FAQ about IEC 61131-3

IEC 61131-3 (PLC Programming Languages)

This document is a compilation of responses by the Experts of IEC TC65B/WG7/TF3 task force to Frequently Asked Questions (FAQs) about the IEC Standard 61131-3. The contents of this document are not normative and do not form a part of the Standard.

- **Will IEC 61131-3 reduce the innovation of new languages and concepts for PLCs?** The main objective of IEC 61131-3 has been to standardise existing PLC languages. There is no intention that IEC 61131-3 should reduce the development of new PLC languages. Any PLC vendor is free to provide extensions and additional languages where required. Because the standard allows proprietary function blocks to be programmed in non IEC 61131-3 languages such as C++, it always possible to provide extensions fairly 'seamlessly' e.g. packaged as function blocks. This is well demonstrated by IEC 61131-7 "Fuzzy control programming" which defines language extensions for implementing fuzzy logic encapsulated as function blocks.
- **Why does IEC 61131-3 have 'resources'? Is a resource just another name for a PLC?** A resource is a general name for anything that is able to provide the appropriate access to I/O and services to allow IEC 61131-3 programs to execute. Normally a PLC that can execute IEC 61131-3 programs can be regarded as a single resource. However, other processors, such as a personal computer (PC) if able to support the execution of IEC 61131-3 programs may also be regarded as resources.

- **IEC 61131-3 seems to be very complicated. Is it still possible to create simple Ladder programs by users unfamiliar with IEC 61131-3?** An IEC 61131-3 system can still be programmed as a single Ladder program if required. Programming systems may provide a option to create a simple IEC 61131-3 configuration containing one resource, one task, and one program instance of a program type. All of this could be created automatically so the user is only concerned with developing a single ladder program. Function blocks and other IEC 61131-3 constructs do not need to be used.
- **Will IEC 61131-3 languages result in applications that run more slowly and require more memory than using simple ladder?** Attempting to implement IEC 61131-3 constructs such as function blocks on PLCs that were originally only designed to support ladder programs, will inevitably have performance and memory overheads. PLCs specifically designed with firmware to support the execution of IEC 61131-3 programs should not be noticeably slower than classical ladder based PLCs. The improvements in software structure from IEC 61131-3 should allow users to be able to write more efficient applications that will be significantly easier to maintain than monolithic ladder programs.
- **Is it really possible to port IEC 61131-3 software from one vendor's PLC to another?**possible No it is not simply to take an application that runs on one type of PLC and copy it over to another type. There are several problems preventing the direct porting of IEC 61131-3 software.

    1. The PLC I/O systems use different addressing schemes.
    2. The task scan rates supported on different PLCs vary.
    3. Each PLC vendor may have implemented a different sub-set of IEC 61131-3 features.
    4. Similarly each vendor may have different values for implementation specific parameters such as maximum array sizes, string lengths etc.
    5. Finally there is not a standard file format in which to store and port IEC 61131-3 applications.

  Notwithstanding these constraints, at the function and function block level, it may be possible to re-implement identical POUs on different vendors PLCs. Textual source code for POUs developed in ST or IL can be ported between different types of PLCs.
- **Is it possible to automatically convert between IEC 61131-3 languages, for example, can a POU written using LD be viewed and edited in ST or FBD?** This is a favourite IEC 61131-3 myth. There has never been any intention that it should be possible to convert any language into any other language. If a restricted sub-set of each language is used some limited portability may be possible but there are some significant problems. For example, there is no way to represent expressions involving array variables in the FBD language.
- **Can function blocks also have execution control variables EN and ENO, like functions?** The standard is not explicit about whether function blocks may have execution control variables, e.g. for connecting function blocks within ladder rungs. However, from the IEC 1131 languages user guidelines, (part 8 of the IEC 61131 standard), it is implied that for consistency, both functions and function blocks should use EN and ENO variables for execution control in ladder diagrams. It is an implementation decision whether function blocks have EN and ENO variables that can be used in the FBD language for explicit execution control. This may be useful in eliminating execution order ambiguities that might arise in FBD networks.
- **In a full graphic implementation of FBD, is there anyway to distinguish between lines that cross-over and lines that join.** The graphical format of lines, cross-overs and junctions in full graphics implementations of languages LD, FDB and SFC is not specified in IEC 61131-3. It is a implementation decision outside the scope of the standard how fine graphic details such a line cross-overs are depicted.
- **Where are type definitions actually defined and what is their scope?** All type definitions for datatypes and POUs can be regarded as outside the entire IEC 61131-3 configuration and apply to all entities within the configuration, i.e. all type definitions

have global scope. This is as if all type definitions exist in a conceptual 'header file' that is pre-processed before any entity within a configuration is compiled. Extensions to IEC 61131-3 are now being considered to provide a more flexible range of type scopes. With large applications, more specific type scopes may be necessary, such as, a library scope for type definitions that only apply to POUs within a specific library.

