Introduction into IEC 1131-3 and PLCopen

Eelco van der Wal
Managing Director PLCopen
PO Box 2015
NL 5300 CA Zaltbommel, The Netherlands

ABSTRACT

Modern programming methods provide tools to improve the intrinsic quality of software, i.e. its correctness in the sense of reliability, robustness, integrity, persistence, and safety. The international standard IEC 1131-3 provides such a tool, dealing with the programming, installation and maintenance phases of software development projects in industrial control.

Basically, IEC 1131-3 consists of two parts, i.e. Common Elements and Programming Languages. The structuring tools within IEC 1131-3 are focused on the common elements, although clearly links to the programming languages are needed.

This article shows that by using IEC 1131-3 in a consistent way, one can generate software code that is better understandable, reusable, verifiable and maintainable.

INTRODUCTION

IEC 1131-3 is the first real endeavor to standardize programming languages for industrial automation. With its nowadays world wide support, it is independent of any single company.

IEC 1131-3 is the third part of the IEC 1131 family. This consists of:

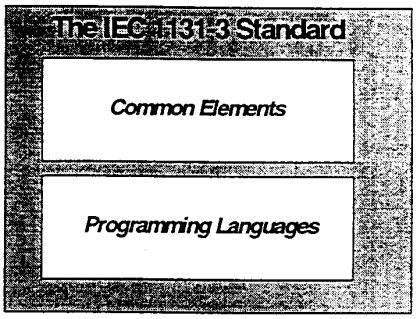
- Part 1: General Overview
- Part 2 Hardware
- Part 3 Porgramming Languages
- Part 4 User Guidelines
- Part 5 Communication

There are many ways to look at part 3 of this standard. Just to name a few:

- the result of the Task Force 3, Programming Languages, within IEC TC65 SC65B
- the result of hard work by 7 international companies adding tens of years of experience in the field of industrial automation
- approx. 200 pages of text, with 60-something tables, including features tables
- the specification of the syntax and semantics of a unified suite of programming languages, including the overall software model and a structuring language.

Another elegant view is by splitting the standard in two parts (see figure):

1. Common Elements
2. Programming Languages



Common Elements

Data Typing

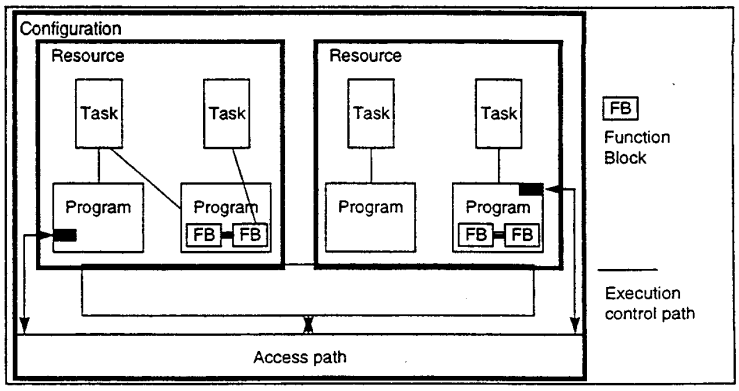
Within the common elements, the data types are defined. Data typing prevents errors in an early stage. It is used to define the type of any variable used. This avoids for instance dividing a Date by an Integer.
Common datatypes are Boolean, Integer, Real and Byte and Word, but also Date, Time-of-Day and String. Based on these, one can define own personal data types, known as derived data types. In this way one can define an analog input channel as data type, and re-use this over and over again.

Variables

Variables are only assigned to explicit hardware addresses (e.g. input and outputs) in configurations, resources or programs (see below). In this way a high level of hardware independency is created, supporting the reusability of the sofware.
The scope of the variables are normally limited to the organization unit in which they are declared, e.g. local. This means that their names can be reused in other parts without any conflict, eleminating another source of errors, e.g. the scratchpad. If the variables should have global scope, they have to be declared as such (VAR_GLOBAL). Variables can be assigned an initial value at start up and cold restart, in order to have the right setting.

Configuration, Resources and Tasks

To understand these better, let us look at the software model, as defined in the standard (see below).



