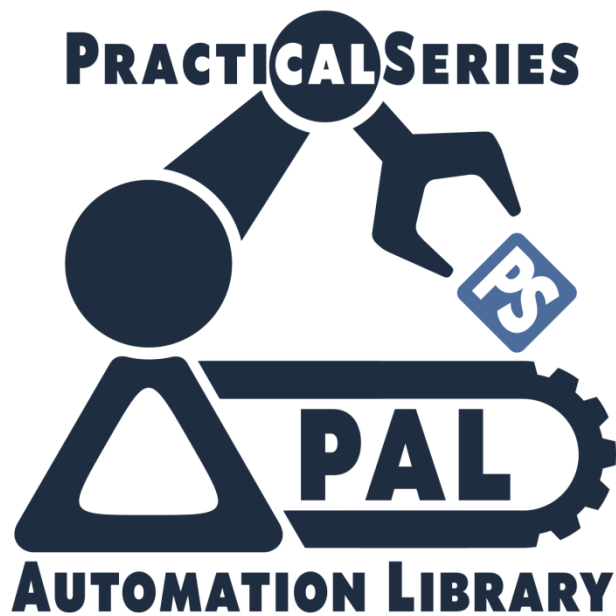


Practical Series

PRACTICAL SERIES AUTOMATION LIBRARY SOFTWARE DESIGN SPECIFICATION

AUTHOR: MICHAEL GLEDHILL



Published By:



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



Copyright 2021

Michael Gledhill

Document No.: PS2001-5-2311-001

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

LICENCE This document and associated software are made available under the MIT License:

The MIT License (MIT)
Copyright © 2021 Michael Gledhill

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Based on template: PS2001-5-nnnnn-nnn R02.00 SHORT GxP Blank (Ind-Calisto) - Indexable PDF format

WebIndex:ber

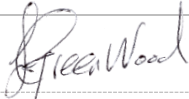
DOCUMENT AUTHORISATION

	NAME	POSITION	SIGNATURE	DATE
Author	Michael Gledhill	Lead Engineer		21 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		21 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	Frank Greenwood	Project Manager		21 May 2022
-------------	-----------------	-----------------	---	-------------

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

REVISION

REVISION	DATE	REVISED BY	DESCRIPTION
R02.00	21 May 2022	Michael Gledhill	Properties standardised across all documents Changes to interrupt and functional group names
R01.00	29 Jul 2020	Michael Gledhill	First release for use

CONTENTS

1.	Introduction	11
1.1	Scope of this document	12
1.2	Ownership, status & relationship to other documents	14
1.2.1	Ownership of the document	14
1.2.2	The status of this document	14
1.2.3	Relationship to other documents	14
1.3	Understanding and using this document	16
2.	Overview	17
2.1	A description of the Project software	17
2.1.1	Standard modules, an overview	19
2.1.2	Application modules, an overview	21
2.1.3	Template modules	26
2.1.4	Documentation modules	27
2.2	Regulatory requirements	27
2.2.1	Software classification	28
2.2.2	Regulation and legislative requirements	29
2.2.3	Software standards	29
2.2.4	Maintenance and publication of verification certificates	29
2.3	A description of the User Documentation	30
2.4	Assumptions and limitations	33
2.5	Nonconformity	33
2.6	Addressing the URS requirements	33
3.	Programming environments and common settings	35
3.1	Engineering stations and Windows settings	36
3.1.1	Engineering station operating system and hardware specifications	36
3.1.2	ES fixed IP address	38
3.1.3	Naming the Engineering Station	39
3.1.4	Windows regional settings	42
3.2	TIA Portal settings	43
3.2.1	Applying PAL settings to TIA Portal	43
3.2.2	TIA Portal block overview column settings	45
3.3	Common CPU Properties	46

4.	Naming, numbering and other conventions.....	49
4.1	Block type and numbering conventions.....	50
4.1.1	Block numbering.....	53
4.1.2	Standard, application and template block numbering.....	55
4.1.3	Data block numbering.....	56
4.1.4	Instance data block numbering.....	58
4.1.5	OB (Interrupt block) numbering.....	59
4.1.6	Document block numbering.....	60
4.1.7	Block numbering summary.....	61
4.2	Module naming Conventions.....	64
4.2.1	Block type.....	64
4.2.2	Block number.....	65
4.2.3	Block class.....	65
4.2.4	Block function.....	66
4.2.5	Block description.....	67
4.2.6	Block naming restrictions.....	67
4.3	Block optimisation & IEC check.....	68
4.4	Tags, parameters, symbolic and absolute representations..	69
4.4.1	EN and ENO parameters.....	71
4.5	Block parameter naming.....	73
4.5.1	Formal parameters.....	74
4.5.2	Temporary (local) data.....	75
4.5.3	Constants.....	76
4.5.4	Static data (function blocks only).....	78
4.6	Naming variables in static UDTs.....	79
4.7	Naming variables in dynamic UDTs.....	80
4.7.1	UDTs holding recipe data.....	81

4.8	Naming variables in static DBs.....	82
4.9	Naming variables in dynamic DBs.....	83
4.9.1	DBs holding recipe data.....	83
4.10	Tags and tag naming.....	84
4.10.1	The PAL system tags (PAL_SystemTags)	84
4.10.2	The PAL Input/Output tags (PAL_IOTags).....	86
4.10.3	Project specific tag tables.....	88
4.11	Control system network device naming.....	89
5.	Common appearance and version control.....	91
5.1	TIA Portal comment fields	91
5.1.1	Maximum size of a comment field	94
5.2	Common headers and networks	95
5.2.1	Block title and comment field	96
5.2.2	Network 1 — Block description.....	98
5.2.3	Network 2 — Current revision and modification history	103
5.3	OB I header and revision network	105
5.3.1	OB I Network 1 — Project description	105
5.3.2	OB I Network 2 — Current revision and modification history	108
5.4	General network comments.....	110
5.5	Specific network comments for sequences	111
5.5.1	Step declaration network — title and comments.....	112
5.6	Data block header and revision	114
5.6.1	Data block revision information.....	116
5.6.2	UDT block revision information	116
5.7	Programmable block properties	117
5.8	Data block and UDT properties	119
5.8.1	Data block properties (static and dynamic)	119
5.8.2	UDT properties (static and dynamic)	121
5.9	Hardware component comments	122

6.	Standard modules.....	123
6.1	SMDS contents.....	123
6.2	Standard block list and associated documentation.....	135
6.2.1	System function modules.....	135
6.2.2	Instrument read modules.....	135
6.2.3	Interlock and protection modules.....	136
6.2.4	Safety and safety system modules.....	137
6.2.5	Calculations and mathematics modules.....	137
6.2.6	Sequential control.....	140
6.2.7	Device drivers — Control loops.....	140
6.2.8	Device drivers — Valves.....	141
6.2.9	Device drivers — Drives.....	141
6.2.10	Message handling.....	142
6.2.11	Communication handling.....	143
6.2.12	Subroutines.....	144
6.2.13	Debug subroutines.....	145
7.	Application modules.....	147
7.1	Application module numbering.....	151
7.2	Sequence annotation.....	153
7.2.1	Sequence IO matrix summary.....	162
8.	Interrupt modules.....	163
8.1	Error detection OBs.....	165
9.	Template modules.....	167
9.1	Templates for application modules.....	167
9.2	Template modules for organisation blocks.....	170
10.	Documentation modules.....	171
11.	Common approach to data handling.....	173
11.1	Conventions for using UDTs.....	173
11.1.1	Static UDT conventions.....	174
11.1.2	Dynamic UDT conventions.....	176

12.	Common modes of operation	179
12.1	Manual mode.....	179
12.2	Bypass mode.....	181
12.3	Simulation mode.....	182
12.4	Remote/local mode.....	183
12.5	Faceplate disable mode.....	185
13.	User documentation	187
13.1	Organising the user documentation	188
13.1.1	The use of a home page	192
13.2	Project specific User Documentation	193
13.2.1	User Documentation for additional items	194
14.	Software security.....	195
14.1	The protecting of software modules.....	195
15.	References and glossary	197
15.1	Document references.....	197
15.2	Glossary of terms.....	198

BLANK PAGE

1

Introduction

This document is the *Software Design Specification* (SDS) for the *Practical Series Automation Library* of software modules (the PAL).

This Software Design Specification has been produced by Michael Gledhill, under his authority as the lead engineer of the Practical Series Automation Library of software modules project (hereafter referred to as the Project).

The Project software consists of a library of software modules and templates that have been made available for the Siemens Simatic S7-1500 range of controllers (and to a lesser extent the S7-1200 range), what used to be referred to as *Programmable Logic Controllers* or PLCs.

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

This document, the Software Design Specification, explains the underlying concepts of how the software works, the fundamental structure applied to the software and the common features and practices needed to implement the software.

The SDS is a very detailed document and assumes a degree of knowledge about PLC programming in general and TIA Portal programming and Simatic Controllers in particular.

1.1 Scope of this document

This document is the main design document for the *Practical Series Automation Library* of software modules. It is a detailed document that explains the underlying philosophy behind the PAL. It does the following:

- ① Explains the PAL Controller programme structure
- ② Establishes the common properties for S7 Controllers and associated hardware
- ③ Establishes the naming and addressing conventions for Controller hardware and programming devices:
 - PAL network addressing
 - Controller naming
 - Remote rack naming
 - HMI, SCADA and server naming
 - Engineering station (ES) naming
- ④ Establishes the block numbering mechanisms for:
 - Programmable blocks (FC, FB and OB)
 - Data blocks and instance data blocks (DB and iDB)
 - User data types (UDT)

- ⑤ Establish naming conventions for:
 - Programmable block names (FC, FB and OB)
 - Data blocks and instance data blocks (DB and iDB)
 - User data types (UDT)
 - Block parameter names
 - Local variables and constants
 - Tag (symbolic) names
 - Data block variables
- ⑥ Explains the usage of data types within the PAL
- ⑦ Establish the standard PAL global signals
- ⑧ Specifies the structure of interrupt organisation blocks
- ⑨ Explains common library features (automatic and manual operations &c.)
- ⑩ Establish PAL conventions for block optimisation
- ⑪ Explains how the operator interfaces (HMI and SCADA) to the system work

In addition to explaining the PAL software, this Software Design Specification also explains the user documentation mechanisms used to provide web-based information about the PAL software; this documentation is embedded within the PAL project files and is accessible using the user documentation facilities available within the TIA Portal application.

1.2 Ownership, status & relationship to other documents

This document (the Software Design Specification) is a design 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 Software Design Specification has been produced, and is controlled and maintained by the Practical Series of Publications (PSP).

This Software Design 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 Software Design Specification (*this document*) is a subordinate document to the Functional Specification (FS) [Ref. 005] and is a deliverable item under the terms of the Project. The Software Design Specification is an internally approved document (approved by the Practical Series of Publications).

1.2.3 Relationship to other documents

The Software Design Specification is a subordinate design document for the Project, it both expands and provides additional detail to the specifications given in the Functional Specification (FS) [Ref. 006]. The SDS also acts as a coordinating document the individual Software Module Design Specification documents (SMDSs) [Ref 008].

The full document flow-path for the Project including the Software Design 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].

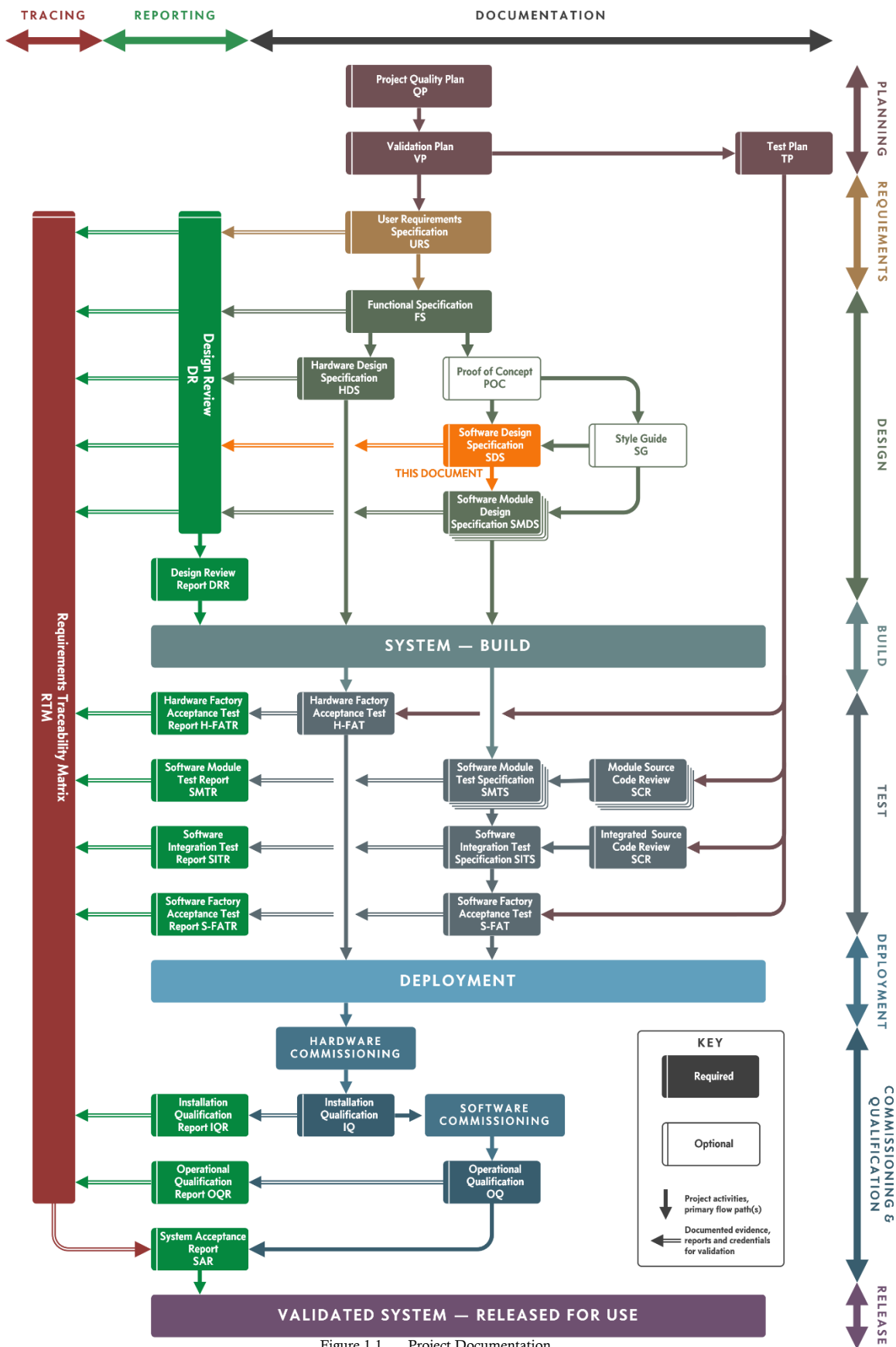


Figure 1.1 Project Documentation

1.3 Understanding and using this document

This document, the SDS, is a detailed design document, it builds upon and expands the information provided in the Functional Specification (FS), [Ref. 005]. The document is concerned specifically with the design and implementation of software for the Simatic S7-1500 and S7-1200 ranges of Controller; as such, this document is intended for those whom have a detailed knowledge of Simatic Controller and the TIA Portal programming environment.

This document uses technical terminology common to both the programming of PLCs in general and Simatic Controllers in particular; where such terminology is used within the accepted engineering conventions and customs of this field, it is done so without further explanation. For example, reference to the “*Clock Memory*” within a Controller, is done without further explanation, the clock memory being a common and well-known component of a Simatic Controller; those reading this document are expected to know what the Controller clock memory is and how to configure and use it.

2

Overview

This overview sets out a brief description of the Project software, its purpose and its design.

2.1 A description of the Project software

The Practical Series Automation Library (PAL) Project is a library of software modules and templates that can be used to control and develop software for use in Siemens Simatic Controllers to control and operate most industrial process applications.

The PLC software is designed to be applicable to virtually all industrial applications that can generally controlled by a programmable logic controller (PLC).

PLCs are general purpose devices designed to control a wide range of process plant and while PLC are relatively fast acting devices, they, and consequently the PAL software, would not be considered suitable for very high-speed applications (bottling machinery, paper mills, rapid assembly equipment &c.), such system usually have their own bespoke controller dedicated to the requirements of the particular application.

Generally, PLCs are suitable for 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; but, a response time of 100 ms is considered to be an acceptable (and widely used) limit for PLC control.

The PLC software was developed for, and has been deployed in, the following industrial applications:

- 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 fundamental level, the PAL is a library of software modules that control 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 software structure contains *application* specific modules; these contain software that is applicable to the plant being controlled, these can be simple coordinating modules that organise the software into logical areas, marshalling modules that call the standard modules or programmable modules that contain the control software required to control specific aspects of the process plant.

In addition, the PAL is supplied with *template* modules, these serve as example modules to demonstrate how the PAL modules should be used, and the best practices for doing so.

Finally, there is a series of *documentation* modules that demonstrate how the modules should be documented, commented and named.

2.1.1 Standard modules, an overview

Standard modules carry out a particular function; an example would be a module that controls the operation of a valve, such a module would 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 commanded 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

The PAL has many of this type of module; in fact these modules make up the bulk of the PAL.

The standard modules within the PAL are fixed modules, the software within these modules has been written, tested and validated and must not be modified or changed in any way (to do so would invalidate the software).

The standard modules cover all aspects of a control system:

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

There are also various *standard* subroutines that, while not associated with any particular piece of equipment, provide common software functions (e.g. timing functions, string functions, format conversions &c.) that are available to all modules within the Controller.

The different options available to a standard module are selected by passing parameters to the module that either activate or deactivate modes of operation or pass configuration values to the module (opening and closing times of a valve, operating range of an instrument for example).

The mechanisms, data structures, numbering and naming conventions for standard modules and the methods of passing data to the module is done in conformance with standard practices set out within the PAL (and within this document).

2.1.2 Application modules, an overview

Application modules are project specific modules; they are written for a particular project and are configured to match the requirements of that project.

The project (that is the Controller project that operates some aspect of a plant) is usually required to control and monitor various devices and instruments. For example, consider a simple filtration plant (Figure 2.1), this expands on the example given in the User Requirements Specification (§ 2.1.2) [*Ref. 003*].

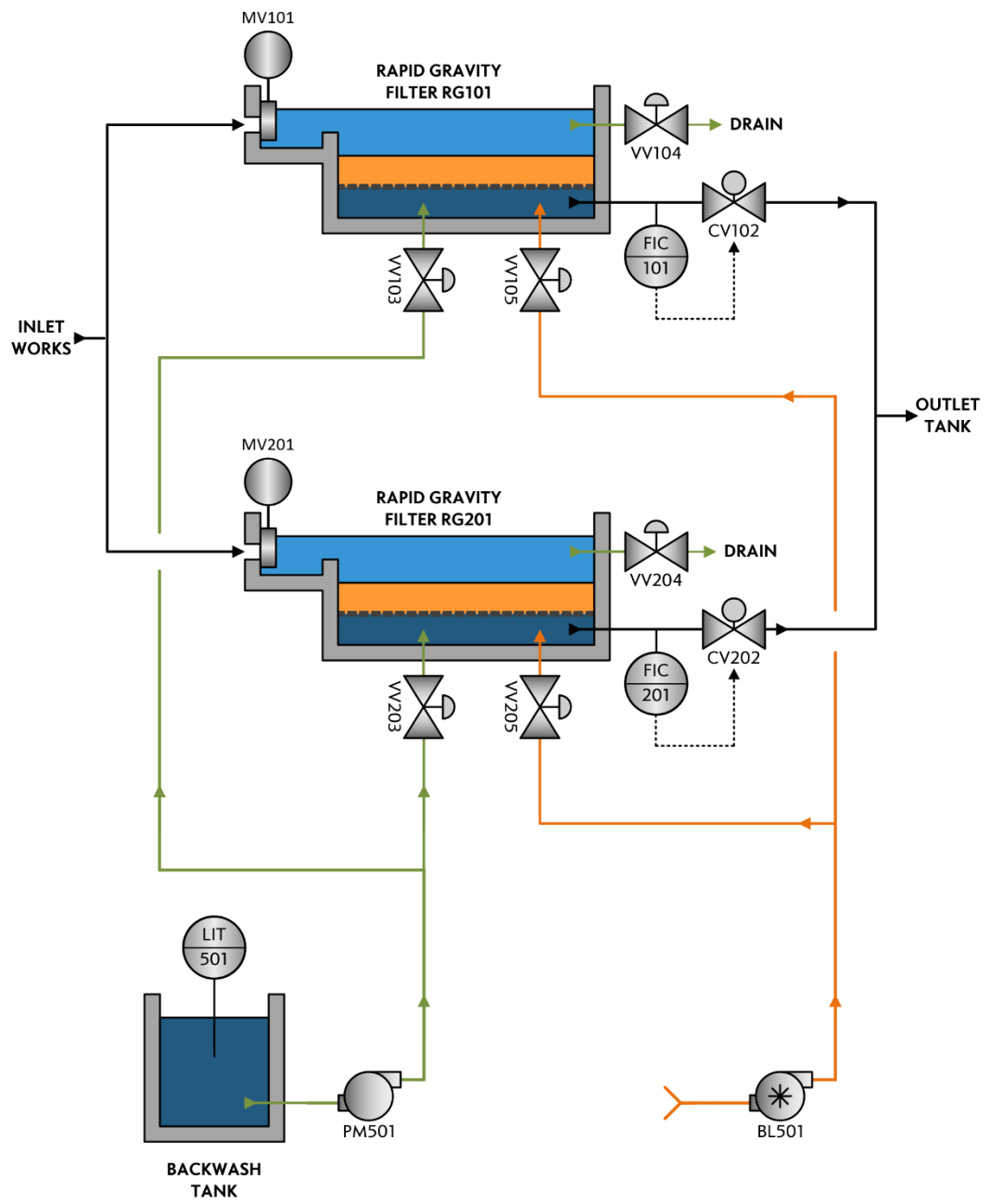


Figure 2.1 Filtration plant schematic

This has the following valves:

BISTABLE MOTORISED VALVES		QUANTITY 2
DEVICE TAG	DESCRIPTION	
MV101	Filter 1 inlet valve	
MV201	Filter 2 inlet valve	

Table 2.1 Filtration plant equipment — Motorised valves

MODULATING VALVES		QUANTITY 2
DEVICE TAG	DESCRIPTION	
CV102	Filter 1 outlet valve	
CV202	Filter 2 outlet valve	

Table 2.2 Filtration plant equipment — Modulating valves

ISOLATING VALVES		QUANTITY 6
DEVICE TAG	DESCRIPTION	
VV103	Filter 1 backwash water inlet valve	
VV104	Filter 1 backwash water outlet valve	
VV105	Filter 1 Air inlet valve	
VV203	Filter 2 backwash water inlet valve	
VV204	Filter 2 backwash water outlet valve	
VV205	Filter 2 Air inlet valve	

Table 2.3 Filtration plant equipment — Isolating valves

It has the following drives:

DOL DRIVES		QUANTITY 2
DEVICE TAG	DESCRIPTION	
BL501	Backwash blower I	
PM501	Backwash pump I	

Table 2.4 Filtration plant equipment — Drives

And the following instruments:

INSTRUMENTS		QUANTITY 3
DEVICE TAG	DESCRIPTION	
FIC101	Filter 1 outlet flow	
FIC102	Filter 2 outlet flow	
LIT501	Backwash tank level	

Table 2.5 Filtration plant equipment — Instruments

In summary, this filtration plant has the following types of devices and instruments:

- 2 × Bistable valves (open/close valve driven by a motor)
- 2 × Modulating valves (can be set to any position between opened and closed)
- 6 × Isolating valves (open/close valve operated by a solenoid)
- 2 × DOL Drives (simple start/stop motors)
- 3 × Instruments

I.e. it has five different types of devices and instruments.

To programme this project using the PAL five standard modules would be needed (one for each type of device):

- Standard module for bistable valves
- Standard module for modulating valves
- Standard module for isolating valves
- Standard module for DOL drives
- Standard module for instruments

There would also be five project specific applications modules:

- Application module for bistable valves
- Application module for modulating valves
- Application module for isolating valves
- Application module for drives
- Application module for instruments

The first of these (for bistable valves) would call the standard bistable valve module two times (once for MV101 and once for MV201) and each instance would link the standard module to the particular IO and internal storage locations associated with the motorised valve in question.

Similarly, the application module for modulating valves would call the standard modulation valve module twice (once for CV102 and once for CV202).

The isolating valve application module would call the standard module (for isolating valves) six times (for VV103, VV104, VV105, VV203, VV204 and VV205).

And so on.

The contents of each application module is dependent on the requirements of the plant being controlled (specifically how many of each type of device exist). I.e. the application modules differ between different projects; the standard modules on the other hand are the same across all projects.

In the above (filtration) example, the application modules are simply co-ordination areas that call the required standard modules the requisite number of times; there is clearly more to a Controller programme than this, something must decide when a valve is to be opened, a drive started &c. and in the case of the modulating valves something must decide what position the valve should adopt.

The type of logic that performs these actions is either continuous logic (operates all the time) or sequential logic (operates as part of a sequence). In this case, the filter would normally operate under continuous control (the inlet valve would open and the modulating outlet valve position adjusted to maintain a specific flow from the filter).

At some point, the filter will need to be cleaned (probably at a specific time of day); the continuous logic would (when the specific time was reached) trigger a cleaning sequence that would then take control of the filter and clean it. A typical sequence would be:

- Isolate the filter (take it out of service and close all valves)
- Aerate the filter (open air inlet valve and start blower)
- Backwash the filter with aeration (open backwash inlet and outlet valves and start backwash pump)
- Washout the filter (stop blower and close air inlet valve)
- Allow filter bed to settle (stop pump and close backwash valves)
- Return filter to service (open inlet and outlet valves)

These modules (continuous logic modules and sequential control modules) are also application modules.

2.1.3 Template modules

Template modules are example modules that explain how to do things within the PAL, a typical template module being one that shows how sequences work within the PAL.

Template modules contain a basic configuration that can be copied, expanded and modified for the application in question; they provide a basic “*skeleton*” software structure that can be used repeatedly for a particular type of application.

Templates exist for most application modules and should be used wherever possible as a model for the application module.

Templates are used to develop the software for a particular plant. Once the software is complete (or at least past the development stage) the template modules themselves should be deleted.

2.1.4 Documentation modules

The PAL software is extensively commented (indeed, there is a Style Guide (SG) [Ref. 010] dedicated to explaining how to comment PAL software); the documentation modules contain examples of different types of comments for the various different software modules and data structures used within the PAL.

Like the template modules, the documentation modules are used to make the development of the software easier, and should be deleted once the software development is complete.

2.2 Regulatory requirements

- (1) 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 GxP¹.
- (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 system software and hardware development and use.
- (3) 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*.
- (4) 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.

¹ GxP is a general term for good ... practice, where the x stands for various things, manufacturing, distribution, laboratory, clinical, engineering, &c.

2.2.1 Software classification

- (1) 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) GAMP 5 provides the following software categories (category 2 is no longer used):

CATAGORY	DESCRIPTION	EXAMPLE	REQUIREMENTS
1 Infrastructure Software	Layered software (i.e., upon which applications are built) Software used to manage the operating environment	Operating System Database Engines Programming languages Statistical packages Spreadsheets	Record version Verify installation
3 Non- Configured Software	Run-time parameters may be entered and stored, but the software cannot be configured to suit the process	Firmware- Commercial off the shelf software	As category 1 plus: URS Supplier assessment Tests against URS
4 Configured Software	Software, often very complex, that can be configured by the user to meet the specific needs of the process. Application software code is not altered.	Data acquisition systems: • SCADA • HMI • ERP • MRPII	As category 3 plus: Verify supplier QMS Design specs. (DS) Tests against DS Procedures for: • Data management • Maintenance
5 Custom (bespoke) Software	Software custom designed and coded to suit the process.	Bespoke IT applications Bespoke control systems Custom ladder logic Custom firmware Spreadsheets (macro)	As category 3 plus: Full life cycle docs: FS, DS, SDS, HDS, SMDS &c. Source code review Structural testing: SMT, SIT, FAT, IQ, OQ

Table 2.6 GAMP 5 software classifications

- (3) The control system and software being developed as part of this Project is a bespoke system and, under the GAMP 5 classification system, is a **category 5** system.

2.2.2 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.
- (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:

2.2.3 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].
- (2) All software will be written using Ladder Logic (other languages including statement list will not be used).

2.2.4 Maintenance and publication of verification certificates

- (1) The software library *will* be validated and *will* be fully GMP compliant (see § 2.2.1). 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.

2.3 A description of the User Documentation

TIA portal supports various mechanisms for storing user documentation for software modules; the PAL makes extensive use of this facility.

All software modules within the PAL are extensively documented within the modules themselves, see the Style Guide [Ref. 010] for details, this includes the block headers and individual network comments discussed in section 5.2.

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.


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.

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. 007]
- Individual Software Module Design Specifications [Ref. 008]
- The Style Guide [Ref. 010]

The PAL user documentation will also be developed as a full website, working under the confines and structures imposed by the TIA User Documentation requirements. 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

Block Revision Data

Document:	003
System ID:	001
File ID:	003
Module Name:	003
Revision:	003

C

FC 01001_StdSysGlobalData

SYSTEM

STANDARD SYSTEM GLOBAL DATA

AUTHOR: MICHAEL GLEDHILL SOFTWARE VERSION: New 0200.01

FC 01000 STANDARD SYSTEM GLOBAL DATA

Abstract

<ul style="list-style-type: none"> 1 Block technical summary 2 Functional overview 3 Detailed block description 3.1 Enabling the clock memory byte 3.2 The PAL system tag table 3.3 Global logic signals 3.4 Global timing signals 3.4.1 Scan synchronised timing pulses 3.4.2 Scan synchronised timing square waves 3.5 Cyclically dependent signals 3.5.1 Record cycle times and properties 3.6 Real time clock (RTC) data 	<ul style="list-style-type: none"> 4 Parameters 5 Data structures & usage 6 Temporary (local) data 7 Block calls 8 Associated blocks 9 System block calls & data types 10 Special properties & requirements 10.1 Block optimisation 10.2 Calling requirements 11 Example usage & revision history
---	---

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 2.2 Typical PAL user documentation web page

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 overlaid on the screen “lightbox” imaging
 - Display code fragments
 - Display mathematical formulae

The PAL user documentation website will be distributed within the library software (distributed as part of the software project itself).

The PAL user documentation website will be available in its own right from with the PSP internal intranet.

The user documentation web pages will utilise the existing Practical Series of Publications web documentation facilities; these will be restructured to accommodate the folder organisation required by the TIA Portal User Documentation (TIA Portal designates specific folders each type of block documentation). The Practical Series of Publications web documentation facilities are designed to accommodate this type of restructuring.

2.4 Assumptions and limitations

- (1) The PAL software will be validated to the GxP requirements that are applicable to Europe and specifically, the United Kingdom at the time of writing.

2.5 Nonconformity

- (1) There are no nonconformities between this document and the User Requirements Specification (URS) [Ref. 003].
- (2) 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.
- (3) 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 9.3 of the Functional Specification (FS) [Ref. 005] 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.
- (4) The use of these modules is entirely optional.

