## What is IEC 1131?

The standard IEC 1131 has been established to standardize the multiple languages, sets of instructions and different concepts existing in the field of automation systems. The great variety of PLC concepts has led to an incompatibility between the different PLC platforms and manufacturers. The result was a great effort to be made for training, hard- and software investments.
IEC 1131 standardizes the programming languages, the interfaces between PLC and programming system, the sets of instructions and the handling and structuring of projects. The advantage of using IEC 1131 conform PLCs and programming systems is a portability of all platforms and the use of same concepts reducing costs for automation systems.

The standard consists of several parts and technical reports. The third part of the standard is dedicated to programming languages.
Obviously this standard has a great influence on the concept, structure, features and the handling of a programming system and the way to program a PLC.
The main changes that have come with IEC 1131-3 are:

- Declaration of variables is similar to the variable declaration in higher programming languages.
- Declaration of data types is possible.
- Global and local data can be differentiated.
- Programming means symbolic programming.

## Configuration elements in IEC 1131-3

An IEC 1131-3 conform PLC programming system reflects the hardware structure with the configuration elements. These configuration elements are basically configurations, resources and tasks.

*Configurations in IEC 1131-3*
A configuration can be compared to a programmable controller system, e.g. a rack. In a configuration one or several resources can be defined.

*Resources in IEC 1131-3*
A resource can be compared to a CPU which can be inserted in the rack. In a resource global variables can be declared, which are only valid within this resource. In a resource one or several tasks can be executed.

*Tasks in IEC 1131-3*
Tasks determine the time scheduling of the programs associated with them. This means that programs have to be associated to tasks. The settings of the task determine the time scheduling.

IEC 1131-3 describes different time scheduling models which results in three different task types:

- Cyclic tasks are activated in a certain time interval and the program is executed periodically.

- System tasks are called automatically by the PLC operating system if an error or a change of the operational state of the PLC occurs. They are also known as system programs or SPGs.

- Event or interrupt tasks are activated if a certain event has happened.

Each task has a certain priority. In so called preemptive scheduling systems, an active task with low priority is interrupted immediately, when a task with higher priority becomes active due to a certain event. In systems with non-preemptive scheduling, task interruptions by tasks with higher priority are not possible.

**POUs in IEC 1131-3**

Program organization units or POUs are the language elements of a PLC program. They are small, independent software units containing the program code. The name of a POU should be unique within the project.
In IEC 1131-3 three types of POUs are distinguished referring to their different use:

- programs
- function blocks
- functions

Each POU consists of two different parts: The declaration part and the code body part.

In the declaration part all necessary variables are declared.

The instruction or code body part of a POU is the part in which the instructions are programmed in the desired programming language.

**Functions in IEC 1131-3**

Functions are POUs with multiple input parameters and exactly one output parameter. Calling a function with the same values returns always the same result. Return values can be single data types. Within a function it is possible to call another function but not a function block or a program. Recursive calls are not allowed. The abbreviation for functions is FU.

IEC 1131-3 describes standard functions which can be used while editing your PLC program. According to your hardware and PLC type it is possible that not all standard functions are available or that firmware functions have been added.

**Function blocks in IEC 1131-3**

Function blocks are POUs with multiple input/output parameters and internal memory. The value that a function block returns depends on the value of its internal memory. Within a function block it is possible to call another function block or functions but not a program. Recursive calls are not allowed. The abbreviation for function blocks is FB.
IEC 1131-3 describes standard function blocks which can be used while editing your PLC program. According to your hardware and PLC type it is possible that not all standard function blocks are available or that firmware function blocks have been added.

**Programs in IEC 1131-3**

Programs are POUs which contain a logical combination of functions and function blocks according to the needs of the controller process. The behavior and the use of programs are similar to function blocks. Programs have input and output parameters and they can have internal memory. Programs must be associated to tasks.
Within a program it is possible to call functions and function blocks. Recursive calls are not allowed.

**Instantiation**

For reusing function block definitions IEC 1131-3 provides the possibility of instantiation. This means that a function block or a program is defined once and that its internal memory is allocated to different instances, different memory regions. Each instance has an associated identifier and contains the input and output parameter and the internal memory of the function block or program. A function block can be instantiated in another function block or in a program. The instance name of a function block has to be used as in the VAR declaration of the program or the function block. Programs can be instantiated within resources.

