Drive & Control profile

# Understanding the IEC61131-3 Programming Languages

It was about 120 years ago when Mark Twain used the phrase "more than one way to skin a cat." In the world of PLC programming, that cliché is still applicable today. Thanks to the International Electrotechnical Commission (IEC), five standard programming languages have emerged as the most common, used for both process and discrete programmable controllers. The IEC is an organization that prepares and publishes international standards for all electrical, electronic and related technologies, including controllers. With its IEC61131-3 publication, the organization identifies these five programming languages and their common abbreviations as: Ladder Diagram (LD), Instruction List (IL), Function Block Diagram (FBD), Structured Text (ST) and Sequential Function Chart (SFC).

Long dismissed as just being a European phenomenon, the IEC's programmable controller languages are gaining traction in the United States. The IEC developed these programming standards in response to the growing number of automation vendors, the growing complexity of applications, and the multiplying methods for implementing control functions. But many controls engineers may be wondering about the characteristics of each programming language. When should one be used over another? What are the benefits and disadvantages of each? This article will provide a brief overview and comparison of each of the five main PLC programming languages.

The complexity of the application, the capabilities of the PLC/PAC and the ability to transfer the program code are among the key factors to consider with selecting controller programming language.

## Choosing Your Language

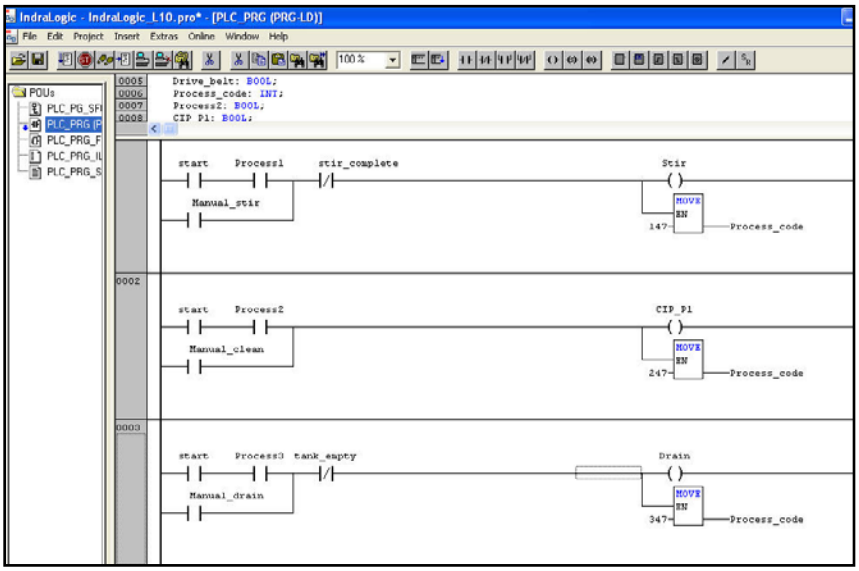
The International Electrotechnical Commission (IEC) identifies five standard programming languages as the most common for both process and discrete programmable controllers: Ladder Diagram (LD), Function Block Diagram (FBD), Sequential Function Chart (SFC), Instruction List (IL), and Structured Text (ST)

With the different programming languages available, it's important to consider a few factors before deciding which to use for your application:

- Ease of maintenance by the final user: SFC
- Universal language acceptance: LD
- Acceptance in Europe: IL or ST
- PLC speed of execution: IL or ST
- Applications mainly using digital I/O and basic processing: LD or FBD
- Ease of changing code: LD
- Ease of use by newer engineers: ST
- Ease of implementing complex mathematical operations: ST
- Applications with repeating processes or processes requiring interlocks and concurrent operations: SFC

### Ladder Diagram (LD)

This programming language, invented in the U.S. decades ago, is probably the most widely used. Invented to replace hardwired relay control systems, Ladder Diagram programming is a mainstay in the U.S. today, used in probably 95 percent of all applications. Visually, this language resembles a series of control circuits, with a series of inputs needing to be "made" or "true" in order to activate one or more outputs.




This language resembles a series of control circuits, with a series of inputs needing to be "made" or "true" in order to activate one or more outputs.