2.6 Addressing the URS requirements

- (1) Where a particular point in the SDS addresses a formal requirement raised in the URS, the point in the SDS 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].
- (2) Paragraphs that are not numbered are not formally addressing a requirement; these may be introductions to a section, explanatory texts, notes or clarifying statements.

BLANK PAGE

3

Programming environments and common settings

- (1) The PAL software is written using the Siemens Simatic TIA Portal programming environment. This is a highly configurable workspace, supporting various windows, task panes and tool bars all of which are adjustable and selectable by the user. The default arrangement is shown below:

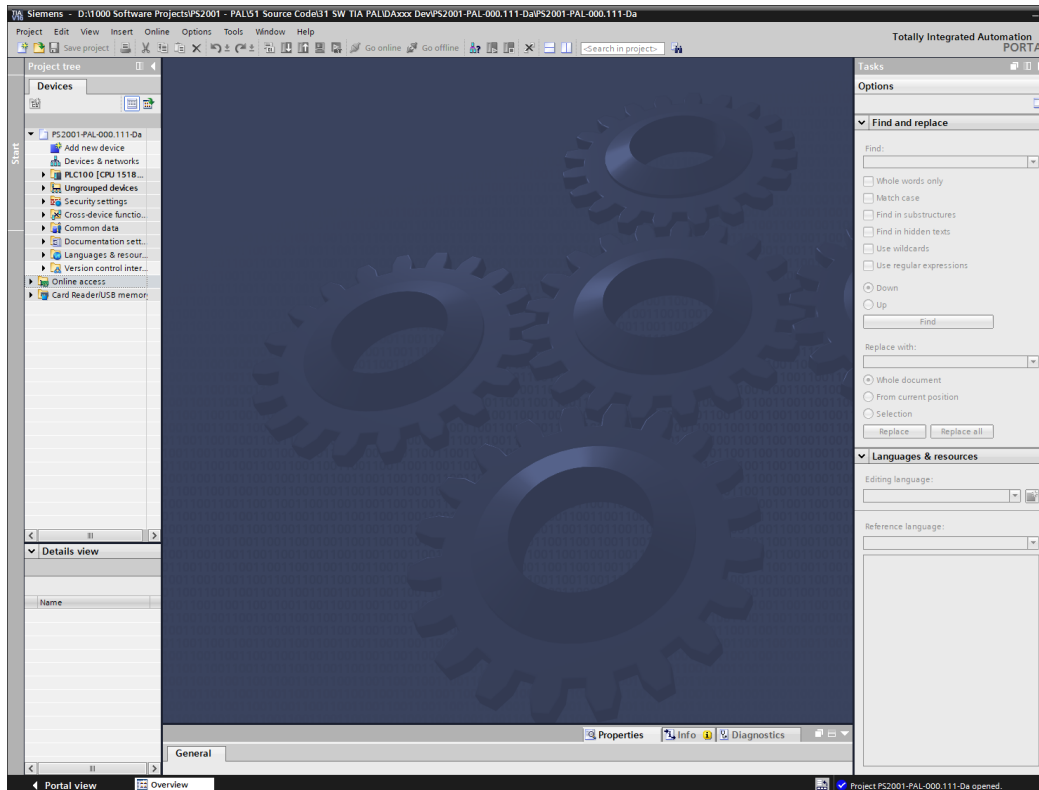


Figure 3.1 Default TIA Portal configuration

- (2) In addition to the TIA Portal configuration settings, TIA Portal also relies on the underlying Windows regional settings; there are some peculiarities with this arrangement that need to be addressed in order to give a consistent and uniform programming experience.
- (3) Finally, there are common settings that must be enabled for each CPU that is to run the PAL software, all of which are explained in the following sections:

3.1 Engineering stations and Windows settings

- (1) The computer upon which TIA Portal is installed is, in Siemens terminology, referred to as an Engineering Station or ES. This usually refers to a machine that holds the development software for the application (as opposed to just the runtime application for, say, a supervisory system). I.e. it is the programming environment for Controllers, HMIs and SCADA systems.
- (2) This section summarises the following aspects of an Engineering Station:
 - ① Operating system and hardware requirements of the ES
 - ② Assigning a fixed IP address to the ES
 - ③ Naming of the ES
 - ④ Regional and date and time setting of the ES

Full details of the Engineering Station configuration including details of all software installations is given in the ES/WDP Configuration Manual [Ref. 019].

3.1.1 Engineering station operating system and hardware specifications

- (1) The PAL software requires TIA Portal (professional) version 16 or higher. Version 16 was released in early 2020 and at the time of writing is the latest version of TIA Portal.
- (2) The professional version of TIA Portal is required, this is the version that supports the S7-1500 range of Controllers. The PAL software is designed specifically to run on the S7-1500 range of Controllers (it will also run to a lesser extent the S7-1200 range).
- (3) As of version 16, TIA Portal professional only supports the Windows 10 and Windows Server operating systems as shown in Table 3.1:

WINDOWS 10 (64-BIT)

Windows 10 Professional Version — build: 1809 and 1903
 Windows 10 Enterprise Version — build: 1809 and 1903
 Windows 10 IoT Enterprise 2015 LTSC, 2016 LTSC or 2019 LTSC

WINDOWS SERVER (64-BIT)

Windows Server 2012 R2 Standard (full installation)
 Windows Server 2016 Standard (full installation) †¹
 Windows Server 2019 Standard (full installation) †¹

Table 3.1 TIA Portal V16 supported operating systems

†¹ The full installation referred to here is the Desktop Experience option, this is selectable during the operating system installation process; the other “standard” option just installs a command line version of the operating system

Note: LTSB is the long-term service branch of Windows (now renamed LTSC, long-term service channel), this is a mechanism for restricting Windows updates to security and bug fixes and allowing the updates to be controlled by the system administrators. Essentially, this prevents the automatic updates implemented by the normal Windows 10 operating system.

(4) The Siemens minimum hardware requirements, and the PSP standard specification, for an Engineering Station running TIA Portal are:

FEATURE	SIEMENS RECOMMENDED (MIN)	PSP RECOMMENDED
Processor	Core i5-6440EQ, 3.4 GHz	Core i7-10875H, 5.1 GHz
RAM	16 GB (32 GB for large projects)	16 GB for large projects
Hard disk	SSD with at least 50 GB of free space	SSD with at least 512 GB of free space
Screen resolution	Single 1920 x1080 px monitor	Dual QHD (2560 x 1440) px monitors

Table 3.2 TIA Portal V16 hardware requirements

3.1.2 ES fixed IP address

- (1) TIA Portal always uses an Ethernet network to connect to individual Controllers. The Controllers and any supervisory systems (HMIs or SCADA) systems will all have fixed IP addresses; this is standard engineering practice with control systems (control system networks require individual devices to communicate with each other in a very specific way and via IP addresses. Unlike office networks where the given IP address of an individual machine is not particularly important).
- (2) Figure 3.2 shows an expanded version of the PAL network architecture, complete with an Engineering Station and Supervisory System:

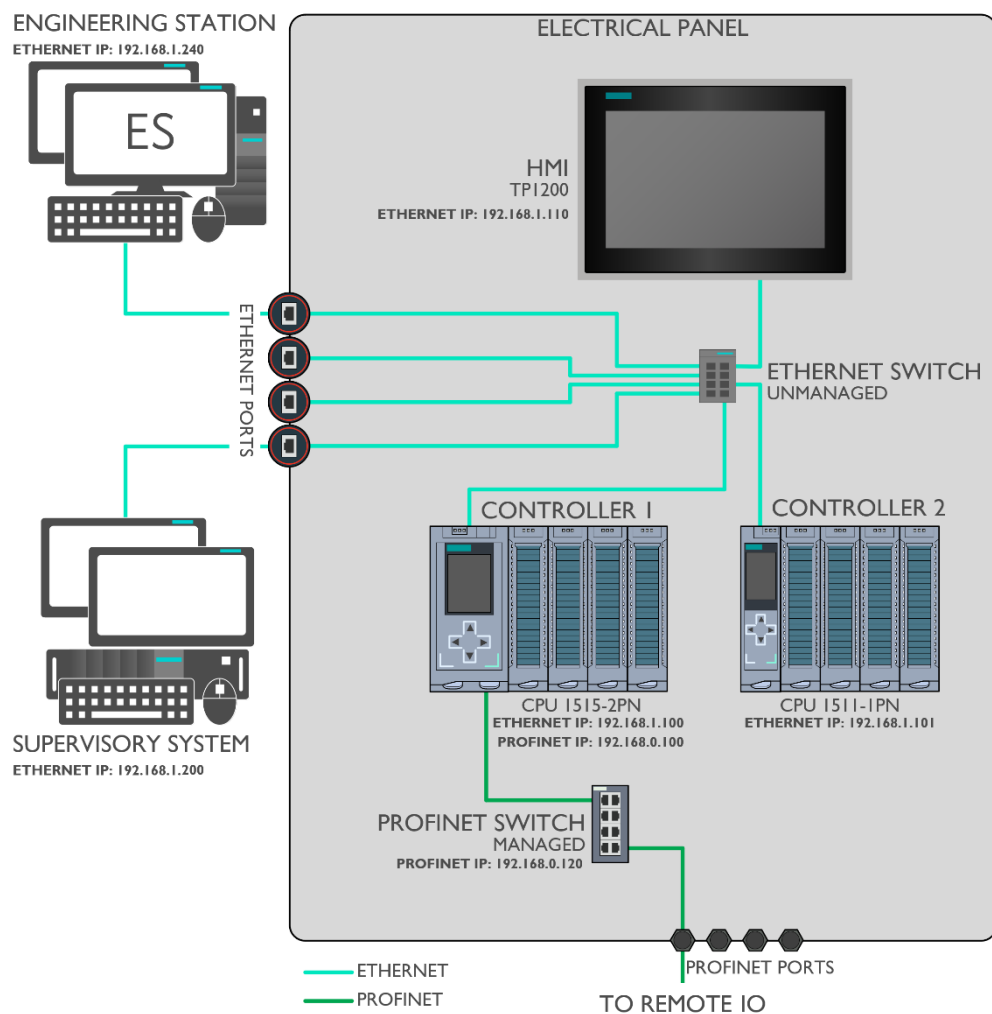


Figure 3.2 Expanded PAL network architecture

- (3) Within this arrangement there are five devices connected via the Ethernet network, these devices all have fixed IP address within the Ethernet Class C address range (192.168.1.nnn). The addresses are assigned as follows:

DEVICE NAME	IP ADDRESS	DESCRIPTION
CON100	192.168.1.100	Controller 1 — CPU 1515-2PN
CON101	192.168.1.101	Controller 2 — CPU 1511-1PN
OS200	192.168.1.200	Operator station (supervisory system)
ES240	192.168.1.240	Engineering Station (development system)

Table 3.3 Device names and IP addresses

- (5) The Engineering Station, the PC upon which the full TIA Portal professional development software is installed, has the fixed IP address of 192.168.1.240 and as such must be assign this fixed IP address within its Windows operating system.
- (6) The ES/WDP Configuration Manual [Ref. 019] gives full details of how to set a fixed IP address for and Engineering Station.

3.1.3 Naming the Engineering Station

- (1) All PCs that form part of the control system should have a network accessible name.
- (2) If the PC in question is to run either the WinCC Professional development environment or the WinCC Professional runtime environment, then the following point should be observed:

Importance of setting a PC name

If WinCC Professional is to be installed on a PC, it is very important to set the name of the PC **BEFORE** installing the TIA Portal software; in particular, before installing the WinCC Professional component. WinCC will hard code the PC name at the time of installation into the Microsoft SQL database manager setup.

It is very difficult to change the name of the PC after this without re-installing TIA Portal.

- (3) The Engineering Station PC will have been assigned a unique name (as part of the Windows installation process); However, Siemens (particularly WinCC) applications

have stricter naming conventions than is permissible in Windows. For Siemens applications the following rules apply:

Siemens computer name restrictions

- The following characters are not permitted:
.
,
;
:
!
?
"
'
^
~
_
+
=
/
\
|
@
*

\$
%
&
\$
°
(
)
[
]
{
}
<
>
`
'
space
- All character must be uppercase
- The first character must be a letter
- The computer name must be less than 12 characters

(4) The restrictions for Windows are less severe:

Windows computer name restrictions

- The following characters are not permitted:
?
"
*
:
/
\
<
>
|
- The name cannot start with a full stop .
- Keep the computer name less than 15 characters

(5) The general rules for naming a PC that is to run the PAL the software are:

- ① Use a dash instead of spaces
- ② Only use the characters [A-Z], the numbers [0-9] and the dash/hyphen [-]
- ③ The name must be less than 12 characters long
- ④ Start the name with a letter

The Siemens PC naming convention

- (6) There is a commonly accepted convention for naming the main equipment within a Siemen control system; it is to start the name with an abbreviation of what the device is (e.g. ES, OS, CON &c.) followed by the last octet (byte) of the device IP address.
- (7) In Figure 3.2, the Engineering Station has the IP address 192.168.1.240, the last octet of the IP address is thus, 240 and the PC name given to the Engineering Station would therefore be:

ES240

- (8) The following abbreviations are commonly used:

ABBREVIATION	DEVICE
AS	Automation System, another name for a PLC (generally used with distributed control systems like PCS 7)
ES	Engineering Station, the PC that runs the full development software (in this case TIA Portal)
OS	Operator Station, a supervisory system (HMI or SCADA). If the system is a server/client arrangement, OS refers to a client
CON	A Controller
SV	A server, usually a supervisory system server
PN	Profinet node, usually a remote IO rack, these have a different numbering arrangement (see § 4.1.1)

Table 3.4 Device naming abbreviations

- (9) The ES/WDP Configuration Manual [Ref. 019] gives full details of how to name and configure an Engineering Station.

3.1.4 Windows regional settings

- (1) TIA Portal is not responsive to the regional settings selected within windows². TIA Portal only uses the REGION AND LANGUAGE settings for ENGLISH (UNITED STATES).
- (2) Where other regions are used — ENGLISH (UNITED KINGDOM) for example, then TIA Portal will ignore this and use the default setting for ENGLISH (UNITED STATES). This tends to mean that dates always default to the American format of mm/dd/yyyy, and this is generally not acceptable.
- (3) The only way to change this is to set the region and language to ENGLISH (UNITED STATES) and then change the defaults to something more English (British) in nature.
- (4) The ES/WDP Configuration Manual [Ref. 019] gives full details of how to set the regional settings for an Engineering Station.

² This is an omission on the part of TIA Portal, all Windows programmes should adopt the region settings selected by Windows — it is however consistent with earlier versions of Siemens programming packages: Simatic Manager (Step 7) and PCS 7; both of which ignored the Windows regional settings.

3.2 TIA Portal settings

- (1) The PAL software will be entirely developed using the Siemens Simatic TIA Portal Professional (Version 16) programming environment.
- (2) To provide a common interface format, the PAL uses the default settings for TIA Portal as much as possible. However, there are a few areas where these settings need to be changed to match the requirements of the PAL.
- (3) The ES/WDP Configuration Manual [Ref. 019] gives full details of TIA Portal should be configured for an Engineering Station. The following sections summarise these settings:

3.2.1 Applying PAL settings to TIA Portal

- (1) Certain changes are needed to make the PAL appear in the correct format within TIA Portal (give TIA Portal a common appearance that matches the requirements of the PAL, variable naming length &c.), other changes are made to give a more consistent and convenient arrangement when using TIA Portal.
- (2) To make the changes, again open the settings page: select [OPTIONS](#) → [SETTINGS](#).
- (3) The following changes should be made:

GENERAL

AREA	OPTION	SETTING
General settings	User name	By default, this is the username of the current windows user. If this is not correct, change it here
Start view	<ul style="list-style-type: none">• Most recent	
	<ul style="list-style-type: none">• Portal	Select PROJECT
	<ul style="list-style-type: none">• Project	
View for objects in overview	<ul style="list-style-type: none">• Details	
	<ul style="list-style-type: none">• List	Select DETAILS
	<ul style="list-style-type: none">• Thumbnail	

PLC PROGRAMMING → LAD/FBD (LADDER/FUNCTION BLOCK DIAGRAM)

AREA	OPTION	SETTING
Operand field	Maximum width	Set to 26 characters

ONLINE & DIAGNOSTICS → DEFAULT CONNECTION PATH FOR ONLINE ACCESS

AREA	OPTION	SETTING
Default connection path for online access	Type of PG/PC interface	Select PN/IE
Default connection path for online access	PG/PC interface	Select the PC network card being used to connect to the controller (e.g. Intel PRO/1000)

VISUALISATION SETTINGS

AREA	OPTION	SETTING
Screens	Settings editor	Select SNAP TO GRID
Screens	Grid	Set grid size X & Y to 2 (this is the smallest size)
Screens	Standard screen size (RT Professional)	Set to match resolution of target monitor e.g.: width 2560 Height: 1440
Resize screen	Size adaptation of objects	Disable both, tick the boxes next to: DISABLE "FIT TO SIZE" FOR TEXT OBJECTS DISABLE "FIT TO SIZE" FOR GRAPHICAL OBJECTS

Table 3.5 Default TIA Portal settings adjustments for the PAL

3.2.2 TIA Portal block overview column settings

- (1) The block overview screen shows a summary of whatever is selected in the project tree. The most used application of the overview screen is to view the blocks within a project.
- (2) The configuration adopted for the overview screen is not project specific, once set TIA Portal will apply it to each subsequent project that is opened.
- (3) The following column settings should be selected:

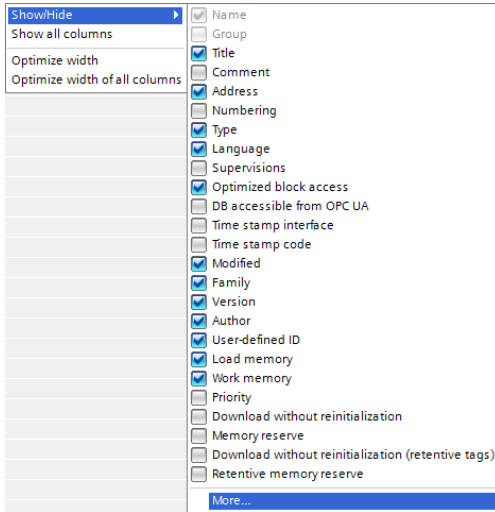


Figure 3.3 Show/hide columns

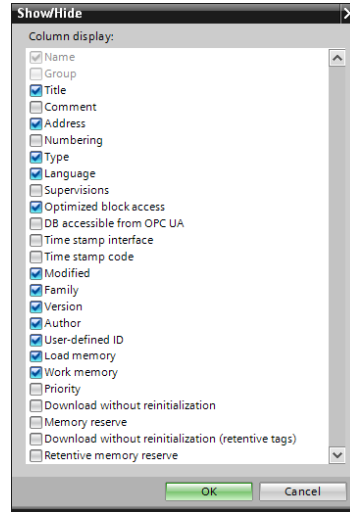


Figure 3.4 Show/hide columns dialog box

- (4) The correct order for the columns (from left to right) is:

- | | |
|--|--|
| <ol style="list-style-type: none"> ① Name ② Title ③ Address ④ Type ⑤ Language ⑥ Optimized block access ⑦ Modified | <ol style="list-style-type: none"> ⑧ Family ⑨ Version ⑩ Author ⑪ User-defined ID ⑫ Load memory ⑬ Work memory |
|--|--|

3.3 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). Those settings are described below.
- (2) The CPU properties are accessed from the **DEVICE CONFIGURATION** entry in the TIA Portal **PROJECT TREE** (Figure 3.5).
- (3) In the project tree select **DEVICE CONFIGURATION** ①, this opens an image of the Controller rack in the central area, right click the CPU ② and from the dropdown menu select **PROPERTIES** ③, this opens the **PROPERTY SETTINGS** window in the centre-bottom area ④.

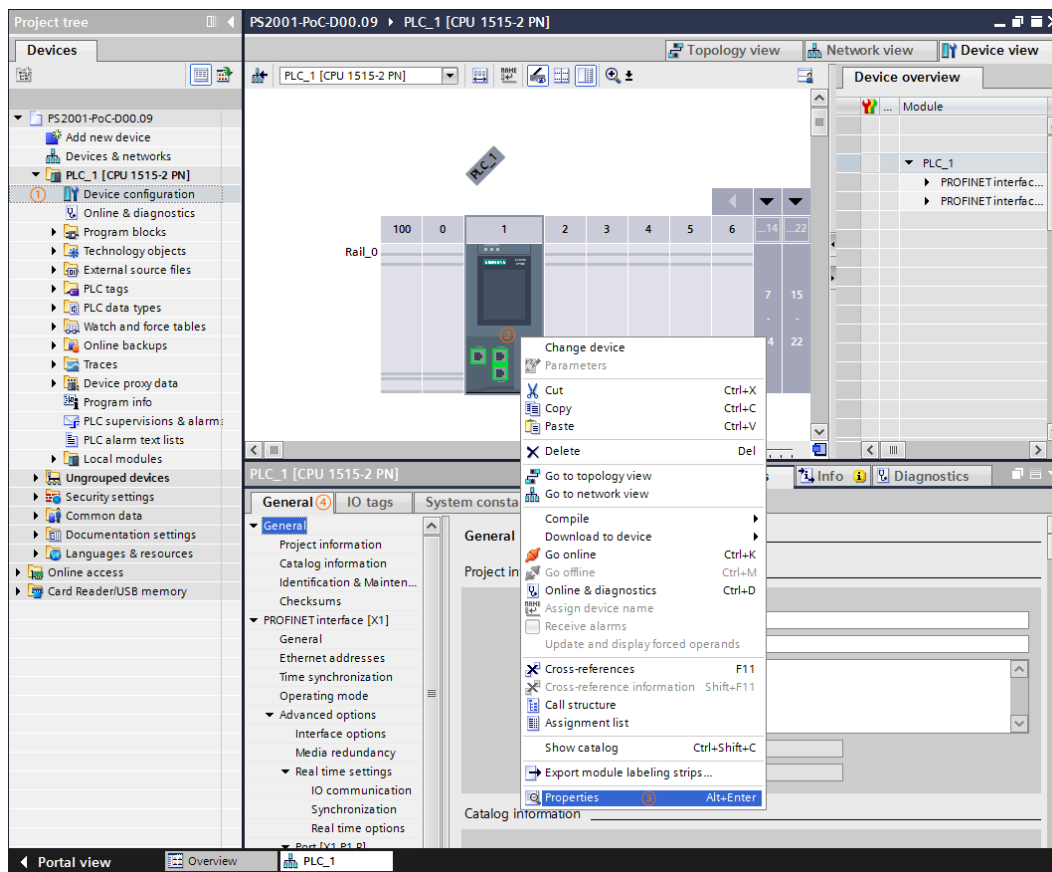


Figure 3.5 CPU properties

- (4) As with TIA Portal, the PAL keeps most CPU settings at the default values. The ones that must be changes are:

PROFINET INTERFACE (XI)

AREA	OPTION	SETTING
Interface networked with	Click ADD NEW SUBNET	This will change the network to PN/IE_1
IP Protocol	Tick SET IP ADDRESS IN THE PROJECT	Enter the correct IP address and subnet for the Controller

SYSTEM AND CLOCK MEMORY

AREA	OPTION	SETTING
System memory bits	Enable the use of system memory byte	This is unticked by default but it is very important that it remains unticked
Clock memory bits	Enable the use of clock memory byte	Ensure this box is 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	Ensure this box is ticked

Table 3.6 Default CPU setting adjustments for the PAL

BLANK PAGE

4

Naming, numbering and other conventions

- (1) The following give details of the conventions used to name the various components of the PAL software:
- ① Numbering conventions
 - ② Block naming conventions
 - ③ Parametric naming conventions
 - ④ Naming conventions for variables and constants
 - ⑤ Symbolic naming of IO and system tags
 - ⑥ Project specific tags
- (2) Some of these conventions are addressed in detail within the Functional Specification (FS) [Ref. 005], in these cases, the details are summarised (for completeness) in the following section; other conventions are explained here for the first time and are explained in more detail.

4.1 Block type and numbering conventions

⁽¹⁾ The PAL uses the block types and data structures available within the Simatic Controllers, these are summarised as follows:

- | | |
|------------------------------|---|
| 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 |
| ④ Data blocks (DB) | User configurable storage areas, generally used to store information required by the standard and application modules |
| ⑤ Instance data blocks (iDB) | Specialised form of a data block, used by FBs to store the retentive data required by the block |
| ⑥ User data type (UDT) | User defined data structures that can be stored in DBs and iDBs or passed as parameters to FCs and FBs |
| ⑦ System blocks | These are predefined blocks that perform specific functions; the blocks are built into each Controller or loadable via TIA Portal |

⁽²⁾ A broad outline of each of these block types and data structures is given in the following sections:

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 Section 8.

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 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 4.1.
- (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.

Data blocks (DBs)

- (12) 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 user (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)

- (13) Instance data blocks are 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)

- (14) 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*.
- (15) 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).
- (16) For clarity, the term UDT (User Data Type) will be used to specify a user defined data structure (or any of the other names it uses).

Built-in system blocks

- (17) 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.
- (18) 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).
- (19) Where system function blocks are used, these, like all FBs, will require an instance DB (see § 4.1.4); these function blocks will generally be contained (*called from*) within a standard module, and this standard module will 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.
- (20) 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).

4.1.1 Block numbering

- (1) The Controller blocks have the following number 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
UDT	Unlimited (symbolic addressing is used)	1-59999

Table 4.1 Controller block and UDT number ranges

- (2) 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)	Tmt	Instance data blocks
60000-60999	Template modules (interrupts)	Tmt	N/A
61000-65535	Documentation modules	Doc	N/A

Table 4.2 Block and number allocations for the PAL

(3) The PAL software itself is structured according to Figure 4.1

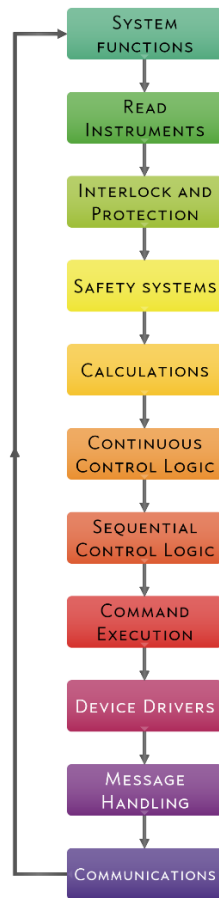


Figure 4.1 PAL structure

Each entry in this structure is referred to as a *function group*; All non-documentation 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.

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 4.1. This shows the complete PAL programme structure.

The structure of Figure 4.1 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 steps (it depends on the application in question). However, where a step is used, it must follow the execution order shown in Figure 4.1

Each of the functional groups in Figure 4.1 usually has both an application block and at least one standard module associated with it.

Table 4.3 expands on this arrangement.

4.1.2 Standard, application and template block numbering

(1) The PAL functional groups are allocated numbers within the block types as follows:

FUNCTION GROUP	STANDARD MODULE NUMBER	APPLICATION MODULE NUMBER	TEMPLATE MODULE NUMBER
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 (control loops)	FC 10ppp	FC 30nnn	FC 50nnn
Device drivers (valves)	FC 11ppp	FC 31nnn	FC 51nnn
Device drivers (drives)	FC 12ppp	FC 32nnn	FC 52nnn
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 4.3 Functional group summary

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

(2) 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).

4.1.3 Data block numbering

- (1) Data blocks are the primary mechanism for storing data within the PAL and for passing data between blocks.
- (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) 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.
- (6) 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.
- (7) 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.

- (8) 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 4.4 PAL static and dynamic data block numbering ranges

- (10) 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.
- (11) For example, if FC 10001 is used, the static UDT will have number 10001 and the dynamic UDT will have number 30001.
- (12) Similarly, the data blocks that hold the static and dynamic data will have the same numbers as the UDT.
- (13) Extending the previous example, FC 10001 would have static UT 10001 and Dynamic UT 30001, these would be stored in DB 10001 (static data) and DB 30001 (dynamic data).

4.1.4 Instance data block numbering

- (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 4.5 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.

4.1.5 OB (Interrupt block) numbering

(1) Interrupt blocks are of two types: general event interrupts (non-fault) that detect specific events within the Controller (e.g. a time of day, hardware interrupt &c.) and fault interrupts that detect errors and other adverse conditions). The following give a full list of both types of interrupt blocks:

(2) General, non-error interrupts:

OB NUMBER	PAL MODULE NAME	DESCRIPTION
OB 1	OB00001_IntlNrmMainProgram	Controller main program cycle Called at the start of each Controller cycle
OB 10	OB00010_IntlNrmNTimeOfDay	Time of day Interrupt Called by time and day of week
OB 20	OB00020_IntlNrmNTimeDelay	Time delay Interrupt Called after a specified delay has expired
OB 30	OB00030_IntlNrmNCyclic	Cyclic Interrupt Called at specified intervals
OB 40	OB00040_IntlNrmNHardware	Hardware Interrupt Called when a specified signal is detected
OB 100	OB00100_IntlNrmNStartUp	Start-up Interrupt Called when the CPU transitions to RUN

Table 4.6 Non-error interrupt modules and organisation blocks

(3) Fault (error) condition interrupts:

OB NUMBER	PAL MODULE NAME	DESCRIPTION
OB 80	OB00080_IntlErrECycleTimeErr	Error Interrupt Maximum cycle time exceeded
OB 82	OB00082_IntlErrEModuleDiag	Error Interrupt Module diagnostics signal received (module fault)
OB 83	OB00083_IntlErrEModuleChange	Error Interrupt Module changed, removed or installed
OB 86	OB00086_IntlErrERackErr	Error Interrupt Pack failure or fault
OB 40	OB00121_IntlErrEProgramErr	Error Interrupt Programming fault or error
OB 100	OB00122_IntlErrEIOErr	Error Interrupt IO card access fault

Table 4.7 Fault interrupt modules and organisation blocks

4.1.6 Document block numbering

- (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 is 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.

4.1.7 Block numbering summary

- (1) There are 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 PAL also supports user data types (UDTs), these are used to define and organise the data needed by each standard module, generally a standard module will have both static data (holding the fixed, configuration information for the module) and dynamic data (the live, changing data required by the module).

- (4) The data required by the standard modules (and defined in the UDTs) is held in data blocks, these being designated *static data blocks* (holding multiple instances of the static UDT) and *dynamic data blocks* (holding multiple instances of the dynamic UDT).
- (5) A third type of data block, the instance data block, is needed whenever a function block (FB) used.
- (6) In summary, 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

- (7) 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	OB00001-00122	Int	Interrupt handling modules
FC/FB	FC/FB00001-19999	Std	Standard modules
FC/FB	FC/FB20001-39999	App	Application modules
FC	FC40000-60999	Tmt	Template modules
FC	FC61000-65535	Doc	Document modules
UDT	UT00001-19999	St_	Static data structure
UDT	UT20001-39999	Dy_	Dynamic data structure
DB	DB00001-19999	St_	Static storage data block
DB	DB20001-39999	Dy_	Dynamic storage data block
iDB	DB40000-59999	iDB	Instance data blocks (associated with FBs)

Table 4.8 Full range and type of module numbering for the PAL

- (8) Each of these number ranges is broken down further in relations to the subdivisions within the PAL software structure.

