Practical Series

PRACTICAL SERIES AUTOMATION LIBRARY

FUNCTIONAL SPECIFICATION

AUTHOR: MICHAEL GLEDHILL

Published By: Practical Series of Publications
Published in the United Kingdom
mg@practicalseries.com

Document No.: PS2001-5-2101-001

Document Template: PS2001-5-nnnnn-nnn R02.00 SHORT GxP Blank (Ind-Calisto)

DOCUMENT AUTHORISATION

	NAME	POSITION	SIGNATURE	DATE
Author	Michael Gledhill	Lead Engineer		01 May 2022

The signature of the author confirms that the document has been prepared in accordance with an approved document management process, that all content is technically complete and that all relevant material has been included.

Reviewed by	Frank Greenwood	Project Manager		01 May 2022

The signature of the reviewer indicates that the document has been checked for technical content and that it complies with the technical standards, specifications and conventions.

Approved by	Christopher Wish	Quality Manager		01 May 2022

The signature of the Approver indicates that the document has been checked for compliance with the quality management Procedures.

THIRD PARTY AUTHORISATION

	NAME	POSITION	SIGNATURE	DATE
Approved by	Alfred Featherstone	Operations Director		01 May 2022

The signature of the Approver indicates that the document satisfies the Project quality and validation requirements of the Third Party's quality system.

REVISION

REVISION	DATE	REVISED BY	DESCRIPTION
R02.00	01 May 2022	Michael Gledhill	Properties standardised across all documents Changes to interrupt functional group names
R01.00	02 Jul 2020	Michael Gledhill	First released

CONTENTS

BLANK PAGE

1 Introduction

This document is the *Functional Specification* (FS) for the *Practical Series Automation Library* of software modules (the PAL).

This *Functional Specification* has been produced by Michael Gledhill, under his own authority as the Lead Engineer of the Practical Series Automation Library of software modules project (hereafter referred to as the Project).

The Functional Specification defines how the system is to function from an operational point of view and the *design* of the system that makes this possible; as such it describes:

① How the system operates

② The functions that are carried out automatically by the system

③ The facilities available to the users of the system

④ The equipment used to control the system

⑤ The interfaces between the various parts of the system

1.1 Scope and purpose of this document

The scope of this document includes the complete control system associated with the PAL, broadly this includes:

① The control system hardware, including the following:

- Simatic S7-1500 Controllers
- Electrical panels
- Instrumentation
- Hardware documentation

② The control system software, including the following:

- PAL software modules
- Software documentation

The purpose of this FS is to ensure that:

③ All the requirements of the Project are properly documented

④ All requirements are clear, precise and unambiguous

⑤ All requirements are specific, measurable, realistic and testable

The FS and its subsidiary documents: the Hardware Design Specification (HDS) *[Ref. 006]*, Software Design Specification (SDS) *[Ref. 006]*, Software Module Design Specifications (SMDSs) *[Ref. 008]* and the Style Guide (SG) *[Ref. 010]* will collectively provide a design that satisfies all the requirements of the Project specified in the User Requirements Specification (URS) *[Ref. 003]*.

1.2 Ownership, status & relationship to other documents

This document, the Functional Specification (FS) is a fundamental document for the Project, the ownership of the document (those whom control it and are able to modify it), its status within the Project and its relationship to all other primary documents are important factors and are explained below:

1.2.1 Ownership of the document

This Functional Specification has been produced, and is controlled and maintained by the Practical Series of Publications (PSP).

This Functional Specification and all the referenced documents produced by the PSP are subject to the change control management procedures for this Project, these are detailed in the Project Quality Plan (QP), *[Ref. 001]*.

1.2.2 The status of this document

The Functional Specification (*this document*) is a contractual document and is a deliverable item under the terms of the Project. The Functional Specification is an approved document and this approval must take place prior to the commencement of any other Project design activity.

The document must be approved by the Practical Series of Publications Operations Manager.

1.2.3 Relationship to other documents

The Functional Specification is the primary design document for the Project, it will form the basis of all the Project design work. The full document flow-path for the Project including the Functional Specification is shown in Figure 1.1; full details of this document within this flow-path can be found in the Project Quality Plan (QP), *[Ref. 001]* and Validation Plan (VP), *[Ref. 002].*

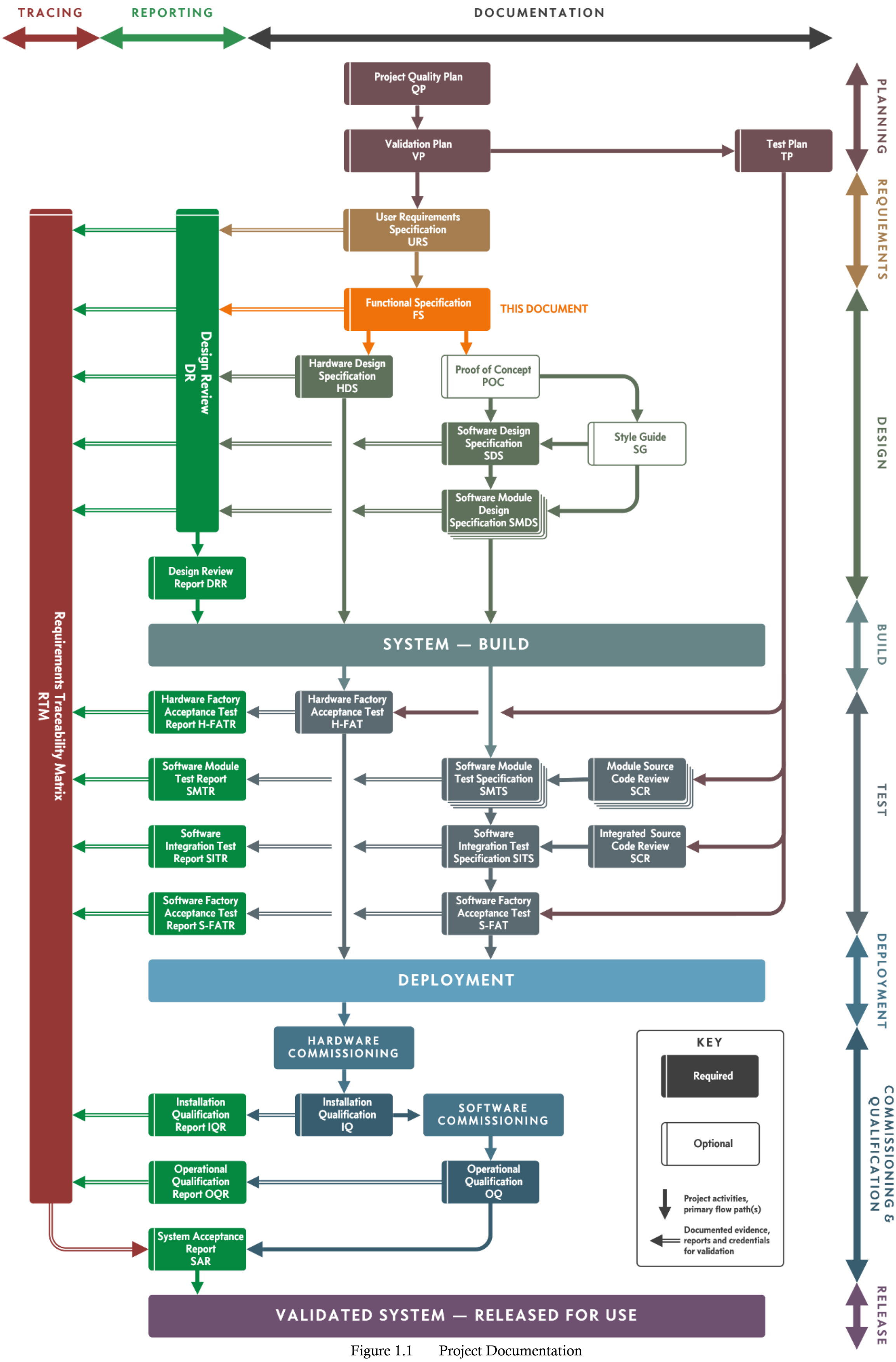


Figure 1.1 Project Documentation

2 Overview

This overview sets out a brief description of the Project and its design. It also explains the approach that is to be taken in defining the specification for the design, this is in terms of the strategy being deployed and the breakdown of the requirements into detailed functional specifications.

2.1 A description of the Project

The Practical Series Automation Library (PAL) Project is a library of software modules and templates that are to be made available for the Siemens Simatic S7-1500 range of Controllers (and to a lesser extent the S7-1200 range).

The PAL is configured and deployed using the Siemens Simatic TIA Portal programming environment.

The PAL software structure is designed such that it is applicable to virtually all industrial applications that can generally controlled by a programmable logic controller (PLC)[1].

Such applications can generally be thought of as processes that operate with a response time of more than 100 ms. I.e. the system would not be expected to respond to some stimuli faster than 100 ms. In practice, a Controller may (and usually will) respond faster than this; however, a response time of 100 ms is considered to be an acceptable limit for PLC control.

---

1 The Siemens Simatic S7-1500 and S7-1200 range of Controllers are, what would be generally understood to be, program logic controllers (PLCs); Controller is simply the common term used within Siemens literature for this type of device. For clarity, where a Siemens Controller is being referred to, the word Controller is capitalised (to indicate it is a Siemens Controller, rather than some non-specific controlling device).

The PAL software being developed as part of this Project, is considered to be suitable for use in the following types of industries (this is not an exhaustive list):

- Water and waste water treatment
- Pharmaceutical and batch production
- Brewing and fermentation
- Chemical manufacturing
- Oil and gas systems
- Power plants
- Food and beverage production

At its most basic level, the PAL will be a library of software modules that control the fundamental aspects of an industrial plant; such modules would for example read the value of an instrument, operate a valve or drive, perform a calculation &c.

Such software modules are referred to as *standard* modules, these are fixed modules that perform a particular function and are identical across all software installations.

The PAL has many such modules; making up the bulk of the PAL.

The PAL also contains *application* specific modules; these contain software that is applicable to the plant being controlled.

For example, if a project were to control five valves, there would be an *application* module that called the *standard* valve device driver five times and each instance would link the *standard* module to the particular signals and internal storage locations associated with the valve in question.

The *standard* modules within the PAL will be fixed modules, the software within these modules will be written, tested and validated as part of this Project and at only that point will the modules be released for use. Once released, the modules must not be modified or changed in any unauthorised way, to do so would invalidate the software.

The further modification of any of these *standard* modules (or indeed the addition of further *standard* modules) will only take place under the Project change control put in place by this the Quality Plan *[Ref. 001]* or under the control of subsequent future projects.

Application modules are specific to each individual plant within which the PAL is deployed; they will be written for a particular project and are configured to match the requirements of that project.

Although individual in nature, the *type* of *application* modules required by a particular project will be part of a universal set of such modules, this set of modules determines the fundamental structure of the software, for the PAL, these are broadly:

- System (internal) signal generation
- Instrumentation
- Safety and interlock systems
- Calculations
- Continuous control
- Sequence control
- Command execution logic
- Device handling (valves, drives &c.)
- Alarm handling
- Communications

Each *application* module will also have to conform to the standards, formats and specifications laid out in the various requirements and design documentation associated with the PAL project.

As such, a comprehensive set of *template* application modules will be designed, developed, tested and issued as part of the Practical Series Automation Library Project.

These modules will serve as example modules to demonstrate how the PAL modules should be used, and the best practices for doing so.

There will also be a series of documentation modules that demonstrate how the modules should be documented, commented and named.

Certain modules within the PAL library, will have operator interfaces; typically, these are modules for reading instruments, managing equipment (drives, valves, loops) and controlling various aspects of the plant control (sequences for example). These interfaces require that the mechanisms for displaying the status of instruments and devices and for controlling those instruments and devices, be established as part of this design.

Note: *Although the interfaces for display and control are defined as part of this Project, the supervisory systems themselves (SCADA, HMI &c.) will not be developed as part of this Project, the interfaces (and to some extent the expected appearance of the graphical symbols that would be used in such systems) will be developed in their entirety.*

2.2 The approach

The requirements for this Project specified in the User Requirements Specification (URS) *[Ref. 003]*, are to build a library of Siemens Simatic Controller software modules that will be applicable to virtually all industrial applications that can generally be controlled by such a Controller.

The design necessary to achieve these requirements can be broken down into the following components:

① Determine the overall structure of the software that is to be used as the basis for all industrial application deployments, this will form the basis of the required *application* modules

② Determine the *standard* modules that are to form the library

③ The design of the end-user interface for certain specific modules that require such an interface

④ Establish a series of *template* and *document* modules that can provide example usage of all the *standard* and *application* module in context

⑤ Design the hardware test environment that allows modules and applications to be developed and tested

A brief overview (a summary or abstract) of each of these five areas is given below, this is intended to provide an introduction to the detailed functional specifications that follow in subsequent sections.

2.2.1 The structure of the software

Software within a Controller is generally structured in a logical order, and that order is determined by the order that Controller is to process the information available to it and then act on that information.

For example, if it were the function of a Controller to close a valve if a tank reached a target level and open it if below that level, the logical order of events would be:

① Read the tank level instrument

② Evaluate the level (is it above the target level)

③ Act on that information to either open or close the valve

There is no hard and fast rule for how a Controller programme should be structured; it can be done many in different ways. That said, there are certain common approaches and some measure of good engineering practice that are generally applied to the structure of a programme and these will be adopted within the PAL.

The PAL will broadly adopt the following overall software structure:

① System Functions

Generates common (global) system signals and timing pulses.

Reads Controller cycle and real time clock information.

Reads and identifies any module and system faults.

② Read Instruments

Reads all analogue and digital instruments.

Analogue instruments are scaled and converted to real engineering units; high and low alarms and warnings are generated.

Digital instruments signals are filtered and stored

③ Interlocks and protection

Interlocks are overriding conditions that prevent something from happening (or ensure something does happen) when a particular condition (or set of conditions) is present.

④ Safety systems

Safety systems are used for both machine and personnel protection (emergency stop systems &c.).

⑤ Calculations

Perform any discreate calculations required by the process, this may be mathematical calculations, timing calculations or even logical calculations

⑥ Continuous Control Logic

Continuous control is the constant monitoring and evaluation of plant devices and process variables. The continuous control logic assess the condition of the plant and generates actions to produce the required process conditions.

⑦ Sequential Control Logic

Sequential logic operates in a series of successive steps, each step carrying out an action and waiting for transition conditions to be satisfied before moving to another step.

Sequential logic is often triggered by the continuous logic

⑧ Command execution

Both continuous and sequential control logic generate actions, these actions require something to happen (a valve to open, a drive to start &c.). The command execution blocks martial these signals and trigger the appropriate response (issues the command).

⑨ Device Drivers (control loops)

Control loop device drivers monitor and control the various different types of control loops employed by an application

Each loop has an individual driver, that determines if fault condition exist, applies any interlock conditions, &c.

⑩ Device Drivers (control loops, valves, drives &c.)

Valve and drive device drivers monitor and control individual valves and drives connected to the controller.

Each device has an individual driver, that driver determines if the devices is in a healthy or fault condition, applies any interlock conditions that are associated with the device and operates the device in response to any command generated within at the command execution stage.

⑪ Messages

Handles Controller messages: alarms, warnings, events and prompts that require some form of user interaction

⑫ Communications

Executes any system-to-system communications (Controller to Controller) and any other form of communication required by the system (point-to-point serial communications, ProfiBus field messaging &c.).

The above list is the complete PAL programme structure, not all Controller programmes will have all these steps (it depends on the application in question). However, where a step is used, it must follow the execution order shown in the above list.

For example, if a programme did not require *interlocks* or *safety*, but had *instruments* and *continuous logic*, then the *continuous logic* would follow the *read instruments* (*interlocks* and *safety* would not be present); *continuous logic* must not precede *read instruments* in the order of execution.

Each of the points in the above list will have generally an *application* block and, usually, at least one *standard* module associated with it; (there are some points, *command execution* being one, that do not have any associated *standard* modules).

2.2.2 The standard modules

The full list of standard modules is given in Section 8. These cover the following aspects of the control system:

- System (internal) signal generation
- Instrumentation
- Safety and interlock systems
- Calculations
- Sequence operation
- Device handling (control loops, valves, drives &c.)
- Messages
- Communications
- General purpose subroutines
- Debug functions

The standard modules will form part of the Controller software structure (§ 2.2.1).

In the context of this Project *standard* modules are software modules that will carry out a particular function; an example would be a module that controls the operation of a valve, such a module would typically do the following:

- Open and close the valve when commanded to do so
- Determine if the valve is in a fault condition (i.e. the valve did not open when required to do so)
- Provide status information about the valve allowing other systems (SCADA, HMI &c.) to display the condition of the valve (i.e. opened, opening, closed, closing, fault, interlocked &c.)

The module would be configurable to accommodate different types of valves and signalling arrangements:

- Different arrangements of position feedback (none, open only, closed only or both open and closed)
- Different opening and closing times
- Handle external fault signals (typical for motorised valves)
- Accommodate different energising states (i.e. energise to open or energise to close)
- Manage different interlock arrangements and signals

The module would also determine how the operator could interface with the valve:

- Provide manual control (operator can take direct control of the valve)
- Restrict specific manual control function (this ranges from full control using simulation to override faults, to no control whatsoever, even restricting the display of faceplate interfaces)
- Allow or restrict the operator from changing operating parameters associated with the valve

Similarly, an instrument read module would do the following:

- Read an analogue instrument signal received via an analogue input card and scale it to the correct engineering units

- generate alarms and warnings whenever the signal is beyond a specific target value (either above or below);

- Alarm or warning may be time filtered (the condition must be present for a preset time before the alarm or warning is activated) and each will automatically reset when the signal is back within the acceptable range by a hysteresis amount.

- Generate out-of-range fault signals if the instrument is outside its normal calibrated range by more than a specified amount

This document contains a full list of all the standard modules that will be developed under this Project (listed in Section 8).

2.2.3 User interface

Although supervisory systems such as SCADA and HMI systems are outside the scope of this Project, the interface between these systems and the PAL software modules is not; and this must be clearly defined in order to provide the necessary signals to display and interact with the any supervisory system.

The interface between a supervisory system and the PAL software modules will be detailed in this document and will include example graphics that may be adopted by any supervisory such supervisory system.

The interface will be different for different types of equipment (the interface to an instrument will for example be completely different to that of a valve); however, a commonality of approach (and where possible, signals) will be adopted to give consistency to this interface.

2.2.4 Templates and documentation

A series of template and documentation modules will be provided to give worked examples of how the PAL software modules should be used in a control system project.

The *template* modules explain how to use and deploy the various *standard* and *application* modules and also the various organisation blocks (OBs) that may be required in various circumstances. The template modules provide detailed example usage for the standard modules and demonstrate different operating modes and configurations.

The *template* modules will be fully commented and will apply all the correct formatting and styling required by the PAL.

A full list of the *template* modules is given in § 11.1.

Documentation modules

The *documentation* modules contain summaries of the various styles and comment formats that can be copied and used within software modules. These are essentially quick reference *(proforma)* guides that can be used as the outline for *application* modules &c.

2.2.5 Hardware test environment

A reconfigurable test environment *(test rig)* will be provided with the necessary equipment needed to test the software developed under this Project.

The test rig will also be suitable as a test environment for subsequent projects developed using the PAL.

2.3 Background to the Project

The Practical Series of Publications has, for some time, had a partially developed set of standard library modules that have been used on various projects in the past.

Over recent years, there has been an increasing amount of such projects and it has been decided that these partially developed modules should be expanded to include a full range of *standard* modules and that modules should be formally structured into a software module library: the Practical Series Automation Library (or PAL).

There has been an increasing amount of pharmaceutical work in recent years and the necessity to reduce testing time and costs within these projects has been recognised; to this end the Practical Series Automation Library software module will be fully tested and validated, removing the need for the extensive design and documentation stages and the formal testing stages This will already have been done (and written) as part of this Project and will be issued in verifiable form by this Project.

2.4 Regulations and standards

The environments within which the PAL software can be used include pharmaceutical applications; as such the software must be written to the standards necessary for *Good Manufacturing Practice* (GMP), generally referred to as GxP[2].

The Validation Plan (VP), *[Ref. 002]* provides a justification and determination of validation requirements of this Project. The result of this determination is that this Project is a category 5 *"bespoke"* system and will comply with, and be written to, the standards necessary for GxP. These are the most rigorous standards used for control systems software and hardware development and use.

2 GxP is a general term for good … practice, where the x stands for various things, manufacturing, distribution, laboratory, clinical, engineering, &c.

The GxP requirements are encapsulated in the International Society for Pharmaceutical Engineering (ISPE) guidelines, referred to as Good Automation Manufacturing Practice (GAMP), currently at revision 5 (GAMP 5), *[Ref. 011]*. Systems that are written to the standards in GAMP 5 are said to be *compliant* systems that can be *validated*.

Validation is the process of making sure a computerised system (such as a PLC and its software) does precisely what it was designed to do; specifically, it is the exercise of correctly and traceably documenting every requirement of the system and making sure that that requirement is formally and exhaustively tested.

This Project, the Practical Series Automation Library, will be written to the standards specified in GAMP 5, it will be a validated and fully compliant GMP Project. The precise details of the validation process are specified in the Validation Plan (VP) document, *[Ref. 002]*.

2.4.1 Regulations, legislation and standards

Section 12 list the various regulatory, legislative and required standards that are to be applied to the hardware and software.

2.5 Assumptions and limitations

The Practical Series Automation Library of software modules will be developed as part of this Project. The scope of this development will be limited in this Project to just the Controller software, it will not include a library of supervisory control and data acquisition system (SCADA) or human machine interface (HMI) graphical objects.

The software will be written to interface with such system in a common manner, but the SCADA and HMI system will not be developed as part of this Project (though it is envisaged that this development will take place in a future project).

The PAL software will be validated to the GxP requirements that are applicable to the United Kingdom at the time of writing.

2.6 Nonconformity

There are no nonconformities between this document and the User Requirements Specification (URS) *[Ref. 003]*.

The URS specifies that the sequence control logic will be IEC 61131-3 *[Ref. 012]* compliant (see the section *Sequential logic control*, § 4.2.2 of the URS, *[Ref. 003])*; and indeed, the associated *standard* modules are compliant, satisfying the requirements of the URS.

There is however, a school of thought that the IEC implementation of sequence control logic has certain impracticalities; this is associated with the *terminating* phase of one step overlapping the *initialising* phase of the next step (both occur in the same PLC cycle, Section #9.39.3 contains a full description of this point). Engineering application often prefer that the sequence steps do not overlap in any way (the steps are completely independent); to satisfy this common engineering practice, a second, **non-IEC compliant**, version of the sequence logic modules is included, these maintain the segregation between steps.

The use of these modules is entirely optional.

2.7 Addressing the URS requirements

Where a particular point in the FS addresses a formal requirement raised in the URS, the point in the FS is given a paragraph number, this allows each point to be uniquely identified by section number and paragraph number. These specifications will be recorded in the Requirement Traceability Matrix (RTM), *[Ref. 004]*.

Paragraphs that are not numbered are not formally addressing a requirement; these may be introductions to a section, explanatory texts, notes or clarifying statements.

3 Hardware

(1) The Project hardware consists of a development platform that can be used to both develop and test the software modules produced as the primary purpose of this project.

(2) The development platform is in the form of a "test rig" that is configurable, and reconfigurable, to provide access to different interfaces, devices and instruments for the purposes of testing and demonstrating the functions of the developed software.

(3) The purpose of the test rig is to provide a set of (typical) devices and instruments that are common to most industrial applications, as such the test rig is equipped with:

- Two fail closed isolating valves[3] with position feedback
- Two fail open isolating valves with position feedback
- A single modulating valve[4] with position feedback
- A single direct online (on/off) motor with rotation sensor
- A single variable speed motor with encoder rotation detection
- A single type K thermocouple probe
- A single resistance thermometer (PT100 type)

---

[3] An isolating valve is a valve that is either opened of closed, it cannot hold an intermediate position. Normally, an isolation valve moves to a particular state if energised (either opened or closed, depending on the type of valve) and will return to the opposite state if power is removed. A normally closed valve is powered to open and returns to the closed state if de-energised, A normally open valve is powered closed and returns to the open state if de-energised.

[4] A modulating valve can be driven to any position (generally from fully closed to fully open and any position in between), modulating valves may also give an analogue signal to indicate the current position

(4) The valve limit switch signals, the rotation detection devices and temperature probes will all be wired to field terminals or to plugs and sockets to allow the system to be reconfigured to accommodate different device arrangements (for example, valves with two, one or no limit switch configurations)

(5) The test rig will also be equipped with various signal generators to simulate common instrument interfaces:

- 16 illuminated switches for the simulation of digital signals
- Two 0-10 VDC signal generators
- Two 4-20 mA signal generators
- A single function generator (sine, square, pulse, ramp, noise and arbitrary waveform generation) with ±10 VDC signal amplitude

(6) Monitoring functions will also be available with the following equipment:

- A dual channel oscilloscope
- Two configurable volt meters to display Controller analogue output signals

(7) The test rig is equipped with two Siemens Simatic Controllers and a touch panel human machine interface (HMI) as follows:

- Controller 1 — S7-1500 CPU 1515-2PN with IO cards
- Controller 2 — S7-1500 CPU 1511-1PN with IO cards
- HMI[5] — Simatic TP1200 touch panel

---

5 The software and configuration of the HMI does not form part of this project, however, it is anticipated that further projects will develop this aspect of the PAL software and as such a suitable HMI has been incorporated into the hardware design of the test rig (it being easier to incorporate it at this stage then modifying the panel under a later project)

(8) Two Ethernet networks are provided, the first (a standard Ethernet network) connecting the two Controllers and the HMI together, the second (an industrial Profinet network) connecting Controller 1 to a remote IO rack and the encoder rotation detector.

(9) The two Controllers and network arrangements are required to develop controller to controller communication software and the Profinet arrangement is the standard form for remote IO connections and this must also be testable.

(10) This section specifies the functions and facilities provided by the system hardware.

(11) The Hardware Design Specification *[Ref. 006]*, expands upon the functions and facilities listed here, identifying individual components and providing additional configuration information for the devices listed.

3.1 Hardware functions

3.1.1 General arrangements

(1) The test rig is modular and portable, a preliminary model is shown in Figure 3.1.

(2) The test rig has two primary components:

④ An electrical panel holding the Controller equipment, switch gear, signal monitoring and generation equipment, and various other electrical components

⑤ A test bed that holds the physical components, the motors, valves, field devices and various reconfigurable terminals and field wiring arrangements

(3) The electrical panel (Figure 3.2) is detachable from the test bed (Figure 3.3) for ease of portability and storage.

(4) All connections between the electrical panel and the test bed are via industrial connectors rated to at least IP65

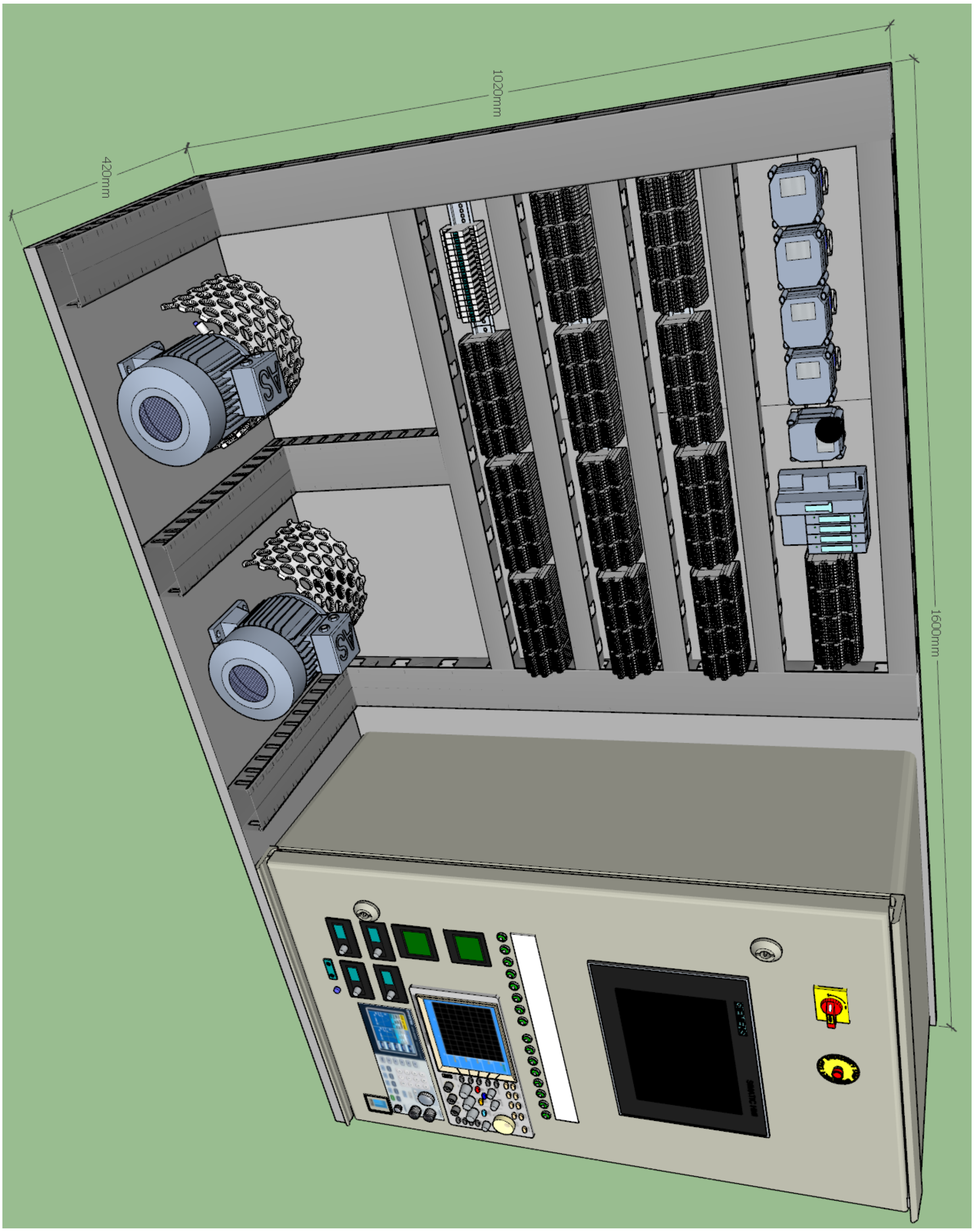


Figure 3.1 Test rig general arrangement

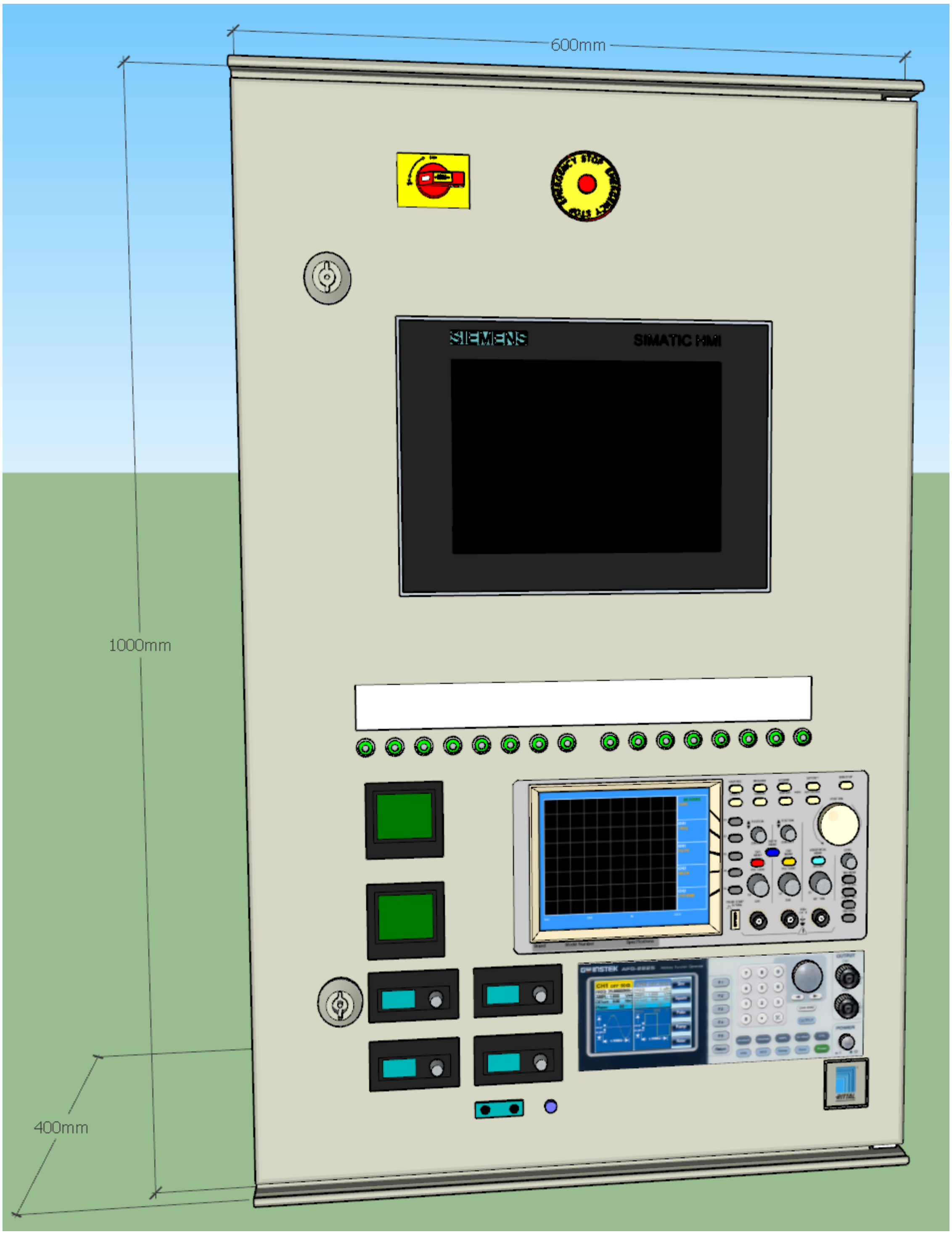


Figure 3.2 Test rig electrical panel

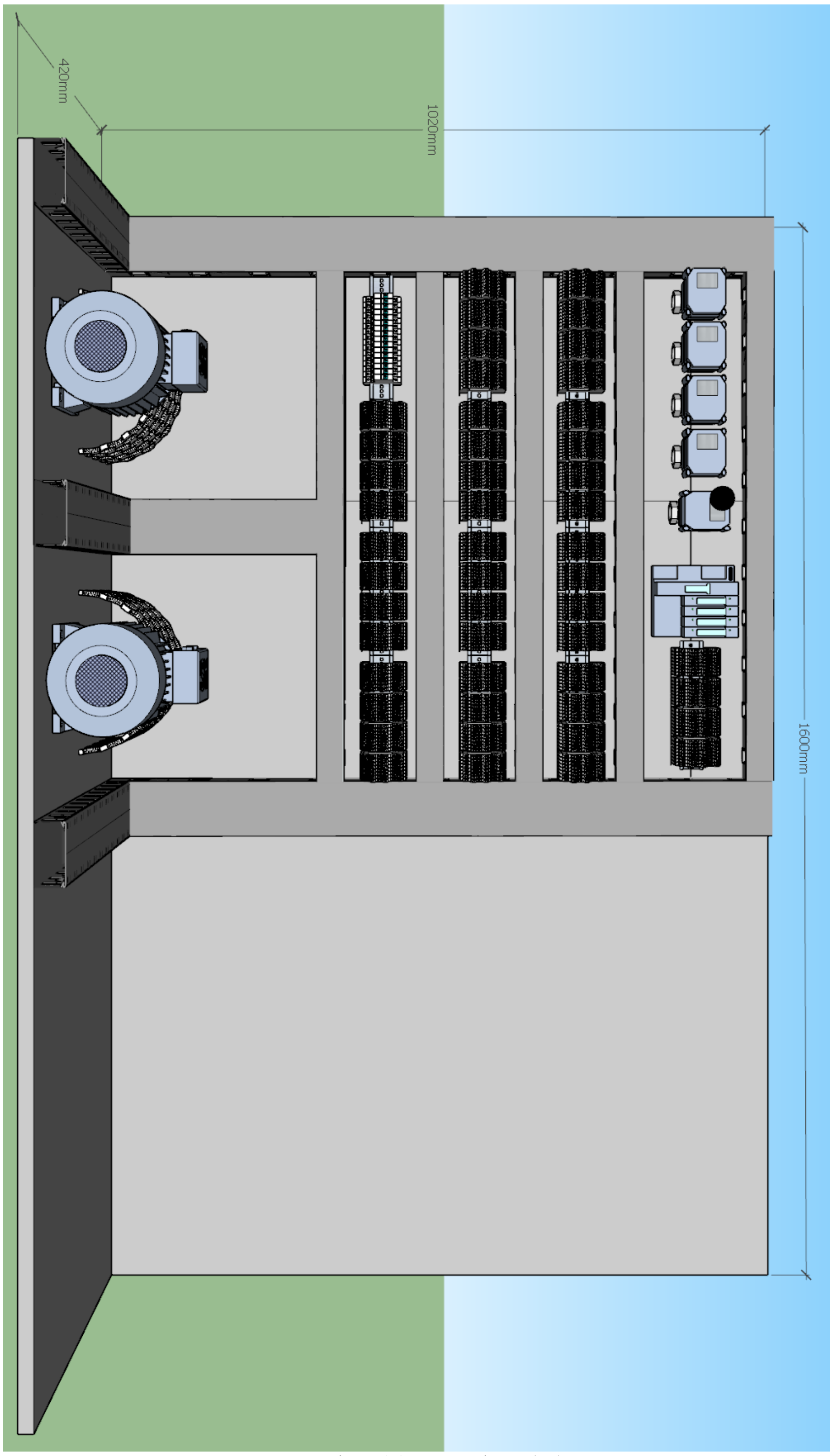


Figure 3.3 Test rig test bed

3.1.2 The test bed

(1) The test bed is constructed of 20mm thick medium density fibreboard (MDF) with dimensions of approximately 1020mm × 1600mm × 420mm (H × W × D).

(2) The test bed is equipped with the following devices:

- A single-phase direct online (DOL) motor (M001)
- A three-phase variable speed drive (VSD) motor (M002)
- The DOL motor will have an inductive proximity switch (PD001) positioned to detect rotation of the drive shaft
- The VSD motor will be directly coupled to a 13-bit encoder (ENC001) equipped with a Profinet interface
- Two normally closed motorised isolating valves (V001, V002) equipped with fully open and fully closed limit switches
- Two normally opened motorised isolating valves (V003, V004) equipped with fully open and fully closed limit switches
- A single 3-way modulating valve (CV001) with 4-20mA position control and 4-20mA position feedback. The valve will also be equipped with fully open and fully closed limit switches
- A Profinet remote IO rack equipped as follows:
    - 8 channel 24 VDC digital input module
    - 8 channel 24 VDC digital output module
    - 2 channel 4-20mA analogue input module
    - 2 channel 4-20mA analogue output module
- All remote IO will be wired to individual terminals on the test bed

- A series of test terminals that can be configured to match individual development requirements

- A series of 24 VDC and 0 VDC terminals to supply power to any additional test equipment

- Easily accessed fused terminals that supply each 24 VDC powered device

(3) All moving components (motor drive shafts, encoder coupling and internal components of the valves) are guarded to IP20 (finger safe), see Figure 3.4. The guards are fixed and permanent (i.e. they do not open, and are permanently fixed in position with bolts)

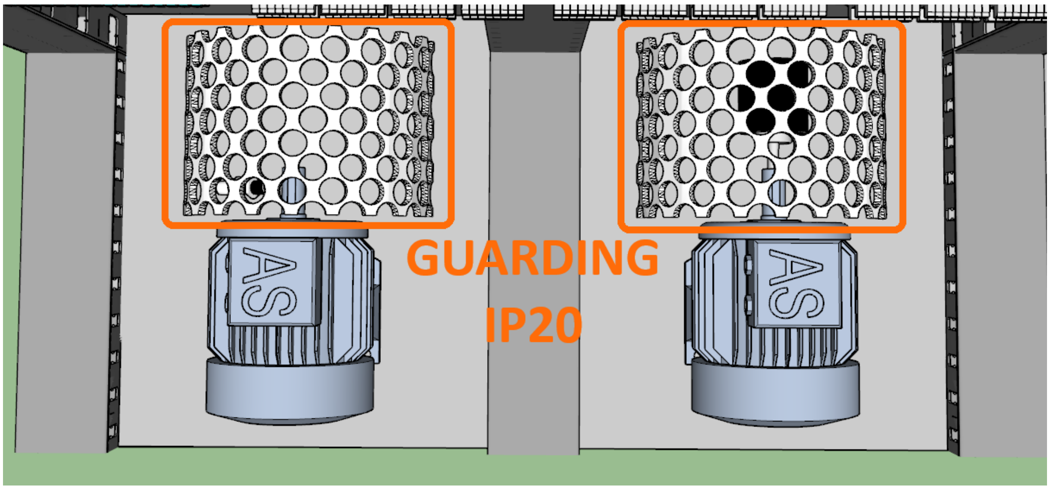


Figure 3.4 Guarding for drive shafts

(4) To prevent access to the internal workings of the valves, blanking caps are fitted to all valve pipework orifices.

(5) All valves are 24 VDC devices (extra low voltage) in terms of both motor operation and limit switch signals. The valve motors are entirely enclosed and cannot be accessed without disassembly.

(6) The single-phase motor has a supply voltage of 230 VAC and the three-phase motor has a phase-to-phase supply voltage rated at 400 VAC. The single and three-phase terminals and wire penetration are entirely enclosed and cannot be accessed without disassembly.

(7) The accessible test terminals available on the test bed all operate at 24 VDC only.

(8) 24 VDC is the only user accessible voltage available on the test bed.

(9) Various IO signals from the electrical panel (see § 3.1.4) are wired to the test terminals on the test bed. These provide connections for the various valve signals (limit switch, open/close demand signals, position feedback &c.), the proximity switch and provide wired connection points for any additional signals or instruments that maybe under test.

(10) All plugs and sockets are configured in an inherently safe state, the "live" conductors will always be connected to a socket that in the disconnected state protects the user from contact with the conductors.

(11) All electrical connections from the test bed to the electrical panel are via enclosed, industrial plugs and sockets. These are rated to at least IP65. If disconnected, there is no physical access to any powered pin within any connector (i.e. in the disconnected state, all sockets are rated to IP20).

(12) The test bed does not have its own power supply, all power is connected via the electrical panel (and this in turn has have a single mains power connection, see below).

3.1.3 The electrical panel

The electrical panel holds the Controller equipment, switch gear, signal monitoring and generation equipment, and various other electrical components

General arrangements

(1) The electrical panel is of sheet steel construction, finished in powder coated textured paint. The paint colour being RAL 7035 (light grey).

(2) The panel has a protection category of IP65 (the panel itself will be IP66, but this will be degraded to IP65 with the addition of door mounted equipment).

(3) The internal mounting plate is zinc plated.

(4) The panel is 1000mm × 600mm × 400mm (H × W × D) and has two mechanical locking points each requiring a profiled tool to open the panel.

(5) The panel door will be the full height and width of the panel and will have various cut outs to hold the signal generation and monitoring equipment.

Power supply and safety systems

(6) The electrical panel has a single mains connection point, requiring a single-phase 230 VAC electrical supply (50-60 Hz).

(7) The electrical supply is connected via a standard EN60309 16 A industrial 3-pole socket located on the right-hand side of the panel.

(8) The electrical panel has a single 3-pole isolator mounted on the front door of the panel; this disconnects power to all equipment within the electrical panel

(9) There is a single emergency stop button located on the front door of the panel. The button is latching (press to activate, twist to reset), EN60947-5-5.

(10) Pressing the emergency stop button removes all electrical energy from the two electrical motors (M001 and M002); power is also be removed from the four isolating valves (V001-V004), these will return to their failsafe states and the modulating valve (CV001), this will remain in its last position.

(11) The electrical panel provides a fused 24 VDC supply to terminals on the test bed (via a plug and socket arrangement) allowing additional instruments and devices to be powered as necessary.

(12) Mains voltages (single-phase and three-phase) are not directly available on the test bed, such supplies are connected directly to the two motors installed on the test bed and all terminations are secured behind fixed, permanent enclosures and entry to those enclosures is via cable glands. These supplies are connected to the electrical panel by uniquely keyed plugs and sockets (to prevent cross or incorrect connections).

Panel equipment

Figure 3.5 and Figure 3.6 show the internal and external (respectively) general arrangements for the electrical panel:

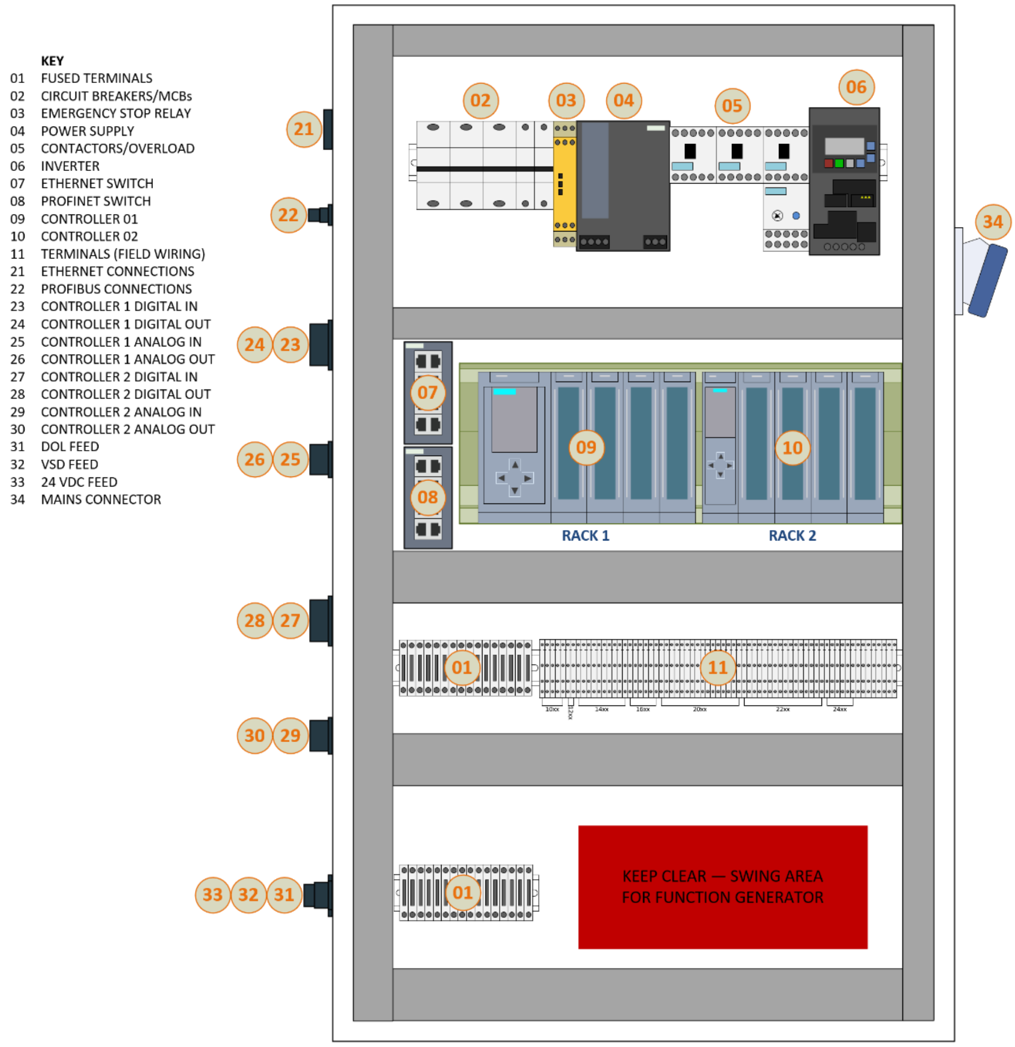
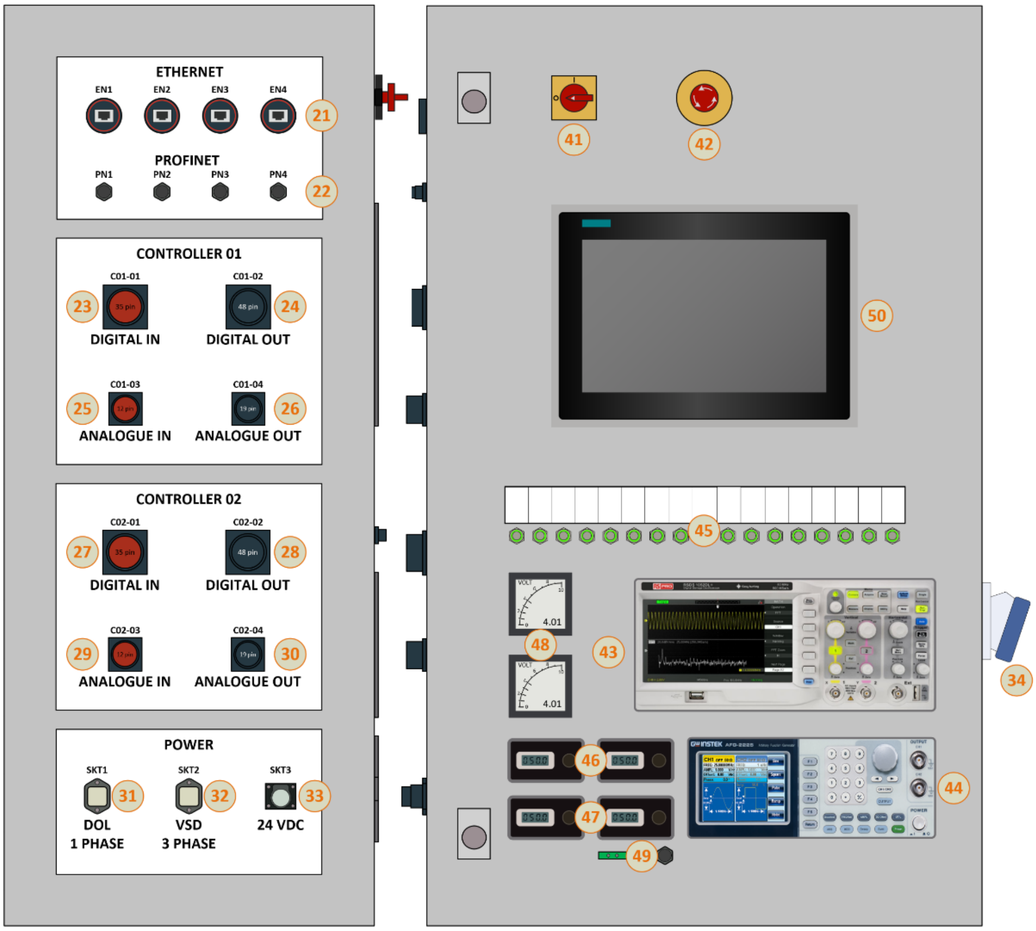


Figure 3.5 Electrical panel — internal arrangement



KEY

21	ETHERNET CONNECTIONS	31	DOL FEED	41	MAINS ISOLATOR
22	PROFIBUS CONNECTIONS	32	VSD FEED	42	EMERGENCY STOP BUTTON
23	CONTROLLER 1 DIGITAL IN	33	24 VDC FEED	43	OSCILLOSCOPE
24	CONTROLLER 1 DIGITAL OUT	34	MAINS CONNECTOR	44	FUNCTION GENERATOR
25	CONTROLLER 1 ANALOG IN			45	LAMPS & PUSH BUTTONS
26	CONTROLLER 1 ANALOG OUT			46	0-10 VDC GENERATOR 01
27	CONTROLLER 2 DIGITAL IN			47	4-20 mA CURRENT SOURCE 01
28	CONTROLLER 2 DIGITAL OUT			48	VOLT METERS
29	CONTROLLER 2 ANALOG IN			49	THERMOCOUPLE & RTD PORT
30	CONTROLLER 2 ANALOG OUT			50	HMI TP1200 COMFORT

Figure 3.6 Electrical panel — external arrangement

(14) Internally, the panel is equipped with the following primary components:

- S7-1500 Controller 1 — CPU1515-2PN
- S7-1500 Controller 2 — CPU1511-1PN
- Each controller will have the following IO cards
    - 32 channel 24 VDC digital input module
    - 32 channel 24 VDC digital output module
    - 8 channel UI/RTD/TC analogue input module
    - 8 channel UI analogue output module
- Switch gear is provided for the two motors:
    - M001 — direct online (DOL) equipped with contactor and overload
    - M002 — variable speed drive (VSD) equipped with a single-phase to three-phase inverter
- An Ethernet switch (linking controller 1, controller 2, the HMI and providing four external 10/100Mbs Ethernet ports)
- A Profinet managed switch (linking controller 1 to the remote IO and encoder located on the test bed)

(15) Externally, the panel door is equipped with

- 16 latching (push on, push off) illuminated switches, the switches being wired to individual Controller 2 digital inputs, the illuminations being wired to individual Controller 2 digital outputs
- Two 4-20 mA signal generators wired to individual Controller 1 analogue inputs
- Two 0-10 VDC signal generators wired to individual Controller 1 analogue inputs
- Two separate volt meters wired to individual Controller 1 analogue outputs
- A single thermocouple (type K) probe port, wired to a Controller 2 analogue input (the thermocouple probe will be provided)
- A single resistance temperature device (PT100 type) probe port, wired to a Controller 2 analogue input (the RTD probe will be provided)
- A dual channel oscilloscope
- A dual channel arbitrary waveform function generator (variable amplitude to a maximum of ± 10 V)
- A single touch panel human machine interface (HMI), see § 3.1.6
- Panel isolator and latching emergency stop push button

(16) The left side of the panel (Figure 3.6) is equipped with the various electrical connections (plugs and sockets) to link the electrical panel with the test bed.

(17) The right side of the panel holds the industrial mains socket.