**Data types in IEC 1131-3**

Data types determine what kind of value the variable can have. Data types define the initial value, range of possible values and the number of bits.
IEC 1131-3 distinguishes three kinds of data types:

- Elementary data types
- Generic data types
- User defined data types

**Elementary data types**

The value ranges and the bitsize of elementary data types are described in IEC 1131-3. Elementary data types are shown in the following table:

| Data type | Description | Size | Range |
|---|---|---|---|
| BOOL | Boolean | 1 | 0...1 |
| SINT | Short integer | 8 | -128...127 |
| INT | Integer | 16 | -32768...32767 |
| DINT | Double integer | 32 | -2.147.483.648 up to 2.147.483.647 |
| USINT | Unsigned short integer | 8 | 0 up to 255 |
| UINT | Unsigned integer | 16 | 0 up to 65535 |
| UDINT | Unsigned double integer | 32 | 0 up to 4.294.967.295 |
| REAL | Real numbers | 32 | 3.4E^-38 up to 3.4E^38 |
| TIME | Duration | 32 | #0S up to 47 days |
| BYTE | Bit string of length | 8 | 8 |
| WORD | Bit string of length | 16 | 16 |
| DWORD | Bit string of length | 32 | 32 |

**Generic data types**

Generic data types are data types which include hierarchical groups of elementary data types. ANY_INT includes the elementary data types DINT, INT, SINT, UDINT, UINT and USINT. If a function can be connected with ANY_INT it means that variables of the data types DINT, INT, SINT, UDINT, UINT and USINT can be handled.
Generic data types are shown in the following table:

ANY

- ANY_NUM
  - ANY_REAL
    - REAL
  - ANY_INT
    - DINT, INT, SINT
    - UDINT, UINT, USINT
- ANY_BIT
  - DWORD, WORD, BYTE, BOOL
- STRING
- TIME

**User defined data types**

User defined data types are edited by the application programmer with a TYPE ... END_TYPE declaration using the text editor in a data type worksheet. Derived data types can be enumerated data types, structures or arrays.
***Array data types*** include several elements of one data type. An array can be used to declare several elements of the same type with only one line in the type declaration.
***Structured data types*** include several elements of different data types
User defined strings are ***STRING data types*** with a variable number of characters.
**Enumerated data types** limit the possible values to the enumerated ones. The variable in which the data type is used can have only the enumerated values.

**Literals in IEC 1131-3**

Literals can be used to enter external representation of data. Literals are necessary for the representation of numeric, character strings and time data. Every time you have to enter values, you have to use literals.

*Numeric literals*

The numeric literals which can be used are shown in the following table:

| Type | Examples |
|---|---|
| Integer literals | -12 0 123_456 +986 |
| Real literals | -12.0 0.0 0.4560 3.14159_26 |
| Real literals with exponents | -1.34E-12 -1.34e-12 1.0E+6 |
| Base 2 literals | INT#2#1111_1111 |
| Base 8 literals | INT#8#377 |
| Base 16 literals | INT#16#FF SINT#16#ff |
| Boolean FALSE and TRUE | FALSE TRUE |
| Boolean 0 and 1 | 0, 1 |

Literals which are used in variable worksheets, literals of data type INT or BOOL can be used without keyword as it is shown in the following examples:

For INT#16#ffyou can use 16#ff.
For BOOL#FALSE you can use FALSE.

In variable declarations you can use "var1 : DINT :=10", but in the code body you have to use "LD DINT#10".

*Character string literals*

A character string literal is a sequence of zero or more characters included in two single quote characters.
The character string literals which can be used are shown in the following table:

| Type | Examples |
|---|---|
| Empty string | '' |
| String with a blank | ' ' |
| Not empty string | 'this is a text' |

*Duration literals*

Duration data can be represented in hours, minutes, seconds, milliseconds and in combination.
The duration literals which can be used are shown in the following table:

| Type | Examples |
|---|---|
| Short prefix | T#14ms t#14ms t#12m18s3.5ms T#25h_15m t#25h_15m |
| Long prefix | TIME#14 ms time#14ms TIME#25h_15m time#25h_15m |

*Date and time of day literals*

The date and time of day literals which can be used are shown in the following table:

| Type | Examples |
|---|---|
| Date | DATE#1996-01-24 |
| | date#1996-01-24 |
| | D#1996-01-24 |
| | d#1996-01-24 |
| Time of day | TIME_OF_DAY#15:36:55.36 |
| | time_of_day#15:36:55.36 |
| | TOD#15:36:55.36 |
| | tod#15:36:55.36 |
| Date and time | DATE_AND_TIME#1996-01-24-15:36:55.36 |
| | date_and_time#1996-01-24-15:36:55.36 |
| | DT#1996-01-24-15:36:55.36 |
| | dt#1996-01-24-15:36:55.36 |

**Variables in IEC 1131-3**

In IEC 1131-3 different types of variables are described:

- Symbolic variables
- Directly represented variables
- Located variables

Variables have to be declared in the variable worksheet of the POU using keywords.