4.2 Module naming Conventions

- (1) Within the PAL all modules (blocks and UDTs) are given a name; that name has a particular structure that includes the block type (above), the block number (address), a block class, a block function attribute and a description formatted as follows:

`BBnnnnn_CccFffffDdddddd`

Where:

ITEM	MEANING	DETAILS
<code>BB</code>	Block type	2 characters
<code>nnnnn</code>	Block number	5 digits (in the range 00000-59999)
<code>Ccc</code>	Block class	3 characters (see block class below)
<code>Fffff</code>	Block function	5 characters max (see block function below)
<code>Ddddddd</code>	Block description	Short description of the block

Table 4.9 Block naming components

Each of these is summarised in the following sections:

4.2.1 Block type

- (1) There are four common types of block associated with Siemens Controllers, three programmable blocks (functions, function blocks and organisation blocks), and global data storage blocks (data blocks). There are also instance data blocks that hold information for specific function blocks. Finally, the PAL treats User Data Types as blocks (UDTs are not blocks, but the PAL always associates them with a specific block and names them accordingly).

- (2) Each of these block types (and the UDTs) is given a two-letter abbreviation, the block identifier (**BB**) that uniquely identifies its type within the software:

BLOCK ID	MEANING
FB	Function block
FC	Function
OB	Organisation block
DB	Data block
ID	Instance data block
UT	User data type

Table 4.10 Two letter block abbreviations

- (3) All block names (and UDT names) will start with one of these block identifier abbreviations.

4.2.2 Block number

- (1) The block number (**nnnnn**) is simply the five-digit block number (with leading zeros) given by the functional groupings, Table 4.3 and the general block numbering arrangements, Table 4.2

4.2.3 Block class

- (1) The block class (**Ccc**) is abbreviated as follows:

ABB.	CLASS	MEANING
Std	Standard	Standard block
App	Application	Application block
Int	Interrupt	Interrupt block
Tmt	Template	Template block
Doc	Documentation	Documentation block
Dy_	Dynamic	Data block and UDT only (contains live, dynamic, data)
St_	Static	Data block and UDT only (contains static data)
Rc_	Recipe (semi static)	Data block and UDT only (data is loaded from a recipe)

Table 4.11 Block naming classes

4.2.4 Block function

(1) The block function (Fffff) is abbreviated as follows:

ABB.	FUNCTION	MEANING
Sys	System	System block
Inst	Instrumentation	Instrument block
ILock	Interlocks	Interlock, permissive and trip logic
Safe	Safety	Safety systems
Calc	Calculations	Calculation and mathematics
Cont	Continuous	Continuous control logic
Seq	Sequence	Sequential control logic
Cmd	Command	Command handling
Dev	Device drivers	Device drivers
Msg	Messages	Alarm, warning, event and prompt handling
Comms	Communications	Communication handling
Sub	Subroutines	Subroutine functions
INrm	Normal Interrupts	Normal (non-error) interrupt functions
IErr	Error Interrupts	Error interrupt functions
Debug	Debug	Debug functions
Gen	General	General (or global) usually applicable to documentation

Table 4.12 Block naming Functions

4.2.5 Block description

- (1) The block description (**Dddddddd**) 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
ValveMod	Modulating valve
DriveVSD	Variable speed drive

Table 4.13 PAL block naming — description

- (2) Block descriptions are always written without spaces using *camel case*³; typically, block descriptions should be 12 characters or fewer in length.

4.2.6 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

³ Camel case is the practice of joining words together and capitalising the start of each word, it is more formal known as *medial capitals*).

4.3 Block optimisation & IEC check

- (1) Within the PAL, all blocks (OB, FB, FC and DB) should be set to use **OPTIMIZED BLOCK ACCESS**, this is selected from the block properties
- (2) The block properties are accessed by right clicking the block (either in the project tree or on the overview screen) and selecting **PROPERTIES** from the dropdown list, this opens the properties dialogue box, block access is under the **ATTRIBUTES** section:

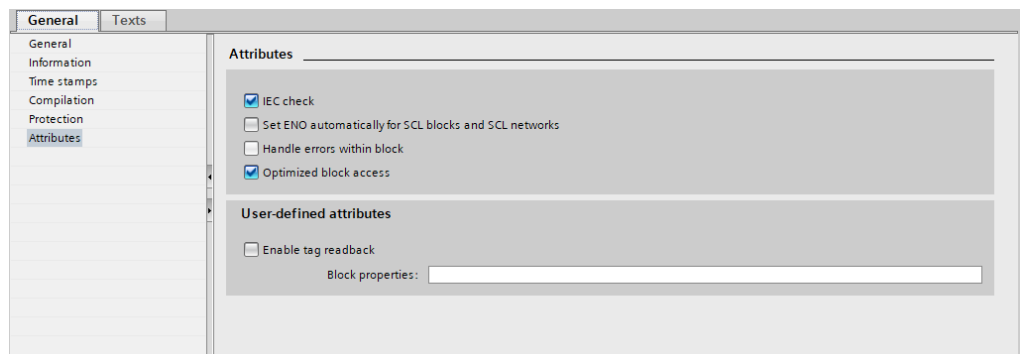


Figure 4.2 Optimized block access

- (3) By default, **OPTIMIZED BLOCK ACCESS** is activated in TIA Portal and generally, this is the arrangement that is wanted, it allows for symbolic addressing of data within block parametric interfaces and in data blocks. It gives faster access within the Controller to data elements.
- (4) There are occasions (usually for non-Simatic supervisory systems) where data blocks need to be addressed absolutely, under these circumstances it is permissible to disable **OPTIMIZED BLOCK ACCESS** for the blocks in question
- (5) The **IEC CHECK** box must be ticked for all standard modules (by default it is unticked), this ensures that the block is compliant with all aspects of IEC modules given in IEC6113-3 [Ref. 012], particularly in reference to library module standards.

4.4 Tags, parameters, symbolic and absolute representations

- (1) Tags, parameters and symbols are terms that are commonly renamed, interchanged and generally misused in PLC programming. To clarify things, this section explains the correct usage and application of the terms. They are shown in context below:

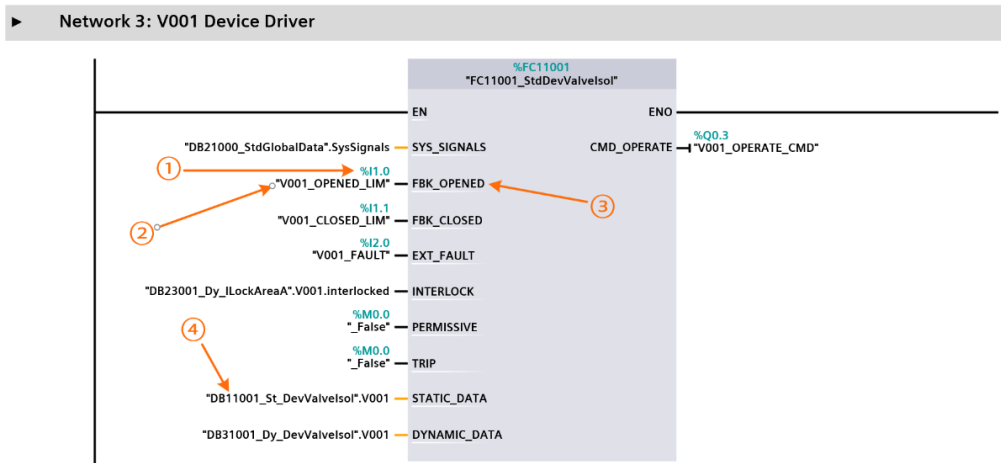


Figure 4.3 Tags, parameters, symbols and absolute addresses

- (2) Figure 4.3 shows tags with both symbolic and absolute representation and parameters, both formal and actual.
- (3) **Tags** are *symbolically* named items within the Controller. Most tags are associated with an element that has an *absolute address*; such elements being inputs, outputs, internal memory, timers, counters &c.
- (4) In Figure 4.3, item ② is a tag with the symbolic name `V001_OPENED_LIM` and the absolute address `I1.0` (this is shown as item ①, and preceded with a % sign — indicating an absolute address).
- (5) Some tags (particularly those associated with data blocks) do not have an associated absolute address (item ④); such tags are identified purely by their *symbolic* name.
- (6) **Parameters:** both functions (FCs) and function blocks (FBs) support the passing of information to and from the block with the use of *parameters*. These parameters are defined within the block itself and are apparent when the block is called as the

parameter name shown within the block (item ③) in Figure 4.3; the parameters have a short connection point outside the block to which logic instructions or tags can be attached.

- (7) These connection points have different colours depending upon the data type of the connection: all Boolean connection points are coloured **black**; these connections can have logical instructions connected to them.
- (8) All other data type connections (integers, real, strings, UDTs, &c) are coloured **orange**; these connections are to direct data points (e.g. some variable within a data block, a constant or local variable or even a hardcoded value) and do not support the connection of instructions.

The block parameter itself, item ③ is referred to as the formal parameter. The tag attached to the parameter (item ② in this case) is called the **actual parameter**.

- (9) As a convention within this document, formal parameters are referred to as just “parameters”, where a distinction is made with actual parameters; the actual parameter will be referred to as an “actual parameter”. This distinction is often not necessary; it is usually made clear by the context of the subject under discussion.
- (10) 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
- (11) It is for this reason that the common system logic and timing signals (see § 4.10.1) are passed parametrically to the block in the **SYS_SIGNALS** parameter; all standard modules have this parameter

4.4.1 EN and ENO parameters

- (1) Within the TIA Programming environment all functions and function blocks have the Boolean connections **EN** (**ENABLE IN**) and **ENO** (**ENABLE OUT**). By default the **ENO** connections are configured in TIA Portal to always return a **true** value (the signals are said to be disabled); this means that blocks can be daisy chained together in a line (the **EN** of downstream block being connected to the **ENO** of the preceding block):

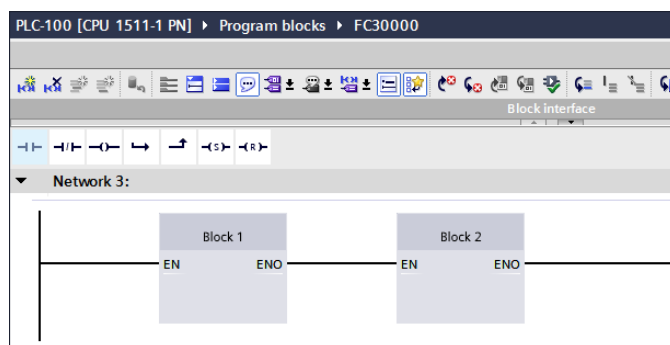


Figure 4.4 Daisy chaining blocks with ENO disabled

- (2) With **ENO** disabled, block 2 will always execute (**ENO** being set constantly to a **true** value). In programming terms, it is identical to the following:

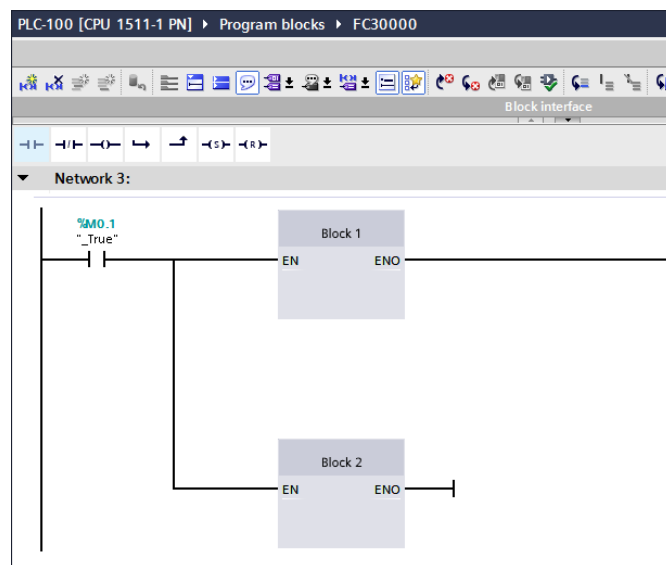


Figure 4.5 Equivalent call mechanism with ENO disabled

- (3) If the **ENO** is not disabled (right click the block and select **GENERATE ENO** from the dropdown), then the function can influence the state of the **ENO**; it could set it to a **false** value and if this were the case block 2 of Figure 4.4 would not be executed.
- (4) The PAL maintains the TIA Portal default of disabling **ENO** on all blocks. This allows blocks to be daisy chained together in the manner of Figure 4.4.
- (5) The **EN** is used to enable the calling of the block (or instruction) to which it is connected. If the **EN** connection is **false**, the block will simply not be executed (and neither will any downstream blocks if they are daisy chained).
- (6) The use of **EN** is permitted under the PAL, although none of the blocks within the PAL use it themselves. Blocks written by the user can use the **EN** as they see fit (even to call the PAL standard blocks). The only caveat being that daisy chained blocks will not be executed if any upstream **EN** signal is **false**.
- (7) Generally, it is better practice to use the structure of Figure 4.5; this is more unequivocal and is the structure used within the PAL templates.
- (8) Some instructions also have the **EN** and **ENO** functions; these are often used within the PAL.

*Note: In Figure 4.5 the always true signal (**_True**) is used before calling multiple blocks, this is a requirement of TIA Portal, branches can only be inserted after an instruction; it is a short hand way of grouping block calls and is used extensively within the PAL software.*

4.5 Block parameter naming

- (1) Both functions (FCs) and function blocks (FBs) support the passing of information to and from the block with the use of *parameters*. These parameters are defined within the block and are apparent when the block is called as the parameter name shown within the block itself.
- (2) The parameters are specified by editing the interface of the block from within the block editor (the [BLOCK INTERFACE](#)):

	Name	Data type	Default value	Comment
1	Input ①			
2	SYS_SIGNALS	*UT21000_Dy_SysSignals*		The system logic and timing signals for parametric access
3	FBK_OPENED	Bool		Valve opened limit switch (1 = opened limit reached)
4	FBK_CLOSED	Bool		Valve closed limit switch (1 = closed limit reached)
5	EXT_FAULT	Bool		Valve external fault signal (1 = fault, 0 = healthy)
6	INTERLOCK	Bool		Valve interlock (1 = interlock active, 0 = OK to operate)
7	PERMISSIVE	Bool		Valve permissive (1 = permissive active, 0 = OK to operate)
8	TRIP	Bool		Valve trip (1 = trip active, 0 = OK to operate)
9	STATIC_DATA	*UT11001_St_DevValvelsol*		Valve static data storage (UDT)
10	Output ②			
11	CMD_OPERATE	Bool		Valve operate signal (1 = valve energised, 0 = de-energised)
12	InOut ③			
13	DYNAMIC_DATA	*UT31001_Dy_DevValvelsol*		Valve dynamic data storage (UDT)
14	Temp ⑤			
15	wrkOpCmd	Bool		Operating Command (internal working value)
16	Constant ⑥			
17	k_MIN_OP_TIME	Real	0.2	Minimum operation time (seconds)
18	Return ④			
19	FC11001_StdDevValvelsol	Void		NOT USED

Figure 4.6 Formal parameter declarations

4.5.1 Formal parameters

- (1) The formal parameters are divided into four groups:
 - ① INPUT
 - ② OUTPUT
 - ③ INOUT
 - ④ RETURN
- (2) The fourth group (**RETURN**) is not generally used within the PAL (or indeed 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.
- (3) The **RETURN** parameter is not used within the PAL, but the PAL does *permit* its use, however, it is not common practice to use this parameter; where it is used it is generally employed to return fault codes from the function.
- (4) The remaining parameter types (**INPUT**, **OUTPUT** and **INOUT**) are widely used throughout the PAL (particularly by the standard blocks).
- (5) The PAL naming convention for formal parameters is as follows:
 - ① Parameter names use uppercase characters only
 - ② Only the characters [A-Z], the numbers [0-9] and the underscore [**_**] can be used
 - ③ Spaces are represented by the underscore character
 - ④ The parameter name must not start with a number or underscore (do not use consecutive occurrences of the underscore)
 - ⑤ The parameter name must be 15 characters or less in length

- (6) All parameters must have a comment in the block interface to explain the function of the parameter. This is important; the text in the comment field will appear as a tool tip when hovering over the parameter on a called version of the block.

4.5.2 Temporary (local) data

- (1) Local (or temporary) data (point ⑤ in Figure 4.6) is used to store temporary or intermediate data locally to the block within which it is defined. The data is not accessible to any other block (i.e. any data stored within the **TEMP** area is not accessible externally to the block).
- (2) Local data is not permanent; data within a local variable is only present until the end of the block is reached. The data has to be reinitialised (or recalculated) each time the block is executed.
- (3) Where local data is used within a PAL block it is always given a symbolic name starting with either the prefix **wrk** (for a working or intermediate value), **act** (for an actual value), **cal** to denote a calculation has taken place or **seq** if it is part of a sequential control block. The rest of the name is given in camel case. E.g.:

```
actElapsedTime  
wrkPermActEn  
calDeadbandPercentage  
seqThisStep
```

- (4) Two other prefixes **rev** and **lic** are also used, these store revision and licensing information and are common to all blocks.

- (5) The basic restrictions on local variable names within the PAL are:
- ① The name must be prefixed with `rev`, `lic`, `act`, `wrk`, `cal` or `seq`
 - ② The rest of the name must be written in camel case
 - ③ The name (including prefix) must be no more than 24 characters
 - ④ Only use the characters [a-z], [A-Z], the numbers [0-9] and the underscore character [`_`]
- (6) All local data must have a comment in the block interface to explain the function of the variable.

4.5.3 Constants

- (1) Constants are declared in the `CONSTANT` area of the `BLOCK INTERFACE` (point ⑥ in Figure 4.6). These are defined with a data type and an `INITIAL VALUE`. This `INITIAL VALUE` is the value given to the constant. Constant values are constant (*somewhat obviously*) and cannot be changed within the block (any attempt to write a value to a constant will be reported as an error).
- (2) Constants can only be used within the block where they are defined; they are not available to other blocks.
- (3) Where a constant is used within a PAL block it is always given a symbolic name starting with the prefix `k_`. The rest of the name is given in uppercase. E.g.:

```
k_MIN_TIME  
k_SECONDS_PER_HOUR
```

- (4) The basic restrictions on constant names within the PAL are:
- ① The name must be prefixed with `k_` (all lowercase)
 - ② The rest of the name must be written in uppercase
 - ③ The name (including prefix) must be no more than 21 characters
 - ④ Only use the characters `[k]`, `[A-Z]`, the numbers `[0-9]` and the underscore character `[_]`
 - ⑤ The underscore character should be used in place of a space to separate words
- (5) All constants must have a comment in the block interface to explain the function and usage of the constant.

4.5.4 Static data (function blocks only)

- (1) Function blocks (FBs) support the use of static (local) data. Static data is permanent data that is stored in the instance data block associated with the function block. Static data is retained permanently between repeated calls of the block (the data will be present until changed by some operation within the block).
- (2) Static data is specified by editing the interface of the block from within the block itself (the **BLOCK INTERFACE**), similar to the formal parameters of Figure 4.6
- (3) Where static data is used within a PAL block it is always given a symbolic name starting with the prefix `St_`. The rest of the name is given in camel case. E.g.:

```
St_LastState  
St_PreviousCount
```

- (4) The basic restrictions on static variable names within the PAL are:
 - ① The name must be prefixed with `St_`
 - ② The rest of the name must be written in camel case
 - ③ The name (including prefix) must be no more than 24 characters
 - ④ Only use the characters [a-z], [A-Z], the numbers [0-9] and the underscore character [`_`]
- (5) All static data must have a comment in the block interface to explain the function of the variable.

4.6 Naming variables in static UDTs

- (1) Static UDTs are static only in terms of how they are used within the PAL, all UDTs are of a read/write nature (it is not possible to declare a constant within a UDT — as is the case with FCs and FBs).
- (2) Static UDT data should not be overwritten by any operation within the PAL (there is an exception for recipe data, see § 4.7.1.).
- (3) Entries within a static UDT are given symbolic names in uppercase:

Name	Data type	Default val.	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Comment	
1	CONFIG_ALM_H_EN	Bool	false	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - High Alarm is Enabled (1 = Enabled, 0 = No Alarm)
2	CONFIG_ALM_L_EN	Bool	false	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Low Alarm is Enabled (1 = Enabled, 0 = No Alarm)
3	CONFIG_DEV_H_EN	Bool	false	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - High Deviation is Enabled (1 = Enabled, 0 = No Alarm)
4	CONFIG_DEV_L_EN	Bool	false	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Low Deviation is Enabled (1 = Enabled, 0 = No Alarm)
5	CONFIG_FP_OFF	Bool	false	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Faceplate is disabled (1 = No Faceplate, 0 = Normal)
6	CONFIG_MAN_OFF	Bool	false	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Manual Mode is disabled (1 = Disabled)
7	CONFIG_SP_RESET	Bool	false	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Reset PID Block when in CTRL_SP - One Shot
8	CONFIG_REV_ACT	Bool	false	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Loop is Reverse Acting (1 = Reverse PV-SP, 0 = Normal SP-PV)
9	CONFIG_DEADBAND	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Deadband (Engineering Units)
10	CONFIG_P_TERM	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Proportional termTerm
11	CONFIG_I_TERM	Time	TR0ms	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Integral Term (ms) - Small number, fast response
12	CONFIG_D_TERM	Time	TR0ms	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Derivative Term (ms) - Large number, fast response
13	CONFIG_LIM_OUT_MAX	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Max Value of CV
14	CONFIG_LIM_OUT_MIN	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Min Value of CV
15	CONFIG_ALM_H	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - High Alarm Limit
16	CONFIG_ALM_L	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Low Alarm Limit
17	CONFIG_ALM_HYST	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - High/Low Alarm Hysteresis
18	CONFIG_DEV_H	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - High Deviation Alarm Limit
19	CONFIG_DEV_L	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Low Deviation Alarm Limit
20	CONFIG_DEV_HYST	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Deviation Alarm Hysteresis
21	CONFIG_DEV_DEL_PRE	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Deviation Alarm Time Filter - Preset (s)
22	CONFIG_SP_LIM_MAX	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Setpoint Maximum Value (Engineering Units)
23	CONFIG_SP_LIM_MIN	Real	0.0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Config - Setpoint Minimum Value (Engineering Units)

Figure 4.7 Example static UDT

E.g.:

CONFIG_ALM_H_EN

CONFIG_FP_OFF

- (4) The basic restrictions on static UDT element names within the PAL are:
- 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.
- (6) All elements within a static UDT should be declared as **SETPOINTS** (the **SETPOINT** box should be ticked).

4.7 Naming variables in dynamic UDTs

- (1) Dynamic UDTs can be freely overwritten by blocks within the PAL, (there are no restrictions).
- (2) Entries within a dynamic UDT are given symbolic names in camel case.

Name	Data type	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Comment
Status_AutoOff	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Status - PID in Auto with Loop turned off (zero output)
Status_AutoSP	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Status - PID in Auto with Set-point
Status_AutoFixOp	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Status - PID in Auto with Fixed Output
Status_ManOff	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Status - PID in Manual with Loop turned off (zero output)
Status_ManSP	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Status - PID in Manual with Set-point
Status_ManFixOp	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Status - PID in Manual with Fixed Output
Status_ILock	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Status - PID is Interlocked (1=Interlock Active)
Mode_Auto	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Automatic Mode (1 = Auto)
Mode_Man	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Manual Mode (1 = Manual)
Mode_Local	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Local Control Mode (1 = Local HMI Control)
Mode_Remote	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Remote Control Mode (1 = Remote SCADA Control)
Mode_BypassOn	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Interlock Bypass (1 = Bypass on, IL ignored)
Mode_AutoOff	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Automatic Mode - Loop OFF (1 = Off)
Mode_AutoSP	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Automatic Mode - Loop PID Mode Operating to Set-point (1 = SP Mode)
Mode_AutoFixOp	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Automatic Mode - Loop in Fixed Output Mode (1 = FixOP Mode)
Mode_ManOff	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Manual Mode - Loop OFF (1 = Off)
Mode_ManSP	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Manual Mode - Loop PID Mode Operating to Set-point (1 = SP Mode)
Mode_ManFixOp	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Manual Mode - Loop in Fixed Output Mode (1 = FixOP Mode)
Mode_D6	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Spare
Mode_D7	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Spare
Mode_D8	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mode - Spare

Figure 4.8 Example dynamic UDT

E.g.:

Status_BypassOn

Mode_AutMan

- (3) The basic restrictions on dynamic UDT element names within the PAL are:
 - ① The name must be written in camel case
 - 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.
- (5) All elements within a dynamic UDT **must not** be declared as **SETPOINTS** (the **SETPOINT** box should always be unticked).

4.7.1 UDTs holding recipe data

- (1) Under certain (very limited) circumstances, the data in a static UDT 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 UDT (essentially the static UDT is being selected for a particular set of production requirements).
- (3) Once a recipe has overloaded a static UDT, the data in that UDT is then fixed (and will not be overwritten) until the operator selects a different recipe.
- (4) Data blocks that hold static UDTs that are under the control of a recipe are given the class **Rc_** (rather than **St_**), the UDT retains the **St_** designation and retain the properties specified in § 4.6 (i.e. all uppercase &c.).

4.8 Naming variables in static DBs

- (1) Static DBs are static only in terms of how they are used within the PAL, all DBs are of a read/write nature (it is not possible to declare a constant within a DB — as is the case with FCs and FBs).
- (2) Static DB data must not be overwritten by any operation within the PAL (there is an exception for recipe data, see § 4.9.1.).
- (3) Entries within a static DB are given symbolic names in uppercase.
- (4) The basic restrictions on static DB element names within the PAL are:
 - 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.
- (6) All elements within a static DB should be declared as **SETPOINTS** (the **SETPOINT** box should be ticked).

4.9 Naming variables in dynamic DBs

- (1) Dynamic DBs can be freely overwritten by blocks within the PAL, (there are no restrictions).
- (2) Entries within a dynamic DB are given symbolic names in camel case.
- (3) The basic restrictions on dynamic DB element names within the PAL are:
 - ① The name must be written in camel case
 - 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.
- (5) All elements within a dynamic DB **must not** be declared as **SETPOINTS** (the **SETPOINT** box should always be **unticked**).

4.9.1 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* 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 that 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 in § 4.8 (i.e. all uppercase &c.).

4.10 Tags and tag naming

- (1) Tags are a mechanism for giving symbolic names to absolutely addressable elements within the Controller itself (timers, counters, inputs, outputs, bit memories &c.).
- (2) Where possible, the PAL uses data blocks to store information and it does not use timers and counters⁴ at all. It does however use some (*but not very many*) bit memories (the number of bit memories available to the S7-1500 range of CPUs is 16384 bytes of data or 131,072 bits) and it also uses symbolic addressing for IO signals.
- (3) The system tags and the IO tags are stored in separate tag tables (as follows):

4.10.1 The PAL system tags (PAL_SystemTags)

- (1) The PAL uses bit memories to store certain global system signals (these are the signal generated for use by FC 01001, the STDSYSGLOBALDATA block).
- (2) The PAL uses three bytes of the bit memory range for specific global signals (Table 4.14). The bytes used are MB0, MB1 and MB10. MB10 is populated by the clock memory signal (generated within the CPU).
- (3) The PAL stores all these tags in their own tag table:

PAL_SystemTags

4

The number of timers and counters available within Siemens Controllers is restricted (although it is better than it was) 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 blocks that again store counts in data blocks and again any number of which are supported.

(4) It contains the following tags:

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 4.14 PAL system bit memory usage

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

- (6) Similarly, the bit memories contained in the bytes **MB0**, **MB1** and **MB10** are reserved by the PAL and must not be reallocated, renamed or used in any other tag table.
- (7) All PAL system tags contained within the **PAL_SystemTags** tag table are identified by a leading underscore character (**_**).
- (8) The PAL system tags 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 [**_**]
- (9) All PAL system tags have a brief explanation of what the tag does stored in the comment field of the tag.

4.10.2 The PAL Input/Output tags (PAL_IOTags**)**

- (1) The inputs and outputs associated with a project are unique to that project (they obviously depend on the plant being controlled). The PAL does not prescribe in anyway what IO can be used. It does however have certain rules for how that IO should be named and where the tags should be stored.
- (2) Taking the later point first, the PAL stores all the IO tags in their own tag table:

PAL_IOTags

(3) The following is an example of an IO tag table:

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 healthy/tripped
M002_RUNNING	Bool	%I0.3	M002 is running/stopped
M002_FAULT	Bool	%I0.4	M002 is healthy/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 4.15 PAL IO tag table (example)

(4) 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

(5) The Functional Specification § 6.2 [Ref. 005] contains more details about the naming of IO tags.

4.10.3 Project specific tag tables

- (1) The PAL defines two tag tables:

TAG TABLE NAME	FUNCTION	USER CONFIGURABLE
PAL_SystemTags	Contains the global system signals generated as part of the PAL system logic	No
PAL_IOTags	Contains all the IO signal tags connected to the controller	Yes

Table 4.16 PAL tag tables

- (2) Of these, only the first one ([PAL_SystemTags](#)) is actually required; it is perfectly possible for a Controller to not have any IO signals (it may be some form of communications coordinator for example).
- (3) If the Controller does have IO, then the IO tags must be stored in a tag table called [PAL_IOTags](#).
- (4) The user is at liberty to create any other tag tables that may be required (there are a lot of memory bits available should the user wish to make use of them). These tag tables are referred to as *project specific tag tables*.
- (5) Project specific tag tables should be prefixed with [PROJ_](#) (to differentiate from the PAL tag tables that are prefixed [PAL_](#)) as follows:

[PROJ_FunctionDescriptionTags](#)

- (6) It is recommended that the project specific tag tables end with the word [Tags](#), but this is not compulsory.
- (7) The user can create as many project specific tag tables as required.

Note: Tag tables must not contain duplicate symbolic names or attempt to use an absolute address that has already been allocated in some other tag table.