3.1.4 IO signals and access

(1) The test rig is preconfigured and wired with the equipped devices being connected to specific fixed IO points as follows:

Symbol	Controller	Address	Rack/ Slot	Card Type	Signal	Range	Description
ESTOP_HEALTHY	01	I0.0	1-2	32×DI	24VDC	1/0	Emergency stop healthy/pressed
M001_RUNNING	01	I0.1	1-2	32×DI	24VDC	1/0	M001 is running/stopped
M001_TRIPPED	01	I0.2	1-2	32×DI	24VDC	1/0	M001 is heathy/tripped
M002_RUNNING	01	I0.3	1-2	32×DI	24VDC	1/0	M002 is running/stopped
M002_FAULT	01	I0.4	1-2	32×DI	24VDC	1/0	M002 is heathy/inverter fault
M001_ROTATION	01	I0.5	1-2	32×DI	24VDC	1/0	M001 rotation sensor (proximity PD001)
CV001_OPENED_LIM	01	I0.6	1-2	32×DI	24VDC	1/0	CV001 opened limit switch active/inactive
CV001_CLOSED_LIM	01	I0.7	1-2	32×DI	24VDC	1/0	CV001 closed limit switch active/inactive
V001_OPENED_LIM	01	I1.0	1-2	32×DI	24VDC	1/0	V001 opened limit switch active/inactive
V001_CLOSED_LIM	01	I1.1	1-2	32×DI	24VDC	1/0	V001 closed limit switch active/inactive
V002_OPENED_LIM	01	I1.2	1-2	32×DI	24VDC	1/0	V002 opened limit switch active/inactive
V002_CLOSED_LIM	01	I1.3	1-2	32×DI	24VDC	1/0	V002 closed limit switch active/inactive
V003_OPENED_LIM	01	I1.4	1-2	32×DI	24VDC	1/0	V003 opened limit switch active/inactive
V003_CLOSED_LIM	01	I1.5	1-2	32×DI	24VDC	1/0	V003 closed limit switch active/inactive
V004_OPENED_LIM	01	I1.6	1-2	32×DI	24VDC	1/0	V004 opened limit switch active/inactive
V004_CLOSED_LIM	01	I1.7	1-2	32×DI	24VDC	1/0	V004 closed limit switch active/inactive
M001_START_CMD	01	Q0.0	1-3	32×DQ	24VDC	1/0	M001 start command
M002_ENABLE_CMD	01	Q0.1	1-3	32×DQ	24VDC	1/0	M002 enable command
CV001_ENABLE_CMD	01	Q0.2	1-3	32×DQ	24VDC	1/0	CV001 enable command
V001_OPERATE_CMD	01	Q0.3	1-3	32×DQ	24VDC	1/0	V001 operate command (energise)
V002_OPERATE_CMD	01	Q0.4	1-3	32×DQ	24VDC	1/0	V001 operate command (energise)
V003_OPERATE_CMD	01	Q0.5	1-3	32×DQ	24VDC	1/0	V001 operate command (energise)
V004_OPERATE_CMD	01	Q0.6	1-3	32×DQ	24VDC	1/0	V001 operate command (energise)
VGEN1	01	IW256	1-4	8×AI	±10VDC	0-10VDC	Voltage signal generator 1
VGEN2	01	IW258	1-4	8×AI	±10VDC	0-10VDC	Voltage signal generator 2
CGEN1	01	IW264	1-4	8×AI	4-20mA	4-20mA	Current signal generator 1
CGEN2	01	IW266	1-4	8×AI	4-20mA	4-20mA	Current signal generator 2
M002_SPEED_ACT	01	IW268	1-4	8×AI	4-20mA	0-100%	M002 actual speed
CV001_POS_ACT	01	IW270	1-4	8×AI	4-20mA	0-100%	CV001 actual position
VMET1	01	QW256	1-5	8×AQ	0-10VDC	0-10VDC	Voltage meter 1 signal
VMET2	01	QW258	1-5	8×AQ	0-10VDC	0-10VDC	Voltage meter 2 signal
M002_SPEED_DEM	01	QW264	1-5	8×AQ	4-20mA	0-100%	M002 demanded speed
CV001_POS_DEM	01	QW266	1-5	8×AQ	4-20mA	0-100%	CV001 demanded position

Table 3.1 Controller 01 fixed input and output signals

Symbol	Controller	Address	Rack/ Slot	Card Type	Signal	Range	Description
PB01	02	I0.0	2-2	32×DI	24VDC	1/0	Push button 01 pressed/not pressed
PB02	02	I0.1	2-2	32×DI	24VDC	1/0	Push button 02 pressed/not pressed
PB03	02	I0.2	2-2	32×DI	24VDC	1/0	Push button 03 pressed/not pressed
PB04	02	I0.3	2-2	32×DI	24VDC	1/0	Push button 04 pressed/not pressed
PB05	02	I0.4	2-2	32×DI	24VDC	1/0	Push button 05 pressed/not pressed
PB06	02	I0.5	2-2	32×DI	24VDC	1/0	Push button 06 pressed/not pressed
PB07	02	I0.6	2-2	32×DI	24VDC	1/0	Push button 07 pressed/not pressed
PB08	02	I0.7	2-2	32×DI	24VDC	1/0	Push button 08 pressed/not pressed
PB09	02	I1.0	2-2	32×DI	24VDC	1/0	Push button 09 pressed/not pressed
PB10	02	I1.1	2-2	32×DI	24VDC	1/0	Push button 10 pressed/not pressed
PB11	02	I1.2	2-2	32×DI	24VDC	1/0	Push button 11 pressed/not pressed
PB12	02	I1.3	2-2	32×DI	24VDC	1/0	Push button 12 pressed/not pressed
PB13	02	I1.4	2-2	32×DI	24VDC	1/0	Push button 13 pressed/not pressed
PB14	02	I1.5	2-2	32×DI	24VDC	1/0	Push button 14 pressed/not pressed
PB15	02	I1.6	2-2	32×DI	24VDC	1/0	Push button 15 pressed/not pressed
PB16	02	I1.7	2-2	32×DI	24VDC	1/0	Push button 16 pressed/not pressed
LED01	02	Q0.0	2-3	32×DQ	24VDC	1/0	LED 01 illuminated/off
LED02	02	Q0.1	2-3	32×DQ	24VDC	1/0	LED 02 illuminated/off
LED03	02	Q0.2	2-3	32×DQ	24VDC	1/0	LED 03 illuminated/off
LED04	02	Q0.3	2-3	32×DQ	24VDC	1/0	LED 04 illuminated/off
LED05	02	Q0.4	2-3	32×DQ	24VDC	1/0	LED 05 illuminated/off
LED06	02	Q0.5	2-3	32×DQ	24VDC	1/0	LED 06 illuminated/off
LED07	02	Q0.6	2-3	32×DQ	24VDC	1/0	LED 07 illuminated/off
LED08	02	Q0.7	2-3	32×DQ	24VDC	1/0	LED 08 illuminated/off
LED09	02	Q1.0	2-3	32×DQ	24VDC	1/0	LED 09 illuminated/off
LED10	02	Q1.1	2-3	32×DQ	24VDC	1/0	LED 10 illuminated/off
LED11	02	Q1.2	2-3	32×DQ	24VDC	1/0	LED 11 illuminated/off
LED12	02	Q1.3	2-3	32×DQ	24VDC	1/0	LED 12 illuminated/off
LED13	02	Q1.4	2-3	32×DQ	24VDC	1/0	LED 13 illuminated/off
LED14	02	Q1.5	2-3	32×DQ	24VDC	1/0	LED 14 illuminated/off
LED15	02	Q1.6	2-3	32×DQ	24VDC	1/0	LED 15 illuminated/off
LED16	02	Q1.7	2-3	32×DQ	24VDC	1/0	LED 16 illuminated/off
RTD001	02	IW256	1-4	8×AI	PT100	-50 to 250°C	Resistance thermometer
TC001	02	IW264	1-4	8×AI	Type K	-200 to 300°C	Type K thermocouple

Table 3.2 Controller 02 fixed input and output signals

(2) All other (spare) IO points are wired to screw terminals on the test bed to allow other instruments and devices to be connected as required.

(3) All IO points on the remote IO rack are wired to screw terminals on the test bed to allow other instruments and devices to be connected as required.

3.1.5 Network arrangements

(1) The system has two Ethernet based networks; these are shown in schematic form in Figure 3.7:

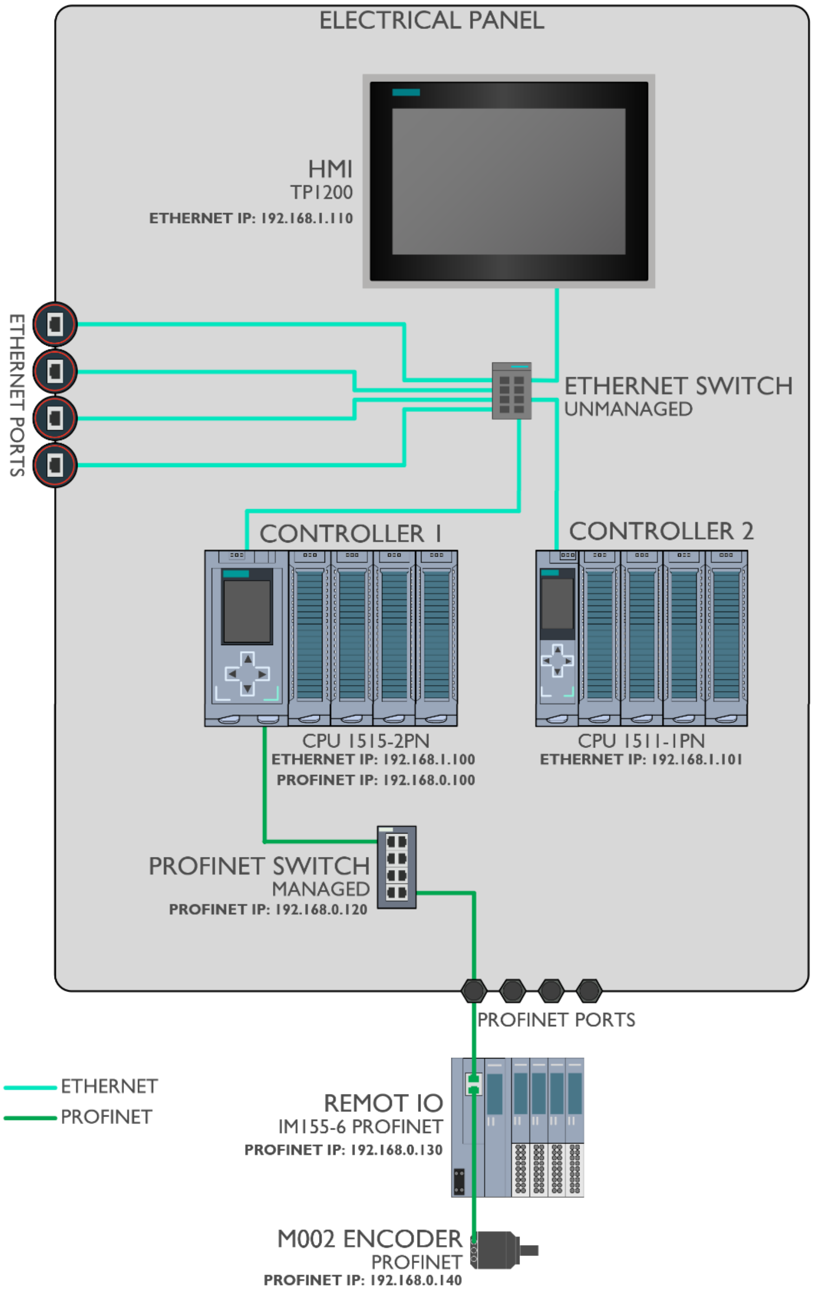


Figure 3.7 Network arrangements

(2) The first network is a standard Ethernet network connecting Controller 1, Controller 2 and the HMI together. This network extends to four additional RJ45 type ports on the side of the panel; allowing other devices (an engineering station or SCADA supervisory system for example) to be connected to the network.

(3) This standard Ethernet network uses the TCP/IP protocols and has the fixed IP addresses shown as *Ethernet IP* in Figure 3.7. An eight-channel unmanaged switch is used to link all the Ethernet devices and panel ports.

(4) The second network is a Profinet network, this being an industrial Ethernet based network suitable for the transmission of data between a Controller and field devices. The Profinet network connects Controller 1 (via its second communication port) to an eight-channel managed Profinet switch within the electrical panel.

(5) The Profinet switch is connected to four Profinet ports on the side of the electrical panel. One of these ports is used to connect the Profinet network to the remote IO rack on the test bed and from there to the Profinet encoder connected to M002 (the variable speed drive).

(6) The Profinet network again uses TCP/IP addressing for each device on the network, again these are the fixed IP addresses shown as *Profinet IP* in Figure 3.7

(7) The Ethernet network and the Profinet network are assigned to different subnets, the Ethernet network using subnet 192.168.**1**.nnn and the Profinet network using subnet 192.168.**0**.nnn. This division of subnets is a necessary requirement of the Profinet standards employed within the Simatic Controllers.

3.1.6 The HMI

(1) The electrical panel is equipped with a Siemens Simatic Touch Panel HMI, this is mounted on the door of the electrical panel.

(2) The HMI is touch operated (no keys, buttons or mouse) and will have a screen resolution of 1280 × 800 pixels.

(3) Only the HMI hardware is provided, the unit will not be programmed or configured as part of this Project.

Note: *Although the HMI does not form part of this Project, it is anticipated that further projects will develop this aspect of the PAL software and as such a suitable HMI has been incorporated into the hardware design of the test rig (it being easier to incorporate it at this stage then modifying the panel under a later project)*

3.1.7 The Controller hardware

(1) The test rig is equipped with two Siemens Simatic Controllers, the first is based on a mid-range processor, the CPU 1515-2PN, this has two communications port, the first port (X1) is connected to the standard Ethernet network. The second port (X2) is used as a Profinet interface and will connect to the remote IO rack and the Profinet encoder associated with M002.

(2) The second processor is based on a low range CPU 1511-1PN processor. This has a single communication port (X1) that is connected to the standard Ethernet network.

(3) The purpose in having two processors is to allow the development and testing of communication modules capable of transferring data between processors.

(4) Both controllers are equipped with identical sets of ET200MP IO cards:

SLOT	CARD TYPE	PART NO.	DESCRIPTION
2	DI 32 × 24VDC	6ES7521-1BL00-0AB0	32 channel digital input card
3	DQ 32 × 24VDC	6ES7522-1BL01-0AB0	32 channel digital output card
4	AI 8 × UI/RTD/TC	6ES7531-7KF00-0AB0	8 channel analogue input card
5	AQ 8 × UI	6ES7532-5HF00-0AB0	8 channel analogue output card

Table 3.3 Controller 01 and 02 IO cards

(5) Controller 1 is designated rack 1 and has the following arrangement:

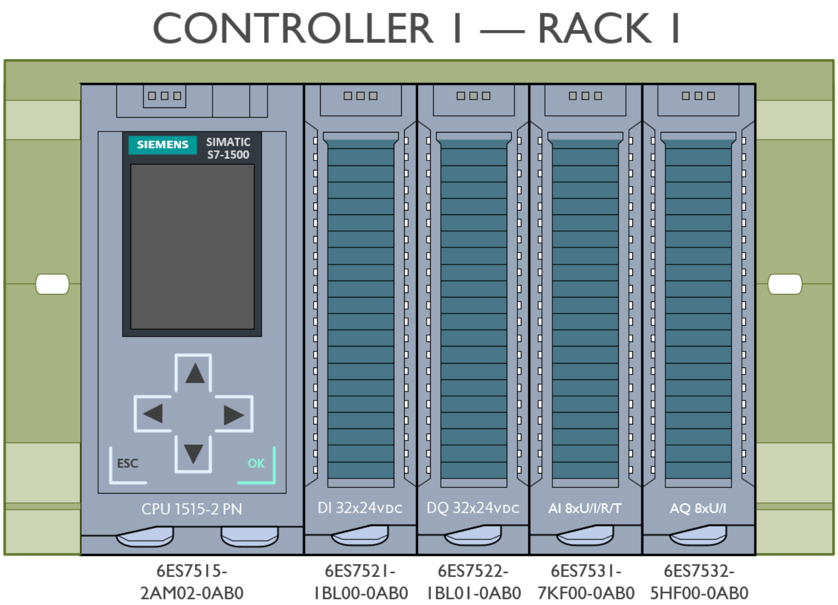


Figure 3.8 Controller 1/Rack 1 arrangements

(6) Controller 2 is designated rack 2 and has the following arrangement:

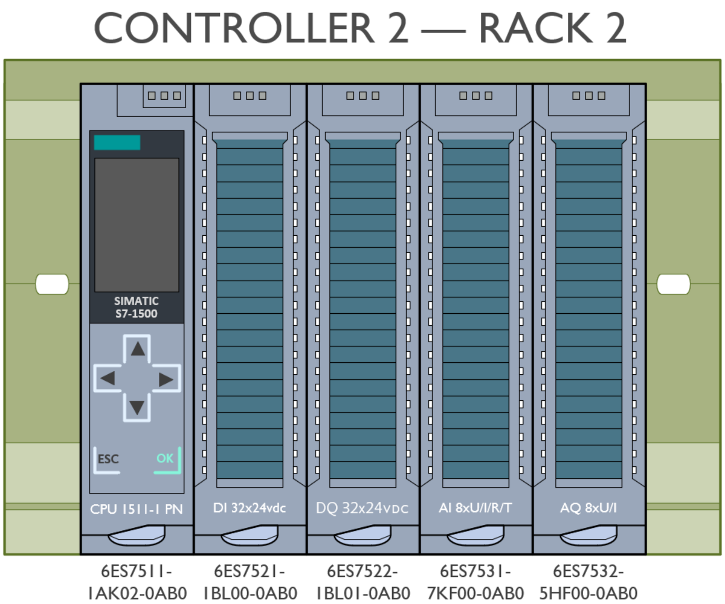


Figure 3.9 Controller 2/Rack 2 arrangements

(7) The remote IO rack interfaces directly with Controller 1 via the Profinet network. The interface module is an IM 155-6 standard ET200SP interface.

(8) The remote IO rack is equipped with the following IO modules:

SLOT	CARD TYPE	PART NO.	DESCRIPTION
1	DI 8 × 24VDC	6ES7131-6BF01-0BA	8 channel digital input card
2	DQ 8 × 24VDC	6ES7132-6BF01-0BA0	8 channel digital output card
3	AI 2 × UI/RTD/TC	6ES7134-6HB00-0CA1	2 channel analogue input card
4	AQ 2 × UI	6ES7135-6HB00-0CA1	2 channel analogue output card

Table 3.4 Controller 01 and 02 IO cards

(9) The remote IO is designated rack 3 and has the following arrangement:

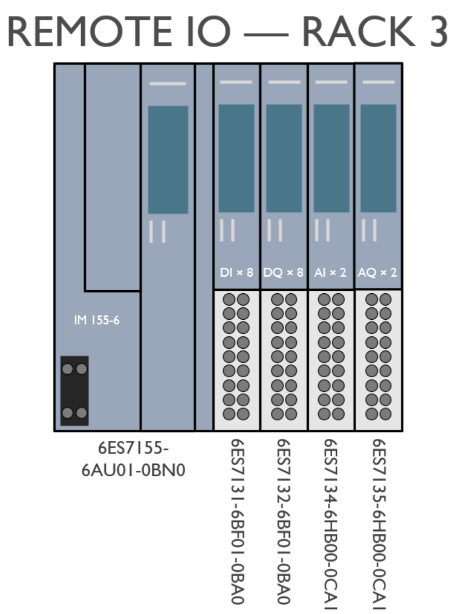


Figure 3.10 Remote IO/Rack 3 arrangements

BLANK PAGE

4 The controller software and structure

(1) The PAL software is intended to run within the S7-1500 and S7-1200 ranges of Siemens Simatic Controllers, as such the PAL software must be compatible with the internal structures present within these Controllers.

(2) The S7-1500 and S7-1200 ranges of Controllers both operate in the same manner and (largely) support the same software modules, software commands and have the identical operating structures within them.

(3) The S7-1200 is restricted in terms of capacity (it supports fewer blocks in total and is restricted in terms of the amount of IO modules that can be connected to it), and is also restricted in terms of the programming languages supported, the S7-1200 does not support the statement list STL[6] programming language; however, STL will not be used by the PAL software, All PAL software is written using ladder logic[7].

(4) Other restrictions apply to the S7-1200, the amount of data that can be transmitted over communications networks is limited (for example) and this has some impact on certain software modules, where such restrictions exist, this is explained in the relevant Software Module Design Specification (SMDS) *[Ref. 008]*.

(5) The following sections explains the pertinent points of the Controller software, its underlying structures and how these structures are adapted to the PAL software modules.

(6) All the software developed as part of the PAL is developed using the Siemens Simatic programming environment: TIA Portal Professional, Version 16.

---

6 STL or statement list is a text-based programming language similar to assembler language

7 Ladder logic is a graphical programming language widely used to programme Controllers and PLCs

4.1 Internal structure of the Controllers

4.1.1 Programmable blocks

(1) The Simatic controllers are programmed using blocks of different types, there are three programmable (blocks that contain software instructions) block types:

①	Organisation block (OB)	Interrupt driven block called in response to a specific event detected by the Controller operating system
②	Function (FC)	A subroutine (with or without parameters) used to structure the software or handle recurring or complex functions
③	Function block (FB)	Similar to an FC but with an allocated retentive data area

(2) All these blocks are *user blocks*; i.e. they are blocks that the user can programme, configure and edit. These blocks are used to subdivide the controller programme into smaller, self-contained modules that perform specific aspects of the programme (e.g. controlling emergency stops, handling communications, operating a valve &c.).

Organisation Blocks (OBs)

(3) Organisation blocks (OBs) serve as the interface between the Controller operating system and the user programme; OB 1, for example, the main organisation block is called at the start of every Controller cycle and is the only user block that the Controller will execute automatically (all other user blocks must be called by elements within the user programme).

(4) Other OBs are called in response to certain events: hardware interrupts, timed interrupts, Controller faults &c. and are given specific numbers, these are discussed in detail in § #5.4.35.4.3.

Functions (FCs)

(5) Functions (FCs) are used to subdivide a programme into meaningful sections or are used to handle frequently recurring or complex functions; a typical example would be to have a FC that control a specific device (a valve for example) and then repeatedly call this FC for each such device in the system.

(6) Using FCs to divide a programme into meaningful sections is common practice and makes for better structuring of the software; allowing the software to be more easily navigated and faults to be readily identified.

(7) This subdivision of the Controller programme will be widely applied within the PAL and will have the prescribed structure detailed in Section 5.

(8) FCs will form the vast majority of blocks within the PAL.

Function Blocks (FBs)

(9) Function blocks are a special version of functions that are automatically assigned a data block within which they can store function block specific data.

(10) In practice, FBs are not used in the PAL. However, where third-party software is required (to interface to specific equipment) these are often provided as FBs and their use is permitted.

(11) The PAL does not restrict the use of FBs in any way, it simply does not require any itself for the library modules within it.

4.1.2 Data storage blocks

(1) The Simatic controllers use data blocks (DBs) to store data for the user programme.

(2) Data blocks support all the standard data types available to the controller: Booleans, integers, bytes, floating point numbers, strings &c. In addition, DBs can be configured using user created data structures, these data structures are referred to as *User Data Types* (UDTs).

(3) Both DBs and UDTs are discussed in the following sections:

Data blocks (DBs)

(4) DBs are configurable by the user, but do not contain programming instructions (unlike the programmable blocks of the previous section), they hold data specified by the users (variables, constants, working values &c). The data stored in a DB can be anything and of any supported type (Booleans, integers, byte, floating point numbers, strings &c.). The structure and configuration of a DB is entirely at the discretion of the user; DBs are a very flexible and convenient mechanism for storing user information.

Instance data blocks (iDBs)

(5) Instance data blocks are a used by function blocks (FBs) as retentive data storage areas. These preserve data between successive calls of the block and are a requirement when using function blocks. Each call of a function block requires its own iDB.

User Data Types (UDTs)

(6) The PAL will rely heavily on the use of data structures to pass information between modules. UDTs are used to define the internal structure of DBs and can be passed as parameters into functions (FCs) and function blocks (FBs). Within the Siemens Controller these data structures are variously called *User Defined Data Types* or *User Data Types* or *PLC Data Types*).

(7) These terms are interchangeable, all meaning a data structure (a collection of named variables made up of standard data types, grouped together in a named structure). The original name (predating TIA Portal) was User Defined Data Type (UDT), with the advent of TIA Portal this became either a User Data Type (again UDT) or PLC Data Type (PDT). They all mean the same thing (a data structure).

(8) For clarity, the term UDT (User Data Type) is used to specify a user defined data structure (or any of the other names it uses).

4.1.3 Built in system blocks

(1) The Simatic Controllers and the TIA Portal programming environment have built in *system* blocks that perform specific functions (for example, a PID control loop,), these blocks will always be used in preference to developing a new block with similar functionality.

(2) These built in system blocks are pre-configured functions (FCs) and function blocks (FBs) written and issued by Siemens, they are given numbers in the range 1-999 (this is a reserved numbering range, reserved for third-party software, and is not occupied by any of the PAL modules, see § #5.15.1).

(3) Where system function blocks are used, these, like all FBs, require an instance DB (see § 4.3.3); these function blocks will generally be contained *(called from)* within a standard module, and this standard module will always be a function FC, this standard module can be considered a *wrapper* for the system function block. To accommodate the need for an instance DB required by the contained system function block, the instance DB to be used will be passed as a parameter to the standard function.

(4) Some system blocks have their own system data structures (referred to as *system data types*), these are similar to UDTs but are predefined within the TIA Portal programming environment, where such system data types are required, they will be installed and issued as part of the PAL software).

4.1.4 Block numbering, quantities and number ranges

(1) The number of blocks that can be used in a Controller program is entirely dependent on the processor running that programme. The pertinent values are shown here:

CPU Order No.	1511 6ES7511-1AK01-0AB0	1513 6ES7513-1AL01-0AB0	1515 6ES7515-2AM02-0AB0	1516 6ES7516-3AN01-0AB0	1517 6ES7517-3AP00-0AB0	1518 6ES7518-4AP00-0AB0
Total No. of Blocks (all blocks)	**2000**	**2000**	**6000**	**6000**	**10000**	**10000**
No. of Functions (FC)	2000	2000	6000	6000	10000	10000
FC No. Range	1-65535	1-65535	1-65535	1-65535	1-65535	1-65535
No. of Function blocks (FB)	2000	2000	6000	6000	10000	10000
FB No. Range	1-65535	1-65535	1-65535	1-65535	1-65535	1-65535
No. of Data blocks (DB)	2000	2000	6000	6000	10000	10000
DB No. Range	1-59999	1-59999	1-59999	1-59999	1-59999	1-59999

Table 4.1 S7-1500 CPU number of blocks

CPU Order No.	1211C 6ES7211-0AE40-0XB0	1212C 6ES7212-1AE40-0XB0	1214C 6ES7214-1AG40-0XB0	1215C 6ES7215-1AG40-0XB0	1217C 6ES7217-1AG40-0XB0
Total No. of Blocks (all blocks)	**1024**	**1024**	**1024**	**1024**	**1024**
No. of Functions (FC)	1024	1024	1024	1024	1024
FC No. Range	1-65535	1-65535	1-65535	1-65535	1-65535
No. of Function blocks (FB)	1024	1024	1024	1024	1024
FB No. Range	1-65535	1-65535	1-65535	1-65535	1-65535
No. of Data blocks (DB)	1024	1024	1024	1024	1024
DB No. Range	1-59999	1-59999	1-59999	1-59999	1-59999

Table 4.2 S7-1200 CPU number of blocks

(2) All S7-1500 CPUs support at least two thousand blocks and this is more than sufficient for virtually any application.

(3) The S7-1200 CPUs all support a maximum of 1024 blocks, this is a practical amount for the simpler type of application at which the S7-1200 CPUs are targeted.

(4) The Practical Series Automation Library is designed to fit in the smallest of the S7-1500 CPUs; the library itself will also fit in the S7-1200 CPUs, but number and complexity of the application modules is constrained by the restrictions of the S7-1200 range.

(5) Irrespective of the CPU (and irrespective of the range, i.e. S7-1500 or S7-1200), the range of numbers that can be assigned to a given block are the same (i.e. any CPU can have a function block with the number in the range 1-65535, only the total number of blocks is limited, not the numbers that can be assigned to them).

(6) The PAL will use this capability to assign meaningful number ranges across all CPUs and Controller ranges:

BLOCK TYPE	PERMISSIBLE NUMBER RANGE	PAL NUMBER RANGE IN USE
FB, FC	1-65535	1-60999 (61000 onwards reserved for doc modules)
DB	1-59999	1-59999
OB	1-32767 (not inclusive)	1-122

Table 4.3 Block number ranges

(7) Organisation blocks typically have predefined (default) numbers, depending on function, in the range 1-122. It is possible to re-allocate these numbers anywhere in the range 123-32767; however, the PAL will only uses the default (automatically assigned) numbers.

(8) The permissible number range of FBs and FCs is wider than that for DBs. The PAL will uses block numbers to denote particular functions; these numbers need to be applied to both programmable blocks (FBs and FCs) and data blocks (DBs). To ensure that all block types can be allocated the same range of numbers, the PAL will only use block numbers in the range 1-59999 for standard and application (obviously, not every number in this range is used). Template modules extend outside this range (up to 60999), however, template modules that give examples of the application modules are in the range 1-59999. The range 60000-60999 is used for template modules that give examples of specific organisation block usage (see § #11.1.211.1.2)

(9) The range 61000-65535 is used by the PAL to store the example documentation modules.

(10) These number ranges have been split further to allocate different number ranges to the different block and data block functions within the PAL. The PAL will use the following number ranges for the specified module classifications:

NUMBER RANGE	FC/FB CLASSIFICATION	ABBREVIATION	DB/UDT CLASSIFICATION
00001-19999	Standard modules	Std	Static data storage
20001-39999	Application modules	App	Dynamic data storage
40000-59999	Template modules (application)	Temp	Instance data blocks
60000-60999	Template modules (interrupts)	Temp	N/A
61000-65535	Documentation modules	Doc	N/A

Table 4.4 Block and number allocations for the PAL

4.2 Execution of Controller software

(1) All Siemens Simatic Controllers (S7-1500 and S7-1200) are event driven devices, the CPU only ever responds to certain specific events (or interrupts). The CPU responds to a specific event by executing a particular organisation block (OB).

(2) For example, if a CPU is started (either by applying power or switching the device from STOP to RUN) it will execute the *start-up* organisation block (OB 100). If OB 100 were to call any functions or function blocks, these would also be executed.

4.2.1 Cyclic programme execution

(1) The principal event for running the PAL software is the main cyclic event interrupt. The Controller triggers a cyclic event that cause the Controller to write output data to the output cards, read input information form the input cards and then execute the main cycle interrupt by calling organisation block 1 (OB 1), any user programme, and any blocks (FCs and FBs) configured by the user and called from within OB1 will also be executed. When the end of OB1 is reached, the Controller retriggers the cyclic event and the process is repeated indefinitely (see Figure 4.1):

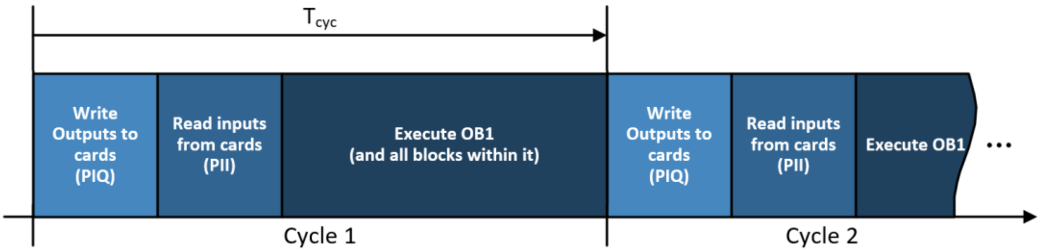


Figure 4.1 Cyclic event interrupt (OB 1)

(2) The PAL software runs predominantly from within OB 1 (the main cyclic event organisation block), there will however, be support for additional interrupt events, these can be timed interrupts (occurring at a specified interval of microseconds, or at a particular time of day &c.), hardware interrupts (occurring when a particular signal is detected), fault interrupts (for card failure, loss of signal, programming error &c.).

(3) These additional interrupt events all have a higher priority than OB 1 and will interrupt the execution of OB 1, causing the programme execution to jump to the associated interrupt OB and execute any programme that is contained within it. Once the interrupt OB has been executed, OB 1 will be resumed from its last point:

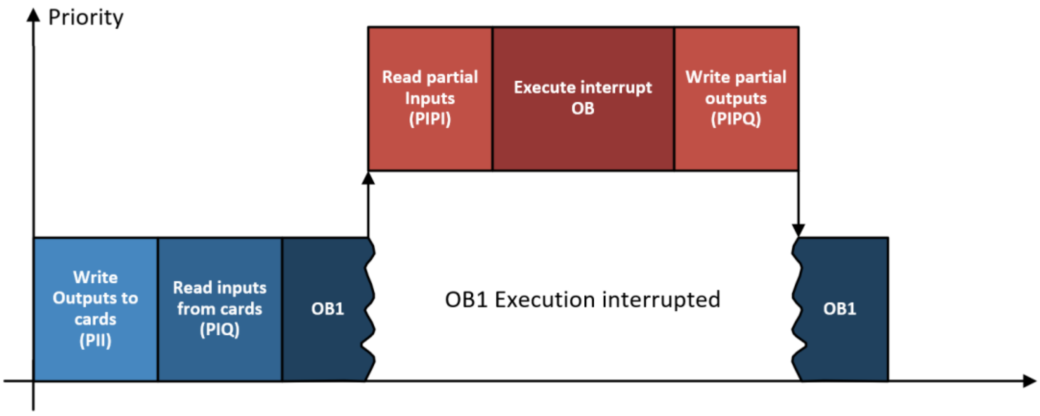


Figure 4.2 Interrupting OB 1 execution

(4) If an additional interrupt occurs whilst the first interrupt is active, and the additional interrupt has a higher priority than the first interrupt, this interrupt, will in turn be interrupted:

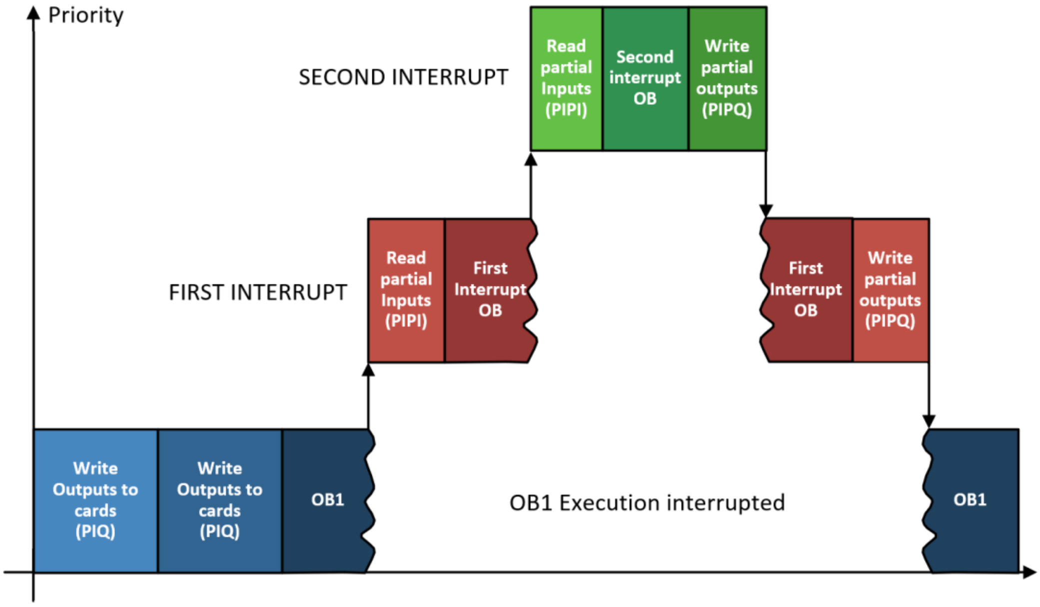


Figure 4.3 Multiple interrupts with increasing priorities

(5) If the second interrupt has the same, or a lower priority than the first interrupt, the second interrupt would be executed immediately after the first was completed:

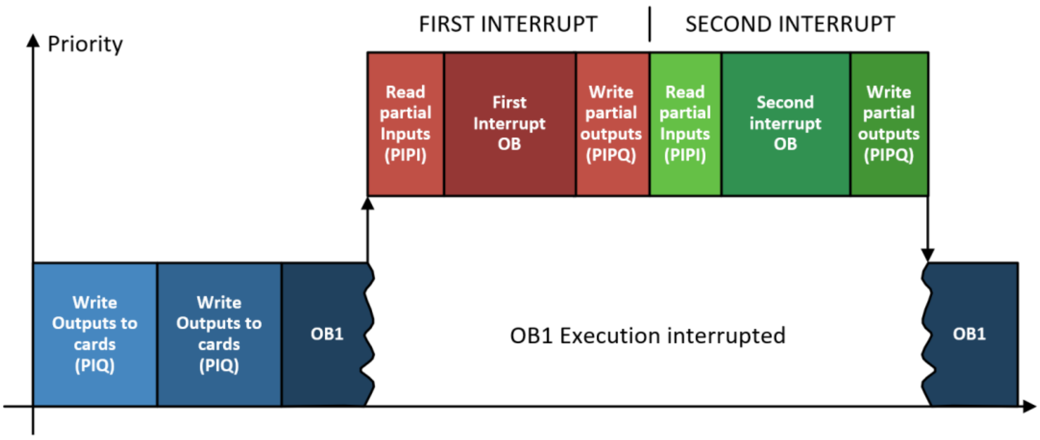


Figure 4.4 Multiple interrupts with the same or lower priority

(6) OB 1 has the lowest interrupt priority and any other interrupt will take precedence over it.

(7) The PAL will include preconfigured interrupt OBs that record the exact time and date the interrupt was called.

4.2.2 The process image

(1) The process image is a mechanism internal to the Controller (and executed automatically by the Controller operating system); essentially, it is the reading and writing of all the input and output card data and copying it either to (in the case of inputs) or from (in the case of outputs) an internal storage area within the Controller (referred to as *system memory*). This storage area is called the *process image* (PI), the process image has two components: the *process image of inputs* (PII) and the *process image of outputs* (PIQ).

(2) The process image is essentially a *snap-shot* of the state of all IO signals taken at the start of the Controller cycle and stored in the system memory of the controller.

(3) The concept of a process image is common to all PLCs, not just the Siemens Simatic Controllers. It generally provides the following benefits:

- The signal state of an input is the same throughout the Controller cycle (it gives signal consistency to all software elements within a programme cycle)
- Access to the process image is considerably faster than accessing the IO cards directly
- The state of outputs can be read by the user programme (outputs to cards are write only and the data cannot be read-back). The reading of output states in the process image is possible because the data is stored in system memory which has read/write access
- Multiple state changes of an output during the Controller cycle will have no direct effect on the output from the card, the final state of the output will not be written to the card until the end of the cycle (effectively at the start of the next cycle)

(4) The PAL is fully compliant with the process image concept and will expect all IO signals to use the process image.

4.2.3 Process images partitions

(1) Process image partitions (PIP) are only available to the S7-1500 range of Controllers the S7-1200 range simply has the cyclically driven process image (§ 4.2.2).

(2) Process image partitions are optional and can be applied to any interrupt driven event that triggers the execution of a particular organisation block.

(3) IO can be assigned specifically to a process image partition, and this data will then be updated whenever the associated OB is triggered.

Note: *When using process image partitions, the whole IO module must be assigned to a particular PIP, it is not possible to add certain signal within a module to one PIP and other signals to a different PIP.*

(4) The process image partition is designed to be used where it is necessary to update the process image mid-cycle; for example if a timed interrupt were set to interrupt every 2 ms and the OB 1 cycle time were 30 ms, then the interrupt would occur approximately 15 times within a particular cycle, if the interrupt did not use a process image partition, any IO signals that were being used by it would have the same state each time (the process image would only update at the start of the OB 1 cycle), the PIP forces the IO states associated with the interrupt OB to be updated each time it is called.

(5) Process image partitions again have two components: process image partition of inputs (PIPI) and process image partition of outputs (PIPQ). Up to 31 separate process image partitions can be supported by the S7-1500 range of Controllers.

(6) The use of process image partitions is not common practice (the facility is used under very specific conditions); the PAL however, is compliant with the process image partitioning concept and it may be used wherever it is required by the user.

4.2.4 Common CPU properties

(1) The PAL is not associated with a particular CPU; it will work on any S7-1500/1200 CPU. It does however require that certain property settings associated with the selected CPU are activated (and some deactivated).

(2) Generally, the PAL uses the default settings for CPU properties (minimising the changes from the default arrangements); however, there are some CPU settings changes that are required:

- System and clock memory allocations
- Communication settings

(3) Specifically, the following setting must be adopted within the CPU properties (accessed via TIA Portal):

System and clock memory		
Area	**Option**	**Setting**
System memory bits	Enable the use of system memory byte	This is unticked by default must not be enabled
Clock memory bits	Enable the use of clock memory byte	This must be enabled; the box must be ticked
Clock memory bits	Address of the clock memory byte (MBx)	Set to the value 10

Protection and security		
Area	**Option**	**Setting**
Connection mechanisms	Permit access with PUT/GET communications from remote partner	This must be enabled; the box must be ticked

Table 4.5 Default CPU setting adjustments for the PAL

4.3 The passing of data between modules

(1) The Siemens Controller functions (FCs) listed in § 4.1.1, will form the basis of all software modules within the PAL, each module being assigned to a particular function or function block.

(2) Each PAL standard software module is stored in a function (FC) and each module will require information to be passed to that module, this could be simple information such as the state of a valve limit switch (a simple on or off digital input) or may be a more complicated data structure determining the full range of options and configuration for the module.

(3) All this information is passed in the form of *parameters* to the blocks. The blocks that hold standard modules will not directly access any data within the Controller (i.e. directly access an IO point using a hard-coded reference), all data is passed to the block indirectly through the use of block parameters.

(4) All data passed to a block will either be Input/Output signals (assigned to the IO cards attached to the Controller) or data stored in data blocks, this data will always be specified in the form of UDTs.

(5) Memory bits will only be used to implement the Controller clock memory functions (see § #6.16.1) and to store specific Controller timing and logic state signals (see § #5.5.35.5.3). Memory bits **will not** be used to store programming data, this will always be done in data blocks, generally with the use of UDT data structures.

4.3.1 Block parameters

(1) Both FCs and FBs accept parameters as a form of passing data to and from the block, there are four types of formal parameters that can be used by an FC ore FB:

① INPUT

② OUTPUT

③ INOUT

④ RETURN

(2) The fourth group (RETURN) will not generally be used within the PAL (this is common practice within wider PLC programming circles). It is included to make the blocks compatible with the IEC requirements for programming languages. By default, the RETURN parameter is given the same symbolic name as the block and is declared as a VOID data type (VOID types are essentially "empty" data types that have no value and cannot hold a value). If the RETURN parameter is declared as a void, it will not be visible when the block is called, again this is standard procedure within the PAL.

(3) The remaining parameter types (INPUT, OUTPUT and INOUT) are widely used throughout the PAL (particularly by the standard blocks).

(4) These three types of parameters have the following definitions and will have the following uses within the PAL:

PARAMETER	DEFINITION	PAL USAGE
INPUT	Read only data — can be read by a block but not modified	For passing digital and analogue input signals to the block For passing read only UDT (static) data to a block
OUTPUT	Write only data — can be written to by a block but not read (attempting to read the data will return an error)	Used by the block to write to digital and analogue output signals
INOUT	Data can be both read and written to by the block	For reading and writing UDT (dynamic) data by the block

Table 4.6 Common block parameter types and their usage

(5) Blocks within the PAL use individual block parameters to pass input and output (IO) signals to the block, these are *real* IO signals assigned to IO cards (not internal memory bits within the Controller). These are assigned on a signal-by-signal basis, and each signal has its own parameter.

(6) For example, a standard PAL module to monitor and control an isolating valve would have INPUT parameters to pass the state of the valve open and closed limit switches to the module and an OUTPUT parameter to either energise or de-energise the valve (i.e. make it open or close).

4.3.2 Data storage and passing of data to blocks

(1) In addition to IO signals, each module will generally require a considerable amount of additional data to be stored, this data will reflect the configuration of the block (e.g. for a valve device driver this will hold valve operation times, determine the number and type of limit switches available to the valve, whether the valve is energise to open or energised to close &c.) and the current state of the block (e.g. is the valve currently opened or closed, is the valve in a fault condition, is it in the process of changing state &c.).

(2) Depending on the nature of the module, there may be a considerable amount of such data and all this data will be stored in data blocks. Within the PAL, this data will fall into two categories:

① Static data

② Dynamic data

(3) Static data specifies constant (preset) values that have some meaning for the block in question (e.g. the opening time of a valve, the hysteresis of an alarm setpoint, limit switch arrangements for a valve &c.). Static data does not change (the data is usually configured during the commissioning of the plant and then remains fixed and unchanging for the lifetime of the plant).

(4) Dynamic data is live, operating data (e.g. if a valve is in the process of opening, the elapsed time of the operation will be stored in a dynamic data area).

(5) To expand on this example, if a valve is designed to change from closed to open within a maximum of 10 seconds, then the static data would have some variable that would be fixed at a value of 10.0 (seconds). The dynamic data would have a variable that counted down from the 10.0s value specified by the static data to zero when the valve was instructed to open.

(6) This data, whether static or dynamic must be passed to the block as parameters. To do this, the data will be configured as data structures within the data blocks. These data structures will be configured as user data types (UDTs). Each block will generally have two such structures, one for static data and one for dynamic data; these structures will be unique to the block in question.

(7) Static data will be passed to a block via an INPUT parameter (i.e. read only), this is data that is required by the block, but will not be modified by it. This static data will be stored in a data block using a UDT data structure, the INPUT parameter to which this data is linked, will use the same UDT as its data type.

Note: *Other data may also be passed in this way, specifically, this will be information that will not be modified by the block, system information for example.*

(8) Dynamic data will be passed to the block via an INOUT parameter (i.e. read/write data), this is data that is required by the block, and that will be modified by it. This dynamic data will be stored in a data block using a UDT data structure, the INOUT parameter to which this data is linked, will use the same UDT as its data type.

(9) Static and dynamic data will always be stored in separate data blocks, designated as static and dynamic and these will have their own numbering ranges:

DB NUMBER RANGE	TYPE OF DATA
00000-19999	Static data
20000-39999	Dynamic data

Table 4.7 PAL static and dynamic data block numbering ranges

(11) The purpose of this separation of static and dynamic data is that the static data is constant and can be verified against a known "offline" version of the software to establish that the data is correct, the dynamic data is "live data" and is constantly changing and such verification would be meaningless.

(12) By separating static data from the dynamic data, it provides and additional means of verifying the software installed in a Controller is the correct version of the software.

(13) Where a standard module has a static data assignment or a dynamic data assignment or both (this is most cases), then UDTs will be defined to hold the static data and the dynamic data. The static UDT will be given the same number as the standard block with which it is associated, the dynamic data will have the same number plus 20000.

(14) For example, if FC10001 is used, the static UDT will have number 10001 and the dynamic UDT will have number 30001.

(15) Similarly, the data blocks that hold the static and dynamic data will have the same numbers as the UDT.

(16) Extending the previous example, FC10001 would have static UDT10001 and Dynamic UDT30001, these would be stored in DB10001 (static data) and DB30001 (dynamic data).

4.3.3 Instance data blocks

(1) Where a function block (FB) is used, this will have an associated instance data block (iDB), this is a requirement of the Simatic Controller software itself.

(2) Generally, only third-party software will use FBs, all standard and application modules will be stored in functions (FCs) that do not require instance data blocks.

(3) The instance data block assigned to a particular function block will be in the numbering range:

DB NUMBER RANGE	TYPE OF DATA
40000-59999	Instance data blocks

Table 4.8 PAL instance data block numbering range

(5) The actual number can be freely allocated within this range; i.e. the instance DB number does not have to match the FB number, the numbering should however reflect logical grouping of the instance DBs.

4.4 Identification of modules and their type

(1) There will be five types of software modules included with the PAL:

①	Standard modules	Library modules that carry out a particular function, for example reading and scaling an instrument connected to the Controller.
②	Application modules	Project specific modules that coordinate the use of the standard modules and apply appropriate logic and signal conditioning relevant to the project in question
③	Template modules	Example modules that show how application modules should be constructed and how standard modules should be used
④	Document modules	Modules containing information explaining how to document project specific modules and examples of such documentation
⑤	Interrupt modules	These are specifically the organisation blocks used to process different types of interrupt operations and fault detection

(2) Within the PAL these individual types of modules are assigned to functions (FCs). The interrupt modules are exclusively assigned to organisation blocks (OBs).

(3) The following types of data structures and data blocks are supported by the PAL:

①	Static user data type	Data structures specific to each standard module that hold fixed, unchanging, configuration data for the module
②	Dynamic user data type	Data structures specific to each standard module that hold live, variable, operational data for the module
③	Static data block	A data block that holds the multiple instances of the static UDT associated with the standard module (one instance per call of the module)
④	Dynamic data block	A data block that holds the multiple instances of the dynamic UDT associated with the standard module (one instance per call of the module)
⑤	Instance data block	A data block that holds function block data for a standard module that is allocated to a function block (FB) rather than a function (FC), there is one instance data block allocated to each instance in which the FB is used

(4) To ensure that the PAL software is compatible with the Siemens Simatic Controller internal structures, the blocks are allocated numbers ranges within the permissible range of block numbers given in § 4.1.4.

(5) The type of module is identified by block number allocated to it. This is summarised in the following table:

BLOCK TYPE	NUMBER RANGE	CLASS	DESCRIPTION
OB	00001-00122	Int	Interrupt handling modules
FC/FB	00001-19999	Std	Standard modules
FC/FB	20001-39999	App	Application modules
FC	40000-60999	Tmt	Template modules
FC	61000-65535	Doc	Document modules
UDT	00001-19999	St_	Static data structure
UDT	20001-39999	Dy_	Dynamic data structure
DB	00001-19999	St_	Static storage data block
DB	20001-39999	Dy_	Dynamic storage data block
iDB	40000-59999	iDB	Instance data blocks (associated with FBs)

Table 4.9 Full range and type of module numbering for the PAL

(6) Each of these number ranges is broken down further in relations to the subdivisions within the PAL software structure (see § #5.15.1).

4.5 Software Control Mechanism

(1) The Validation Plan *[Ref. 002]* Appendix A requires that a robust mechanism be put in place to manage the revision control for the software modules developed as part of this Project. This mechanism is encapsulated in a separate Software Control Mechanism (SCM) document *[Ref. 019].*

(2) There are two principal requirements for the PAL Software Control Mechanism:

① Establish a mechanism for numbering and storing the various software module versions throughout the development, test and qualification phases of the Project

② Establish a mechanism for the storage and tracking of software module revisions within a formal Version Control System (VCS)

Expanding on these subjects:

4.5.1 Module revision numbering mechanism

(1) The Validation Plan (VP) *[Ref. 002]*, established that software version control was a necessary requirement for the project and that all software modules within the Project must have individual revision and status information that covers all phases of the software development:

- Software development (system build)
- Testing (at both a modular and integrated level)
- Qualification
- Release for use

(2) The revision system must also be applicable to the TIA Projects as a whole (rather than just the individual modules within the projects); to clarify, the software modules do not exist within their own right, each software module is stored in TIA Portal project that expands as each new software module is developed.

4.5.2 A version control system

(1) A version control system (VCS) is a mechanism for recording changes made to any files within a software project. It records all the changes, what files were affected by each change and a reason explaining why those changes were made. It also records who made the change and the time and date of the change.

(2) The VCS keeps a record of every change made within the project and allows any file that has been modified to be reverted back to a previous state. It means that if a software module is changed, the original module can always be recovered by the VCS.

(3) Version control systems generally have other facilities too, they are able to show the differences between two different versions of the software (even down to lines within a file), they allow multiple people to work on the project at the same time—even to work on the same file at the same time, and they provide mechanisms for resolving conflicts (where two different people have modified the same section of a file).

(4) Version control systems can be applied to any kind of project; it can be a website, a documentation project, a software application, engineering control system—anything at all, as long as it's a collection of files that can be stored on computer.

(5) The version control system does not itself edit or modify any of the files within the project; it simply records the changes and, where it recognises a file type, is able to display those changes that have occurred to it.

(6) The version control system does not care what software application is used to modify files within the project, it can be anything: text editor, word processor, file manager, graphics editor, specialist programming application &c. It cares only, that a file under its control has been modified and why the modification was made.

(7) **Version control systems simply record any change made within a collection of files (the project), who made it, when it was made and the reason why. That is all.**

(8) With the advent of TIA Portal V16, Siemens introduced the concept of *Workspaces*, these are environments (essentially, just Windows folders) into which the

programmable aspects of a TIA Project (blocks, data types and tags) can be exported (or imported) as XML[8]files.

(9) This is a new concept, previous versions of TIA Portal did not offer the facility of exporting software modules in a widely accessible (text based) format, the software could only be read by the proprietary TIA Portal package itself.

(10) The benefit of this new Workspace facility is that the exported files are stored as XML files, and XML files are an ideal format for version control systems (VCSs), version control systems can read every aspect of an XML file and identify any changes that have been made, and, just as importantly, keep track of all these changes. Additionally, each block, data type and tag table is exported as its own XML file and as such allows the tracking of each individual element within the software library. It would for example, be possible to identify all the changes made to a particular Function (e.g. FC01001) and determine at which point in the revision history each change was made.

(11) This was the purpose of Siemens adding this Workspace facility to TIA Portal, it allows proper version control of the software being developed in a TIA Portal project. It also does not require a proprietary Siemens VCS, any and all VCS systems can track text-based files (it is fundamentally, what they were designed for).

(12) To make things easier, Siemens also allow third-party *"add-ins"* to be created that can interface with these new Workspaces. One such add-in (created by Siemens) provides an interface to the version control system Git and its online partner GitHub.

(13) The Git add-in allows TIA Portal to interface with a Git controlled Workspace, Git also supports various graphical user interfaces, in particular, Git can be controlled and managed from within the Visual Studio Code (VSC) text editor, VSC is widely used within the Practical Series of Publications and will be the preferred solution for providing a VCS interface for the PAL software.

---

8 XML or eXtensible Mark-up Language files are text files that are both machine and human readable; very similar to HTML (HyperText mark-up Language) and widely used to store documents in a manageable and readable format; it contains both content and structure.

5 The PAL software structure

(1) All non-documentation[9] software modules within the PAL (be they standard modules, application modules, or template modules) are grouped into subcategories or *functional groups* that identify more closely the purpose of each module.

(2) These functional groups also determine the execution order of the PAL software. The PAL has a predetermined order of programme execution; this is shown in Figure 5.1. This shows the complete PAL programme structure.

(3) The structure of Figure 5.1 is the complete structure of the PAL software and is applicable to any software developed using the PAL. Not all Controller programmes will require all these groups (it depends on the application in question). However, where a group is used, it must follow the execution order shown in Figure 5.1.

(4) For example, if a programme did not require INTERLOCKS AND PROTECTION or SAFETY SYSTEMS, but had READ INSTRUMENTS and CONTINUOUS LOGIC, then the CONTINUOUS LOGIC would follow the READ INSTRUMENTS (the INTERLOCKS and SAFETY would not be present); CONTINUOUS LOGIC must not precede READ INSTRUMENTS in the order of execution.

(5) Each of the steps in Figure 5.1 (referred to as *functional groups*) usually has both an application block and at least one standard module associated with it; (there are some steps, COMMAND EXECUTION being one, that do not have any associated standard modules).

---

9 Documentation modules contain examples of how the Controller software is commented, and are applicable to all modules irrespective of the function of the module.

