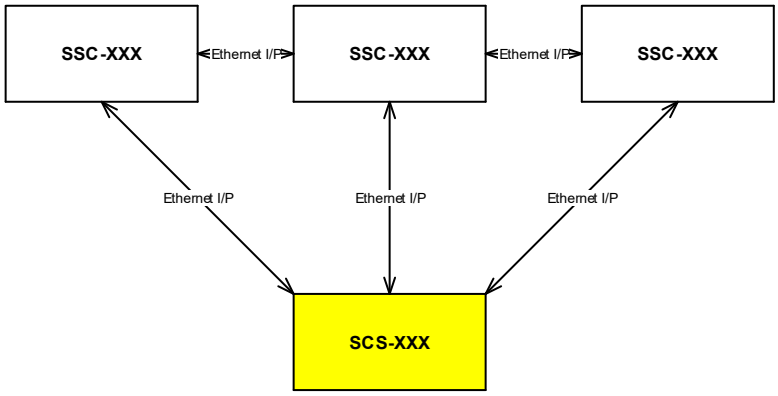

Figure 1-1, Control System Architecture (placeholder)

2.1 SSC Control System Architecture

The SSC is comprised of one ControlLogix PLC (controller) and associated I/O.

3 Software Development Environment

3.1 Rockwell

3.1.1 Studio5000

Studio5000 is the Allen-Bradley programming tool used to configure, program, monitor, and troubleshoot the controller.

3.2 Schneider

3.2.1 EcoStruxure Control Expert

EcoStruxure (Formerly Unity Pro) is the Schneider programming tool used to configure, program, monitor, and troubleshoot the controller.

4 Code Organization

4.1 Documentation

All Routines, Rungs and Tags will have appropriate descriptions and definitions. Routines need to have brief descriptions of the tasks that are implemented. Rungs need to have descriptions of the process that is being accomplished. All Tags should have a description describing the event/process/reading that it is associated with.

4.2 Tasks, Programs, and Routines

The SCS/SSC has one task, the Main Task and is configured to be continuous with a 500ms watchdog.

The execution periods and watchdog values are shown in the following table:

Task	Continuous	Watchdog (ms)
T00_MainTask	X	500

Within the Main Task are the Programs which are used to organize the software into modules that accomplish one major function or similar functions. The Programs are listed in the Controller Organizer in the order of execution. Each Program name begins with Pxx_, where xx is a two-digit number that begins at 00 and increments with each additional program.

Below each Program are Routines which contain the executable code. The Routine is used to isolate the software into manageable cells containing one or more lines of code executing a defined process. The Routines can be modularized into subroutines that can be called multiple times.

Each Routine name begins with Rxx_, where xx is a two-digit number that begins at 00 and increments with each additional routine (numbering restarts at 00 within each new Program). The first Routine is always considered to be the “Main Routine” and is assigned as such for its Program; it can either contain all of the logic for that Program, or it can use a series of subroutine calls to jump to other Routines with the Program (if the logic is sufficiently complex that multiple Routines are needed). Its name is always R00_MainRoutine.

This naming convention allows Routines to be listed in order of execution, assuming the programmer numbers them accordingly (the Rockwell development application only lists Routines alphabetically, regardless of how they are called; by following this naming scheme, the on-screen list matches the order of execution). The following table lists all Tasks, Programs, and Routines commonly used within the SCS/SSC.

Controller Task	Programs (in order of execution)	Routines (in order of execution)	Contents
MainTask (Continuous)	P00_Inputs	R00_MainRoutine	General Routine for calling required subroutines in Program
		R01_System	Mapping and manipulation of all system inputs
		R02_Digitals	Mapping and manipulation of all digital inputs
		R03_Analogs	Mapping and manipulation of all analog inputs
		R04_Derived	Mapping and manipulation of all calculated/derived inputs
		R05_HMI	Mapping and manipulation of all HMI inputs
		R06_XXX	Continuation of any additional input programs (added as required)
	P01_Outputs	R00_MainRoutine	General Routine for calling required subroutines in Program
		R01_System	Mapping and manipulation of all system outputs
		R02_Digitals	Mapping and manipulation of all digital outputs
		R03_Analogs	Mapping and manipulation of all analog outputs
		R04_Derived	Mapping and manipulation of all calculated/derived outputs
		R05_HMI	Mapping and manipulation of all HMI outputs
		R06_XXX	Continuation of any additional output programs (added as required)
	P02_Alarming	R00_MainRoutine	Fault Reset pulse
		R01_System_AlmSum	System Alarm Summary
		R02_XXX_AlmSum	Process Alarm Summary (added as required)
	P10_Control	R00_MainRoutine	General Routine for calling required subroutines in Program
		R01_XXX	Continuation of additional control programs named appropriately (added as required)
	P20_Communication	R00_MainRoutine	Process fault messages/status
		R01_Messaging	Configuration of all messaging instructions to other SSC controllers
		R02_XXX	Continuation of additional comm. programs named appropriately (added as required)

5 Alarms/Faults/Messages

Various parts of the program can generate alarms, messages and faults. Alarms are alerts resulting from physical I/O. Messages typically are just text strings that are displayed on an HMI for the benefit of operators, maintenance, etc. so that they can determine the current state of the system and also to help with troubleshooting. Faults are a message (text string) plus some response by the control system (shut down a piece of equipment, or trigger an Emergency Stop, for example).

5.1 Alarms

All alarms shall have ability to be shelved, silenced or placed into maintenance mode to avoid nuisance alarms, both locally through an HMI or the SCADA system. Each instrument with an associated alarm shall have its own dedicated Input or Output for alarm notification as applicable and shall not be combined into only one general fault.

5.1 Faults and Messages

For both faults and messages, the local (SSC) HMI as well as the SCS HMI can print these messages. They are triggered to do so by receiving the messages tag, which is a UDT of Booleans that represent the individual messages (for example, messages.valve101FailedToClose).

The messagesUDT is defined as follows:

UDT: messagesUDT		
Member	Data Type	Description
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	
	Boolean	

There is one tag of this type, called messages. This tag is read by the SCS/SSC HMI.

6 User-Defined Data Types

There are various User-Defined Data types that are used throughout the application. They are used to combine data types (often Boolean and Double Integer, but it can be any data type, including other UDTs) into more complex ones.

Typically, these complex data structures are then used to represent collective information about physical or logical equipment (for example, MSG_CommData which contains the Data used in messaging between controllers) or to represent the state of a particular piece of logic (for example, the systemModeUDT which contains Booleans to represent the different modes that the SCS can be in).

All UDTs need to be defined and approved by an LVMWD representative.

7 Predefined Data Types

Utilization of the native controller library needs to be used as defined below.

7.1 Rockwell

The basis for this guide is the use the latest edition of the Rockwell Automation Library of Process Objects PROCES-RM002G-EN-P. The most current Process Library version should be used, which as of Aug. 2019 is Version 4.10.00

We recommend that you use this manual along with these additional references:

- PROCES-RM001 – Reference Manual
- PROCES-RM013 – Describes the logic per Library object
- PROCES-RM014 – Describes the display elements per Library object
- PROCES-UM003 – Application User Manual

7.2 Schneider

The general library should be used and all User-Defined Data Types should be created to match the Rockwell library.