4.11 Control system network device naming

- (1) All devices that are in some way connected to a Controller via an Ethernet or Profinet network must be named.
- (2) The naming of an Engineering Station was discussed in § 3.1.3; and similar procedures should be followed for naming other PC based systems that may be connected to the Controller (supervisory systems for example).
- (3) In addition, all Controllers and any remote IO installations must also be named. The following table summarises the general abbreviations for naming devices (this is an expanded version of Table 3.4):

ABBREVIATION	DEVICE
CONnnn	A Controller
ESnnn	Engineering Station, the PC that runs the full development software (in this case TIA Portal)
ESWnnn	Ethernet network switch or router
HMIInnn	HMI a panel mounted computer-based system similar to a SCADA system, but with restricted in capabilities.
OSnnn	Operator Station, a supervisory system (SCADA). If the system is a server/client arrangement, OS refers to a client
PNnnn	Profinet node, usually a remote IO rack, or Profinet enabled device
PNmmm_nnn	Profinet node on a separate subnet
PSWnnn	Profinet network switch or router
PSWmmm_nnn	Profinet network switch or router on a separate subnet
SVnnn	A server, usually a supervisory system server

Table 4.17 Device naming abbreviations

Where nnn is the last octet of the IP address
mmm is the second to last octet of the IP address

- (4) An example (fully expanded) network arrangement for the test rig (expanded to include a supervisory system with clients and an Engineering Station) is shown below:

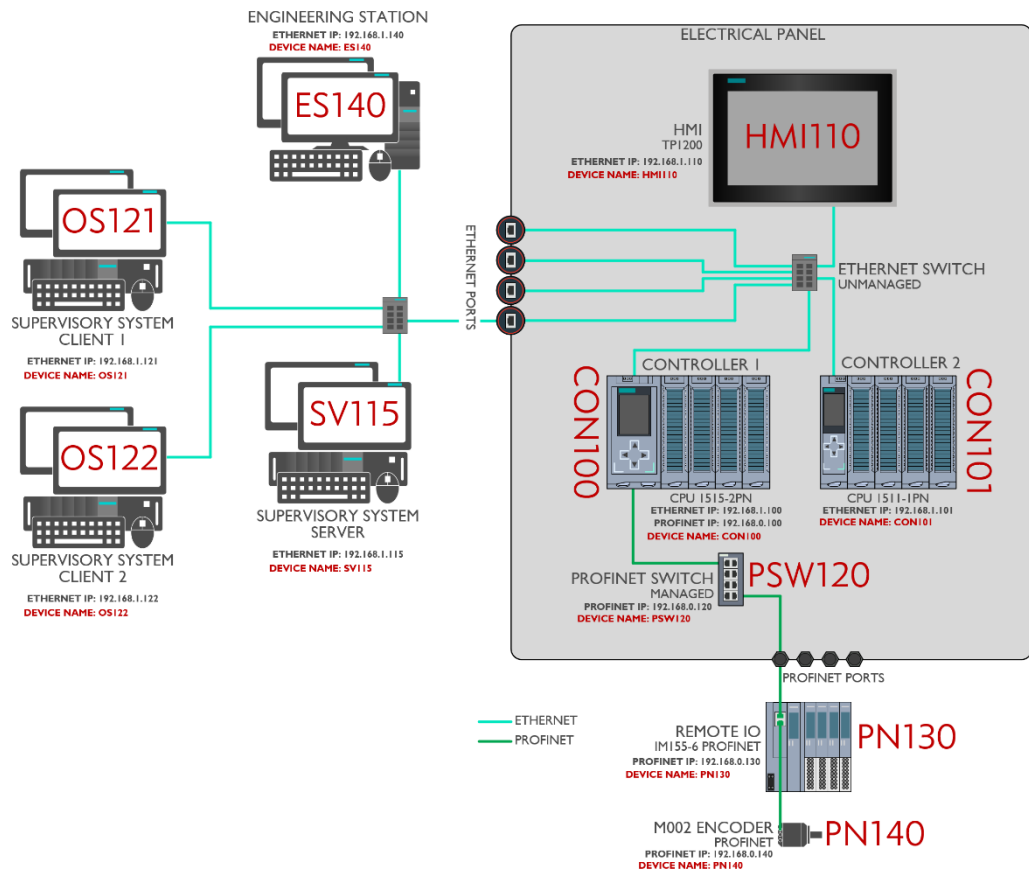


Figure 4.9 Expanded network with device names

- (5) In Figure 4.6 Figure 4.9, all network addressable devices (those devices with an IP address) have been named (shown in red).
- (6) If more than one Profinet network were in use, then these would have different subnets, the Profinet network shown here has subnet **192.168.0.nnn**, a second Profinet network would have a different subnet, e.g. **192.168.101.nnn**. Devices on the first subnet would be labelled (for example) **PN000_130** or **PSW000_120** &c. Device on the second subnet would have addresses with 101 as the first set of digits (for example **PSW101_120** &c.).
- (7) Where only one Profinet network is present, it is not necessary to use the dual numbering system.
- (8) It is not necessary to distinguish between the Ethernet subnet and the Profibus subnet, there is no duplication of unique device name between these two networks.

5

Common appearance and version control

- (1) All the blocks in the PAL have a common style and appearance. This is to give the blocks a similar look and feel and ensures that the same type of information is available within each block and that this information is located with a degree of commonality that makes finding and interpreting the block easy and predictable.
- (2) This section looks at common elements of programmable blocks (FC, FB and OB):
 - ① Definition of the title line for a block
 - ② A standard PAL block header
 - ③ Common networks and how they are used
 - ④ The block description network for a typical block
 - ⑤ A current revision and modification history template for a typical block
 - ⑥ Special examples of the above for the main execution block (OB 1)
- (3) The Style Guide (SG) [Ref. 010] provides additional detail on the topics covered here.

5.1 TIA Portal comment fields

- (1) TIA Portal supports comment fields within blocks and within individual networks within a block.
- (2) These comment fields are not particularly sophisticated and cannot be customised; they use a fixed font at non-adjustable point size. The font in question is [Siemens TIA Portal Basic](#) and it uses a fixed-point size of 9 pt that cannot be changed.
- (3) The font is a *proportional* sans-serif font; it is shown Figure 5.1:



Figure 5.1 Siemens TIA Portal basic font

- (4) There are two restrictions with the use of this font, the first is the fixed-point size is quite small (and it cannot be made bigger), the second is the fact that it is a proportional font.
- (5) The older programming package, *Simatic Manager* (the forerunner to TIA Portal), had a similar comment arrangement; however, it used a *non-proportional*⁵ font (Lucida console):

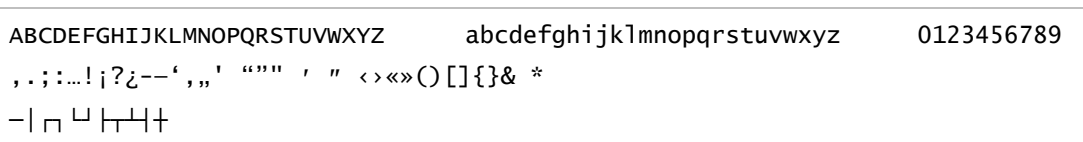


Figure 5.2 Lucida Console font

⁵ Proportional fonts have letters that are different widths, the M character is wider than the I character. With non-proportional fonts all characters are the same width (like text written on a typewriter).

- (6) The advantage of the non-proportional font was that it was easy to make lists (e.g. lists of parameters) that all lined up correctly.
- (7) The proportional font can make TIA Portal comments look untidy:

FUNCTION	MEMORY BIT	SYMBOL
Always Off	M 000.0	_FALSE
Always On	M 000.1	_True
50ms Pulse	M 000.2	_50ms
100ms Pulse	M 000.3	_100ms
200ms Pulse	M 000.4	_200ms
500ms Pulse	M 000.5	_500ms
1.0s Pulse	M 000.6	_1s
2.0s Pulse	M 000.7	_2s
Scan tick signal	M 001.0	_ScanTick
Scan tock signal	M 001.1	_ScanTock
First scan active	M 001.2	_ScanFirst
100ms Square Wave	M 001.3	_100ms_SqW
200ms Square Wave	M 001.4	_200ms_SqW
500ms Square Wave	M 001.5	_500ms_SqW
1.0s Square Wave	M 001.6	_1s_SqW
2.0s Square Wave	M 001.7	_2s_SqW

Figure 5.3 Lining columns using spaces (without tabs)

- (8) In Figure 5.3 the columns are aligned using spaces, the discrepancies are most clearly seen with the Ms and underscores, they do not correctly line-up (the orange lines are vertical and show the misalignment).
- (9) It is possible to accurately align these columns within TIA Portal comments (even with the proportional font in question). The [Siemens TIA Portal Basic](#) font characters all have varying widths; however, they are all either a whole multiple of the normal space character or a multiple of the normal space character plus $\frac{1}{3}$ of a normal space, or a multiple of the normal space character plus $\frac{2}{3}$ of a normal space.
- (10) Additional whitespace characters are available to the TIA Portal font that accommodate the $\frac{1}{3}$ and $\frac{2}{3}$ of a normal space: a [six-per-em](#) space that has a width of $\frac{2}{3}$ of a normal space and a [three-per-em](#) space that has a width of $1\frac{1}{3}$ of a normal space.
- (11) The consequence of this is that with judicious use of the [normal space](#), [three-per-em](#) space and the [six-per-em](#) space it is possible to always get a perfect fit.

- (12) Here is the same column arrangement aligned correctly with the additional whitespace characters (again the orange lines are vertical to show the alignment):

FUNCTION	MEMORY BIT	SYMBOL
Always Off	M000.0	_FALSE
Always On	M000.1	_True
50ms Pulse	M000.2	_50ms
100ms Pulse	M000.3	_100ms
200ms Pulse	M000.4	_200ms
500ms Pulse	M000.5	_500ms
1.0s Pulse	M000.6	_1s
2.0s Pulse	M000.7	_2s
Scan tick signal	M001.0	_ScanTick
Scan tock signal	M001.1	_ScanTock
First scan active	M001.2	_ScanFirst
100ms Square Wave	M001.3	_100ms_SqW
200ms Square Wave	M001.4	_200ms_SqW
500ms Square Wave	M001.5	_500ms_SqW
1.0s Square Wave	M001.6	_1s_SqW
2.0s Square Wave	M001.7	_2s_SqW

Figure 5.4 Lining columns using additional whitespaces characters

5.1.1 Maximum size of a comment field

- (1) A comment field for any network can hold approximately 65000 characters including spaces and line break characters.
- (2) This equates to approximately 670 lines of text, and with (a typical) 19.2 words per line, this is in the region of 13,000 words per comment field.
- (3) Where these restrictions provide insufficient space for the comments, it is permissible to add an empty network and continue the comments in the comment field for that network.

5.2 Common headers and networks

- (1) Each programmable block within the PAL (FB, FC or OB) has several common areas at the start of the block:
 - ① **Block title and comment field**
This holds a plain English title for the block and a generic copyright message along with links to the PAL documentation
 - ② **Network 1 — Detailed description of the block**
This contains a full and detailed functional description of the block including its parameters and data structures
 - ③ **Network 2 — Current revision and modification history**
This holds a full list of modification made to the block along with the revision number, data and the author of the revision
- (2) These networks can be considered a standard requirement of all blocks within the PAL; all blocks must contain these elements and be in accordance with the requirements discussed in the following sections.
- (3) Taking each of these elements in turn:

5.2.1 Block title and comment field

- (1) The first item in any programmable block (FB, FC or OB) is the **block title** field with a comment area underneath, highlighted in Figure 5.5:

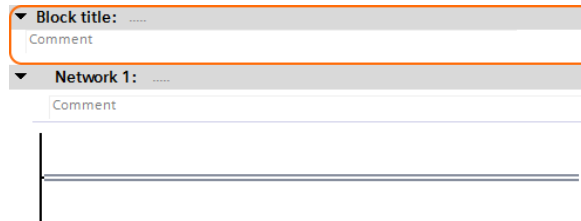


Figure 5.5 Block title

- (2) The **block title** line contains the block title in the form

Class Module [FUNCTION] – Description

- (3) For example, the block title for the isolating valve module (FC11001) is:

Block title: Standard Module [DEVICE DRIVER] — Isolating Valve

- (4) Generally, the block title should be no more than 80 characters (including spaces) and should be in title case (i.e. all principle⁶ words are capitalised), with the function field in capitals.
- (5) The block title comment field contains a standard copyright message for the PAL, details of how to access the specific User Documentation for the block and a link to the full PAL documentation It is shown below:

PRACTICAL SERIES AUTOMATION LIBRARY (PAL) — COPYRIGHT 2020 – M. GLEDHILL (MIT LICENCE)
FULL ONLINE DOCUMENTATION IS AVAILABLE AT: <https://practicalseries.com/2001-pal/index.html>

- (6) The block title comment field is common to all blocks that are issued with the PAL.

⁶ Principal word are words that are not:

- Articles (a, an, the)
- Conjunctions (and, or, but &c.)
- Prepositions (in, with, on &c.)

(7) The following is a typical example of a block title and comment field

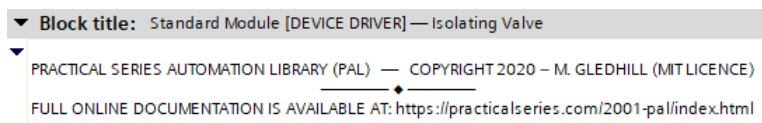


Figure 5.6 Example block title and comment field

5.2.2 Network 1 — Block description

- (1) The first network in the block (always numbered Network 1) immediately follows the block title and comment field. The network itself is not used; it is just an empty network. Its importance is purely as a comment field containing a description of the block.
- (2) Network 1 is always given the title **Block description**. The main function of Network 1 is to hold a full description of what the block does, how it does it and the parameters and data structures used within the block. The block description should contain the following sections:

Title

Overview

- ① Block technical summary
- ② Functional description
- ③ Detailed block description
- ④ Supervisory system interface
- ⑤ Parameters
- ⑥ Data structures and usage [and instance DBs]
- ⑦ Constants and Temporary (local) data
- ⑧ Block calls and associations
- ⑨ Example usage
- ⑩ Test and verification path

- (3) Each of these areas has its own section in the block description.
- (4) The typographical styles used to specify titles, headings, subheadings and area dividers are discussed further in the *Style Guide (SG) [Ref. 010]*.
- (5) Network 1 itself, is simply an empty ladder network:

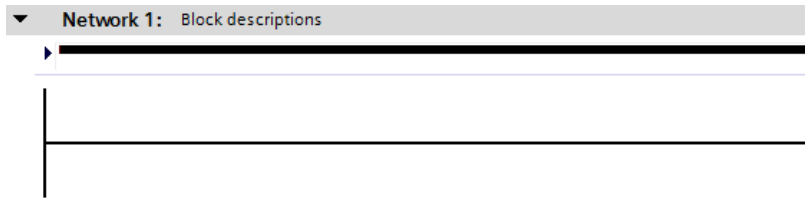


Figure 5.7 Block description with an empty network

- (6) The contents of the block description are based upon the software module design specification (SMDS) [Ref. 008] for the module in question.
- (7) The following is an abbreviated extract from an example block description:

Network 1: Block description — Sections 1 to 10 **STANDARD**

TITLE: SUBROUTINE — EVENT TIMER RTC

FC18151 — SOFTWARE MODULE DESIGN SPECIFICATION (SMDS)

The PDF version of the SMDS for this module is available online here:
<https://ips.op.uk/ips-ids-fc18151-p>

It is also available as part of the Project User Defined Documentation, it can be accessed by selecting this module in the Project Tree (on the left) and pressing **SHIFT+F1**

FC18151_SubTimeEventRTC is a subroutine block that times the duration of an event to nanosecond resolution.

The block records the time the event started (rising edge of the TRIGGER signal), the time the event ended (falling edge of the TRIGGER signal) 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.

This is a subroutine block.

1. BLOCK TECHNICAL SUMMARY

NOMENCLATURE & ADDRESSING

BLOCK TITLE	Event Timer RTC
BLOCK FUNCTION GROUP	Subroutine
BLOCK ADDRESS	FC18151
BLOCK SYMBOL	FC18151_SubTimeEventRTC
BLOCK USER ID	SubTimeEventRTC

BLOCK TYPE & USAGE

BLOCK TYPE	Function (FC)
BLOCK NUMBER	18151
LANGUAGE	LAD
OPTIMISED ACCESS	Yes
PAL USAGE TYPE	Standard block

METRICS

EXECUTION TIME	18.4 µs ¹
LOAD MEMORY	20.1 kB
WORK MEMORY	0.4 kB

¹ Measured on: CPU-1515-2PN (order no. 6ES7512-2AM02-0AB0)
<https://ips.op.uk/ips-ids-cpu1515>

3. DETAILED BLOCK DESCRIPTION

The purpose of this block is to determine with a high degree of accuracy the duration of an event. The event itself must be in the form of a Boolean trigger signal; the rising edge of the signal is considered to be the start of the event and the falling edge the end of the event.

The event signal is passed to the block as the TRIGGER parameter.
The block returns five values; these are stored in a UDT passed to the block in the DYNAMIC_DATA parameter:

PARAM	ASSOCIATED UDT	ASSOCIATED DATA BLOCK
DYNAMIC_DATA	UT38151_Dy_SubTimeEventRTC	DB38151_Dy_SubTimeEventRTC

The block returns the following data:

SIGNAL NAME	FUNCTION	TYPE
status_Running	STATUS — The event timer is running	Bool
actual_StartTime	ACTUAL — Time at which the TRIGGER signal went high (event start time)	DTL
actual_EndTime	ACTUAL — Time at which the TRIGGER signal went low (event end time)	DTL
actual_Duration	ACTUAL — Event duration	LTime
actual_DurationSec	ACTUAL — Event duration in seconds	REAL

Block returned signals

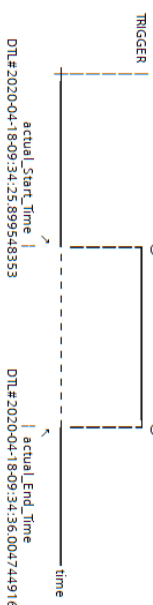
3.1 Timing an event

The block constantly monitors the TRIGGER parameter; if a rising edge is seen on the TRIGGER parameter (point ① in the figure below), the Controller RTC is read and its value is stored in actual_StartTime, at the same time the running signal (status_Running) is activated.

The block continues to monitor the TRIGGER parameter; when a falling edge is detected (point ②), the RTC is read once more and its value is stored in actual_EndTime, and the running signal (status_Running) is deactivated.

Once the end time is stored, the following calculation is carried out:

$$\text{actual_Duration} = \text{actual_EndTime} - \text{actual_StartTime}$$



1.1 Additional document references

In addition to the documents referenced in OB1 (section DOCUMENTREFERENCES), the following documents are associated with this module:

REF	DOCUMENTNO.	TITLE
101	PS2001-5-2331-001-01	FC19512 Calculation Sheet
102	PS2001-5-2331-001-02	FC19512 2nd Order Response Simulation Spreadsheet

[Alternate text, use if there are no additional references]
Only those documents specified in OB1 (section DOCUMENTREFERENCES) are listed within this block. There are no additional document references.

2. FUNCTIONAL DESCRIPTION

FC18151 is the Subroutine Event Timer using RTC block; it simply records the duration of an event with a high degree of accuracy. The event being timed is a Boolean signal passed to the block as the TRIGGER parameter.

Three times are recorded for the event: the start time (rising edge of the TRIGGER parameter), the end time (falling edge of the TRIGGER parameter) and the duration of the event (end time minus the start time). The duration is calculated on the falling edge of the TRIGGER parameter, immediately after the end time is recorded.

The start and end times are read from the Controller's internal real time clock (RTC) and are stored in DTL (date time long) format; this format contains year, month, day, hour, minutes, seconds, milliseconds, microseconds and nano seconds.

The duration is stored in LTime (long time, 64-bit) format and is very accurate; it resolves to days, hours, minutes, seconds, milliseconds, microseconds and nanoseconds; it keeps this resolution irrespective of the duration.

The duration is also calculated as a real number in seconds (or fractions thereof); this is a more convenient format for holding the duration but is not as accurate as the LTime format.



4. Supervisory system interface

See the online documentation for this block — select the block in the project tree and press: SHIFT+F1 or follow the link: <https://ps.op.uk/pal/smds-kt18151-p>

An explanation of the block icons and faceplates available to the supervisory system for this module is given in section 4 of the SMD5 document [Ref. 011].

A downloadable copy of the SMD5 (and all other project documentation) is available here:

<https://practicalseries.com/2001-pal/21-project01-00-docs.html>

For this module, the document number is: PS2001-5-231-kt.1815

[Alternate text]

This module does not interface with a supervisory system. — ◆◆◆◆◆◆◆◆

5. PARAMETERS

The following parameters are associated with the block:

PARAM	FUNCTION	TYPE	IN-OUT
SYS_SIGNALS	Common system signals logic and timing signals for parametric access	UTZ1000	In
TRIGGER	Event trigger signal (a rising edge marks the start of the event, the falling edge the end)	Bool	In
DYNAMIC_DATA	Stores the results of the event timing	UTS8151	In/Out

— ◆◆◆◆◆◆◆◆

6. DATA STRUCTURES AND USAGE

The block has the following associated data structure:

DATA STRUCTURE	DESCRIPTION
UTD1000_S1_SysRevision	Revision information for this block
UTD1001_L5C_SysLicence	Licence information for this block
UTZ1000_Dy_SysSignals	System signals for logic and timing
UTS8151_Dy_SubTimeEventRTC	Dynamic data structure for the event timing block

6.1 UTS8151_Dy_TimeEventRTC

Stores the timing values for the event:

UTS8151_Dy_TimeEventRTC

SIGNAL NAME	FUNCTION	TYPE
status_Running	STATUS — The event timer is running	Bool
actual_StartTime	ACTUAL — Time at which the TRIGGER signal went high (event start time)	DTL
actual_EndTime	ACTUAL — Time at which the TRIGGER signal went low (event end time)	DTL
actual_Duration	ACTUAL — Event duration	LTime
actual_DurationSec	ACTUAL — Event duration in seconds	REAL

— ◆◆◆◆◆◆◆◆

7. CONSTANTS AND TEMPORARY (LOCAL) DATA

7.1 Constants

The following constants data is used:

CONSTANTS DATA

SIGNAL NAME	Value	FUNCTION	TYPE
k_Pi	3.141592	Value of Pi (π)	Real
k_2Pi	6.283185	Value of 2 X Pi (2π)	Real

7.2 Temporary (local) data

The following temporary data is used:

SIGNAL NAME	FUNCTION	TYPE
revInfo	Revision information for the block	UTD1000
licInfo	Licensing information for the block	UTD1001
workInt	Working storage area (integer)	Int
workReal	Working storage area (Real)	Real
workLint	Working storage area (long integer)	Lint

— ◆◆◆◆◆◆◆◆

8. BLOCK CALLS AND ASSOCIATIONS

This section details any blocks which may be called from within this software module (subroutine functions; for example), any partner blocks with which it may be associated (for example a receive module that is partner with a transmit module &c.). It lists any system functions which may be called (e.g. reading the real time clock) and any system data types that may be used.

Finally, it lists any special calling requirements for the block (for example, must be called from within a cyclic interrupt organisation block) and if the block is using "optimised access" (this is the default arrangement).

8.1 Block calls from within this module

There are no PAL block calls from within this block.

8.2 Blocks associated with this module

This is a stand-alone block and is not associated with any other blocks.

8.3 System block calls and system data types

The following calls are made to system blocks:

System block calls	TITLE	DESCRIPTION
BLOCK		Read the local RTC time
RD_LOCAL_T		Reads the CPU real time clock

There are no special system data types used.

8.4 Special properties and requirements

8.4.1 Block optimisation, IEC compatibility and library conformance

- Block optimisation is ACTIVE for this block.
- The block has been checked for IEC compatibility and is compliant.
- The block is compatible with all IEC library-conformance module constraints.

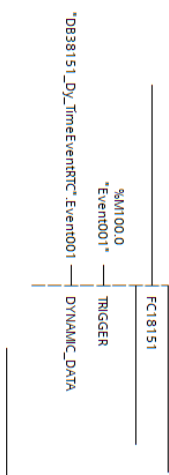
8.4.2 Calling requirements

This block is a documentation block and as such is a reference block that should not be called. The block itself does not contain any executable code, with the exception of the revision and licence networks, all networks are empty. The block itself does nothing, and its execution will have no effect, other than extending the cycle time.



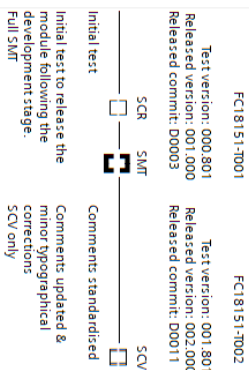
9. EXAMPLE USAGE

The following shows a typical deployment of FC18151:



10. TEST AND VERIFICATION PATH

The following diagram shows the full test and verification path for the formal release of each version of the software module. See the Test Plan [Ref: 003] for full details, available here: <https://psop.uk/pa/pspp>



Where:

SCR Source Code Review
A visual inspection of the software to ensure it has been written to the correct standard, uses the correct structures within the software and is generally suitable for deployment.

SMT Software Module Test
A full and detailed test of an individual software module in isolation, such testing requires that all branches of the software are tested.

SCV Software Compatibility
It tests all the interfaces to the module, any data recorded or stored by the module, all error and exception handling functions and tests all timed and interrupt driven operations.

Verification
A mechanism for verifying that no executable code software changes have been made to a module following changes to comment fields.

Allows typographical errors &c. to be corrected without forcing a full SMT on a module that has not functionally changed.

An SCV can only be performed on a module that has at some point previously, successfully completed both an SCR and SMT. Where an SCV is implemented, it replaces the requirement for both the SCR and the SMT for the re-iteration in question.



END

5.2.3 Network 2 — Current revision and modification history

- (1) Network 2 contains the current revision of the software module. This network is not empty; it contains the current revision number of the block, the revision date and the author's initials. These are hard coded in the network.
- (2) Network 2 always has the title: **Current revision and modification history**. The network comment field contains the modification history of the block (up to and including the current revision).
- (3) The comment network has the following appearance:

Network 2: Current revision and modification history

MODIFICATION HISTORY

This is a complete summary of all software modifications made to this block. The current revision is at the top of the list.

The current revision, author and the date of the revision are hardcoded into the Controller in this network (allowing revision data to be obtained directly from the Controller).

The revision data shown here, in the network comments, contains additional information reflecting the software development workflow, under the version control system (VCS).

The VCS in use is the GIT Source Code Management system in conjunction with the GitHub online hosting system. The software in its entirety is available in the GitHub remote repository:

<https://github.com/mgledhill/PS2001-pal-software>

This repository is public and can be freely cloned and used under the MIT Licence. The MIT Licence is reproduced in full in the last network of this software module.

— FC01001	REVISION (nnn.amm)	COMMIT TAG	MASTER BRANCH (Base → Merge)	DEV BRANCH	AUTHOR
2020.11.09	001.000	D0003	D0002 D0003 FC01001 — RELEASED FOR USE Merge back to master branch	None	M. Gledhill
2020.11.09	001.000	D0002A-001.000	D0002 N/A FC01001 — Post test RELEASED FOR USE	D0002A	M. Gledhill
2020.11.08	000.801	D0002A-000.801	D0002 N/A FC01001 — Released for (SMT) Software module testing	D0002A	M. Gledhill
2020.11.04	000.102	D0002A-000.102	D0002 N/A FC01001 — Incremental build Software based on tested pre VCS version	D0002A	M. Gledhill
2020.11.03	000.101	D0002A-000.101	D0002 N/A FC01001 — Block created	D0002A	M. Gledhill

Where: nnn = Major revision
a = Type (1-7 Development, 8 Proving, 9 Qualification, 0 Release)
mm = Minor revision (must be 00 if a = 0)
Block properties version number should be set to nnn.a (only one decimal place)

#SYS_SIGNALS_False

Figure 5.8 Network 2 — Current revision and modification history

- (4) All blocks within the PAL have the block revision number stored in Network 2. Each field of the revision data is stored as a string, the four revision fields are:
- ① Block number (e.g. FC02001)
 - ② Revision number of the block
 - ③ Revision date in the format `YYYY-MM-DD`
 - ④ Author of the revision, initials and surname
- (5) The revision numbering mechanism is detailed in the Software Control Mechanism (SCM) *[Ref. 018]*.

5.3 OB 1 header and revision network

- (1) OB 1 (the main organisation block) is considered a special organisation block in terms of the Practical Series Automation Library (and in terms of most Siemens Controller software). It is the block that executes all the rest of the controller software.
- (2) As such it contains information about the whole project rather than just a software module. The revision data is also project specific (not module specific).

5.3.1 OB 1 Network 1 — Project description

- (1) The OB 1 block description contains a summary of the project, rather than of a particular module, as well as copyright, licence, file and project details &c. as follows:
 - Copyright details
 - Customer details
 - Project name
 - Project number
 - Controller type(e.g. CPU-1515-2PN)
 - Controller name(The CPU name assigned in TIA Portal)
 - IP (Ethernet) address
 - TIA project name
 - Software status(e.g. Development, Release &c.)
 - Project overview(a summary of the project and its purpose)
 - Document references
 - Completed modules (list of)
 - Licence(details of any software licence)

Network 1: Project description

TITLE: PS2001 — PRACTICAL SERIES AUTOMATION LIBRARY

COPYRIGHT: © 2020 Michael Gleghill
Part of the Practical Series of Publications
Published in the United Kingdom
mg@practicalseries.com
<https://practicalseries.com>

CUSTOMER: Practical Series of Publications (PSP)

PROJECT: Practical Series Automation Library (PAL)

PROJECT NO.: PS2001

CONTROLLER: CPU 1515-2PNUD

CONTROLLER NAME: CONT100

IP ADDRESS: 192.168.001.100

TIA PROJECT NAME: PS2001-PAL-D0014

STATUS: DEVELOPMENT

PROTECTION: To minimise the risk of inadvertent modification to tested modules, certain blocks will be released for use with 'protected access' (referred to as 'write protection' in Siemens terminology), this allows the block to be used normally, but prevents the block being accidentally modified.

This is in accordance with the risk assessment given in the Validation Plan (VP), Appendix A [Ref: 002].

THE WRITE PROTECTION PASSWORD IS: PS2001

PROJECT OVERVIEW

The PAL is a library of software modules and templates that have been developed for the Siemens Simatic 57-1500 range of controllers (and to a lesser extent the 57-1200 range).

The full library and all necessary documentation is available from the Practical Series website:

<https://practicalseries.com/2001-pal/index.html>

The PAL is configured and deployed using the Siemens Simatic TIA Portal programming environment (version 16 or higher).

The PAL software structure is designed such that it is applicable to virtually all industrial applications that can generally be controlled by a programmable logic controller.

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
- Food and beverage production

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.

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 etc.

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

The Practical Series Automation Library is freely available under the MIT Open Source licence (see below). Those who find it useful may, if they wish, make a donation to support the library.

Donations can be made here:

<https://practicalseries.com/2001-pal/11-web/81-00-pay.html>

The PAL contains fully deployable software that has been developed by the author in his profession as a Chartered Electrical Engineer. It is currently in use on various live plants throughout the UK and in some other parts of the world.

This software is suitable for controlling and automating most industrial applications (typical process applications), it is easy to use and configure, but does have a degree of practical complexity appropriate for the environments within which it is employed. It is heavily configurable, has various operating modes and is suitable for a multitude of industrial applications.

IF YOU DON'T UNDERSTAND IT, DON'T USE IT. IF YOU DO USE IT, YOU DO SO AT YOUR OWN RISK.



DOCUMENT REFERENCES

REF.	DOCUMENT NO.	TITLE
001	PS2001-5-0101-001	Quality Plan (QP)
002	PS2001-5-0121-002	Validation Plan (VP)
003	PS2001-5-0131-003	Test Plan (TP)
004	PS2001-5-1101-001	User requirements specification (URS)
005	PS2001-5-1111-001	Requirement Traceability Matrix (RTM)
006	PS2001-5-2101-001	Functional Specification (FS)
007	PS2001-5-2101-001	Hardware Design Specification (HDS)
008	PS2001-5-2211-001	Software Design Specification (SDS)
009	PS2001-5-2311-001	Style Guide (SG)
010	PS2001-5-2341-01-001	ES/MDP Configuration Manual
011	PS2001-5-2312-46No.	Software Module Design Specification (SMD5)
012	PS2001-5-2301-001	Software Module Register (SMR)
013	PS2001-5-2302-011	Software Control and Mechanism (SCM)
014	PS2001-5-7111-001	User Guide (UG)

Note: Where a block references another document not listed above (a calculation sheet for example), that document will be referenced separately in section 1.1 of the block description for the block in question.