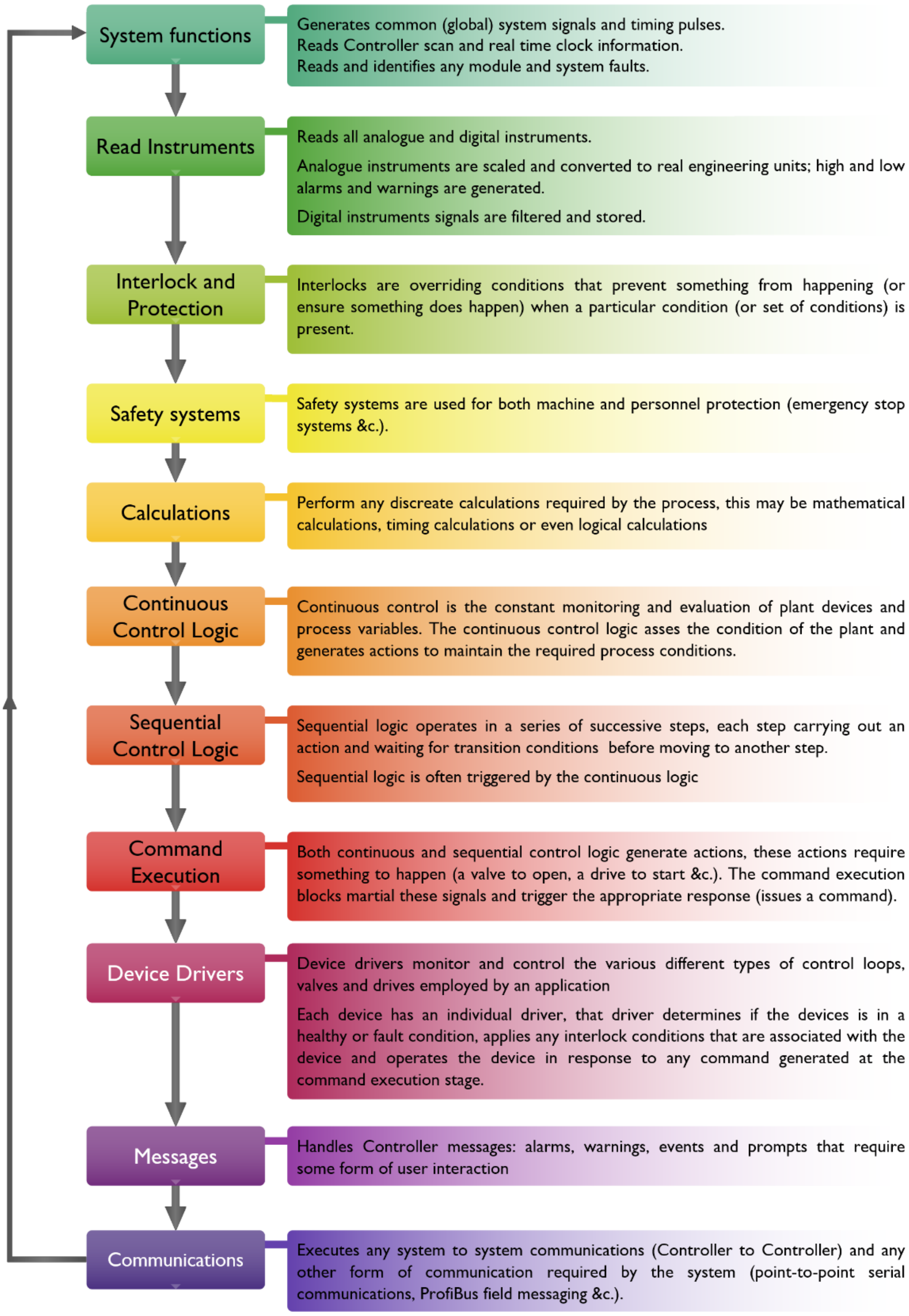


Figure 5.1 Programme structure

5.1 Functional group module numbering

(1) The PAL functional groups are allocated numbers within the block types of Table 4.9:

RANGE	CLASS	FUNCTION	RANGE	CLASS	FUNCTION	RANGE	CLASS	FUNCTION
00ppp	Std	Reserved (third party blocks)	20nnn	App	Start of cycle debug (SoC)	40nnn	Tmt	RESERVERD
01ppp	Std	System functions	21nnn	App	System call handling	41nnn	Tmt	RESERVERD
02ppp	Std	Instrument read modules	22nnn	App	Instrument handling	42nnn	Tmt	Instrument typical
03ppp	Std	Interlock modules	23nnn	App	Interlock handling	43nnn	Tmt	Interlock typical
04ppp	Std	Safety modules	24nnn	App	Safety system handling	44nnn	Tmt	Safety typical
05ppp	Std	Calculations & mathematics	25nnn	App	Calculations handling	45nnn	Tmt	Calculations typical
06ppp	Std	Continuous control modules	26nnn	App	Continuous control logic	46nnn	Tmt	Continuous logic typ.
07ppp	Std	Sequential control modules	27nnn	App	Sequences	47nnn	Tmt	Sequence typical
08ppp	Std	RESERVED	28nnn	App	Command handling	48nnn	Tmt	Commands typical
09ppp	Std	RESERVED	29nnn	App	RESERVED	49nnn	Tmt	RESERVED
10ppp	Std	Device driver (control loops)	30nnn	App	Control loop handling	50nnn	Tmt	Control loop typical
11ppp	Std	Device driver (valves)	31nnn	App	Valve module handling	51nnn	Tmt	Valves typical
12ppp	Std	Device driver (drives)	32nnn	App	Drive module handling	52nnn	Tmt	Drives typical
13ppp	Std	Device driver (RESERVED)	33nnn	App	RESERVED	53nnn	Tmt	RESERVED
14ppp	Std	Device driver (RESERVED)	34nnn	App	RESERVED	54nnn	Tmt	RESERVED
15ppp	Std	Device driver (RESERVED)	35nnn	App	RESERVED	55nnn	Tmt	RESERVED
16ppp	Std	Alarm/warning modules	36nnn	App	Alarm handling	56nnn	Tmt	Alarms typical
17ppp	Std	Communication modules	37nnn	App	Communication handling	57nnn	Tmt	Comms typical
18ppp	Std	Subroutines	38nnn	App	RESERVED	58nnn	Tmt	RESERVED
19ppp	Std	Debug subroutines	39nnn	App	End of cycle debug (EoC)	59nnn	Tmt	RESERVED

Table 5.1 Functional group number ranges

Where:	Range	Specifies the functional group number range in the format GGnnn or GGppp where GG is a two-digit number that represents the function group nnn indicates any number in the range 0 to 999; thus, 37nnn is any number in the range 37000-37999 ppp indicates any number in the range 1 to 999; thus, 02ppp is any number in the range 02001-02999
	Class	Specifies the type of module, **Std** is a standard module, **App** is an application module and **Tmt** is a template module

5.1.1 Functional group summary

FUNCTION GROUP	STANDARD MODULE NUMBER	APPLICATION MODULE NUMBER	TEMPLATE MODULE NUBER
Debug (start of cycle)	N/A	FC 20nnn	FC 40nnn
System functions	FC 01ppp	FC 21nnn	FC 41nnn
Read instruments	FC 02ppp	FC 22nnn	FC 42nnn
Interlock & protection	FC 03ppp	FC 23nnn	FC 43nnn
Safety systems	FC 04ppp	FC 24nnn	FC 44nnn
Calculations & mathematics	FC 05ppp	FC 25nnn	FC 45nnn
Continuous control	N/A	FC 26nnn	FC 46nnn
Sequential control	FC 07ppp	FC 27nnn	FC 47nnn
Command handling	N/A	FC 28nnn	FC 48nnn
Reserved	N/A	N/A	N/A
Device drivers	FC 10ppp-15ppp	FC 30nnn-35nnn	FC 50nnn-55nnn
Message handling	FC 16ppp	FC 36nnn	FC 56nnn
Communication handling	FC 17ppp	FC 37nnn	FC 57nnn
(subroutines)	FFC 18ppp	N/A	N/A
Debug (end of cycle)	FC 19ppp	FC 39nnn	FC 59nnn

Table 5.2 Functional group summary

5.2 Module naming conventions

(1) All software modules within the PAL are assigned to specific programming blocks (OBs, FCs) and the data for these modules is stored using predefined user data types (UDTs) and stored in data blocks (DBs). All such blocks and UDTs are identified by a specific number (see Table 4.9 and Table 5.1).

(2) In addition to the number, all blocks and UDTs are also given a name. The combination of number and name form a unique symbolic address for the block

(3) The block name has the following structure:

`ClassFunctionDescription`

`Class`, `Function` and `Description` are explained below

5.2.1 Block class

(1) The `Class` is a three-letter abbreviation that specifies the category that the block belongs to. The abbreviation is in lower case with a leading capital letter:

Abb.	Class	Meaning
`Std`	Standard	Standard block —These are blocks that carry out a particular function; for example, a valve device driver block.
`App`	Application	Application block — These are project specific blocks, written for a particular project and configured to match the requirements of that project.
`Int`	Interrupt	Interrupt block — Executed when specific interrupt conditions are detected, this includes the main program execution interrupt (OB 1)
`Tmt`	Template	Template block — This is an example block that explains how functions should be configured and executed
`Doc`	Documentation	Documentation block — Contains documentation examples for different components of a project
`Dy_`	Dynamic	DB/UDT only (contains live, dynamic, data)
`St_`	Static	DB/UDT only (contains fixed, static data)
`Rc_`	Recipe (semi-static)	DB only (data is loaded from a recipe)

Table 5.3PAL block naming — class

5.2.2 Block function

(1) The `Function` is a five letter (max) abbreviation that identifies the functional area within the programme structure (Figure 5.1) that the block is associated with. The abbreviation is in lower case with a leading capital letter:

ABB.	FUNCTION	MEANING
`Sys`	System	**System blocks** Common system functions: common (global) signals, diagnostic functions, system timing, clock synchronisation, &c.
`Inst`	Instrumentation	**Instrument block** Analogue and digital instrument functions (read, scale, filter, threshold detection, &c.)
`ILock`	Interlocks	**Interlock, permissive and trip logic** Identifies and maps the various interlock conditions
`Safe`	Safety	**Safety systems** Handles emergency stop and safety rated devices. Manages redundant and high availability systems
`Calc`	Calculations	**Calculation and mathematics** Calculation, mathematical functions and algorithms (generally of a complex nature i.e. not simple arithmetic)
`Cont`	Continuous	**Continuous control logic** Constant monitoring, evaluation and operation of plant devices and process variables.
`Seq`	Sequences	**Sequential control logic** Sequential (step-transition) based operations
`Dev`	Device drivers	**Device drivers** Monitor and control individual devices connected to the controller (valves, drives, PID loops &c.)
`Msg`	Messages	**Alarm, warning, event and prompt handling** Marshals the various alarms, organising them for SCADA and HMI applications
`Comms`	Communications	**Communication handling** Executes system to system communications (Controller to Controller, point-to-point, ProfiBus FMS &c.)

ABB.	FUNCTION	MEANING
`Sub`	Subroutines	**Subroutine functions** Various subroutines (called by other blocks) to execute particular functions (subroutines are organised into similar function areas)
`INrm`	Normal Interrupts	**Normal (non-error) interrupt functions** Usually associated with specific OBs, interrupts generated by standard system events (time of day, cyclic, hardware &c.)
`IErr`	Error Interrupts	**Error interrupt functions** Usually associated with specific OBs, interrupts generated in response to a system fault (IO failure, card fault &c.)
`Debug`	Debug	**Debug functions** Generally, start of cycle and end of cycle debug operations and process simulation
`Gen`	General	**General scope** Applies to the whole project (such as explanatory information and instructions)

Table 5.4 PAL block naming — function

5.2.3 Block description

(1) The block description does not have a prescribed list of naming options; it is simply a short form description of what the block does. Examples are:

Abb.	Meaning
`AnalogRead`	Analogue read
`ScaleAI`	Scale analogue input
`ValveMod`	Modulating valve
`DriveVSD`	Variable speed drive

Table 5.5 PAL block naming — description

(2) Block descriptions are always written without spaces using *camel case*[10].

10 Camel case is the practice of joining words together and capitalising the start of each word, it is more formal known as *medial capitals*).

5.2.4 Block naming restrictions

(1) The basic restrictions on naming blocks within the PAL are:

① The `Class` abbreviation is three characters long and starts with a capital letter

② The `Function` abbreviation is no more than five characters long and must start with a capital letter

③ The `Description` does not have a restriction on the number of characters but should generally be kept short

④ Each separate word in the description is capitalised with all other letters in lowercase (this includes the first word)

⑤ The overall length of the name (including `class`, `function` and `description`) must be 20 characters or less

⑥ Only the characters [a-z], [A-Z], the numbers [0-9], the dash/hyphen [-] and the underscore [_] are permitted

5.3 Module symbolic names

(1) Within the PAL, symbolic names are simply the block number (e.g. FC11001) followed by an underscore character (_) and then the block name (see § 5.2). For example, if FC11001 had the block name `StdDevValveIsol`, the full symbolic name for FC11001 would be:

`FC11001_StdDevValveIsol`

(2) The two letters at the start (FC) in the above example are the standard abbreviations for the blocks within the PAL as follows:

Abb.	Meaning
FB	Function block
FC	Function
OB	Organisation block
DB	Data block
ID	Instance data block
UT	User data type

Table 5.6 PAL block naming — block type prefixes

(3) Data blocks and user data types have the exactly the same name as the function or function block with which they are associated. Only the class changes; this will be either `St_` (static) if the block holds configuration and constant values or `Dy_` (dynamic) if it holds live (changing) data.

(4) A third option `Rc_` is also possible if recipes are being used, see § #6.3.26.3.2.

(5) Thus, extending the previous example, a full set of blocks and data types for the isolating valve within a project would be:

ADDRESS	FULL SYMBOLIC NAME	DESCRIPTION
FC11001	FC11001_StdDevValveIsol	Isolating valve device driver block
FC31001	FC31001_AppDevValveIsol	Isolating valve application block
DB11001	DB11001_St_DevValveIsol	Static data block for isolating valves
DB31001	DB31001_Dy_DevValveIsol	Dynamic data block for isolating valves
UT11001	UT11001_St_DevValveIsol	Static data type structure for isolating valves
UT31001	UT31001_Dy_DevValveIsol	Dynamic data type structure for isolating valves

Table 5.7 Block numbering, naming and symbols (an example)

5.4 The PAL structure within a Controller

5.4.1 Application modules

(1) The PAL structure with in a Controller is primarily determined by the use of application modules called from within OB 1.The complete OB 1 PAL structure is shown in Figure 5.2. This shows application block calls to the thirteen functional groups (this includes the 11 functional groups listed in Figure 5.1, plus two *debug* groups: a start of cycle debug and end of cycle debug — debug functional groups are discussed in § #8.138.13).

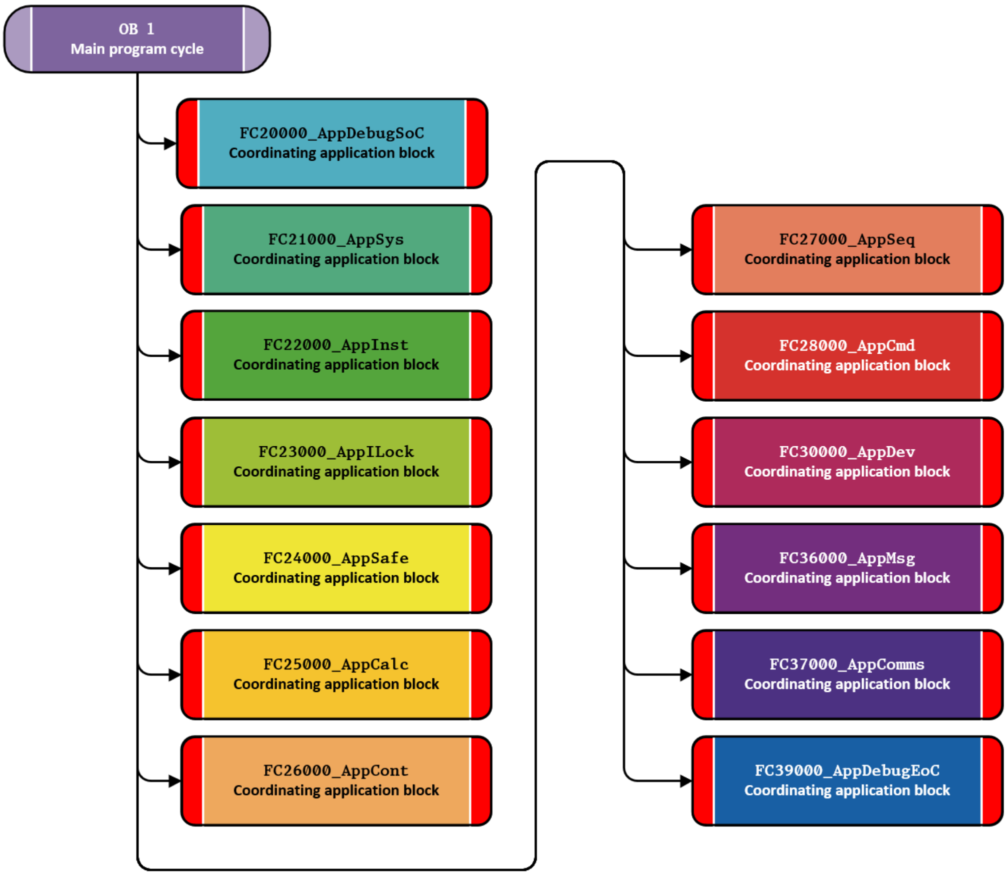


Figure 5.2 Complete OB 1 PAL structure

(2) All of these functional groups with the exception of the system functions (`FC21000_AppSys`) are optional (the requirements for these applications depends

entirely on the purpose of the Controller); most Controllers will have a subset of these functional groups.

(3) Application modules are specific to the software project in question and are programmed specifically for that project, they are not fixed modules like the standard modules.

(4) There are three categories of application modules:

①	Coordinating	*Coordinating* application blocks exist for each function group and are used to organise all the block calls within that particular function group.
②	Marshalling	*Marshalling* modules subdivide the coordinating application modules into logical groupings within the functional group.
③	Programmed	*Programmed* modules contain extensive programming statements, rather than the configuration exercises used with coordinating and marshalling modules.

(5) These concepts are explained in section 7.

5.4.2 Standard modules within the PAL structure

(1) Standard modules are the library modules issued with the PAL software.

(2) There are standard modules associated with most of the functional groups listed in Table 5.2. These are summarised below:

FUNCTION GROUP	STANDARD MODULE NUMBER	QUALIFICATIONS
Debug (start of cycle)	N/A	Application level software only
System functions	FC 01ppp	
Read instruments	FC 02ppp	
Interlock & protection	FC 03ppp	
Safety systems	FC 04ppp	
Calculations & mathematics	FC 05ppp	
Continuous control	N/A	Application level software only
Sequential control	FC 07ppp	
Command handling	N/A	Application level software only
Reserved	FC 09ppp	Reserved for future expansion
Device drivers (Control loops)	FC 10ppp	
Device drivers (Valves)	FC 11ppp	
Device drivers (Drives)	FC 12ppp	
Device drivers (Reserved)	FC 13ppp	Reserved for future expansion
Device drivers (Reserved)	FC 14ppp	Reserved for future expansion
Device drivers (Reserved)	FC 15ppp	Reserved for future expansion
Message handling	FC 16ppp	
Communication handling	FC 17ppp	
(subroutines)	FC 18ppp	Standard subroutine functions
Debug (end of cycle)	FC 19ppp	Contains debug subroutines

Table 5.8 Standard module groups

`ppp` indicates any number in the range 1 to 999; thus, 02ppp is any number in the range 02001-02999

(3) The last three digits of a standard module number (e.g. `FC GGppp`) are never `000`; standard module numbering starts at `GG001` and can range up to `GG999` where `GG` represents the functional group to which the standard module belongs (this itself will be in the range 01 to 19).

(4) Those groups that do not have standard modules associated with them: debug (start of cycle), continuous control and command handling, do so because these groups are entirely dependent on the purpose of the project software in question and are addressed wholly with the use of programmed application modules (see § #7.37.3).

(5) The PAL software contains a large number of standard modules (see Section 8 for a full list). Standard modules are programmed using functions (FCs) — the PAL does not use FBs (the mechanisms for data storage using UDTs see Section 6, makes the use of FBs largely unnecessary).

(6) All standard modules are parameterised and, generally, have the following appearance:

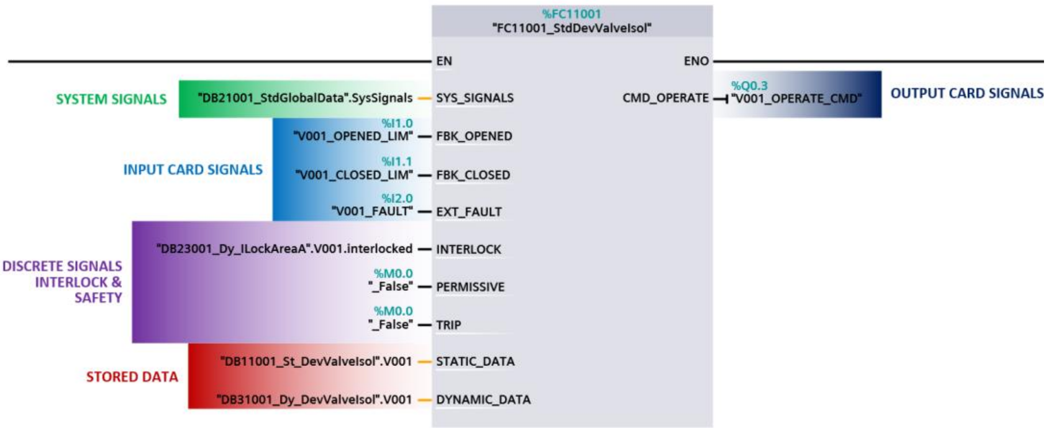


Figure 5.3 Typical arrangement for a standard module

(7) The block in Figure 5.3, shows a typical arrangement for the calling of a standard module, in this case the isolating valve, device driver, standard module. This module was chosen as an example because it has a full set of the parameters types typically associated with a standard module.

(8) All standard modules have parameters that conform with those shown in Figure 5.3:

PARAMETER CATEGORY	TYPE	OPTIONAL	DESCRIPTION
System signals	In	No	Passes the full set of Controller logic and timing signals to the module — needed for the internal operation of the standard module (see § 5.4.4)
Input card signals	In	Yes	All input signals (such as valve limit switches) needed by the block are passed as discrete parameters into the block
Output card signals	Out	Yes	All output signals (such as valve energise outputs) generated by the block are passed as discrete parameters from the block
Discrete signals	In	Yes	The discrete signals are direct digital signals (usually interlock and safety signals) generated elsewhere within the software, but that have a direct impact on the operation of the standard module in question
Stored data: STATIC_DATA	In	Yes	Most standard modules are configurable in some way. The stored data contained in the STATIC_DATA parameter determines this configuration. STATIC_DATA is not modified by the module. STATIC_DATA is always stored in a data block in the form of a UDT
Stored data: DYNAMIC_DATA	InOut	Yes	Most standard modules require a read/write data area that stores operational information (elapsed time, status information &c.), all this information is passed to the block in the DYNAMIC_DATA parameter. DYNAMIC_DATA is always stored in a data block in the form of a UDT

Table 5.9 Parameter categories for standard modules

(9) Standard modules are true library modules and conform to the standards required of such modules, in terms of the Siemens Simatic programming standards this is:

- Library modules must not use global data access (of memory bits, IO signals, timers, counters &c.)
- Library modules must not directly access data blocks or instance data blocks

(10) It is for this reason that the common system logic and timing signals (see § 5.4.4) are passed parametrically to the block in the SYS_SIGNALS parameter; all standard modules have this parameter and it is always the first parameter of the block.

(11) The SYS_SIGNALS parameter is always an In parameter[11] (read only); the standard modules require the signals in the SYS_SIGNALS parameter, but may not modify the signals within it.

(12) Standard modules are used repeatedly within the PAL software, for example if the Controller software had two isolating valves, then the isolating valve standard module of Figure 5.3 (*FC11001_StdDevValveIsol*) would be called twice, from a marshalling or coordinating application module (in this case a marshalling application module) once for each valve, each such call of the module is referred to as an *instance* of the block. In diametric form, it would have the following structure:

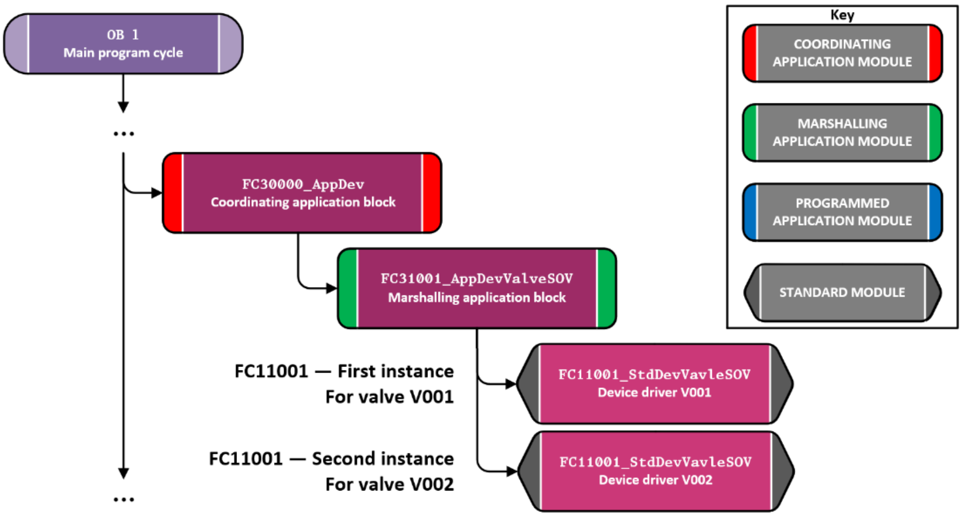


Figure 5.4 Multiple instances of a standard module

11 The exception being the standard module *FC01001_StdSysGlobalData*, this is the standard module that generates the logic and timing signal and is the only compulsory standard module that must be present in the Controller software (see § 8.1); here, the SYS_SIGNALS parameter is an InOut type.

(13) In parametric terms, the two calls to `FC11001_StdDevValveIsol` would be:

Figure 5.5 Parametric difference for multiple instances of a standard module

(14) In the first instance (Network 3), all references are to V001, in the second instance (Network 4) all references are to V002. The differences are highlighted in red.

(15) This is the mechanism by which all standard modules work. The blocks can be called multiple times, each time the block is called, it receives different parameters that are applicable to that *instance* of the call and no other; in Figure 5.5, the first call (Network 3) passes all the V001 data to the block and that *instance* of the block is entirely association with V001. In the second *instance* (Network 4), all the parameters are for V002 and that *instance* of the block is entirely association with V002.

5.4.3 Interrupt modules within the PAL structure

(1) The only interrupt module that is required by the PAL is OB1, this is the block that is automatically executed by the Controller operating system at the start of each cycle.

(2) OB1 is the master programming block within the PAL software and is used to call the subsequent marshalling application modules, this can be seen in the programming examples shown in Figure 7.1 to Figure 7.4.

(3) OB1 however, is not the only interrupt module, there are various organisation blocks, each one supporting a different type of interrupt. The most commonly used is a cyclic timed interrupt, this interrupts the main Controller cycle at regular intervals (ranging from 100µs to 60 s). The following is a full list of standard interrupt organisation blocks available within a Controller:

OB NUMBER	PAL MODULE NAME	DESCRIPTION
OB1	OB00001_IntlNrmMainProgram	Controller main program cycle Called at the start of each Controller cycle
OB10	OB00010_IntlNrmTimeOfDay	Time of day Interrupt Called by time and day of week
OB20	OB00020_IntlNrmTimeDelay	Time delay Interrupt Called after a specified delay has expired
OB30	OB00030_IntlNrmCyclic	Timed cyclic Interrupt Called at specified intervals
OB40	OB00040_IntlNrmHardware	Hardware Interrupt Called when a specified signal is detected
OB100	OB00100_IntlNrmStartUp	Start-up Interrupt Called when the CPU transitions to RUN

Table 5.10 Standard interrupt modules and organisation blocks

(4) Interrupt modules are also used to detect certain fault conditions:

OB NUMBER	PAL MODULE NAME	DESCRIPTION
OB80	OB00080_IntlErrCycleTimeErr	Error Interrupt Maximum cycle time exceeded
OB82	OB00082_IntlErrModuleDiag	Error Interrupt Module diagnostics signal received (module fault)
OB83	OB00083_IntlErrModuleChange	Error Interrupt Module changed, removed or installed
OB86	OB00086_IntlErrRackErr	Error Interrupt Rack failure or fault
OB40	OB00121_IntlErrProgramErr	Error Interrupt Programming fault or error
OB100	OB00122_IntlErrIOErr	Error Interrupt IO card access fault

Table 5.11 Fault interrupt modules and organisation blocks

5.4.4 Third-party modules

(1) Third-party equipment manufacturers often provide their own software to interface with their equipment, this is usually in the form of functions (FCs) and function blocks (FBs) that can be installed within the project software.

(2) The PAL accepts that this is the case and such third-party modules can be installed and used within the PAL. Such modules should be re-numbered to fall in the range 1-999.

(3) It is also the case, that the project in question may have some equipment that is not covered by the standard modules in the PAL. Where this is true, a new project specific module may be required, these can be added to the PAL, preferably in the third-party module area (1-999 numbering range); or alternatively, for devices, in one of the reserved areas (13000-15999). This latter option should only be used if it is logical to do so, this would be where there are a substantial number of modules required or where such modules make a practical contribution to the PAL and may at some future point be incorporated into it.

(4) Any project specific standard modules are new modules and not by default part of the PAL (they have been written for the particular project in question). As such, those modules must be thoroughly tested to the level required by the particular project.

5.5 Common signals within the PAL

(1) There are several common logic and timing signals that are needed by all the PAL software modules; these are referred to collectively as *system global data signals* (or in short form as just *system signals*). These are the signals passed to the standard modules with the SYS_SIGNALS parameter discussed in the previous section.

(2) These system signals are generated by the only compulsory standard module required within the PAL: *FC01001_StdSysGlobalData*. This standard module is called at the start of OB 1 (highlighted below):

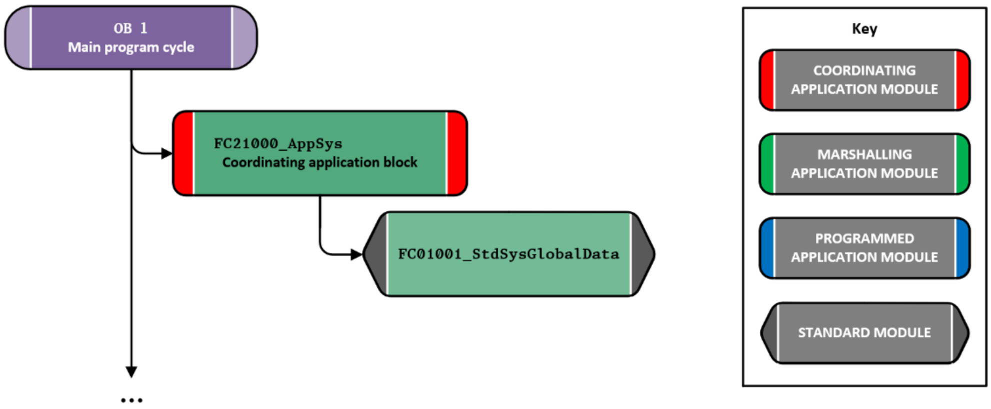


Figure 5.6 Standard module for system signals

(3) The system signals standard module is called from the associated coordinating application module (*FC21000_AppSys*); the only block that may precede this is the start of cycle debug block (see § 8.13), this is a temporary block used during the testing phase of software production and it will not be present in any final software developed using the PAL.

(4) The PAL system signals are stored in two formats, first as a UDT data structure (*UT21001_Dy_SysSignals*) in the system global data block (*DB21001_Dy_SysGlobalData*) in the variable **SysSignals**. This form of the system signals is designed to be passed as a parameter to all the standard modules using the SYS_SIGNALS parameter.

(5) Secondly, the same data is stored in bit memories, these can be accessed globally in all application (project specific) blocks. The bit memories used to store the system signals

are `MB0` and `MB1`; the individual signals within the bytes being given symbolic tags in the tag table `PAL_SystemTags`.

(6) Both forms of the data are discussed further in the following sections:

Note: *There is absolutely no difference between the two form of the signals, it is simply a question of which to use under what circumstances: parametric for standard modules, direct for application modules*

5.5.1 System signals: parametric access and direct access

(1) The guidelines for library modules state that standard blocks must not use global data access to gather information from within the Controller *(i.e. must not directly access data)*, to do so, means that the block cannot be a true library module that can be used on any system, it requires that system to have an underlying set of variables that existed outside the block.

(2) Consequently, all standard (library) blocks have to receive all the data they require to operate, via parameters passed to the block *(parametric access)*. Hence the use here of UDT data that can be passed as a single parameter into every standard block.

(3) The application blocks are by their nature, specific to the project being developed, they are not library modules. As such the application modules *can* use direct access to read the system signals. Hence the two versions:

- Parametric access — UDT parameter for standard modules
- Direct access — memory bits for application modules

5.5.2 UDT system signals for parametric access

(1) The system signals for parametric access are stored in *DB21001_Dy_SysGlobalData* in the variable **SysSignals**; this variable is a UDT of type *UT21001_Dy_SysSignals*, it contains the 16 logic and timing signals of Table 5.12:

Data structure	*UT21000_Dy_SysSignal*	
Signal	Type	Function
_False	Bool	System Logic Bit — Always FALSE
_True	Bool	System Logic Bit — Always TRUE
_50ms	Bool	System Timing — 50 ms Pulse Scan synchronised
_100ms	Bool	System Timing — 100 ms Pulse Scan synchronised
_200ms	Bool	System Timing — 200 ms Pulse Scan synchronised
_500ms	Bool	System Timing — 500 ms Pulse Scan synchronised
_1s	Bool	System Timing — 1 s Pulse Scan synchronised
_2s	Bool	System Timing — 2 s Pulse Scan synchronised
_CycleTick	Bool	System Timing — Cycle tick (active odd cycles, alternates with _CycleTock)
_CycleTock	Bool	System Timing — Cycle tock (active even cycles, alternates with _CycleTick)
_CycleFirst	Bool	System Timing — First cycle detected
_100msSqW	Bool	System Timing — 100 ms square wave Scan synchronised
_200msSqW	Bool	System Timing — 200 ms square wave Scan synchronised
_500msSqW	Bool	System Timing — 500 ms square wave Scan synchronised
_1sSqW	Bool	System Timing — 1 s square wave Scan synchronised
_2sSqW	Bool	System Timing — 2 s square wave Scan synchronised

Table 5.12 Data structure: UT21001_Dy_SysSignals

(2) This data is passed as a parameter to all standard modules; the parameter is named SYS_SIGNALS on all standard modules, and is always the first IN parameter:

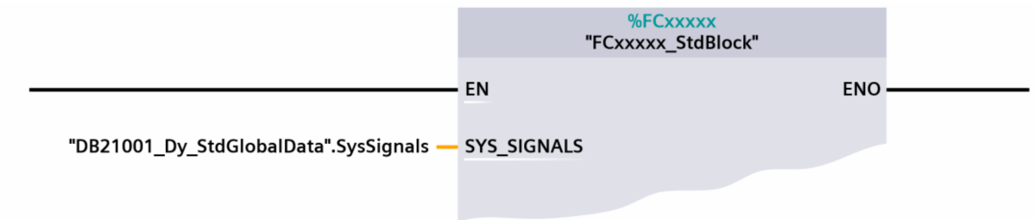


Figure 5.7 Example usage of the SYS_SIGNALS parameter

5.5.3 Bit memory direct access and the PAL system tag table

(1) The direct access version of the system signals are stored in two consecutive memory bytes: MB0 and MB1, see Table 5.13. These are given symbolic names *(tags)* that are then used throughout the remaining PAL application modules.

(2) The tags for these bytes (MB0 and MB1) are specified in the PLC TAGS entry in the project tree, and are stored in the tag table:

PAL_SystemTags

(3) This tag table is provided as standard as part of the PAL. The system signals within its contents are listed in Table 5.13

(4) The PAL_SystemTags tag table is a fixed tag table and is a fundamental part of the PAL. It must not be modified.

(5) The bit memories contained in the bytes PAL_SystemTags tag table are similarly reserved by the PAL and must these not be reallocated, renamed or used in any other tag table.

(6) All PAL system tags contained within the PAL_SystemTags tag table are identified by a leading underscore character (_).

(7) The memory bit system signals are given identical names to those in the *UT21001_Dy_SysSignals* data type (those used for parametric access, see § 5.5.2); as follows (Table 5.13):

NAME	TYPE	ADDRESS	DESCRIPTION
_SysSignals	Int	%MW0	System signals (logic and timing signals for direct access)
_SysSignals01	Byte	%MB0	System memory byte 01 — Logic and scan synchronised pulses
_False	Bool	%M0.0	System Logic Bit — Always FALSE
_True	Bool	%M0.1	System Logic Bit — Always TRUE
_50ms	Bool	%M0.2	System Timing — 50 ms Pulse Scan synchronised
_100ms	Bool	%M0.3	System Timing — 100 ms Pulse Scan synchronised
_200ms	Bool	%M0.4	System Timing — 200 ms Pulse Scan synchronised
_500ms	Bool	%M0.5	System Timing — 500 ms Pulse Scan synchronised
_1s	Bool	%M0.6	System Timing — 1 s Pulse Scan synchronised
_2s	Bool	%M0.7	System Timing — 2 s Pulse Scan synchronised
_SysSignals02	Byte	%MB1	System memory byte 02 — Scan signals and common square waves
_CycleTick	Bool	%M1.0	System Timing — Cycle tick (active odd cycles, alternates with _CycleTock)
_CycleTock	Bool	%M1.1	System Timing — Cycle tock (active even cycles, alternates with _CycleTick)
_CycleFirst	Bool	%M1.2	System Timing — First cycle detected
_100msSqW	Bool	%M1.3	System Timing — 100 ms square wave Scan synchronised
_200msSqW	Bool	%M1.4	System Timing — 200 ms square wave Scan synchronised
_500msSqW	Bool	%M1.5	System Timing — 500 ms square wave Scan synchronised
_1sSqW	Bool	%M1.6	System Timing — 1 s square wave Scan synchronised
_2sSqW	Bool	%M1.7	System Timing — 2 s square wave Scan synchronised

Table 5.13 PAL direct access system signals

5.5.4 System signal naming conventions

(1) The PAL direct access system signal tags and parametric access variables with in the *UT21001_Dy_SysSignals* data structure are named according to the following conventions:

① Each tag is prefixed with the underscore [_] character

② The remaining tag name is written in camel case

③ The name (including prefix) must be no more than 24 characters

④ It is permissible to separate parts of the name with an underscore [_] character (e.g. `_ClockMem_100msSqW`)

⑤ Units (such as milliseconds, ms) are not capitalised

⑥ The dash/hyphen [-] is not to be used (use the underscore instead)

⑦ Only use the characters [a-z], [A-Z], the numbers [0-9], and the underscore [_]

(2) All PAL system tags have a brief explanation of what the tag does stored in the comment field of the tag.

5.5.5 Global logic signals

(1) The system signals have two fixed logic signals, these are allocated as follows:

NAME	TYPE	DESCRIPTION
`_False`	`Bool`	System Logic Bit — Always FALSE
`_True`	`Bool`	System Logic Bit — Always TRUE

(2) The two signals `_False` and `_True` are logically testable signals, the `_False` signal being always set to `0` and the `_True` signal being always set to `1`.

5.5.6 Global timing signals

(1) There are two types of timing signals within the system signals: isochronous[12] pulses that are active for a single controller cycle (scan) and isochronous, even mark/space ratio square waves. All timing signals are derived from the CPU clock memory functions (see § 6.1).

Isochronous timing pulses

(2) The system signals include six individual timing pulses, these occur at intervals of 50 ms, 100 ms, 200 ms, 500 ms, 1 s and 2 s:

Name	Type	Description
`_50ms`	`Bool`	System Timing — 50 ms pulse
`_100ms`	`Bool`	System Timing — 100 ms pulse
`_200ms`	`Bool`	System Timing — 200 ms pulse
`_500ms`	`Bool`	System Timing — 500 ms pulse
`_1s`	`Bool`	System Timing — 1 s pulse (1000 ms)
`_2s`	`Bool`	System Timing — 2 s pulse (2000 ms)

(3) Each pulse is active for a single CPU cycle, and is activated at the start of the cycle following the termination of the specified time interval.

(4) These timing pulses form the basis for all timed actions within the PAL software. Timed events are measured by counting a number of occurrences of a timing pulse signal (for example the duration of an hour would be 3600 occurrences of the `_1s` pulse, a 10 s duration would be 100 pulses of the `_100ms` pulse).

(5) Timed events should generally use the shortest interval pulse compatible with the Controller cycle time and the duration of the event being measured.

12 Isochronous signals (sometimes scan synchronised signals) are signals that are synchronised with the Controller cycle, such signals only change state at the end of one scan and before the start of the next, presenting the same state to all the software modules in a given Controller cycle.

Isochronous timing square waves

(6) The system signals include five individual timing square wave signals, these have frequencies of 10 Hz (100 ms period), 5 Hz (200 ms period), 2 Hz (500 ms period), 1 Hz (1 s period) and 0.5 Hz (2 s period):

Name	Type	Description
`_100msSqW`	`Bool`	System Timing — 100 ms square wave (10 Hz)
`_200msSqW`	`Bool`	System Timing — 200 ms square wave (5 Hz)
`_500msSqW`	`Bool`	System Timing — 500 ms square wave (2 Hz)
`_1sSqW`	`Bool`	System Timing — 1 s square wave (1 Hz)
`_2sSqW`	`Bool`	System Timing — 2 s square wave (0.5 Hz)

(7) In a similar manner to the timing pulses, the rising and falling edges of the timing square wave occur at the start of a Controller cycle.

5.5.7 Cyclically dependent signals

(1) The system signals include three *cycle* dependent signals:

Name	Type	Description
`_CycleTick`	`Bool`	System Timing — Cycle tick (active odd cycles, alternates with `_CycleTock`)
`_CycleTock`	`Bool`	System Timing — Cycle tock (active even cycles, alternates with `_CycleTick`)
`_CycleFirst`	`Bool`	System Timing — First cycle detected

(2) The first two of these signals (`_CycleTick` and `_CycleTock`) are alternating signals that change state at the start of each cycle, the `_CycleTick` being active on every odd cycle since the CPU started (cycles 1, 3, 5, 7 …). `_CycleTock` activates on each even numbered cycle since the CPU started (cycles 2, 4, 6, 8 …).

(3) The `_CycleTick` and `_CycleTock` signals are often used as *"dead-man"* signals that show the CPU is running.

(4) The `_CycleFirst` signal is active on the first cycle of the CPU after a STOP → RUN transition. The `_CycleFirst` signal is an important signal and is generally used to set the Controller to a given *start-up* condition. It should be interpreted as telling the software that the processor has just started and all modules should be initialised and set to the correct start-up conditions.

BLANK PAGE

6 Data handling within the PAL

(1) There are three forms of data commonly used with the PAL:

- Memory bits (reserved for the common system signals, see § 5.4.4)
- IO signals read from, and written to IO cards
- Data block data in the form of UDTs and symbolic variables

Note: *The Simatic Controllers support additional data forms: timers and counters.*

The number of timers and counters available within Siemens Controllers is restricted, typically being 2048 of each. The PAL generally replaces the timers with edge triggered pulse counters of which there can be any number and they can be stored in data blocks. Counters are replaced with specific standard modules that again store the derived counts in data blocks and again any number of which are supported.

(2) All data within the PAL will be symbolically addressed (in the case of IO and memory bits, with the use of tag tables).

6.1 Data in the form of memory bits

(1) The PAL does not generally use the memory bits available to a Controller, instead storing data within the more flexible data blocks.

(2) There are two exceptions to this, the first is the use of the CPU clock memory (see § 4.2.4 for full details), this stores various CPU generated timing signals within a designated area of the memory bits (in this case `MB10`), these signals are required in the generation of the isochronous system timing signals (see § 5.5.6).

(3) Secondly, the direct access system signals are store in a two-byte area of the memory bits (`MB0` and `MB1`), these are listed in § 5.5.3.

(4) The remaining areas of the memory bits are unused (the S7-1500 has 131,072 such bits, arranged in 16,384 bytes — the S7-1200 has either 32,768 or 65,536 such bits depending on the CPU in question, again arranged in bytes).

(5) Where memory bits are used, they must be addressed symbolically, each bit, byte, word or double word must be given a unique symbol, referred to as a *tag*, these tags are stored in a specific tag table.

(6) The tags for the CPU clock memory (`MB10`) and the direct access system signals (`MB0` and `MB1`) are stored in the predefined tag table:

`PAL_SystemTags`

(7) This tag table is provided as standard as part of the PAL. A full list of its contents is provided in Table 5.13.

(8) The `PAL_SystemTags` tag table is a fixed tag table and is a fundamental part of the PAL. It must not be modified.

(9) The bit memories contained in the bytes `MB0`, `MB1` and `MB10` are similarly reserved by the PAL and must these not be reallocated, renamed or used in any other tag table.

(10) The PAL makes very limited use of the memory bit allocations within the Controller, essentially just for system signals; the further use of memory bits, *while not encouraged*, in not prohibited by the PAL. The user is free to allocate memory bits as required; however, the following restrictions apply:

- Memory bits cannot be passed to the PAL standard modules[13], these expect data to be passed in the form of UDTs
- Any additional memory bits must use a separate tag table, they must not be added to the predefined `PAL_SystemTags` tag table

13 It would only be possible to pass memory bits to standard modules as discrete signals (see § 5.5.1.

Name	Type	Address	Description
_SysSignals	Int	%MW0	System signals (logic and timing signals for direct access)
_SysSignals01	Byte	%MB0	System memory byte 01 — Logic and scan synchronised pulses
_False	Bool	%M0.0	System Logic Bit — Always FALSE
_True	Bool	%M0.1	System Logic Bit — Always TRUE
_50ms	Bool	%M0.2	System Timing — 50 ms Pulse Scan synchronised
_100ms	Bool	%M0.3	System Timing — 100 ms Pulse Scan synchronised
_200ms	Bool	%M0.4	System Timing — 200 ms Pulse Scan synchronised
_500ms	Bool	%M0.5	System Timing — 500 ms Pulse Scan synchronised
_1s	Bool	%M0.6	System Timing — 1 s Pulse Scan synchronised
_2s	Bool	%M0.7	System Timing — 2 s Pulse Scan synchronised
_SysSignals02	Byte	%MB1	System memory byte 02 — Scan signals and common square waves
_CycleTick	Bool	%M1.0	System Timing — Cycle tick (active odd cycles, alternates with _CycleTock)
_CycleTock	Bool	%M1.1	System Timing — Cycle tock (active even cycles, alternates with _CycleTick)
_CycleFirst	Bool	%M1.2	System Timing — First cycle detected
_100msSqW	Bool	%M1.3	System Timing — 100 ms Square wave Scan synchronised
_200msSqW	Bool	%M1.4	System Timing — 200 ms Square wave Scan synchronised
_500msSqW	Bool	%M1.5	System Timing — 500 ms Square wave Scan synchronised
_1sSqW	Bool	%M1.6	System Timing — 1 s Square wave Scan synchronised
_2sSqW	Bool	%M1.7	System Timing — 2 s Square wave Scan synchronised
_ClockMem	Byte	%MB10	Clock Memory (populated by the CPU)
_ClockMem_100msSqW	Bool	%M10.0	Clock Memory — 10.0 Hz square wave 0.1 s Period
_ClockMem_200msSqW	Bool	%M10.1	Clock Memory — 5.00 Hz square wave 0.2 s Period
_ClockMem_400msSqW	Bool	%M10.2	Clock Memory — 2.50 Hz square wave 0.4 s Period
_ClockMem_500msSqW	Bool	%M10.3	Clock Memory — 2.00 Hz square wave 0.5 s Period
_ClockMem_800msSqW	Bool	%M10.4	Clock Memory — 1.25 Hz square wave 0.8 s Period
_ClockMem_1000msSqW	Bool	%M10.5	Clock Memory — 1.00 Hz square wave 1.0 s Period
_ClockMem_1600msSqW	Bool	%M10.6	Clock Memory — 0.62 Hz square wave 1.6 s Period
_ClockMem_2000msSqW	Bool	%M10.7	Clock Memory — 0.50 Hz square wave 2.0 s Period

Table 6.1 PAL system bit memory usage (PAL_SystemTags table)

(11) All PAL system tags contained within the PAL_SystemTags tag table are identified by a leading underscore character (_).

6.2 IO Data

(1) The inputs and outputs associated with a project are unique to that project (they depend on the plant being controlled). The PAL does not prescribe in anyway what IO can be used. It does however, define certain rules for how that IO should be named and where the tags should be stored.

(2) IO tags within the PAL are stored in their own tag table:

`PAL_IOTags`

(3) The following is an example of an IO tag table (this is part of the IO listed in Table 3.1 for the test rig):

SYMBOL	TYPE	ADDRESS	DESCRIPTION
ESTOP_HEALTHY	Bool	%I0.0	Emergency stop healthy/pressed
M001_RUNNING	Bool	%I0.1	M001 is running/stopped
M001_TRIPPED	Bool	%I0.2	M001 is heathy/tripped
M002_RUNNING	Bool	%I0.3	M002 is running/stopped
M002_FAULT	Bool	%I0.4	M002 is heathy/inverter fault
M001_ROTATION	Bool	%I0.5	M001 rotation sensor (proximity PD001)
CV001_OPENED_LIM	Bool	%I0.6	CV001 opened limit switch active/inactive
CV001_CLOSED_LIM	Bool	%I0.7	CV001 closed limit switch active/inactive
V001_OPENED_LIM	Bool	%I1.0	V001 opened limit switch active/inactive
V001_CLOSED_LIM	Bool	%I1.1	V001 closed limit switch active/inactive
V002_OPENED_LIM	Bool	%I1.2	V002 opened limit switch active/inactive
V002_CLOSED_LIM	Bool	%I1.3	V002 closed limit switch active/inactive
V003_OPENED_LIM	Bool	%I1.4	V003 opened limit switch active/inactive
V003_CLOSED_LIM	Bool	%I1.5	V003 closed limit switch active/inactive
V004_OPENED_LIM	Bool	%I1.6	V004 opened limit switch active/inactive
V004_CLOSED_LIM	Bool	%I1.7	V004 closed limit switch active/inactive
M001_START_CMD	Bool	%Q0.0	M001 start command
M002_ENABLE_CMD	Bool	%Q0.1	M002 enable command
CV001_ENABLE_CMD	Bool	%Q0.2	CV001 enable command
V001_OPERATE_CMD	Bool	%Q0.3	V001 operate command (energise)
V002_OPERATE_CMD	Bool	%Q0.4	V001 operate command (energise)
V003_OPERATE_CMD	Bool	%Q0.5	V001 operate command (energise)
V004_OPERATE_CMD	Bool	%Q0.6	V001 operate command (energise)
M002_SPEED_ACT	Int	%IW268	M002 actual speed
CV001_POS_ACT	Int	%IW270	CV001 actual position
M002_SPEED_DEM	Int	%QW264	M002 demanded speed
CV001_POS_DEM	Int	%QW266	CV001 demanded position

Table 6.2 PAL IO tag table (example)

6.2.1 IO Tag naming conventions

(1) There are some general rules for naming IO tags:

① The IO tag name is in uppercase

② The IO tag name must be no more than 24 characters

③ Only use the characters [A-Z], the numbers [0-9] and the underscore character [_]

④ The underscore character should be used in place of a space to separate words

(2) The structure of an IO tag is also defined (to some extent) within the PAL. This specifies the nomenclature used for various common signals.

(3) All IO tags are associated with some form of device or instrument (valves, drives, flow meters, level transducers &c.). All of these devices will be allocated a particular equipment number (also confusingly referred to as *"tag numbers"*).

(4) Equipment numbers usually have the form:

`FFFnnn`

Where `FFF` indicates the function of the equipment (e.g. `FIC` for `FLOW INDICATION CONTROL` or `LT` for `LEVEL TRANSMITTER`); and `nnn` indicates a loop number.

(5) The requirement and format for equipment tags is dictated by the design of the plant in question and the PAL will accommodate any format of equipment number. The only restriction being that each device and instrument must have a unique equipment number (that is the combination of the equipment function and its loop number must be unique within the plant).

Note *The requirement for unique equipment numbers is usually easy to accomplish; instruments and devices are generally uniquely identified on the piping and instrumentation diagrams (P&ID) for the plant.*

(6) While a particular device or instrument is uniquely identified by its equipment number. In terms of the IO associated with that equipment, there are usually several signals that have to be named within the tag table. For example, an isolating valve may have an open limit signal (the valve has achieved the fully opened state), a closed limit signal (the valve has reached the fully closed state) and an operate signal (energised to open the valve and de-energised to close the valve).

(7) All three signals would have the same equipment number; consequently the PAL IO tag table requires further information to uniquely identify the individual signals associated with a device linked to the controller.

(8) PAL IO tags generally have the following naming format:

`FFFnnn_SIGNAL_QUALIFIER`

(9) Where `FFFnnn` is the equipment number. `SIGNAL` indicates the primary function of the signal (e.g. `LIMIT` for a valve limit switch); `QUALIFIER` is some qualifying parameter that further explains the function of the signal (e.g. `LIMIT_CLOSED` to identify the closed limit switch of a valve).

(10) Both the `SIGNAL` and `QUALIFIER` components of the tag are optional; some digital instruments have only one signal and the equipment number is sufficient to fully specify the function of the instrument (e.g. `LSL001` identifies the instrument as a low level switch, it would not be necessary to further qualify the tag: `LSL001_LOW` or `LSL001_LEVEL_LOW` would not provide any more information than that given in the equipment number).

(11) There are various predefined values for the `SIGNAL` and `QUALIFIER` components of an IO tag. These are listed in the following tables and should be used where they are applicable.

SIGNAL	APPLIES TO	MEANING	STATES
AUTO	Input	Equipment is switched to automatic control	1 = Auto, 0 = Man (or not used)
CLOSE	Output	Signal to close a bistable valve, damper, louver &c.	1 = Close signal is energised
DISABLE	Output	Signal to disable the operation of a device	1 = Disable, 0 = Enable
DISABLED	Input	Device is disabled	1 = Disabled, 0 = Enabled
ENABLE	Output	Signal to enable the operation of a device	1 = Enabled, 0 = Disabled
ENABLED	Input	Device is enabled	1 = Enabled, 0 = Disabled
ESTOP	Input	Emergency stop	Requires qualifier
FAULT	Input	Device is in fault	1 = Fault, 0 = OK
FBK	Input	Feedback signal	Requires qualifier
FORWARD	Output	Signal to start a drive in the forwards direction	1 = Run Forwards signal is energised
HEALTHY	Input	Device is healthy	1 = Healthy, 0 = not healthy
ILOCK	Input	Interlock	Requires qualifier
LIMIT	Input	Limit switch condition	1 = limit switch active, 0 = inactive
MAN	Input	Equipment is switched to manual control	1 = Man, 0 = Auto (or not used)
OPEN	Output	Signal to open a bistable valve, damper, louver &c.	1 = Open signal is energised
OPERATE	Output	Signal to operate a (monostable) device	1 = Device operate signal is energised
POSN	Both	Position (of something e.g. a modulating valve)	Requires qualifier
RAW	Both	Raw (unscaled or unfiltered) signal	Requires qualifier
REVERSE	Output	Signal to start a drive in the reverse direction	1 = Run Reverse signal is energised
RUNNING	Input	Device is running	1 = running, 0 = not running
SPEED	Both	Speed (of something e.g. a variable speed drive)	Requires qualifier
START	Output	Signal to start a bistable device.	1 = Start signal is energised
STOP	Output	Signal to stop a bistable device.	1 = Stop signal is energised
TRIP	Input	Device is tripped	1 = Tripped, 0 = OK

Table 6.3 PAL IO tag SIGNAL values

(12) The above SIGNAL list for PAL IO is not exhaustive (there willl always be some special device that is not accomodated by the entries above), but it does cover a wide range of common signals and should be used in preference to other non-standard values.

QUALIFIER	APPLIES TO	MEANING	EXAMPLE
CLOSED	Input	The SIGNAL (e.g. LIMIT) represents a closed state	LIMIT_CLOSED
CMD	Output	The SIGNAL is a command to a device (usually digital)	OPERATE_CMD
DMD	Output	The SIGNAL is a demand to a device (usually analogue)	SPEED_DMD
FBK	Input	Marks a SIGNAL as a feedback signal	SPEED_FBK
OPENED	Input	The SIGNAL (e.g. LIMIT) represents an opened state	LIMIT_OPENED
RAW	Both	Raw (unscaled or unfiltered) signal	SPEED_RAW

Table 6.4 PAL IO tag QUALIFIER values

(13) Again, the above `QUALIFIER` list for PAL IO is not exhaustive, but it does cover a wide range of common signals and should be used in preference to other non-standard values.

A note on monostable and bistable output signals

(14) In the `SIGNAL` list (Table 6.3) there are four output signals that are specified as bistable:

- `OPEN`
- `CLOSE`
- `START, FORWARDS, REVERSE`
- `STOP`

(15) There is also one monostable output:

- `OPERATE`

(16) **Bistable** signals should be used where a device has two signals to make it change state; consider a valve that has both an `OPEN` output signal and a `CLOSE` output signal.

(17) To open the valve the `OPEN` signal must be energised and the `CLOSE` signal de-energised. When the valve reaches the `OPENED` position, both signals can be de-energised and the valve will remain in the `OPENED` state (it is stable in this condition and does not require any signal to maintain it in this state, if there were a power failure the valve would remain opened).

(18) To close the valve the `CLOSE` signal must be energised and the `OPEN` signal de-energised. When the valve reaches the `CLOSED` position, both signals can again be de-energised and the valve will remain in the `CLOSED` state (again the valve is stable in this condition and does not require any signal to maintain it in this state, if there were a power failure the valve would remain closed).

(19) This is said to be a bistable device because once it is in a particular state, it does not require any signal to be energised to maintain that state.

(20) The `START` and `STOP` signals operate in exactly the same way for drives and devices that can broadly be described as running or stopped (it might for example be a more complicated standalone piece of machinery such as a labelling device).

(21) Bistable devices are not very common; they tend to be used with very large motorised valves and specialist machinery.

(22) **Monostable** devices are what would be consider the standard type of device. These are usually things like a normally closed valve and direct online drives. Monostable devices usually have just one signal that operates the device.

(23) Take for example a normally closed valve. This will have a single `OPERATE` signal. If the `OPERATE` signal is energised, the valve will (either electrically or electro-pneumatically) open. If the `OPERATE` signal is de-energised, the valve will return (mechanically, usually via a spring) to the closed position. To keep the valve open, the `OPERATE` signal must remain energised.

(24) Direct online drives work in much the same way. The `OPERATE` signal activates a relay or contactor that applies electrical power to the drive, if the `OPERATE` signal is de-energised, the relay or contactor is mechanically opened (usually a spring mechanism that opens the electrical contacts) and power is removed from the drive.

(25) Again, the `OPERATE` signal must remain active for the drive to run.

Most valves and drives are monostable and use the `OPERATE` command rather than the `OPEN`/`CLOSE` or `START`/`STOP` signals.

6.3 Data block data storage

(1) Data blocks are the primary mechanism for storing data within any PAL based project. Data stored within data blocks is the main method for standard modules to communicate with the rest of the project software, it is how application modules pass information to and from the standard modules.

(2) Most standard modules received data block data in two forms: static (read only) data and dynamic (read and write) data[14]. This data is passed to the block via the parameters STATIC_DATA and DYNAMIC_DATA (see § 4.3.1). This data is always passed to the standard module in the form of a user data type (UDT) that is specific to both the module and to the static/dynamic data in question (the static data will use a different UDT to that of the dynamic data).