- **Do IEC 61131-3 languages enforce data type consistency?** All IEC 61131-3 languages except IL, enforce strict data type consistency, i.e. it is not possible to directly assign (or connect) variables of one data type to variables of different data types. Data type conversion functions are necessary to convert the values of variables to the appropriate type, e.g. an INT value should be converted to a REAL before being assigned to a REAL variable. In the IL language, it is not always possible for a compiler to check that the type of value in the accumulator will match the data type of any variable to be loaded from the accumulator. With IL, run-time checks are required to ensure data type consistency.
- **When are the actions in an SFC actually executed?** Every SFC is encapsulated in a function block or program POU. When the POU is invoked, e.g. because it has been scheduled by a task, the contained SFC is evaluated once, i.e.
  - The current set of active steps is determined.
  - All transitions associated with active steps are evaluated
  - Actions which nominally ceased execution in the previous SFC evaluation (because their Q flag has been cleared) are executed one last time.
  - All actions that are active are executed once.
  - Any active steps that precede transition conditions that are true are deactivated and their succeeding steps are activated.

  The encapsulating POU should be repeatedly invoked for the SFC to progress through its various steps.
- **How can the execution of all actions in an SFC be halted and the SFC restarted?** The execution of all actions in an SFC can be halted by suspending the invocation of the encapsulating POU, see When are the actions in an SFC actually executed? If, at a later time, the POU is again repeatedly invoked, the active actions in the contained SFC will continue to be executed. The only way to re-start an SFC from its initial step and clear all active actions is for the resource containing the POU to have a 'cold start'. A jump back to the initial step is always possible using an explicit branch, e.g. back from the last step in a sequence. However, this cannot guarantee to clear any stored actions or simultaneous sequences which may be been started.
- **Can more than one variable be fixed at the same direct address using the AT construct?** The standard does not forbid this. Allowing variables to be at the same or to use overlapping memory locations is an implementation issue. This however may invalidate data type consistency - see Do IEC 61131-3 languages enforce data type consistency?.
- **When using the AT attribute, does the size of a memory location specified by a direct address have to match the size of the variable?** The standard is unclear; there are two ways of interpreting the purpose of the direct address. It either specifies a) the actual memory location, in which case the location size and variable type size should match or b) the starting address from which the variable will be located, in which case sizes do not need to match.
- **When are user specified initialisation values for variables used?** User specified initial values apply to non-retentive variables both at "cold restart" and "warm restart", but they only apply to retentive variables at "cold restart". On "warm restart" retentive variables have the same values as existed when their resource stopped executing, e.g. due to a power outage. The 61131-3 amendment allows initialisation values defined by the VAR_CONFIG construct to override type specific initial values. Therefore, VAR_CONFIG specified initial values apply at "cold restart" or "warm restart" for non-retentive variables, and to "cold restart" for retentive variables.

- **Can function block instances be passed as inputs to other blocks?** Yes if function block FB1 is passed as an input to a second function block FB2, it is possible to invoke the function block FB1 within the body of FB2. Any input parameters of FB1 not defined in the invocation call within FB2 will take values defined by earlier invocations of FB1 made outside of FB2. Function block instances should be passed as VAR_IN_OUT parameters otherwise the values produced within the internal invocation will not be preserved.
- **Are the assignments of inputs and outputs to programs at the resource level fixed or can they change dynamically?** They are fixed. Programs are the highest level IEC 61131-3 programmable organisation unit. It is not possible to have language statements outside of a program. In a resource declaration it is only possible to assign input and output variables to the program interface. If it is necessary to change the inputs used by part of a program, then all inputs concerned should be passed to the program and any dynamic selection of particular inputs should be done within the program body, e.g. to switch between using two different sets of sensors.