A full list of all project documentation is available here:

<https://practicalseries.com/2001-pal/21-project/01-00-docs.html>

COMPLETED MODULES

BLOCK	NAME	REVISION		STATIC DATA		DYNAMIC DATA	
		REVISION	UDTDB	REVISION	UDTDB	REVISION	UDTDB
FC01001	StdSysGlobalData	002.000	None	UT21000	002.000	UT21001	002.000
		2022-04-16		DB21001	002.000	DB21001	002.000
FC02001	StdInstAnalogRead	002.000	UT02001	002.000	UT22001	002.000	002.000
		2022-04-16	DB02001	002.000	DB22001	002.000	
FC11001	StdDevValveIcol	002.000	UT11001	002.000	UT31001	002.000	002.000
		2022-04-16	DB11001	002.000	DB31001	002.000	
FC18001	StdSubScaleAI	002.000	None	None	None	None	None
		2022-04-16	None	None	None	None	
FC18151	StdSubTimeEventRTC	002.000	None	None	UT38151	002.000	002.000
		2022-04-16	None	None	DB38151	002.000	
FC19512	StdDebugInst2Order	002.000	None	None	UT39512	002.000	002.000
		2022-04-16	None	None	DB39512	002.000	
FC61000	DocGenExample	002.000	None	None	None	None	None
		2022-04-16	None	None	None	None	

See Software Module Register [Ref. 012] for full details

LICENCE AND CONTACT INFORMATION

This software and its associated documentation is made available under the MIT Licence:

The MIT License (MIT)

Copyright © 2020 Michael Gledhill

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

You can reach the author by email here:

mg@practicalseries.com

Questions, corrections, constructive criticism and complaints (polite ones) are invited.



END

5.3.2 OB 1 Network 2 — Current revision and modification history

- (1) OB 1 Network 2 contains the current revision of the whole *software project* (rather than of a particular block). The whole project is issued at a particular release and this is independent of individual module revisions — most modules do not change their revision when the project as a whole is reissued at a new release, the new release may be to include new blocks or it may be to include a change or correction to a particular block (in which case the revision of the affected block would change, but not the others).
- (2) Like all other software blocks, Network 2 in OB 1 has the title: **Current revision and modification history**. The network comment field contains the modification history of the software project as a whole (up to and including the current revision).
- (3) OB 1 stores more revision information than is done with the general software block revision data (see § 5.2.3) and it stores it in a data block.
- (4) In all other blocks, the revision, the revision date and author's initials are hard coded as statements that simply transfer the data to a temporary area within the block.
- (5) In OB 1 the data is loaded in the same way, but this time it is transferred to the system data block (**DB21001_Dy_SysGlobalData**). This allows the data to be read elsewhere within the software or by an external device (such as a SCADA or HMI system). In short, OB 1 makes the current revision of the project available to anything that has access to the controller.
- (6) In the case of OB 1 revision data, the block number is replaced with the project number.
- (7) Figure 5.9 contains an example of the OB 1 Current revision and modification history network and comments:

Network 2: Project revision and modification history

MODIFICATION HISTORY

The revision data herein contains the current revision of the whole software project (rather than of a particular block). The whole project is issued at a particular release and this is independent of individual module revisions.

The revision data shown here, in the network comments, contains additional information reflecting the software development workflow under the version control system (VCS) employed to track all software changes.

The VCS in use is the Git Source Code Management system in conjunction with the GitHub online hosting system. The software in its entirety is available in the GitHub remote repository:

<https://github.com/practicalseries/PS2001-pal-software>

The repository is public and can be freely copied (forked in GitHub terminology) and used under the MIT licence.

The MIT licence is reproduced in full in the last network of this software module.

DATE	COMMIT TAG	AUTHOR	REASON FOR MODIFICATION
2022.03.20	D0010	M. Gledhill	FC19512 — RELEASED FOR USE
2021.08.23	D0009	M. Gledhill	FC11001 — RELEASED FOR USE
2021.05.26	D0008	M. Gledhill	FC02001 — RELEASED FOR USE
2021.05.22	D0007	M. Gledhill	FC18001 — RELEASED FOR USE
2021.05.11	D0006	M. Gledhill	FC01001 — RELEASED FOR USE
2021.05.02	D0005	M. Gledhill	Typographical corrections only
2021.05.02	D0004	M. Gledhill	Typographical corrections only
2021.02.19	D0003	M. Gledhill	FC18151 — RELEASED FOR USE
2021.02.18	D0002	M. Gledhill	Baseline build
2021.02.18	D0001	M. Gledhill	Hardware build
2021.02.18	D0000	M. Gledhill	Initial commit — repository created

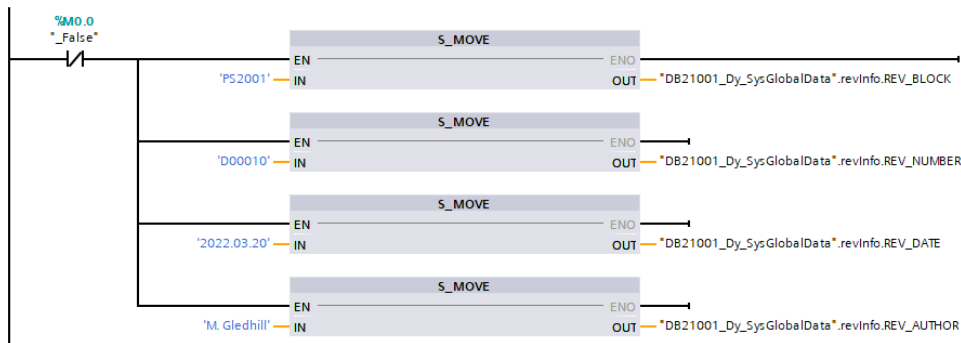


Figure 5.9 Network 2 — OB 1 Current revision and modification history

5.4 General network comments

- (1) All coded networks within the PAL should be given a title and be commented (all networks have a title and comment field). The following shows a typical example:

▼ **Network 13:** Load the system RTC and store in local variable

The system block RD_SYS_T reads the time of day, the date and the day of week from the system real time clock as a DTL (or Date_And_Time_Long) data type (one just being the shorthand notation of the other). The result is stored in a local variable for decoding.

The DTL data type stores the date and time information in 8 consecutive bytes in BCD format as follows:

DATA	DESCRIPTION
0	Year (00h-89h = 2000 to 2089) (90h-99h = 1990 to 1999)
1	Month (00h-12h = Jan to Dec)
2	Day (01h-31h)
3	Hour (00h-23h)
4	Minute (00h-59h)
5	Month (00h-12h = Jan to Dec)
6-7	Millisecond (Uses byte 6 and most significant 4 bits of byte 7)
7	Day of Week (least significant 4 bits (1h-7h = Sun to Sat))

The local variable is of type DTL (Date_And_Time_Long), this is compatible with the DT output from RD_SYST, but allows the individual elements of the DT structure to be accessed individually as symbols.

Note: The DTL format can store more information (it can for example handle years in the range 1970 to 2262; however, it will still only store the data in the table above when used with RS_SYS_T.

Figure 5.10 A typical network with comments

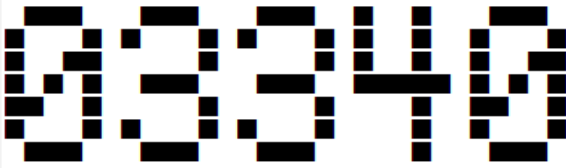
- (2) All networks must have a title (even if they are a continuation of a previous network). The title should be written in sentence case, uppercase and title case are permissible where the author requires emphasis but generally, use sentence case.
- (3) The network title should summarise the function of the network and it should be fairly short, keep it to less than 60 characters (not including spaces).
- (4) The Style Guide (SG) [Ref. 010] give full details for constructing network comments.

5.5 Specific network comments for sequences

- (1) Sequences have a particular arrangement of comments that differs somewhat from the normal network comments of non-sequential blocks.
- (2) Sequences are divided into discrete steps and each step has a unique (five digit) number in the range 00000 to 65000. Each step occupies several networks, each network having one of the following functions:
 - **DECLARATION** — (single network) manages the step, its operation and its timers
 - **ACTIONS** — (multiple networks) split into **INITIALISING**, **PROCESSING** and **TERMINATING** phases, each phase can carry out a specific or multiple actions
 - **TRANSITIONS** — (multiple networks) determines the conditions needed to leave the current step and proceed to another. Each step can have up to eight transition conditions

5.5.1 Step declaration network — title and comments

- (1) The declaration network contains a detailed description of the step: its function, phase operations and transition conditions. An example is shown in Figure 5.11.
- (2) The network title consists of two full block (■) characters (Unicode 2588h) followed by the text `STEP XXXXX – DECLARATION (STATE)` where `XXXXX` is the five-digit step number. The text in brackets reflects the state logic of the step (`STARTING`, `RUNNING`, `STOPPING`, `ABORTING` &c.).
- (3) The full block characters at the start make the start of each step identifiable if the networks are collapsed in TIA Portal.
- (4) The most notable feature of the network comment is the large five-digit step number at the top. This form of step number is used to make the first (declaration) network of each step easily identifiable.
- (5) The large number is preceded by a line of 44 `top half block` characters (■), Unicode character 2580h, followed by a blank line. The number is followed by a blank line and then 44 `bottom half block` characters (■), Unicode character 2584h.
- (6) The number is produced in the Excel spread sheet: Dot Matrix Generator (DMG) [Ref. 018], select the `DOT MATRIX GENERATOR` work sheet and type the required number in cell A7, copy the result from cells D8:D14 and paste it into the network comment field (the spread sheet itself contains full instructions).
- (7) The spread sheet can also convert uppercase letters to the same dot matrix form.
- (8) The Style Guide (SG) [Ref. 010] give full details for constructing sequence network comments.



STEP DESCRIPTION

Open the dry air purge valve (SV4201), activate the operator prompt ABORT PURGE.
 If the main chamber humidity level falls below the target value proceed to the next step.
 If the operator presses the ABORT PURGE button proceed to step 03400.

STEP PHASE AND DELAY CONDITIONS

PHASE	ACTION
INITIALISING	Activate the ABORT PURGE enable signal
PROCESSING	Open SV4201
TERMINATING	Clear the ABORT PURGE enable signal and any ABORT PURGE pressed signal
STEP DELAY	No delay

STEP TRANSITION CONDITIONS

TRANSITION	TO STEP	TRANSITION CONDITIONS
1.	03350	HIC001 ≤ HUMIDITY_LIMIT
2.	03400	Operator has pressed ABORT PURGE button

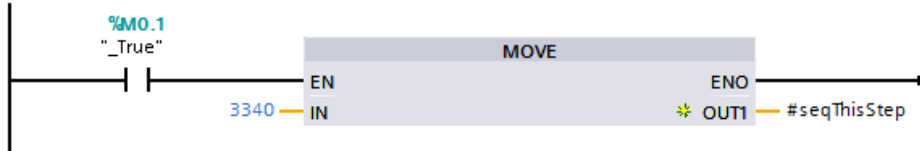


Figure 5.11 Sequence declaration comments

5.6 Data block header and revision

- (1) Data blocks do not have a similar facility to the comment field available to programmable blocks (FCs, FBs and OBs); however, each line within a data block has a comment line (just one line high) that can be adapted to hold a block description:

DB31001_Dy_DevValvlsol				
	Name	Data type	Start value	Comment
1	Static			
2	_DB_Header	Array[0..79] of Bool		Dynamic DB [DEVICE DRIVER] — Isolating Valve
3	_DB_Header[0]	Bool	false	STANDARD
4	_DB_Header[1]	Bool	false	
5	_DB_Header[2]	Bool	false	TITLE: DEVICE DRIVER — ISOLATING VALVE
6	_DB_Header[3]	Bool	false	
7	_DB_Header[4]	Bool	false	
8	_DB_Header[5]	Bool	false	TYPE: DYNAMIC
9	_DB_Header[6]	Bool	false	
10	_DB_Header[7]	Bool	false	
11	_DB_Header[8]	Bool	false	This is the dynamic data block associated with the standard device driver for an isolating valve block:
12	_DB_Header[9]	Bool	false	FC11001_StdDevValvlsol
13	_DB_Header[10]	Bool	false	
14	_DB_Header[11]	Bool	false	
15	_DB_Header[12]	Bool	false	The dynamic data holds the live data for the device (its mode, status and messages and any other dynamic information required by the module)
16	_DB_Header[13]	Bool	false	
17	_DB_Header[14]	Bool	false	
18	_DB_Header[15]	Bool	false	Each valve is given the user data type:
19	_DB_Header[16]	Bool	false	UT31001_Dy_DevValvlsol
20	_DB_Header[17]	Bool	false	This holds the full set of dynamic data for the device
21	_DB_Header[18]	Bool	false	
22	_DB_Header[19]	Bool	false	◆◆◆◆◆
23	_DB_Header[20]	Bool	false	
24	_DB_Header[21]	Bool	false	
25	_DB_Header[22]	Bool	false	MODIFICATION HISTORY
26	_DB_Header[23]	Bool	false	
27	_DB_Header[24]	Bool	false	This is a summary of the recent software modifications made to this block (most recent at top)
28	_DB_Header[25]	Bool	false	
29	_DB_Header[26]	Bool	false	
30	_DB_Header[27]	Bool	false	DATE REVISION MASTER BRANCH DEV
31	_DB_Header[28]	Bool	false	(nnn.amm) COMMITTAG (Base → Merge) BRANCH Author
32	_DB_Header[29]	Bool	false	
33	_DB_Header[40]	Bool	false	2021.08.23 001.000 D0009 D0008 D0009 None M. Gledhill
34	_DB_Header[41]	Bool	false	FC11001 — RELEASED FOR USE
35	_DB_Header[42]	Bool	false	Merge back to master branch
36	_DB_Header[43]	Bool	false	
37	_DB_Header[44]	Bool	false	2021.08.17 000.801 D0008A-000.801 D0008 N/A D0008A M. Gledhill
38	_DB_Header[45]	Bool	false	FC11001 Released for SMT
39	_DB_Header[46]	Bool	false	
40	_DB_Header[47]	Bool	false	2021.08.10 000.103 D0008A-000.103 D0008 N/A D0008A M. Gledhill
41	_DB_Header[48]	Bool	false	FC11001 Incremental build
42	_DB_Header[49]	Bool	false	
43	_DB_Header[50]	Bool	false	2021.07.21 000.102 D0008A-000.102 D0008 N/A D0008A M. Gledhill
44	_DB_Header[51]	Bool	false	FC11001 Incremental build
45	_DB_Header[52]	Bool	false	
46	_DB_Header[53]	Bool	false	2021.07.08 000.101 D0008A-000.101 D0008 N/A D0008A M. Gledhill
47	_DB_Header[54]	Bool	false	FC11001 Block created
48	_DB_Header[55]	Bool	false	
49	_DB_Header[56]	Bool	false	Where: nnn = Major revision
50	_DB_Header[57]	Bool	false	a = Type (1-7 Development, 8 Proving, 9 Qualification, 0 Release)
51	_DB_Header[58]	Bool	false	mm = Minor revision (must be 00 if a = 0)
52	_DB_Header[59]	Bool	false	Block properties version number should be set to nn.a (only one decimal place)
53	_DB_Header[60]	Bool	false	
54	_DB_Header[61]	Bool	false	END
55	_DB_Header[62]	Bool	false	

Figure 5.12 Comments within a DB

- (2) Within the PAL , the first entry within the data block is given to an array of 80 Boolean elements (10 bytes of data) called:

`_DB_Header`

- (3) This array does not contain any useful data (each entry in the array is set to zero and is never used by the PAL — in this respect, it is just wasted space at the start of the DB); it does however, provide 80 blank lines that can be used to hold a block title, block description and a (rudimentary) current revision and history area.
- (4) In Figure 5.12 the large blank area in `_DB_Header` (lines 31 to 81 in the leftmost column numbers) has been edited out to keep the image to a practical size.
- (5) The `_DB_Header` declaration itself (line 2) acts as the block title for the DB (similar to the block title for a block, see § 5.2.1).
- (6) The DB block title (line 2) should be in title case and be no more than 80 characters (not including spaces) long.
- (7) Beneath this is something analogous to the title of a block description network (see § 5.2.2).
- (8) The title entry is on the next line (line 5) and consists of the uppercase word `TITLE:` followed by the title itself. The title must be in all uppercase characters.
- (9) Line 8 holds the type of the data block; this is either `static`, `dynamic` or `recipe`.
- (10) This is followed by a brief description of the block and its function.
- (11) Finally, there is a modification history (similar to that of Network 2 for a programmable block, see § 5.2.3)

5.6.1 Data block revision information

- (1) The revision information for the data block is hardcoded into the variable `revInfo`, this is of type `UT01000_St_SysRevision` and contains the following information:
- ① Block number (e.g. DB02001)
 - ② Revision number of the block
 - ③ Revision date in the format `YYYY-MM-DD`
 - ④ Author of the revision, initials and surname
- (2) The revision numbering mechanism is detailed in the Software Control Mechanism (SCM) [Ref. 018].

5.6.2 UDT block revision information

- (1) All UDTs start with the same variables used in the data block for revision and licencing information:
- ① `revInfo` of type `UT01000_St_SysRevision`
- (2) Again, these contain hardcoded values for the revision and licence information. The UDT **do not** contain a comment area that holds the modification history.

5.7 Programmable block properties

- (1) All programmable blocks (FCs, FBs and OBs) have block property information fields.
- (2) The block properties are accessed by right clicking the block (either in the project tree or on the overview screen) and selecting **PROPERTIES** from the dropdown list, this opens the properties dialogue box (Figure 5.13).
- (3) The properties for the programmable blocks within the PAL are listed below along with the format and conventions that are applied (only those properties that can be changed by the user are listed):

BLOCK PROPERTIES FOR FC, FB AND OB

AREA	PROPERTY	FORMAT	EXAMPLE
General	Name	Block name	FCI1000_StdDevValvelsol
	Language	The language the block is written in	LAD
	Number	Block number (manually specified)	I1000
Information	Title	Block title in the form: Class Module [FUNCTION] — Description	Standard Module [DEVICE DRIVER] — Isolating Valve
	Comment	Copyright message from block header	See block header § 3.3.1
	Version	For a revision in the form nnn.amm, it shows nn.a	00.8
	Family	Block function group (without spaces)	DeviceDriver
	Author	The block author, initial and surname use underscore in place of spaces	M_Gledhill
	User-defined ID	The block ID	StdDevValvelsol
Attributes	IEC Check	Should be active for Standard Modules	Box is ticked
	Optimized block access	Should be active for all FC, FB and OB blocks	Box is ticked

Table 5.1 Block properties for FBs, FCs and OBs

Note: For all Standard Modules, the entry **LIBRARY CONFORMANCE** (in the **COMPILATION** area) must read: **The object is library-conformant**

(4) Worked examples of each of these are shown below:

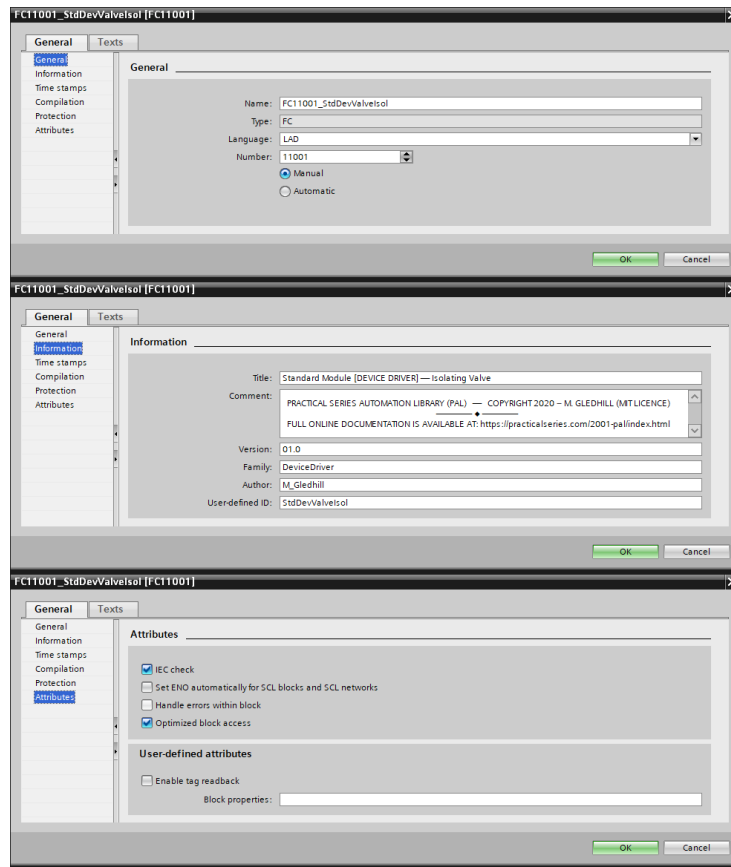


Figure 5.13 FC, FB and OB block properties

5.8 Data block and UDT properties

5.8.1 Data block properties (static and dynamic)

- (1) Data block properties are very similar to programmable block properties; the difference is a minor technicality in how some of the entries are made.
- (2) Data block properties are accessed in exactly the same way as those for programmable blocks: right click the block in either the project tree or on the overview screen.
- (3) The properties for the various types of data blocks within the PAL are listed below along with the format and conventions that are applied (only those properties that can be changed by the user are listed):

BLOCK PROPERTIES FOR STATIC DBS

AREA	PROPERTY	FORMAT	EXAMPLE
General	Name	Block name	DB11001_St_DevValvelsol
	Language	The language the block is written in	DB
	Number	Block number (manually specified)	11000
Information	Title	Block title in the form: Static DB [FUNCTION] — Description	Static DB [DEVICE DRIVER] — Isolating Valve
	Comment	Copyright message from block header	See block header
	Version	For a revision in the form nnn.amm, it shows nn.a	00.8
	Family	Block function group (without spaces)	DeviceDriver
	Author	The block author, initial and surname use underscore in place of spaces	M_Gledhill
	User-defined ID	The block ID	St_DevValvelsol
Attributes	Optimized block access	Should be active for Static DBs	Box is ticked

Table 5.2 Block properties for Static DBs

BLOCK PROPERTIES FOR DYNAMIC DBS

AREA	PROPERTY	FORMAT	EXAMPLE
General	Name	Block name	DB31001_Dy_DevValvelsol
	Language	The language the block is written in	DB
	Number	Block number (manually specified)	31000
Information	Title	Block title in the form: Dynamic DB [FUNCTION] — Description	Dynamic DB [DEVICE DRIVER] — Isolating Valve
	Comment	Copyright message from block header	See block header
	Version	For a revision in the form nnn.amm, it shows nn.a	00.8
	Family	Block function group (without spaces)	DeviceDriver
	Author	The block author, initial and surname use underscore in place of spaces	M_Gledhill
Attributes	User-defined ID	The block ID	Dy_DevValvelsol
	Optimized block access	Should be active for Static DBs	Box is ticked

Table 5.3 Block properties for Dynamic DBs

(5) Worked examples of each of these are shown below:

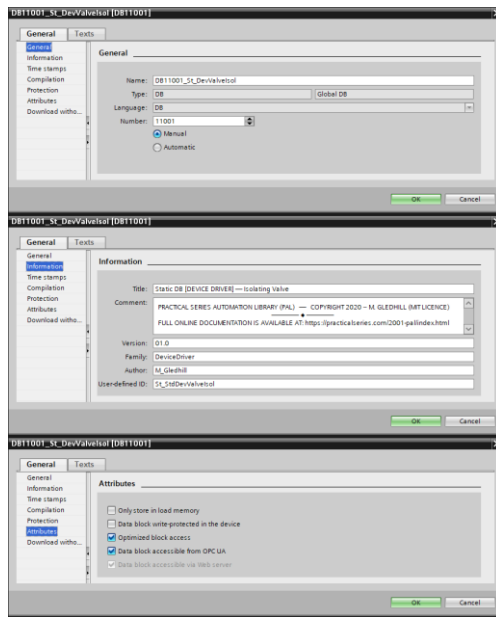


Figure 5.14 Static DB block properties

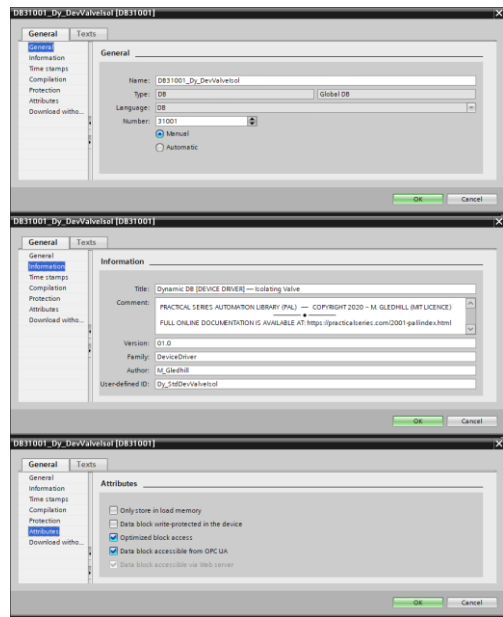


Figure 5.15 Dynamic DB block properties

5.8.2 UDT properties (static and dynamic)

- (1) UDT block properties have fewer entries than programmable and data blocks and have limited scope.
- (2) UDT properties are accessed in exactly the same way as those for programmable blocks and data block: right click the UDT in either the project tree or on the overview screen.
- (3) The properties for the various types of UDT within the PAL are listed below along with the format and conventions that are applied (only those properties that can be changed by the user are listed):

BLOCK PROPERTIES FOR STATIC UDTs

AREA	PROPERTY	FORMAT	EXAMPLE
General	Name	Block name	UT11001_St_DevValvlsol
Information	Title	Block title in the form: Static UDT [FUNCTION] — Description	Static UDT [DEVICE DRIVER] — Isolating Valve
	Comment	Copyright message from block header	See block header

Table 5.4 Block properties for Static UDTs

BLOCK PROPERTIES FOR DYNAMIC UDTs

AREA	PROPERTY	FORMAT	EXAMPLE
General	Name	Block name	UT31001_Dy_DevValvlsol
Information	Title	Block title in the form: Static UDT [FUNCTION] — Description	Dynamic UDT [DEVICE DRIVER] — Isolating Valve
	Comment	Copyright message from block header	See block header

Table 5.5 Block properties for Dynamic UDTs

- (5) Worked examples of each of these are shown below:

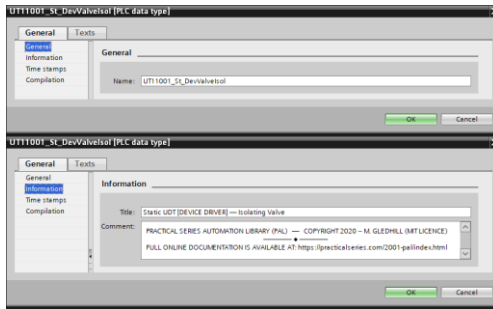


Figure 5.16 Static UDT block properties

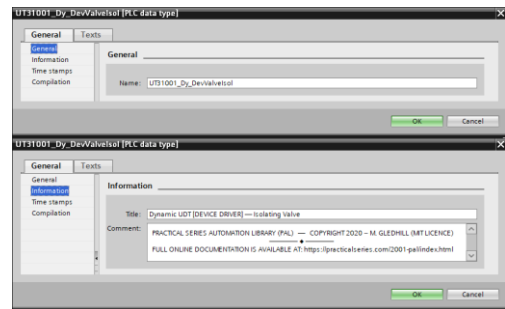


Figure 5.17 Dynamic UDT block properties

5.9 Hardware component comments

- (1) Comment fields within the hardware configuration are not used within the PAL.
- (2) The names of hardware objects that are connected to networks (CPUs, Profinet devices) are changed within the hardware configuration to reflect the naming conventions detailed in § 4.11, where such changes require further explanation, the hardware comment fields may be used
- (3) Hardware that has no such connection should not generally have its names changed from the default values supplied by TIA Portal. Some flexibility may be required where redundant cards are used.

6

Standard modules

- (1) The PAL standard modules are listed below; each standard module has its own Software Module Design Specification (SMDS). Each SMDS contains a very detailed description of what the block does and how it works.

6.1 SMDS contents

- (1) Broadly, each SMDS contains the following for the module in question:

Abstract

- ① Block technical summary
- ② Functional description
- ③ Detailed block description
- ④ Supervisory system interface
- ⑤ Parameters
- ⑥ Data structures and usage [and instance DBs]
- ⑦ Constants and Temporary (local) data
- ⑧ Block calls and associations
- ⑨ Example usage
- ⑩ Test and verification path

- (2) This is the standard format for all SMDS document and each of the above sections must satisfy the following:

Abstract (overview)

- (1) The abstract provides a concise summary or overview of the block and its functions. It serves as an introduction to the block and provides a synopsis of the block: what it does and why it does it.
- (2) The abstract should be:
 - presented as an article that can stand on its own
 - written in plain English with limited use of abbreviations and engineering jargon
 - Be short, three or four paragraphs at most
 - written in a formal tone, avoiding the use of first-person pronouns

The abstract should conform to these guidelines:

- Clearly state the purpose of the block
- Quickly summarise the functions of the block
- avoid introducing information that is not addressed in the following sections

When writing the abstract, ensure that:

- ① The message is clear
- ② It includes the key points and functions
- ③ The message is unambiguous (i.e. a reader reading the abstract could not miss the main point of the block)

Section 1 — Block technical summary

- (3) The technical summary contains the following specific information about the block:
- ① Nomenclature and addressing
 - Block title
 - Functional group
 - Address
 - Symbolic name
 - ② Block type and usage
 - Type of block (FC, FB, OB)
 - Block number
 - Programming language (LAD, STL &c.)
 - Optimise access status
 - PAL type (standard, application, template &c.)
 - ③ Software version
 - Version number
 - Status (Development, proofing, qualification &c.)
 - Date of last revision
 - Author
 - ④ Metrics
 - Execution time of the block (in microseconds)
 - Load memory size (in kilobytes)
 - Work memory size (in kilobytes)
- (4) The work memory is the size of the executable software (without comments &c.), the load memory is the size of the block (with all comments) stored on the CPU memory card.

Section 2 — Functional description

- (5) The functional description is a comprehensive examination of what the block does and how it works. It may include diagrams and supporting tables if required (the use of diagrams and tables is encouraged).
- (6) The functional description should be written in plain English with limited use of abbreviations and engineering jargon, it should clearly define the purpose of the block and the mechanisms use to achieve those purposes.
- (7) The functional description can (and usually does) contain subsection and inline sections identifying and explaining each aspect of the block.
- (8) The functional description should:
 - Describes precisely the operations performed by the block
 - Identify and explain each operating mode
 - Describe any operator interfaces (controls from an HMI or SCADA)
 - Explain any interfaces to other blocks or systems
 - Clarify any design assumptions and limitations
 - Identify all error and failure modes
 - List all calculations made by the block
- (9) The functional description should be sufficiently technically detailed, such that an engineer programming the block understands the precise requirements of the block and would have sufficient information to begin coding the block.