(3) Standard modules will only have one STATIC_DATA parameter and one DYNAMIC_DATA parameter each; all the stored data needed by the module must be passed to the module by these parameters. Consequently, the UDTs that hold this data can be extensive and relatively complex in nature.

(4) To simplify these UDTs, common naming practices are adopted for similar types of signal; for example, data that reflects the status of the device or instrument being operated by a standard module is prefixed with the label **`status`**, similarly where operating modes can be selected, the data is prefixed with the label **`mode`**. Configuration data is prefixed **`CONFIG`** and alarms, messages and warning with the prefix **`msg`** &c.

(5) In this context, static data specifies constant (preset) values that have some meaning for the block in question (e.g. the opening time of a valve, the hysteresis of an alarm setpoint, limit switch arrangements for a valve &c.). Static data does not change (the data is usually configured during the commissioning of the plant and then remains fixed and unchanging for the lifetime of the plant).

(6) Dynamic data is live, operating data (e.g. if a valve is in the process of opening, the elapsed time of the operation will be stored in the dynamic data area).

---

14 While most standard modules have both static and dynamic data, some have only dynamic data and some (certain simple subroutines) require neither.

(7) Static and dynamic data is always stored in a data block, the data block in question is dependent on the number allocated to the standard module.

(8) This process is best explained with the use of an example, consider the standard module associated with the reading, scaling and monitoring of an analogue instrument connected to a Controller via an analogue input card.

(9) This standard module is designated (*FC02001_StdInstAnalogRead*) and is allocated to the function FC 02001 within a Controller. This module would be called from a marshalling application block (*FC22001_AppInstAnalogRead*):

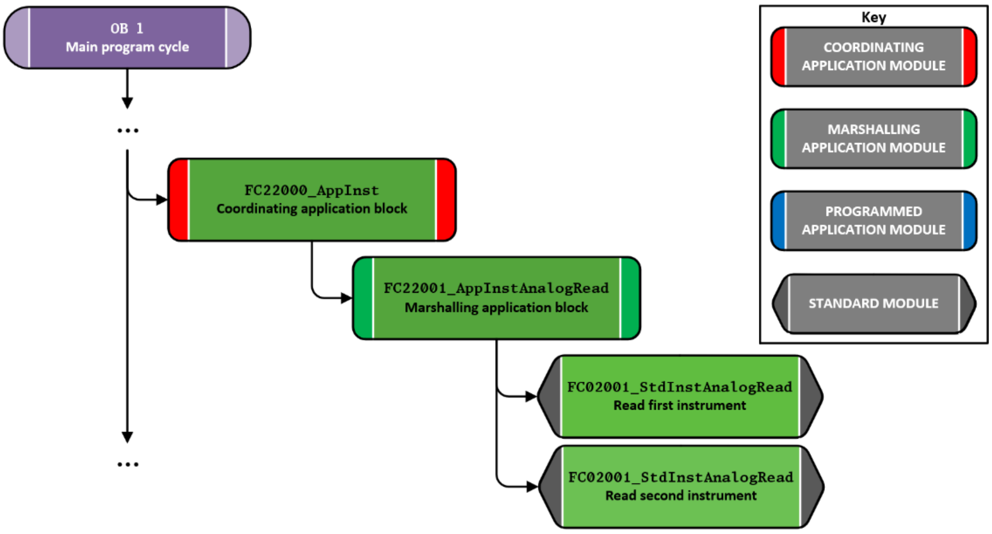


Figure 6.1 Analogue instrument read example calling structure

(10) In this example, the first instance of *FC02001_StdInstAnalogRead* is reading the value of a flow transmitting instrument (FT001), the second instance is reading the value of a level transmitting instrument (LT001).

(11) In practical terms, the called blocks would be programmed as follows (within the marshalling block):

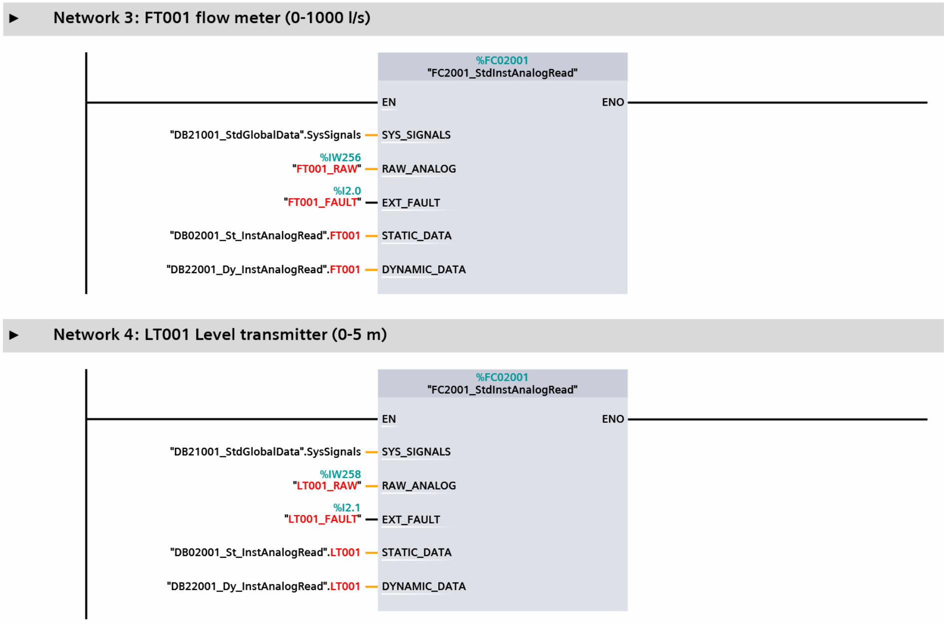


Figure 6.2 Analogue instrument read example programmed blocks

(12) The standard module is assigned to the function FC 02001, the static data is assigned to the data block with exactly the same number, in this case DB 02001 (specifically: DB02001_St_InstAnalog Read). The dynamic data is assigned to the data block with the same number as the standard block + 20000, i.e. DB 22001 (specifically **DB22001_Dy_InstAnalog Read**).

(13) The rules for the two data blocks are as follows:

① The static DB has the same number as the standard module

② The dynamic DB has the same number as the standard module + 20000

③ The static DB has the same name as the standard module with `Std` replaced by `St_` (static)

④ The dynamic DB has the same name as the standard module with `Std` replaced by `Dy_` (dynamic)

(14) In the example of Figure 6.2, both calls to the standard module (*FC02001_StdInstAnalogRead*) use the same data block for the static data (DB02001_St_InstAnalog Read), they also use the same data block for the dynamic data (**DB22001_Dy_InstAnalog Read**).

(15) I.e. all instruments that are read using FC 02001 use the same data block to store the static data: DB 02001. The same is true for the dynamic data, all analogue instrument reads use DB 22001. This can be seen by examining the two data blocks:

DB02001_St_InstAnalogRead

	Name	Data type	Start value	Comment
1	Static			
2	_DB_Header	Array[0..79] of Bool		STANDARD ANALOGUE INSTRUMENT READ
3	__________0000_0	Bool	false	
4	__________0000_1	Bool	false	ANALOGUE INSTRUMENTS
5	FT001	"UT02001_St_InstAnalogRead"		FT001 Flow Transmitter (0-1000 l/s)
6	LT001	"UT02001_St_InstAnalogRead"		LT001 Level Transmitter (0-5 m)

Figure 6.3 Analogue instrument read static data block

DB22001_Dy_InstAnalogRead

	Name	Data type	Start value	Comment
1	Static			
2	_DB_Header	Array[0..79] of Bool		STANDARD ANALOGUE INSTRUMENT READ
3	__________0000_0	Bool	false	
4	__________0000_1	Bool	false	ANALOGUE INSTRUMENTS
5	FT001	"UT22001_Dy_InstAnalogRead"		FT001 Flow Transmitter (0-1000 l/s)
6	LT001	"UT22001_Dy_InstAnalogRead"		LT001 Level Transmitter (0-5 m)

Figure 6.4 Analogue instrument read dynamic data block

(16) The two instruments FT001 and LT001 are each present in the two data block. In the static data block, each instrument has a data type of the UT02001_St_InstAnalogRead, this is the static UDT associated with the data, this again has the same number as the standard module UT02001 and has the same name as the static data block.

(17) Similarly, in the dynamic data block, each instrument has a data type of the **UT22001_St_InstAnalogRead**, this is the dynamic UDT associated with the data, like the dynamic DB, this has the same number as the standard module + 20000 and has the same name as the dynamic data block.

(18) By expanding the instrument variables within the two DBs, the internal structure of the UDTs can be seen (here the FT001 instrument is expanded):

DB02001_St_InstAnalogRead

	Name	Data type	Start value	Comment
1	Static			
2	_DB_Header	Array[0..79] of Bool		STANDARD ANALOGUE INSTRUMENT READ
3	__________0000_0	Bool	false	
4	__________0000_1	Bool	false	ANALOGUE INSTRUMENTS
5	FT001	"UT02001_St_InstAnalogRead"		FT001 Flow Transmitter (0-1000 l/s)
6	__________0000_0	Int	0	INSTRUMENT CONFIGURATION
7	CONFIG_ALM_H_ENABLE	Bool	true	CONFIG — High alarm is enabled (1 = enabled, 0 = no alarm)
8	CONFIG_ALM_L_ENABLE	Bool	true	CONFIG — Low alarm is enabled (1 = enabled, 0 = no alarm)
9	CONFIG_WRN_H_ENABLE	Bool	true	CONFIG — High warning is enabled (1 = enabled, 0 = no warning)
10	CONFIG_WRN_L_ENABLE	Bool	true	CONFIG — Low warning is enabled (1 = enabled, 0 = no warning)
11	CONFIG_FP_DISABLE	Bool	false	CONFIG — Faceplate is disabled (1 = no Faceplate, 0 = normal)
12	CONFIG_SIM_DISABLE	Bool	false	CONFIG — Simulation is disabled (1 = no Simulation, 0 = normal)
13	CONFIG_RL_ENABLE	Bool	false	CONFIG — Remote/local mode enabled (1 = remote/local permitted, 0 = remote/local N/A)
14	__________0010_0	Int	0	
15	__________0010_1	Int	0	INSTRUMENT RANGE & SCALING DATA
16	RANGE_RAW_MIN	Int	0	RANGE — Minimum value of the raw analogue signal (at the card)
17	RANGE_RAW_MAX	Int	27648	RANGE — Maximum value of the raw analogue signal (at the card)
18	RANGE_SCALE_MIN	Real	0.0	RANGE — Minimum value of the scaled analogue signal [engineering units]
19	RANGE_SCALE_MAX	Real	1000.0	RANGE — Maximum value of the scaled analogue signal [engineering units]
20	RANGE_OOR_PERCENT	Real	2.5	RANGE — Out of range percentage (of raw range), beyond which the instrument is out of range
21	__________0020_0	Int	0	
22	__________0020_1	Int	0	INSTRUMENT INFORMATION (TAG & UNITS)
23	INFO_TAG	String[20]	'FT001'	INFORMATION — Instrument ID tag
24	INFO_UNITS	String[10]	'l/s'	INFORMATION — Engineering units of the instrument (unit of measure)
25	__________0030_0	Int	0	
26	__________0030_2	Int	0	INSTRUMENT ALARM/WARNING & HYSTERESIS SETPOINTS
27	SP_ALM_H_VAL	Real	750.0	SETPOINT — High alarm threshold value [engineering units]
28	SP_ALM_L_VAL	Real	250.0	SETPOINT — Low alarm threshold value [engineering units]
29	SP_WRN_H_VAL	Real	625.0	SETPOINT — High warning threshold value [engineering units]
30	SP_WRN_L_VAL	Real	375.0	SETPOINT — Low warning threshold value [engineering units]
31	SP_ALM_H_HYST_VAL	Real	62.5	SETPOINT — High alarm hysteresis value [engineering units]
32	SP_ALM_L_HYST_VAL	Real	62.5	SETPOINT — Low alarm hysteresis value [engineering units]
33	SP_WRN_H_HYST_VAL	Real	62.5	SETPOINT — High warning hysteresis value [engineering units]
34	SP_WRN_L_HYST_VAL	Real	62.5	SETPOINT — Low warning hysteresis value [engineering units]
35	__________0040_0	Int	0	
36	__________0040_2	Int	0	INSTRUMENT TIMER DEFAULT VALUES
37	TIME_ALM_H_ON_DEL	Real	10.0	TIMER — High alarm on delay time (activation delay), [seconds]
38	TIME_ALM_L_ON_DEL	Real	10.0	TIMER — Low alarm on delay time (activation delay), [seconds]
39	TIME_WRN_H_ON_DEL	Real	10.0	TIMER — High warning on delay time (activation delay), [seconds]
40	TIME_WRN_L_ON_DEL	Real	10.0	TIMER — Low warning on delay time (activation delay), [seconds]
41	TIME_ALM_H_OFF_DEL	Real	15.0	TIMER — High alarm off delay time (deactivation delay), [seconds]
42	TIME_ALM_L_OFF_DEL	Real	15.0	TIMER — Low alarm off delay time (deactivation delay), [seconds]
43	TIME_WRN_H_OFF_DEL	Real	15.0	TIMER — High warning off delay time (deactivation delay), [seconds]
44	TIME_WRN_L_OFF_DEL	Real	15.0	TIMER — Low warning off delay time (deactivation delay), [seconds]
45	LT001	"UT02001_St_InstAnalogRead"		LT001 Level Transmitter (0-5 m)

Figure 6.5 FT001 static data block UDT structure

(19) In Figure 6.6, the configuration information for FT001 is visible, it can be seen, amongst other things, that the scaled range of the instrument is set to be from `0.0` (RANGE_SCALE_MIN) to `1000.0` (RANGE_SCALE_MAX), it can also be seen that all four alarms and warnings are enabled (CONFIG_ALM_H_ENABLE, CONFIG_ALM_L_ENABLE, CONFIG_WRN_H_ENABLE and CONFIG_WRN_L_ENABLE are all set to `true`).

(20) Comparing this with the same information for LT001:

DB02001_St_InstAnalogRead

	Name	Data type	Start value	Comment
1	Static			
2	_DB_Header	Array[0..79] of Bool		STANDARD ANALOGUE INSTRUMENT READ
3	________0000_0	Bool	false	
4	________0000_1	Bool	false	ANALOGUE INSTRUMENTS
5	FT001	"UT02001_St_InstAnalogRead"		FT001 Flow Transmitter (0-1000 l/s)
6	LT001	"UT02001_St_InstAnalogRead"		LT001 Level Transmitter (0-5 m)
7	________0000_0	Int	0	INSTRUMENT CONFIGURATION
8	CONFIG_ALM_H_ENABLE	Bool	false	CONFIG — High alarm is enabled (1 = enabled, 0 = no alarm)
9	CONFIG_ALM_L_ENABLE	Bool	true	CONFIG — Low alarm is enabled (1 = enabled, 0 = no alarm)
10	CONFIG_WRN_H_ENABLE	Bool	false	CONFIG — High warning is enabled (1 = enabled, 0 = no warning)
11	CONFIG_WRN_L_ENABLE	Bool	true	CONFIG — Low warning is enabled (1 = enabled, 0 = no warning)
12	CONFIG_FP_DISABLE	Bool	false	CONFIG — Faceplate is disabled (1 = no Faceplate, 0 = normal)
13	CONFIG_SIM_DISABLE	Bool	false	CONFIG — Simulation is disabled (1 = no Simulation, 0 = normal)
14	CONFIG_RL_ENABLE	Bool	false	CONFIG — Remote/local mode enabled (1 = remote/local permitted, 0 = remote/local N/A)
15	________0010_0	Int	0	
16	________0010_1	Int	0	INSTRUMENT RANGE & SCALING DATA
17	RANGE_RAW_MIN	Int	0	RANGE — Minimum value of the raw analogue signal (at the card)
18	RANGE_RAW_MAX	Int	27648	RANGE — Maximum value of the raw analogue signal (at the card)
19	RANGE_SCALE_MIN	Real	0.0	RANGE — Minimum value of the scaled analogue signal [engineering units]
20	RANGE_SCALE_MAX	Real	5.0	RANGE — Maximum value of the scaled analogue signal [engineering units]
21	RANGE_OOR_PERCENT	Real	2.5	RANGE — Out of range percentage (of raw range), beyond which the instrument is out of range
22	________0020_0	Int	0	
23	________0020_1	Int	0	INSTRUMENT INFORMATION (TAG & UNITS)
24	INFO_TAG	String[20]	'LT001'	INFORMATION — Instrument ID tag
25	INFO_UNITS	String[10]	'm'	INFORMATION — Engineering units of the instrument (unit of measure)
26	________0030_0	Int	0	
27	________0030_2	Int	0	INSTRUMENT ALARM/WARNING & HYSTERESIS SETPOINTS
28	SP_ALM_H_VAL	Real	0.0	SETPOINT — High alarm threshold value [engineering units]
29	SP_ALM_L_VAL	Real	0.5	SETPOINT — Low alarm threshold value [engineering units]
30	SP_WRN_H_VAL	Real	0.0	SETPOINT — High warning threshold value [engineering units]
31	SP_WRN_L_VAL	Real	0.75	SETPOINT — Low warning threshold value [engineering units]
32	SP_ALM_H_HYST_VAL	Real	0.0	SETPOINT — High alarm hysteresis value [engineering units]
33	SP_ALM_L_HYST_VAL	Real	0.1	SETPOINT — Low alarm hysteresis value [engineering units]
34	SP_WRN_H_HYST_VAL	Real	0.0	SETPOINT — High warning hysteresis value [engineering units]
35	SP_WRN_L_HYST_VAL	Real	0.1	SETPOINT — Low warning hysteresis value [engineering units]
36	________0040_0	Int	0	
37	________0040_2	Int	0	INSTRUMENT TIMER DEFAULT VALUES
38	TIME_ALM_H_ON_DEL	Real	10.0	TIMER — High alarm on delay time (activation delay), [seconds]
39	TIME_ALM_L_ON_DEL	Real	10.0	TIMER — Low alarm on delay time (activation delay), [seconds]
40	TIME_WRN_H_ON_DEL	Real	10.0	TIMER — High warning on delay time (activation delay), [seconds]
41	TIME_WRN_L_ON_DEL	Real	10.0	TIMER — Low warning on delay time (activation delay), [seconds]
42	TIME_ALM_H_OFF_DEL	Real	15.0	TIMER — High alarm off delay time (deactivation delay), [seconds]
43	TIME_ALM_L_OFF_DEL	Real	15.0	TIMER — Low alarm off delay time (deactivation delay), [seconds]
44	TIME_WRN_H_OFF_DEL	Real	15.0	TIMER — High warning off delay time (deactivation delay), [seconds]
45	TIME_WRN_L_OFF_DEL	Real	15.0	TIMER — Low warning off delay time (deactivation delay), [seconds]

Figure 6.6 LT001 static data block UDT structure

(21) It can be seen that the data is different; in this case it is applicable to LT001, the scaled range of the instrument is set to be from `0.0` (RANGE_SCALE_MIN) to `5.0` (RANGE_SCALE_MAX), it can also be seen that only the low alarm and low warnings are enabled (only CONFIG_ALM_L_ENABLE and CONFIG_WRN_L_ENABLE are set to `true`).

(22) This is the mechanism by which data is stored and passed to the standard modules, each instance of the standard modules is given static (or dynamic) data in the same data block, but from a different variable within that data block, here the first instance uses the variable FT001 and the second instance LT001.

(23) This can be seen with the dynamic data too:

DB22001_Dy_InstAnalogRead

	Name	Data type	Start value	Comment
1	Static			
2	_DB_Header	Array[0..79] of Bool		STANDARD ANALOGUE INSTRUMENT READ
3	________0000_0	Bool	false	
4	________0000_1	Bool	false	———— ANALOGUE INSTRUMENTS
5	FT001	"UT22001_Dy_InstAnalogRead"		FT001 Flow Transmitter (0-1000 l/s)
6	________0000_0	Int	0	———— STATUS (FOR BLOCK ICON AND SYMBOL)
7	status_Alm_H	Bool	false	STATUS — High alarm is active (1 = alarm active, 0 = no alarm)
8	status_Alm_L	Bool	false	STATUS — Low alarm is active (1 = alarm active, 0 = no alarm)
9	status_Wrn_H	Bool	false	STATUS — High warning is active (1 = warning active, 0 = no warning)
10	status_Wrn_L	Bool	false	STATUS — Low warning is active (1 = warning active, 0 = no warning)
11	status_Alm_H_Mask	Bool	false	STATUS — High alarm is masked (1 = alarm masked, 0 = normal)
12	status_Alm_L_Mask	Bool	false	STATUS — Low alarm is masked (1 = alarm masked, 0 = normal)
13	status_Wrn_H_Mask	Bool	false	STATUS — High warning is masked (1 = warning masked, 0 = normal)
14	status_Wrn_L_Mask	Bool	false	STATUS — Low warning is masked (1 = warning masked, 0 = normal)
15	status_Alm_H_Dis	Bool	false	STATUS — High alarm is disabled (1 = alarm disabled, 0 = normal)
16	status_Alm_L_Dis	Bool	false	STATUS — Low alarm is disabled (1 = alarm disabled, 0 = normal)
17	status_Wrn_H_Dis	Bool	false	STATUS — High warning is disabled (1 = warning disabled, 0 = normal)
18	status_Wrn_L_Dis	Bool	false	STATUS — Low warning is disabled (1 = warning disabled, 0 = normal)
19	status_Fault	Bool	false	STATUS — Instrument is in fault (1 = fault present, 0 = healthy)
20	status_SimOn	Bool	false	STATUS — Instrument is in simulation mode (1 = simulation mode on, 0 = normal)
21	status_RemoteOn	Bool	false	STATUS — Instrument is in remote mode (1 = remote mode, 0 = remote mode off)
22	status_LocalOn	Bool	false	STATUS — Instrument is in local mode (1 = local mode, 0 = local mode off)
23	status_RLOff	Bool	false	STATUS — Remote/local mode disabled (1 = ALL mode on, 0 = RL mode selected)
24	________0010_0	Int	0	
25	________0010_1	Int	0	———— OPERATING MODE SELECTION (FROM FACEPLATE OR PANEL)
26	mode_Alm_H_MaskOn	Bool	false	MODE — Mask Alm_H (1 = mask alarm, 0 = normal)
27	mode_Alm_L_MaskOn	Bool	false	MODE — Mask Alm_L (1 = mask alarm, 0 = normal)
28	mode_Wrn_H_MaskOn	Bool	false	MODE — Mask Wrn_H (1 = mask warning, 0 = normal)
29	mode_Wrn_L_MaskOn	Bool	false	MODE — Mask Wrn_H (1 = mask warning, 0 = normal)
30	mode_SimOn	Bool	false	MODE — Simulation mode (1 = simulation mode active, 0 = normal)
31	mode_SimValue	Real	0.0	MODE — Simulation value [engineering units]
32	mode_LocalOn	Bool	false	MODE — Local HMI control enabled (1 = control active, 0 = control disabled or N/A)
33	mode_RemoteOn	Bool	false	MODE — Remote SCADA control enabled (1 = control active, 0 = control disabled or N/A)
34	________0020_0	Int	0	
35	________0020_1	Int	0	———— MESSAGES (ALARMS, WARNINGS, FAULTS AND EVENTS)
36	msg_Alm_H	Bool	false	MESSAGE — Alarm - high alarm is active
37	msg_Alm_L	Bool	false	MESSAGE — Alarm - low alarm is active
38	msg_Wrn_H	Bool	false	MESSAGE — Warning - high warning is active
39	msg_Wrn_L	Bool	false	MESSAGE — Warning - low warning is active
40	msg_Flt_External	Bool	false	MESSAGE — Fault - external fault signal is active
41	msg_Flt_OverRange	Bool	false	MESSAGE — Fault - instrument is over-range
42	msg_Flt_UnderRange	Bool	false	MESSAGE — Fault - instrument is under-range
43	msg_Flt_OutOfRange	Bool	false	MESSAGE — Fault - instrument is out-of-range
44	________0030_0	Int	0	
45	________0030_1	Int	0	———— BATCH AND BOOKING DATA
46	batch_ID	Int	0	BATCH — Booking ID (optional for batch operations)
47	________0040_0	Int	0	
48	________0040_1	Int	0	———— LIVE DATA (SCALED READING & TIMER VALUES)
49	actual_Value	Real	0.0	ACTUAL — SCALED INSTRUMENT VALUE [engineering units]
50	actual_Alm_H_Timer	Real	0.0	ACTUAL — Timer value — alarm high operation timer [seconds]
51	actual_Alm_L_Timer	Real	0.0	ACTUAL — Timer value — alarm low operation timer [seconds]
52	actual_Wrn_H_Timer	Real	0.0	ACTUAL — Timer value — warning high operation timer [seconds]
53	actual_Wrn_L_Timer	Real	0.0	ACTUAL — Timer value — warning low operation timer [seconds]
54	________0090_0	Int	0	
55	________0090_1	Int	0	———— BLOCK INTERNAL WORKING AND STORAGE AREA
56	$pret_AlmHEn	Bool	false	INTERNAL — Edge retention (+ve) alarm high enable
57	$Pret_AlmLEn	Bool	false	INTERNAL — Edge retention (+ve) alarm low enable
58	$pret_WrnHEn	Bool	false	INTERNAL — Edge retention (+ve) warning high enable
59	$pret_WrnLEn	Bool	false	INTERNAL — Edge retention (+ve) warning low enable
60	LT001	"UT22001_Dy_InstAnalogRead"		LT001 Level Transmitter (0-5 m)

Figure 6.7 FT001 dynamic data block UDT structure

(24) Compared with LT001:

DB22001_Dy_InstAnalogRead

	Name	Data type	Start value	Comment
1	▼ Static			
2	▶ _DB_Header	Array[0..79] of Bool		STANDARD ANALOGUE INSTRUMENT READ
3	________0000_0	Bool	false	
4	________0000_1	Bool	false	—— ANALOGUE INSTRUMENTS
5	▶ FT001	"UT22001_Dy_InstAnalogRead"		FT001 Flow Transmitter (0-1000 l/s)
6	▼ LT001	"UT22001_Dy_InstAnalogRead"		LT001 Level Transmitter (0-5 m)
7	________0000_0	Int	0	—— STATUS (FOR BLOCK ICON AND SYMBOL)
8	status_Alm_H	Bool	false	STATUS — High alarm is active (1 = alarm active, 0 = no alarm)
9	status_Alm_L	Bool	false	STATUS — Low alarm is active (1 = alarm active, 0 = no alarm)
10	status_Wrn_H	Bool	false	STATUS — High warning is active (1 = warning active, 0 = no warning)
11	status_Wrn_L	Bool	false	STATUS — Low warning is active (1 = warning active, 0 = no warning)
12	status_Alm_H_Mask	Bool	false	STATUS — High alarm is masked (1 = alarm masked, 0 = normal)
13	status_Alm_L_Mask	Bool	false	STATUS — Low alarm is masked (1 = alarm masked, 0 = normal)
14	status_Wrn_H_Mask	Bool	false	STATUS — High warning is masked (1 = warning masked, 0 = normal)
15	status_Wrn_L_Mask	Bool	false	STATUS — Low warning is masked (1 = warning masked, 0 = normal)
16	status_Alm_H_Dis	Bool	false	STATUS — High alarm is disabled (1 = alarm disabled, 0 = normal)
17	status_Alm_L_Dis	Bool	false	STATUS — Low alarm is disabled (1 = alarm disabled, 0 = normal)
18	status_Wrn_H_Dis	Bool	false	STATUS — High warning is disabled (1 = warning disabled, 0 = normal)
19	status_Wrn_L_Dis	Bool	false	STATUS — Low warning is disabled (1 = warning disabled, 0 = normal)
20	status_Fault	Bool	false	STATUS — Instrument is in fault (1 = fault present, 0 = healthy)
21	status_SimOn	Bool	false	STATUS — Instrument is in simulation mode (1 = simulation mode on, 0 = normal)
22	status_RemoteOn	Bool	false	STATUS — Instrument is in remote mode (1 = remote mode, 0 = remote mode off)
23	status_LocalOn	Bool	false	STATUS — Instrument is in local mode (1 = local mode, 0 = local mode off)
24	status_RLOff	Bool	false	STATUS — Remote/local mode disabled (1 = ALL mode on, 0 = RL mode selected)
25	________0010_0	Int	0	
26	________0010_1	Int	0	—— OPERATING MODE SELECTION (FROM FACEPLATE OR PANEL)
27	mode_Alm_H_MaskOn	Bool	false	MODE — Mask Alm_H (1 = mask alarm, 0 = normal)
28	mode_Alm_L_MaskOn	Bool	false	MODE — Mask Alm_L (1 = mask alarm, 0 = normal)
29	mode_Wrn_H_MaskOn	Bool	false	MODE — Mask Wrn_H (1 = mask warning, 0 = normal)
30	mode_Wrn_L_MaskOn	Bool	false	MODE — Mask Wrn_H (1 = mask warning, 0 = normal)
31	mode_SimOn	Bool	false	MODE — Simulation mode (1 = simulation mode active, 0 = normal)
32	mode_SimValue	Real	0.0	MODE — Simulation value [engineering units]
33	mode_LocalOn	Bool	false	MODE — Local HMI control enabled (1 = control active, 0 = control disabled or N/A)
34	mode_RemoteOn	Bool	false	MODE — Remote SCADA control enabled (1 = control active, 0 = control disabled or N/A)
35	________0020_0	Int	0	
36	________0020_1	Int	0	—— MESSAGES (ALARMS, WARNINGS, FAULTS AND EVENTS)
37	msg_Alm_H	Bool	false	MESSAGE — Alarm - high alarm is active
38	msg_Alm_L	Bool	false	MESSAGE — Alarm - low alarm is active
39	msg_Wrn_H	Bool	false	MESSAGE — Warning - high warning is active
40	msg_Wrn_L	Bool	false	MESSAGE — Warning - low warning is active
41	msg_Flt_External	Bool	false	MESSAGE — Fault - external fault signal is active
42	msg_Flt_OverRange	Bool	false	MESSAGE — Fault - instrument is over-range
43	msg_Flt_UnderRange	Bool	false	MESSAGE — Fault - instrument is under-range
44	msg_Flt_OutOfRange	Bool	false	MESSAGE — Fault - instrument is out-of-range
45	________0030_0	Int	0	
46	________0030_1	Int	0	—— BATCH AND BOOKING DATA
47	batch_ID	Int	0	BATCH — Booking ID (optional for batch operations)
48	________0040_0	Int	0	
49	________0040_1	Int	0	—— LIVE DATA (SCALED READING & TIMER VALUES)
50	actual_Value	Real	0.0	ACTUAL — SCALED INSTRUMENT VALUE [engineering units]
51	actual_Alm_H_Timer	Real	0.0	ACTUAL — Timer value — alarm high operation timer [seconds]
52	actual_Alm_L_Timer	Real	0.0	ACTUAL — Timer value — alarm low operation timer [seconds]
53	actual_Wrn_H_Timer	Real	0.0	ACTUAL — Timer value — warning high operation timer [seconds]
54	actual_Wrn_L_Timer	Real	0.0	ACTUAL — Timer value — warning low operation timer [seconds]
55	________0090_0	Int	0	
56	________0090_1	Int	0	—— BLOCK INTERNAL WORKING AND STORAGE AREA
57	$pret_AlmHEn	Bool	false	INTERNAL — Edge retention (+ve) alarm high enable
58	$Pret_AlmLEn	Bool	false	INTERNAL — Edge retention (+ve) alarm low enable
59	$pret_WrnHEn	Bool	false	INTERNAL — Edge retention (+ve) warning high enable
60	$pret_WrnLEn	Bool	false	INTERNAL — Edge retention (+ve) warning low enable

Figure 6.8 LT001 dynamic data block UDT structure

6.3.1 Data block and UDT naming conventions

(1) The following rules apply to naming variables within *static* data blocks and static UDTs:

① The name must be written in uppercase

② The name must be no more than 21 characters

③ Only use the characters [A-Z], the numbers [0-9] and the underscore character [_]

④ The underscore character should be used in place of a space to separate words

⑤ All elements must have a comment in the block interface to explain the function and usage of the element.

(2) The following rules apply to naming variables within *dynamic* data blocks and static UDTs:

① The name must be written in camel case (unless it is an equipment tag, in which case it will be in the case dictated by the tag)

② The name must be no more than 25 characters

③ Only use the characters [a-z], [A-Z], the numbers [0-9] and the underscore character [_]

④ All elements must have a comment in the block interface to explain the function and usage of the element.

6.3.2 DBs holding recipe data

(1) Under certain (very limited) circumstances, the data in a static DB can be overwritten. These circumstances arise when some form of recipe handling is being performed.

(2) Recipes consist of preconfigured data sets that are selected by the operator and then loaded into the Controller (via some external device such as a SCADA or HMI). Such recipe data sets are permitted to overwrite *(overload)* a static DB (essentially the static DB is being selected for a particular set of production requirements).

(3) Once a recipe has overloaded a static DB, the data in that DB is then fixed (and will not be overwritten) until the operator selects a different recipe.

(4) Data blocks that hold recipe data, and are under the control of a recipe, are given the class `Rc_` (rather than `St_`), the individual elements within the DB will retain the properties specified for static DBs in § 6.3.1 (i.e. all uppercase &c.).

BLANK PAGE

7 Application modules

(1) The complete OB 1 PAL structure is shown in Figure 7.1. This shows application block calls to the thirteen functional groups (this includes the 11 functional groups listed in Figure 5.1, plus two *debug* groups: a start of cycle debug and end of cycle debug — debug functional groups are discussed in § 8.13).

(2) All of these functional groups with the exception of the system functions (`FC21000_AppSys`) are optional (the requirements for these applications depends entirely on the purpose of the Controller); most Controllers will have a subset of these functional groups.

(3) Application modules are always installed in functions (FC) within the Controller and do not, under any circumstances, use parametric assignments (unlike standard modules, application blocks have no parameters associated with them).

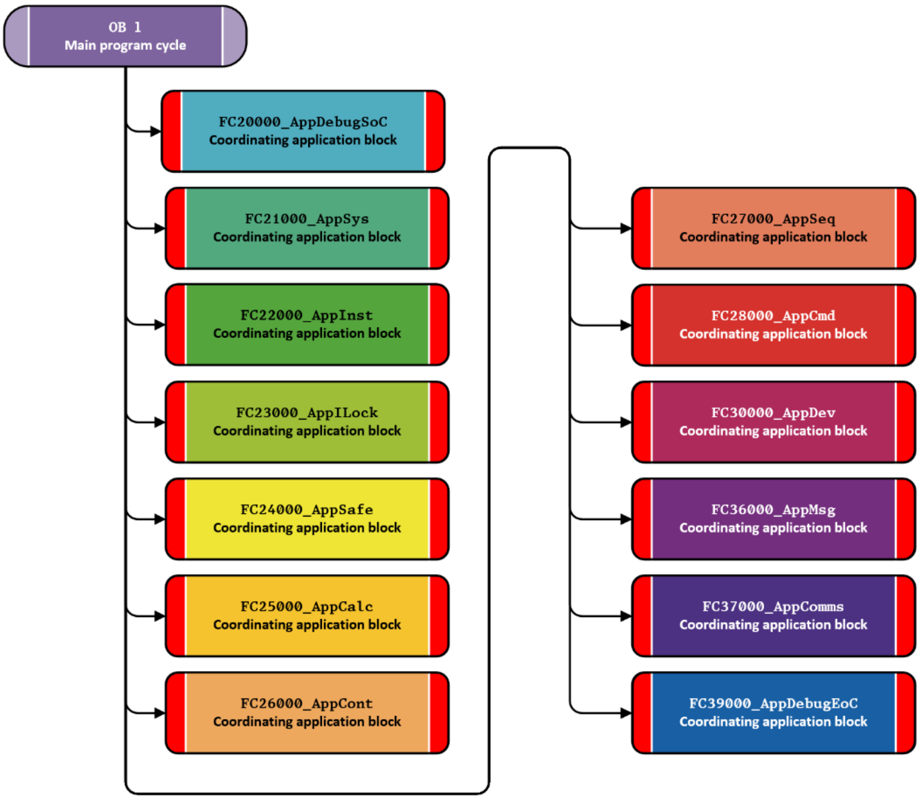


Figure 7.1 Complete OB 1 PAL structure

7.1 Coordinating application modules

(1) The application blocks shown in Figure 7.1 are categorised as *coordinating* application blocks, and these are used to coordinate all the block calls within that particular function group.

(2) Coordinating blocks are always functions (FCs) and the last three digits of the block number are always zero; i.e. `FCgg000` where `gg` reflects the functional group listed in Table 5.1.Each coordinating application block can directly call the standard modules that are associated with that functional group, or can call *marshalling* application modules that further subdivide the functional groups into logical areas, this can be seen in. Figure 7.2

(3) Figure 7.2 shows a coordinating application module (*`FC21000_AppSys`*) calling two standard modules: *`FC01001_StdSysGlobalData`* (used to generates the global timing and logic signals) and *`FC01101_StdSysTimeSync`* (used to synchronise the Controller's internal real time clock).

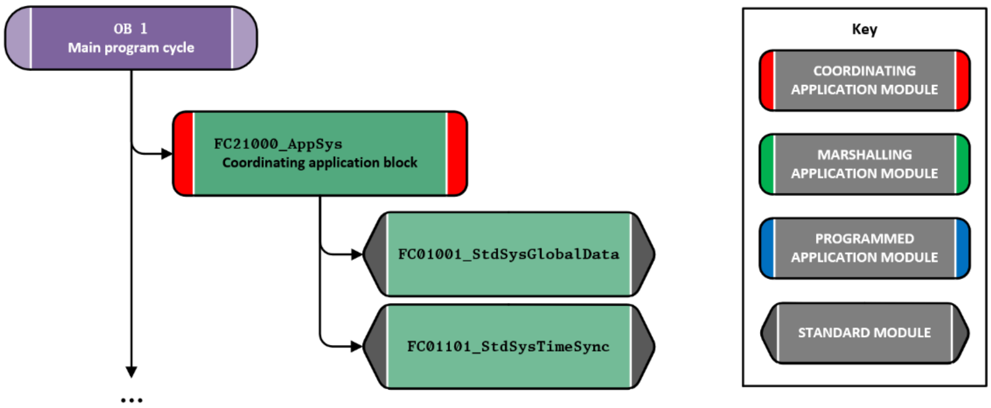


Figure 7.2 Coordinating applications calling standard modules

(4) Coordinating modules may contain simple signal conditioning programming instructions that are directly associated with the standard modules being called from within the module.

7.2 Marshalling application modules

(1) Figure 7.3 shows a coordinating application module calling two *marshalling* modules that subdivide the coordinating application modules into logical groupings within the functional group.

(2) In Figure 7.3 the coordinating application module (`FC22000_AppInst`) calls two *marshalling* application modules in turn, firstly: `FC22001_AppInstAnalogRead` and secondly `FC22501_AppInstDigitalRead`.

(3) Each of these marshalling blocks then call the standard modules associated with subdivision of the functional group, in this instance, `FC22001_AppInstAnalogRead` repeatedly calls the standard module `FC02001_StdInstAnalogRead` (called repeatedly, once for each analogue instrument, to scale and monitor each instrument).

(4) `FC22501_AppInstDigitalRead` repeatedly calls the standard module `FC02501_StdInstDigitalRead` (again, called repeatedly, once for each digital instrument, to filter and monitor each instrument).

(5) Marshalling modules may contain simple signal conditioning programming instructions that are directly associated with the standard modules being called from within the module.

(6) As many marshalling blocks as required can be used, marshalling blocks have the following restrictions:

- ① Marshalling blocks are always functions (FCs)
- ① The last three digits of the marshalling block number must not be `000`, this is reserved for coordinating application modules

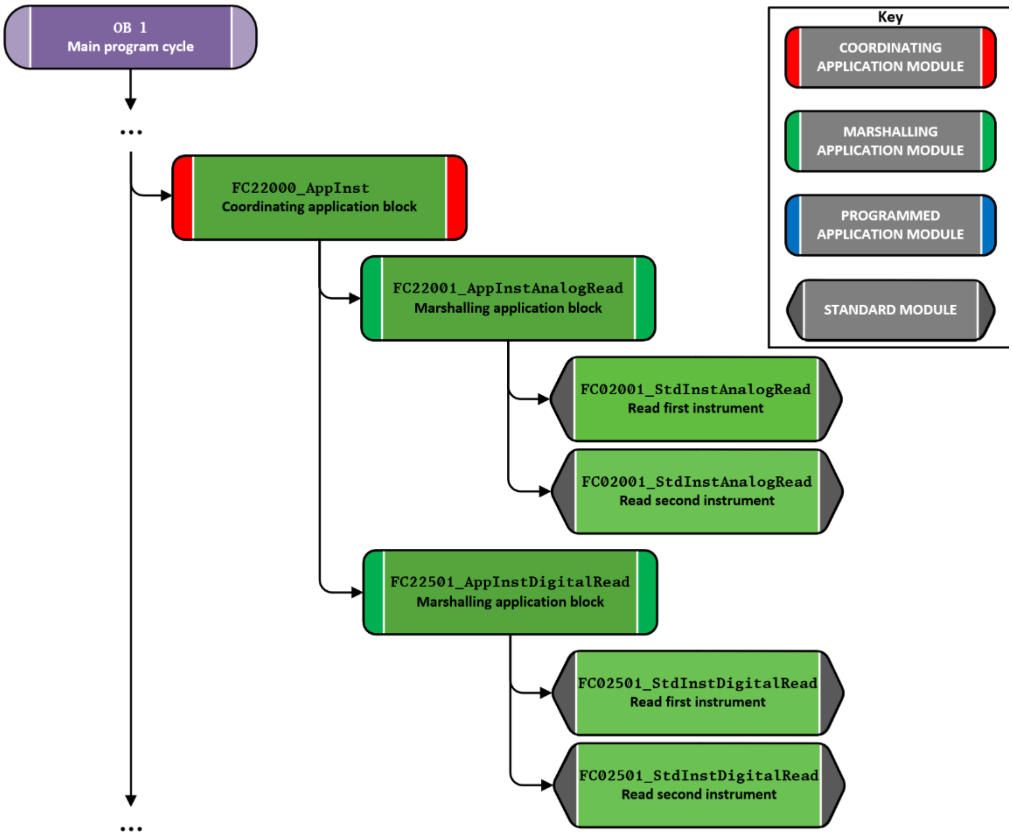


Figure 7.3 Coordinating applications, marshalling applications and standard modules

7.3 Programmed application modules

(1) There is a third type of application module: the *programmed* application module, these are shown in Figure 7.4:

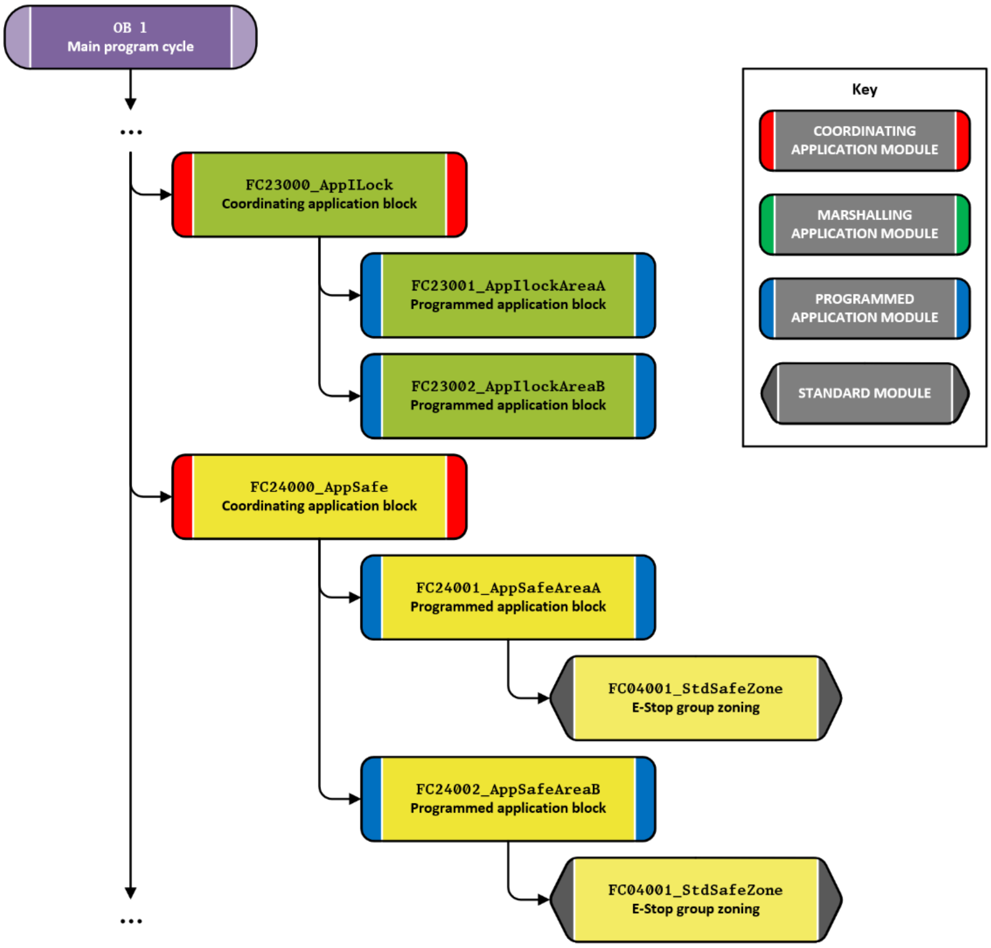


Figure 7.4 Programmed application modules

(2) Programmed application modules contain extensive programming statements, rather than the configuration exercises used when calling standard modules.

(3) A programmed application module contains software that is specific to the purpose of the Controller in question, such modules will contain logical statements (rather than

simply calling a standard module and providing it with parameters applicable to the device in question).

(4) For example, in Figure 7.4, the coordinating application module `FC23000_AppIlock` calls two programmed application modules (`FC23001_AppIlockAreaA` and `FC23002_AppIlockAreaB`), both these modules will contain project specific software that analyses the instrument readings (see Figure 7.3) and device states to determine what interlock conditions exist for devices in a particular plant area (Area A and Area B). The logic contained within these programmed application modules is entirely dependent on the requirements of the Controller application and it will be written entirely for that application, there will be no pre-determined format for the software (it will essentially be written from scratch).

(5) It is permissible for programmed applications to use standard modules (see `FC24001_AppSafeAreaA` of Figure 7.4) wherever required. The standard modules used must be either subroutine modules (see § 8.12) or be standard modules associated with the same functional group as the programmed application module.

(6) As many programmed blocks as required can be used, with the following restrictions:

① Programmed application blocks are always functions (FCs)

② The last three digits of the programmed block number must not be `000`, this is reserved for coordinating application modules

7.4 A summary of application module types

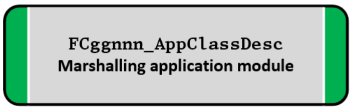
NAME AND SYMBOL	BLOCK	DESCRIPTION
Coordinating Application modules 	FC	Coordinating modules call either marshalling blocks or directly call standard modules Coordinating blocks are limited to signal conditioning and minor logic expressions associated with the standard modules
Marshalling Application modules 	FC	Marshalling modules directly call standard modules marshalling blocks are limited to signal conditioning and minor logic expressions associated with the standard modules
Programmed Application modules 	FC	Programmed modules contain extended programming instructions and may call standard modules if required

Table 7.1 Application module categories

BLANK PAGE

8 Standard module library

(1) The standard modules associated with the PAL software are broken down in to functional groups (see § 5.1), these are summarised below:

STANDARD MODULE NUMBER	FUNCTION GROUP
01ppp	System functions
02ppp	Read instruments
03ppp	Interlock & protection
04ppp	Safety systems
05ppp	Calculations & mathematics
07ppp	Sequential control
09ppp	Reserved
10ppp	Device drivers (Control loops)
11ppp	Device drivers (Valves)
12ppp	Device drivers (Drives)
16ppp	Message handling
17ppp	Communication handling
18ppp	(subroutines)
19ppp	Debug (end of cycle)

Table 8.1 Standard module functional groups

(2) The following sections list by function group, all the standard modules that are part of the PAL software.

(3) Each entry gives a brief overview of the purpose of the module and the functions available to it. The Software Design Specification *[Ref. 006]* contains further information and each standard module has its own Software module Design Specification (SMDS) *[Ref. 008]* that give full details of the module and all associated data structure (UDTs) and data blocks.

(4) Standard blocks are self-contained units of software, they do not use subroutines, they may however use the built-in system blocks. Certain standard modules are associated with or work in partnership with other standard modules (certain communication mechanisms require both a send and receive module and the sequence modules have more than one component).

8.1 System function modules

TITLE	Standard system global data	
BLOCK	FC 01001	*FC01001_StdSysGlobalData*
DESCRIPTION	Generates the internal logic and timing signals needed by all the other PAL software modules. The block identifies the first cycle after start-up, and determines the last, maximum and minimum cycle times. The block converts the Controller real time clock value to discrete integers, making the data globally available to all systems including non-Siemens equipment. **This block is a compulsory block within the PAL and must be called at the start of OB 1.**	
TITLE	Standard system time synchronisation (singe master server)	
BLOCK	FC 01101	*FC01101_StdSysMonoTimeSync*
DESCRIPTION	Updates the Controller real time clock with a single (master) server. Updates take place either daily or hourly (selectable) and can be set to automatically update if a (configurable) time difference exists between the server and the CPU.	
TITLE	Standard system time synchronisation (dual master/slave servers)	
BLOCK	FC 01102	*FC01102_StdSysDualTimeSync*
DESCRIPTION	Updates the CPU time with master/slave server pair. Updates take place either daily or hourly (selectable) and can be set to automatically update if a (configurable) time difference exists between the master server and the CPU. Should the master server fail, synchronisation will automatically switch to the slave server.	

8.2 Instrument read modules

TITLE	Standard analogue instrument read, scale and monitor
BLOCK	FC 02001 *FC02001_StdInstAnalogRead*
DESCRIPTION	This block reads and scales an analogue instrument signal received via an analogue input card. The resultant scaled value is a real (floating point) number that matches the calibrated range of the instrument in engineering units. The block has the facility to generate up to two alarms and two warnings whenever the signal is beyond a specific setpoint value (either above or below); the four signals are: ① Alarm high ② Warning high ③ Warning low ④ Alarm low All signals can be time filtered and have associated hysteresis. Generates out-of-range fault signals if the instruments is outside its normal calibrated range and also generate an optional *external fault* signal from a hardwired fault input from the instrument itself. The block offers simulation facilities to allow the operator to override the signal.
TITLE	Standard real value instrument read and monitor
BLOCK	FC 02011 *FC02011_StdInstRealValRead*
DESCRIPTION	This block reads an analogue instrument signal received as real (floating point) value (usually from a ProfiBus or Profinet enabled instrument). The received value can be rescaled by the block allowing the signal to be converted to different measurement units, the resultant value is a real (floating point) number. The block has the facility to generate up to two alarms and two warnings whenever the signal is beyond a specific setpoint value (either above or below); the four signals are: ① Alarm high ② Warning high ③ Warning low ④ Alarm low All signals can be time filtered and have associated hysteresis. Generates out-of-range fault signals if the instruments is outside its normal calibrated range and also generate an optional *external fault* signal from a hardwired fault input from the instrument itself. The block offers simulation facilities to allow the operator to override the signal.

TITLE	Standard instrument threshold detection
BLOCK	FC 02101 *FC02101_StdInstRealLimit*
DESCRIPTION	The block has the facility to generate a limit (threshold) signal whenever the signal is beyond a specific setpoint value (configurable to be either above or below); the signal is: Limit active The limit signal can be time filtered and has associated hysteresis.
TITLE	Standard digital instrument read and monitor
BLOCK	FC 02501 *FC02501_StdInstDigitalRead*
DESCRIPTION	Reads and controls the operation of a digital instrument signal, the instrument can be active high or low (configurable) and time filtering is provided on both the activation edge (signal must be active for a specified time) and on the deactivation edge (signal remains active for a specified time). The block offers simulation facilities to allow the operator to override the signal.
TITLE	Standard digital filter
BLOCK	FC 02601 *FC02601_StdInstDigitalFilt*
DESCRIPTION	Provides digital signal filtering for any digital signal, the signal can be active high or low (configurable) and time filtering is provided on both the activation edge (signal must be active for a specified time) and on the deactivation edge (signal remains active for a specified time).

8.3 Interlock and protection modules

(1) Interlock handling modules provided the following types of interlock:

① **Interlock**: a simple interlock that is active whenever a set of conditions is true, it will force any associated devices to a safe state

② **Permissive**: takes no action if a device is in a non-safe state, but once the device is in a safe state will prevent a transition to a non-safe state (i.e. will not force a valve to close, but once it is closed, will prevent it from re-opening)

③ **Trip**: a latching interlock, it activates whenever a set of events are true (like an interlock), but will not deactivate until the triggering conditions are removed and a reset signal has been given (effectively a latching interlock), it will force any associated devices to a safe state

(2) The modules here are effectively configurable AND or OR gate structures that can combine either 2, 4 or 8 discrete signals into a single interlock, permissive or trip condition.

(3) The modules are used in place of the standard AND or OR logic instruction available to the Controller and provide individual indication for supervisor systems to highlight the active path or paths through the modules.

(4) The permissive modules also monitor the state of the affected device to determine whether the device is currently in a permitted non-safe state &c.

(5) The trip modules are latching modules that need a reset signal to remove the interlock once the triggering conditions have cleared.

TITLE	Standard interlock 2 signal interlock with status reporting
BLOCK	FC 03002 *FC03002_StdILock02*
DESCRIPTION	This block monitors up to two discrete signals to determine if an interlock condition exists. The block is configurable as OR (interlock active if any signal is active) or AND (interlock active when all signals are active). The interlock condition is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any interlock, the block can be combined with other blocks in this series to produce more complex interlock arrangements.
TITLE	Standard interlock 4 signal interlock with status reporting
BLOCK	FC 03004 *FC03004_StdILock04*
DESCRIPTION	This block monitors up to four discrete signals to determine if an interlock condition exists. The block is configurable as OR (interlock active if any signal is active) or AND (interlock active when all signals are active). The interlock condition is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any interlock, the block can be combined with other blocks in this series to produce more complex interlock arrangements.
TITLE	Standard interlock 8 signal interlock with status reporting
BLOCK	FC 03008 *FC03008_StdILock08*
DESCRIPTION	This block monitors up to eight discrete signals to determine if an interlock condition exists. The block is configurable as OR (interlock active if any signal is active) or AND (interlock active when all signals are active). The interlock condition is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any interlock, the block can be combined with other blocks in this series to produce more complex interlock arrangements.

TITLE	Standard interlock 2 signal permissive interlock with status reporting
BLOCK	FC 03102 *FC03102_StdILockPerm02*
DESCRIPTION	This block monitors up to two discrete signals to determine if a permissive interlock condition exists. The block also monitors the affected device to determine whether the device is currently in a permitted non-safe state. The block is configurable as OR (permissive active if any signal is active) or AND (permissive active when all signals are active). The permissive condition is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any permissive, the block can be combined with other blocks in this series to produce more complex interlock arrangements.
TITLE	Standard interlock 4 signal permissive interlock with status reporting
BLOCK	FC 03104 *FC03104_StdILockPerm04*
DESCRIPTION	This block monitors up to four discrete signals to determine if a permissive interlock condition exists. The block also monitors the affected device to determine whether the device is currently in a permitted non-safe state. The block is configurable as OR (permissive active if any signal is active) or AND (permissive active when all signals are active). The permissive condition is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any permissive, the block can be combined with other blocks in this series to produce more complex interlock arrangements.
TITLE	Standard interlock 8 signal permissive interlock with status reporting
BLOCK	FC 03108 *FC03108_StdILockPerm08*
DESCRIPTION	This block monitors up to eight discrete signals to determine if a permissive interlock condition exists. The block also monitors the affected device to determine whether the device is currently in a permitted non-safe state. The block is configurable as OR (permissive active if any signal is active) or AND (permissive active when all signals are active). The permissive condition is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any permissive, the block can be combined with other blocks in this series to produce more complex interlock arrangements.

