

Technical Talk

By Mark Chisnall

International Standards for Industrial Control Programming

This article is targeted towards the more technically aware but it also has an underlining message to anyone commissioning new PLC work - to insist on the use of International standard IEC 61131-3.

Until recently, the manufacturers of programmable logic controllers (PLCs) developed their own programming languages. This tended to lock users to a particular manufacturer because of existing software, maintenance and training etc. Now, however, IEC 61131-3 provides the open standard for programming of a wide range of industrial control systems and is universally accepted by the majority of PLC systems.

It is important to realise that IEC 61131-3 is not just another programming language, it supports modern 'software engineering' methods leading to improved programs, easier maintenance and fault finding, reusable and transportable code. It is in everyone's interest to insist on the International standard for PLC programming.

What is IEC 61131-3?

IEC 61131-3 incorporates five different approaches to programming control systems; each has a simple language aimed at a particular type of problem or application area. The languages can be easily combined to produce good-quality readable code. These simplify and standardise the programming across different manufacturers and different types of control system (programmable drives, intelligent sensors, SCADA packages, process controllers etc). The main objective of IEC 61131-3 has been to standardise existing PLC languages and it is worthwhile pointing out there is no intention that the standard should reduce the development of new PLC languages. Any PLC vendor is free to provide extensions and additional languages where required and because the standard allows proprietary function blocks to be programmed in non IEC 61131-3 languages such as C++, it is always possible to provide extensions fairly 'seamlessly' e.g. packaged as function blocks.

Ladder Logic

To understand the difference between this type of PLC programming and the more conventional ladder programming we need to go through some basics of the traditional PLC ladder logic programming.

Ladder logic is a method of drawing electrical logic schematics. It was originally invented to describe logic made from relays. The name is based on the observation that programs in this language resemble ladders, with two vertical "rails" and a series of "rungs" between them.

A program in ladder logic, also called a ladder diagram, is similar to a schematic for a set of relay circuits. Ladder logic is useful because a wide variety of engineers and technicians can understand and use it without much additional training because of the resemblance.

Ladder logic is widely used to program PLCs, where sequential control of a process or manufacturing operation is required. Ladder logic is useful for simple but critical control systems, or for reworking old hardwired relay circuits. As programmable logic controllers became more sophisticated it has also been used in very complex automation systems.

Most manufacturers of programmable logic controllers also provide associated ladder logic programming systems. Typically, the ladder logic languages from two manufacturers will not be completely compatible; ladder logic is better thought of as a set of closely related programming languages rather than one language. Even different models of programmable controller within the same family may have different ladder notation such that programs cannot be seamlessly interchanged between models.

Example of a simple ladder logic program

The language itself can be seen as a set of connections between logical checkers (relay contacts) and actuators (coils). If a path can be traced between the left side of the rung and the output, through asserted (true or "closed") contacts, the rung is true and the output coil storage bit is asserted (1) or true. If no path can be traced, then the output is false (0) and the "coil" by analogy to electromechanical relays is considered "de-energized".

Ladder logic has "contacts" that "make" or "break" "circuits" to control "coils." Each coil or contact corresponds to the status of a single bit in the programmable controller's memory. Unlike electromechanical relays, a ladder program can refer any number of times to the status of a single bit,

equivalent to a relay with an indefinitely large number of contacts.

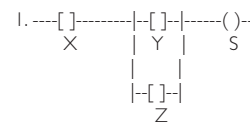
So-called "contacts" may refer to inputs to the programmable controller from physical devices such as pushbuttons and limit switches, or may represent the status of internal storage bits or of output bits generated elsewhere in the program.

Each rung of ladder language typically has one coil at the far right. Some manufacturers may allow more than one output coil on a rung.

--()-- a regular coil, true when its rung is true
--(/)-- a "not" coil, false when its rung is true

The "coil" (output of a rung) may represent a physical output which operates some device connected to the programmable controller; or may represent an internal storage bit for use elsewhere in the program.

Here is an example of what one rung in a ladder logic program might look like. In real life, there may be hundreds or thousands of rungs.