Ladder Diagram language has experienced such widespread adoption that almost every programmer in any country or industry can read and write this language. Because it resembles the familiar electric circuit format, even a non-programmer with an electrical background can follow the program for purposes of troubleshooting a problem. It's also easy to start writing a program in Ladder Diagram. With just a basic outline of input and output signals, one can sit down are start churning out code. Most of the other IEC languages require more preparation, such as flowcharting all the potential process flows. Finally, most implementations of Ladder Diagram allow a program to be organized into folders or subprograms that are downloaded to the PLC, allowing for easy segmentation.

Ladder Diagram programming is ideal for a simple material handling application, for example, where a sensor detects the presence of a box, other sensors check for obstructions, and then an output fires an actuator to push the box to another conveyor. Digital inputs are checking for various conditions, and a basic program is analyzing the inputs and firing digital outputs in response. There may be timers in the program, or some basic comparisons or math, but there are no complex functions involved.

As the complexity of PLC functionality has grown, however, Ladder Diagram language has been challenged to meet these advances and still maintain the paradigm of easy visualization and understanding. Functions such as PID, trigonometry and data analysis are commonly required in many control applications, but difficult to implement. Another challenge is that as program size grows, the ladder can become very difficult to read and interpret, unless it's extensively documented. Finally, implementing full processes in Ladder Diagram can be daunting—picture a ladder rung with an output used in several phases of a process with many input conditions attempting to control exactly when that output needs to turn on.

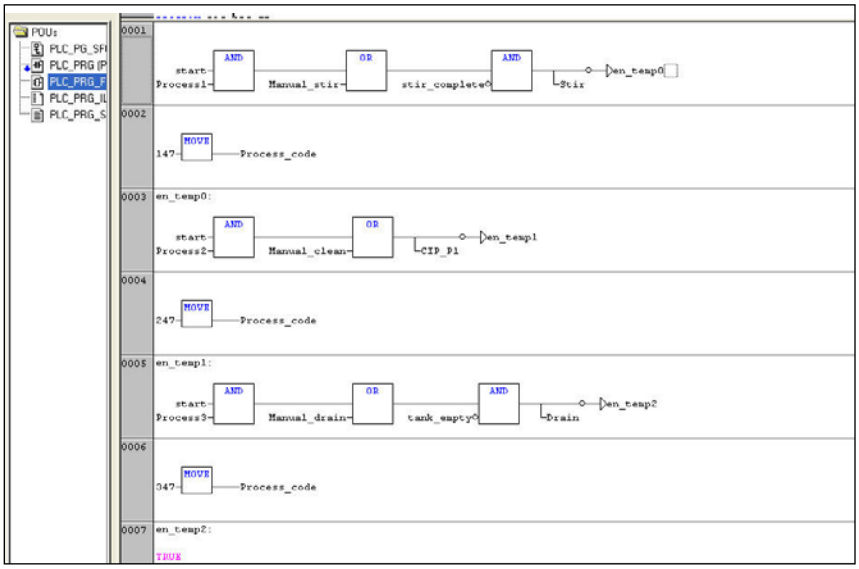
### Function Block Diagram

Although Ladder Diagramming may be the most widespread language, a survey conducted by *Control Engineering* magazine several months ago highlighted growth in the use of programming languages other than ladder. Function Block Diagram programming is an example. Even though the adoption rate for this language has recently slowed relative to other languages such as Structured Text, Function Block Programming is probably the second most widely used language.

In many ways, this graphical language resembles a wiring diagram even more so than

Ladder code. With Function Block Diagram, the blocks are "wired" together into a sequence that's easy to follow. It uses the same instructions as Ladder, but visually is more understandable to a viewer who is not versed in relay logic. The major advantage is that programs written in Function Block tend to be easy to follow—just follow the path! This language is ideal for simpler programs consisting of digital inputs, such as photoelectric sensors, and outputs such as valve manifolds, and could be appropriate in any application where Ladder is ideal.