At the highest level, the entire software required to solve a particular control problem can be formulated as a *Configuration*. A configuration is specific to a particular type of control system, including the arrangement of the hardware, i.e. processing resources, memory addresses for I/O channels and system capabilities.
Within a configuration one can define one or more *Resources*. One can look at a resource as a processing facility that is able to execute IEC programs, like a CPU.
Within a resource, one or more *Tasks* can be defined. Tasks control the execution of a set of programs and/or function blocks. These can either be executed periodically or upon the occurrence of a specified trigger, such as the change of a variable.
A *Program* can be built with any of the IEC defined languages. Typically, a program consist of a network of *Functions* and *Function Blocks*, which are able to exchange data. Function and Function Blocks are the basic building blocks, containing a datastructure and an algorithm.

Let's compare this to a conventional PLC: this contains one resource, running one task, controlling one program, running in a closed loop. IEC 1131-3 adds much to this, making it open to the future. A future that includes multi-processing and event driven programs. And this future is even here: just look at distributed systems or especially softPLCs. These latter only could take off based on a standard suitable for a broad range of applications, without having to learn additional programming languages.

Program Organization Units

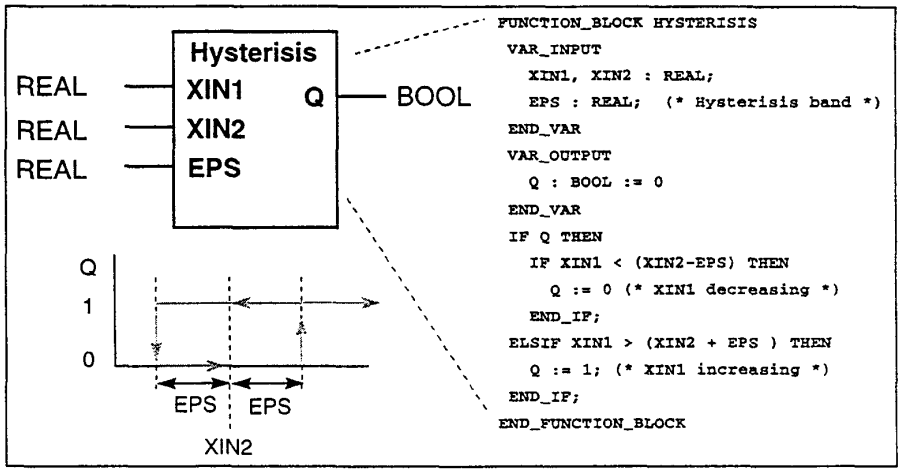
Within IEC 1131-3, the Programs, Function Blocks and Functions are called Program Organization Units, or POUs.

Functions

IEC has defined standard functions and user defined functions. Standard functions are for instance ADD(ition), ABS (absolute value), SQRT, SINe and COSine. User defined functions, once defined, can be used over and over again.

Function Blocks, FBs

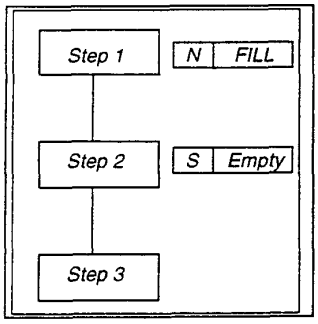
Function Blocks are the equivalent to Integrated Circuits, ICs, representing a specialized control function. They contain data as well as the algorithm, so they can keep track of the past (which is one of the differences with respect to Functions). They have a well defined interface and hidden internals, like an IC or black box. In this way they give a clear separation between different levels of programmers, or maintenance people.
A temperature control loop, or PID, is an excellent example of a Function Block. Once defined, it can be used over and over again, in the same program, different programs, or even different projects. This makes them highly re-usable.
Function Blocks can be written in any of the IEC languages, and in most cases even in "C". It this way they can be defined by the user. Derived Function Blocks are based on the standard defined FBs, but also completely new, customized FBs are possible within the standard: it just provides the framework.



