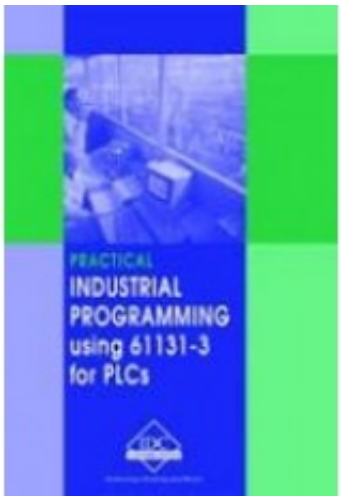
IDC Technologies - Books - 1031 Wellington Street West Perth WA 6005

Phone: +61 8 9321 1702 - Email: books@idc-online.com

PR-E - Industrial Programming using 61131-3 for PLCs



Price: $139.94

Ex Tax: $127.22

Short Description

PLCs have become part of the backbone of industrial automation. The International Electro-technical Commission (IEC) has developed a standard set of programming languages for industrial PLCs. The success of these languages can be measured by the large number of major PLC manufacturers who are developing products that are 61131-3 compliant.

Description

PLCs have become part of the backbone of industrial automation. The International Electro-technical Commission (IEC) has developed a standard set of programming languages for industrial PLCs. The success of these languages can be measured by the large number of major PLC manufacturers who are developing products that are 61131-3 compliant.

IEC 61131-3 is becoming the standard of choice in many industries, and will boost productivity and enhance software quality. If you master the subject today your programming knowledge will be applicable across brands well into the future. This knowledge is vital for personal career development. The aim of this manual is to go beyond the basic concepts and introduce you to the practical

techniques and applications of 61131-3. It cuts across apparent differences wherever PLCs are used and introduces standards that are widely applicable.

If you ever need to program PLCs or just understand more about their capabilities, then this manual is for you. It is pitched at an intermediate level suitable for anyone with some experience with PLCs. If you are a trainee engineer, graduate, control systems engineer, technician, or senior operator you will gain essential knowledge that will significantly enhance existing knowledge of PLCs.

Table of Contents
Download Chapter List

Table of Contents

First Chapter
An introduction to Practical Industrial Programming using 61131-3 for PLCs

1 An introduction to IEC standard 1131 part-3 on PLC programming

This chapter contains information on the use and growth of programmable controllers in industry, the basic problems in the earlier approach adopted for programming these devices and the move towards development of standards for programming. It also discusses the contribution of the standard in improvement of software quality, productivity and maintainability.

Objectives

On completing the study of this chapter, you will learn:

- The basics of Programmable controllers and their role in modern industry
- The need for standardization of PLC languages
- A review of the programming approach prevailing before the evolution of the standard and its shortcomings
- The features of IEC 1131-3 and its contribution towards qualitative improvements to control software
- Move towards open vendor independent systems and software portability

Note

Before we go further, we will get a few basic aspects clear in our minds.
The **International Electro-technical Commission** (IEC for short) is the Geneva based international standards making body, which formulates standards for electrical and electronic equipment. These are adopted both within Europe and in most other industrial nations of the world and integrated into their national standards (incorporating regional variations where required). IEC 1131 is the standard relating to programmable controllers. Part 3 of this standard deals with the languages used for programming these devices and is commonly referred as IEC-1131-3. Even though IEC has renumbered its standards since 1997 by prefixing the numeral 6, we will refer to it by the earlier designation of 1131-3, which is still widely used in the industry rather than 61131-3.

The standard IEC-1131 is organized as follows:

Part	**Title**
1	General information
2	Equipment requirements and tests
3	Programmable languages
4	User guidelines
5	Messaging service specification
6	Communication via fieldbus
7	Fuzzy control programming
8	Guidelines for implementation of languages for programmable controllers

The standard uses the acronym of PC while referring to Programmable Controllers, but in deference to the common use of this abbreviation for the Personal Computer, we will use the term **PLC** (Programmable Logic Controllers) in this manual instead of PC. This is in spite of the fact that the scope of present day programmable controllers extends beyond the conventional interlocking function and includes highly complex control requirements.