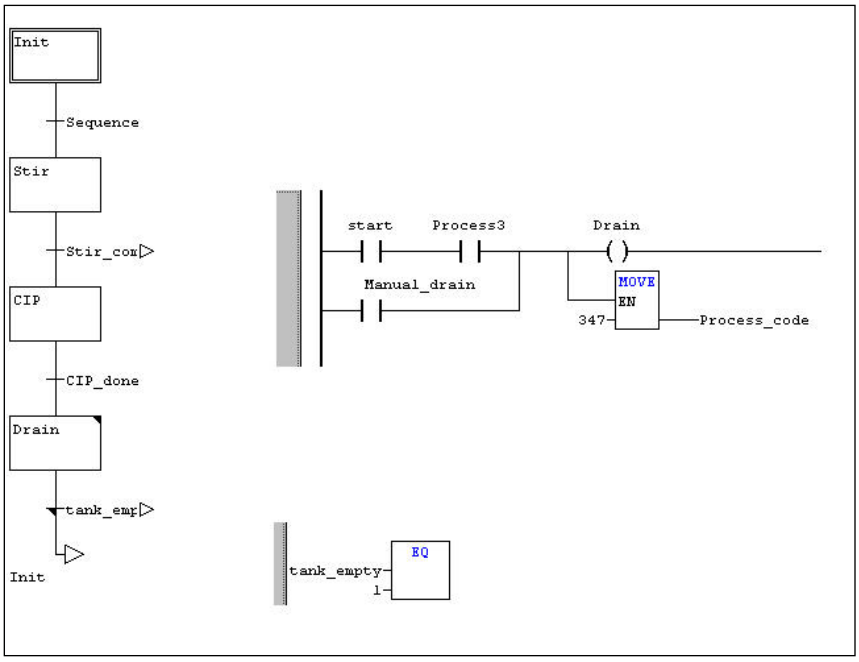
However, this language is not ideal for large programs using special I/O and functions. The large amount of screen space required by this style of programming can quickly make a program unwieldy if it reaches any substantial size. Also, writing a program in Function Block requires more preparation upfront to understand the program and how it will flow before any code is written, since it can be more difficult to make corrections later.




With Function Block Diagram, the blocks are "wired" together into a sequence that's easy to follow. It uses the same instructions as Ladder, but is visually more understandable to a viewer who is not versed in relay logic.

### Sequential Function Chart

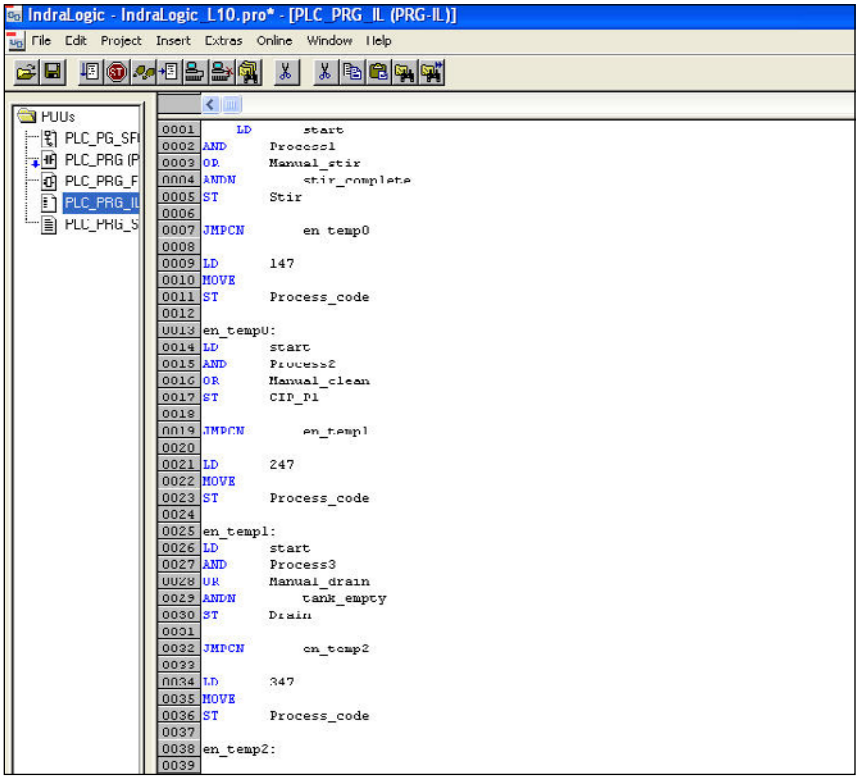
Sequential Function Chart (SFC) programming resembles the computer flowcharts that many will remember drawing up in their college days. An initial step "action box" (the starting point of a flowchart) is followed by a series of transitions and additional action steps. The concept of SFC is simple: an action box, with code inside written in any language of the programmer's choice, is active until the transition step below it activates. The current action box is turned off, and the next one in the sequence is active. The transition step also has code to check that the necessary conditions are met to allow the program to advance to the next step.