TITLE	Standard interlock 2 signal trip interlock with status reporting
BLOCK	FC 03202 *FC03202_StdILockTrip02*
DESCRIPTION	This block monitors up to two discrete signals to determine if a trip interlock condition exists. The block is configurable as OR (trip active if any signal is active) or AND (trip active when all signals are active). The trip condition is not automatically deactivated when triggering conditions are no longer present, a reset signal must be supplied to actively clear the interlock once the activation conditions are removed. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any trip, the block can be combined with other blocks in this series to produce more complex interlock arrangements.
TITLE	Standard interlock 4 signal trip interlock with status reporting
BLOCK	FC 03204 *FC03204_StdILockTrip04*
DESCRIPTION	This block monitors up to four discrete signals to determine if a trip interlock condition exists. The block is configurable as OR (trip active if any signal is active) or AND (trip active when all signals are active). The trip condition is not automatically deactivated when triggering conditions are no longer present, a reset signal must be supplied to actively clear the interlock once the activation conditions are removed. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any trip, the block can be combined with other blocks in this series to produce more complex interlock arrangements.
TITLE	Standard interlock 8 signal trip interlock with status reporting
BLOCK	FC 03208 *FC03208_StdILockTrip08*
DESCRIPTION	This block monitors up to eight discrete signals to determine if a trip interlock condition exists. The block is configurable as OR (trip active if any signal is active) or AND (trip active when all signals are active). The trip condition is not automatically deactivated when triggering conditions are no longer present, a reset signal must be supplied to actively clear the interlock once the activation conditions are removed. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any trip, the block can be combined with other blocks in this series to produce more complex interlock arrangements.
TITLE	Standard interlock message signal generation
BLOCK	FC 03501 *FC03501_StdILockMsgGen*
DESCRIPTION	This block generates a specific message when linked to any of the previous interlock modules. The message can be configured as an alarm, a warning or an event.

8.4 Safety and safety system modules

(1) Safety modules group various emergency stop signals into zones that, if active, remove power from specific devices.

(2) The safety systems operate at a hardwired level (the power is physically removed from the devices, rather than by any software within the Controller).

(3) The purpose of the safety system modules is to ensure that the state of an affected device will match the hardwired state of the device (for example, if the system requires a drive to run for normal process reasons, but the safety system has physically removed power from the drive, the safety system module detects this and stops the drive within the software, following the true state imposed upon the drive).

(4) The safety modules provided the following types of zone control:

① **E-stop group**: a simple group that is active whenever an emergency stop signal is detected within the group, it will force any associated devices to a safe state

② **E-stop trip group**: a latching group, it activates whenever an emergency stop signal is detected within the group, but will not deactivate until the triggering conditions are removed and a reset signal has been given, it will force any associated devices to a safe state

(5) The modules here are always OR gate structures that can combine either 2, 4 or 8 discrete signals into a single emergency stop group.

(6) The modules provide individual indication for supervisor systems to highlight the active path or paths through the e-stop groupings.

TITLE	Standard safety 2 signal E-stop zone group with status reporting
BLOCK	FC 04002 *FC04002_StdSafeZoneNorm02*
DESCRIPTION	This block monitors up to two discrete signals to determine if an emergency stop condition exists. Activation of either signal will cause the group emergency stop to activate The group is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any E-stop signal, the block can be combined with other blocks in this series to produce more complex group arrangements.
TITLE	Standard safety 4 signal E-stop zone group with status reporting
BLOCK	FC 04004 *FC04004_StdSafeZoneNorm04*
DESCRIPTION	This block monitors up to four discrete signals to determine if an emergency stop condition exists. Activation of any signal will cause the group emergency stop to activate The group is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any E-stop signal, the block can be combined with other blocks in this series to produce more complex group arrangements.
TITLE	Standard safety 8 signal E-stop zone group with status reporting
BLOCK	FC 04008 *FC04008_StdSafeZoneNorm08*
DESCRIPTION	This block monitors up to eight discrete signals to determine if an emergency stop condition exists. Activation of any signal will cause the group emergency stop to activate The group is automatically deactivated when triggering conditions are no longer present. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any E-stop signal, the block can be combined with other blocks in this series to produce more complex group arrangements.

TITLE	Standard safety 2 signal E-stop latching zone group with status reporting
BLOCK	FC 04202 *FC04202_StdSafeZoneTrip02*
DESCRIPTION	This block monitors up to two discrete signals to determine if an emergency stop condition exists. Activation of either signal will cause the group emergency stop to activate The group is not automatically deactivated when the triggering conditions are no longer present, a reset signal must be supplied to actively reset the group once the activation conditions are removed. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any E-stop signal, the block can be combined with other blocks in this series to produce more complex group arrangements.
TITLE	Standard safety 4 signal E-stop latching zone group with status reporting
BLOCK	FC 04204 *FC04202_StdSafeZoneTrip04*
DESCRIPTION	This block monitors up to four discrete signals to determine if an emergency stop condition exists. Activation of any signal will cause the group emergency stop to activate The group is not automatically deactivated when the triggering conditions are no longer present, a reset signal must be supplied to actively reset the group once the activation conditions are removed. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any E-stop signal, the block can be combined with other blocks in this series to produce more complex group arrangements.
TITLE	Standard safety 8 signal E-stop latching zone group with status reporting
BLOCK	FC 04208 *FC04208_StdSafeZoneTrip08*
DESCRIPTION	This block monitors up to eight discrete signals to determine if an emergency stop condition exists. Activation of any signal will cause the group emergency stop to activate The group is not automatically deactivated when the triggering conditions are no longer present, a reset signal must be supplied to actively reset the group once the activation conditions are removed. Common format status signals are provided to allow supervisory system to determine and display the state and cause of any E-stop signal, the block can be combined with other blocks in this series to produce more complex group arrangements.
TITLE	Standard safety message signal generation
BLOCK	FC 04501 *FC04501_StdSafeMsgGen*
DESCRIPTION	This block generates a specific message when linked to any of the previous E-stop zone modules. The message can be configured as an alarm, a warning or an event.

8.5 Calculations and mathematics modules

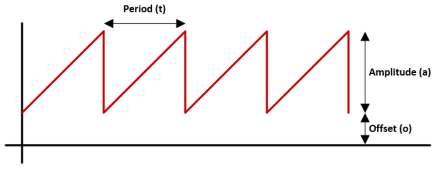
TITLE	Standard calculation — simple average	
BLOCK	FC 05001	*FC05001_StdCalcAvg*
DESCRIPTION	Calculates the average value of a set of n real numbers stored within a data block. The set can be of any size up to the maximum capacity of a data block. $$Avg = \frac{x_1 + x_2 + \cdots + x_n}{n}$$	
TITLE	Standard calculation — rolling average	
BLOCK	FC 05002	*FC05002_StdCalcAvgRolling*
DESCRIPTION	Calculates an unweighted rolling average of a specified number of real samples (n). The samples will be taken at specified intervals and stored in a data block, when more than n samples have been taken, the oldest sample will be removed from the bottom of the list and a new sample added at the top. The average (mean) is calculated for the current number of samples in the list $$RollingAvg = \frac{x_{curr} + x_{curr-1} + \cdots + x_{curr-(n-1)}}{n}$$	
TITLE	Standard calculation — cumulative average	
BLOCK	FC 05003	*FC05003_StdCalcAvgCumulate*
DESCRIPTION	Calculates an unweighted cumulative average of a continuing stream of real values. $$CumulativAvg_n = \frac{x_1 + x_2 + \cdots + x_n}{n}$$ $$CumulativAvg_{n+1} = \frac{x_{n+1} + n(CumulativAvg_n)}{n+1}$$ The cumulative average can be restarted by triggering a reset signal.	
TITLE	Standard calculation — weighted rolling average	
BLOCK	FC 05004	*FC05004_StdCalcAvgWeighted*
DESCRIPTION	Calculates a weighted rolling average of a specified number of real samples (n). The samples will be taken at specified intervals and stored in a data block, when more than n samples have been taken, the oldest sample will be removed from the bottom of the list and a new sample added at the top. The weighted average gives more emphasis to the most recent samples in the list $$WeightedAvg = \frac{nx_n + (n-1)x_{n-1} + \cdots + 2x_2 + x_1}{n + (n-1) + \cdots + 2 + 1}$$	

TITLE	Standard calculation — exponential rolling average
BLOCK	FC 05005 *FC05005_StdCalcAvgExp*
DESCRIPTION	Calculates an exponential moving average of a specified number of real samples (n). The samples will be taken at specified intervals and stored in a data block, when more than n samples have been taken, the oldest sample will be removed from the bottom of the list and a new sample added at the top. The exponential moving average will return the same value as the rolling average until n samples have been taken, after this point the $n + 1$ sample will give a true exponential moving average. The exponential rolling average is calculated as: $ExponentialAvg_{n+1} = \frac{2}{n+1}(x_{n+1} - ExponentialAvg_n) + ExponentialAvg_n$ Where: $\frac{2}{n+1}$ is the standard smoothing coefficient.
TITLE	Standard calculation — rate-of-change
BLOCK	FC 05101 *FC05101_StdCalcDiffRoC*
DESCRIPTION	Calculates the rate-of-change of a value over a given time period: $RateOfChange_n = \frac{x_n - x_{n-1}}{t_n - t_{n-1}} = \frac{dx_n}{dt}$
TITLE	Standard calculation — average rate-of-change
BLOCK	FC 05102 *FC05102_StdCalcDiffRoCAvg*
DESCRIPTION	Calculates an unweighted rolling average of a specified number of real samples (n) of a rate-of-change value. The samples will be taken at specified intervals and the calculated rate-of-change between values will be stored in a data block, when more than n samples have been taken, the oldest sample will be removed from the bottom of the list and a new sample added at the top. The average (mean) is calculated for the current number of rate-of-change samples in the list (the rate of change calculation is given in the previous module): $RateOfChange_{Avg} = \frac{\frac{dx_n}{dt} + \frac{dx_{n-1}}{dt} + \cdots + \frac{dx_1}{dt}}{n}$

TITLE	Standard calculation — signal integration (area)	
BLOCK	FC 05201	*FC05201_StdCalcIntArea*
DESCRIPTION	Continuously integrates a signal $x(t)$ relative to time: $$Integral_n = \int_0^{nt} x(t)dt$$ The integration uses piecewise linear intervals to calculate the current integral value, if the time between samples is t, the integral value after n samples is: $$Integral_n = t\left(\frac{x_n + x_{n-1}}{2}\right) + t\left(\frac{x_{n-1} + x_{n-2}}{2}\right) + \cdots + t\left(\frac{x_1 + x_0}{2}\right)$$ The cumulative integral value for the next sample is thus: $$Integral_{n+1} = t\left(\frac{x_{n+1} + x_n}{2}\right) + Integral_n$$ The integral value can be restarted by triggering a reset signal.	
TITLE	Standard calculation — convert a ranged value to a percentage	
BLOCK	FC 05301	*FC05301_StdCalcValToPercent*
DESCRIPTION	Converts a real value (x) in the range x_{min} to x_{max} to a percentage value, using the following formulae: $$Percentage = 100\left(\frac{x - x_{min}}{x_{max} - x_{min}}\right)$$	
TITLE	Standard calculation — convert a percentage to a ranged value	
BLOCK	FC 05302	*FC05302_StdCalcPercentToVal*
DESCRIPTION	Converts a percentage value (p) to a real value (x), x is in the range x_{min} to x_{max}, using the following formulae: $$x = \frac{p}{100}(x_{max} - x_{min}) + x_{min}$$	
TITLE	Standard calculation — convert a percentage to a variable mark/space square wave	
BLOCK	FC 05351	*FC05351_StdCalcPercentToPulse*
DESCRIPTION	Converts a percentage value (p) to a square wave pulse train with a variable mark/space ratio, the period of the square wave is a specified value (t). The mark/space ratio is determined by the percentage value (p), a value of 33.3% would give a mark time of $\frac{t}{3}$ and a space time of $\frac{2t}{3}$; the mark/space time calculations are given by: $$mark_{tim} = \frac{p}{100}t$$ $$space_{tim} = t\left(1 - \frac{p}{100}\right)$$	

TITLE	Standard calculation — convert a percentage to a variable mark/space square wave
BLOCK	FC 05352 *FC05352_StdCalcPulseToPercent*
DESCRIPTION	Converts the mark/space ratio of a square wave pulse train to percentage value (*p*) The period of the square wave (*t*) is automatically determined by the module. The percentage value (*p*), is calculated as: $$P = 100\frac{mark_{tim}}{t}$$ The percentage value is calculated at on the rising edge of the square wave (i.e. recalculated after each square wave period and is adjusted for variations in the square wave period)
TITLE	Standard calculation — convert a pulse train to an ON/OFF state
BLOCK	FC 05361 *FC05361_StdCalcPulseToState*
DESCRIPTION	If a square wave pulse train with a period shorter than a specified time (*t*) is present, the block returns a TRUE. If the pulse train period is longer than the time (*t*), the block returns a FALSE state; the mark/space ratio of the signal is not relevant. The block is typically use to detect rotation of a device and ensure it is above a particular frequency.
TITLE	Standard calculation — convert an ON/OFF state to a pulse train
BLOCK	FC 05362 *FC05362_StdCalcStateToPulse*
DESCRIPTION	If the monitored signal is TRUE, a square wave pulse train with a period (*t*) is generated by the module. If the monitored signal is FALSE, the square wave is stopped (set to zero).
TITLE	Standard calculation — convert a square wave pulse train to a frequency
BLOCK	FC 05363 *FC05363_StdCalcPulseToFreq*
DESCRIPTION	Monitors a square wave pulse train and converts its period to a frequency value in both Hertz and revolutions per minute The period of the square wave (*t*) in seconds is automatically determined by the module. The frequency in Hz (f) is calculated as: $$f = \frac{1}{t}$$ The revolutions per minute (RPM) is calculated as: $$RPM = \frac{60}{t}$$

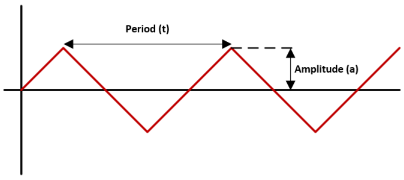
TITLE	Standard calculation — pulse generator 2 (dual) state
BLOCK	FC 05502 *FC05502_StdCalcPulseDual*
DESCRIPTION	Produces two repeating pulses of variable length. Each ON state will be active for a given time period (this can be set to zero). The sequence of Time1/State1, Time2/State2 will repeat continuously while the enable signal is active, if the enable signal is reset, all pulse states are set to zero. A pause signal will pause all timing functions and hold the signals in their last state until the pause signal is released An integer signal as well as discrete digital signals are provided to indicate the current state
TITLE	Standard calculation — pulse generator 3 (tri) state
BLOCK	FC 05503 *FC05503_StdCalcPulseTri*
DESCRIPTION	Produces three repeating pulses of variable length. Each ON state will be active for a given time period (this can be set to zero). The sequence of Time1/State1, Time2/State2, Time3/State3 will repeat continuously while the enable signal is active, if the enable signal is reset, all pulse states are set to zero. A pause signal will pause all timing functions and hold the signals in their last state until the pause signal is released An integer signal as well as discrete digital signals are provided to indicate the current state
TITLE	Standard calculation — pulse generator 4 (quad) state
BLOCK	FC 05504 *FC05504_StdCalcPulseQuad*
DESCRIPTION	Produces four repeating pulses of variable length. Each ON state will be active for a given time period (this can be set to zero). The sequence of Time1/State1, Time2/State2 … Time4/State4 will repeat continuously while the enable signal is active, if the enable signal is reset, all pulse states are set to zero. A pause signal will pause all timing functions and hold the signals in their last state until the pause signal is released An integer signal as well as discrete digital signals are provided to indicate the current state

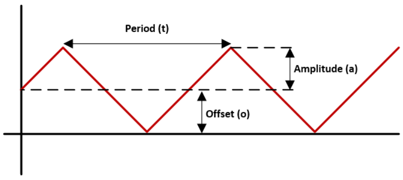
TITLE	Standard calculation — pulse generator 8 (octa) state
BLOCK	FC 05508 *FC05508_StdCalcPulseOcta*
DESCRIPTION	Produces eight repeating pulses of variable length. Each ON state will be active for a given time period (this can be set to zero). The sequence of Time1/State1, Time2/State2 … Time8/State8 will repeat continuously while the enable signal is active, if the enable signal is reset, all pulse states are set to zero. A pause signal will pause all timing functions and hold the signals in their last state until the pause signal is released An integer signal as well as discrete digital signals are provided to indicate the current state
TITLE	Standard calculation — pulse generator 16 (hexa) state
BLOCK	FC 05516 *FC05516_StdCalcPulseHexa*
DESCRIPTION	Produces 16 repeating pulses of variable length. Each ON state will be active for a given time period (this can be set to zero). The sequence of Time1/State1, Time2/State2 … Time16/State16 will repeat continuously while the enable signal is active, if the enable signal is reset, all pulse states are set to zero. A pause signal will pause all timing functions and hold the signals in their last state until the pause signal is released An integer signal as well as discrete digital signals are provided to indicate the current state
TITLE	Standard calculation — waveform generator ramp function
BLOCK	FC 05601 *FC05601_StdCalcWaveRamp*
DESCRIPTION	Generates a single ramp waveform moving from start value to an end value over a specified time period. Triggering the function will cause a single ramp waveform to be produced, at the end of which the module output will remain at the end value until reset or re-triggerd
TITLE	Standard calculation — waveform generator continuous sawtooth wave function
BLOCK	FC 05602 *FC05602_StdCalcWaveSaw*
DESCRIPTION	Generates a continuous sawtooth waveform. The amplitude (*a*) of the waveform can be specified, as can the period (*t*) and the offset (*o*) of the waveform: 

TITLE Standard calculation — waveform generator continuous triangular wave function

BLOCK FC 05603 *FC05603_StdCalcWaveTri*

DESCRIPTION Generates a continuous triangle waveform. The amplitude (*a*) of the waveform can be specified, as can the period (*t*) and the offset (*o*) of the waveform:

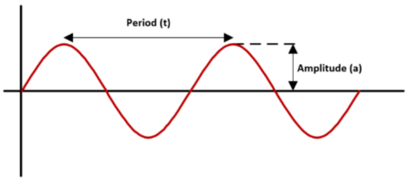


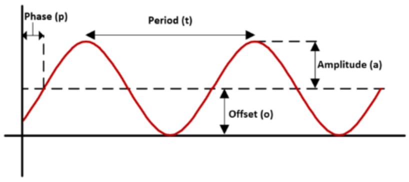


TITLE Standard calculation — waveform generator continuous sine wave function

BLOCK FC 05604 *FC05604_StdCalcWaveSin*

DESCRIPTION Generates a continuous sine waveform. The amplitude (*a*) of the waveform can be specified, as can the period (*t*) and the offset (*o*) of the waveform. The phase of the wave (*p*) can also be adjusted:

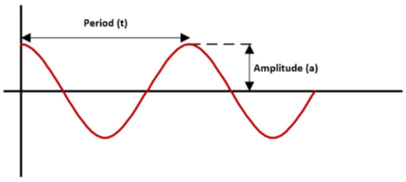


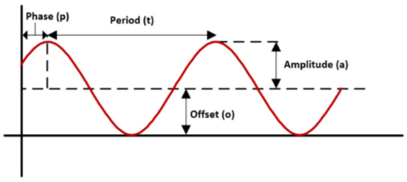


TITLE Standard calculation — waveform generator continuous cosine wave function

BLOCK FC 05605 *FC05605_StdCalcWaveCos*

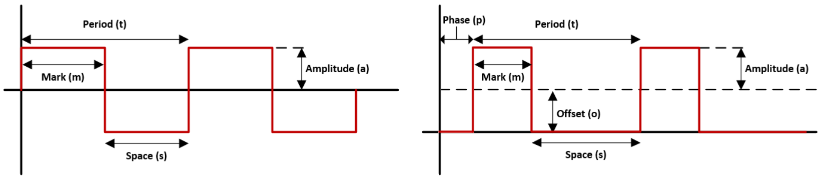
DESCRIPTION Generates a continuous cosine waveform. The amplitude (*a*) of the waveform can be specified, as can the period (*t*) and the offset (*o*) of the waveform. The phase of the wave (*p*) can also be adjusted:





The cosine waveform is identical to the sine wave form with a phase offset of 90°.

TITLE	Standard calculation — waveform generator continuous square wave function
BLOCK	FC 05606 *FC05605_StdCalcWaveSquare*
DESCRIPTION	Generates a continuous square waveform. The amplitude (*a*) of the waveform can be specified, as can the period (*t*) and the offset (*o*) of the waveform. The mark (m)/space (s) ratio can be adjusted and the onset can be delayed by a phase time (*p*):



8.6 Sequential control

(1) Sequential control has its own section (section 9) that covers in detail the modules listed here, the following is a summary of the standard sequence modules available within the PAL.

TITLE	Standard sequence — IEC compliant sequence manager (controller)
BLOCK	FC 07001 *FC07001_StdSeqIEC_Control*
DESCRIPTION	Sequence management module, it ensures that a sequence progresses correctly through the operating state logic applied to it (see § 9). Each sequence has a single FC07001 module associated with it; this manages all the commands that can be issued to the sequence, performs error checking within the sequence and identifies the current state of the sequence This module is IEC compliant (see § 9.3.1).
TITLE	Standard sequence — IEC compliant sequence operating state logic (OSL)
BLOCK	FC 07011 *FC07011_StdSeqIEC_OSL*
DESCRIPTION	Identifies the current operating state of a given sequence (see § 9). The operating state is the determined by the operating state logic diagram and is identified by the numeric range of the current sequence step (see § 9.1). This module is IEC compliant (see § 9.3.1).
TITLE	Standard sequence — IEC compliant sequence step/transition manager
BLOCK	FC 07021 *FC07021_StdSeqIEC_Step*
DESCRIPTION	Controls the *phased* operation of a single step within a sequence, each step has its own instance of this module. The module handles the transition from one step to another (up to eight different transitions are possible) and handles the three phases within a step: • Initialising • Processing • Terminating The module manages step delay timers (specifying the minimum time within a step) and step duration timers (measures how long the step has been active) This module is IEC compliant, in that the terminating phase of the current step overlaps the initialising phase of the next step (see § 9.3.1).

(2) The following modules are non-IEC compliant version of the previous modules, the URS *[Ref. 003]* requires IEC compliant modules and these are provided above.

(3) The non-compliant versions below are provided to observe the more common practices used widely within the PLC programming community. Section 9.3.2 explains this distinction in more detail.

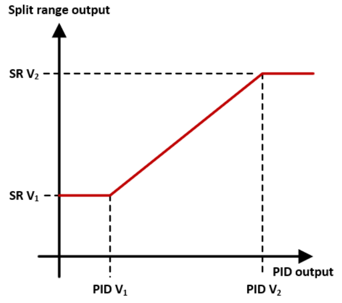
TITLE	Standard sequence — non-IEC compliant sequence manager (controller)
BLOCK	FC 07501 *FC07501_StdSeqNonIEC_Control*
DESCRIPTION	Sequence management module, it ensures that a sequence progresses correctly through the operating state logic applied to it (see § 9.1). Each sequence has a single FC07001 module associated with it; this manages all the commands that can be issued to the sequence, performs error checking within the sequence and identifies the current state of the sequence This module is **NOT** IEC compliant (see § 9.3.2).
TITLE	Standard sequence — non-IEC compliant sequence operating state logic (OSL)
BLOCK	FC 07511 *FC07511_ StdSeqNonIEC _OSL*
DESCRIPTION	Identifies the current operating state of a given sequence (see § 9). The operating state is the determined by the operating state logic diagram and is identified by the numeric range of the current sequence step (see § 9.1). This module is **NOT** IEC compliant (see § 9.3.2).
TITLE	Standard sequence — non-IEC compliant sequence step/transition manager
BLOCK	FC 07521 *FC07521_ StdSeqNonIEC _Step*
DESCRIPTION	Controls the *phased* operation of a single step within a sequence, each step has its own instance of this module. The module handles the transition from one step to another (up to eight different transitions are possible) and handles the three phases within a step: • Initialising • Processing • Terminating The module manages step delay timers (specifying the minimum time within a step) and step duration timers (measures how long the step has been active) This module is **NOT** IEC compliant, in that the terminating phase of the current step occurs before the initialising phase of the next step, the two are not coincident (see § 9.3.2).

8.7 Device drivers — control loops

(1) Device drivers are split into multiple sections: control loops, valves and drives. This section is exclusively associated with control loops.

TITLE	Standard device driver — control loops — standard PID loop
BLOCK	FC 10001 *FC10001_StdDevPID_Standard*
DESCRIPTION	Implements a standard three term (PID) controller. The module has three operating modes: • Off (loop is disabled, the output is zero or minimum value) • Setpoint (the output is automatically adjusted to maintain a process variable at the specified setpoint) • Fixed Output (the PID loop maintains a fixed output) The loop can be switched between manual and automatic, in manual all data is provided by the operator. In automatic, all data is provided by the Controller programme. Switching between modes and between automatic and manual is *bumpless*. The block supports the use of interlock signals, these will set the PID output to a particular value
TITLE	Standard device driver — control loops — standard PID loop with gain scheduling
BLOCK	FC 10011 *FC10011_StdDevPID_Sched*
DESCRIPTION	Implements a standard three term (PID) controller with gain scheduling. The module allows for all three PID terms to be changed as the process moves through different phases, the PID terms applied are dependent on the PID loop error signal (the difference between the process value and the setpoint), up to 10 different sets of PID terms can applied to different error signal ranges. The module has three operating modes: • Off (loop is disabled, the output is zero or minimum value) • Setpoint (the output is automatically adjusted to maintain a process variable at the specified setpoint) • Fixed Output (the PID loop maintains a fixed output) The loop can be switched between manual and automatic, in manual all data is provided by the operator. In automatic, all data is provided by the Controller programme. Switching between modes and between automatic and manual is *bumpless*. The block supports the use of interlock signals, these will set the PID output to a particular value

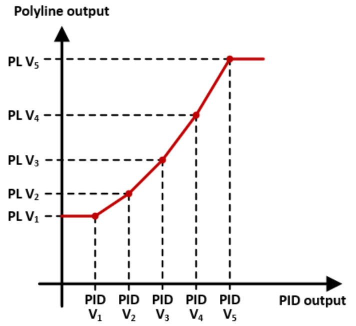
TITLE	Standard device driver — control loops — split range modifier
BLOCK	FC 10021 *FC10021_StdDevPID_Split*
DESCRIPTION	The split range module accepts a PID loop output signal and converts it into a separately scaled signal that can be applied to a particular actuator



This module allows the output of a PID loop to be split into multiple sub-ranges and each sub-range can be applied to a separate actuator. The sub-ranges can be mutually exclusive or can overlap to meet the requirements of the process.

The loop can be switched between manual and automatic, in manual all data is provided by the operator. In automatic, all data is provided by the Controller programme.

TITLE	Standard device driver — control loops — polyline modifier
BLOCK	FC 10022 *FC10022_StdDevPID_Poly*
DESCRIPTION	The polyline module accepts a PID loop output signal and converts it into a piecewise linear polyline:



The polyline has a minimum of two points and a maximum of 100 points.

The loop can be switched between manual and automatic, in manual all data is provided by the operator. In automatic, all data is provided by the Controller programme.

TITLE	Standard device driver — control loops — external (stand-alone) controller
BLOCK	FC 10022 *FC10101_StdDevPID_External*
DESCRIPTION	Provides an interface to an external (stand-alone) PID controller. The module supports automatic and manual modes. The module monitors the external module for hardwired faults and for failure to achieve setpoint within a specified period.

TITLE	Standard device driver — control loops — look-up table
BLOCK	FC 10501 *FC10501_StdDevPID_LookUp*
DESCRIPTION	Provides a two-dimensional look-up table, that monitors two discrete values (X and Y) and depending on the relative values, returns a third output value (OUT) from the look-up table:

2D table	X_{00}	X_{01}	X_{02}	...	X_{99}
Y_{00}	$OUT_{0,0}$	$OUT_{1,0}$	$OUT_{2,0}$	...	$OUT_{99,0}$
Y_{01}	$OUT_{0,1}$	$OUT_{1,1}$	$OUT_{2,1}$	...	$OUT_{99,1}$
Y_{02}	$OUT_{0,2}$	$OUT_{1,2}$	$OUT_{2,2}$	...	$OUT_{99,2}$
...	...	...	...	...	...
Y_{99}	$OUT_{0,99}$	$OUT_{1,99}$	$OUT_{2,99}$	...	$OUT_{99,99}$

The table can accommodate a 100 × 100 grid

The loop can be switched between manual and automatic, in manual all data is provided by the operator. In automatic, all data is provided by the Controller programme.

8.8 Device drivers — Valves

TITLE	Standard device driver — valves — isolating valve
BLOCK	FC 11001 FC11001_StdDevValveIsol
DESCRIPTION	This module controls the operation of either a normally closed or normally open isolating valve configured with either open, closed, both open and closed or no position feedback. The module supports automatic and manual control and can be configured with simulation mode to allow the valve limit switch signals to be overwritten and set to follow the demand output. The module can be configured for normally closed (energise to open) or normally open (energise to close) valves. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to Open ② Failed to Close ③ Failed while Open ④ Failed while Closed ⑤ External Fault Separate operation times for opening and closing can be defined. The valve module supports all forms of interlock, permissive and trip signals, and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Closed • Closing • Open • Opening • Fault The module also generates status signals for the selected operating modes and conditions.

TITLE	Standard device driver — valves — 3-way valve
BLOCK	FC 11011 *FC11011_StdDevValve3Way*
DESCRIPTION	This module controls the operation of 3-way valve with a common open port (the action of the valve switches the common port to one of the other two ports) configured with either position feedback or no position feedback. The module supports automatic and manual control and can be configured with simulation mode to allow the valve limit switch signals to be overwritten and set to follow the demand output. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to Energise ② Failed to De-energise ③ Failed while Energised ④ Failed while De-energised ⑤ External Fault Separate operation times for energising and de-energising can be defined. The valve module supports all forms of interlock, permissive and trip signals, and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • De-energised port open • Deenergising • Energised port open • Energising • Fault The module also generates status signals for the selected operating modes and conditions.

TITLE	Standard device driver — valves — bistable isolating valve
BLOCK	FC 11101 *FC11101_StdDevValveBi*
DESCRIPTION	This module controls the operation of a bistable isolating valve configured with either open, closed, both open and closed or no position feedback. The module supports automatic and manual control and can be configured with simulation mode to allow the valve limit switch signals to be overwritten and set to follow the demand position. The module can be configured to either maintain the output when the valve reaches the demanded position, or de-energise the outputs when position is reached. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to Open ② Failed to Close ③ Failed while Open ④ Failed while Closed ⑤ External Fault Separate operation times for opening and closing can be defined. The valve module supports all forms of interlock, permissive and trip signals, and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Closed • Closing • Open • Opening • Fault The module also generates status signals for the selected operating modes and conditions.

TITLE	Standard device driver — valves — modulating valve
BLOCK	FC 11501 *FC11501_StdDevValveMod*
DESCRIPTION	This module controls the operation of either a positive acting (opens with increasing signal) or negative acting (closes with increasing signal) modulating valve optionally configured with an analogue position confirmation and, additionally, with either none, open, closed or both open and closed limit switch position feedback. The module supports automatic and manual control and can be configured with simulation mode to allow the analogue position feedback and valve limit switch signals to be overwritten and set to follow the demand position. The valve can be configured as positive acting (0% output = fully closed, 100% output = fully opened) or negative acting (0% output = fully opened, 100% output = fully closed) The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to achieve demanded position ② External Fault Separate operation times for opening and closing can be defined. The valve module supports all forms of interlock, permissive and trip signals, and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Closed • Open (or partially open) • Fault • Actual/demanded position The module also generates status signals for the selected operating modes and conditions.

8.9 Device drivers — Drives

TITLE	Standard device driver — drives — direct online
BLOCK	FC 12001 *FC12001_StdDevDriveDOL*
DESCRIPTION	This module controls the operation of a direct online drive configured either with or without running feedback. The module supports automatic and manual control and can be configured with simulation mode to allow the feedback signal to be overwritten and set to follow the demand output. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to Start ② Failed to Stop ③ Failed while Running ④ Failed while Stopped ⑤ External Fault Separate operation times for starting (ramp up) and stopping (ramp down) can be defined. The drive module supports all forms of interlock, permissive and trip signals, and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Stopped • Starting • Running • Stopping • Fault The module also generates status signals for the selected operating modes and conditions.

TITLE	Standard device driver — drives — direct online reversing
BLOCK	FC 12011 *FC12011_StdDevDriveDOLRev*
DESCRIPTION	This module controls the operation of a reversable direct online drive configured either with or without running feedback. The module supports automatic and manual control and can be configured with simulation mode to allow the feedback signal to be overwritten and set to follow the demand output. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to Run Forward ② Failed to Run Reverse ③ Failed to Stop ④ Failed while Running Forwards ⑤ Failed while Running Reverse ⑥ Failed while Stopped ⑦ External Fault Separate operation times for starting forwards (ramp up forward), starting reverse (ramp up reverse) and stopping (ramp down) can be defined. The drive module supports all forms of interlock, permissive and trip signals, and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Stopped • Starting Forwards • Starting Reverse • Running Forwards • Running Reverse • Stopping Forwards • Stopping Reverse • Fault The module also generates status signals for the selected operating modes and conditions.

TITLE	Standard device driver — drives — bistable
BLOCK	FC 12101 *FC12101_StdDevDriveBi*
DESCRIPTION	This module controls the operation of a bistable direct online drive configured either with or without running feedback. The module supports automatic and manual control and can be configured with simulation mode to allow the feedback signal to be overwritten and set to follow the demand state. The module can be configured to either maintain the drive outputs when the drive achieves the required state, or de-energise the outputs when required state is achieved. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to Start ② Failed to Stop ③ Failed while Running ④ Failed while Stopped ⑤ External Fault Separate operation times for starting (ramp up) and stopping (ramp down) can be defined. The drive module supports all forms of interlock, permissive and trip signals and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Stopped • Starting • Running • Stopping • Fault The module also generates status signals for the selected operating modes and conditions.

TITLE	Standard device driver — drives — bistable reversing
BLOCK	FC 12111 *FC12111_StdDevDriveBiRev*

DESCRIPTION

This module controls the operation of a reversable, bistable, direct online drive configured either with or without running feedback.

The module supports automatic and manual control and can be configured with simulation mode to allow the feedback signal to be overwritten and set to follow the demand state.

The module can be configured to either maintain the drive outputs when the drive achieves the required state, or de-energise the outputs when required state is achieved.

The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are:

① Failed to Run Forward
② Failed to Run Reverse
③ Failed to Stop
④ Failed while Running Forwards
⑤ Failed while Running Reverse
⑥ Failed while Stopped
⑦ External Fault

Separate operation times for starting forwards (ramp up forward), starting reverse (ramp up reverse) and stopping (ramp down) can be defined.

The drive module supports all forms of interlock, permissive and trip signals, and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions.

Various status signals are generated for supervisory systems:

- Stopped
- Starting Forwards
- Starting Reverse
- Running Forwards
- Running Reverse
- Stopping Forwards
- Stopping Reverse
- Fault
- Actual and demanded speed

The module also generates status signals for the selected operating modes and conditions

TITLE	Standard device driver — drives — variable speed
BLOCK	FC 12501 *FC12501_StdDevDriveVSD*
DESCRIPTION	This module controls the operation of a variable speed drive optionally configured with analogue speed feedback, and positive running indication. The module supports automatic and manual control and can be configured with simulation mode to allow the feedback signal to be overwritten and set to follow the demand output. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to achieve demanded speed ② External Fault Separate operation times for starting (ramp up) and stopping (ramp down) can be defined. The drive module supports all forms of interlock, permissive and trip signals and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Stopped • Starting • Running • Stopping • Fault • Actual/demanded speed The module also generates status signals for the selected operating modes and conditions.

TITLE	Standard device driver — drives — variable speed reversing
BLOCK	FC 12511 *FC12511_StdDevDriveVSDRev*
DESCRIPTION	This module controls the operation of a reversable, variable speed drive optionally configured with analogue speed feedback, and positive running indication. The reversing mode can be controllable via the analogue signal, or by digital signals to select the direction. The module supports automatic and manual control and can be configured with simulation mode to allow the feedback signal to be overwritten and set to follow the demand output. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to achieve demanded speed ② External fault Separate operation times for increasing and decreasing speed can be defined. The drive module supports all forms of interlock, permissive and trip signals and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Stopped • Running Forwards • Running Reverse • Fault • Actual/demanded speed The module also generates status signals for the selected operating modes and conditions.

TITLE	Standard device driver — drives — multiple speed
BLOCK	FC 12601 *FC12601_StdDevDriveMSD*
DESCRIPTION	This module controls the operation of a multiple speed drive, where multiple fixed speeds are available and are selectable by digital signals. The module can be optionally configured with speed feedback, and positive running indication. The module supports up to 10 different speed selections. The module supports automatic and manual control and can be configured with simulation mode to allow the feedback signal to be overwritten and set to follow the demand output. The module generates fault logic for the valve that will trigger specific alarms depending on the fault in question. The alarms within this block are: ① Failed to Start ② Failed to Stop ③ Failed while Running ④ Failed while Stopped ⑤ Failed to achieve demanded speed ⑥ External fault Separate operation times for increasing and decreasing speed can be defined. The drive module supports all forms of interlock, permissive and trip signals and emergency stop signals. The module has the conditional facility to allow the operator to bypass interlocks, permissive and trip conditions. Various status signals are generated for supervisory systems: • Stopped • Running • Fault • Selected speed The module also generates status signals for the selected operating modes and conditions.

8.10 Message handling

TITLE	Standard message handler — analogue alarm	
BLOCK	FC 16001	*FC16001_StdMsgAnalogAlm*
DESCRIPTION	The module compares an analogue value to a specified threshold setpoint; it has the facility to generate an alarm whenever the signal is beyond the setpoint value (either above or below). The alarm signal can be time filtered and has associated hysteresis. The alarm can be internally acknowledged (from within the Controller) or can rely on the supervisory system alarm handling acknowledgment facilities.	
TITLE	Standard message handler — analogue warning	
BLOCK	FC 16002	*FC16002_StdMsgAnalogWrn*
DESCRIPTION	The module compares an analogue value to a specified threshold setpoint; it has the facility to generate a warning whenever the signal is beyond the setpoint value (either above or below). The warning signal can be time filtered and has associated hysteresis. The warning can be internally acknowledged (from within the Controller) or can rely on the supervisory system alarm handling acknowledgment facilities.	
TITLE	Standard message handler — analogue event	
BLOCK	FC 16003	*FC16003_StdMsgAnalogEvent*
DESCRIPTION	The module compares an analogue value to a specified threshold setpoint; it has the facility to generate an event whenever the signal is beyond the setpoint value (either above or below). The event signal can be time filtered and has associated hysteresis.	
TITLE	Standard message handler — digital alarm	
BLOCK	FC 16101	*FC16101_StdMsgDigitalAlm*
DESCRIPTION	The module generate an alarm whenever the digital signal is active (signal can be active high or active low). The alarm signal can be time filtered. The alarm can be internally acknowledged (from within the Controller) or can rely on the supervisory system alarm handling acknowledgment facilities.	

TITLE	Standard message handler — digital warning	
BLOCK	FC 16102	*FC16102_StdMsgDigitalWrn*
DESCRIPTION	The module generates a warning whenever the digital signal is active (signal can be active high or active low). The warning signal can be time filtered. The warning can be internally acknowledged (from within the Controller) or can rely on the supervisory system alarm handling acknowledgment facilities.	
TITLE	Standard message handler — digital event	
BLOCK	FC 16103	*FC16103_StdMsgDigitalEvent*
DESCRIPTION	The module generates an event whenever the digital signal is active (signal can be active high or active low). The event signal can be time filtered.	
TITLE	Standard message handler — digital time-stamped alarm	
BLOCK	FC 16201	*FC16201_StdMsgAlmTime*
DESCRIPTION	The module generates an alarm whenever the digital signal is active (signal can be active high or active low). The time at which the alarm occurs is recorded (time-stamped), the recorded time is extracted from the Controller real time clock and is accurate to the millisecond. The block also records the signal deactivation time, the duration and the time of acknowledgement. The alarm can be internally acknowledged (from within the Controller) or can rely on the supervisory system alarm handling acknowledgment facilities.	
TITLE	Standard message handler — digital time-stamped warning	
BLOCK	FC 16202	*FC16202_StdMsgWrnTime*
DESCRIPTION	The module generates a warning whenever the digital signal is active (signal can be active high or active low). The time at which the warning occurs is recorded (time-stamped), the recorded time is extracted from the Controller real time clock and is accurate to the millisecond. The block also records the signal deactivation time, the duration and the time of acknowledgement. The warning can be internally acknowledged (from within the Controller) or can rely on the supervisory system alarm handling acknowledgment facilities.	
TITLE	Standard message handler — digital time-stamped event	
BLOCK	FC 16203	*FC16203_StdMsgEventTime*
DESCRIPTION	The module generates an event whenever the digital signal is active (signal can be active high or active low). The time at which the event occurs is recorded (time-stamped), the recorded time is extracted from the Controller real time clock and is accurate to the millisecond. The block also records the signal deactivation time and the duration.	

TITLE	Standard message handler — prompt manager	
BLOCK	FC 16501	*FC16501_StdMsgPrompMgr*
DESCRIPTION	Manages operator prompts (that appear on a supervisory system) on a first come, first served basis (there is no queuing of prompts). The prompt can be acknowledged by the operator (the acknowledgement being passed back to the originating software module) or can be forcibly acknowledged by the Controller software.	
TITLE	Standard message handler — prompt queue	
BLOCK	FC 16502	*FC16502_StdMsgPrompQueue*
DESCRIPTION	Manages operator prompts (that appear on a supervisory system), by storing the prompts in a queue. The queue can support different priority prompts (the prompt priority is a number in the range 0 (low priority) to 99 (high priority), the priority is issued when the prompt is raised); higher priority prompts take precedence over lower priority prompts. The active prompt can be acknowledged by the operator (the acknowledgement being passed back to the originating software module) or can be forcibly acknowledged by the Controller software. Once a prompt has been acknowledged, the next prompt in the queue becomes active.	

8.11 Communication handling

TITLE	Standard communication handler — get data from a controller (small)	
BLOCK	FC 17001	*FC17001_StdCommsGetSmall*
DESCRIPTION	Uses a single **get** instruction to read data from a partner controller via an Ethernet network. This is the fastest mechanism for reading data, but the amount of data is restricted: For S7-1500 to S7-1500 a maximum of 880 bytes of data can be read. If either Controller is an S7-1200 a maximum of 160 bytes of data can be read.	
TITLE	Standard communication handler — put data into a controller (small)	
BLOCK	FC 17002	*FC17002_StdCommsPutSmall*
DESCRIPTION	Uses a single **put** instruction to write data to a partner controller via an Ethernet network. This is the fastest mechanism for writing data, but the amount of data is restricted: For S7-1500 to S7-1500 a maximum of 880 bytes of data can be written. If either Controller is an S7-1200 a maximum of 160 bytes of data can be written.	
TITLE	Standard communication handler — read data from a controller (65K of data)	
BLOCK	FC 17101	*FC17101_StdCommsRead65K*
DESCRIPTION	Read module in a read/write partnership (in association with FC17102), used to transfer a large amount of data between controllers via an Ethernet network. The maximum amount of data that can be transferred is 65535 bytes and requires multiple Controller cycles to complete (asynchronous operation). This module cannot be used with S7-1200 Controllers.	
TITLE	Standard communication handler — write data to a controller (65K of data)	
BLOCK	FC 17102	*FC17102_ StdCommsWrite65K*
DESCRIPTION	Write module in a read/write partnership (in association with FC17101), used to transfer a large amount of data between controllers via an Ethernet network. The maximum amount of data that can be transferred is 65535 bytes and requires multiple Controller cycles to complete (asynchronous operation). This module cannot be used with S7-1200 Controllers.	

TITLE	Standard communication handler — dynamically configure Ethernet interface	
BLOCK	FC 17401	*FC17401_ StdCommsSetIP*
DESCRIPTION	Used to dynamically configure or reconfigure an Ethernet or Profinet interface, the module can change the following: • IP address • Subnet mask • Router address • Profinet device name (Profinet networks only)	
TITLE	Standard communication handler — read data via a point-to-point interface	
BLOCK	FC 17501	*FC17501_StdCommsPtP_Rx*
DESCRIPTION	Read module in a read/write partnership (in association with FC17502), used to transfer data between controllers via a point-to-point network (RS232/RS485 &c.). Requires multiple Controller cycles to complete (asynchronous operation).	
TITLE	Standard communication handler — write data via a point-to-point interface	
BLOCK	FC 17502	*FC17502_ StdCommsPtP_Tx*
DESCRIPTION	Write module in a read/write partnership (in association with FC17501), used to transfer data between controllers via a point-to-point network (RS232/RS485 &c.). Requires multiple Controller cycles to complete (asynchronous operation).	

8.12 Subroutines

(1) Subroutine modules are common modules that perform some specific function. Subroutine modules can be called from within any other block.

(2) Subroutines are simple modules the perform some function (convert a number to a string for example) and are intended to provide commonly required utilities that are often required in Controller programming.

(3) Subroutines can be called by any other software modules, as a generally rule, the PAL standard modules do not use subroutines, simply for the reason that the standard modules should be stand-alone modules that do not require other modules to work.

TITLE	Standard subroutines — scale an analogue input signal	
BLOCK	FC 18001	*FC18001_StdSubScaleAI*
DESCRIPTION	This module reads and scales an analogue instrument signal received via an analogue input card. The resultant scaled value is a real (floating point) number that matches the calibrated range of the instrument in engineering units.	
TITLE	Standard subroutines — scale an analogue output signal	
BLOCK	FC 18002	*FC18002_StdSubScaleAQ*
DESCRIPTION	This module takes a real number in a specified range and converts it to an integer value suitable for writing to an analogue output card.	
TITLE	Standard subroutines — timer module (100 ms resolution)	
BLOCK	FC 18101	*FC18101_StdSubTime100ms*
DESCRIPTION	This module is a countdown timer, counting down in 100 ms intervals. The initial time and the elapsed time are specified in seconds as real numbers. The timer is accurate for periods up to 27.78 hours (100,000 seconds). The timer is accurate to within 100 ms.	
TITLE	Standard subroutines — timer module (1 s resolution)	
BLOCK	FC 18104	*FC18104_StdSubTime1s*
DESCRIPTION	This module is a countdown timer, counting down in 1 s intervals. The initial time and the elapsed time are specified in seconds as real numbers. The timer is accurate for periods up to 11.5 days (1,000,000 seconds). The timer should not be used to time events of less than 1 hour duration. The timer is accurate to within 1 s.	

TITLE	Standard subroutines — timer module, long duration timer	
BLOCK	FC 18111	*FC18111_StdSubTimeLong*
DESCRIPTION	This module is a count-up timer that is capable of measuring long time durations with a resolution of 1 ms. The timer measures in integer units of days, hours, minutes, seconds and milliseconds. The maximum value of the timer is 65535 days (approx. 179 years). The overall accuracy of the timer is that of the internal clock of the CPU.	
TITLE	Standard subroutines — event duration timer (using the RTC)	
BLOCK	FC 18151	*FC18151_StdSubTimeEventRTC*
DESCRIPTION	This module times the duration of an event to nanosecond resolution. The block records the time the event started, the time the event ended and calculates the duration of the event (end time minus the start time). The start and end times are read from the real time clock of the Controller.	
TITLE	Standard subroutines — count up/down function	
BLOCK	FC 18201	*FC18201_StdSubCounter*
DESCRIPTION	This module counts the number of rising edges detected on a signal. The module can be configured for two signals, a rising edge on the first increments the counter by a specified amount, a rising edge on the second decrements the counter by a specified amount. The count ranges positive and negative, the count is given as a real number. The counter can be pre-loaded with a starting value and can be reset at any point.	
TITLE	Standard subroutines — string function — convert an integer to ASCII	
BLOCK	FC 18901	*FC18901_StdSubStrIntToASC*
DESCRIPTION	Converts a decimal number stored in a double integer to an ASCII string. The number can be in the range -2,147,483,648 to +2, 147,483,647, the number of characters can be specified (the result will contain leading zeros where necessary). The result can be shifted to include decimal places (the decimal will be encoded in the string)	
TITLE	Standard subroutines — string function — convert a real to ASCII	
BLOCK	FC 18902	*FC18902_StdSubStrRealToASC*
DESCRIPTION	Converts a real number to an ASCII string. The string result will reflect the real value exactly, including any exponents.	

TITLE	Standard subroutines — string function — convert a string to an integer value
BLOCK	FC 18911 *FC18911_StdSubStrASCtoInt*
DESCRIPTION	Converts a decimal number stored as a string to an integer value. Non numeric characters are ignored (including any decimal point), a leading minus sign will generate a negative number.
TITLE	Standard subroutines — string function — convert a string to a real value
BLOCK	FC 18912 *FC18912_StdSubStrASCtoReal*
DESCRIPTION	Converts a decimal number stored as a string to a real value. Non numeric characters are ignored, the decimal point and exponentials are supported; a leading minus sign will generate a negative number.
TITLE	Standard subroutines — string function — case conversion
BLOCK	FC 18921 *FC18921_StdSubStrCaseConv*
DESCRIPTION	Converts a string to upper case, lower case or sentence case.
TITLE	Standard subroutines — string function — concatenate strings
BLOCK	FC 18931 *FC18931_StdSubStrConcat*
DESCRIPTION	Concatenates two strings.
TITLE	Standard subroutines — string function — split a string
BLOCK	FC 18932 *FC18932_StdSubStrSplit*
DESCRIPTION	Splits a string into two strings at a particular character point.
TITLE	Standard subroutines — string function — find a string within a string
BLOCK	FC 18933 *FC18933_StdSubStrFind*
DESCRIPTION	Finds the first occurrence of a string within another string, the starting point can be specified.

8.13 Debug subroutines

(1) Debug routines are generally used in the testing stages of software development, they should not under any circumstances be present in deployed software.

(2) Debug subroutines are used in two separate locations:

- Start of cycle (SoC) debug, executed before any other software (even `FC01001StsSysGlobalData`)
- End of scan (EoC) debug, called as the last entry in OB 1 and executed after all other software

(3) The start of cycle debug is intended to allow IO signals to be manipulated, overwriting any real IO data from the Controller IO card. The SoC debug also provides various switch mechanism to allow various different aspects of the debug software to be activated or deactivated. Typically, these are:

- IO signal simulation
- Instrument simulation
- Device simulation
- Communication simulation
- Process simulation
- Sequence break point operation

(4) The end of cycle debug generally generates simulation signals, this can be limit switch signals for a valve (allowing the valve to appear to operate correctly or to force fault conditions), it can also include more complex simulations, even simulating process operations (the heating of a vessel for example).

(5) The EoC debug is also responsible for setting sequence break point (stopping a sequence at a particular point to allow signal conditions to be assessed) or allowing "single-step" operations of sequences.

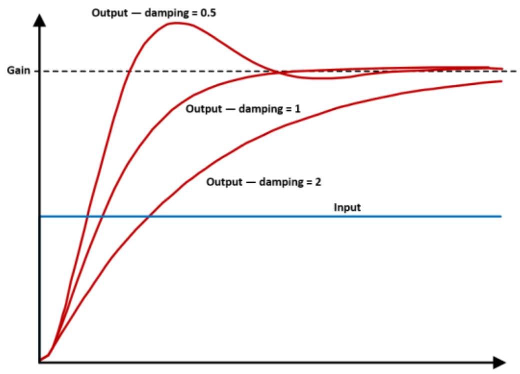
TITLE	Standard debug subroutines — simulation — isolating valve
BLOCK	FC 19001 *FC19001_StdDebugValveIsol*
DESCRIPTION	Simulates the response of an isolating valve IO signals. The simulation can be configured for a normally closed or normally open isolating valve, with either opened, closed, or both opened and closed position feedback (simulation is not required for valves with no position feedback). The open and close times can be specified individually, each feedback signal can be manually changed to simulate fault conditions.
TITLE	Standard debug subroutines — simulation — bistable isolating valve
BLOCK	FC 19002 *FC19002_StdDebugValveBi*
DESCRIPTION	Simulates the response of a bistable isolating valve IO signals. The simulation can be configured with either opened, closed, or both opened and closed position feedback (simulation is not required for valves with no position feedback). The open and close times can be specified individually, each feedback signal can be manually changed to simulate fault conditions.
TITLE	Standard debug subroutines — simulation — modulating valve
BLOCK	FC 19003 *FC19003_StdDebugValveMod*
DESCRIPTION	Simulates the response of a modulating valve IO signals. The simulation can be configured for a positive acting or negative acting modulating valve, the block will generate an analogue position feedback signal and opened, closed, or both open and closed position feedback. The open rate of change and close rate of change times can be specified individually, each feedback signal can be manually changed to simulate fault conditions.

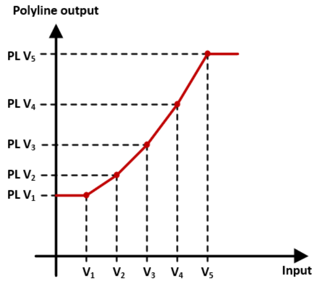
TITLE	Standard debug subroutines — simulation — drive DOL	
BLOCK	FC 19011	*FC19011_StdDebugDriveDOL*
DESCRIPTION	Simulates the IO signal responses of a standard or reversing DOL drive. The simulation can be configured to generate positive running feedback. The ramp-up and ramp-down times can be specified individually, each feedback signal can be manually changed to simulate fault conditions.	
TITLE	Standard debug subroutines — simulation — drive bistable	
BLOCK	FC 19012	*FC19012_StdDebugDriveBi*
DESCRIPTION	Simulates the IO signal responses of a standard or reversing, bistable DOL drive. The simulation can be configured to generate positive running feedback. The ramp-up and ramp-down times can be specified individually, each feedback signal can be manually changed to simulate fault conditions.	
TITLE	Standard debug subroutines — simulation — drive variable speed	
BLOCK	FC 19013	*FC19013_StdDebugDriveVSD*
DESCRIPTION	Simulates the IO signal responses of a standard or reversing, variable speed drive. The simulation can be configured to generate an analogue speed feedback and positive running feedback. The ramp-up and ramp-down times can be specified individually, each feedback signal can be manually changed to simulate fault conditions.	
TITLE	Standard debug subroutines — simulation — drive multiple speed	
BLOCK	FC 19014	*FC19014_StdDebugDriveMSD*
DESCRIPTION	Simulates the IO signal responses of a multiple speed drive. The simulation can be configured to generate positive running feedback and selected speed feedback. The ramp-up and ramp-down times can be specified individually, each feedback signal can be manually changed to simulate fault conditions.	

TITLE	Standard debug subroutines — simulation — instrument flow	
BLOCK	FC 19101	*FC19101_StdDebugInstFlow*
DESCRIPTION	Simulates the response of a flow instrument to a change in the process configuration (opening or closing a valve). The flow range can be specified as can the response time. The module can simulate a response to either a modulating valve (variable response) or an isolating valve (on/off response). The generated signal can be manually changed to simulate fault conditions.	
TITLE	Standard debug subroutines — simulation — instrument level	
BLOCK	FC 19102	*FC19102_StdDebugInstLevel*
DESCRIPTION	Simulates the response of a level instrument to a change in the process configuration. The level range can be specified. The module can simultaneously accommodate both a feed and a discharge arrangement (feed increases the level; discharge reduces the level). The rate of level change for both feed and discharge can be defined separately. The module can simulate a response to either a modulating feed/discharge (variable response) or a fixed feed/discharge (on/off response) or a combination of both. The generated signal can be manually changed to simulate fault conditions.	
TITLE	Standard debug subroutines — simulation — instrument temp	
BLOCK	FC 19103	*FC19103_StdDebugInstTemp*
DESCRIPTION	Simulates the response of a temperature instrument to a change in the process configuration. The temperature range can be specified. The module can simultaneously accommodate both a heating and cooling arrangement (heating increases the temperature; cooling reduces the temperature). The rate of temperature change for both heating and cooling can be defined separately. The module can simulate a response to either a modulating heating/cooling (variable response) or a fixed heating/cooling (on/off response) or a combination of both. The generated signal can be manually changed to simulate fault conditions.	

TITLE	Standard debug subroutines — simulation — instrument pressure
BLOCK	FC 19104 *FC19104_StdDebugInstPres*
DESCRIPTION	Simulates the response of a pressure instrument to a change in the process configuration. The pressure range can be specified. The module can simultaneously accommodate both a feed and a discharge arrangement (feed increases the pressure; discharge reduces the pressure). The rate of pressure change for both feed and discharge can be defined separately. The module can simulate a response to either a modulating feed/discharge (variable response) or a fixed feed/discharge (on/off response) or a combination of both. The generated signal can be manually changed to simulate fault conditions.

TITLE	Standard debug subroutines — simulation — instrument 1st order response
BLOCK	FC 19511 *FC19151_StdDebugInst1Order*
DESCRIPTION	Simulates a first order process reaction in response to an input signal. The range of both the input and output signals can be specified as can the gain and lag constants. $Output = Input \cdot Gain \cdot (1 - e^{-\frac{t}{Lag}})$ The generated signal can be manually changed to simulate fault conditions.