Section 3 — Detailed block description

- (10) The detailed block description provides the technical detail needed to build the block, this is done at an engineering level, using technical terminology common to both the programming of PLCs in general and Simatic Controllers in particular; where such terminology is used within the accepted engineering conventions and customs of this field, it is done so without further explanation.
- (11) The detailed block description expands the information given in the functional description section, it explains how the parametric information passed to the block is used to control and achieve the requirements of the module. It explains in detail any calculations that are performed by the block. It explains the function of each constant and variable passed to the modules as static and dynamic data, its purpose and how it should be interpreted by the module software.
- (12) The detailed block description describes any supervisory system interfaces and the signals passed to and from such systems. Such interfaces control the symbolic representation of the device on a mimic screen, drive any block icon and allow operator interfaces via supervisory system faceplates.
- (13) The detailed block description should as a minimum provide:
- An explanation of all formal parameters and their use
 - Details of all temporary (local) data employed by the block and how this data is used
 - An explanation of all data passed to and from the block, this should include details of all variables and constants used within the block
 - Permitted ranges of all signals
 - Interpretation of encoded data (i.e. where the value of a variable can indicate some specific mode or operation, what those values are and what meaning is applied)
 - Explanations of all operating modes and how they are selected

- Full details of any supervisory system interface (at the variable level)
- Details of all timed events (including ranges and resolutions)
- Details of all alarms, warnings and events generated by the block and the circumstances under which they are generated
- Precises details of all calculations performed by the block. This should also include any temporary or partial calculations used by the block and stored in the temporary (local) data area of the block
- Detailed explanations of any algorithms or iterative processes employed by the block

Section 4 — Supervisory system interface

⁽¹⁴⁾ Gives a full and detailed explanation of the supervisory system interface. This includes the following:

- Examples of any symbols used
- Examples of any block icons used
- Examples of any faceplates used
- Precise details of the signals used to animate the graphical objects
- Precise details of the signals that can be operated (changed) via the interface

Section 5 — Parameters

- (15) This section contains a list in tabular form of all the formal parameters associated with the block. The table includes the following:
- **Parameter** — the name of the parameter
 - **Function** — A summary explanation of what the parameter does: its purpose, and any specific states and requirements
 - **Type** — the data type of the parameter (real, int, Bool &c.). or, if used, the number of a UDT
 - **In-Out** — Identifies the nature of the parameter interface:
In: (read only)**Out:** (write only)**InOut:** (read/write)

- (16) The section must be included even if the block has no parameters, under these circumstances the following text is used:

This block has no formal parameters.

Section 6 — Data structures and usage (and instance data blocks)

- (17) This section contains a list in tabular form of any UDTs that are used by the block.
- (18) Where such structures are used, each element of the structure that is used within the block must be identified and an explanation given of how the block uses or modifies the data. To avoid duplication, this section may reference the data structure content table in some other block; however, if the block uses the data in a way that is not covered elsewhere a full description must be given. Again, this data is given in tabulated form.
- (19) The section must be included even if the block has no associated UDTs; under these circumstances the following text is used:

There are no data structures associated with this block.

- (20) If the block is a function block (FB), it will have an associated instance data block. Under these circumstances a full description of the structures and elements within the instance data block must be given (generally, this will be the static elements of the DB; all other aspects will be duplicates of section 4 (parameters) and section 6 (constants and local data), these will be created automatically when the DB is initialised).
- (21) For functions (FCs) the “(and instance data blocks)” can be removed from the section heading.

Section 7 — Constants and temporary (local) data

- (22) This section contains a list in tabular form of any constants or temporary (local) data used by the block.
- (23) Where such data is used, each constant and variable must be identified and an explanation of how the block uses that data given.
- (24) The section must be included even if the block has no associated constants or temporary (local) data, under this circumstance the following text is used:
- (25) The section is broken down into the following subsections:

Section 7.1 — Constants

- (26) Lists in tabular form all constants used by the block. If there are no constants, the following text is used:

No constants are used in this block.

Section 7.2 — Temporary (local) data

- (27) Lists in tabular form all temporary (local) data used by the block. If there are no temporary variables, the following text is used:

No temporary (local) data is used in this block.

Section 8 — Block calls and associations

- (28) This section details any blocks which may be called from within this software module (subroutine functions for example), any partner blocks with which it may be associated (for example a receive module that is partner with a transmit module &c.). It lists any system functions which may be called (e.g. reading the real time clock) and any system data types that may be used.
- (29) Finally, it lists any special calling requirements for the block (for example, must be called from within a cyclic interrupt organisation block) and if the block is using “optimised access” (this is the default arrangement).
- (30) It is broken down into the following subsections:

Section 8.1 — Block calls from within this module

- (31) This section contains a list of all the non-system block that are called from within the block (these are other PAL blocks or third-party blocks for specific equipment). The list is presented in tabular form.
- (32) The list identifies the block number, gives its title and explains how it is to be used.
- (33) The section must be included even if the block has no calls to other blocks; if this is the case, the following text is used:

There are no PAL block calls from within this block.

Section 8.2 — Blocks associated with this module

- (34) If the block is associated with other blocks, i.e. it is part of a set of blocks that together form a particular function (an example of this would be a *transmit* communication block that had a counterpart *receive* block, both being required for data to be passed between controllers); then the associated blocks must be listed here.
- (35) The list is presented in tabular form.
- (36) The list identifies the block number, gives its title and explains how it is to be used in conjunction with the block being described.

- (37) The section must be included even if the block is not associated with any other blocks; if this is the case, the following text is used:

This is a stand-alone block and is not associated with any other blocks.

Section 8.3 — System block calls and system data types

- (38) This section contains a list of all the system blocks and extended instructions that are called from within the block (these are either built into the Controller or made available via TIA Portal). The list is presented in tabular form.
- (39) The list identifies the block name (and if specified its number), gives its title and explains how it is to be used.
- (40) Some system blocks and extended instructions used preconfigured data structures referred to a *system data types* (SDTs), where such structures are used a list of the structures must be given along with an explanation of how the data elements are used.
- (41) The section must be included even if the block has no calls to system blocks or extended instructions; under these circumstances, the following text is used:

There are no system block calls.

Section 8.4 — Special properties and requirements

- (42) This section specifies any specific requirements for calling and using the block, this is usually split into two subsections:

8.4.1 Block optimisation , IEC compatibility and library conformance⁷

Does the block use Optimised Block Access (all blocks should generally be optimised, if not explain why)

Has the block been checked for IEC compatibility and is it compliant?

If the block is a standard module, it must be compatible with all IEC library-conformance module constraints

8.4.2 Calling requirements

Is the block time dependant (i.e. is it called from a timed interrupt) or can the block be called as part of the main (OB 1) cycle

- (43) If block optimisation is used, section 8.4.1 should have the following standard text:

This block is configured with Block Optimised Access (default arrangement).

- (44) Where there are no special calling requirements, Section 8.4.2 should have the following standard text:

This are no special calling requirements for this block.

⁷ Optimised access dynamically optimises the data storage within a block, it means however, that absolute addressing cannot be used to access the data (all access is symbolic).

By default block optimisation should always be used. The exceptions are where an older system (HMI's for example) can only access data using absolute addressing, under these circumstances, it is permissible to disable the optimised accessing of associated data blocks.

Section 9 — Example usage

- (45) This section contains a typical example of how the block is used and called. It should show the block call and typical data contents for the data structures used by the block
- (46) Where a block has multiple types of usage, additional examples can be included.

Section 10 —Test and verification path

- (47) This section shows all the software module tests and verification activities that have taken place to achieve the current release of the module, it has the following appearance:

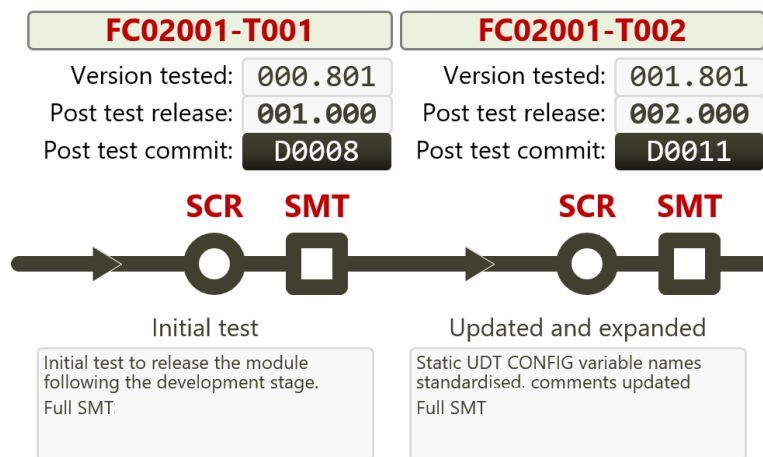


Figure 6.1 Example test and verification path

6.2 Standard block list and associated documentation

- (1) The following is a full list of all the standard modules included within the PAL software. Each module has its own Software Module Design Specification (SMDS) that gives the precise details of the block functions and how they are implemented.
- (2) The Functional Specification (FS) [Ref. 005] contains a summary of all these blocks and their functions.

6.2.1 System function modules

FC 01001	FC01001_StdSysGlobalData	SMDS: P2001-5-2312-fc01001
Standard system global data		

FC 01101	FC01101_StdSysMonoTimeSync	SMDS: P2001-5-2312-fc01101
Standard system time synchronisation (single master server)		

FC 01102	FC01102_StdSysDualTimeSync	SMDS: P2001-5-2312-fc01102
Standard system time synchronisation (dual master/slave servers)		

6.2.2 Instrument read modules

FC 02001	FC02001_StdInstAnalogRead	SMDS: P2001-5-2312-fc02001
Standard analogue instrument read, scale and monitor		

FC 02011	FC02011_StdInstRealValRead	SMDS: P2001-5-2312-fc02011
Standard real value instrument read and monitor		

FC 02101	FC02101_StdInstRealLimit	SMDS: P2001-5-2312-fc02101
Standard instrument threshold detection		

FC 02501	FC02501_StdInstDigitalRead	SMDS: P2001-5-2312-fc02501
Standard digital instrument read and monitor		

FC 02601	FC02601_StdInstDigitalFilt	SMDS: P2001-5-2312-fc02601
Standard digital filter		

6.2.3 Interlock and protection modules

FC 03002	FC03002_StdILock02	SMDS: P2001-5-2312-fc03002
Standard interlock 2 signal interlock with status reporting		
FC 03004	FC03004_StdILock04	SMDS: P2001-5-2312-fc03004
Standard interlock 4 signal interlock with status reporting		
FC 03008	FC03008_StdILock08	SMDS: P2001-5-2312-fc03008
Standard interlock 8 signal interlock with status reporting		
FC 03102	FC03102_StdILockPerm02	SMDS: P2001-5-2312-fc03102
Standard interlock 2 signal permissive interlock with status reporting		
FC 03104	FC03104_StdILockPerm04	SMDS: P2001-5-2312-fc03104
Standard interlock 4 signal permissive interlock with status reporting		
FC 03108	FC03108_StdILockPerm08	SMDS: P2001-5-2312-fc03108
Standard interlock 8 signal permissive interlock with status reporting		
FC 03202	FC03202_StdILockTrip02	SMDS: P2001-5-2312-fc03202
Standard interlock 2 signal trip interlock with status reporting		
FC 03204	FC03204_StdILockTrip04	SMDS: P2001-5-2312-fc03204
Standard interlock 4 signal trip interlock with status reporting		
FC 03208	FC03208_StdILockTrip08	SMDS: P2001-5-2312-fc03208
Standard interlock 8 signal trip interlock with status reporting		
FC 03501	FC03501_StdILockMsgGen	SMDS: P2001-5-2312-fc03501
Standard interlock message signal generation		

6.2.4 Safety and safety system modules

FC 04002	FC04002_StdSafeZoneNorm02	SMDS: P2001-5-2312-fc04002
Standard safety 2 signal E-stop zone group with status reporting		
FC 04004	FC04004_StdSafeZoneNorm04	SMDS: P2001-5-2312-fc04004
Standard safety 4 signal E-stop zone group with status reporting		
FC 04008	FC04008_StdSafeZoneNorm08	SMDS: P2001-5-2312-fc04008
Standard safety 8 signal E-stop zone group with status reporting		
FC 04202	FC04202_StdSafeZoneTrip02	SMDS: P2001-5-2312-fc04202
Standard safety 2 signal E-stop latching zone group with status reporting		
FC 04204	FC04204_StdSafeZoneTrip04	SMDS: P2001-5-2312-fc04204
Standard safety 4 signal E-stop latching zone group with status reporting		
FC 04208	FC04208_StdSafeZoneTrip08	SMDS: P2001-5-2312-fc04208
Standard safety 8 signal E-stop latching zone group with status reporting		
FC 04501	FC04501_StdSafeMsgGen	SMDS: P2001-5-2312-fc04501
Standard safety message signal generation		

6.2.5 Calculations and mathematics modules

FC 05001	FC05001_StdCalcAvg	SMDS: P2001-5-2312-fc05001
Standard calculation — simple average		
FC 05002	FC05002_StdCalcAvgRolling	SMDS: P2001-5-2312-fc05002
Standard calculation — rolling average		
FC 05003	FC05003_StdCalcAvgCumulate	SMDS: P2001-5-2312-fc05003
Standard calculation — cumulative average		
FC 05004	FC05004_StdCalcAvgWeighted	SMDS: P2001-5-2312-fc05004
Standard calculation — weighted rolling average		
FC 05005	FC05005_StdCalcAvgExp	SMDS: P2001-5-2312-fc05005
Standard calculation — exponential rolling average		

FC 05101	FC05101_StdCalcDiffRoC	SMDS: P2001-5-2312-fc05101
Standard calculation — rate-of-change		
FC 05102	FC05102_StdCalcDiffRoCAvg	SMDS: P2001-5-2312-fc05102
Standard calculation — average rate-of-change		
FC 05201	FC05201_StdCalcIntArea	SMDS: P2001-5-2312-fc05201
Standard calculation — signal integration (area)		
FC 05301	FC05301_StdCalcValToPercent	SMDS: P2001-5-2312-fc05301
Standard calculation — convert a ranged value to a percentage		
FC 05302	FC05302_StdCalcPercentToVal	SMDS: P2001-5-2312-fc05302
Standard calculation — convert a percentage to a ranged value		
FC 05351	FC05351_StdCalcPercentToPulse	SMDS: P2001-5-2312-fc05351
Standard calculation — convert a percentage to a variable mark/space square wave		
FC 05352	FC05352_StdCalcPulseToPercent	SMDS: P2001-5-2312-fc05352
Standard calculation — convert a variable mark/space square wave to a percentage		
FC 05361	FC05361_StdCalcPulseToState	SMDS: P2001-5-2312-fc05361
Standard calculation — convert a pulse train to an ON/OFF state		
FC 05362	FC05362_StdCalcStateToPulse	SMDS: P2001-5-2312-fc05362
Standard calculation — convert an ON/OFF state to a pulse train		
FC 05363	FC05363_StdCalcPulseToFreq	SMDS: P2001-5-2312-fc05363
Standard calculation — convert a square wave pulse train to a frequency		

FC 05502	FC05502_StdCalcPulseDual	SMDS: P2001-5-2312-fc05502
Standard calculation — pulse generator 2 (dual) state		
FC 05503	FC05503_StdCalcPulseTri	SMDS: P2001-5-2312-fc05503
Standard calculation — pulse generator 3 (tri) state		
FC 05504	FC05504_StdCalcPulseQuad	SMDS: P2001-5-2312-fc05504
Standard calculation — pulse generator 4 (quad) state		
FC 05508	FC05508_StdCalcPulseOcta	SMDS: P2001-5-2312-fc05508
Standard calculation — pulse generator 8 (octa) state		
FC 05516	FC05516_StdCalcPulseHexa	SMDS: P2001-5-2312-fc05516
Standard calculation — pulse generator 16 (hexa) state		

FC 05601	FC05601_StdCalcWaveRamp	SMDS: P2001-5-2312-fc05601
Standard calculation — waveform generator ramp function		
FC 05602	FC05602_StdCalcWaveSaw	SMDS: P2001-5-2312-fc05602
Standard calculation — waveform generator continuous sawtooth wave function		
FC 05603	FC05603_StdCalcWaveTri	SMDS: P2001-5-2312-fc05603
Standard calculation — waveform generator continuous triangular wave function		
FC 05604	FC05604_StdCalcWaveSin	SMDS: P2001-5-2312-fc05604
Standard calculation — waveform generator continuous sine wave function		
FC 05605	FC05605_StdCalcWaveCos	SMDS: P2001-5-2312-fc05605
Standard calculation — waveform generator continuous cosine wave function		

6.2.6 Sequential control

FC 0700I	FC0700I_StdSeqIEC_Control	SMDS: P2001-5-2312-fc0700I
Standard sequence — IEC compliant sequence manager (controller)		
FC 0701I	FC0701I_StdSeqIEC_OSL	SMDS: P2001-5-2312-fc0701I
Standard sequence — IEC compliant sequence operating state logic (OSL)		
FC 0702I	FC0702I_StdSeqIEC_Step	SMDS: P2001-5-2312-fc0702I
Standard sequence — IEC compliant sequence step/transition manager		
FC 0750I	FC0750I_StdSeqNonIEC_Control	SMDS: P2001-5-2312-fc0750I
Standard sequence — non-IEC compliant sequence manager (controller)		
FC 0751I	FC0751I_StdSeqNonIEC_OSL	SMDS: P2001-5-2312-fc0751I
Standard sequence — non-IEC compliant sequence operating state logic (OSL)		
FC 0752I	FC0752I_StdSeqNonIEC_Step	SMDS: P2001-5-2312-fc0751I
Standard sequence — non-IEC compliant sequence step/transition manager		

6.2.7 Device drivers — Control loops

FC I000I	FCI000I_StdDevPID_Standard	SMDS: P2001-5-2312-fcI000I
Standard device driver — control loops — standard PID loop		
FC I001I	FCI001I_StdDevPID_Sched	SMDS: P2001-5-2312-fcI001I
Standard device driver — control loops — standard PID loop with gain scheduling		
FC I002I	FCI002I_StdDevPID_Split	SMDS: P2001-5-2312-fcI002I
Standard device driver — control loops — split range modifier		
FC I0022	FCI0022_StdDevPID_Poly	SMDS: P2001-5-2312-fcI0022
Standard device driver — control loops — polyline modifier		
FC I0022	FCI010I_StdDevPID_External	SMDS: P2001-5-2312-fcI010I
Standard device driver — control loops — polyline modifier		
FC I050I	FCI050I_StdDevPID_LookUp	SMDS: P2001-5-2312-fcI050I
Standard device driver — control loops — polyline modifier		

6.2.8 Device drivers — Valves

FC 11001	FC11001_StdDevValveSol	SMDS: P2001-5-2312-fc11001
Standard device driver — valves — isolating valve		
FC 11011	FC11011_StdDevValve3Way	SMDS: P2001-5-2312-fc11011
Standard device driver — valves — 3-way valve		
FC 11101	FC11101_StdDevValveBi	SMDS: P2001-5-2312-fc11101
Standard device driver — valves — bistable isolating valve		
FC 11501	FC11501_StdDevValveMod	SMDS: P2001-5-2312-fc11501
Standard device driver — valves — modulating valve		

6.2.9 Device drivers — Drives

FC 12001	FC12001_StdDevDriveDOL	SMDS: P2001-5-2312-fc12001
Standard device driver — drives — direct online		
FC 12011	FC12011_StdDevDriveDOLRev	SMDS: P2001-5-2312-fc12011
Standard device driver — drives — direct online reversing		
FC 12101	FC12101_StdDevDriveBi	SMDS: P2001-5-2312-fc12101
Standard device driver — drives — bistable		
FC 12111	FC12111_StdDevDriveBiRev	SMDS: P2001-5-2312-fc12111
Standard device driver — drives — bistable reversing		
FC 12501	FC12501_StdDevDriveVSD	SMDS: P2001-5-2312-fc12501
Standard device driver — drives — variable speed		
FC 12511	FC12511_StdDevDriveVSDRev	SMDS: P2001-5-2312-fc12511
Standard device driver — drives — variable speed reversing		
FC 12601	FC12601_StdDevDriveMSD	SMDS: P2001-5-2312-fc12601
Standard device driver — drives — multiple speed		

6.2.10 Message handling

FC I6001	FCI6001_StdMsgAnalogAlm	SMDS: P2001-5-2312-fcI6001
Standard message handler — analogue alarm		
FC I6002	FCI6002_StdMsgAnalogWrn	SMDS: P2001-5-2312-fcI6002
Standard message handler — analogue warning		
FC I6003	FCI6003_StdMsgAnalogEvent	SMDS: P2001-5-2312-fcI6003
Standard message handler — analogue event		
FC I6101	FCI6101_StdMsgDigitalAlm	SMDS: P2001-5-2312-fcI6101
Standard message handler — digital alarm		
FC I6102	FCI6102_StdMsgDigitalWrn	SMDS: P2001-5-2312-fcI6102
Standard message handler — digital warning		
FC I6103	FCI6103_StdMsgDigitalEvent	SMDS: P2001-5-2312-fcI6103
Standard message handler — digital event		
FC I6201	FCI6201_StdMsgAlmTime	SMDS: P2001-5-2312-fcI6201
Standard message handler — digital time-stamped alarm		
FC I6202	FCI6202_StdMsgWrnTime	SMDS: P2001-5-2312-fcI6202
Standard message handler — digital time-stamped warning		
FC I6203	FCI6203_StdMsgEventTime	SMDS: P2001-5-2312-fcI6203
Standard message handler — digital time-stamped event		
FC I6501	FCI6501_StdMsgPrompMgr	SMDS: P2001-5-2312-fcI6501
Standard message handler — prompt manager		
FC I6502	FCI6502_StdMsgPrompQueue	SMDS: P2001-5-2312-fcI6502
Standard message handler — prompt queue		

6.2.11 Communication handling

FC I7001	FCI7001_StdCommsGetSmall	SMDS: P2001-5-23 I2-fcI7001
Standard communication handler — get data from a controller (small)		
FC I7002	FCI7002_StdCommsPutSmall	SMDS: P2001-5-23 I2-fcI7002
Standard communication handler — put data into a controller (small)		
FC I7101	FCI7101_StdCommsRead65K	SMDS: P2001-5-23 I2-fcI7101
Standard communication handler — read data from a controller (65K of data)		
FC I7102	FCI7102_StdCommsWrite65K	SMDS: P2001-5-23 I2-fcI7102
Standard communication handler — write data to a controller (65K of data)		
FC I7401	FCI7401_StdCommsSetIP	SMDS: P2001-5-23 I2-fcI7401
Standard communication handler — dynamically configure Ethernet interface		
FC I7501	FCI7501_StdCommsPtP_Rx	SMDS: P2001-5-23 I2-fcI7501
Standard communication handler — read data via a point-to-point interface		
FC I7502	FCI7502_StdCommsPtP_Tx	SMDS: P2001-5-23 I2-fcI7502
Standard communication handler — write data via a point-to-point interface		

6.2.12 Subroutines

FC 18001	FC18001_StdSubScaleAI	SMDS: P2001-5-2312-fc18001
Standard subroutines — scale an analogue input signal		
FC 18002	FC18002_StdSubScaleAQ	SMDS: P2001-5-2312-fc18002
Standard subroutines — scale an analogue output signal		
FC 18101	FC18101_StdSubTime100ms	SMDS: P2001-5-2312-fc18101
Standard subroutines — timer module (100 ms resolution)		
FC 18104	FC18104_StdSubTime1s	SMDS: P2001-5-2312-fc18104
Standard subroutines — timer module (1 s resolution)		
FC 18111	FC18111_StdSubTimeLong	SMDS: P2001-5-2312-fc18111
Standard subroutines — timer module, long duration timer		
FC 18151	FC18151_StdSubTimeEventRTC	SMDS: P2001-5-2312-fc18151
Standard subroutines — event duration timer (using the RTC)		
FC 18201	FC18201_StdSubCounter	SMDS: P2001-5-2312-fc18201
Standard subroutines — count up/down function		
FC 18901	FC18901_StdSubStrIntToASC	SMDS: P2001-5-2312-fc18901
Standard subroutines — string function — convert an integer to ASCII		
FC 18902	FC18902_StdSubStrRealToASC	SMDS: P2001-5-2312-fc18902
Standard subroutines — string function — convert a real to ASCII		
FC 18911	FC18911_StdSubStrASCtoInt	SMDS: P2001-5-2312-fc18911
Standard subroutines — string function — convert an ASCII string to an integer value		
FC 18912	FC18912_StdSubStrASCtoReal	SMDS: P2001-5-2312-fc18912
Standard subroutines — string function — convert an ASCII string to a real value		

FC 18921	FC18921_StdSubStrCaseConv	SMDS: P2001-5-2312-fc18921
Standard subroutines — string function — case conversion		
FC 18931	FC18931_StdSubStrConcat	SMDS: P2001-5-2312-fc18931
Standard subroutines — string function — concatenate strings		
FC 18932	FC18932_StdSubStrSplit	SMDS: P2001-5-2312-fc18932
Standard subroutines — string function — split a string		
FC 18933	FC18933_StdSubStrFind	SMDS: P2001-5-2312-fc18933
Standard subroutines — string function — find a string within a string		

6.2.13 Debug subroutines

FC 19001	FC19001_StdDebugValveSol	SMDS: P2001-5-2312-fc19001
Standard debug subroutines — simulation — isolating valve		
FC 19002	FC19002_StdDebugValveBi	SMDS: P2001-5-2312-fc19002
Standard debug subroutines — simulation — bistable isolating valve		
FC 19003	FC19003_StdDebugValveMod	SMDS: P2001-5-2312-fc19003
Standard debug subroutines — simulation — modulating valve		
FC 19011	FC19011_StdDebugDriveDOL	SMDS: P2001-5-2312-fc19011
Standard debug subroutines — simulation — drive DOL		
FC 19012	FC19012_StdDebugDriveBi	SMDS: P2001-5-2312-fc19012
Standard debug subroutines — simulation — drive bistable		
FC 19013	FC19013_StdDebugDriveVSD	SMDS: P2001-5-2312-fc19013
Standard debug subroutines — simulation — drive variable speed		
FC 19014	FC19014_StdDebugDriveMSD	SMDS: P2001-5-2312-fc19014
Standard debug subroutines — simulation — drive multiple speed		

FC 19101	FC19101_StdDebugInstFlow	SMDS: P2001-5-2312-fc19101
Standard debug subroutines — simulation — instrument flow		
FC 19102	FC19102_StdDebugInstLevel	SMDS: P2001-5-2312-fc19102
Standard debug subroutines — simulation — instrument level		
FC 19103	FC19103_StdDebugInstTemp	SMDS: P2001-5-2312-fc19103
Standard debug subroutines — simulation — instrument temperature		
FC 19104	FC19104_StdDebugInstPress	SMDS: P2001-5-2312-fc19104
Standard debug subroutines — simulation — instrument pressure		
FC 19151	FC19151_StdDebugInst1Order	SMDS: P2001-5-2312-fc19151
Standard debug subroutines — simulation — instrument 1 st order response		
FC 19152	FC19152_StdDebugInst2Order	SMDS: P2001-5-2312-fc19152
Standard debug subroutines — simulation — instrument 2 nd order response		
FC 19153	FC19153_StdDebugInstPoly	SMDS: P2001-5-2312-fc19153
Standard debug subroutines — simulation — polyline response		
FC 19701	FC19701_StdDebugSeqBreak	SMDS: P2001-5-2312-fc19701
Standard debug subroutines — sequence breakpoint		
FC 19999	FC19999_StdDebugForceStop	SMDS: P2001-5-2312-fc19999
Standard debug subroutines — Force CPU stop		

7

Application modules

- (1) The PAL software consists mainly of standard modules; these are the library modules of the PAL. The application modules are project specific modules that call the standard modules as needed by the project in question.
- (2) It is the application modules that provide the structure for Controller software.
- (3) While it is true that the application modules are different for each Controller project; the actual arrangement and numbering of the application modules does form part of the PAL. It is the application modules that are called from the main program cycle organisation block (OB 1) and this determines the structure of the software.
- (4) The complete OB 1 PAL structure is shown in Figure 7.1. This shows application block calls to the thirteen functional groups.

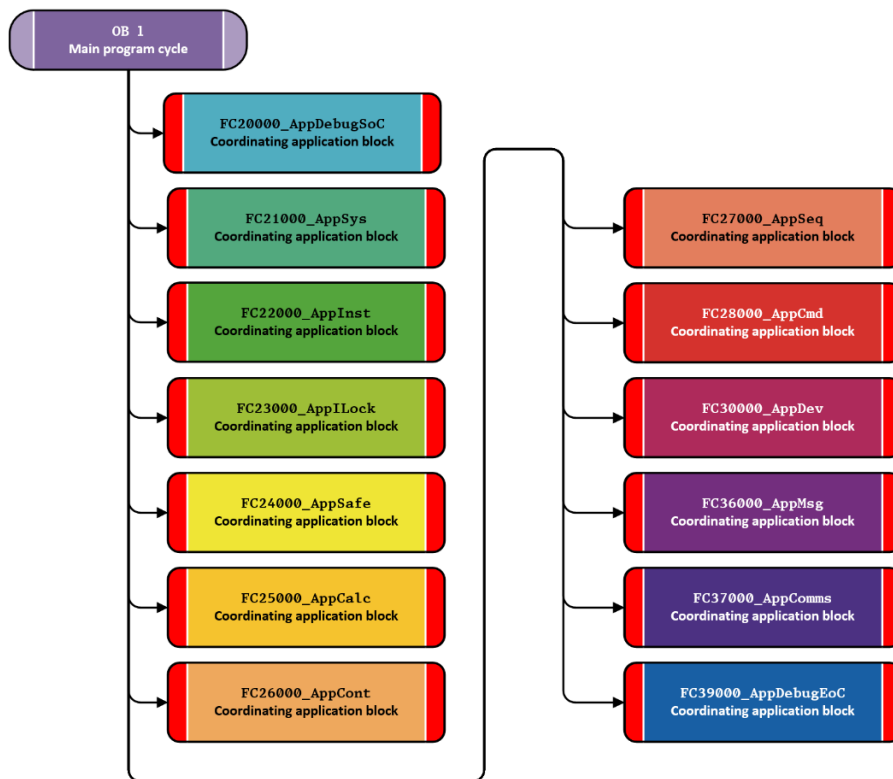


Figure 7.1 Complete OB 1 PAL structure

- (5) 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.
- (6) 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.
- (7) 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 **Programming** *Programming* modules contain extensive programming statements, rather than the configuration exercises used with coordinating and marshalling modules.
- Programmed module contains software specific to the purpose of the Controller in question and contain substantial logical statements and software
- (8) All the application modules within OB 1 are *coordinating* application modules, these are the highest level of application module and all such modules have numbers ending in 000.