1.1 Development and growth of Programmable Controllers (PLC)-An introduction

The PLC is now an indispensable part of any industrial control system. Originally developed in the late 60’s to serve the automation needs of the automobile industry in USA, PLCs have grown much beyond this sector and today it is difficult to name an industry segment that does not use a PLC. The initial purpose was to replace hardwired relay based interlocking circuits by a more flexible device. The flexibility came through the programmability of the device, which made it possible to use the same basic hardware for any application as well as the ability to quickly change the program and modify the behavior of a circuit.

This obviously, is not possible with a hard-wired relay logic circuit.

Thus, the original PLC had:

- Inputs in the form of contacts (called as digital inputs)
- A processor
- A software to control the processor
- Outputs in the form of contacts, referred as a digital outputs (or sometimes as voltages to drive external relays)

The **Inputs** and **Outputs** (called as I/O) were grouped in printed circuit boards, usually plug-in type modules each containing circuits for several inputs or outputs. Such modules grouped together in rack formation along with the Processor module, the power supply module etc. form the hardware of the PLC. Large PLC configurations usually contain several additional racks containing I/O cards daisy chained with the main PLC. More complex systems have redundant power supply modules and additional processors to increase the processing power or to execute multiple tasks simultaneously.

The PLC market thus comprises various sizes of configurations:

- Micro PLC’s of up to 100 I/O’s
- Small PLC’s of between 100 and 200 I/O’s.
- Medium PLC’s of up to a 1000 I/O’s.
- Large PLC’s of more than 1000 I/O’s

PLC’s are now extensively used in many industrial sectors including petrochemicals and food processing and are largely replacing conventional devices in almost all fields of activity.

As the use of PLCs grew, they became more versatile and started including the capabilities of 3 term PID controllers with analog inputs and outputs in addition to the combinational logic systems of hardwired circuits. The analog input and output signals usually follow the 4 to 20 mA signal standard, also developed in the 60’s and which have become the de-facto standard of the instrumentation industry.

Also, as the equipment, which the PLCs served to control became complex, with several of them (each served by its own PLC) acting in tandem, it became imperative to connect them together and share the information between them.

Communication links thus came to be a part of the modern PLC system. Figure 1.1 shows a typical PLC system incorporating several of the features cited above.

Figure 1.1

A typical PLC system

1.2 Need for standardization in programming approach

The software used in the early PLCs was of the ladder diagram type, which closely resembles the circuit diagram with which all electrical engineers are familiar and still remains one of the most popular PLC programming languages. (We will see more about this language later in this chapter). The modern PLC system has however grown far beyond its initial capabilities as a programmable logic controller and needed more versatile tools for programming. The simple ladder diagram method of programming was unequal to this task and had to be supplemented. The resulting multiplicity of languages and sometimes dialects of a basic language became too complex for users to handle, with each vendor's product requiring use of their proprietary programming language/tools. Interoperability of PLC's of different vendors also caused problems of achieving integrated control.

The development of **MAP** (Manufacturing Automation Protocol) by General Motors was an initiative to enable communication between the PLC's of diverse manufacturers. More than the standardization of programming languages, the MAP initiative's main objective was communication between PLC's. The MAP standard could achieve this objective but at a very high hardware cost and still had performance limitations.

PLC manufacturers realized that the future growth of PLC and their widespread use in industry would not be possible unless the fundamental issue of program portability is addressed. Thus started a move for a uniform programming approach to be adopted by all the vendors through a basic programming standard. While certain additional capabilities or extensions could be built-in to their product by different vendors, the basic features were to be uniform thus ensuring portability of code and interoperability. IEC-1131-3 is a result of this effort and has been evolved on a consensual basis by largely adopting the prevalent programming practices of major manufacturers in the PLC industry.

Another standardization initiative is by the **Instrument Society of America** (ISA)

whose Fieldbus is an attempt to facilitate interconnection of devices distributed in the field such as pressure transmitters, temperature controllers, valve positioning systems etc. Though some of the issues of the structure of internal software of these devices are addressed in the fieldbus standard, the standard does not cover languages used for their programming.

1.3 Drawbacks in conventional programming methodology

As discussed in the previous section, most PLCs use some form of ladder Diagram based programming a typical example of which is given in figure 1.2 below.