**Variable declaration keywords**

In variable declarations variable declaration keywords have to be used. The keywords are described in the following table:

| Keyword | Description |
|---|---|
| VAR | o for internal variables which can be used only within POU.<br>o for declaring the instances of function blocks.<br>o can be used for the declaration of directly represented, located and symbolic variables.<br>o can be used with the keyword 'RETAIN' for declaring retentive variables |
| VAR_INPUT | o for variables which are inputs to functions, function blocks and programs.<br>o to give a value to the POU coming e.g. from another POU.<br>o its value is only read within the POU. can be used only for the declaration of symbolic variables |
| VAR_OUTPUT | o for variables which are outputs to function blocks and programs.<br>o supplies an output value for e.g. other POUs.<br>o its value is written within the POU.<br>o it is also allowed to read the value.<br>o can be used with the keyword 'RETAIN' for declaring retentive variables |
| VAR_IN_OUT | o address of the variable is passed by reference.<br>o the variable can be read or written.<br>o typically used for complex data types such as strings, arrays and structures |
| VAR_EXTERNAL | o for global variables in the POU.<br>o its value is supplied by the declaration of VAR_GLOBAL·<br>o its value can be modified within the POU. |

| | |
|---|---|
| | o can be used only for the declaration of symbolic variables |
| VAR_GLOBAL | o for global variables which can be used in all programs and function blocks of the project.<br>o can be used for the declaration of directly represented, located and symbolic variables.<br>o can be used with the keyword 'RETAIN' for declaring retentive variables |
| END_VAR | o to finish a variable declaration block |

In addition to these keywords three more keywords can be used in variable declarations: RETAIN for retentive variables and the keyword AT which is needed for directly represented and located variables.

The variable declarations differ in their structure according to the different variable types. Variable declarations are done either in the variable worksheet of each POU or in the worksheet for global variables.

**Global and local variables**

The scope of each variable which is determined by the use of the variable keyword is limited either to a POU or to the whole project. Therefore two types can be distinguished:

- Local variables
- Global variables

If a variable can be used only within a POU it is called local variable. In those cases the variable keywords VAR, VAR_INPUT and VAR_OUTPUT can be used.

If a variable can be used within the whole project it is called global variable. It has to be declared as VAR_GLOBAL in the global declaration and as VAR_EXTERNAL in each POU where it is used.

---

It might be useful to declare all I/Os as global variables. In the global variable declaration they should be declared as located variables and in the VAR_EXTERNAL declaration of the POU they should be declared as symbolic variables. The typing effort in case of address changes is less doing it this way.

---

**Symbolic variables**

Symbolic variables are declared with a symbolic name and a data type. The initial value is optional.
The programming system stores the variable to free memory areas of the PLC memory which are not known to the user.
The following example shows a variable declaration of two symbolic variables:

```
VAR
        var1:   BOOL;
        var2:   INT (-22..12);
END_VAR
```

Symbolic variables can be initialized and/or can be declared as retentive variables using the keyword 'RETAIN'.

**Directly represented and located variables**

Directly represented variables are declared without symbolic name but using a logical address.
Located variables are declared using a symbolic name and a logical address.
Both, directly represented and located variables are stored at the declared logical address and it is up to the application programmer to check that no memory address is used twice.
A location declaration consists of the keyword AT, the percent sign "%", a location prefix, a size prefix and the name of the logical address.

Directly represented and located variables can be declared in the global variable worksheet using VAR_GLOBAL or in programs.