TITLE	Standard debug subroutines — simulation — instrument 2nd order response
BLOCK	FC 19512 *FC19152_StdDebugInst2Order*
DESCRIPTION	Simulates a second order process reaction in response to an input signal. The range of both the input and output signals can be specified as can the gain and damping constants.  The generated signal can be manually changed to simulate fault conditions.

TITLE	Standard debug subroutines — simulation — polyline response	
BLOCK	FC 19513	*FC19153_StdDebugInstPoly*
DESCRIPTION	Simulates a piecewise linear polyline response to an input signal:  The polyline has a minimum of two points and a maximum of 100 points. The generated signal can be manually changed to simulate fault conditions.	
TITLE	Standard debug subroutines — simulation — sequence breakpoint	
BLOCK	FC 19701	*FC19701_StdDebugSeqBreak*
DESCRIPTION	Interrupts the normal sequence progression, and forces a sequence pause (breakpoint) at each transition, allowing the sequence step/transition conditions to be examined and debugged.	
TITLE	Standard debug subroutines — simulation — sequence breakpoint	
BLOCK	FC 19999	*FC19999_StdDebugForceStop*
DESCRIPTION	Conditionally forces the CPU to a stop state.	

BLANK PAGE

9 Standard sequence operation

(1) The standard sequence modules associated with the PAL software are designed to allow multiple, sequential operations to be configured and implemented within a Controller.

(2) The sequences use a step and transition arrangement to control the progression of a sequence.

9.1 Operating states and commands

(1) Sequential control within the PAL use a series of standard modules included with the library these in turn are based upon the sequential function chart requirements of the IEC 61131-3 standards *[Ref. 012]*.

(2) Each sequence within the PAL operates by moving through a series of states that broadly indicate what the sequences itself is doing (i.e. idle, starting, running, aborting &c.).

(3) These states are referred to as the *Operating State Logic* (OSL) of the sequence; Figure 9.1 shows the full arrangement of Operating States available to the PAL sequences.

Note: *It is not necessary for every sequence to use every state available to the OSL; most sequences have a subset of the OSL depending on the functionality and complexity of the sequence in question.*

It is true to say that all sequences must have at the very least: idle, starting, running, completing and completed states (even if these states are empty), these states are necessary for the sequence to operate normally.

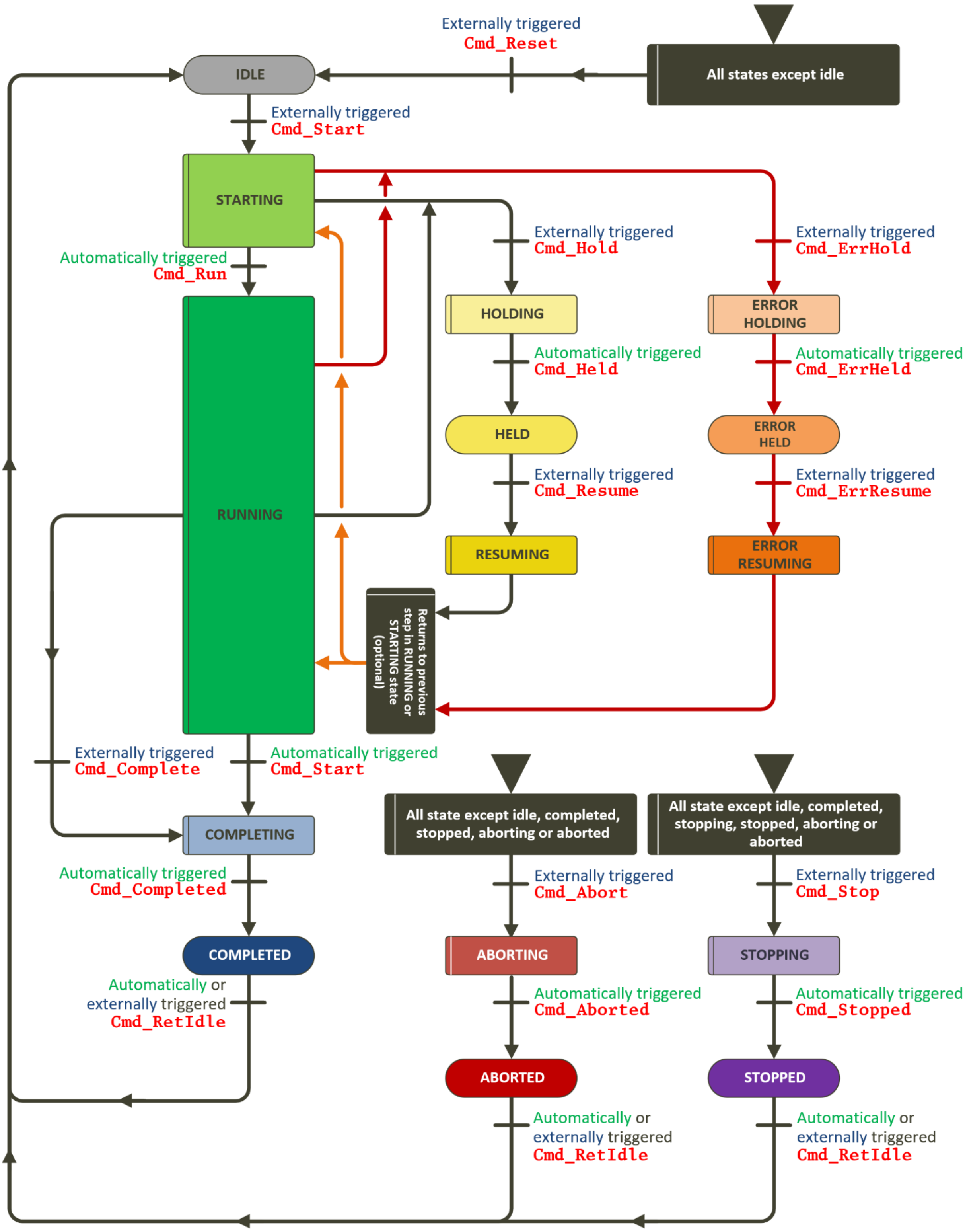


Figure 9.1 Sequential operating state logic (OSL)

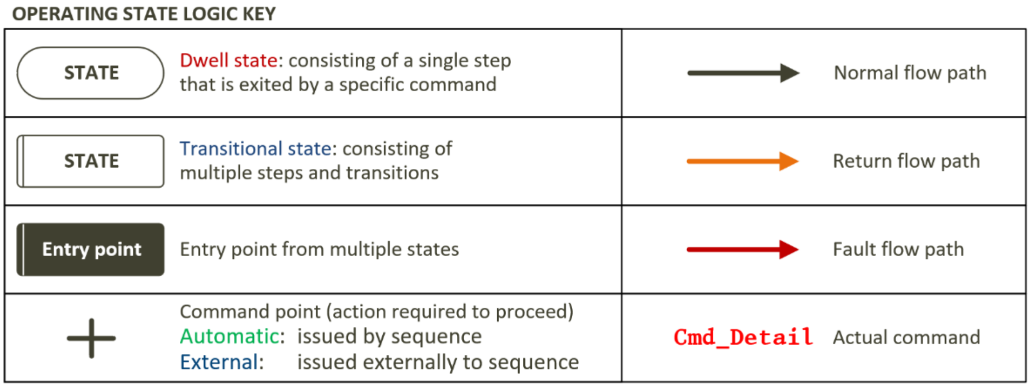


Figure 9.2 Operating state logic — key

(4) Each state of the sequence consists of various *steps* and *transitions*. The step performs some action (opens a valve, start a drive &c.) and the transition consists of a series of logical conditions that must be satisfied before the sequence can move to the next step.

(5) Each step within a sequence is given a unique number. The numbering ranges of the steps indicate the OSL to which the step belongs:

STEP NO.	STATE (WITHIN THE OSL)
0	IDLE
01000-09999	STARTING
10000-19999	RUNNING
20000-28999	COMPLETING
29000	COMPLETED
30000-34999	HOLDING
35000	HELD
36000-39999	RESUMING
40000-44999	ERROR HOLDING
45000	ERROR HELD
46000-49999	ERROR RESUMING
50000-54999	STOPPING
55000	STOPPED
60000-64999	ABORTING
65000	ABORTED

Table 9.1 OSL sequence step numbers

(6) The step number is held in an unsigned integer, this spans the range 0 to 65,535.

(7) The dwell states are single points within the sequence, the sequence will remain in a dwell state until it receives a command from an external source (external to the sequence itself); the IDLE state for example, is the state applicable to a non-active sequence (i.e. a sequence that is not currently running).

(8) The sequence will remain in the IDLE state (or any dwell state) until it receives a command (the Cmd_) of Figure 9.1. Such commands could be directly from the operator (via a supervisory system) or from elsewhere within the software (for example, a backwash sequence might start at a particular time of day, or the sequence to empty at tank may occur when the tank reaches a certain level).

9.1.1 Normal sequential operation

(1) The normal progression of a sequence would follow the flow paths shown in Figure 9.1. A sequence that is not running will be in the IDLE state and will wait in this condition until it receives a start command (Cmd_Start). The start command causes the sequence to activate and move to step 1000 (the first step of the STARTING state), the STARTING state is used to setup the initial conditions for the sequence or to carry out some preliminary actions prior to the main purpose of the sequence (for example perform a pressure test, run a sterilisation process, collect data from the operator &c.). The STARTING state steps are numbered in the range 1000-9999 (i.e. there is a maximum of 9000 steps available to the state).

(2) It is perfectly possible for the STARTING state to be empty, in which case step 1000 will simply trigger the run command.

(3) When the STARTING state is complete, the sequence itself will automatically trigger the Cmd_Run signal, at which point the sequence will advance to the RUNNING state at step 10000. This is the sequence proper, and carries out the primary function of the sequence.

(4) The RUNNING state is generally the largest section of the sequence and can accommodate up to10000 individual steps. A sequence will always have code within the RUNNING state.

(5) The RUNNING state automatically triggers the transition to the COMPLETING state by triggering the Cmd_Complete signal, this forces the sequence to step 20000.

(6) The COMPLETING state is analogous to the STARTING state, it allows the sequence to carry out various terminating (housekeeping) activities prior to the sequence ending (this may be collecting data from the operator, recording final readings &c.).

(7) The final action of the COMPLETING state is to trigger the Cmd_Completed signal, this forces the sequence to the COMPLETED dwell state at step 29000. The sequence will now remain in this state, indicating that the sequence has run to completion and no further actions will be taken.

(8) At this point, the sequence is effectively stopped and is once more inactive, the COMPLETED dwell state informs any software monitoring the sequence that it has finished. The sequence returns to the IDLE state when the Cmd_RetIdle signal is issued.

(9) The sequence can be optionally configured to automatically trigger the Cmd_RetIdle signal once the sequence is in the COMPLETED dwell state (this is a normal practice for sequences that operate independently of other sequences or have little interaction with other systems).

(10) Where sequences are used within other sequences (usually as part of a parallel arrangement, see § 9.2.3) the COMPLETED dwell state is used to identify when all sections of the parallel arrangement have completed.

9.1.2 Hold and error hold operation

(1) There are two modes of holding operations: these are triggered by the Cmd_Hold and the Cmd_ErrHold signals.

(2) Both can only be triggered during the STARTING or RUNNING states (the commands will be ignored in any other state), the Cmd_ErrHold signal is triggered in the event of a fault being detected that is applicable to the sequence. The Cmd_ErrHold signal is normally generated by monitoring logic, this may or may not be part of the sequence itself, the signal can also be triggered by the operator if necessary.

(3) Once triggered, the Cmd_ErrHold signal causes the sequence change state to ERROR HOLIDNG and to advance to step 40000.

(4) The ERROR HOLIDNG state allows the sequence to put the areas of the plant controlled by the sequence into a known safe condition (for example, isolating a filter or stopping feed supplies &c.). Once this is done, the Cmd_ErrHeld signal is automatically trigged and the sequence enters the ERROR HELD dwell state at step 45000. The sequence will remain in this state, allowing the fault condition to be rectified.

(5) The operator must issue the Cmd_ErrResume signal to allow the sequence to continue.

(6) Triggering the Cmd_ErrResume signal forces the sequence to the ERROR RESUMING state at step 46000. This state is used to return the plant to an operational state (by restoring the plant to the previous condition prior to the ERROR HOLIDNG operations.

(7) At the end of the ERROR RESUMING state, there are various configurable options:

- The sequence can automatically return to the last step (in either the RUNNING or STARTING states) it was at prior to the error condition being detected
- It can start from a particular step (i.e. not the last step it was at, but any specified step in the sequence
- It can restart from the beginning

(8) The required response is entirely dependent on the nature of the sequence in question.

(9) The Cmd_Hold signal operates in a similar fashion to the Cmd_ErrHold signal, however in this case, the Cmd_Hold signal can only be triggered by the operator (it is a manual action).

(10) Triggering the Cmd_Hold signal moves the sequence to the HOLIDNG state (analogous to the ERROR HOLIDNG state) beginning at step 30000; the automatic triggering of the Cmd_Held signal places sequence in the HELD dwell state at step 35000. The sequence will again remain in this state, allowing the operator to take whatever action is required.

(11) The operator must issue the Cmd_Resume signal to allow the sequence to continue.

(12) the Cmd_Resume signal forces the sequence to the RESUMING state at step 36000. Again, this state is used to return the plant to an operational state

(13) At the end of the RESUMING state, the sequence can continue with any of the configurable options listed for the ERROR RESUMING state (listed above).

9.1.3 Stop and abort operation

(1) The stop and abort operations are alternative mechanisms for terminating a sequence in the event that something goes wrong and the sequence operations are unrecoverable (for example, the sequence is waiting for a condition such as a level or pressure that can never be achieved because of some fault that cannot easily be rectified).

(2) The two modes of operation allow the sequence to be shut down in either a coordinated and controlled manner (stopping) or more abruptly carrying out only those steps that are necessary to safely terminate the sequence (aborting).

(3) Stopping and aborting are always triggered manually (or at least via logic external to the sequence itself) by issuing either the Cmd_Stop or Cmd_Abort signal. The Cmd_Stop signal can be issued during any state of the sequence excepting IDLE, COMPLETED, STOPPING, STOPPED, ABORTING or ABORTED. The Cmd_Abort signal has the same restrictions, except it can also be triggered in the STOPPING state (in this regard aborting has a higher priority than stopping and can interrupt it).

(4) Triggering the Cmd_Stop signal forces the sequence to the STOPPING state (at step 50000) at the end of the STOPPING state, the Cmd_Stopped signal is automatically trigged and the sequence enters the STOPPED dwell state at step 55000. The sequence will remain in this state until the issues Cmd_RetIdle signal is triggered (usually by the operator), at which point the sequence returns to the IDLE state.

(5) Triggering the Cmd_Abort signal forces the sequence to the ABORTING state (at step 60000) at the end of the ABORTING state, the Cmd_Aborted signal is automatically trigged and the sequence enters the ABORTED dwell state at step 65000. The sequence will remain in this state until the issues Cmd_RetIdle signal is triggered (usually by the operator), at which point the sequence returns to the IDLE state.

9.1.4 The reset operation

(1) The sequence command: Cmd_Reset this will force a reset of the sequence back to the IDLE state, irrespective of whatever state the sequence is currently in.

(2) The reset command is an overriding command and will take precedence over any other command that may have been issued.

(3) The Cmd_Reset signal should only be triggered by the operator. It is intended as a recovery mechanism for a sequential operation that cannot be recovered by any other mechanism (hold, error, stop or abort).

9.1.5 The pause operation

(1) The sequence has a final command: Cmd_Pause this is to some extent is a debug function, if active, it will pause the sequence in its current step, no transitions will be evaluated and the step duration and delay timers will pause at their current values.

(2) The sequence will remain in this state whilever the Cmd_Pause signal is in set to true. Once released, the sequence will continue as if nothing had happened.

9.2 Steps and transitions within a sequence

(1) The non-dwell states within a sequence hold a series of steps and transitions that make up the sequence, the steps perform an action, the transitions are a series of logical tests, which, once satisfied, cause the sequence to progress from the current step to another step (usually, the next step).

(2) Graphically, these sequences can be represented as the step-transition diagram of a sequential flow chart (sometimes referred to as a GRAFCET[15] diagram), see Figure 9.3:

---

15 GRAFCET, *GRAPHe de Commande Etape-Transition*, French. Literally, "stage-transition command graph" a diagrammatic mechanism for showing steps and transitions within a sequence.

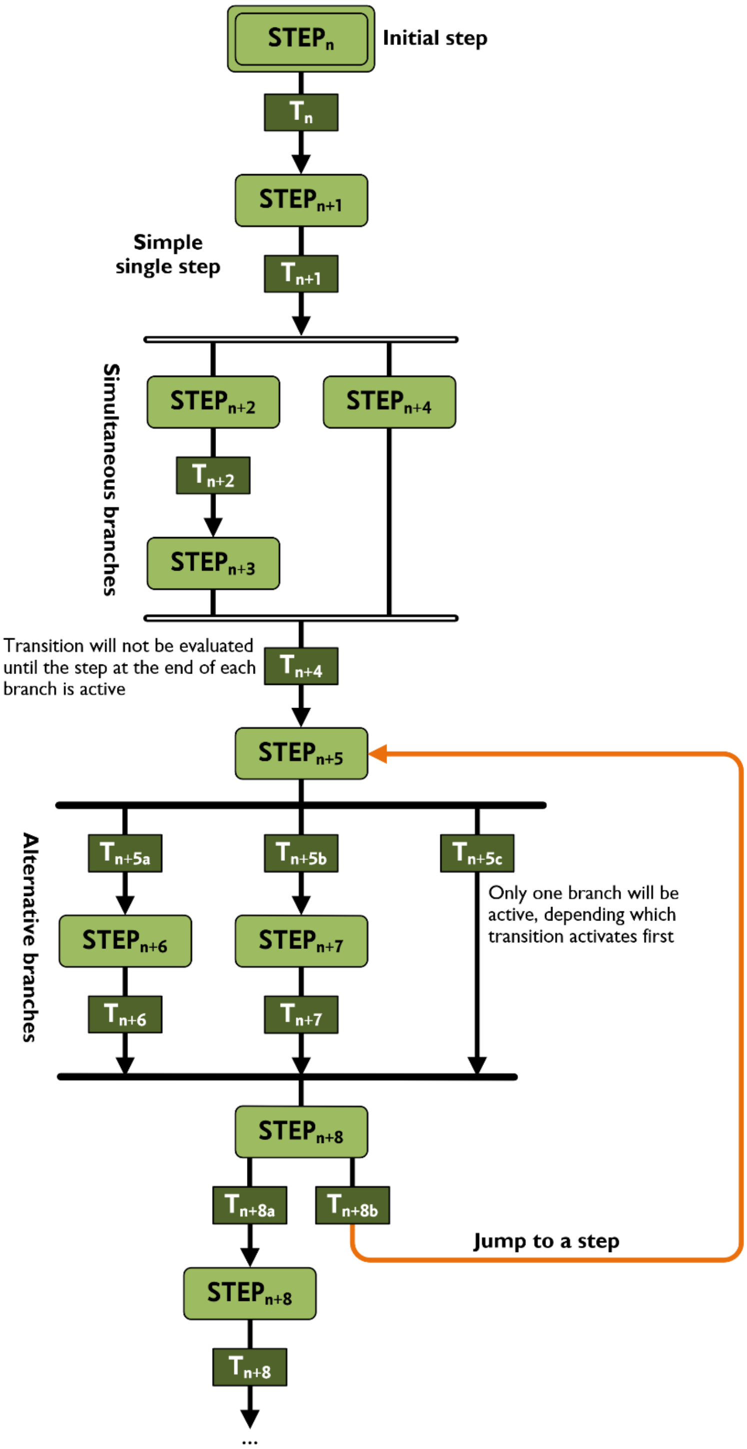


Figure 9.3 Step transition diagram

(3) Figure 9.3 shows all the step transition and branching mechanisms available to the PAL sequences:

- Simple step and single transition
- Alternative (divergent) branches
- Simultaneous (parallel) branches
- Jumps (and loops) to a particular step

9.2.1 Simple steps and transitions

(1) Most steps within a sequence are simple step and transition arrangements that move from one step directly to the next step when the transition conditions are satisfied:

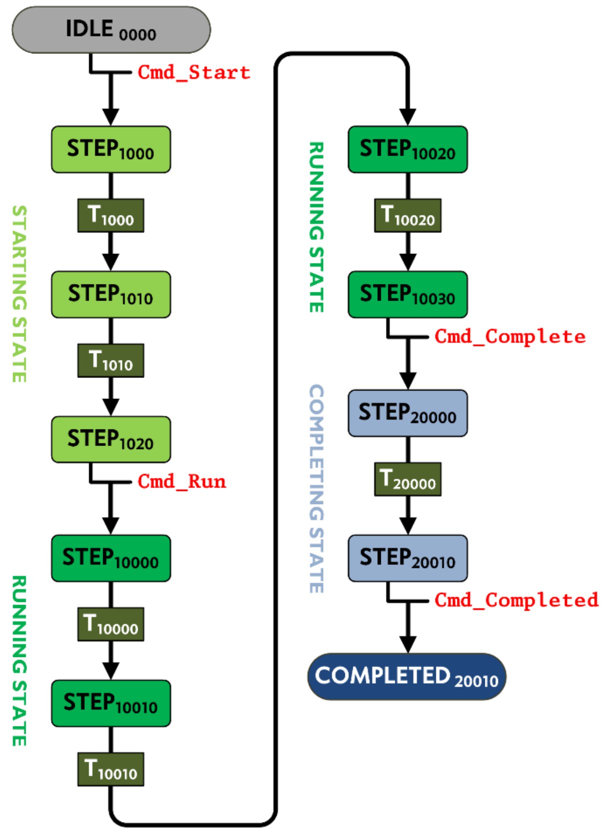


Figure 9.4 Simple step and transition arrangements

(2) Figure 9.4 shows a simple sequence progressing through the minimum number of states (`IDLE`, `STARTING`, `RUNNING`, `COMPLETING` and `COMPLETED`). Figure 9.4 shows the step numbers (and associated transitions) that would be assigned for such a sequence.

(3) Where there is a transition from one state to anther (e.g. `starting` to `running`), the final step of the first state automatically triggers the command to move to the next, this can be an instantaneous action that triggers when the step becomes active, or it can be linked to a transition condition for the step. Figure 9.5 shows the two symbols for firstly (on the left) automatic instantaneous command triggering and (on the right) transition dependent command triggering:

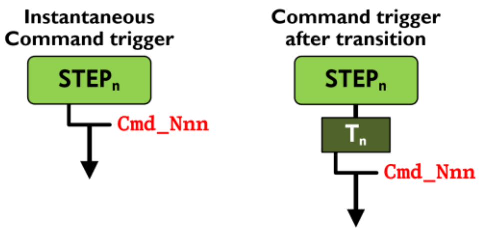


Figure 9.5 Automatic command issuing arrangements

(4) The step numbers used in Figure 9.4 increment in intervals of 10; this is done to allow space for additional step between the existing steps (for example it would be possible to add an additional 9 steps between step 1000 and step 1010). This approach is not an essential requirement (it would be perfectly possible to increment the steps by 1 and leave no gaps), it does however, reflect good practice and is a recommended approach for sequences using the PAL software.

9.2.2 Alternative branching

(1) Alternative branching is a common requirement for sequential actions, it allows the sequence to progress down multiple divergent paths. The following diagrams show divergent sequence arrangements.

(2) The simplest alternative branch splits the sequence path into two, Figure 9.6:

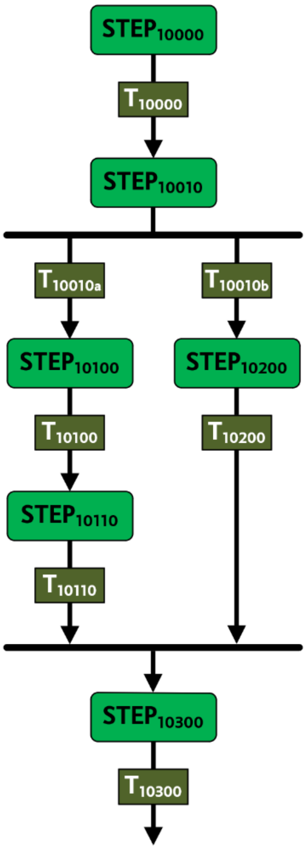


Figure 9.6 Simple alternative branch

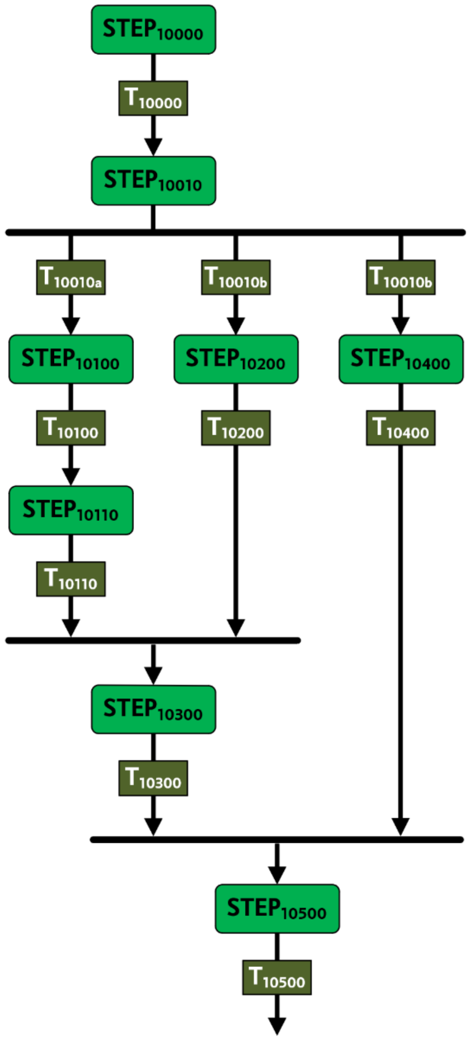


Figure 9.7 More complex alternative branch

(3) In Figure 9.6, step `10010` has two transitions associated with it (`10010a` and `10010b`), if transition `10010a` activates first, the sequence will move to step `10100` and the alternative leg (with step `10200`) will be ignored and never executed.

(4) Similarly, if `10010b` activates first, the sequence will move to step `10200` and the alternative leg (with step `10100`) will be ignored and never executed. Whichever branch is executed, the result will arrive at step `10300` and the sequence will continue from there.

(5) More complex arrangements can be made (Figure 9.7), here the 10010a and 10010b transitions operate exactly as Figure 9.6, the 10010c transition, however, diverts the sequence down the step 10400 path and this bypasses completely the merge point of the 10010a and 10010b transitions (at step 10300) and moves to a new merge point at step 10500.

(6) Alternative branches can be as complicated as required and can include commands:

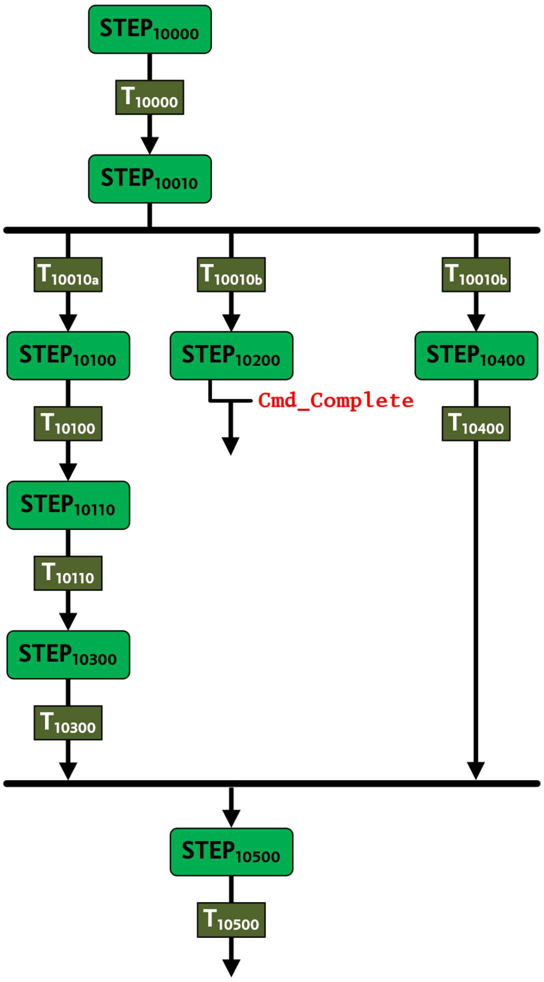


Figure 9.8 Alternative branch with commands

(7) Each step within the PAL sequences can have up to eight separate transitions

(8) In the event of two transitions becoming active at the same time, the lowest number transition will take priority.

9.2.3 Simultaneous branches

(1) Simultaneous branches are more complicated in their execution than alternative branches. With alternative branches there is only ever one step active at any given point in time; simultaneous branches have multiple steps active at the same time and this is not possible with the PAL sequence arrangements.

(2) Within the PAL, simultaneous branches are achieved by using separate sub-sequences for each branch as follows:

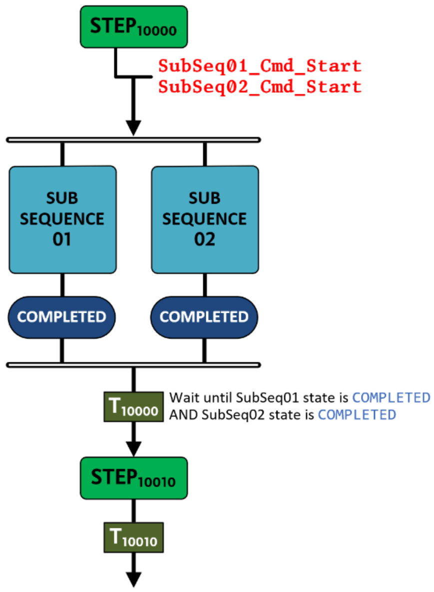


Figure 9.9 Simultaneous branches with sub-sequences

(3) The sub-sequences are just PAL sequences, the main sequence triggers the two sub-sequences by activating the `Cmd_Start` signal for each sub-sequences, each sub-sequence will then operate in its own right.

(4) At some point both sub-sequences will finish and will be at the `COMPLETED` dwell state, the transition `T10000` is waiting for this condition (i.e. for both sub-sequence 01 to be in the `COMPLETED` state `AND` for sub-sequence 02 to be in the `COMPLETED` state), at this point the main sequence will advance to step `10010`, this step would issue the `Cmd_RetIdle` signals for the sub-sequences.

(5) As many sub-sequences as required can be used (i.e. there is no limit to the number of simultaneous branches).

9.2.4 Jumps and loops

(1) Jumps and loops are very similar in operation to alternative branches, they cause the sequence to jump to different points depending upon which transition becomes true:

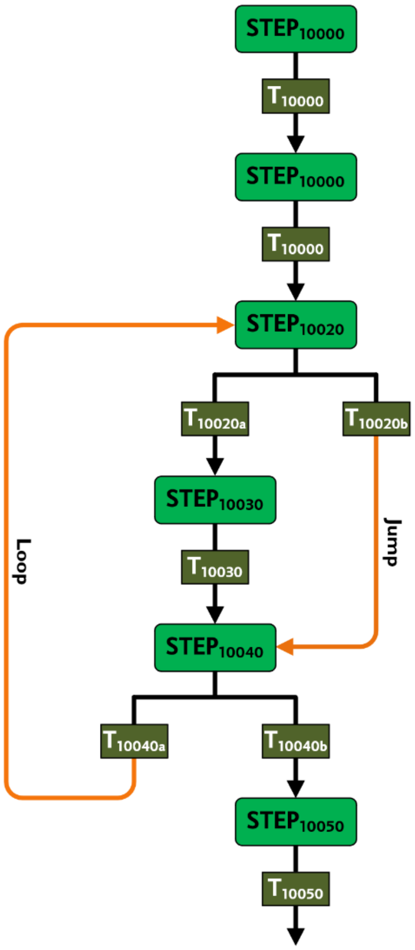


Figure 9.10 Jumps and loops

(2) In practical terms, jumps and loops are identical, they simply move the sequence to a step that is not the logical next step in the sequence. The terminology simply reflects the direction of the movement: a *jump* advances the sequence forward to a step that has not yet been executed, a *loop* moves the sequence back to a previous step (creating the potential for a loop)

(3) The execution of a jump or loop is identical to that of an alternative branch, the step has multiple transitions, each jumping (or looping) to a different step.

9.3 Phases within a step

(1) At its simplest level, each step within a sequence executes a set of actions (close a valve, start a drive, wait for a time period &c.); however, each step is equipped with *phases* that reflect different aspects of the step:

① Initialising

② Processing

③ Terminating

(2) These three phases effectively allow a step to be interpreted as a three-step sequence in its own right:

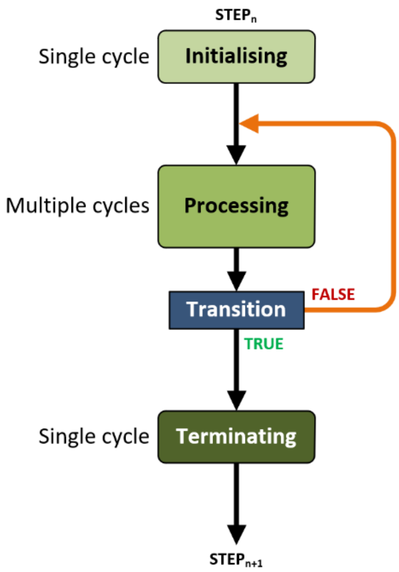


Figure 9.11 Jumps and loops

(3) Each step has three digital signals that identify the current phase: PHS_INIT, PHS_PROC and PHS_TERM. The *initialising* phase is active for one Controller cycle when the step first becomes active. After this cycle, the step will advance to, and remain in the *processing* phase until a transition condition becomes true. At this point, the *processing* phase is deactivated and the *terminating* phase activates. The *terminating* phase is active for just one Controller cycle.

(4) This phased approach to a step allows a single step to carry out a complete set of actions, consider a sequence that is filling a tank, it will take the following actions:

① Open valve V001 (tank inlet)

② Wait for the tank level (LIT001) to reach the target level (T001)

③ Close valve V001

(5) In practical terms, if a step within a sequence simply carried out a single set of actions (the unphased approach), this series of events would require two steps:

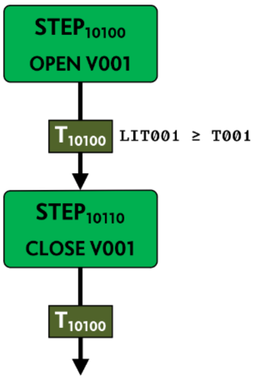


Figure 9.12 Unphased approach to a sequence step

(6) With a phase approach, the series of events is accomplished within the phases of a single step:

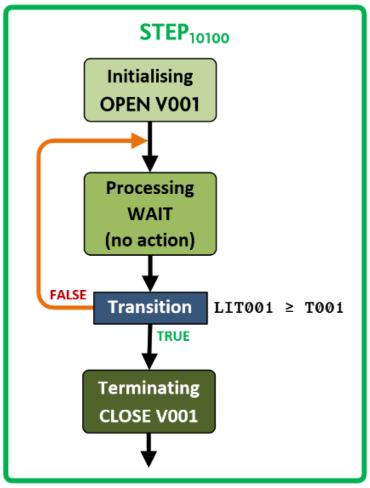


Figure 9.13 Phased approach to a sequence step

(7) With the phase approach, the actions of the step take place (in this instance) in the *initialising* and *terminating* phases, the *processing* phase is simply waiting for the transition condition.

(8) This phased approach to sequence steps is a practical approach to simplifying sequences; it allows the scope of a single step to accommodate multiple actions that are related to each other.

9.3.1 Phase timings for IEC compliant sequence steps

(1) The following diagram shows the phase timing arrangements between two consecutive steps:

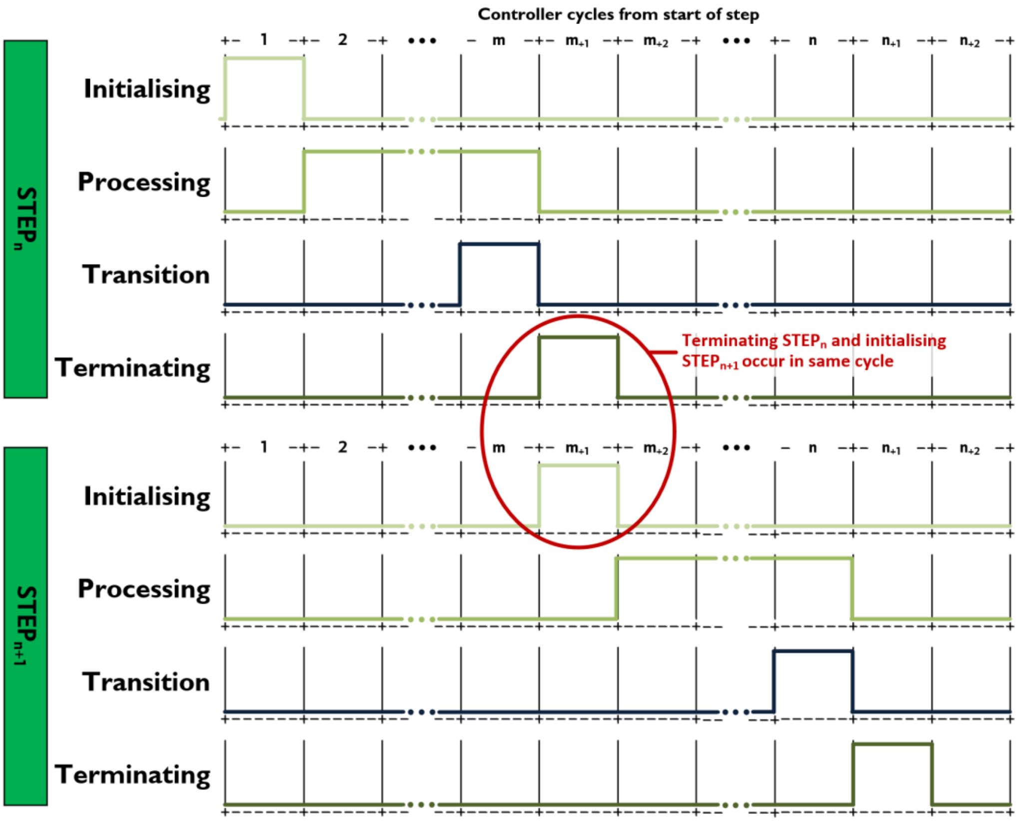


Figure 9.14 IEC compliant phase timing

(2) Here, it can be seen that the *terminating* phase of *step$_n$* is coincident (occurs in the same cycle) with the *initialisation* phase of *step$_{n+1}$*.

(3) This arrangement is required for compliance with IEC 61131-3 *[Ref. 012]*.

(4) This effectively means the $step_{n+1}$ and $step_n$ are both active in at the same time (within the same Controller cycle).

(5) This methodology is required by the User Requirement Specification *[Ref. 003, § 4.2.2 (21)]* and has consequently been implemented in the PAL sequence software.

9.3.2 Phase timings for non-IEC compliant sequence steps

(1) This methodology highlighted in the previous section, whilst being complainant with the IEC 61131-3 specification, is not widely used or highly regarded by those who practice the programming of Controllers and PLCs.

(2) The more conventional view is that steps within a sequence should not overlap, the preferred timing arrangement being:

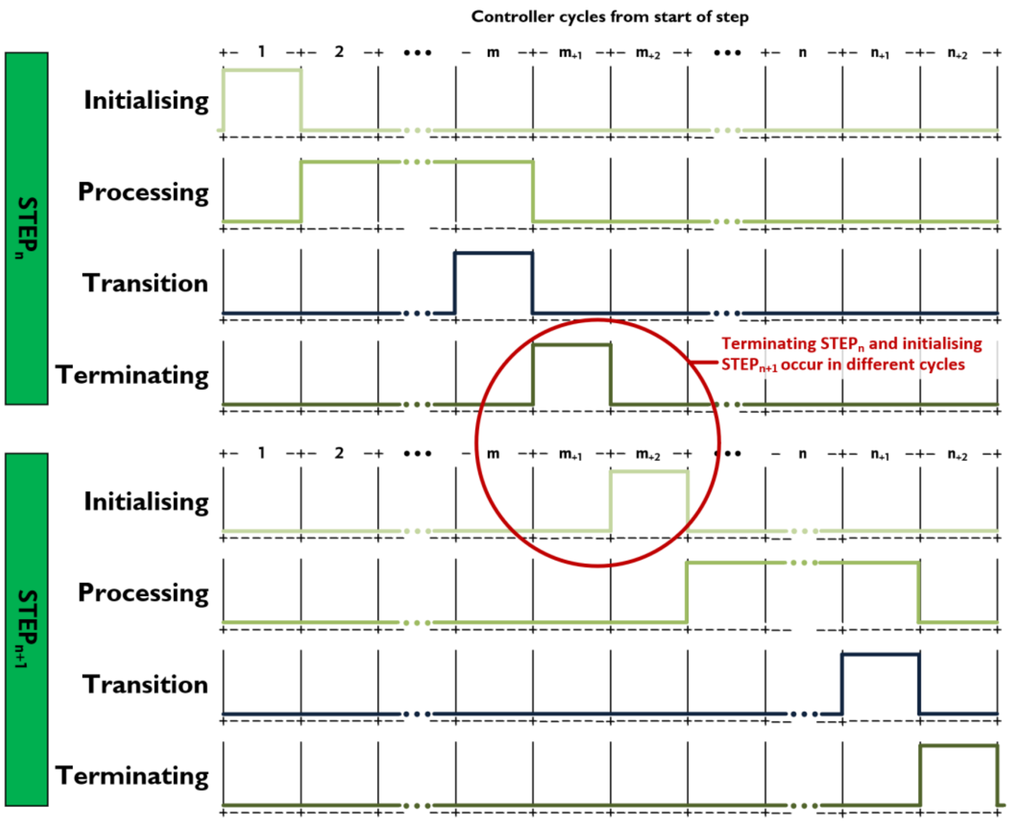


Figure 9.15 Non-IEC compliant phase timing

(3) Here, it can be seen that the *terminating* phase of $step_n$ is concluded the cycle before the *initialisation* phase of $step_{n+1}$.

(4) Both the IEC 61131-3 *[Ref. 012]* compliant and the non-IEC compliant versions are offered as part of the PAL sequence software. This satisfies the requirement specified in the User Requirement Specification *[Ref. 003];* whilst providing the more conventional, and widely used, implementation as well.

9.4 Automatic step timing functions

(1) Every step within a sequence has two timers that operate automatically:

- A step duration timer
- A step delay timer

(2) The step duration time is a measure of how long the particular step has been active, it counts up from zero (from when the step first became active) in 100 ms intervals.

(3) The step duration counter is stored as a `real` variable that measures the current time the step has been active in seconds (accurate to 0.1 of a second).

(4) The step duration timer can be used to trigger a transition (for example, a step could transition to the next step if a particular level is reached or if the step has been active for a specified time).

(5) The step delay timer ensures that the step will remain active for a minimum period of time (given in the step delay timer).

(6) The transition conditions for a step will not be evaluated until the step delay timer has counted down to zero.

(7) The step delay timer can be specified for any step and is again a `real` variable that specifies the minimum time the step will be active in seconds (accurate to 0.1 second). The step delay timer counts down from the specified value. If the step delay timer is set to zero (the default value), there is will be no delay associate with that step.

9.5 Manual modes of operation

(1) All sequences have both a manual and a semi-manual mode. Both allow the operator to take control of the sequence.

9.5.1 Semi-manual mode

(1) In semi-manual mode, the sequence will not automatically issue any commands, it will simply wait at the point where the command would have been issued and wait for the operator to issue the command manually.

(2) Once the command is issued, the sequence will continue automatically through the next state (carrying out steps and transmissions automatically), until it again reaches the point where a command is needed to progress to the next state.

(3) At any point, the operator can issue a command to divert the sequence to another state (or even back to a previous state).

9.5.2 Full manual mode

(1) Full manual mode provides all the same features as semi-manual mode, however, manual intervention is required at each step, to activate the transition conditions. It effectively allows the operator to *single-step* through a sequence and lets the operator choose which transition is activated after each step.

(2) Full manual mode also allows the operator to jump to a particular step in a sequence.

BLANK PAGE

10 Supervisory system user interface

(1) All physical equipment (valves, drives, instrumentation &c.) connected to a Controller usually have some form of graphical representation on a supervisor system such as a SCADA or HMI[16]. While these systems are outside the scope of this Project, the interface between these systems and the PAL software modules is not; and this must be clearly defined in order to provide the necessary signals to display and interact with the any supervisory system.

(2) The interface between a supervisory system and the PAL software modules is defined in this section; it includes example graphical arrangements that are compliant with the data available from each type of device.

(3) The interface for each of the different types of equipment will all be different (the interface to an instrument will for example be completely different to that of a valve); however, a commonality of approach (and where possible, signals) is adopted to give consistency to these interfaces, for example, where devices have a manual mode, the same signal name will be used across all devices and the mechanisms of operation and selection will be as common as possible.

(4) Additionally, signals of similar type (alarms, device status, operating modes &c) will have common prefixes and grouping to allow the type of signal to be readily identified (for example, status signals are prefixed `status_`, alarms and warnings prefixed `msg_` and operating modes prefixed `mode_`).

---

16 SCADA (supervisory control and data acquisition) system is a computer system used to gather and analyse real time data from the control system, it will display the status of the equipment (graphically showing if a valve is opened or closed, if a drive is running, instrument readings &c.), it will show alarm and warnings and will allow the operator to issue commands to the control system (start a sequence, take manual control of a device &c.).

HMI (human machine interface) is generally a panel mounted computer-based system similar in functionality to a SCADA system, but generally more restricted in its facilities and capabilities.

Collectively SCADA and HMI systems are referred to as supervisory systems.

(5) Figure 10.1 shows an example of how a PAL mimic is expected to look:

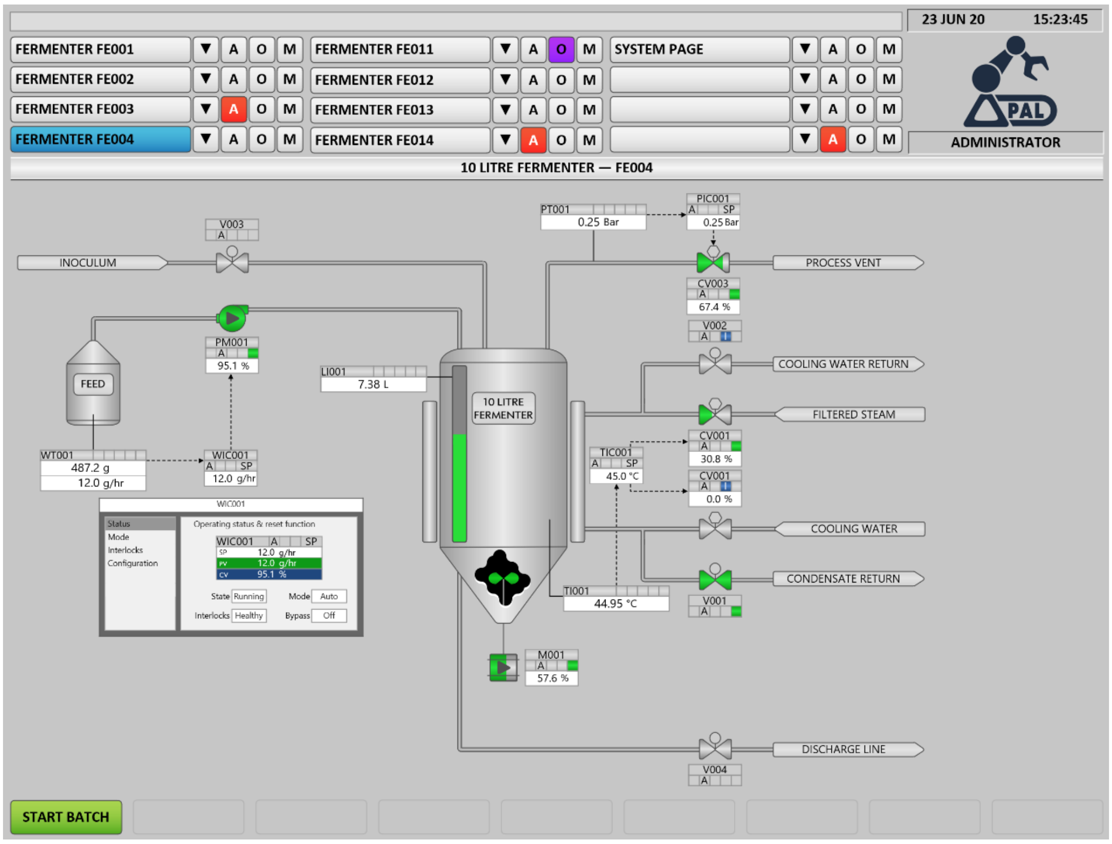


Figure 10.1 Example supervisory system graphical mimic

(6) Graphical mimics have several aspects:

- There are fixed graphics that are not animated (typically, pipe-work, tanks, vessels, labelling &c.)
- There are dynamic objects (valves, drives, instruments &c.)
- There are navigation areas (the buttons at the top of the page) that allow the operator to select different parts of the plant
- Thera are command areas (the buttons at the bottom) that allow the operator to perform some action (start batch in the example)

(7) The PAL has specific requirements in terms of graphical objects.

10.1 Scope restrictions with the PAL

(1) The development of supervisory systems (SCADA and HMI) is not within the scope of this Project. However, consideration has been given to the nature of the interface between the PAL software modules and any such supervisory system.

(2) It is anticipated that a future project will undertake the development of a supervisory system and that this system will utilise the PC based WinCC Professional application available within the TIA Portal software.

(3) To this end, the PAL expects the supervisory system to interface with the PAL software modules in a particular way and to have specific graphical objects that link correctly with the standard modules.

(4) The objects described in the following sections demonstrate how the PAL software expects a supervisory system to be configured and the facilities and functions listed here establish the full functions that would be available to an operator via that supervisory system.

(5) The objects listed here and the design implications inherent within them are all capable of being implemented by the WinCC Professional application, this being the baseline system for any such development. This does not preclude other supervisory systems being used — however if such systems have restricted capabilities compared with the WinCC system, the full functionality of the PAL software modules may not be available to the operator.

(6) All the graphical objects and mimics listed here are compliant with the current engineering standards for supervisory systems specified in the EEMUA[17] 201 *[Ref. 016]* standard for Process Plant Control.

(7) All alarm handling and reporting capabilities listed here are compliant with the current standards for such mechanisms: the EEMUA 191 *[Ref. 015]* Guide for Alarm Systems.

---

17 EEMUA — Engineering Equipment and Materials Users' Association

10.2 Symbols block icons and faceplates

(1) All plant equipment whether devices that can be operated by the Controller (valves, drives, motors, pumps &c.) or pseudo-devices such as PID loops (these are internal constructs of the Controller, but act as devices in their own right) or instruments that are read by the Controller, require an operator interface, the operator must be able to see what the devices are doing or what the instruments are reading, and where necessary take control of those devices.

(2) The PAL achieves these requirements through the use of *symbols* and *block icons*, two such groups of symbol and block icon are shown below:

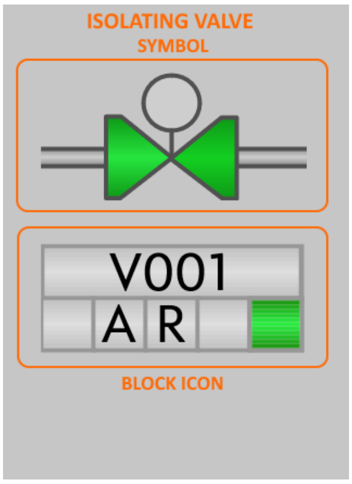


Figure 10.2 Isolating valve symbol and block icon

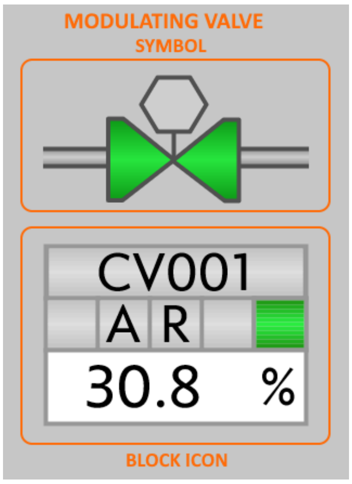


Figure 10.3 Modulating valve symbol and block icon

(3) The symbol provides a graphical representation of a device and what state it is in (open, closed, fault &c.), the block icon provides additional information about the device (operating modes, energised state &c.).

(4) Examining these in turn:

10.2.1 Symbols

(1) Symbols are animated objects that represent a particular device and visually indicate what the device is doing (what state it is in; e.g. for a valve this could be opened, closed, in fault &c.).

(2) Each symbol is designed to link directly with the static and dynamic data that is applicable to a particular standard module (typically a device driver).

(3) All the symbols listed here are based on the standard process and instrumentation diagram (P&ID) symbols.

(4) All the symbols shown here are compliant with EEMUA 201 *[Ref. 016]* standard for Process Plant Control; this dictates that objects appear in a grey colour when inactive and are highlighted only when the device is in a non-passive state (i.e. when a valve is energised or a drive is running), the PAL uses muted greens to indicate open and running states and muted reds for fault conditions.

(5) This section lists all the symbolic representations available to the standard device drivers listed in §§ 8.7, 8.8 and 8.9.

(6) Multiple symbols are available for each module, this is particularly true of drives, for example, a direct online drive may be a pump, motor, compressor, &c. it may also be a completely different form of device (such as a heating element), these all however, operate in an identical fashion to a direct online drive.

(7) Similarly, the isolating valve module may utilise normally closed or normally open symbology, or it may use a motorised valve symbol.

(8) The following sections show the most common symbols for the device driver modules.

Analogue instruments

(9) Generally, analogue instruments are represented by block icons (see § 10.2.2) rather than symbols, there are exceptions when showing values associated with vessels (level, pressure &c.), here it is often necessary to give a dynamic, animated indication of the property, this can be seen in Figure 10.1, where the tank level is shown graphically as a green bar rising vertically.

(10) Where instruments have alarm and warning points, these can also be shown:

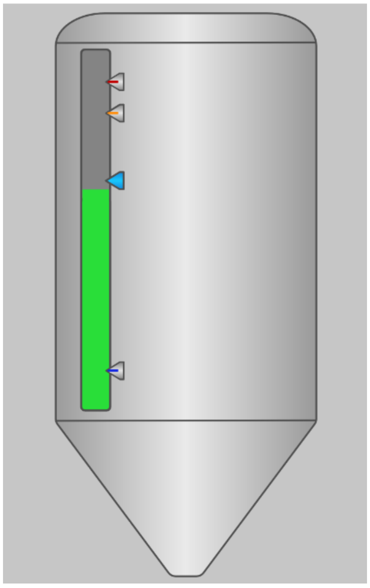


Figure 10.4 Analogue instrument symbols

(11) The alarm and warning points (the triangles in Figure 10.1) are dynamically positioned relative to the bottom of the bar graph. The actual position is determined by the alarm and warning setpoint values specified in the individual module data.

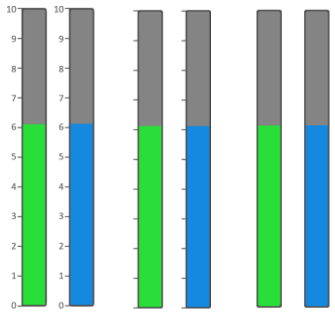
Analogue instrument symbols		FC02001_StdInstAnalogRead/ FC02011_StdInstRealValRead	
Scaled Graduated Clean		SIGNAL	DESCRIPTION
		actual_Vale	Scaled value
Inactive	Active	SIGNAL	DESCRIPTION
		Status_AlmH	Alarm high condition
		Status_WrnH	Warning high condition
		Status_Desc	Any message condition (or threshold)
		Status_WrnL	Warning low condition
		Status_AlmL	Alarm low condition

Table 10.1 Symbols — Analogue instruments

Digital instruments

(12) Digital instruments are represented by block icons (see § 10.2.2) rather than symbols; occasionally, where necessary, the state of the instrument can be graphically represented using the alarm and warning condition symbols specified for the analogue instruments.