Figure 1.2

A typical ladder diagram

Note how similar this diagram is to the conventional circuit diagram and how easy it is to follow its action. The Diagram describes the logic used for starting a motor direct on line. START and STOP are command inputs received from a control station. FAULT is a signal from protection scheme of the motor. When START is ON it causes the output COIL to pick up. The START and STOP commands are of momentary (pulsed) type and the status of this output is used to lock the output in the status ‘true’ till STOP or FAULT inputs change status. In addition to being simple, many PLCs also give a dynamic display that shows the status of all these inputs and outputs in real time during the running of the system. Any malfunction both in the external signals or in the program itself is thus easily identified and corrected.

However, this programming approach has a number of limitations in the context of the modern PLC. We will discuss the areas where conventional programming approach proves inadequate in some detail below.

The limitations are:

- Lack of software structure
- Problems in reusability
- Poor data structure definition
- Lack of support for describing sequential behavior
- Difficulty in handling complex arithmetical operations

Software structure

One of the main programming requirements when dealing with complex control systems is to be able to break down the task into a number of smaller, less complex problems and establish clear interconnections to one another. These individual pieces of code are called program blocks or program units. Since these program units may be coded by different programmers and used in different parts of a control system by others, care should be taken to ensure that internal registers and memory locations of a subroutine do not get changed inadvertently by another program block as a result of faulty code. This needs the data to be properly encapsulated or hidden, which is not possible with the ladder programming approach. This makes use of this program technique difficult for complex tasks as any errors can result in catastrophic behavior of the control system.

Software reuse

In many control problems, one finds that certain logic gets repeated in a number of places. With the ladder diagram programs it is necessary to duplicate the same circuit over and over again. This makes the memory usage and the program execution inefficient. For example, the basic motor starting scheme cited in figure 1.2 may get repeated for a number of motors in a system. Arranging the logic sequence of this control in a block, which can be invoked many times (with minor variations and changes in the input/output designations) would simplify the program greatly. Facility for such code reuse is usually limited in conventional ladder diagram programs.

Data structure

In the conventional approach to programming, digital data (both input and output) are represented as single bits. Analog data is kept in the form of registers, which are generally 16 bits long. In this approach, there is no facility to represent related data in a group in the form of a predetermined structure.

Modern programs approach control problems using object orientation. For example, pressure sensors may be represented as a class of objects with each different sensor being an instance of this class. A pressure sensor may have certain data associated with it. These can typically be: the current value of pressure, a set point for the pressure, a time value for setting an output flag when the set value of pressure is reached, a digital alarm output etc. It is possible to carry out a set of logical operations (such as generating an alarm in the event of a set value of pressure being exceeded beyond a set interval of time) by using

the values in the associated data structure. It is possible to use this data block for different pressure sensors (each an instance of the class) by changing the data object's contents. To be able to do this, the data values associated with each sensor must have a unique name but all of them will have a common data definition. In PLC programming terminology, this data class is called as a Data Structure with each instance of the sensor being represented by a variable.

Figure 1.3

Data structure for pressure sensors

Normally, in the conventional ladder programming methodology, the different pieces of data described above are spread throughout program (and the PLC memory) and because of this, data violations can occur easily. The lack of facility for defining a data structure in the conventional ladder programming method will therefore impose constraints in implementing object oriented control strategies.

Support for sequential operations

Many industrial controls perform operations as per a set sequence, particularly those relating to automate operation of processes or machinery. Such operations involve:

- An initial step
- A transition with an associated condition; On fulfilling the condition, deactivate the previous step and go to the next step
- The subsequent step where a set of operations will be performed
- The next transition followed by the next step and so on till the end of the sequence

Representing a sequence of this nature logically in a ladder diagram is somewhat cumbersome. We will illustrate this by an example. A typical chemical batch process for a reactor vessel works in the fashion described below. (Refer to figure 1.4).