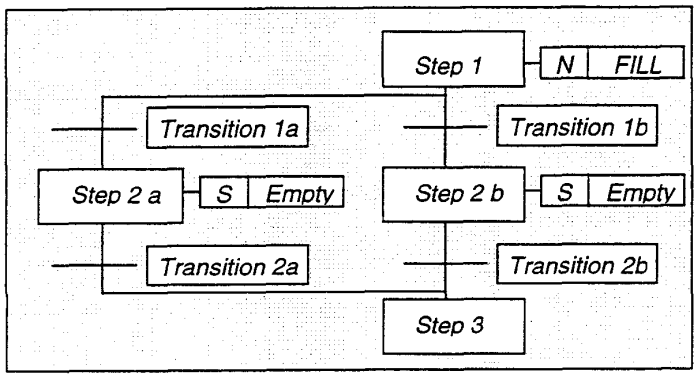
Programs

With the above mentioned basic building blocks, one can say that a program is a network of Functions and Function Blocks. A program can be written in any of the defined programming languages.

Sequential Function Chart, SFC



SFC describes graphically the sequential behaviour of a control program. It is derived from Petri Nets and IEC 848 Grafcet, with the changes necessary to convert the representation from a documentation standard to a set of execution control elements.
SFC structures the internal organization of a program, and helps to decompose a control problem into manageable parts, while maintaining the overview.
It consists of Steps, linked with Action Blocks and Transitions.. Each Step represents a particular state of the systems being controlled. A Transition is associated with a condition, which, when true, causes the Step before the Transition to be deactiviated, and the next Step to be activated. Steps are linked to Action Blocks, performing a certain control action. Each element can be programmed in any of the IEC languages, including SFC itself.
One can use alternative sequences and even parallel sequences, such as commonly required in batch applications. For instance, one sequence is used for the primary process, and the second for monitoring the overall operating constraints.



Because of its general structure, SFC provides also a communication tool, combining people of different backgrounds, departments or countries.

Programming Languages

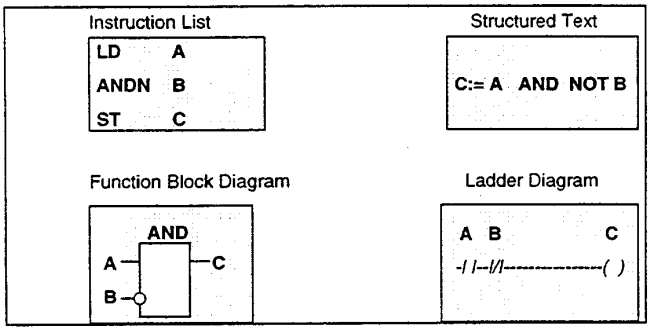
Within the standard four programming languages are defined. This means that their syntax and semantics have been defined, leaving no room for dialects in this part. Once you have learned them, you can use a wide variety of systems based on this standard.
The languages consist of two textual and two graphical versions:

Textual:

- Instruction List, IL
- Structured Text, ST

Graphical:

- Ladder Diagram, LD
- Function Block Diagram, FBD



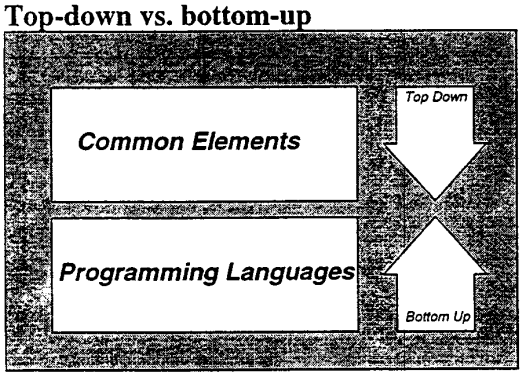
In the figure left, all four languages describe the same simple program part.
The choice of programming language is dependent on:

- the programmers' background
- the problem at hand
- the level of describing the problem
- the strucuture of the control system
- the interface to other people / departments

All four languages are interlinked: they provide a common suite, with a link to existing experience. In this way they also provide a communication tool, combining people of different backgrounds.