In this example, SFC programming includes a flowchart on one side and two small programs to the right. In an SFC program, the flowchart boxes (called actions) and the little horizontal lines with names (called transitions) actually have small programs running inside them.

For appropriate applications which have a repeatable multi-step process or series of repeatable processes, this form of programming is the easiest to implement. An example would be a pick and place application, where product is constantly picked up from one area, moved through a specific path, and placed in another area. While exceptions exist, since there is typically only one active piece of code and one transition to be concerned with, condition checking and the control of the process should be achievable without large rungs. The language is also very friendly to maintenance engineers because the visual nature of the program plus code segmentation makes it easy to troubleshoot. For example, if the mechanism in a pick-and-place application has moved to the product but not picked it, the troubleshooter could bring up the program and look at the transition condition between the "move to product" box and the "pick product" box to see what is holding up the process.

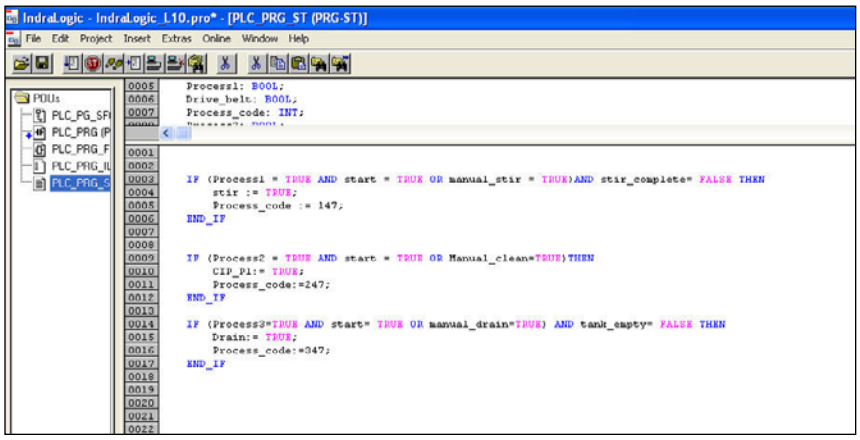
On the downside, this style of programming is not suitable for every application, as the structure that is forced on a program could add unneeded complexity. A large amount of time must be spent up front preparing and planning before any programming is attempted or else the functions charts could become unwieldy and difficult to follow. The overhead required for this type of program causes it to execute slower than the other languages. A final consideration is the inability to convert to other languages. Instruction List, Function Block and Ladder programs can easily be converted into each other, allowing a piece of code to be displayed in the way most comfortable to the user. Structured Text can also be converted into any of these three languages, but SFC stands alone. It cannot be converted. Therefore, you may want to consider this language only for end users who are comfortable with the language and are unlikely to display it in a different format, or for applications where the hardware has the speed and memory necessary to store and execute an SFC program.




Instruction List consists of many lines of code, with each line representing exactly one operation.

### Instruction List

Anyone who has experience programming microprocessors or experience with Assembler language programming will see similarities with Instruction List programming. This language consists of many lines of code, with each line representing exactly one operation. Thus, it is very step-by-step in layout and format, which makes the entry of a series of simple mathematical functions easy. In addition, if the programmer uses only the IEC-defined instructions, a program written in this language can be moved easily between hardware platforms. These advantages make this language very popular in Europe, a fact that is surprising to many U. S. programmers who prefer the ease of maintenance in the graphical languages, and place a lower premium in the transferability of programs.




Structured Text language has seen the greatest increase in adoption and closely resembles a high-level computer programming language such as PASCAL or C.

Instruction List language is a low-level language and as such, will execute much faster in the PLC than a graphical language, like Ladder. This language is also much more compact and will consume less space in PLC memory. The simple one line text entry method supported by this language also allows for very fast program entry—no mouse required, no tab to click! In legacy systems, programs written in this language are easier to display and edit on a handheld programming unit, with no software or laptop required.

Despite the advantages this language provides to a programmer, it seems that maintenance and service engineers do not prefer Instruction List. Perhaps because it is less visual than Ladder, and therefore more difficult to get a sense of what the program is doing and what errors it is experiencing. Similar to the issues with Ladder Diagram and increasing PLC program complexity, it can be a struggle to enter complex functions such as PID in Instruction List. This also applies to complex mathematical computations. Instruction List does not lend itself well to any form of structured programming, such as state programming or step ladder, further limiting its usefulness for implementing large programs. It is also arguable that the advantages of speed and compactness are less relevant, given the processing speeds of modern PLCs and the large amounts of memory available.