The following tables show the location and size prefix for directly represented and located variables:

| Location prefix | Description |
|---|---|
| I | Physical input |
| Q | Physical output |
| M | Physical address in the PLC memory |

| Size prefix | Description |
|---|---|
| X | Single bit size (only with data type BOOL) |
| None | Single bit size |
| B | Byte size (8 bits) |
| W | Word size (16 bits) |
| D | Double word size (32 bits) |
| L | Long word size (64 bits) |

In the following example a declaration of directly represented and located variables is shown:

```
VAR
        var1    AT %QX 2.4  : BOOL;
        var2    AT %IW4     : WORD;
        Var3    AT %QB 7    : BYTE;
END_VAR
```

**Retentive variables**

Retentive variables are variables whose values are stored even if the power is switched off. In the case of a warm start the last value of the variable is going to be used. Retentive variables are declared using the keyword RETAIN as it is shown in the following example:

```
VAR RETAIN
        var1    :       BOOL := TRUE;
END_VAR
```

In this example the variable has got the initial value 'TRUE' which is the initial value for a cold start. In case of a warm start the current value of the variable is used.

**Initializing variables**

According to IEC 1131-3 initial values can be assigned to variables. This means that a variable which is going to be used for the first time in the PLC program is used with its initial value. Initial values can be given to all kind of variables except in VAR_EXTERNAL declarations.
Initial values have to be inserted at the end of the declaration line of the variable using ':=' as it is shown in the following example:

```
VAR
        var1:   INT := 28;
        var2:   TIME := T#1s;
        var3:   AT%QX0.0 := TRUE;
END_VAR
```

It is not possible to initialize variables which are located at physical inputs.

The initial value has to fit to the data type. It is not possible to use e.g. the data type BOOL and an initial value '5'. In this case the system displays an error message.

The initial value is optional. If no initial value is used, the variable is initialized with the default initial value of the data type or with the retained value in case of retentive variables.

## Programming languages in IEC 1131-3

IEC 1131-3 defines the syntax of 5 programming languages, defines a certain representation and describes the different elements which can be used in the language. The programming languages can be differentiated by the physical appearance into 2 textual languages and 3 graphical languages.
The textual languages are Instruction List (IL) and Structured Text (ST).
The graphical languages are Function Block Diagram (FBD), Ladder Diagram (LD) and Sequential Function Chart (SFC).

**Instruction List - IL**

A code body programmed in the textual language IL consists of a sequence of instructions. Each instruction starts at a new line.
An example of a simple IL instruction sequence is shown in the following figure:

```
LD    var1   (* loading operand into accumulator *)
AND   var2   (* processing accumulator with operand *)
ST    var3   (* storing accumulator into memory *)
```

Each line begins with an operator followed by one operand. Additionally modifiers can be used.
In IL code bodies jumps can be used. An example for a jump and the use of its label is shown in the following figure:

```
      LD      var1
      EQ      INT#100
      JMPC    label
      LD      var2
      ADD     var3
      ST      var4
label: LD     %IX2.2
```

In IL code bodies comments can be inserted using asterisks and parentheses.

**Structured Text - ST**

A code body programmed in the textual language ST consists of statements and expressions.
Different types of statements can be used while editing. All statements have to finish with a semicolon.
An expression is a construct which returns one value for the execution of statements. Expressions consist of operators and operands. The operators have to be applied to the operands in the way that the operator with the highest precedence is followed by the operators with the next lower precedence.
In ST code body comments can be inserted using asterisks and parentheses.

**Function Block Diagram - FBD**

A code body programmed in the graphical language FBD is composed of functions and function blocks which are connected with each other or with variables using lines . These lines can also be connected with each other. In FBD networks it is not possible to connect outputs with outputs.
The set of connected objects is called FBD network.
Connectors can be used in case of large networks for better structuring the elements in the worksheet. Connectors replace connection lines.

Jumps can be used to jump to a label in the current worksheet.
Returns can be inserted to go back to the calling POU.

**Ladder Diagram - LD**

A code body programmed in the graphic language LD is composed of contacts and coils. According to IEC 1131-3 different types of contacts and coils can be used. The contacts lead according to their type the power from the left to the right. Coils store the incoming value to Boolean variables.
These contacts and coils are connected with lines and are bounded on the left and on the right with power rails. The state of the left power rail is considered ON all the time. The right rail is optional.

In addition to the serial connections of contacts and coils parallel branches can be created. Parallel branches are also called wired-ORs.
The set of connected objects is called LD network. Every LD network shall at least contain one coil and a left power rail.
Connectors can be used in case of large networks for better structuring the elements in the worksheet. Connectors replace connection lines.
Jumps can be used to jump to a label in the current worksheet.