Ladder Diagram has its roots in the USA. It is based on the graphical presentation of Relay Ladder Logic.
Instruction List is its European counterpart. As textual language, it resembles assembler.
Function Block Diagram is very common to the process industry. It expresses the behaviour of functions, function blocks and programs as a set of interconnected graphical blocks, like in electronic circuit diagrams. It looks at a system in terms of the flow of signals between processing elements.
Structured Text is a very powerfull language with its roots in Ada, Pascal and "C". It can be used excellently for the definition of complex function blocks, which can be used within any of the other languages.

Top-down vs. bottom-up



Also, the standard allows two ways of developing your program: top down and bottom up. Either you specify your whole application and divide it into sub parts, declare your variables, and so on. Or you start programming your application at the bottom, for instance via derived functions and function blocks. Whichever you choose, the development environment will help you through the whole process.

Implementations

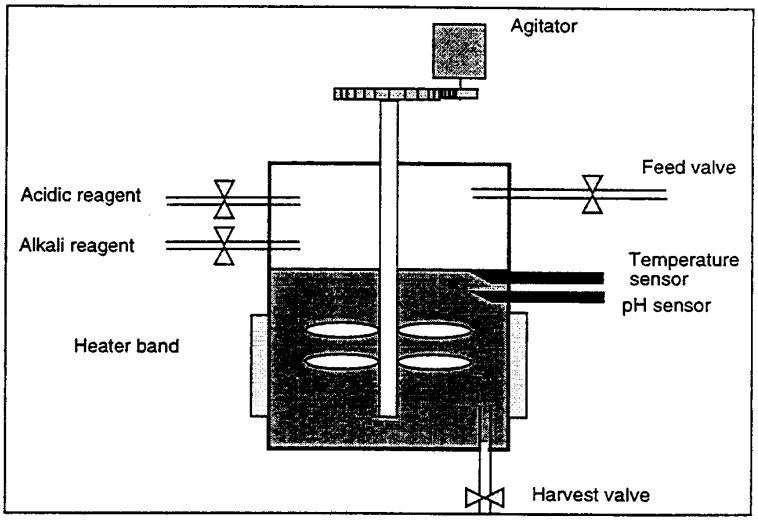
The overal requirements of IEC 1131-3 are not easy to fulfill. For that reason, the standard allows partial implementations in various aspects. This covers the number of supported languages, functions and function blocks. This leaves freedom at the supplier side, but a user should be well aware of it during his selection process. Also, a new release can have a dramatically higher level of implementation.

Many current IEC programming environments offer everything you expect from modern environments: mouse operation, pull down menus, graphical programming screens, support for multiple windows, built in hypertext functions, verification during design. Please be aware that this is not specified within the standard itself: it is one of the parts where suppliers can differentiate.

A PRACTICAL EXAMPLE : Fermentation control system

An example tells more than 1000 words..., so let us look at a fermentation process and its control, as shown below.

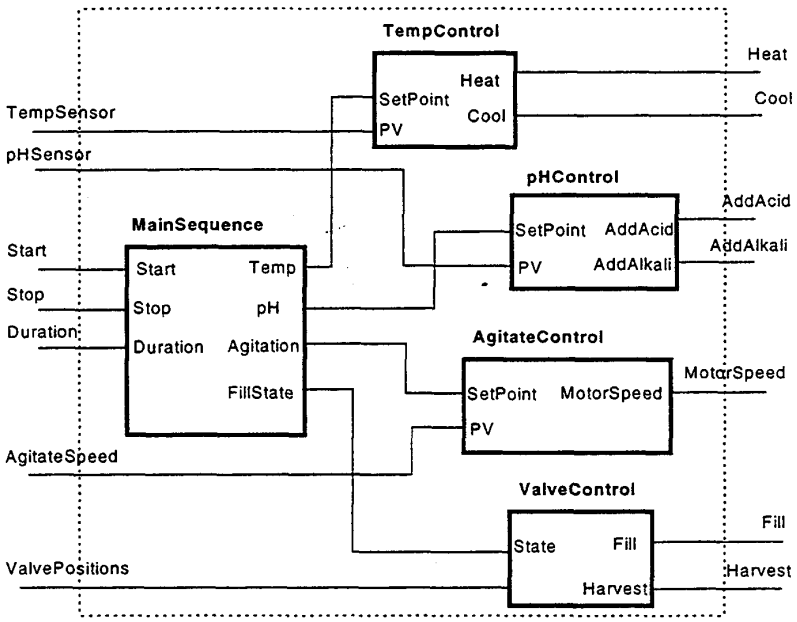
All external interfaces are defined here (step 1). There is a large vessel, which can be filled (Feed Valve) with the liquid, can be heated with the heater band (cooling via convection), can be stirred via the motor, and where acid and alkali fluid can be added into the vessel.