Figure 1.4

A typical chemical process

The process sequence goes like this:

1. Start of process
2. Check readiness of all systems. If ready, go forward
3. Open valve for reagent A
4. Measure volume of flow of A and compare against set value
5. Close A when the set value is reached and open valve for reagent B
6. Measure volume of flow of B and compare with set value
7. Close B when the set value is reached. Start stirring. Start timer
8. Check for time to reach a predetermined set value
9. Stop stirring when the set value is reached. Reset timer. Open Drain valve
10. Check flow and compare to sum of volumes of A and B
11. Close drain when total flow is equal to the sum. Reset all flow sensors. Go back to step 1

We may introduce further complexities in this process by incorporating additional parallel steps. For example, it may be required that a certain temperature needs to be maintained during step 7. To do this we may introduce a sequence for monitoring the temperature and switching an electrical heater on and off at certain temperature limits.

To represent the sequence of steps 1 to 11 logically in a ladder diagram, we may need to arrange the ladder rungs in different groups. One group consists of the transition checks (represented in s. no. 2, 4, 6, 8 and 10). The next group consists of the transition states to signify which step is currently active. (In the above example the steps are 1, 3, 5, 7, 9, and 11 with one of them being active at any point of time.) The third is the step actions, i.e., perform certain predetermined tasks at each step as dictated by the process. Based on this approach, the above example can be represented by the following ladder diagram in figure 1.5.

While this looks fairly straight forward, the complexities of the nature cited in the above example, with additional parallel action sequences, will make the process more difficult to represent and understand if we have to extend the above ladder diagram program. (We would invite the readers who may be familiar with creating ladder logic circuits to try doing this to have a ‘feel’ of the programming limitations this method imposes).

Figure 1.5

Ladder diagram of a typical batch process

Execution control

Execution control is another aspect of process control systems, which calls for more sophisticated programming methodologies. To understand this, we have to first look the sequence of operation of a PLC system. Refer figure 1.6 below.

Figure 1.6

PLC operation sequence

PLCs execute a program in a cyclic order. To begin with, all input values are read via the I/O modules and the status stored in PLC memory. Next the ladder rungs are executed starting from the top and proceeding downwards till all rungs are covered. The output values are stored in the PLC memory as each rung is executed. Finally all the output values held in the memory are written to the physical outputs in a single operation. The time of executing the entire program thus depends on the length of the program and the complexity of the logic. Generally, longer the program, higher is the time for each pass of execution (scan cycle time). This makes the behavior of such a system non-deterministic. That is, there is no guarantee that when there is a change of status of a particular input, the actions to be taken as a result will be performed within a stipulated time. (Remember, we are talking of milliseconds here).

While the delay may be an acceptable in many cases, there may be situations where a non-deterministic behavior can cause problems. For example, in a process, a particular condition may call for extremely fast sequence of corrective actions without which catastrophic failures may occur. To delay such corrective action for a whole scan cycle time (the maximum possible delay) may not be an acceptable option. If it is required that the action is initiated within 100 milliseconds and the PLC has a scan rate of 1000 milliseconds, the objective is NOT going to be achieved. The only way this can be done is to divide the program into different sections and arrange the execution in such a way that critical sections are scanned at faster intervals. This kind of control is generally difficult to implement in the ladder diagram approach.

Similarly, implementing PID control functions in PLCs for processes requiring fast control behavior will need more sophisticated programming approach. The non-deterministic nature of the simple ladder diagram program can give rise to

problems of control and a change in the program due to, say the addition of a few rungs of interlock conditions, may influence the way the system performs its control function.

Arithmetic operations

It is possible that certain logic steps in a control system may require arithmetic operations to be performed. For example, in the control system described in the earlier section, step no. 10 involves an addition of two set values to decide when the transition to the next step should occur. Ladder diagram programming in most implementations can perform such simple arithmetical operations. However more complex arithmetical or trigonometric expressions will pose difficulties and will have to be tackled by workarounds.

1.4 Features of IEC-1131-3 language definition

IEC 1131-3 was based on a review of the languages that were already in use with the PLCs of market leading brands. A new set of programming languages were then defined based on the language variants available, their shortcomings (as explained in the previous section) and the need for enhancements and capabilities to meet the new demands on PLCs. In doing so, the possibilities of new PLC capabilities that may be developed in the near term future were also kept in view. The major new features offered by the new set of PLC languages are as follows.

Ability to break down a large complex task into many **Program Organisational Units** (POU) each performing a part of the main task. This includes functions, function blocks and programs. A **hierarchical program structure** has thus been established. Both top-down and bottom-up development approach is now possible using the software architecture defined in the standard.