Isolating valves

Isolating valve symbols				FC11001_StdDevValveIsol	
Standard NC valve	Standard NO valve	Motorised NC valve	Motorised NO valve	SIGNAL	DESCRIPTION
				Status_Closed	Closed
				Status_Opening	Opening
				Status_Opened	Opened
				Status_Closing	Closing
				Status_Fault	Fault (valve body shows state)
				N/A	Loss of communications

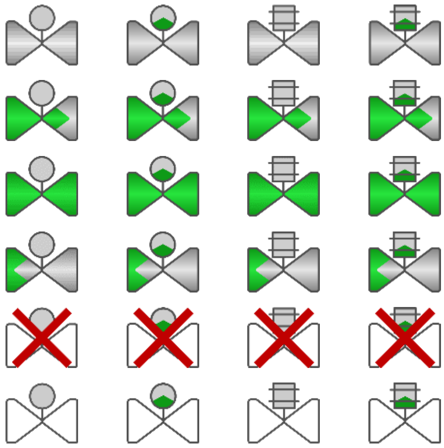


Table 10.2 Symbols — Isolating valve NC — Normally closed NO — Normally open

3-way isolating valves

3-way isolating valve symbols				FC11011_StdDevValve3Way	
				SIGNAL	DESCRIPTION
				Status_PortD	Closed
				Status_PortDtoE	Opening
				Status_PortE	Opened
				Status_PortEtoD	Closing
				Status_Fault	Fault (valve body shows state)
				N/A	Loss of communications

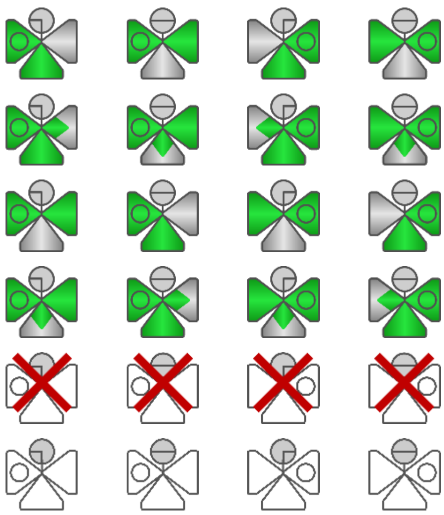


Table 10.3 Symbols — 3-way isolating valve E = Energised, D = De-energised

(14) Three-way valves have many orientations, only a selection are shown here, the head of the valve shows the de-energised path, the port with a circle is the common port.

Bistable valves

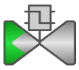
Bistable (motorised) valve symbols	FC1101_StdDevValveBi	
Motorised bistable	SIGNAL	DESCRIPTION
	Status_Closed	Closed
	Status_Opening	Opening
	Status_Opened	Opened
	Status_Closing	Closing
	Status_Indeterminate	Indeterminate (unknown) state
	Status_Fault	Fault (valve body shows state)
	N/A	Loss of communications

Table 10.4 Symbols — Bistable (motorised) valve

Modulating valve

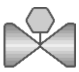
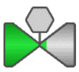
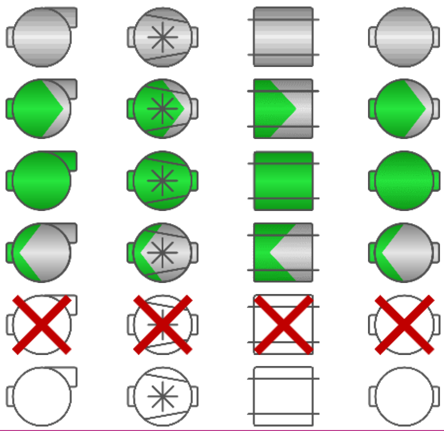
Modulating valve	FC1501_StdDevValveMod	
Modulating (control) valve	SIGNAL	DESCRIPTION
	Status_Closed	Closed (0%) or closed limit
	Status_PartOpen1	Partially open (≤20%)
	Status_PartOpen2	Partially open (20-40%)
	Status_PartOpen3	Partially open (40-60%)
	Status_PartOpen4	Partially open (60-80%)
	Status_Opened	Opened (≥80%) or opened limit
	Status_Fault	Fault (valve body shows state)
	N/A	Loss of communications

Table 10.5 Symbols — Modulating valve

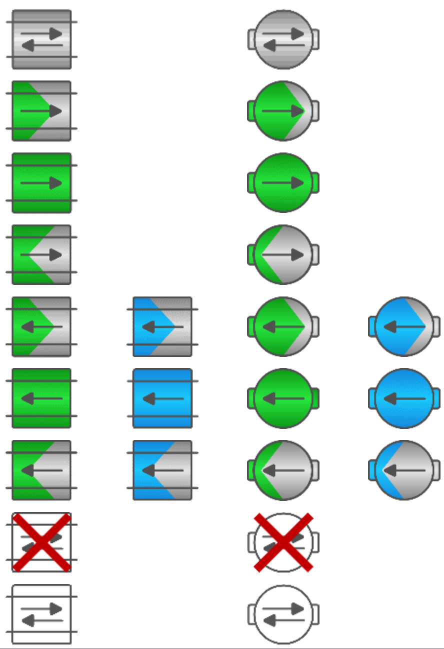
Direct online drive



Direct online drive symbols				FC12001_StdDevDriveDOL	
Pump	Blower	Motor	General	SIGNAL	DESCRIPTION
				Status_Stopped	Stopped
				Status_Starting	Starting
				Status_Running	Running
				Status_Stopping	Stopping
				Status_Fault	Fault (body shows state)
				N/A	Loss of communications

Table 10.6 Symbols — DOL drive Note: starting and stopping states are momentary unless slow ram times are used

Reversing direct online drive



Reversing direct online drive symbols				FC12011_StdDevDriveDOLRev	
Motor	Alternate	General	Alternate	SIGNAL	DESCRIPTION
				Status_Stopped	Stopped
				Status_StartingF	Starting forwards
				Status_RunningF	Running forwards
				Status_StoppingF	Stopping forwards
				Status_StartingR	Starting reverse
				Status_RunningR	Running reverse
				Status_StoppingR	Stopping reverse
				Status_Fault	Fault (body shows state)
				N/A	Loss of communications

Table 10.7 Symbols — Reversing DOL Note: starting and stopping states are momentary unless slow ramp times are used

Bistable drive

Direct online bistable drive symbols	FC12101_StdDevDriveBi
Bistable drives generally use the symbols for the FC12001_StdDevDriveDOL (direct online drive) module — it is not generally necessary to identify a drive as a bistable device within a supervisory system symbol	

Table 10.8 Symbols — Bistable DOL drive

Bistable reversing drive

Direct online bistable reversing drive symbols	FC12111_StdDevDriveBiRev
Bistable reversing drives generally use the symbols for the FC12002_StdDevDriveRevDOL (direct online reversing drive) module — it is not generally necessary to identify a drive as a bistable device within a supervisory system	

Table 10.9 Symbols — Bistable DOL drive

Variable speed drive

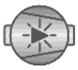
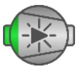
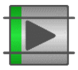
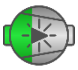
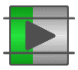
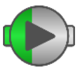
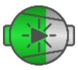
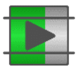
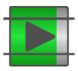
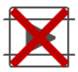
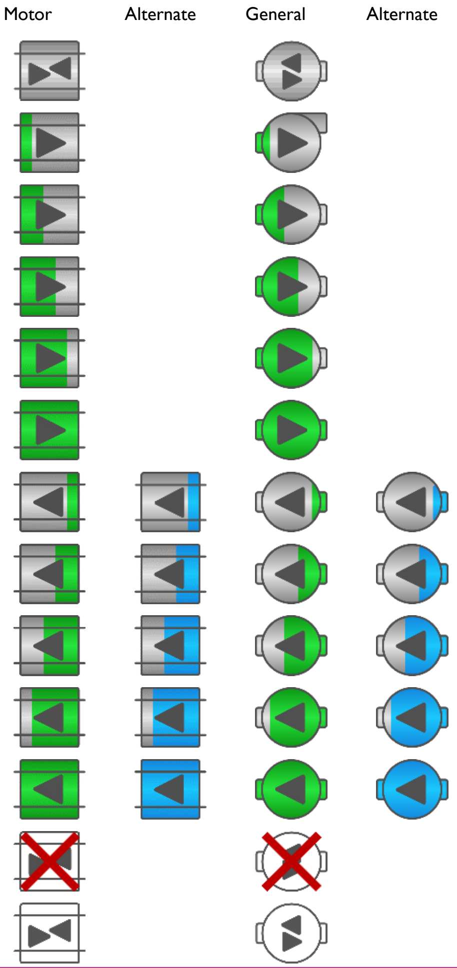
Variable speed drive symbols				FC12501_StdDevDriveVSD	
Pump	Blower	Motor	General	SIGNAL	DESCRIPTION
				Status_Stopped	Stopped
				Status_PartRun1	Running (≤20% speed)
				Status_PartRun2	Running (20-40% speed)
				Status_PartRun3	Running (40-60% speed)
				Status_PartRun4	Running (60-80% speed)
				Status_Running	Running (≥80% speed)
				Status_Fault	Fault (body shows state)
				N/A	Loss of communications

Table 10.10 Symbols — Variable speed drive

Reversing variable speed drive

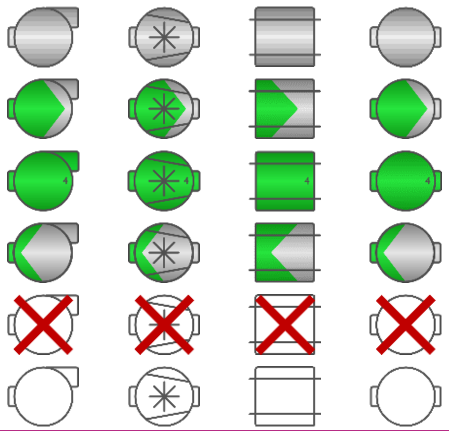
Reversing variable speed drive symbols — **FC12511_StdDevDriveVSDRev**



SIGNAL	DESCRIPTION
Status_Stopped	Stopped
Status_PartRunF1	Running forward (≤20% speed)
Status_PartRunF2	Running forward (20-40% speed)
Status_PartRunF3	Running forward (40-60% speed)
Status_PartRunF4	Running forward (60-80% speed)
Status_RunningF	Running forward (≥80% speed)
Status_PartRunR1	Running reverse (≤20% speed)
Status_PartRunR2	Running reverse (20-40% speed)
Status_PartRunR3	Running reverse (40-60% speed)
Status_PartRunR4	Running reverse (60-80% speed)
Status_RunningR	Running reverse (≥80% speed)
Status_Fault	Fault (body shows state)
N/A	Loss of communications

Table 10.11 Symbols — Reversing variable speed drive

Multiple speed drive



Multiple speed drive symbol				FC12601_StdDevDriveMSD	
Pump	Blower	Motor	General	SIGNAL	DESCRIPTION
				Status_Stopped	Stopped
				Status_Starting	Starting
				Status_Running Status_Speed	Running (small number indicates speed)
				Status_Stopping	Stopping
				Status_Fault	Fault (body shows state)
				N/A	Loss of communications

Table 10.12 Symbols — Multispeed Drive Note: starting and stopping states are momentary unless slow ram times are used

10.2.2 Block icons

(1) All devices have a block icon, the block icon identifies the device (by tag number, see § 6.2.1) and provides additional information about the device. In the case of analogue instruments, the primary use of the block icon is to display the value the instrument is reading.

(2) Block icons are located adjacent to any device symbol that may be in use; generally, block icons are positioned below the device in question and this is the preferred position. It is accepted however, that this is not always possible and it is permissible to position the block icon either above the device or to either side.

(3) There are generally multiple styles of block icons available to each device, these are of different size and complexity. For example, the PID loop block icon has three formats:

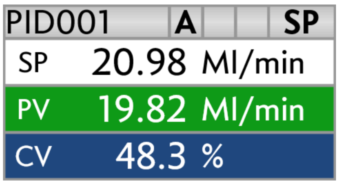


Figure 10.5 PID full block icon

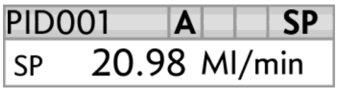


Figure 10.6 PID standard block icon

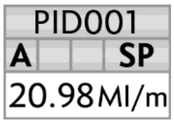


Figure 10.7 PID compact block icon

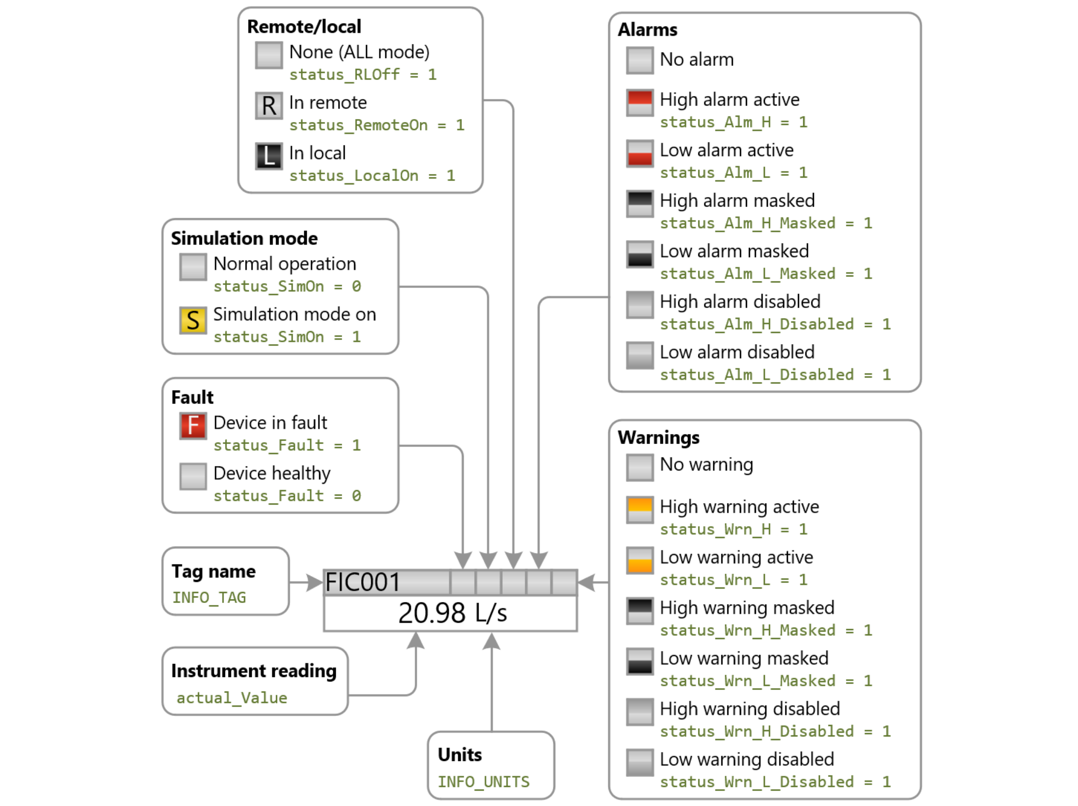
(4) The style of block icon is entirely at the user's discretion. Generally, however, whichever style is chosen should be applied to all similar objects on the graphical display page.

(5) Where alternative block icons exist, these are also shown in the following sections. Where the block icon display changes to reflect particular operating modes, these are also shown.

(6) Each block icon is designed to link directly with the static and dynamic data that is applicable to a particular standard module (typically a device driver or instrument block); where possible, the variables that drive the individual aspects of the block icon are listed — for certain block icons, these variables require further explanation, the details of which are contained in the Software Module Design Specification (SMDS) *[Ref. 008]* for the block in question.

Analogue instruments

Analogue instrument block icon	FC02001_StdInstAnalogRead/ FC02011_StdInstRealValRead



ALTERNATIVE BLOCK ICON STYLES

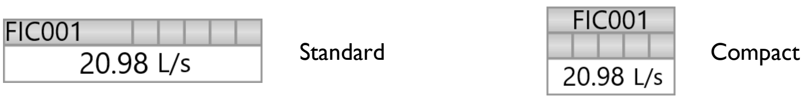
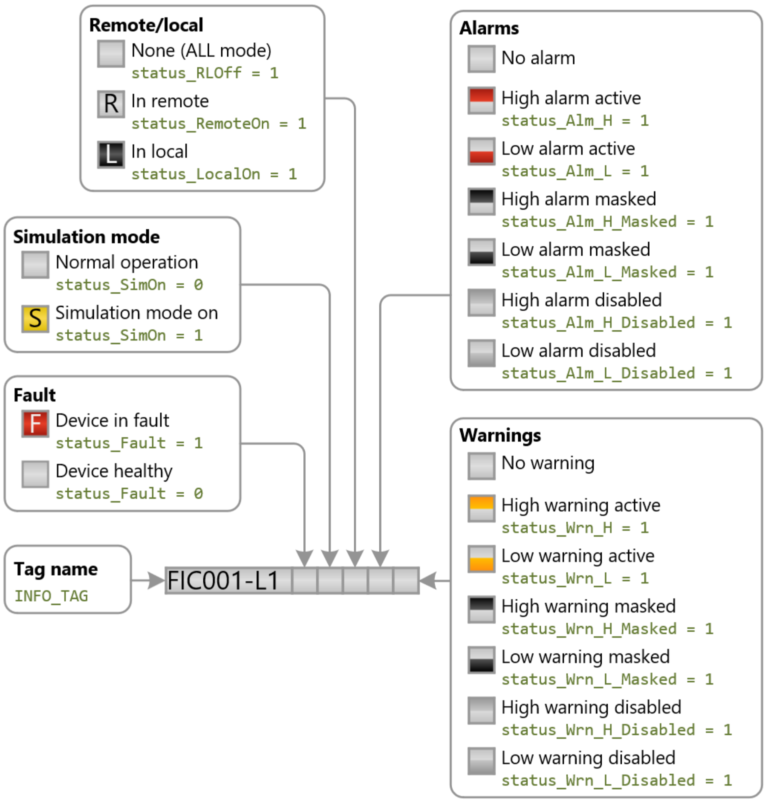


Table 10.13 Block icon — Analogue instruments

(7)

Analogue threshold block icon FC02101_StdInstRealLimit



ALTERNATIVE BLOCK ICON STYLES

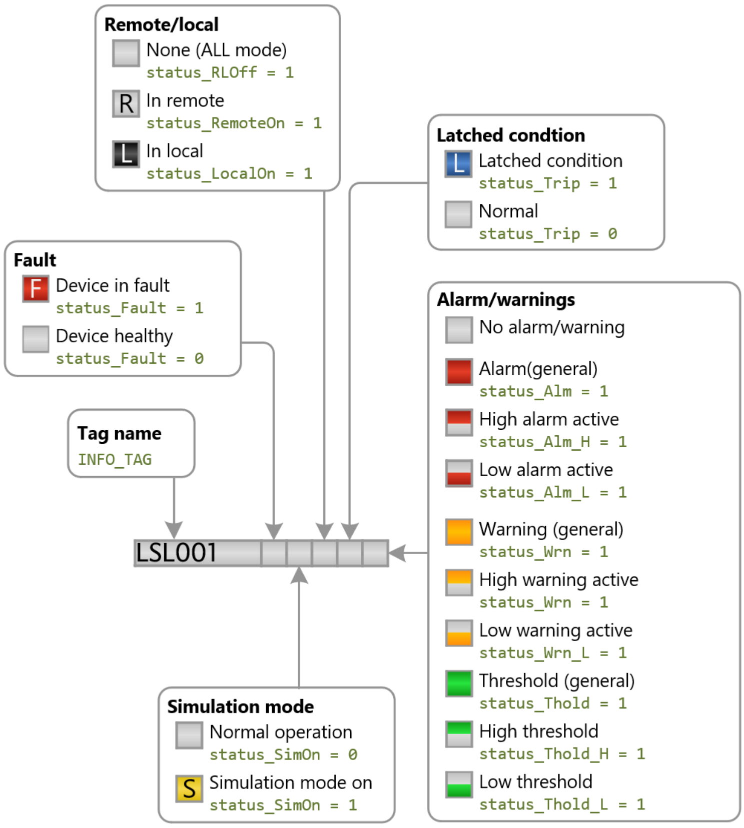
Standard

Compact

Table 10.14 Block icon — Analogue threshold

Digital instruments

Digital instrument/filter block icon FC02501_StdInstDigitalRead/FC02601_StdInstDigitalFilt



ALTERNATIVE BLOCK ICON STYLES

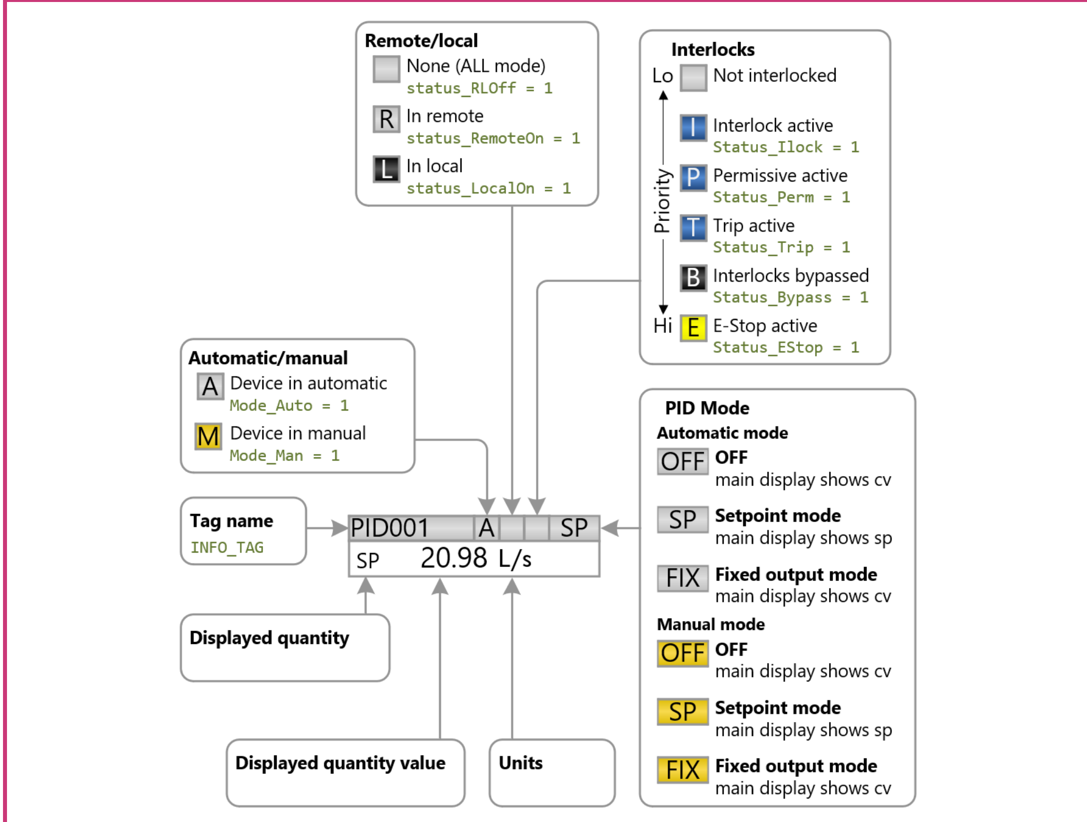
Standard

Compact

Table 10.15 Block icon — Digital instruments

Control loops

PID loop block icon — FC10001_StdDevPID_Standard/ FC10011_StdDevPID_Sched



ALTERNATIVE BLOCK ICON STYLES AND MODES

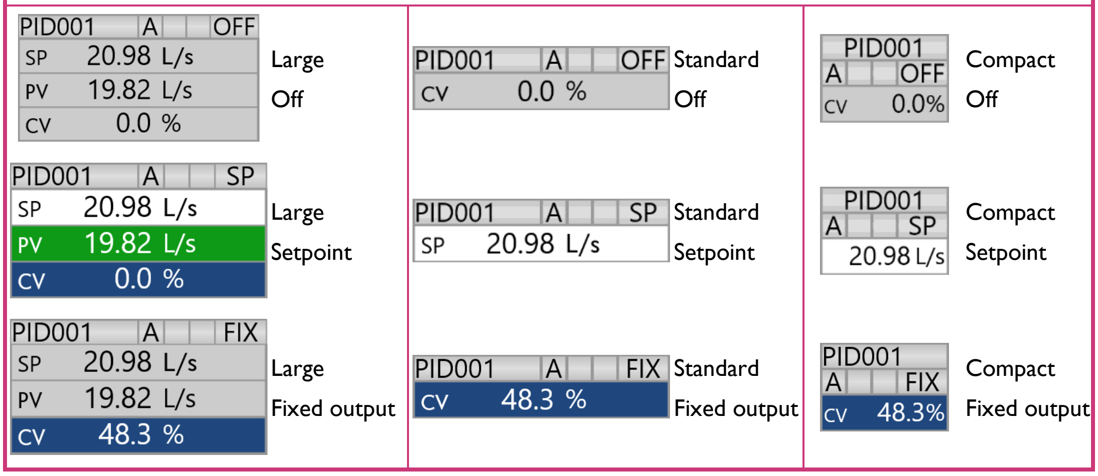


Table 10.16 Block icon — Control loops

Details of the displayed quantity, displayed quantity value and units variables are explained in the associated SMDS

Isolating valves

Isolating valve block icon	FC11001_StdDevValveIsol

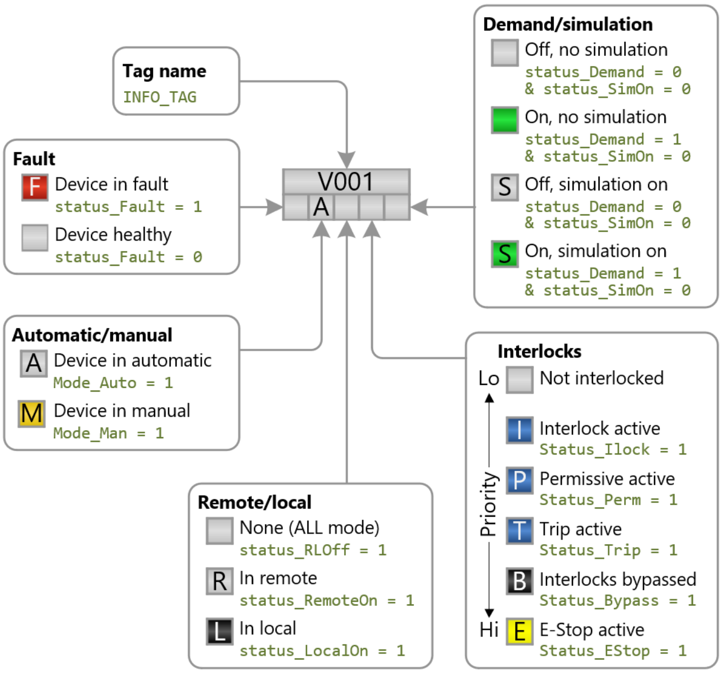


Table 10.17 Block icon — Isolating valve

3-way isolating valves

3-way Isolating valve block icon	FC11011_StdDevValve3Way
3-way isolating valves use the block icon for the FC11001_StdDevValveIsol (isolating valve) module	

Table 10.18 Block icon — 3-way isolating valve

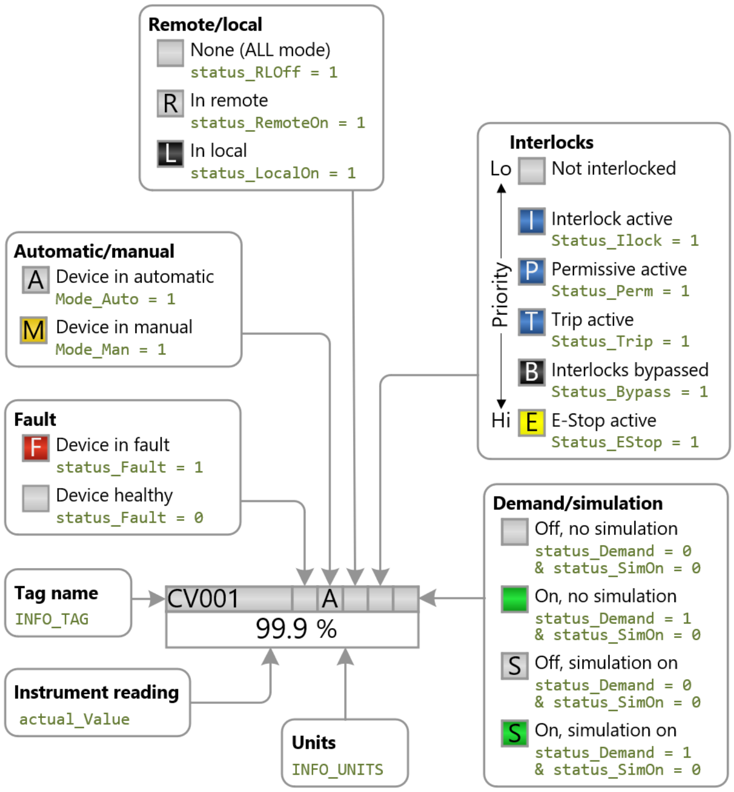
Bistable valves

Bistable (motorised) valve block icon	FC11101_StdDevValveBi
Bistable valves use the block icon for the FC11001_StdDevValveIsol (isolating valve) module	

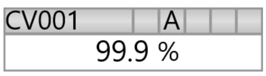
Table 10.19 Block icon — Bistable isolating valve

Modulating valve

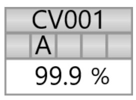
Modulating valve block icon FC1501_StdDevValveMod



ALTERNATIVE BLOCK ICON STYLES



Standard



Compact

Table 10.20 Block icon — Modulating valve

Direct online drive

Direct online drive block icon	FC12001_StdDevDriveDOL

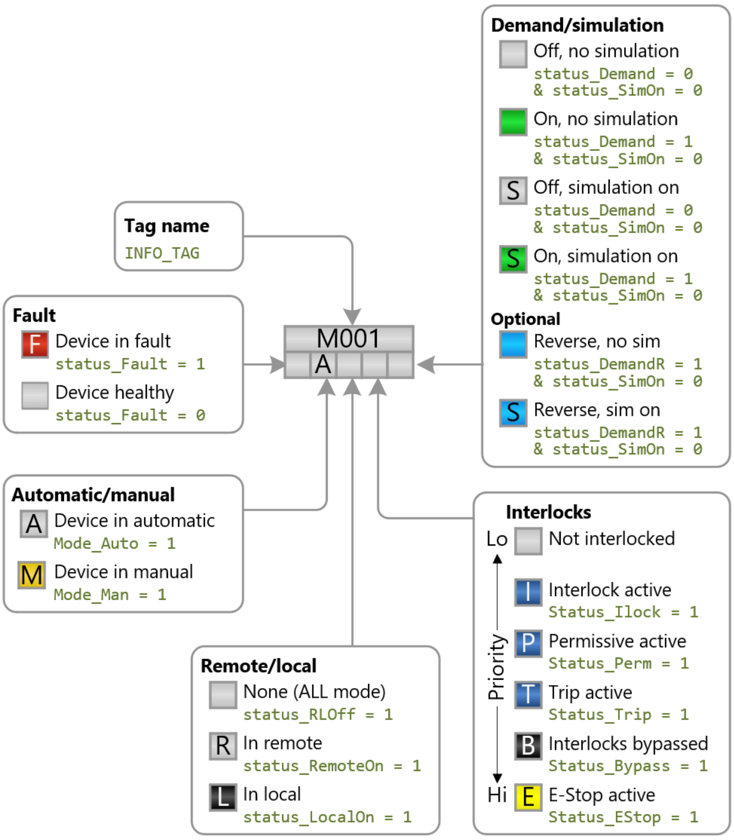


Table 10.21 Block icon — Direct online drive

Reversing direct online drive

Reversing direct online drive block icon	FC12011_StdDevDriveDOLRev
DOL reversing drives use the block icon for the FC12001_StdDevDriveDOL (direct online drive)	

Table 10.22 Block icon — Reversing DOL drive

Bistable drive

Direct online bistable drive block icon	FC12101_StdDevDriveBi
Bistable drives use the block icon for the FC12001_StdDevDriveDOL (direct online drive)	

Table 10.23 Block icon — Bistable drive

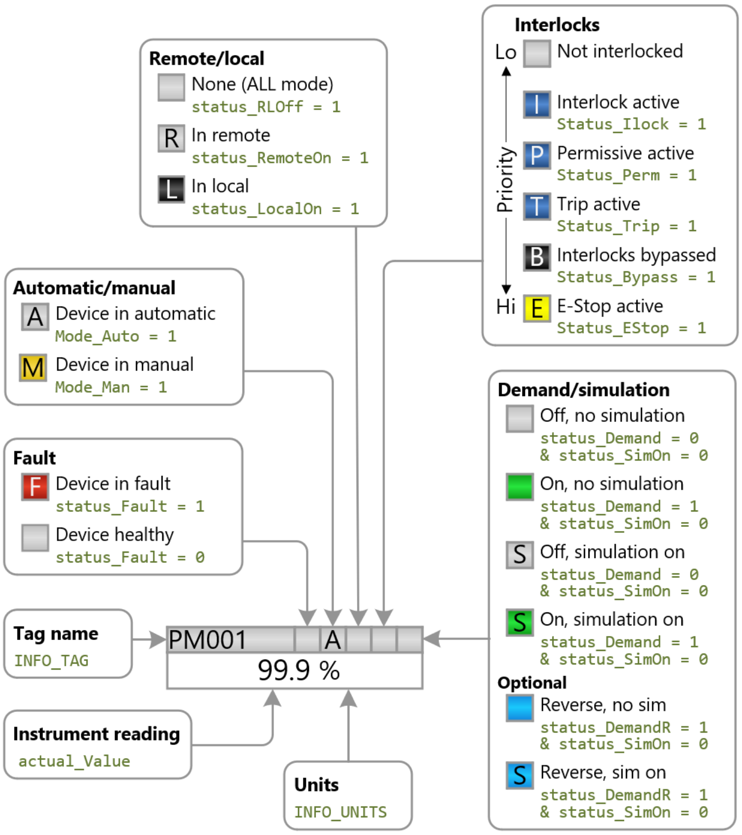
Bistable reversing drive

Direct online bistable reversing drive block icon	FC12111_StdDevDriveBiRev
Bistable reversing drives use the block icon for the FC12001_StdDevDriveDOL (direct online drive)	

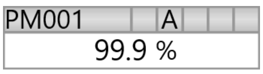
Table 10.24 Block icon — Bistable reversing drive

Variable speed drive

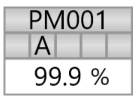
Variable speed drive block icon	FC12501_StdDevDriveVSD



ALTERNATIVE BLOCK ICON STYLES



Standard

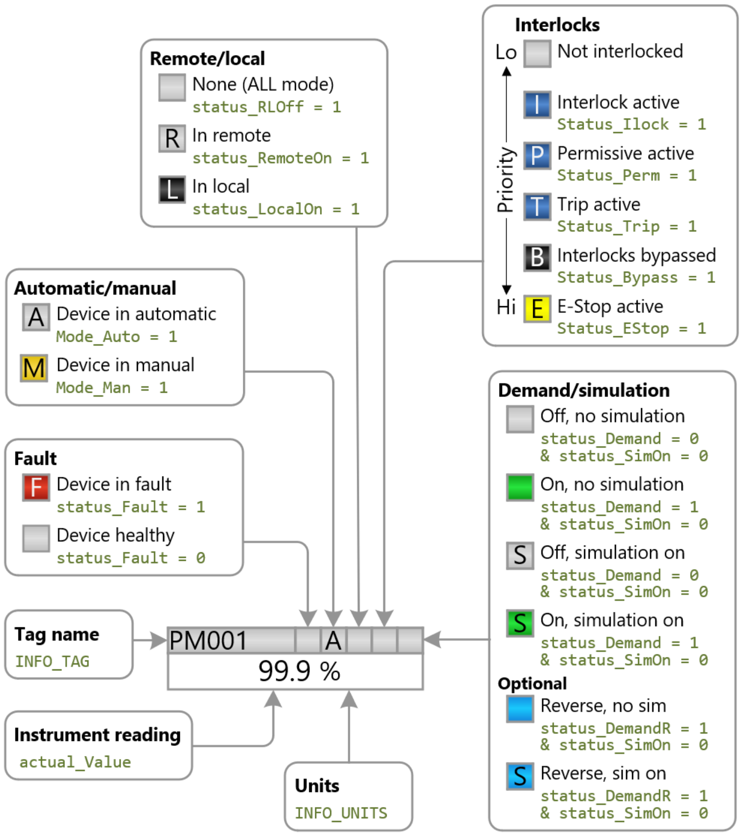


Compact

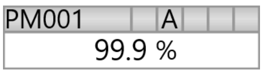
Table 10.25 Block icon — Variable speed drive

Reversing variable speed drive

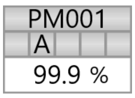
Reversing variable speed drive block icon FC12511_StdDevDriveVSDRev



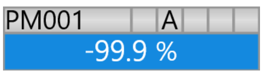
ALTERNATIVE BLOCK ICON STYLES



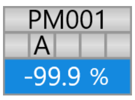
Standard



Compact



Standard
Running reverse

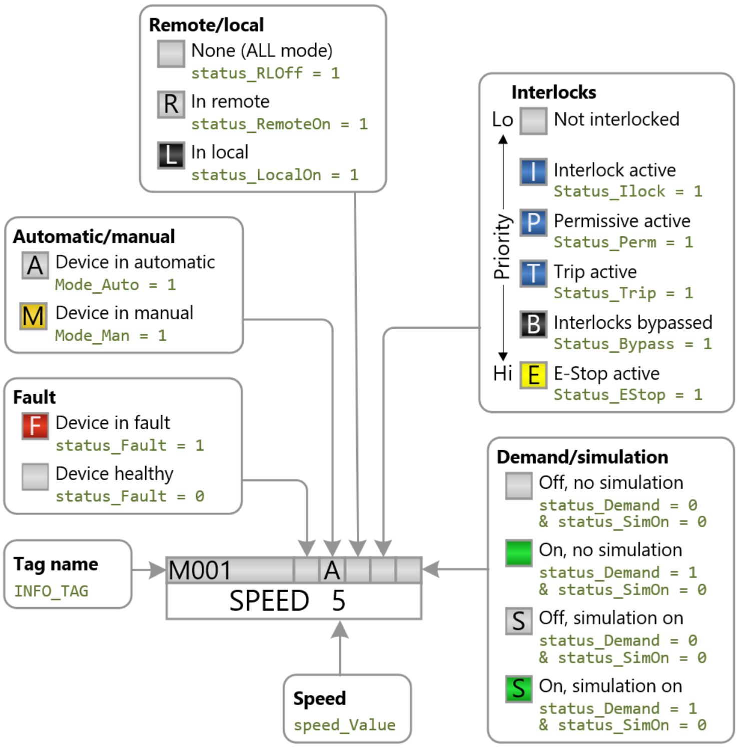


Compact
Running reverse

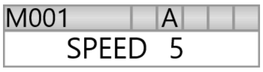
Table 10.26 Block icon — Reversing variable speed drive

Multiple speed drive

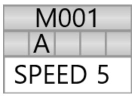
Multiple speed drive block icon FC12511_StdDevDriveMSD



ALTERNATIVE BLOCK ICON STYLES



Standard



Compact

Table 10.27 Block icon — Multiple speed drive

10.2.3 Faceplates

(1) All plant equipment (instruments, valves, drives, motors, pumps &c.) or pseudo-devices such as PID loops have selectable and configurable operating modes (e.g. manual mode, simulation mode &c.). These modes are optional (each mode can be disabled in the static data for the device); however, where these options are used, it is necessary to have an operator interface that allows the various modes to be selected.

(2) To this end, the modes are selected through the use of a *faceplate*, this is a pop-up window that appears on the supervisory system, overlaying the plant mimic. Each type of device has its own faceplate and multiple devices can have their faceplates open at the same time.

(3) An active faceplate can be considered to be a window in its own right and can be dragged to different positions on the screen (Figure 10.8).

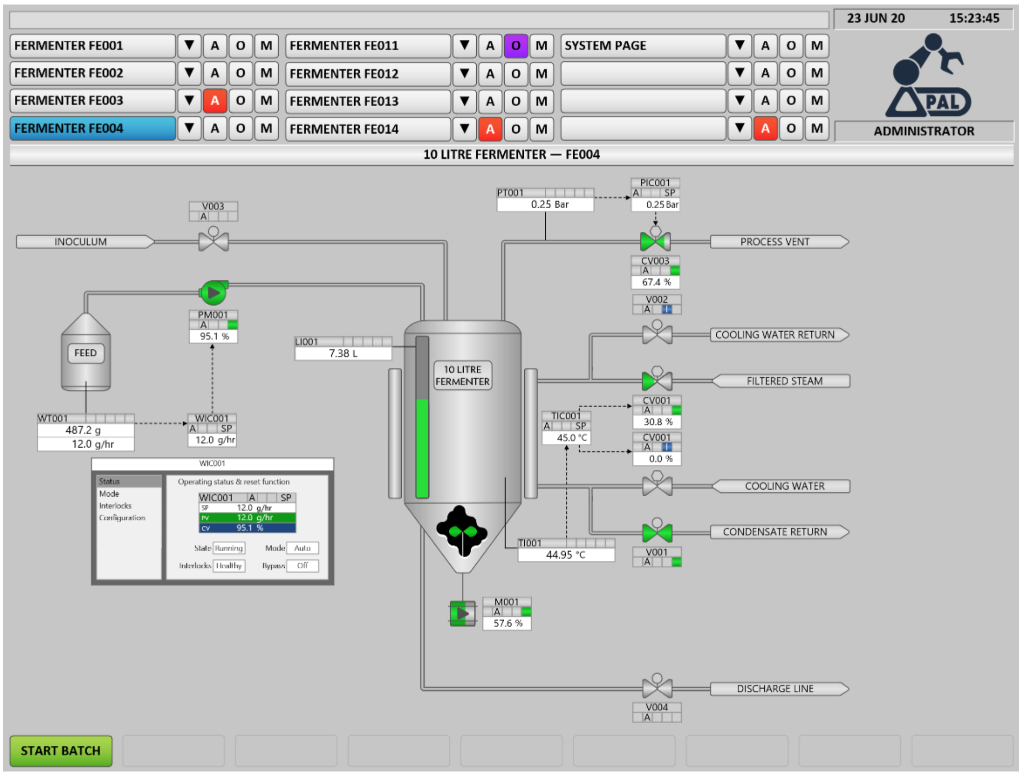


Figure 10.8 Example Faceplate

(4) Each device has the ability to disable the faceplate operation from within its configuration data within the Controller.

(5) Where a device permits a faceplate to be used, the supervisory system generally limits the access of a particular user to ensure that only specific user groups are able to operate particular faceplate functions.

(6) Faceplates typically have six functional areas:

① **Status**
Shows the status of the device and clearly identifies the selected operating modes

② **Mode**
Displays the operating modes available to the device and allows the operator to activate or deactivate any such modes

③ **Interlocks**
Shows the interlock states and allows the interlocks to be by-passed (if permitted)

④ **Simulation**
Allows the device to be switched to simulation mode and lets the operator select the various simulation modes

⑤ **Configuration**
Displays the primary configuration information for the device (operating times, alarm limits &c.) and in certain cases allows the operator the modify the values

⑥ **Messages**
Displays any alarms, warning or messages that may be active for the device

(7) The signals needed to operate the faceplates are provided by the dynamic and static data interfaces to the block. The detailed requirements for which are specified in the Software Module Design Specification (SMDS) *[Ref. 008]* for the relevant module.

(8) The following sections show typical examples of various types of faceplates (it should be noted that these are examples and the final faceplates may have minor differences. However, these examples remain representative of any final faceplates).

Analogue instruments

Analogue instrument faceplate	FC02001_StdInstAnalogRead/ FC02011_StdInstRealValRead

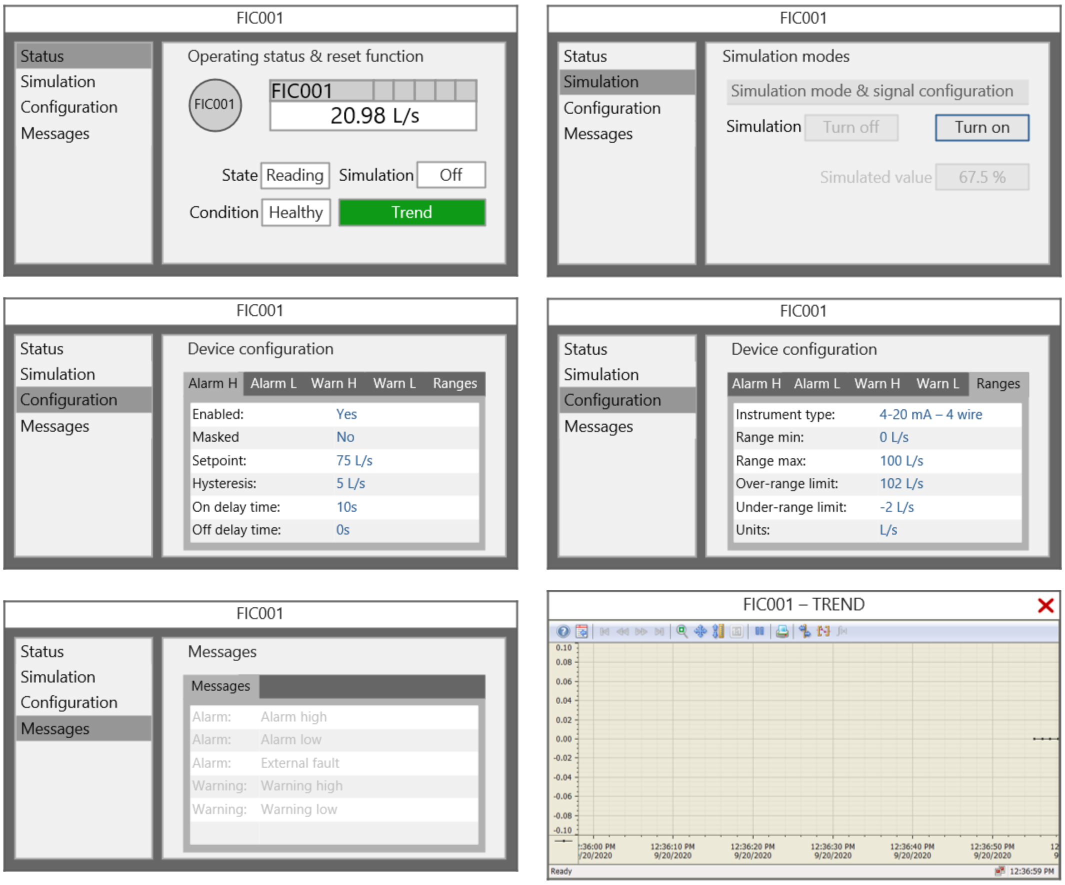


Table 10.28 Faceplate — Analogue instruments

Digital instruments

Digital instrument/filter faceplate FC02501_StdInstDigital/FC02601_StdInstDigitalFIlter

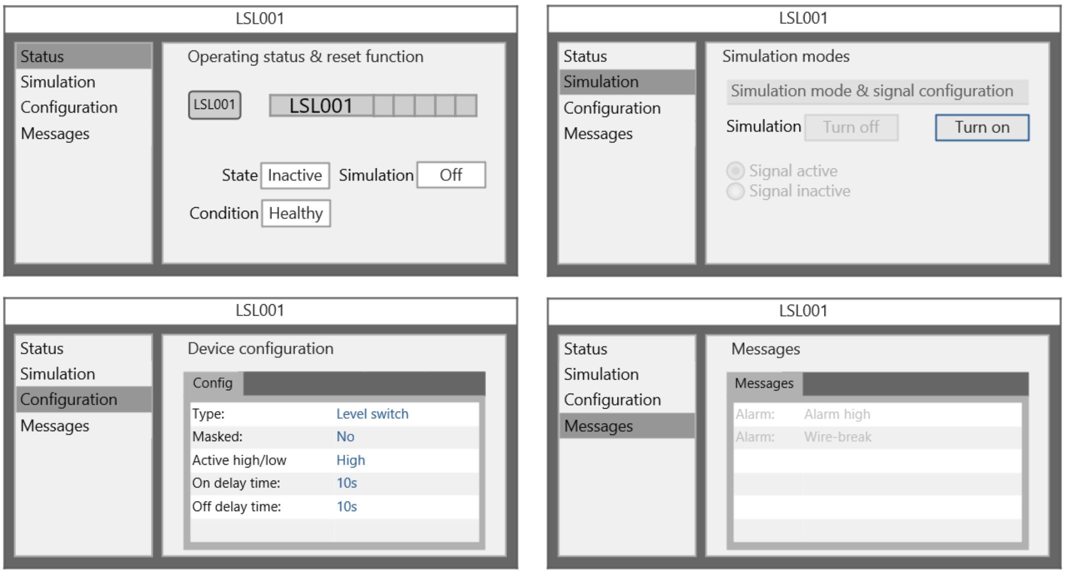


Table 10.29 Faceplate — Digital instruments

Control loops

PID loop faceplate FC10001_StdDevPID_Standard/ FC10011_StdDevPID_Sched

Table 10.30 Faceplate — Control loops

Isolating valves

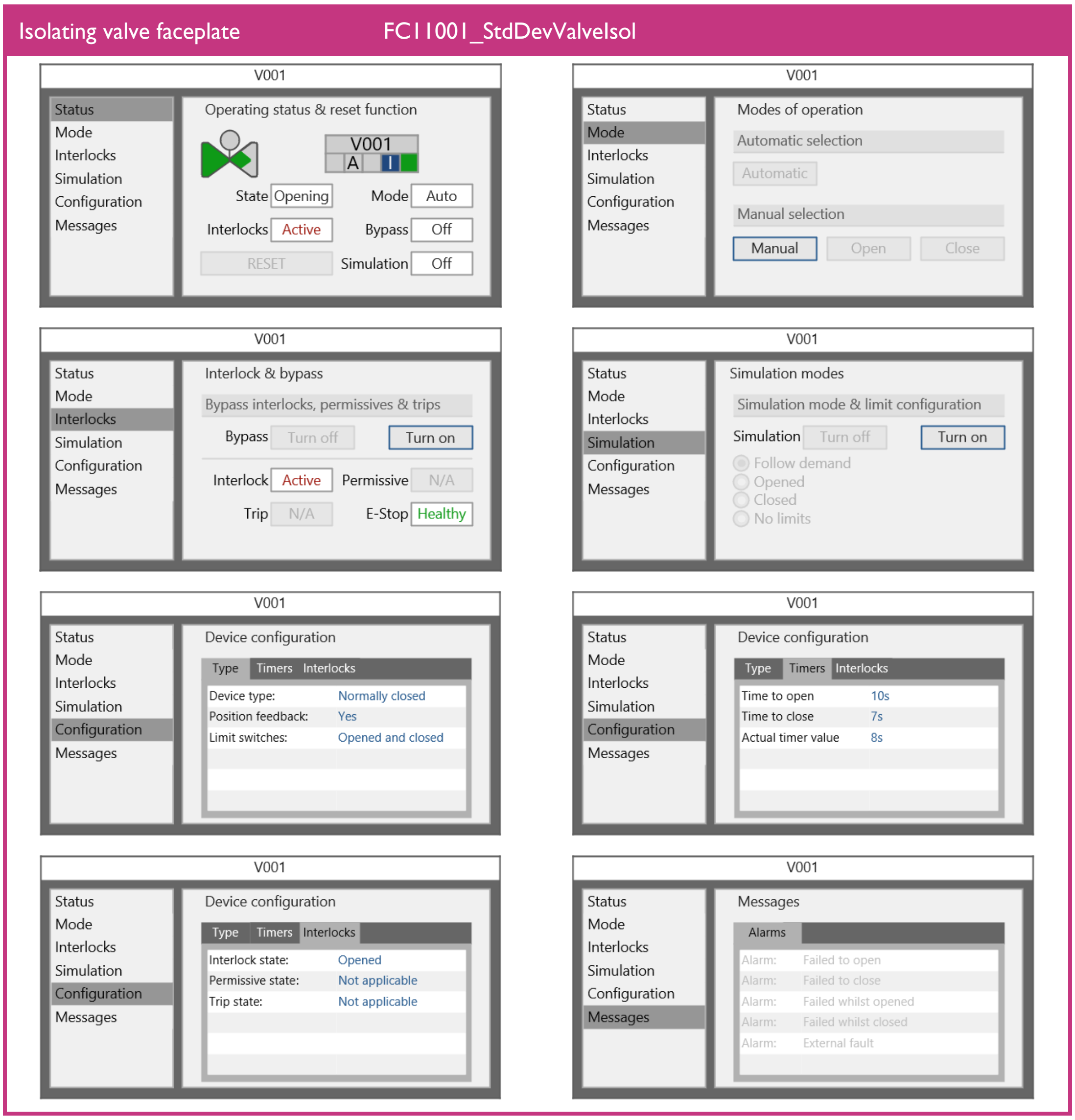


Table 10.31 Faceplate — Isolating valve

3-way isolating valves

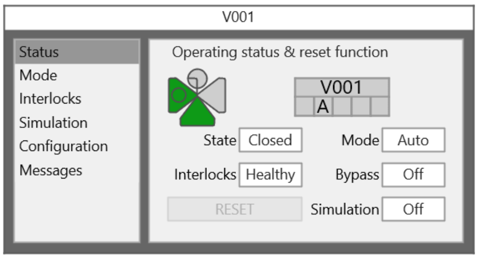
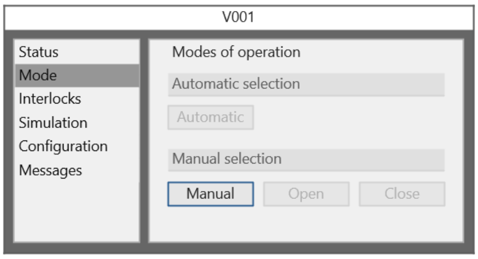
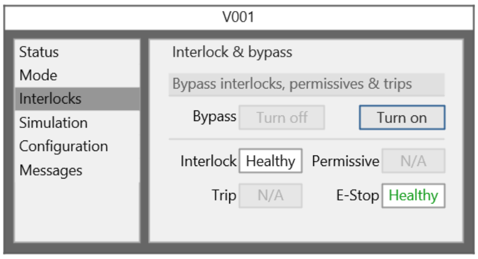
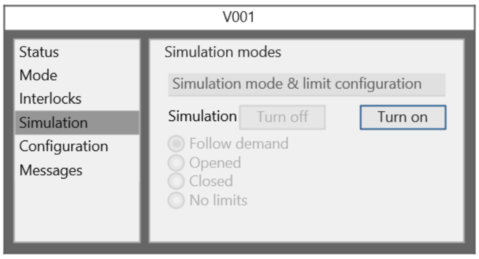
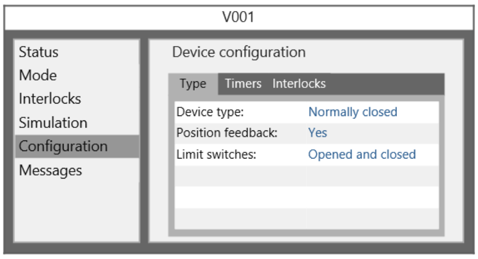
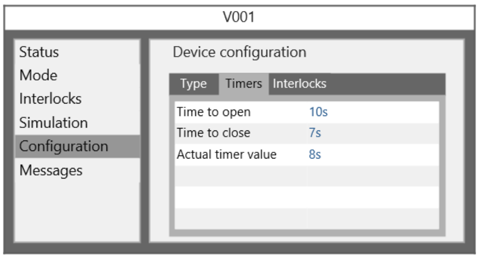
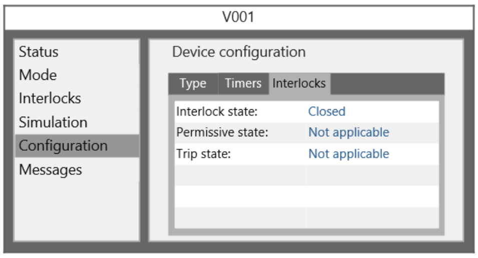
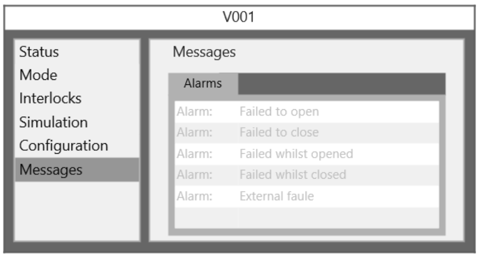
3-way Isolating valve faceplate	FC11011_StdDevValve3Way
	
	
	
	

Table 10.32 Faceplate — 3-way Isolating valve

Bistable valves

Bistable (motorised) valve faceplate	FC11101_StdDevValveBi
Bistable valves use the faceplate for the FC11001_StdDevValveIsol (isolating valve) module	