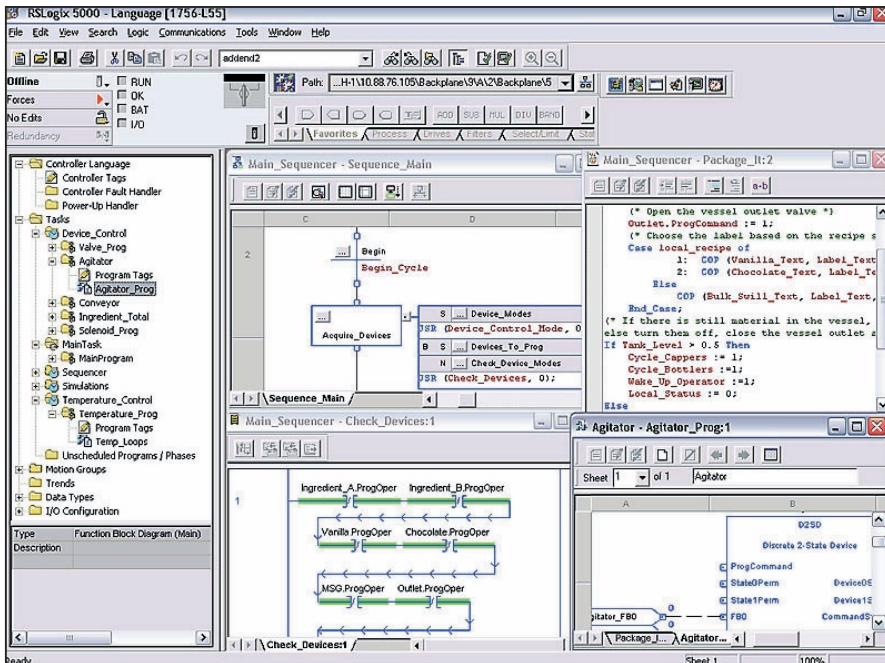
Limitations and successor languages

Ladder notation is best suited to control problems where only binary variables are required and where interlocking and sequencing of binary is the primary control problem. Since execution of rungs is sequential within a program and may be undefined or obscure within a rung, some logic "race" conditions are possible which may produce unexpected results; complex rungs are best broken into several simpler steps to avoid this problem. Analogue quantities and arithmetical operations are clumsy to express in ladder logic and each manufacturer has different ways of extending the notation for these problems.

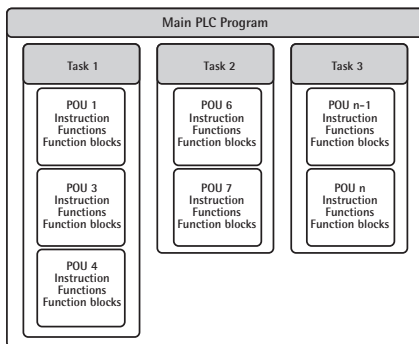
PLC Programming

The IEC 61131-3 standard has a structured programming approach, which replaces the former unwieldy collection of individual instructions with a

TECHNICAL TALK



clear arrangement of the program into individual program modules. These modules are referred to as Program Organisation Units (POUs), which form the basis of this new approach to programming.



The actual PLC program code contained in the program organisation units (POUs) and the steps and transitions of an SFC can be written in any one of the following programming languages,

- **The Text Editors:**
Instruction List (IL)
Structured Text (ST)
- **The Graphical Editors:**
Ladder Diagram (LD)
Function Block Diagram (FBD)
Sequential Function Chart (SFC)

Instruction List (IL)

It is a low level language and resembles assembler. All of the languages share IEC Common Elements. The variables and function call are defined by the common elements so different languages can be used in the same program.

Sample Programme

```
LD Speed
GT 1000
JMPCN VOLTS_OK
LD Volts
VOLTS_OK LD I
ST %Q75
```

Structured text (ST)

It is a high level language that is block structured and resembles PASCAL. All of the languages share IEC Common Elements. The variables and function call are defined by the common elements so different languages can be used in the same program. Complex statements and nested instructions are supported:

- Iteration loops (REPEAT-UNTIL; WHILE-DO)
- Conditional execution (IF-THEN-ELSE; CASE)
- Functions (SQRT(), SIN())

Sample Programme:

```
(* simple state machine *)
TxtState := STATES[StateMachine];
CASE StateMachine OF
  0: ;; (* FAIL *)
  1: ClosingValve();
ELSE
  ;; BadCase();
END_CASE;
```

Functional Block Diagram (FBD)

A function block diagram describes a function between input variables and output variables. A function is described as a set of elementary blocks. Input and output variables are connected to blocks by connection lines. An output of a block may also be connected to an input of another block.

Inputs and outputs of the blocks are wired together with connection lines, or links. Single lines may be used to connect two logical points of the diagram:

- An input variable and an input of a block
- An output of a block and an input of another block
- An output of a block and an output variable

The connection is oriented, meaning that the line carries associated data from the left end to the right end. The left and right ends of the connection line must be of the same type.

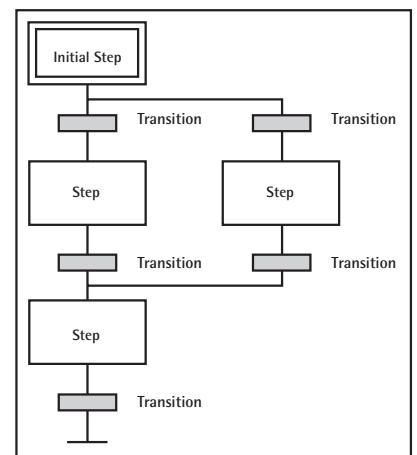
Sequential Function Chart (SFC)

SFC is built up of steps with transition conditions in-between, where each step represents one state. When a step is active it means that the control program is in this state. When the succeeding transition conditions become fulfilled the control program execution continues in the next step. Then the succeeding step gets active and the previous active step becomes inactive. It is also possible to build parallel and alternative step constructions.

The main components of SFC are:

- Steps with associated actions
- Transitions with associated logic condition
- Directed links between steps and transitions

Sample Programme



Conclusion

Developing application programs using IEC 61131-3 offers the following advantages:

- Reduces training costs by learning one set of programming languages used by multiple control vendors.
- Provides flexibility for selecting the best programming approach and methods for specific application tasks and requirements.
- Offers the ability for the programmer to develop and deploy reusable function blocks which can reduce future software development costs and protect your company's intellectual property.
- Reduced misunderstanding and errors
- Combining different components from different programs, projects, locations, companies and countries.

Mark Chisnall is Managing Director of Suffolk Automation, a company specialising in the design, development and installation of process control systems.

Please contact Mark if you have any questions on plant Automation or if you require a specific subject to be considered for future publication.

e: mark@suffolk-automation.co.uk
t: +44(0)1473 829188