### Structured Text

With its IF…THEN loops, CASE selectors, and lines ending in semicolons, Structured Text language closely resembles a high-level computer programming language such as PASCAL or C. The aforementioned *Control Engineering* survey indicated that of all the IEC61131-defined programming languages, Structured Text has seen the greatest increase in adoption.

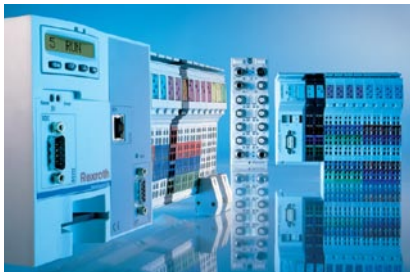
This language perhaps best embraces the growing complexity of PLC programming, such as the process control functions involved in plastics or chemical manufacturing. Trigonometry, calculus, and data analysis can be implemented far easier in this language than in Ladder or Instruction List. Decision loops and pointers (variables used to do indirect addressing) allow for a more compact program implementation than can be achieved in Ladder. The flexible Structured Text editor that is common in most programming packages makes it easy to insert comments throughout a program, and to use indents and line spacing to emphasize related sections of code. This makes the task of structuring a complex program easier. The text-based, non-graphical nature of Structured Text, similar to Instruction List, also runs much faster than Ladder. An additional benefit of Structured Text is that it comes closer than most of the other languages in achieving the transferability promise of the IEC61131 standard. Copying and pasting Structured Text from the editor of one programming package to another can often be done with just a few changes, emancipating a programmer from the hardware platform. A final benefit is that many students currently graduating from engineering studies have a better background in computer languages than in the basics of electrical wiring, and therefore can be more proficient in Structured Text than Ladder programming.

A disadvantage is that for many previously experienced programmers or maintenance and service personnel, the Structured

Text environment is somewhat unfamiliar and unsuitable for troubleshooting. In many ways, the code and structure necessary to make this code maintenance friendly can reduce some of the advantages gained from its compactness. As a result, the main tendency is to use Structured Text "behind the scenes." For example, IEC 61131 allows a programmer to build his or her own functions in one language, which can then be used in another language. Thus the programmer is most likely to encapsulate a Structured Text program inside an instruction called on in Ladder. While this may not necessarily be a disadvantage, the programmer will need to thoroughly test any code that is "hidden" and make sure it is bug-free, since others will not have access to it.

### Choosing an Appropriate Language

With the different programming languages available, it's important to consider a few factors before deciding which to use for your application. Of course, if you're already familiar with a certain language, then the tendency may be to stick with what you know. However, look at some of these match-ups:



Not all PLCs are capable of running the various IEC languages due to lack of memory or processor speed. However, some companies like Bosch Rexroth make robust PLCs that can run all of the languages.

- Ease of maintenance by the final user: SFC
- Universal acceptance of language: Ladder
- Acceptance in Europe: Instruction List or Structured Text
- Speed of execution by the PLC: Instruction List or Structured Text
- Applications mainly using digital I/O and basic processing: Ladder or Function Block
- Ease of changing code later: Ladder
- Ease of use by newer engineers: Structured Text
- Ease of implementing complex mathematical operations: Structured Text
- Applications with repeating processes or processes requiring interlocks and concurrent operations: SFC

Finally, your PLC or PAC platform may also affect the choice of programming languages. Not all automation vendors have programming software that is fully IEC61131-3 compatible. In fact, most of the non-European vendors do not offer this functionality, or only have a very limited spectrum of options, say Ladder and SFC, but none of the other languages or tag-based addressing, etc.

Another consideration is that not all PLCs are capable of running the various IEC languages due to lack of memory or processor speed. This tends to be the case with many micro PLCs. However, some companies like Bosch Rexroth make robust PLCs that can run all of the languages. While many programmers are locked in to a customer specification, if they have the freedom to choose a hardware platform, they should decide which language or languages will work best for the application and then select the hardware and software accordingly.

Rexroth
Bosch Group



Bosch Rexroth Corporation
5150 Prairie Stone Parkway,
Hoffman Estates, IL USA 60192-3707
Telephone (847) 645-3600
www.boschrexroth-us.com