FUNCTION GROUP COORDINATING MODULE MARSHALLING MODULE PROGRAMMING MODULE

System functions	FC21000_AppSys	None	None
Instrumentation	FC22000_AppInst	FC22001_AppInstAnalogRead	None
		FC22501_AppInstDigitalRead	None
Interlocks & protection	FC23000_AppLock	FC23001_AppLockXXX	Optional
		FC23101_AppLockXXX	Optional
		FC23011_AppLockYYY	Optional
		FC23112_AppLockYYY	Optional
Safety systems	FC24000_AppSafe	FC24001_AppSafeXXX	Optional
		FC24101_AppSafeXXX	Optional
		FC24012_AppSafeYYY	Optional
		FC24112_AppSafeYYY	Optional
Calculations & mathematics	FC25000_AppCalc	FC25001_AppCalcAvg	Optional
		FC25012_AppCalcWaveRamp	Optional
		FC25011_AppCalcAvgYYY	Optional
Continuous control logic	FC26000_AppCont	FC26001_AppContXXX	Optional
		FC26101_AppContXXX	Optional
		FC26012_AppContYYY	Optional
		FC26111_AppContYYY	Optional
		FC26112_AppContYYY	Optional
FC26113_AppContYYY	Optional		

FUNCTION GROUP	COORDINATING MODULE	MARSHALLING MODULE	PROGRAMMING MODULE
Sequential control logic	FC27000_AppSeq	Allocated if sequences can be grouped into meaningful areas	Optional FC27011_AppSeqName FC27012_AppSeqName
Command handling	FC28000_AppCmd	FC28001_AppCmdName	Optional FC28011_AppCmdName FC28012_AppCmdName FC28011_AppCmdName
Device driver — control loops	FC30000_AppDevPID	FC30001_AppDevPID_Standard	Optional None
		FC30011_AppDevPID_Sched	Optional None
Device Driver — Valves	FC31000_AppDevValves	FC31001_AppDevValveSol	Optional None
		FC31501_AppDevValveMod	Optional None
Device Driver — Drives	FC32000_AppDevDrives	FC32001_AppDevDriveDOL	Optional None
		FC32501_AppDevDriveVSD	Optional None
Messages	FC36000_AppMsg	FC36001_AppMsgAnalog	Optional FC36011_AppMsgAnalogArea
		FC36101_AppMsgDigital	Optional FC36111_AppMsgDigitalArea
		FC36501_AppMsgPrompts	Optional FC36511_AppMsgPromptsArea
Communication handling	FC37000_AppComms	FC37001_AppCommsArea	Optional FC37011_AppCommsArea
		FC37101_AppCommsArea	Optional FC37111_AppCommsArea

Table 7.1 Typical application module allocation

7.1 Application module numbering

- (1) Application modules are always functions (FCs) without parameters, they are numbered according to functional group, this is summarised as follows:

FUNCTION GROUP	FUNCTION GROUP NUMBER (FG)	APPLICATION MODULE NUMBER RANGE
Debug (start of cycle)	20	FC 20nnn
System functions	21	FC 21nnn
Read instruments	22	FC 22nnn
Interlock & protection	23	FC 23nnn
Safety systems	24	FC 24nnn
Calculations & mathematics	27	FC 25nnn
Continuous control	26	FC 26nnn
Sequential control	27	FC 27nnn
Command handling	28	FC 28nnn
Reserved	29	N/A
Device drivers (control loops)	30	FC 30nnn
Device drivers (valves)	31	FC 31nnn
Device drivers (drives)	32	FC 32nnn
Device drivers (Reserved)	33	FC 33nnn
Device drivers (Reserved)	34	FC 34nnn
Device drivers (Reserved)	35	FC 35nnn
Message handling	36	FC 36nnn
Communication handling	37	FC 37nnn
(subroutines)	38	N/A
Debug (end of cycle)	39	FC 39nnn

Table 7.2 Application module numbering by function group

- (2) Each function group has a single coordinating module and this always has the block number **FG000** where **FG** is the function group number.
- (3) Where marshalling modules are used to subdivide the functional groups, but programming modules are not used (this is specifically: instruments and device drivers) the marshalling modules adopt the last three digits of the standard modules that they contain; for example, the following would be the complete set of marshalling modules for the drives, device driver:

COORDINATING MODULE	MARSHALLING MODULES	STANDARD MODULES CALLED
FC32000_AppDevDrives	FC32001_AppDevDriveDOL	FC12001_StdDevDriveDOL
	FC32011_AppDevDriveDOLRev	FC12011_StdDevDriveDOLRev
	FC32101_AppDevDriveBi	FC12101_StdDevDriveBi
	FC32111_AppDevDriveBiRev	FC12111_StdDevDriveBiRev
	FC32501_AppDevDriveVSD	FC12501_StdDevDriveVSD
	FC32511_AppDevDriveVSDRev	FC12511_StdDevDriveVSDRev
	FC32601_AppDevDriveMSD	FC12601_StdDevDriveMSD

Table 7.3 Marshalling application module numbering example for instruments and device drivers

- (4) In Table 7.3 the last three digits of the marshalling application module number matches the last three digits of the standard module block number that is called from within the marshalling block.
- (5) Where programming application modules are (or may be) used, the numbering system is more flexible. Here, marshalling modules may be used (optionally) to organise the programming modules into specific areas or subgroups that have some meaning for the Controller program in question (these may be plant areas or logical areas that have some relevance in terms of the program itself).
- (6) The allocation of marshalling modules under these circumstances is at the discretion of the programmer. Good practice dictates that large gaps are left in the numbering allocation of the marshalling modules (if there are fewer than ten marshalling modules within the functional group, it is recommended that each marshalling module has an increment of 100 from the previous module e.g. FC28001, FC28101, FC28201 &c.).
- (7) Programming application modules called from within these marshalling modules should adopt the three most significant digits of the marshalling module, thus extending the previous example, the marshalling module FC28101 could support within it the programming modules numbered FC28102 to FC28199 (98 modules in total).
- (8) Again, it is good practice to leave a gap between programming modules, for example, incrementing by 10 such that the programming modules would be numbered FC28111, FC28121, FC28131 &c.
- (9) Where programming application modules are called directly from the coordinating module, then there are no marshalling modules and the programming modules can have any number in the functional group (excepting FG000, this being the number of the coordinating module, FC28000 in the previous example)

7.2 Sequence annotation

- (1) One particular form of programming application module is that which contains a sequence.
- (2) The Functional Specification (FS) [Ref: 005], annotated the steps and transitions of a sequence using a step-transition diagram of a sequential flow chart (sometimes referred to as a GRAFCET⁸ diagram), and example of which is shown below in Figure 7.2:

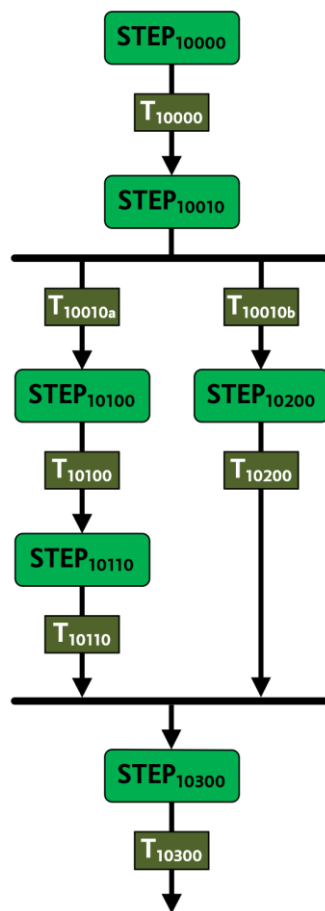


Figure 7.2 Step transition diagram

⁸ GRAFCET, *GRAPHe de Commande Etape-Transition*, French. Literally, “stage-transition command graph” a diagrammatic mechanism for showing steps and transitions within a sequence.

- (3) Such step-transition diagrams can be time consuming and difficult to produce, requiring a graphical drawing or CAD package to render the diagrams. An alternative arrangement is to use sequential IO matrices, these can be produced on a spread sheet and list all the devices controlled by the sequence; identifying each step and the required state of each device within that step.
- (4) Consider the backwash sequence discussed in the example of § 2.1.2, a single filter from this example is shown below:

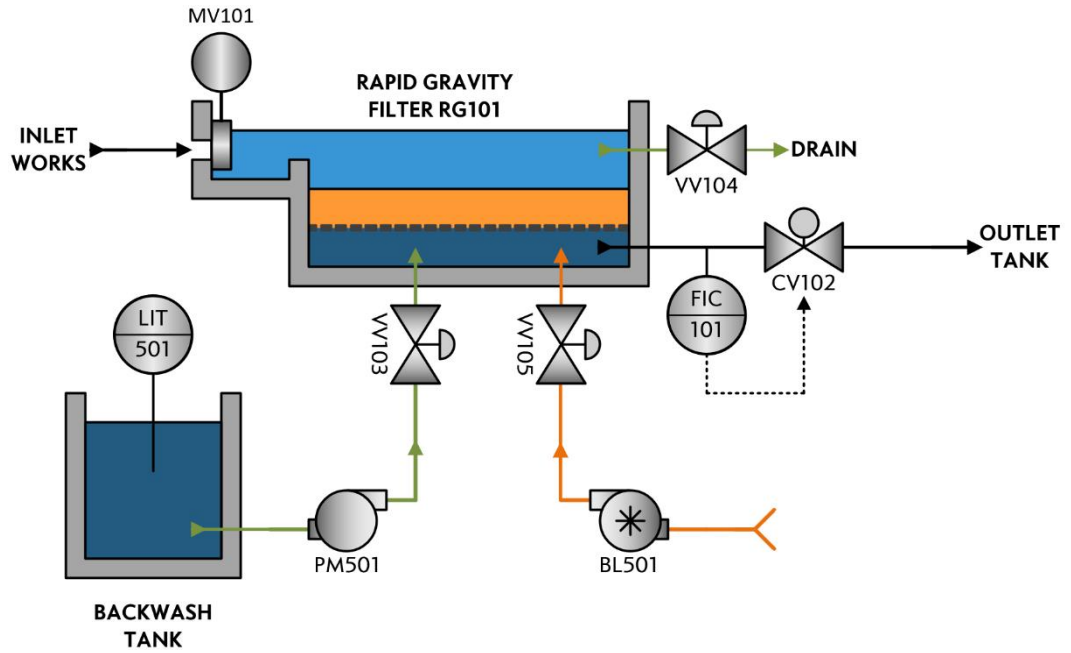


Figure 7.3 Step transition diagram

- (5) This is the full list of devices associated with filter RG101:

MV101	Filter I inlet valve
CV102	Filter I outlet valve
VV103	Filter I backwash water inlet valve
VV104	Filter I backwash water outlet valve
VV105	Filter I Air inlet valve
BL501	Backwash blower I
PM501	Backwash pump I

Table 7.4 Equipment associated with filter RG101

- (6) The backwash sequence for the filter had the following plain English description:
- Isolate the filter (take it out of service and close all valves)
 - Aerate the filter (open air inlet valve and start blower)
 - Backwash the filter with aeration (open backwash inlet and outlet valves and start backwash pump)
 - Washout the filter (stop blower and close air inlet valve)
 - Allow filter bed to settle (stop pump and close backwash valves)
 - Return filter to service (open inlet and outlet valves)
- (7) This sequence would more formally be documented using a Sequential Input/Output (IO) Matrix as follows:

Filter RG101 Backwash Sequence NORMAL OPERATION		Inlet valve	Outlet valve	Backwash inlet valve	Backwash outlet valve	Air inlet valve	Backwash blower	Backwash pump
Step	Step description (green) and transition conditions (blue)	MY101	CV102	VI103	VI104	VI105	BL501	PM501
00000	Sequence Idle							
	Wait for timed backwash start signal (RG101_BW_start = true)	N/A	N/A	N/A	N/A	N/A	N/A	N/A
IDLE								

Starting

Generate backwash started message		Inlet valve	Outlet valve	Backwash inlet valve	Backwash outlet valve	Air inlet valve	Backwash blower	Backwash pump
01000	Generate message. Activate Cmd_Run							
	No transition condition	N/A	N/A	N/A	N/A	N/A	N/A	N/A
STARTING	Msg_RG1010_001: Filter 1 backwash started							

Filter RG101 Backwash Sequence NORMAL OPERATION		Inlet valve	Outlet valve	Backwash inlet valve	Backwash outlet valve	Air inlet valve	Backwash blower	Backwash pump
Step	Step description (green) and transition conditions (blue)	MV101	CV102	VV103	VV104	VV105	BL501	PM501

Running

Close inlet and outlet valves								
10000 RUNNING	Close all filter valves							
	All valves are confirmed closed	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP	STOP
	NB. VV103, VV104, VV105 will already be closed							
Open air inlet valve								
10010 RUNNING	Open VV105							
	VV105 is confirmed open	CLOSE	CLOSE	CLOSE	CLOSE	OPEN	STOP	STOP
Start air blower								
10020 RUNNING	Start BL501							
	Wait for aeration time RG101_BW_AirTime	CLOSE	CLOSE	CLOSE	CLOSE	OPEN	RUN	STOP
Open backwash inlet and outlet valves								
10030 RUNNING	Open VV103 and VV104							
	VV103 and VV104 are confirmed open	CLOSE	CLOSE	OPEN	OPEN	OPEN	RUN	STOP
Start backwash pump (filter is now cleaning, both backwash pump and blower are running)								
10040 RUNNING	Start PM501							
	Wait for clean time RG101_BW_CleanTime	CLOSE	CLOSE	OPEN	OPEN	OPEN	RUN	RUN
Stop aeration, stop blower								
10050 RUNNING	Stop BL501							
	BL501 is confirmed stopped	CLOSE	CLOSE	OPEN	OPEN	OPEN	STOP	RUN

Filter RG101 Backwash Sequence NORMAL OPERATION		Inlet valve	Outlet valve	Backwash inlet valve	Backwash outlet valve	Air inlet valve	Backwash blower	Backwash pump
Step	Step description (green) and transition conditions (blue)	MV101	CV102	VV103	VV104	VV105	BL501	PM501
Start close air inlet valve								
10060 RUNNING	Close VV105			OPEN	OPEN	CLOSE	STOP	RUN
	VV105 is confirmed closed	CLOSE	CLOSE					
Filter is washing out								
10070 RUNNING	No additional actions			OPEN	OPEN	CLOSE	STOP	RUN
	Wait for washout time RG101_BW_WashOutTime	CLOSE	CLOSE					
Stop backwash pump								
10080 RUNNING	Stop PM501			OPEN	OPEN	CLOSE	STOP	STOP
	PM501 is confirmed stopped	CLOSE	CLOSE					
Close backwash valves								
10090 RUNNING	Close VV103 and VV104			CLOSE	CLOSE	CLOSE	STOP	STOP
	VV103 and VV104 are confirmed closed	CLOSE	CLOSE					
Filter is settling								
10100 RUNNING	No additional actions			CLOSE	CLOSE	CLOSE	STOP	STOP
	Wait for settling time RG101_BW_SettleTime	CLOSE	CLOSE					
Open filter inlet valve								
10110 RUNNING	Open MV101			OPEN	CLOSE	CLOSE	STOP	STOP
	MV101 is confirmed open	OPEN	CLOSE					

Filter RG101 Backwash Sequence NORMAL OPERATION		Inlet valve MV101	Outlet valve CV102	Backwash inlet valve V103	Backwash outlet valve V104	Air inlet valve V105	Backwash blower BL501	Backwash pump PM501
Step	Step description (green) and transition conditions (blue)							
Open filter outlet valve (under PID control)								
10120 RUNNING	Activate CV102 PID Control	OPEN	PID	CLOSE	CLOSE	CLOSE	STOP	STOP
	FIC101 is reading above minimum flow FIC101 >= FiltMinFlow							
Trigger completing								
10130 RUNNING	Activate Cmd_Complete	OPEN	PID	CLOSE	CLOSE	CLOSE	STOP	STOP
	No transition condition							
Completing								
Generated backwash complete message								
20000 RUNNING	Generate message. Activate Cmd_Completed	OPEN	PID	CLOSE	CLOSE	CLOSE	STOP	STOP
	FIC101 is reading above minimum flow FIC101 >= FiltMinFlow							
	Msg_RG1010_002: Filter 1 backwash complete							
Completed								
Completed, return to IDLE state								
29000 RUNNING	Activate Cmd_RetIdlee	OPEN	PID	CLOSE	CLOSE	CLOSE	STOP	STOP
	No transition condition							

- (8) If a fault occurs during the sequence, all backwash operations will stop and the filter will be placed in an isolated condition until the operator either issues a **RESUME** command, in which case the backwash sequence will restart from step 01000 (from the beginning).
- (9) The operator could also **ABORT** the backwash, under which condition the filter will be returned to service.
- (10) A fault condition is any condition affecting the devices associated with the filter or backwash equipment:
- MV1010 fault
 - CV102 fault
 - VV103 fault
 - VV104 fault
 - VV105 fault
 - BL501 fault
 - PM501 fault

Note: If, during a backwash, a low alarm is generated by LIT501 indicating a low level in the backwash tank, this is not considered a fault; there is sufficient capacity in the backwash tank at this point to complete the backwash. LIT501 low alarm would however, prevent a backwash sequence from starting

- (11) The hold, resume and abort logic for the sequence is as follows:

Filter RG101 Backwash Sequence FAULT OPERATION		Backwash pump	Backwash blower	Air inlet valve	Backwash outlet valve	Backwash inlet valve	Outlet valve	Inlet valve
Step	Step description (green) and transition conditions (blue)	PM501	BL501	VV105	VV104	VV103	CV102	MV101

Error Holding

Stop backwash pump and blower								
40000 ERROR HOLDING	Stop PM501 and BL501	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	STOP
	Wait for 10 seconds	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	STOP
		UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	UNCHANGED	STOP
Close all valves								
40010 ERROR HOLDING	Close VV103, VV104, VV105, MV101 and CV102	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
	Wait for 10 seconds	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
		CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
Close all valves								
40020 ERROR HOLDING	Activate Cmd_ErrHeld	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
	No transition condition	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
		CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP

Error Held

Close all valves								
45000 ERROR HOLDING	Generate message	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
	Msg_RG1010_003: Filter 1 backwash failed; sequence HELD	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
		CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP

Error Resuming

Restart the backwash sequence (trigger RUN command and return to step 10000)								
50000 ERROR HOLDING	Generate message. Activate Cmd_Run	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
	Msg_RG1010_003: Filter 1 backwash restarted	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP
		CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	STOP

Filter RG101 Backwash Sequence ABORT OPERATION		Inlet valve	Outlet valve	Backwash inlet valve	Backwash outlet valve	Air inlet valve	Backwash blower	Backwash pump
Step	Step description (green) and Transition conditions (blue)	MV101	CV102	VI103	VI104	VI105	BL501	PM501

Aborting

Return filter to service, open filter inlet valve								
60000 ABORTING	Open MV101	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	STOP	STOP
	MV101 is confirmed open							
Open filter outlet valve (under PID control)								
60010 ABORTING	Activate CV102 PID Control	OPEN	PID	CLOSE	CLOSE	CLOSE	STOP	RUN
	FIC101 is reading above minimum flow FIC101 >= FiltMinFlow							
Trigger aborted								
60020 ABORTING	Activate Cmd_Aborted	OPEN	PID	CLOSE	CLOSE	CLOSE	STOP	RUN
	No transition condition							

Aborted

Generated backwash complete message								
65000 ABORTED	Generate message. Activate Cmd_RetIdle	OPEN	PID	CLOSE	CLOSE	CLOSE	STOP	RUN
	No transition							
	Msg_RG1010_004: Filter I returned to service backwash not complete							

7.2.1 Sequence IO matrix summary

TITLE Additional notes		Device descriptions	
Step	Step description (green) and transition conditions (blue)	Device tags	
STEP	DESCRIPTION OF ACTIONS	REQUIRED STATES FOR THIS STEP	
	PROMPT		TRANSITION CONDITIONS
STATE	MESSAGES		NOTES
STATE			
STEP SUMMARY OR DESCRIPTION			
STEP	DESCRIPTION OF ACTIONS	REQUIRED STATES FOR THIS STEP	
	PROMPT		TRANSITION CONDITIONS
STATE	MESSAGES		NOTES

- (1) Sequence IO matrices are the preferred mechanism for documenting sequence, the use of spread sheets for this operation makes such documentation easier to generate and maintain.
- (2) Sequence IO Matrices are easy to adapt and use as test documents, the requirements for each step being easily determined.

8

Interrupt modules

- (1) Programme execution within the PAL (and in all Simatic Controllers) is entirely event driven, some event must take place in order to execute any software. Such events are detected by the Controller operating system and in response to the specific event, the Controller triggers a specific organisation block.
- (2) The main example of this is the cyclic triggering of OB 1, OB1 is triggered automatically by the Controller operating system at the start of each scan after the process image has been updated.
- (3) Two forms of interrupt organisation block exist: the first being operational interrupts that occur in response to some normal, expected event (a time of day, a hardwired signal &c.). The second form is in response to an unexpected event, an error or a fault condition (a card failure, a programming error &c.).
- (4) The following tables list the two types of interrupt organisation blocks

NORMAL INTERRUPTS		
OB NUMBER	PAL MODULE NAME	DESCRIPTION
OB 1	OB00001_IntlNrmMainProgram	Controller main program cycle Called at the start of each Controller cycle
OB 10	OB00010_IntlNrmNTimeOfDay	Time of day Interrupt Called by time and day of week
OB 20	OB00020_IntlNrmNTimeDelay	Time delay Interrupt Called after a specified delay has expired
OB 30	OB00030_IntlNrmNCyclic	Timed cyclic Interrupt Called at specified intervals
OB 40	OB00040_IntlNrmNHardware	Hardware Interrupt Called when a specified signal is detected
OB 100	OB00100_IntlNrmNStartUp	Start-up Interrupt Called when the CPU transitions to RUN

Table 8.1 Standard (normal) interrupt modules and organisation blocks

ERROR INTERRUPTS

OB NUMBER	PAL MODULE NAME	DESCRIPTION
OB 80	OB00080_IntlErrECycleTimeErr	Error Interrupt Maximum cycle time exceeded
OB 82	OB00082_IntlErrEModuleDiag	Error Interrupt Module diagnostics signal received (module fault)
OB 83	OB00083_IntlErrEModuleChange	Error Interrupt Module changed, removed or installed
OB 86	OB00086_IntlErrERackErr	Error Interrupt Rack failure or fault
OB 121	OB00121_IntlErrEProgramErr	Error Interrupt Programming fault or error
OB 122	OB00122_IntlErrEIOErr	Error Interrupt IO card access fault

Table 8.2 Fault interrupt modules and organisation blocks

- (5) Irrespective of the type of interrupt and with the exception of the main cycle organisation block (OB 1), all interrupt OBs carryout a certain set of (minimum) activities.
- (6) All such OBs record the time of their last call in the *DB21001_Dy_SysGlobalData* data block. The time is stored in the *DTL* format (*DateTimeLong* data type). The *DTL* format provides the following data (stored as unsigned integers):

NAME	TYPE	DESCRIPTION
YEAR	UInt	Year (1970-2262)
MONTH	USInt	Month (01-12)
DAY	USInt	Day (01-31)
WEEKDAY	USInt	Day of week (1-7 where 1 = Sunday, 7 = Saturday)
HOUR	USInt	Hour (00-23)
MINUTE	USInt	Minute (00-59)
SECOND	USInt	Second (00-59)
NANOSECOND	USDInt	Nanoseconds (000,000,000- 999,999,999)

Table 8.3 Execution time record of an OB

- (7) All organisation blocks with the exception of OB 1 must do this, and the software associated with the time stamp is stored in the first network following the current revision and modification history (this will usually be network 3), the time stamp is recorded as follows:

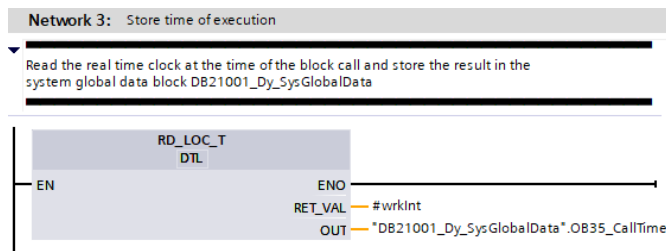


Figure 8.1 Step transition diagram

8.1 Error detection OBs

- (1) Error interrupt OBs (see Table 8.2) detect specific faults that can occur within a Controller. Such faults are exceptions, and are not expected to occur during the normal operation of the Controller.
- (2) By default, if an error occurs (say a remote IO module panel is turned off), then the Controller CPU will automatically enter the STOP mode, all Controller outputs will be turned off and user programme will no longer be executed.
- (3) By most definitions, this is an extreme reaction, and while the failure of a remote IO rack may drastically affect the operation of the plant, it is generally better to keep the Controller running and reporting the existence of the fault rather than just entering STOP mode.
- (4) To prevent this reaction, the appropriate error organisation block must be present within the Controller (in the case of the previous example, the OB in question would be OB86, rack failure or fault). If the block is present (even if it does not contain any code) the CPU will not enter STOP mode.
- (5) By default, the PAL contains a full set of error interrupt OBs, and, unless there is a specific reason not to do so, all should be installed in the target software project. This prevents the CPU stopping under all error conditions.

BLANK PAGE

9

Template modules

- (1) A full set of template modules are supplied with the PAL software. These template modules give worked examples of how the *standard* and *application* modules should be used in a control system project.
- (2) The template modules 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. There are also template modules for each type of interrupt OB.
- (3) The template modules are based around an example Fermenter project, this Fermenter project is the basis of the User Guide (UG) [Ref. 009] documentation that is issued as part of the training documentation associated with this project.
- (4) The Fermenter project is a relatively simple project, but covers all aspects of the PAL software, and provides a comprehensive guide to using all aspects of the PAL software.

9.1 Templates for application 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) All template modules will be fully documented and will reflect the PAL documentation standards given in the Style Guide (SG) [Ref. 010].
- (3) The following table gives the associated numbering between template modules and application modules:

FUNCTION GROUP	TEMPLATE MODULE NUMBER	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 9.1 Template module and application module associations

- (4) The Fermenter project represented in the template modules and detailed in the User Guide [Ref: 009] contains the following set of blocks:

TEMPLATE MODULES			ASSOCIATED
COORDINATING	MARSHALLING	PROGRAMMING	APPLICATION MODULE
FC40000_TmtDebugSOS		FC40101_TmtDebugInst	FC20000_AppDebugSOS FC20101_AppDebugInst
FC41000_TmtSysFunctions			FC21000_AppSysFunctions
FC42000_TmtInstRead	FC42001_TmtInstAnalogRead FC42001_TmtInstDigitalRead		FC22000_AppInstRead FC22001_AppInstAnalogRead FC22001_AppInstAnalogRead
FC43000_TmtLock		FC43101_TmtLockArea1 FC43201_TmtLockArea2 FC43301_TmtLockArea3 FC43401_TmtLockArea4	FC23000_AppLock FC23101_AppLockArea1 FC23201_AppLockArea2 FC23301_AppLockArea3 FC23401_AppLockArea4
FC44000_TmtSafe		FC44101_TmtSafeZone1	FC24000_AppSafe FC24101_AppSafeZone1
FC45000_TmtCalc	FC45001_TmtCalcAvg	FC45700_TmtCalcNabla	FC25000_AppCalc FC25001_AppCalcAvg FC25700_AppCalcNabla

TEMPLATE MODULES			ASSOCIATED
COORDINATING	MARSHALLING	PROGRAMMING	APPLICATION MODULE
FC46000_TmtContLogic		FC4610I_TmtContStt FC4620I_TmtContInoc FC4630I_TmtContVent	FC26000_AppContLogic FC4610I_AppContStt FC4620I_AppContInoc FC4630I_AppContVent
FC47000_TmtSeqLogic		FC4710I_TmtSeqExec FC4720I_TmtSeqSter FC4730I_TmtSeqFerm FC4740I_TmtSeqCIP FC4760I_TmtSeqAgit	FC27000_AppSeqLogic FC2710I_AppSeqExec FC2720I_AppSeqSter FC2730I_AppSeqFerm FC2740I_AppSeqCIP FC2760I_AppSeqAgit
FC48000_TmtCmdHandler		FC4800I_TmtCmdPID FC4810I_TmtCmdVlvIsol FC4815I_TmtCmdVlvMod FC4820I_TmtCmdDriveDOL FC4825I_TmtCmdDriveVSD	FC28000_AppCmdHandler FC2800I_AppCmdPID FC2810I_AppCmdVlvIsol FC2815I_AppCmdVlvMod FC2820I_AppCmdDriveDOL FC2825I_AppCmdDriveVSD
FC50000_TmtDevDriver	FC5000I_TmtDevPID FC5100I_TmtDevVlvIsol FC5150I_TmtDevVlvMod FC5200I_TmtDevDrvDOL FC5250I_TmtDevDrvVSD		FC30000_AppDevDriver FC3000I_AppDevPID FC3100I_AppDevVlvIsol FC3150I_AppDevVlvMod FC3200I_AppDevDrvDOL FC3250I_AppDevDrvVSD
FC56000_TmtMsgHandling		FC5610I_TmtMsgClassify	FC36000_AppMsgHandling FC3610I_AppMsgClassify
FC57000_TmtCommsHandling	FC5510I_TmtCommsCon2		FC37000_AppCommsHandling FC3510I_AppCommsCon2
FC59000_TmtDebugEOS		FC5910I_TmtDebugSim FC5920I_TmtDebugSeq	FC39000_AppDebugEOS FC3910I_AppDebugSim FC3920I_AppDebugSeq

Table 9.2 Full list of template modules and associated application modules

(5)

9.2 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 FC 60035.
- (3) The following lists all the template modules for organisation block and their associated OB number:

TEMPLATE MODULE	ASSOCIATED OB	INTERRUPT TYPE
FC60001_TmdlNrmNMainProgram	OB00001_IntlNrmNMainProgram	Controller main program cycle Called at the start of each Controller cycle
FC60010_TmdlNrmNTimeOfDay	OB00010_IntlNrmNTimeOfDay	Time of day Interrupt Called by time and day of week
FC60020_TmdlNrmNTimeDelay	OB00020_IntlNrmNTimeDelay	Time delay Interrupt Called after a specified delay has expired
FC60030_TmdlNrmNCyclic	OB00030_IntlNrmNCyclic	Timed cyclic Interrupt Called at specified intervals
FC60040_TmdlNrmNHardware	OB00040_IntlNrmNHardware	Hardware Interrupt Called when a specified signal is detected
FC60080_TmdlErrECycleTimeErr	OB00080_IntlErrECycleTimeErr	Error Interrupt Maximum cycle time exceeded
FC60082_TmdlErrEModuleDiag	OB00082_IntlErrEModuleDiag	Error Interrupt Module diagnostics signal received (module fault)
FC60083_TmdlErrEModuleChange	OB00083_IntlErrEModuleChange	Error Interrupt Module changed, removed or installed
FC60086_TmdlErrERackErr	OB00086_IntlErrERackErr	Error Interrupt Rack failure or fault
FC60100_TmdlErrEStartUp	OB00100_IntlErrEStartUp	Start-up Interrupt Called when the CPU transitions to RUN
FC60121_TmdlErrEProgramErr	OB00121_IntlErrEProgramErr	Error Interrupt Programming fault or error
FC60122_TmdlErrEIOErr	OB00122_IntlErrEIOErr	Error Interrupt IO card access fault

Table 9.3 Template modules for organisation blocks

10

Documentation 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 Section 4 and are further discussed in the Style Guide [Ref. 010].
- (3) 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.
- (4) The document modules have the following allocations:

NUMBER	CLASS	FUNCTION
		Example block comments, containing the following:
FC61000	Doc	<ul style="list-style-type: none">• 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 I 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 10.1 Document modules for the PAL

BLANK PAGE

11

Common approach to data handling

- (1) All standard modules receive all their data via parameters, these can be simple IO and discrete signals passed as individual items to the block, or can be more complex data structures that form the `STATIC_DATA` and `DYNAMIC_DATA` passed to the block in the form of UDTs.
- (2) This section concentrates on the `STATIC_DATA` and `DYNAMIC_DATA`.
- (3) 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.
- (4) By separating static data from the dynamic data, it provides an additional means of verifying the software installed in a Controller is the correct version of the software.