Returns can be inserted to go back to the calling POU.

Variables in LD code bodies are always Boolean variables. While inserting contacts or coils the variable name can be entered. The variable name is displayed above the contact or coil in the worksheet.

**Sequential Function Chart - SFC**

A code body programmed in the graphic language SFC is composed of steps and transitions which are connected with directed links.

One or several action blocks can be associated to a step. While the step is active the associated action is executed according to the action qualifier. The action can be a Boolean variable. It is also possible to edit the code to be executed in an additional code body named detail. In this case the name of the code body worksheet has to be used as the name for the action.

A transition represents the condition in which the process passes from one step to another. If a transition becomes TRUE the preceding step is executed once again and the succeeding step becomes active. The transition can be either a Boolean variable or a directly connected Boolean expression in FBD or LD. It is also possible to edit the code to be executed in an additional code body named detail. In this case the name of the code body worksheet has to be inserted at the transition.

The set of connected objects is called SFC network. A SFC network must have always one initial step which is the first step to be executed when the POU is called.

Simultaneous branches or alternative branches can be inserted in the SFC network.

## IEC 1131-3 and your programming system

Your programming system is a standard programming system for IEC designed PLCs and traditional PLCs. It is based on the standard IEC 1131-3 and includes the full range of IEC features.

## Projects in your programming system

This programming system bases on the windows technology using the graphic surface of MS-Windows. Therefore the IEC 1131-3 principles are mostly realized graphically with symbols and icons or dialogs are used in which properties of elements can be set.

An IEC 1131-3 project contains all necessary elements of an automation system. It consists of libraries, data types, POUs and the set of configuration elements (such as resources, tasks and programs) which are represented in the subtree 'Physical Hardware'. A project is represented in the project tree.

## Libraries in your programming system

Libraries are projects which have been announced as libraries. You can reuse the programs, function blocks, functions and the user defined data types of the library in the project you are editing.
Firmware libraries are libraries containing POUs prepared by the PLC manufacturer. The file extension for firmware libraries is *.fwl.
User libraries are projects which you have created before and from which you want to reuse POUs. The file extension for user libraries is *.mwt. If a project is created with an earlier programming system version, younger than 1.3.6.14, the project file name has the extension *.pwt. Projects which are created earlier cannot be announced as libraries.

Libraries have an own subtree in the project tree. You can either display the complete project tree or only the subtree 'Libraries' by clicking on the tab 'Libraries' at the bottom of the project tree.
The subtree 'Library' consists of two or more icons. The first icon is a directory node. The element of this directory node represents an announced library.

**POUs in your programming system**

Programs, function blocks and functions can be edited in the project tree.
A POU consists of several worksheets. The set of these worksheets builds the POUs.
These worksheets are:

- A description worksheet which contains text entered by the PLC application programmer. These user notes which describe the POU or configuration elements are serving for documentation purposes. These worksheets are optional.
  Note: It is possible to use the description worksheet as context sensitive help if you are calling help on a user defined function or FB (e.g. when working with the Edit Wizard).
  For detailed information please refer to the topic Calling help on FU/FB.

- A variables worksheet containing the declaration of the local variables.
- A code body worksheet containing the actual code, edited one of the available programming languages.

In case of a POU in the programming language SFC some more worksheets can be edited: action and transition worksheets.

**Data types in your programming system**

Data types in your programming system are basically the same data types as described in IEC 1131-3. User defined data types can be edited in data type worksheets which are contained in the subtree 'Data Types' in the project tree.

**Configuration elements in your programming system**

Configuration elements are represented graphically in the project tree.
The programming system reflects the structure of configuration elements in the subtree 'Physical Hardware' which may differ from PLC to PLC.

In general one or several configurations can be used. In every configuration one or several resources can be declared. Several tasks with their associated programs can be used within one resource.

**Identifier**

Identifiers can consist of letters, digits and underscores. The identifier has to begin with a letter or an underscore. Upper and lower case can be mixed.
Multiple underscores or blanks are not allowed.
For identifiers different lengths are allowed, as shown in the following table:

| ***Identifier for...*** | ***Number of characters*** |
|---|---|
| POUs, description worksheets, projects, configuration elements, bitmaps, pagelayouts, archive files | 24 characters |
| Variable and code body worksheets | 24 characters |
| Steps, actions and transitions | 24 characters |
| Variables, instance names | 30 characters |