Software reuse has been made possible by the use of functions and function blocks. Standard functions are available for mathematical and logical operations and the need for coding by individual programmers is avoided. Repeatable tasks of complex nature can be coded in the form of Function blocks, which can be called wherever required in a program. There is now a very real possibility of third party software modules being developed for standard control applications and made available to the developer community on a commercial basis.

Definition of **data types** with error control is possible. The variables used are categorised into various data types and if a program has mismatches of data type, it will be shown up by the program station (or the language compiler) before

loading it into the PLC.

Full **execution control** is possible. Different parts of a program, for example, may be scanned at different rates, thus enabling faster scan times for fast changing and critical parameters and a slow scan time for variables that are not likely to change quickly (e.g. temperature of the reactor vessel in the example in figure 1.4).

Ability to define process control sequences has been made possible using **Sequential Function Charts** (one of the languages defined by the standard). The sequence may have paths, which operate in parallel as well as alternate branches depending on specific process conditions.

Definition of data structure enables an entire set of data to be defined as a single data object (usually representing a particular class of object). As seen in the previous section, a pressure sensor may be associated to a particular data structure, which defines all the variables that are possibly required for this class of object in the process control context.

Flexible use of language is permitted by the standard. Different parts of a control problem may be expressed using any of the languages defined in the standard based on the needs of the control and its suitability to the task in question.

Language standard and methods of programming recommended by the standard help in solving problems of all possible PLC applications as well as ensure a degree of portability of the control programs between equipment of different vendors who comply with the standard.

In order to achieve these features IEC 1131-3 has defined a set of five languages, two of them textual and the others graphical. We will describe these in the ensuing modules in detail. However a brief note on each of these languages here will be appropriate.

Structured text is a language of textual type, which resembles the high level Pascal language and can be used for expressing various common control requirements.

Function block diagram is a graphical language, which represents signal and data flow through interconnected function blocks and functions. Functions and function blocks are both reusable software elements.

Ladder diagram is a graphical language as we saw earlier and is based on relay logic diagrams used to represent process interlocks. The standard in addition, permits use of functions and function blocks within a ladder diagram.

Instruction list is an assembler-like textual language, which manipulates the internal registers of a PLC and reads or writes values from/into variables and memory locations.

Sequential function chart is a graphical language used for representing complex sequential behaviour of processes as shown in the example in figure 1.4.

Many implementations offer facilities for conversion of a program module written using one of the above IEC languages into another and are said to have the feature of **inter-language portability**. This facilitates code review and maintenance by permitting the code to be viewed in any language with which a reviewer or maintenance programmer is most familiar. There is however some limitations to this method as certain types of operations specific to a language may not easily translate into a different language directly.

Inter-system portability is another issue that has been receiving a lot of attention. There are a few constraints to be overcome before a control program written for a certain PLC vendor's implementation is fully portable to another vendor's. IEC standard defines a broad set of language features but does not insist that all of them must be supported by an implementation. If a particular feature is not supported by a target implementation but is used in the original program, porting can pose problems. Similarly, certain features are implementation dependent, (e.g., maximum number of array subscripts) which can come in the way of successful portability. These issues arising out of weak compliance requirements adopted by the standard have been sought to be addressed by agencies such as **PLC Open**. Conformity to the compliance level requirements of these bodies can ensure inter-system portability.

And finally, the matter of **encapsulation of data and procedure**: in order that software integrity is not compromised, it is necessary to hide well-tested code modules (such as functions) and if possible, the data that they contain, so that they are not directly 'visible' to a programmer. This is because errors can inadvertently be introduced in the program in the course of working on it. Use of functions, function blocks, structures etc. stipulated in the standard effectively encapsulates data and code to varying degrees and preserves the integrity of the system.

1.5 Summary

IEC-1131-3 standard tackles many of the limitations imposed by the earlier generation of PLC languages. It provides a variety of languages suitable for different control tasks and provides both inter-language and inter-system portability. Reuse of software is possible though functions and function blocks. Integrity of code and data is ensured by encapsulation. Hierarchical software design approach makes development and maintenance of applications more structured and enhances both software quality and productivity.