11.1 Conventions for using UDTs

- (1) As a general rule, all standard modules have both static and dynamic data passed to them via the `STATIC_DATA` and `DYNAMIC_DATA` parameters, this data is different for every standard module. The static data for an isolating valve is entirely different to the static data required when reading an instrument, even closely associated devices (an isolating valve and a bistable valve for example) have differences in the data structures.
- (2) In all cases, a standard module will have a unique static data structure with the same number as the standard module, and will have a unique dynamic data structure with the same number as the standard module Plus 20000. For example, the isolating valve module has the block number FC11001, and this uses a static UDT with number UT11001 and a dynamic UDT with the number UT310001.

- (3) While all static UDTs are different from each other and all dynamic UDTs also differ from each other, there are commonalities in terms of function. For example, compare some of the signals generated for an isolating valve and for a DOL drive:











ISOLATING VALVE			DOL DRIVE		
SYMBOL	SIGNAL	MEANING	SYMBOL	SIGNAL	MEANING
	Status_Closed	Closed		Status_Stopped	Stopped
	Status_Opening	Opening		Status_Starting	Starting
	Status_Opened	Opened		Status_Running	Running
	Status_Closing	Closing		Status_Stopping	Stopping
	Status_Fault	Fault		Status_Fault	Fault

Table 11.1 Commonality of signals

- (4) Although the signals are different, the valve using closed/open terminology and the drive stopped/running terminology, there is clearly a commonality of function, all signals are *status* signals for example, they show the status of the device. And to this end, each signal starts with the word `status_` to indicate this.
- (5) This approach is adopted thorough out the static and dynamic data structures:

11.1.1 Static UDT conventions

- (1) Static UDT entries (variables and constants) are in uppercase, using the following conventions (see § 4.6):
- 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

(2) Further, each entry name is in the format:

FUNC_NAME

Where:

ITEM	MEANING	DETAILS
FUNC	Function	Up to 8 characters
NAME	Variable name	

(3) The name component can be any meaningful name given to the variable; the name should be chosen such that the total name of the variable (FUNC_NAME) is no more than 21 characters.

(4) Examples are:

CONFIG_ALM_H_EN

RANGE_RAW_MIN

TIME_OPEN_MAX

- (5) The **FUNC** component of the name has various preset values:

FUNC VALUE	DESCRIPTION
CONFIG_	Configuration of an option, usually a Boolean switch value that turns a particular mode or function on or off
INFO_	Provides some form of information about the object (such as a tag name or instrument units &c.) usually in the form of a string
RANGE_	Specifies a range for a value, for example, it could specify the minimum and maximum range of a scaled analogue signal
REVINFO_	Contains revision information
SP_	Setpoint, identifies a particular analogue value (such as the alarm setpoint for an instrument). Setpoints are usually real numbers
TIME_	Identifies a timed operation value (for example, the maximum time it takes a valve to open)
K_	A constant used for some specific purpose

Table 11.2 Static UDT function names

11.1.2 Dynamic UDT conventions

- (1) Static UDT entries (variables and constants) are in uppercase, using the following conventions (see § 4.6):

- ① The name must be written in camel case
- 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 [_]

- (2) Further, each entry name is in the format:

func_Name

Where:

ITEM	MEANING	DETAILS
func	Function	Up to 8 characters
Name	Variable name	

(3) The name component can be any meaningful name given to the variable; the name should be chosen such that the total name of the variable (`func_Name`) is no more than 25 characters.

(4) Examples are:

`status_Alm_H`

`mode_SimOn`

`time_Opening`

`actual_Value`

(5) The `func` component of the name has various preset values:

FUNC VALUE	DESCRIPTION
<code>actual_</code>	Indicates an actual value (such as the true reading of an instrument)
<code>batch_</code>	Indicates the variable belongs to a batch process (usually storing an ID number that indicates which batch process has control of the device)
<code>cal_</code>	A calculated value
<code>ctrl_</code>	A control signal (used to directly control the device, an output for example or a signal which may be applied to an output under specific conditions)
<code>status_</code>	Status, indicates the status of the device for use elsewhere within the software (e.g. <code>status_Open</code> indicates a valve is in the open state)
<code>mode_</code>	Indicates an operating mode (e.g. <code>mode_Manual</code> indicates the device is under manual control)
<code>msg_</code>	Identifies the variable as a message (alarm, warning or event)
<code>prompt_</code>	Identifies the variable as a prompt (a special form of a message that requires a user response)
<code>result_</code>	Indicates the result of a calculation or other evaluation process
<code>revInfo</code>	Revision information
<code>time_</code>	Identifies a variable that store a timer value (this is the actual value of a running timer)
<code>\$xxxx_</code>	The <code>\$</code> indicates that the variable is an internal working value for the module in question (for example <code>\$pret</code> would indicate a positive edge retention variable). Variable beginning with <code>\$</code> should not be used externally to the standard module.

Table 11.3 Dynamic UDT function names

BLANK PAGE

12

Common modes of operation

- (1) Devices and instruments that have some form of interface that can be linked to a supervisory system have several common modes of operation, the PAL accommodates the following modes of operation (usually, but not exclusively selected by the operator via a supervisory system).
- Automatic/manual mode
 - Bypass mode for interlocks
 - Simulation mode
 - Remote/local operation
 - Disabling supervisory system faceplates

12.1 Manual mode

- (1) Manual mode is applicable to devices (rather than instruments): control loops, valves and drives.
- (2) Manual mode allows the operator to take control of a device and override the automatic operations of the Controller software.
- (3) If a device is in automatic mode, the Controller software determines the state of the device and the device will respond accordingly, if the software requires that a valve be open, the valve will be opened automatically by the software (usually within the command handling function group).
- (4) If the operator switches a device to manual mode, then the requirements of the Controller software are ignored and the valve will only respond to commands from the operator.
- (5) The exception to this rule is that if a device is in manual and an interlock condition arises that requires the valve to be in a particular state (interlock, permissive or trip),

then this will take priority over the manual operation and the device will be driven to the state required by the interlock condition (for example if a valve is in manual mode and has been opened by the operator when an interlock condition arises that requires the valve to close, then the valve will close, overriding the manual command. Once the interlock condition is removed, the valve will go back to the manual state required by the operator).

- (6) Interlock conditions can also be overridden, see the bypass mode, § 12.2.
- (7) Switching a device from automatic mode to manual mode is a *bumpless* action; that is to say, when the device enters manual mode, the state applied to the device will be the last state that it had in automatic mode (for example, if a drive were running under automatic control, when that drive was switched to manual mode, it would continue to run — the manual state adopts the same state as the last automatic state).
- (8) The following data points are associated with the simulation mode:

SIGNAL	FUNCTION	TYPE	DETAILS
<code>CONFIG_MAN_DIS</code>	Prevents manual mode being activated under all circumstances	Bool	1 = manual mode disabled 0 = manual mode permitted
<code>mode_AutMan</code>	Activates manual mode, if active, the device will adopt the state given in <code>ctrl_ManState</code>	Bool	1 = automatic mode on (manual off) 0 = manual mode on (auto off)
<code>ctrl_Man_State</code>	Sets the device to a particular state if <code>mode_AutMan</code> is set to manual (if <code>mode_AutMan</code> = off). For example for a valve this would be <code>ctrl_Man_OpenClose</code>	Bool	1 = set manual device to state 1 0 = set manual device to state 2
<code>ctrl_Aut_State</code>	Sets the device to a particular state if <code>mode_AutMan</code> is set to automatic (if <code>mode_AutMan</code> = on). For example for a valve this would be <code>ctrl_Aut_OpenClose</code>	Bool	1 = set manual device to state 2 0 = set manual device to state 1
<code>status_Man</code>	Status indication, shows if manual mode is active	Bool	1 = manual mode active 0 = manual mode inactive
<code>status_Aut</code>	Status indication, shows if automatic mode is active	Bool	1 = automatic mode active 0 = automatic mode inactive

Table 12.1 Manual mode signals

- (9) Manual mode can only be selected by the operator, usually via a supervisory system.
- (10) If manual mode is disabled (configuration signal `CONFIG_MAN_DIS` is set to 1), then the manual mode cannot be turned on under any circumstances and the standard module will set the device permanently to automatic mode.

12.2 Bypass mode

- (1) Bypass mode is applicable to devices (rather than instruments): control loops, valves and drives.
- (2) Bypass mode overrides any interlock, permissive or trip that may be active.

Note: Emergency stop signals cannot be bypassed

- (3) If bypass mode is active, the interlock, permissive or trip signal will be ignored and the device will operate as if the interlock, permissive or trip signal were not active. Bypass mode can be activated in both automatic mode and in manual mode.
- (4) The following data points are associated with bypass mode:

SIGNAL	FUNCTION	TYPE	DETAILS
CONFIG_BYPASS_DIS	Prevents bypass mode being activated under all circumstances	Bool	1 = bypass mode disabled 0 = bypass mode permitted
mode_BypassOn	Activates bypass mode, if active an interlock, permissive or trip signal will be ignored (bypassed)	Bool	1 = bypass mode on 0 = bypass mode off
status_BypassOn	Status indication, shows if manual mode is active	Bool	1 = bypass mode active 0 = bypass mode inactive

Table 12.2 Bypass mode signals

- (5) Bypass mode is selected by the operator, usually via a supervisory system or via a key switch type operation.
- (6) If bypass mode is disabled (configuration signal CONFIG_BYPASS_DIS is set to 1), then the bypass mode cannot be turned on under any circumstances and the standard module will ignore any attempt to do so.

12.3 Simulation mode

- (1) Both instrument and devices can be switched to a *simulation* mode. If an instrument is switched to simulation mode, the actual reading from the instrument is replaced by a simulated value provided by the operator.
- (2) If a device is switched to simulation mode, any feedback signals (such as limit switch signals from a valve) are automatically simulated and follow the demanded state of the device.
- (3) It is possible when in simulation mode for a device to be given a specific set of signals (i.e. to simulate a valve being closed, open or to have no limit switch signals) instead of the simulating the actual state of the device
- (4) Simulation mode is generally used during testing, but can also be applied in a process environment if a fault condition is detected within an instrument or device; this scenario allows the plant to continue operating (albeit under some degree of manual control) until the equipment is repaired or replaced.
- (5) The following data points are associated with the simulation mode:

SIGNAL	FUNCTION	TYPE	DETAILS
<code>CONFIG_SIM_DIS</code>	Prevents simulation mode being activated under all circumstances	Bool	1 = no simulation, 0 = simulation permitted
<code>mode_SimOn</code>	Activates simulation mode, if active for an instrument the scaled reading of the instrument (<code>Value</code>) will be set to <code>SimValue</code> If active for a device, the device will adopt either the demanded state or any of the state signals below	Bool	1 = simulation mode on, 0 = simulation mode off
<code>mode_SimValue</code>	Scaled instrument reading (<code>Value</code>) will be set to this if simulation mode is on (<code>CONFIG_SIM_OFF</code> = 0 and <code>mode_SimOn</code> = 1)	Real	Simulated instrument value in engineering units
<code>status_SimOn</code>	Status indication, shows if simulation mode is active	Bool	1 = sim mode active 0 = sim mode inactive

Table 12.3 Simulation mode signals

- (6) Simulation mode can only be activated by the operator usually via a supervisory system.
- (7) If simulation mode is disabled (configuration signal `CONFIG_SIM_DIS` is set to **1**), then the simulation mode cannot be turned on under any circumstances, the option is greyed out on the supervisory system faceplate and the block will constantly reset the `mode_SimOn` signal.

12.4 Remote/local mode

- (1) Remote and local operating modes refer to the supervisory system that has control of the instrument (i.e. which system can write to the device and change the operating mode of the device).
- (2) This type of mode is usual present where a plant has a remote central control location (a control room) that normally controls the device (remote control), but also has a field panel with a local HMI that an operator in the field can select to take over control of the device (local control) for maintenance purposes.
- (3) The modes are as follows (and are mutually exclusive, only one will be active):

Remote	Only the remote system in the control room can control the device
Local	A local system has taken control of the device and the remote system can no longer issue commands to it
Remote/local disabled (ALL mode)	The system does not use remote/local modes and any supervisory system can issue commands to the device

- (4) Remote/local control is generally handled by the supervisory systems themselves; however, the supervisory systems need a storage area per device to record which mode that device is in, local control is often activated by a panel key-switch, that changes the state of several devices from remote to local (usually all the devices controlled by the local panel).
- (5) The block simply identifies the remote/local mode from the mode signals and configuration signals provided to the block.
- (6) For systems that do not use remote/local modes (any supervisory system can control the device, or control is determined by operator privileges), the remote/local modes can be disabled (`CONFIG_RL_EN` is set to `false`) and the device is effectively in ALL mode (any system can control the device).
- (7) If remote/local operation is in use, the supervisory system that **does not** have control will still display all the information from the device, but will not be able to control the

device (it could not for example, activate simulation mode) and all control function (faceplate functions) will be greyed out.

(8) The following signals are associated with remote/local/all modes:

SIGNAL	FUNCTION	TYPE	DETAILS
<code>CONFIG_RL_EN</code>	Allows Remote and Local modes to be selected. If set to zero, the device does not have remote and local operation and both <code>mode_RemoteOn</code> and <code>mode_LocalOn</code> are set to zero (is in ALL mode).	Bool	1 = Remote/local can be selected 0 = All mode is active
<code>mode_RemoteOn</code>	Activates remote mode and resets local mode (<code>CONFIG_RL_ENABLE</code> must be set to 1)	Bool	1 = remote mode on, 0 = remote mode off
<code>mode_LocalOn</code>	Activates local mode and resets remote mode (<code>CONFIG_RL_ENABLE</code> must be set to 1)	Bool	1 = local mode on, 0 = local mode off
<code>status_RemoteOn</code>	Status indication, active if device is in remote mode (<code>mode_RemoteOn = 1</code>)	Bool	1 = remote mode on, 0 = remote mode off
<code>status_LocalOn</code>	Status indication, active if device is in local mode (<code>mode_LocalOn = 1</code>)	Bool	1 = local mode on, 0 = local mode off
<code>status_RLOff</code>	Status indication, ALL mode is active (<code>mode_RemoteOn = 0</code> and <code>mode_LocalOn = 0</code>)	Bool	1 = ALL mode on, 0 = ALL mode off

Table 12.4 Remote/local mode signals

(9) By default, remote/local is disabled — ALL mode is selected.

12.5 Faceplate disable mode

- (1) It is possible to disable the supervisory system faceplate for any device or instrument, as follows:

SIGNAL	FUNCTION	TYPE	DETAILS
CONFIG_FP_DIS	Disable the supervisory system faceplate; if this signal is active, clicking the block icon or device symbol will not activate the device faceplate pop-up. I.e. the operator will never be able to issue instructions to the device	Bool	1 = Faceplate disabled 0 = normal

Table 12.5 Faceplate disable signal

- (2) If the faceplate is disabled (`CONFIG_FP_DIS = 1`), the supervisory system will not allow the device faceplate to be opened (normally achieved by clicking the block icon or device symbol), this in turn prevents the operator from affecting the device in anyway, it would not, for example, be possible to put the device into simulation mode.
- (3) Similarly, if the faceplate is disabled, the standard module will set the device to auto-automatic mode (if applicable), will set remote/local mode to ALL and will disable all other modes (simulation, bypass, manual &c.).

BLANK PAGE

13

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. 007]
 - 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, working under the confines and structures imposed by the TIA User Documentation requirements. This website provides a standard format for displaying the PAL user documentation.

13.1 Organising the user documentation

- (1) When a project is created in TIA Portal (for example the PAL project), TIA Portal stores the project in new directory with the given name of the project. The project is a series of files and directories all stored within a root directory with the given name of the project itself.
- (2) For example, this is the project directory structure for a TIA Portal project, in this case from one of the proof-of-concept projects from the early stages of the PAL project:

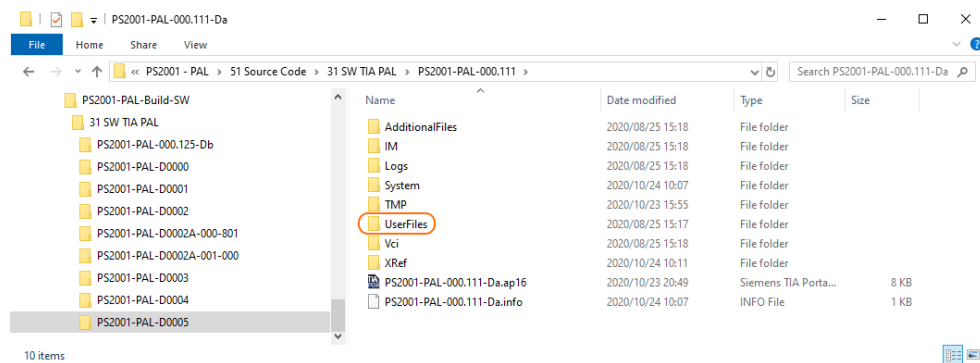


Figure 13.1 Step transition diagram

- (3) All of these folders within the TIA project are created by TIA Portal when the project itself is created.
- (4) The folders themselves are required by TIA Portal and the contents of those folders should not be directly modified or changed externally to TIA Portal (any attempt to do so will render the project corrupt).
- (5) The one folder that is directly accessible to the user is the **UserFiles** folder; it is this folder that holds the user documentation files that can be opened within TIA Portal.
- (6) Such user documentation is accessed from within TIA Portal by selecting a block within the project tree and pressing **<shift> F1**, if a user document can be found for the selected block, it will be automatically opened.
- (7) To access the user documentation, the documents must be stored in a particular location. For English language documentation (that used by the PAL), the structure is:

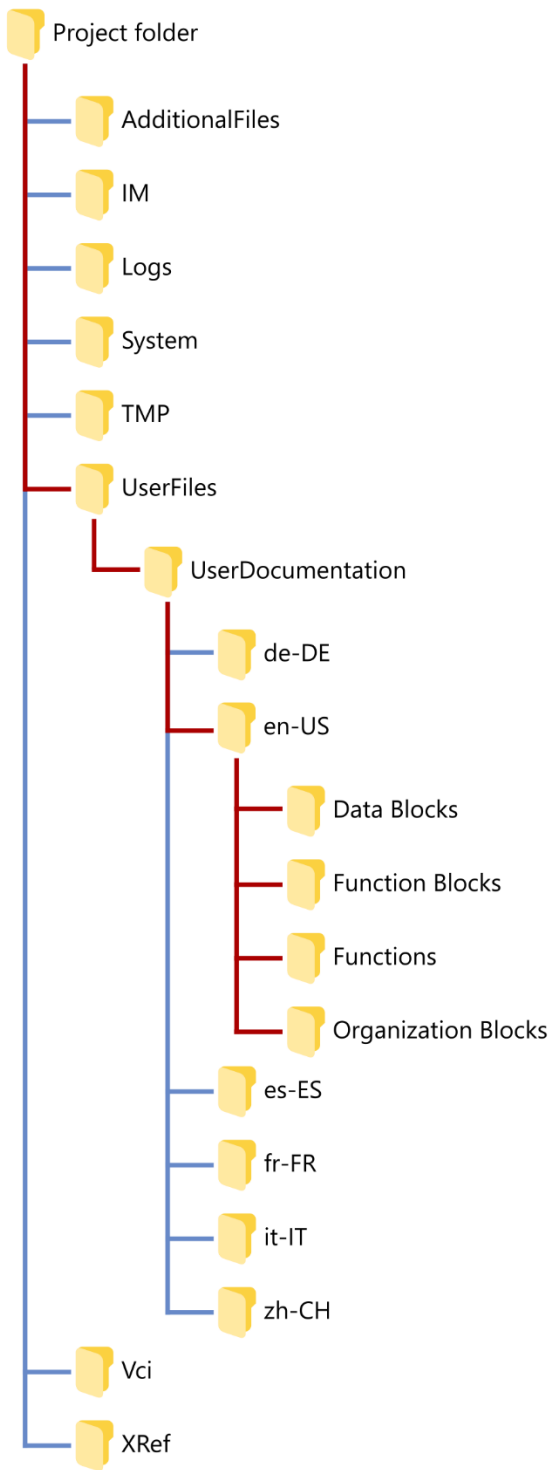


Figure 13.2 User documentation folder structure

The directory `UserDocumentation` must be created under the automatically created `UserFiles`.

The `UserDocumentation` contains subdirectories that correspond to the language selected as the main user language for TIA Portal, the language should be set to English.

This means that for the PAL, all the document data is stored under the `en-US` directory.

The other directories that can be used are:

LANGUAGE	FOLDER NAME
English	<code>en-US</code>
Chinese	<code>zh-CH</code>
French	<code>fr-FR</code>
German	<code>de-DE</code>
Italian	<code>it-IT</code>
Spanish	<code>es-ES</code>

Table 13.1 User documentation language folders

To use any of the other folders, the TIA Portal User interface language must be changed, this also requires that support for those languages was installed during the installation of TIA Portal.

Again, the PAL expects the language to be English and requires only the `en-US` directory.

- (8) The folders under the **en-US** directory contain the documentation for each type block within the project, the documentation for each FC is stored in the **Functions** directory, documentation for data blocks is stored the **Data Blocks** folder &c. The full list of block related folders is:

FOLDER NAME	CONTAINS
Data blocks	Documentation for data blocks (DBs)
Function blocks	Documentation for function blocks (FBs)
Functions	Documentation for functions (FCs)
Organization blocks	Documentation for organisation blocks (OBs)

Table 13.2 Folders for each type of block documentation

*Note: The organisation block folder (**Organization blocks**) must be spelt with a Z (not an S as is more common in Great Britain).*

Although the blocks all start with a capital letter, TIA Portal is not case sensitive in regard to the folder names.

- (9) The PAL user documentation is in the form of web pages; all web pages are PDF⁹ documents and end with the extension **.pdf**.
- (10) The document for a particular module or block must be stored in the relevant folder and the document given the same name as the block itself. For example, the System Global Data module uses the function FC01001 and has the name **FC01001_StdGlobalData**, this can be seen in the project tree below:

⁹ PDF: Portable document format, now standardised as ISO 32000, is a file format developed by Adobe in 1992 to present documents, including text formatting and images, in a manner independent of application software and hardware.

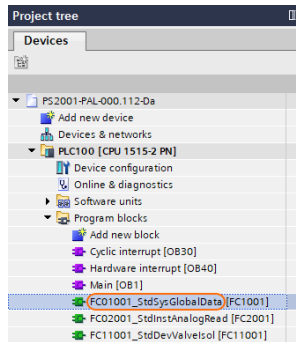


Figure 13.3 Function name

- (11) The corresponding document file for FC01001 must be stored in the Functions folder and must have the same name as the block to which it is related. In this case the full path and file name are:

UserFiles/UserDocumentation/en-US/Functions/FC01001_StdGlobalData.pdf

- (12) The file name for the block documentation is [FC01001_StdGlobalData.pdf](#); i.e. the same name as the block with the [.pdf](#) extension.
- (13) By highlighting the block in the TIA Portal and pressing **<SHIFT> FI**, TIA Portal will open the file [FC01001_StdGlobalData.pdf](#) in the default web browser (or in the application associated with the file type, e.g. Adobe reader).
- (14) The PAL has user documentation for all the software modules contained within it, selecting any block within the project tree will open a pdf document for that module, this will be the Software Module Design Specification (SMDS) for that module.
- (15) The PAL user documentation also contains other documentation relevant to the library, and these documents can be accessed from any of the block document files. The additional documents that can be accessed are:

- The User Guide *[Ref. 009]*
- The software Design Specification *[Ref. 007]*
- The Style Guide *[Ref. 010]*

13.1.1 The use of a home page

- (1) If a document (in the case of the PAL a pdf web page) for a module does not exist, when **<SHIFT> F1** is selected for that module, TIA Portal will generate an error message highlighting its absence.
- (2) To avoid this, a default document can be placed in the language folder (**en-US** in this case). The document must be called **home** (with which ever extension is being used), in the case of the PAL this is **home.pdf**.
- (3) If no specific document can be found for the particular module, TIA Portal will open the default **home** document instead.

13.2 Project specific User Documentation

- (1) All the software modules issued as part of the PAL have their own User Documentation files that can be accessed via TIA Portal.
- (2) A project developed using the PAL software may require its own documentation (particularly for the application modules that are specific to that particular project). The user can freely add documentation to any of the folders of Table 13.2 to provide documentation for any modules developed for a particular project.
- (3) These documents do not have to be in pdf format, TIA Portal will open any of the following types of document:
 - ① Microsoft Word (.docx)
 - ② Web page (.html or .htm)
 - ③ Portable document format (.pdf)
 - ④ Microsoft PowerPoint (.ppsx or .pptx)
 - ⑤ Rich text format (.rtf)
 - ⑥ Text documents (.txt)
 - ⑦ Microsoft Excel (.xlsx)
 - ⑧ Microsoft XML paper selection (.xps)

Note: The TIA Portal suggests the .chm files (compiled HTML help files) are also supported, this has been found to be incorrect, TIA Portal does not support .chm files and they should not be used as User Documentation

- (4) If multiple file formats exist, TIA Portal will open them in the order listed above, ① first (this is broadly the alphabetical order of the file extensions).

13.2.1 User Documentation for additional items

- (1) The PAL limits its User Documentation to the documentation of each block within it (and some other documents that are accessed via the block User Documentation).
- (2) TIA Portal allows other aspects of a project to be accessible via the User Documentation facilities; the folders of Table 13.2 can be expanded to include other parts of the project tree. The full list is

FOLDER NAME	CONTAINS
Data blocks	Documentation for data blocks (DBs)
Function blocks	Documentation for function blocks (FBs)
Functions	Documentation for functions (FCs)
Organization blocks	Documentation for organisation blocks (OBs)
Projects	Documentation for the project node (top line) within the project tree
Folders	Any folder within the project tree (e.g. Software units or Program blocks)
ShortCut	Any link within the project tree (e.g. Add new block or Add new device)
Library Types	A “type” in the library
Master Copies	Master copies within the master library
Libraries	Individual libraries in the library task card

Table 13.3 Folders for each type of block documentation, full list

- (3) Again, any document file in the additional folders must have the same name as the object it represents e.g.:

`/Projects/projectname.pdf`

`/Folders/Program blocks.pdf`

14

Software security

- (1) The software within a Controller has the facility to be password protected; this is a function of the Controller and the TIA Portal programming environment. By default, the PAL software **will not** be password protected.
- (2) The released, validated modules within the software library will not be password protected in anyway, the purpose of the software is that it is a reusable library and will be deployed on new projects as they arise, the software thus, needs to be accessible.
- (3) The storage of the released (validated) version of the software library in its entirety will be protected on the company servers, the software will be downloadable at the current released version (along with its documentation), this will be a read-only process. The software library will be under change control management (CCM) and only authorised personnel will be able to modify the software.

14.1 The protecting of software modules

- (1) The Validation Plan (VP), *[Ref. 002]* risk assessments require that certain modules and types of modules will have protection activated at certain points during the Project (usually after formal testing). This protection will use the TIA Portal operation referred to as “write protect”, it does not affect the content of the module, it simply prevents it from being changed either intentionally or inadvertently.
- (2) Write protect will be applied to specific modules at specific point in the Project. The final released (and verified) version of the software will have all the write protect restrictions removed.

BLANK PAGE

15

References and glossary

15.1 Document references

The following documents are referenced in this manual:

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)
006	PS2001-5-2211-001	PSP	Hardware Design Specification (HDS)
007	PS2001-5-2311-001	PSP	Software Design Specification (SDS) (THIS DOCUMENT)
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	ISO 8601	ISO	Date and time format
016	PS2001-5-2319-901	PSP	Dot matrix generator
017	PS2001-5-2301-001	PSP	Register of software modules and revisions
018	PS2001-5-2302-011	PSP	Software Control Mechanism (SCM)
019	PS2001-5-234101-001	PSP	ES/WDP Configuration Manual

Table 15.1 Table of references

15.2 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)
CAD	Computer Aided Design
CFR	Code of Federal Regulations
CPU	Central Processing Unit
CSS	Cascading Style Sheet
DC	Direct Current
DB	Data Block
DI	Digital Input
DNS	Domain Name System
DOL	Direct Online
DQ	Digital Output
DS	Design Specification (general reference to any design document)
DTL	Date Time Long
EEMUA	Engineering Equipment and Materials Users' Association
EoC	End of Cycle
EN	European Standards
ERP	Enterprise Resource Planning
ES	Engineering Station
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

ABBREVIATION	DESCRIPTIONS
GRAF CET	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
ID	Instance data block or Identifier
iDB	Instance Data Block
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
IT	Information Technology
JavaScript	A web-based scripting language
jQuery	A library of JavaScript objects, commonly used in web development
LAD	Ladder Logic (PLC programming language)
Ladder	Ladder Logic (PLC programming language)
LTSB	Long-Term Service Branch
MDF	Medium-density Fibreboard
MIT	Massachusetts Institute of Technology (Licence)
MRPII	Management Resource Planning 2
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
PC	Personal Computer
PDF	Portable Document Format
PDT	PLC Data Type

ABBREVIATION	DESCRIPTIONS
PG	Programmer (or programming device, see ES)
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
PN/IE	Profinet/Industrial Ethernet
ProfiBus	Process Field Buss
Profinet	Process Field Net
PSP	Practical Series of Publications
QHD	Quad High Definition
QMS	Quality Management System
QP	Quality Plan
RAL	Colour standards (Reichs-Ausschuß für Lieferbedingungen und Gütesicherung)
RAM	Random Access Memory
RoC	Rate of Change
RTD	Resistance Temperature Device
RT	Run Time
RTM	Requirements Traceability Matrix
SCADA	Supervisory Control and Data Acquisition
SCM	Software Control Mechanism
SDS	Software Design Specification
SDT	System Data Type
SG	Style Guide
SIT	Software Integration Test document
SMDS	Software Module Design Specification
SMT	Software Module Test document
SoC	Start of Cycle
SQL	Structural Query Language
SSD	Solid State Drive
STL	Statement List (PLC programming language)

ABBREVIATION	DESCRIPTIONS
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)
UK	United Kingdom
URS	User requirements specification
US	United States of America
USB	Universal Serial Bus
UT	User Data Type (alternative abbreviation)
VAC	Voltage (alternating current)
VDC	Voltage (direct current)
VP	Validation Plan
VSD	Variable Speed Drive
WinCC	A Siemens Simatic SCADA system

Table 15.2 Glossary