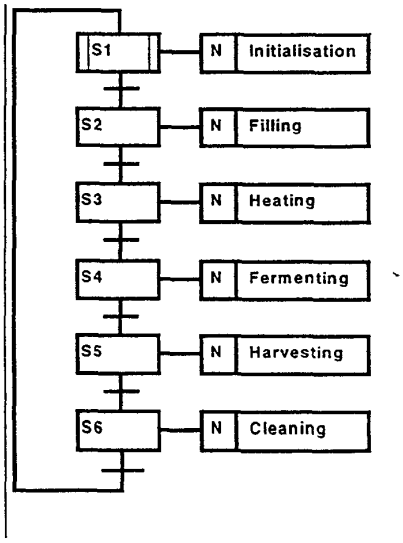
Within this process, one can easily identify 5 basic functions:

1. **Main Sequence,** *e.g., top level process steps - filling, heating, agitating, fermenting, harvesting, cleaning.*
2. **Valve control,** *e.g., operating valves used to fill and empty the vessel*
3. **Temperature control** *for monitoring the temperature of the vessel modulating the heater.*
4. **Agitator control** *for the agitator motor activated as demanded by the main process sequence.*
5. **pH Control** *for monitoring the acidity of the fermentation contents, adding acidic or alkali reagents as required.*

Presenting these in the Function Block Diagram programming language, the overview of the fermentation control program could look like this: (Read from left to right side. On the left are the inputs; on the right the outputs)



If we take a closer look at the Main Sequence, we could structure it with Sequential Function Charts, SFC, as follows:



We start at the top with the Initialization: since we do not know the status of the system when we first switch it on, we must check the position of the valves, etc.
Then we start filling till the right level has been reached.
The next phase is the heating till the fermentation process starts. When it does, we move to the next phase: the actual fermentation process control part.
After completion, we harvest, and after that clean, and we are ready to restart at the top.
This decomposition gives everybody involved a clear overview which sequences are involved, and further modularization into the function blocks which can be programmed in any of the four languages.
Or, stated differently: our user requirement specification is (nearly) done!
The programming work now to be done is at the level of the action blocks. Those could be divided between different people, with different backgrounds. For this, the IEC 1131-3 standard has defined 2 graphical and 2 textual programming languages, as desribed above, to best suit the needs and the problem at hand. Also, further decomposition of the action blocks can be done via SFC, if needed.

Conclusion

The requirements on current control systems have increased. The added functionality translates itself into more people involved, linking between different disciplines, and in the end in more software code.

The software development process has been adopted to fulfill these increased requirements. Structuring, reusable code, modularity and decomposition are essential elements of modern software development processes. This technique provides a method to improve the intrinsic quality of software, i.e. its correctness in the sense of reliability, robustness, intêgrity, persistence, and safety.

The international standard IEC 1131-3 provides such a tool, dealing with the programming, installation and maintenance phases of software development projects in industrial control. The technical implications of the IEC 1131-3 standard are high, leaving enough room for growth and differentiation. This make it suitable to evolve well into the next century.
IEC 1131-3 has a great impact on the whole industrial control industry. It certainly does not restrict itself to the conventional PLC market. Nowadays, one sees it adopted in the motion control market, distributed systems and softlogic / PC based control systems, including SCADA packages. And the areas are still growing.

Having a standard over such a broad application area, brings numerous benefits for users / programmers. The benefits for adopting this standard are various, depending on the application areas. Just to name a few:

- reduced waste of human resources, in training, debugging, maintenance and consultancy
- creating a focus to problem solving
- reduced misunderstanding and errors
- programming techniques usable in a broad environment: general industrial control
- combining different components from different programs, projects, locations, companies and/or countries