Table 10.33 Faceplate — Bistable Isolating valve

Modulating valve

Modulating valve faceplate	FC11501_StdDevValveMod

Table 10.34 Faceplate — Modulating valve

Direct online drive

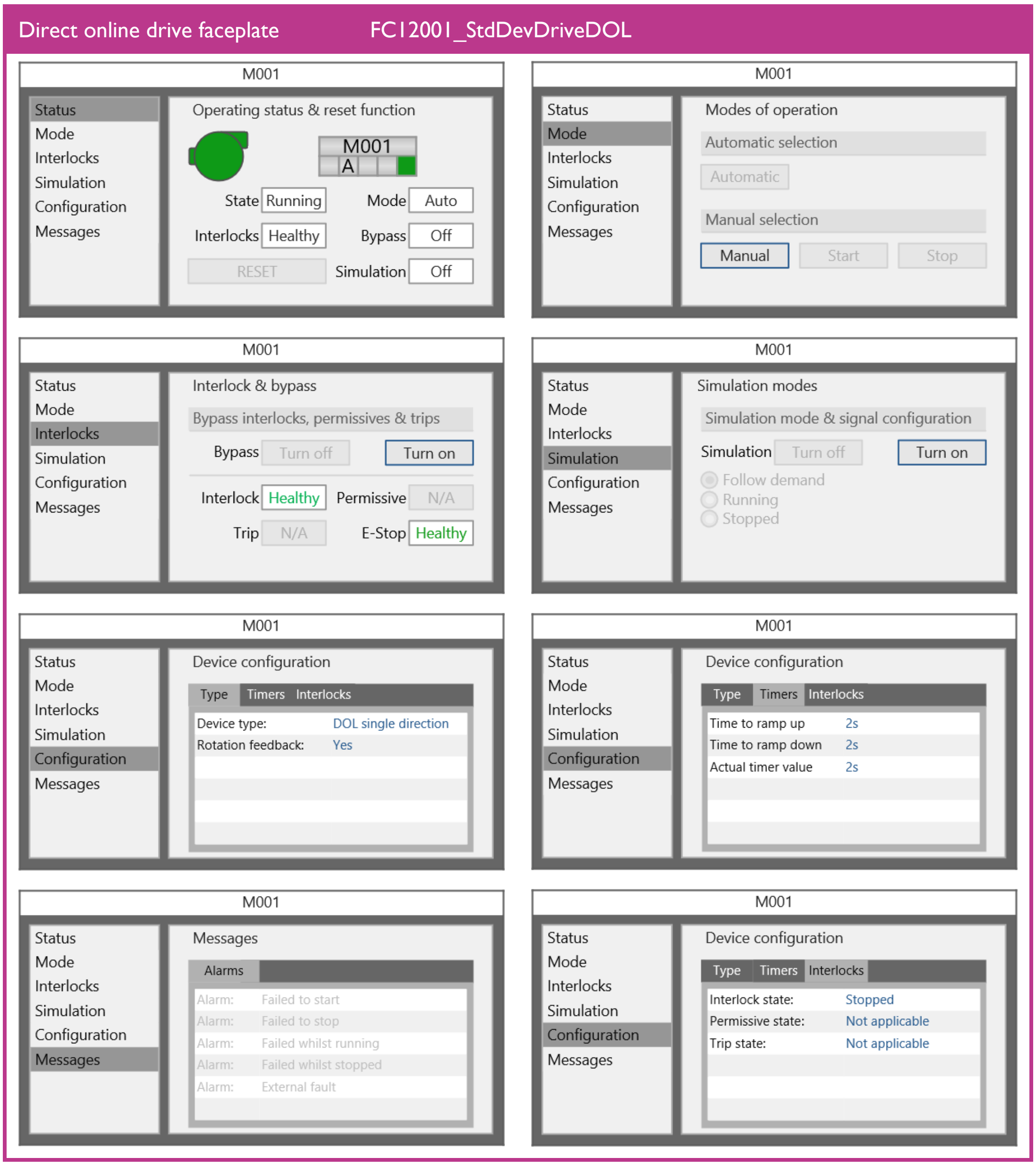


Table 10.35 Faceplate — Direct online drive

Reversing direct online drive

Reversing direct online drive faceplate	FC12011_StdDevDriveDOLRev

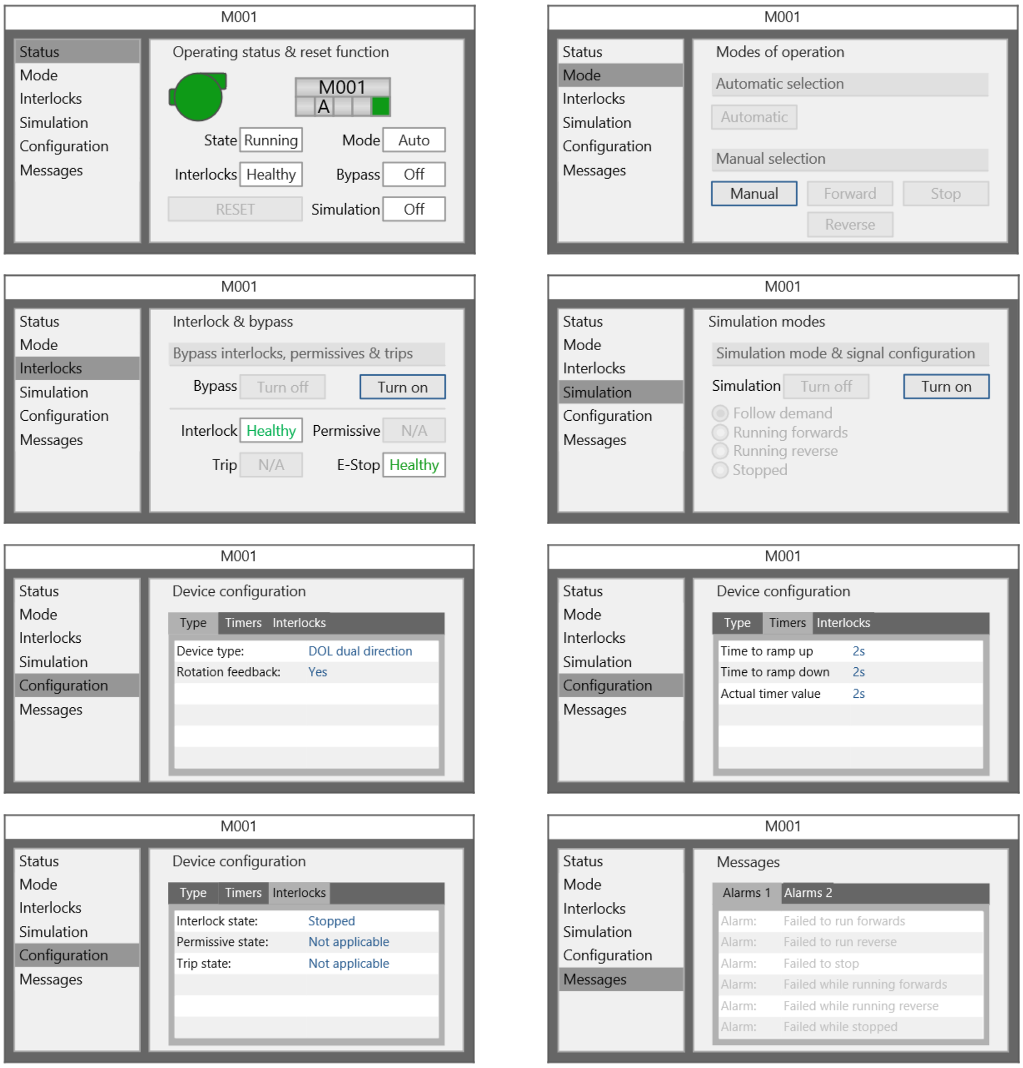


Table 10.36 Faceplate — Reversing DOL

Bistable drive

Direct online bistable drive faceplate	FC12101_StdDevDriveBi
Bistable drives use the faceplate for the FC12001_StdDevDriveDOL (direct online drive)	

Table 10.37 Faceplate — Bistable DOL drive

Variable speed drive

Variable speed drive faceplate FC12501_StdDevDriveVSD

Table 10.38 Faceplate — Variable speed drive

Reversing variable speed drive

Reversing variable speed drive faceplate FC12511_StdDevDriveVSDRev

Table 10.39 Faceplate — Reversing variable speed drive

Multiple speed drive

Multiple speed drive faceplates	FC12601_StdDevDriveMSD

Table 10.40 Faceplates — Multiple speed Drive

10.3 Graphical styles

(1) The PAL does not at this stage formally prescribe the graphical styles that must be used (although future projects may do so), this to a certain extent is dependent on the supervisory system being used. The PAL does require that all the graphical objects and mimics are compliant with the current engineering standards for supervisory systems specified in the EEMUA 201 *[Ref. 016]* standard for Process Plant Control.

(2) The graphical depictions shown in the previous sections (for symbols, block icons and faceplates) are accurate representations of what the graphical representation must accommodate as a minimum requirement.

(3) The following figures illustrate some of the different graphical approaches that may be taken:

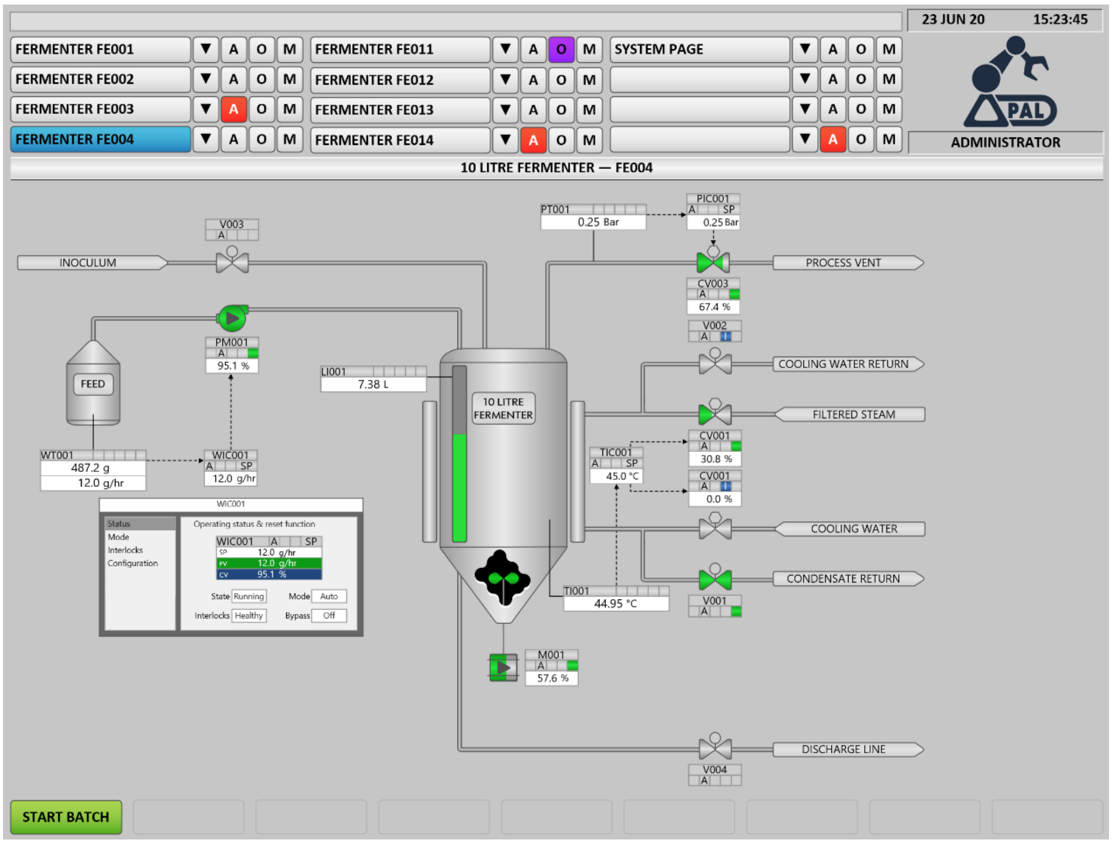


Figure 10.9 Standard graphical arrangement with "3D" effects

(4) The subtle three-dimensional effect of Figure 10.9 is considered to be a typical, *standard* graphical arrangement.

(5) Figure 10.10 shows the same arrangement with a "flattened" appearance, there are no gradient colours and fewer embellishments.

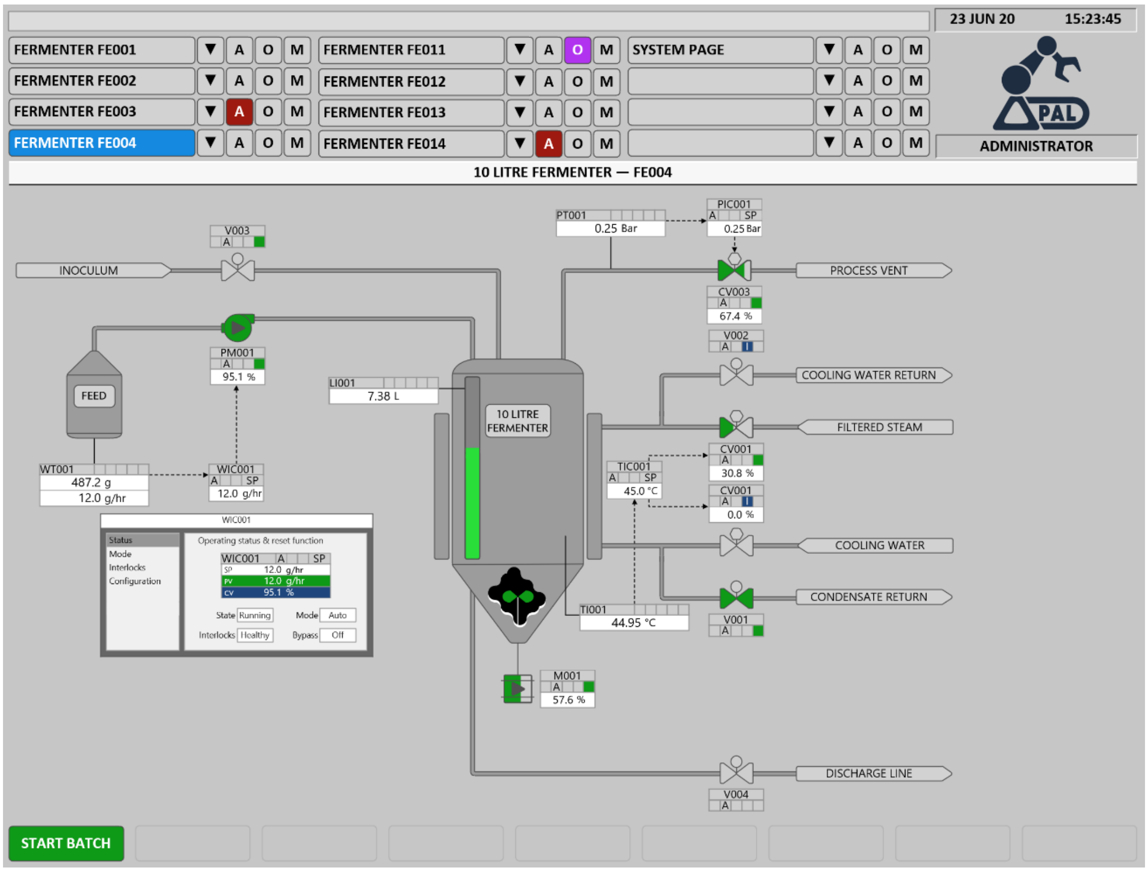


Figure 10.10 Graphical arrangement with "flat" appearance

(6) The final example is a variation of the flattened version and is more minimalistic.

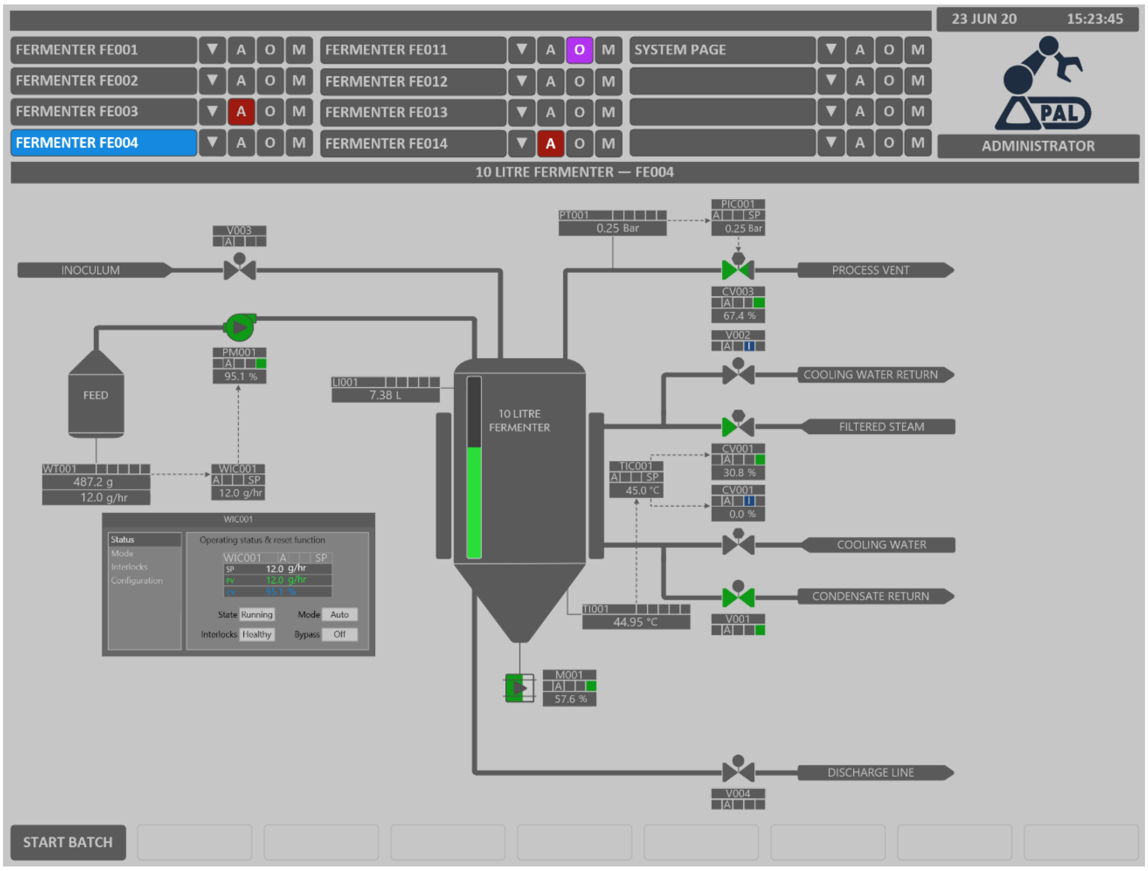


Figure 10.11 Graphical arrangement with minimal effects

(7) All of these are just examples to illustrate the nature of a graphical interface.

10.4 PAL Graphical arrangements

(1) All supervisory systems have some mechanism for navigation around the various graphical mimics and for issuing commands to the Controller, the PAL defines certain arrangements for navigation, command and other displayed functions.

(2) The basic arrangements for a graphical page that is compliant with the PAL is shown below:

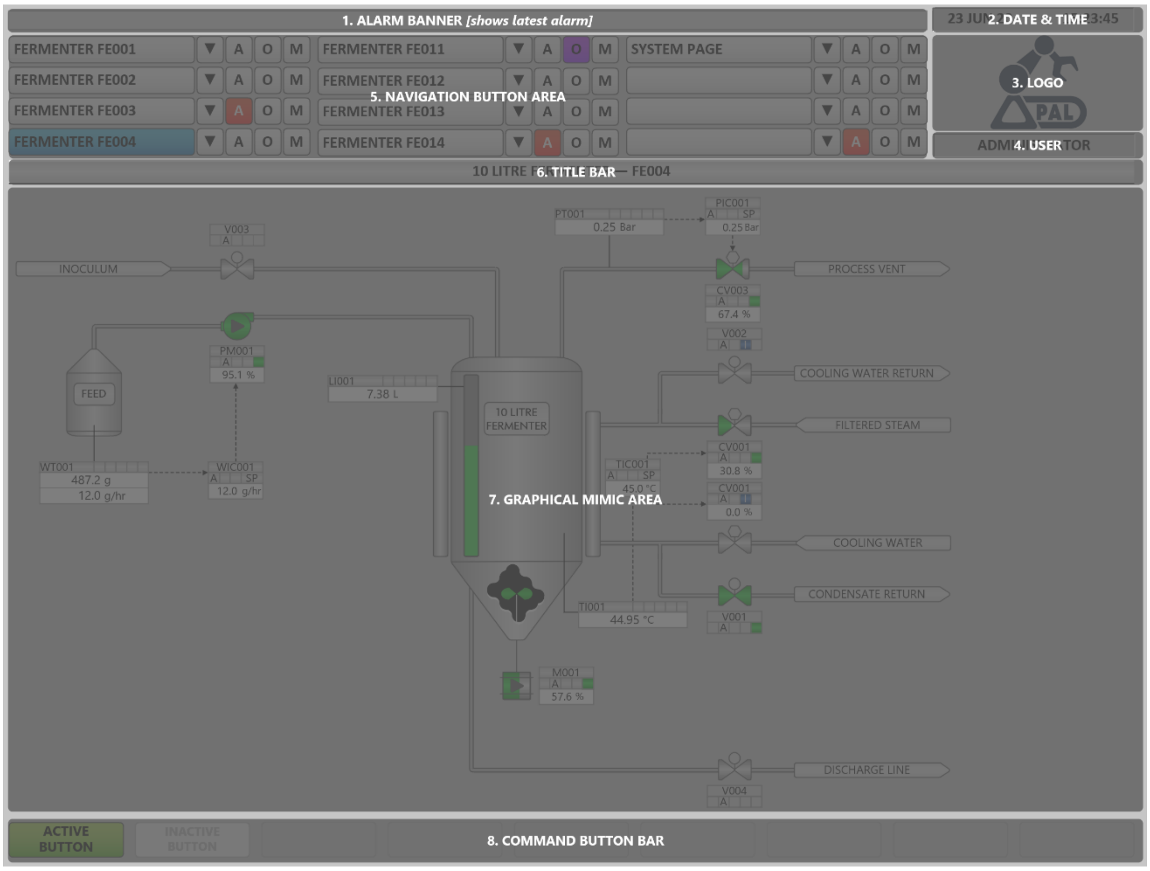


Figure 10.12 PAL graphical page arrangements

(3) The graphical page is broken down into eight discrete areas:

① **Alarm banner**
This is a single line area at the top of the screen, it displays the most recent unacknowledged alarm, if all alarms are acknowledged, it displays the most recent acknowledge, but active alarm

② **Date and time**
Displays the system time and date. The time must use the 24-hour clock with leading zeros (i.e. 13:04:20). The date must be displayed in an unambiguous format:
23 Jun 20 (month identified as letters)
2020-07-23 (ISO 8601[18] international standard)

③ **Logo**
Display area for a logo of some description.

④ **User**
Shows the username of the current user logged on to the particular supervisory system terminal.

⑤ **Navigation button area**
This shows the active plant area mimic (active mimic is in blue in the background), the ▼ button allows the user to select any sub-screens associated with the plant area.

The other buttons show any active alarms/warnings (A button), operator prompts (O button) and messages (M button).

⑥ **Title bar**
Contains the title of the graphical mimic being displayed.

---

18 ISO 8601 [Ref. 017], is the international format for recording dates. It has a standard form: YYYY-MM-DD and avoids the ambiguity of the American/English date formats: MM/DD/YY and DD/MM/YY respectively.

⑦ **Graphical mimic area**
Displays the graphical interpretation of the plant area (usually based on the P&ID diagram for the plant area). The symbols, block icons and faceplates used in this area are discussed in §§ 10.2-10.3.

⑧ **Command button bar**
This is a mimic dependent area that allows the operator to issue specific commands to the Controller (e.g. to start a particular sequence or function &c.).

Command buttons are dynamic and can be enabled or disabled by the Controller depending on the current plant conditions.

10.4.1 Screen sizes and resolutions

(1) Supervisory systems are generally PC based SCADA systems or smaller (panel mounted) HMI devices.

HMI systems

(2) Where HMIs are used, the size of the HMI is usually dependent on the complexity of the plant in question, for plants of a simple arrangement (or where the function of the HMI is extremely limited, or restricted to a very small aspect of plant control), small screen HMIs may be used.

(3) More typically, where HMIs are representing a local area of plant in the same detail as a SCADA system, larger HMIs must be used.

(4) The default PAL HMI is considered to be a HMI with a capacity equal to or better than a Siemens Simatic TP1200 Comfort Touch panel (part no.: 6AV2128-3MB06-0AX0), this is the same HMI specified as part of the test rig (see Section 3).

(5) This HMI has a screen resolution of 1280 × 800 pixels and this is considered to the minimum practical screen resolution for an HMI application of all but the simplest functions.

PC based supervisory systems

(6) PC based supervisory (or SCADA) systems are more powerful than local HMI panel and may be single stand-alone stations or multiple-user, server-client systems.

(7) The PAL minimum requirement for a supervisory system is based around the Siemens Simatic WinCC Professional application.

(8) This WinCC application does not in itself set specifications for the PC hardware upon which it is to run (apart from a basic minimum specification), the PAL, however, does set certain minimum specifications:

- Each stand-alone station or client station (if using a server arrangement) will be dual monitor station and each monitor will have a minimum resolution of 2460 × 1440 pixels.
- Stand-alone stations and servers will have 32 GB of internal RAM
- Stand-alone stations and servers will have 6 TB of hard disk storage available
- Client stations will have a minimum of 16 GB of internal RAM
- Client stations will have a minimum of 3 TB of hard disk storage available
- High power CPUs will be used (Intel i7 or better)

10.5 Alarm handling

(1) Alarm handling is generally a supervisory system function; all instrumentation and plant devices connected to the Controller (either for monitoring or control purposes) are constantly examined for fault and failure conditions. The failure of any such equipment will result in an alarm condition being generated and displayed on the supervisory system.

(2) Whatever supervisory system is used; it must be compliant with the current standards for such alarm handling: the EEMUA 191 *[Ref. 015]* Guide for Alarm Systems. Broadly, this requires that alarms and warnings have the following facilities:

- All alarms and warnings are time stamped and can be filtered
- Alarms and warnings can be suppressed by process area with a dedicated display showing all suppressed alarms
- State-based *smart* alarm/warning hiding that will hide alarms and warnings when not required (i.e. when part of the plant is not in operation)
- Alarms and warnings will be given priorities and will be colour coded to indicate the type of alarm
- alarms and warnings will be logged (archived) and can be recovered when required

(3) The supervisory system will log and display all plant alarm and warning conditions, the most recent unacknowledged, (or if all alarms and warnings are acknowledged, the most recent acknowledged alarm/warning that is still active) alarm/warning will be displayed in the alarm banner on each graphical mimic screen and will always be visible.

(4) Where required, alarms and warning signals may be marshalled into specific data blocks (or other common area), this is dependent on the specific requirement of the supervisory system in question.

(5) The system will accommodate the following types of alarms:

ALARM TYPE	DESCRIPTION
Process Alarm/ warning	Process alarms are generally derived from analogue instruments (such as flow, temperature, pressure &c.) and indicate that there is something wrong from the point of view of the manufacturing process (e.g. low pressure). Process alarms can also be generated from digital instruments monitoring for a particular alarm condition associated with the process (e.g. a low seal pressure switch).
Discrete (Digital) Alarm	Discrete alarms are generally digital alarms reporting specific occurrences of an event outside of the normal process, for example an emergency stop condition, or failure of a service supplied to the system.
Device Alarm	Device alarms are associated with a particular piece of equipment controlled by the system (a valve or drive). Device alarms are generated whenever the device is not responding correctly (or within it operational time) to the demands of the system. E.g.: Valve Failed to Open Valve Failed to Close
Instrument Alarm	Instrument alarms are associated with the state of an instrument itself (rather than the process value it is reading). Instrument alarms are generated whenever the instrument is giving out of range or fault signals.
Derived Alarm/ warning	Derived alarms are conditions that are determined by the system performing a calculation based on two or more monitored values, for example a low rate of change would be a derived alarm based on a value changing with time (value and time being the two monitored values).
System Alarm	A system alarm is associated directly with the Control System and its infrastructure (communication networks &c.). E.g. Communication Failure Component Failure – Failure of a Controller card, rack or component &c.

Table 10.41 Alarm Types

(6) The Supervisory system will organise alarms into groups associated with different plant areas (typically, organised in the same arrangement as the navigation area).

(7) The Supervisory system will also have a global alarm page that shows all the currently active alarms and warnings across the whole system. The Operator will be able to access this global alarm page directly from all screens (usually by clicking the alarm banner at the top of each screen).

(8) Alarm logging (the recording of when alarms occur, when and by whom they were acknowledged and when they were cleared from the system) for record keeping purposes, is a supervisory system activity requiring that an Operator be logged-on to the system.

(9) All alarms screens have different fields associated with the alarms and warnings (time of occurrence, time of acknowledgement, alarm description, current state &c.), these screens will be filterable by any of the associated fields.

(10) The PAL software has some independent (Controller based) alarm routines that allow the Controller to time stamp particular alarm conditions (this improve the accuracy of the alarm time stamp) under specific conditions (usually where high speed reporting is required, e.g. electrical switchgear monitoring).

(11) It is also possible for PAL software to disable specific alarms under designated conditions and even automatically acknowledge active alarms.

10.6 User management

(1) The supervisory system must support individual user logon and user groups.

(2) Different users will have different capabilities within the system. Each user will be assigned to a specific user group and each group will have specific privileges and restrictions.

(3) The PAL does not prescribe the number of user groups, nor does it specify the privileges and restrictions to be applied to each group; these are determined by the plant in question and the requirements of the plant operators. It does however, require that the supervisory system (both SCADA and HMI) support such facilities.

(4) Generally, if no user is logged on, the supervisory system will display a blank screen showing only the logon window. It is permissible for a supervisory system to allow read-only access to the graphical screens when no user is logged on (the plant can be viewed and alarms and warnings examined, but no actions can be taken, including the acknowledgement of alarms); this read only facility, while permissible is not generally recommended, its use should only be considered where necessary; e.g. plant mounted HMIs may be required to constantly display specific information.

BLANK PAGE

11 Template and documentation modules

(1) A series of template and documentation modules will be provided to give worked examples of how the *standard* and *application* modules should be used in a control system project.

(2) The template modules will provide an example of each type of application module, demonstrating how each application module is to be used and how it calls its associated standard modules.

(3) The documentation modules are specific examples of how to comment the various aspects of software written using the PAL, these give a consistent look and feel to the software. The documentation modules contain summaries of the various styles and comment formats that can be copied and used within software modules. These are essentially quick reference *(proforma)* guides that can be used as the outline for *application* modules &c.

11.1 Template modules

(1) The *template* modules explain how to use and deploy the various *standard* and *application* modules and also the various organisation blocks (OBs) that may be required in various circumstances. The template modules provide detailed example usage for all the standard modules and demonstrate different operating modes and configurations.

11.1.1 Template modules for application and standard modules

(1) There is a template module associated with each of the application modules. Each template module gives an example of how its associated application module should be used and coded. Where application modules are numbered 20,000 to 39,999, the template modules are numbered 40,000 to 59,999; thus, template module 42,000 is an example of how application module 22,000 is to be used.

(2) The following table gives the associated numbering between template modules and application modules:

FUNCTION GROUP	TEMPLATE MODULE NUBER	ASSOCIATED APPLICATION MODULE NUMBER
Debug (start of cycle)	FC 40nnn	FC 20nnn
System functions	FC 41nnn	FC 21nnn
Read instruments	FC 42nnn	FC 22nnn
Interlock & protection	FC 43nnn	FC 23nnn
Safety systems	FC 44nnn	FC 24nnn
Calculations & mathematics	FC 45nnn	FC 25nnn
Continuous control	FC 46nnn	FC 26nnn
Sequential control	FC 47nnn	FC 27nnn
Command handling	FC 48nnn	FC 28nnn
Device drivers (control loops)	FC 50nnn	FC 30nnn
Device drivers (valves)	FC 51nnn	FC 31nnn
Device drivers (drives)	FC 52nnn	FC 32nnn
Message handling	FC 56nnn	FC 36nnn
Communication handling	FC 57nnn	FC 37nnn
Debug (end of cycle)	FC 59nnn	FC 39nnn

Table 11.1 Template module and application module associations

(3) The template modules are based around a small fermenter project; this is a relatively simple project, but covers all aspects of the PAL software, to this end it provides a worked example of all the common elements of a project:

① System management and global signal generation

① Instrumentation handling (read, scale and evaluation)

② Interlock functions

③ Safety systems

④ Calculations

⑤ Continuous logic control

⑥ Sequence logic control

⑦ Command execution logic (convert continuous and sequential logic decisions to physical output signals)

⑧ Device handling (control loops)

⑨ Device handling (valves, drives &c.)

⑩ Message handling (alarms, warnings, user prompts &c.)

⑪ Communications

(4) All template modules will be fully documented and will reflect the PAL documentation standards given in the Style Guide (SG) *[Ref. 010]*.

(5) The documentation for the template fermenter project is contained within document module FC 65000.

(6) The following table give a full list of the template modules included in the PAL software:

Template modules			Associated
Coordinating	Marshalling	Programming	application module
FC40000_TmtDebugSOS			FC20000_AppDebugSOS
		FC40101_TmtDebugInst	FC20101_AppDebugInst
FC41000_TmtSysFunctions			FC21000_AppSysFunctions
FC42000_TmtInstRead			FC22000_AppInstRead
	FC42001_TmtInstAnalogRead		FC22001_AppInstAnalogRead
	FC42001_TmtInstDigitalRead		FC22001_AppInstAnalogRead
FC43000_TmtILock			FC23000_AppILock
		FC43101_TmtILockArea1	FC23101_AppILockArea1
		FC43201_TmtILockArea2	FC23201_AppILockArea2
		FC43301_TmtILockArea3	FC23301_AppILockArea3
		FC43401_TmtILockArea4	FC23401_AppILockArea4
FC44000_TmtSafe			FC24000_AppSafe
		FC44101_TmtSafeZone1	FC24101_AppSafeZone1
FC45000_TmtCalc			FC25000_AppCalc
	FC45001_TmtCalcAvg		FC25001_AppCalcAvg
		FC45700_TmtCalcNabla	FC25700_AppCalcNabla
FC46000_TmtContLogic			FC26000_AppContLogic
		FC46101_TmtContStt	FC46101_AppContStt
		FC46201_TmtContInoc	FC46201_AppContInoc
		FC46301_TmtContVent	FC46301_AppContVent
FC47000_TmtSeqLogic			FC27000_AppSeqLogic
		FC47101_TmtSeqExec	FC27101_AppSeqExec
		FC47201_TmtSeqSter	FC27201_AppSeqSter
		FC47301_TmtSeqFerm	FC27301_AppSeqFerm
		FC47401_TmtSeqCIP	FC27401_AppSeqCIP
		FC47601_TmtSeqAgit	FC27601_AppSeqAgit
FC48000_TmtCmdHandler			FC28000_AppCmdHandler
		FC48001_TmtCmdPID	FC28001_AppCmdPID
		FC48101_TmtCmdVlvIsol	FC28101_AppCmdVlvIsol
		FC48151_TmtCmdVlvMod	FC28151_AppCmdVlvMod
		FC48201_TmtCmdDriveDOL	FC28201_AppCmdDriveDOL
		FC48251_TmtCmdDriveVSD	FC28251_AppCmdDriveVSD
FC50000_TmtDevDriver			FC30000_AppDevDriver
	FC50001_TmtDevPID		FC30001_AppDevPID
	FC51001_TmtDevVlvIsol		FC31001_AppDevVlvIsol
	FC51501_TmtDevVlvMod		FC31501_AppDevVlvMod
	FC52001_TmtDevDrvDOL		FC32001_AppDevDrvDOL
	FC52501_TmtDevDrvVSD		FC32501_AppDevDrvVSD
FC56000_TmtMsgHandling			FC36000_AppMsgHandling
		FC56101_TmtMsgClassify	FC36101_AppMsgClassify
FC57000_TmtCommsHandling			FC37000_AppCommsHandling
	FC55101_TmtCommsCon2		FC35101_AppCommsCon2
FC59000_TmtDebugEOS			FC39000_AppDebugEOS
		FC59101_TmtDebugSim	FC39101_AppDebugSim
		FC59201_TmtDebugSeq	FC39201_AppDebugSeq

Table 11.2 Full list of template modules and associated application modules

11.1.2 Template modules for organisation blocks

(1) The PAL utilises organisation blocks for fault and interrupt handling. Each such organisation block has a template module that can be copied into the relevant OB to provide the necessary functions required by the PAL, these templates form the basis of each interrupt block providing the basic functions and minimum requirements needed by each.

(2) The template modules for organisation blocks are numbered in the FC 60000 to FC 60999 range, specifically they have the default OB number plus 60000, thus the OB 35 template module is given the number FC60035.

(3) The following lists all the template modules for organisation block and their associated OB number:

TEMPLATE MODULE	ASSOCIATED OB	INTERRUPT TYPE
FC60001_TmtlNrmMainProgram	OB00001_IntlNrmMainProgram	Controller main program cycle Called at the start of each Controller cycle
FC60010_TmtlNrmTimeOfDay	OB00010_IntlNrmTimeOfDay	Time of day Interrupt Called by time and day of week
FC60020_TmtlNrmTimeDelay	OB00020_IntlNrmTimeDelay	Time delay Interrupt Called after a specified delay has expired
FC60030_TmtlNrmCyclic	OB00030_IntlNrmCyclic	Timed cyclic Interrupt Called at specified intervals
FC60040_TmtlNrmHardware	OB00040_IntlNrmHardware	Hardware Interrupt Called when a specified signal is detected
FC60080_TmtlErrCycleTimeErr	OB00080_IntlErrCycleTimeErr	Error Interrupt Maximum cycle time exceeded
FC60082_TmtlErrModuleDiag	OB00082_IntlErrModuleDiag	Error Interrupt Module diagnostics signal received (module fault)
FC60083_TmtlErrModuleChange	OB00083_IntlErrModuleChange	Error Interrupt Module changed, removed or installed
FC60086_TmtlErrRackErr	OB00086_IntlErrRackErr	Error Interrupt Rack failure or fault
FC60100_TmtlErrStartUp	OB00100_IntlErrStartUp	Start-up Interrupt Called when the CPU transitions to RUN
FC60121_TmtlErrProgramErr	OB00121_IntlErrProgramErr	Error Interrupt Programming fault or error
FC60122_TmtlErrIOErr	OB00122_IntlErrIOErr	Error Interrupt IO card access fault

Table 11.3 Template modules for organisation blocks

11.2 Document modules

(1) The PAL software is extensively documented and makes use of various naming conventions for variables, constants &c.

(2) The standards and conventions for documenting the PAL software are detailed in a separate document, the Style Guide *[Ref. 010]*.

(3) The Style Guide, defines a series of rules, guidelines and practices that produce a consistent (and pleasing) programming style. It is the basis for all documentation within the PAL modules and templates.

(4) The practices specified in the style guide are summarised within the documentation modules, these are intended to be proforma examples of comments, variable and constant naming and block parameterisation.

(5) The document modules have the following allocations:

NUMBER	CLASS	FUNCTION
FC61000	Doc	Example block comments, containing the following: • Block title • Block description (typical) • Revision and modification history • Headings, list and indented text • Body text • Table, equations & figures • Special characters • Network comments
FC62001	Doc	Block allocations and block naming conventions
FC62002	Doc	Tag, variable and constant naming conventions
FC62003	Doc	UDT and data block variable naming conventions
FC62101	Doc	Structuring block comments (general)
FC62102	Doc	Building tables, equations and figures in block comments
FC62103	Doc	Special requirements for OB 1 block comments
FC62201	Doc	UDT and data block comments
FC62202	Doc	Block properties and how to use them
FC63001	Doc	Version control and revision management
FC65000	Doc	Template project documentation

Table 11.4 Document modules for the PAL

12 Regulatory requirements

12.1 Hardware regulatory requirements

12.1.1 GxP requirements

(1) This Project will comply with, and be written to, the standards necessary for *Good Manufacturing Practice* (GMP), generally referred to as GxP (see § 2.4).

(2) The GxP requirements are encapsulated in the International Society for Pharmaceutical Engineering (ISPE) guidelines, referred to as Good Automation Manufacturing Practice (GAMP), currently at revision 5 (GAMP 5), *[Ref. 011]*.

(3) The hardware requirements are determined by GAMP 5, this provides two hardware categories:

CATAGORY	DESCRIPTION	EXAMPLE	REQUIREMENTS
1 Standard hardware components	Commercially available equipment Assembled equipment using standard components	Instruments, PLCs, valves, drives, inverters &c. Electrical panels	Record: Version, model No., Serial No. &c. Verify installation Terminal schedules &c.
2 Custom built hardware components	Specialist laboratory equipment Hardware design specifically to suit the process	Custom interfaces non-standard instruments bespoke valve or drive	As category 1 plus: URS Supplier assessment Tests against URS

Table 12.1 GAMP 5 hardware classifications

(4) All hardware used within the Project will be of category 1, i.e. standard hardware components that are commercially available from multiple sources.

(5) Standard components are often referred to as *"commercial, off-the-shelf"*, indicating that these are common, commercially available items that have not been specifically designed or built for this particular application. Such items are readily available, can easily be replaced and allow for spares holding.

12.1.2 Regulatory requirements

(1) The Project hardware: electrical installation, panel, instrumentation and all associated equipment and wiring will comply with the following standards and regulations:

- Electrical Equipment (Safety) Regulations 2016
- Supply of Machinery (safety) Regulations 2008
- BS 7671 — IET Wiring Regulations 17th Edition
- BS EN60204 — Safety of machinery - Electrical equipment of machines
- EN 13850 — Safety of machinery – Emergency stop function
- EN 60947-5-5 — Electrical emergency stop devices with mechanical latching functions
- BS 6739 — Code of Practice for Instrumentation in Process Control Systems: Installation Design and Practice
- BS EN60439-1 — Specification for low voltage switchgear and control gear assemblies.
- IEC 61508 — Functional safety of electrical/electronic/programmable electronic safety related systems

12.2 Software regulatory requirements

12.2.1 Regulation and legislative requirements

(1) There are two specific sets of regulations that apply to control systems in pharmaceutical environments:

- CFR 21 Part 11 — US Code of Federal Regulations, Title 21, Food and Drugs, Part 11 – Electronic Records, Electronic Signatures [Ref. 013]
- EudraLex Vol 4 Annex 11 — EU Regulations Volume 4: Pharmaceutical legislation – Medicinal Products for Human and Veterinary use – Good Manufacturing [Ref. 014]

(2) Generally, if a system is compliant with GAMP 5 it will satisfy the EU Regulations Volume 4, Annex 11[19].

---

19 There are some additional documentation requirements and these are specifically addressed in the Project Validation Plan (VP), [Ref. 002].

(3) CFR 21 Part 11 is concerned with the accuracy, reliability and storage of electronic signatures; this is more relevant to supervisory systems rather than the Controller software of this Project; however, were applicable the PAL software will comply with these regulations.

(4) The Practical Series Automation Library software will be written to comply with the above regulations, the software will also conform to the standards specified below:

12.2.2 Software standards

(1) The Practical Series Automation Library software will be written to the standards set down in the *International Electrotechnical Commission* (IEC) publication 61131-3: Programmable controllers - Part 3: Programming languages, listed here as *[Ref. 012]*.

12.2.3 Maintenance and publication of verification certification

(1) The software library *will* be validated and *will* be fully GMP compliant (see § 2.4). The details of the validation process are given in the Validation Plan (VP), *[Ref. 002]*.

(2) The completed verification documents (e.g. test specification, calibration certificates, &c.) will be made available as secure documents that clearly identify the software module and its version number. Each document will be complete with signatures and all attachments.

12.3 Software restrictions

(1) This software must not be deployed within high-speed applications. The software is designed to run on systems with a response time of 100 ms or greater.

BLANK PAGE

13 PAL user documentation

(1) TIA portal supports various mechanisms for storing the user documentation of software modules; the PAL makes extensive use of this facility.

(2) All software modules within the PAL are extensively documented within the modules themselves, see the Style Guide *[Ref. 010]* for details, this includes block headers and individual network comments.

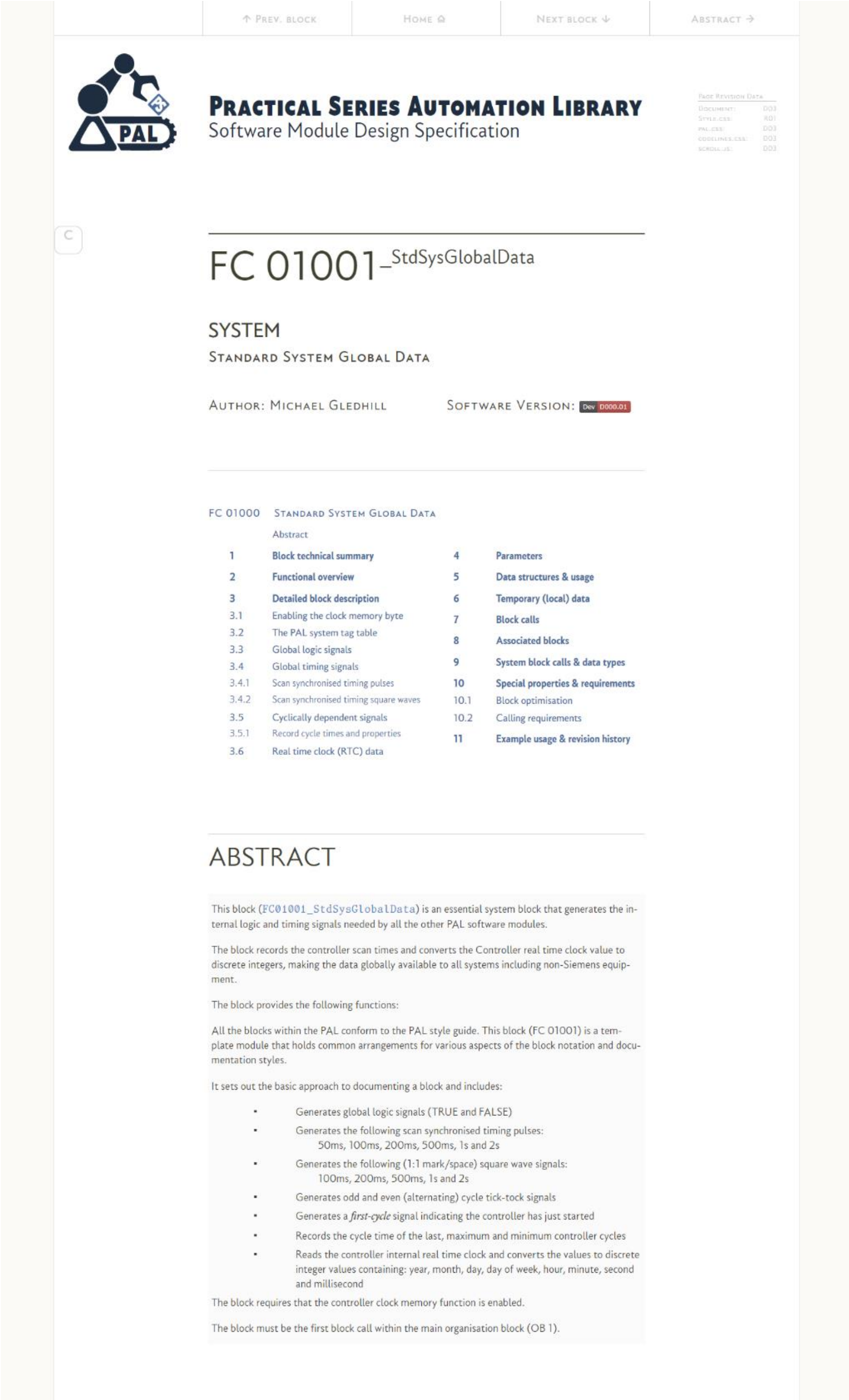
(3) In addition, the TIA facility for user documentation (referred to as *TIA User Documentation)* is also used. This facility allows documents to be stored in a variety of formats: PDF documents, text documents, Microsoft Word documents and also as web pages.

(4) Of all these formats, the PDF format offers the most flexibility, it is readily produced from the Software Module Design Specifications *[Ref.* 008*]* (written in Word DOCX format), can be configured to use the document headings as navigable bookmarks and can be rendered in most standard web browser.

(5) The PAL user documentation will also provide links to the various documents generated within this project. This includes the following:

- The User Guide *[Ref.* 009*]*
- The software Design Specification *[Ref. 006]*
- Individual Software Module Design Specifications *[Ref.* 008*]*
- The Style Guide *[Ref. 010]*

(6) The PAL user documentation will also be developed as a full website. This website provides a standard format for displaying the PAL user documentation, it has the following appearance:



↑ Prev. block | Home | Next block ↓ | Abstract →

Practical Series Automation Library
Software Module Design Specification

Page Revision Data
Document: D03
Style.css: R01
pal.css: D03
codelines.css: D03
scroll.js: D03

FC 01001_StdSysGlobalData

SYSTEM

Standard System Global Data

Author: Michael Gledhill   Software Version: Dev D000.01

ABSTRACT

This block (`FC01001_StdSysGlobalData`) is an essential system block that generates the internal logic and timing signals needed by all the other PAL software modules.

The block records the controller scan times and converts the Controller real time clock value to discrete integers, making the data globally available to all systems including non-Siemens equipment.

The block provides the following functions:

All the blocks within the PAL conform to the PAL style guide. This block (FC 01001) is a template module that holds common arrangements for various aspects of the block notation and documentation styles.

It sets out the basic approach to documenting a block and includes:

- Generates global logic signals (TRUE and FALSE)
- Generates the following scan synchronised timing pulses:
  50ms, 100ms, 200ms, 500ms, 1s and 2s
- Generates the following (1:1 mark/space) square wave signals:
  100ms, 200ms, 500ms, 1s and 2s
- Generates odd and even (alternating) cycle tick-tock signals
- Generates a *first-cycle* signal indicating the controller has just started
- Records the cycle time of the last, maximum and minimum controller cycles
- Reads the controller internal real time clock and converts the values to discrete integer values containing: year, month, day, day of week, hour, minute, second and millisecond

The block requires that the controller clock memory function is enabled.

The block must be the first block call within the main organisation block (OB 1).

Figure 13.1 PAL Typical PAL user documentation web page

(7) The PAL user documentation website will support the following functions in addition to the standard displaying of text:

- Utilise embedded fonts
- Be responsive to screen resolution (support for phone and tablet devices)
- Utilise JavaScript and jQuery
- Utilise persistent *"sticky"* navigation to ensure ease of use
- Provide facilities for:
    - Allowing images to be overlayed on the screen "lightbox" imaging
    - Display code fragments
    - Display mathematical formulae

(8) The PAL user documentation website will be distributed within the library software (distributed as part of the software project itself).

(9) The PAL user documentation website will be available in its own right from with the PSP internal intranet.

13.1 Training

(1) The User Guide *[Ref.* 009*]* forms the principle training document for the PAL software, formal training based around the User Guide will be provided for all PSP personnel involved with the deployment and use of the software.

(2) The PAL software requires the implementation of software within the Simatic S7-1500 and S7-1200 ranges of Controller; as such, it should only be used by those whom have a detailed knowledge of Simatic Controller and the TIA Portal programming environment.

BLANK PAGE

14 References and glossary

14.1 Document references

REF	DOCUMENT NO.	AUTHOR	TITLE/DESCRIPTION
001	PS2001-5-0101-001	PSP	Quality Plan (QP)
002	PS2001-5-0121-002	PSP	Validation Plan (VP)
003	PS2001-5-1101-001	PSP	User Requirements Specification (URS)
004	PS2001-5-1111-001	PSP	Requirement Traceability Matrix (RTM)
005	PS2001-5-2101-001	PSP	Functional Specification (FS) (THIS DOCUMENT)
006	PS2001-5-2211-001	PSP	Hardware Design Specification (HDS)
007	PS2001-5-2311-001	PSP	Software Design Specification (SDS)
008	PS2001-5-2312-fcNo	PSP	Software Module Design Specifications (SMDSs)
009	PS2001-5-7111-001	PSP	User Guide (UG)
010	PS2001-5-2313-011	PSP	Style Guide (SG)
011	GAMP 5	ISPE	Good Automated Manufacturing Practice
012	IEC6113-3	IEC	Programmable controllers - Part 3: Programming languages
013	CFR 21, Part 11	US CFR	US Code of Federal Regulations, Title 21, Food and Drugs, Part 11 – Electronic Records, Electronic Signatures
014	EudraLex Vol 4 Annex 11	EU Regulations	Vol 4: Pharmaceutical legislation – Medicinal Products for Human and Veterinary use – Good Manufacturing
015	EEMUA 191	EEMUA	Alarm systems - a guide to design, management and procurement
016	EEMUA 201	EEMUA	Control rooms: a guide to their specification, design, commissioning and operation
017	ISO 8601	ISO	Date and time format
018	PS2001-5-2301-001	PSP	Register of software modules and revisions
019	PS2001-5-2302-011	PSP	Software Control Mechanism (SCM)

Table 14.1 Table of references

14.2 Glossary of terms

ABBREVIATION	DESCRIPTIONS
AC	Alternating Current
AI	Analogue Input
AQ	Analogue Output
ASCII	American Standard Code for Information Interchange
BS	British Standard
BS EN	British standards (BS) adoption of a European Standard (EN)
CFR	Code of Federal Regulations
CPU	Central Processing Unit
CSS	Cascading Style Sheet
DC	Direct Current
DB	Data Block
DI	Digital Input
DOL	Direct Online
DQ	Digital Output
EEMUA	Engineering Equipment and Materials Users' Association
EoC	End of Cycle
EN	European Standards
EudraLex	European Union Drug Regulation Authority Legislation
EU	European Union
FAT	Factory Acceptance Test
FB	Function Block
FC	Function
FMS	Fieldbus Message Specification
FS	Functional Specification
GAMP	Good Automated Manufacturing Practice
GMP	Good Manufacturing Practice
GRAFCET	GRAPHe de Commande Etape-Transition (sequence documentation)
GxP	Collective abbreviation for GMP and GXP
HDS	Hardware Design Specification
HMI	Human Machine Interface
HTML	Hypertext Mark-up Language
iDB	Instance Data Block

ABBREVIATION	DESCRIPTIONS
IEC	International Electro-technical Commission
IEC 61131-3	IEC standard for the syntax and semantics for PLC programming
IET	Institution of Engineering and Technology
IM	Interface Module
IO	Input/Output
IP	Internet Protocol
IQ	Installation Qualification
ISPE	International Society for Pharmaceutical Engineering
ISO	International Standards Organisation
JavaScript	A web-based scripting language
jQuery	A library of JavaScript objects, commonly used in web development
Ladder	Ladder Logic (PLC programming language)
MDF	Medium-density Fibreboard
MIT	Massachusetts Institute of Technology (Licence)
NC	Normally Closed (type of valve)
NO	Normally Open (type of valve)
OB	Organisation Block
OQ	Operational qualification
OSL	Operating State Logic
PAL	Practical Series Automation Library
P&ID	Piping and Instrumentation Diagram
PDF	Portable Document Format
PDT	PLC Data Type
PI	Process Image
PID	Proportional, Integral, Derivative — a common type of control loop
PII	Process Image of Inputs
PIP	Process Image Partition
PIPI	Process Image Partition of Inputs
PIPQ	Process Image Partition of Outputs
PIQ	Process Image of Outputs
PLC	Programmable Logic Controller (another name for a Siemens
ProfiBus	Process Field Buss
Profinet	Process Field Net
PSP	Practical Series of Publications
QP	Quality Plan

ABBREVIATION	DESCRIPTIONS
RAL	Colour standards (Reichs-Ausschuß für Lieferbedingungen und Gütesicherung)
RAM	Random Access Memory
RoC	Rate of Change
RTD	Resistance Temperature Device
RTM	Requirements Traceability Matrix
SCADA	Supervisory Control and Data Acquisition
SCM	Software Control Mechanism
SDS	Software Design Specification
SG	Style Guide
SMDS	Software Module Design Specification
SoC	Start of Cycle
STL	Statement List (PLC programming language)
TIA	Totally Integrated Solutions (TIA Portal, a Siemens programming tool)
TC	Thermocouple (when referring to IO cards)
TCP/IP	Transmission Control Protocol/Internet Protocol
UDT	User Data Type
UG	User Guide
UI or U/I	Voltage and current (when referring to IO cards)
URS	User Requirements Specification
US	United States of America
UT	User Data Type (alternative abbreviation)
VAC	Voltage (alternating current)
VDC	Voltage (direct current)
VP	Validation Plan
VSD	Variable Speed Drive

Table 14.2 Glossary