

The 3rd Edition of IEC 61131-3

Eelco van der Wal
Managing Director PLCopen

Overview of the IEC 61131 and why a 3rd edition?

Living standards

IEC 61131 Parts

Project	Title	Valid till
61131- 1, Ed 2.0	General information, 2003-05	2013
61131- 2, Ed 3.0	Equipment requirements and tests, 2007-07	2012
61131- 3, Ed 3.0	Programming languages (Currently CDV - Committee Draft for Voting)	2012+5
61131- 4, Ed 2.0	User guidelines (TR), 2004-07	<i>2010</i>
61131- 5, Ed 1.0	Communications, 2000-11	2013
61131- 6, Ed 1.0	Functional safety for PLC (Currently CDV - Committee Draft for Voting)	2012+5
61131- 7, Ed 1.0	Fuzzy control programming, 2000-08	2013
61131- 8, Ed 2.0	Guidelines applic. & implem. progr. languages (TR), 2003-09	<i>2008</i>
61131- 9, Ed 1.0	Single-drop digital communication interface for small sensors and actuators (SDCI) aka "IO-Link" (Currently CD - Committee Draft)	2012+5

Why include Object Orientation?

Main reason:

**To link better to the scared resources
of engineers in the future**

**We should better adopt to them
then vice versa**

3rd edition fully compatible with 2nd edition + extensions

**The OOP features are selectable
and usable over time**

And many other enhancements

and proposals to reduce – like delete IL

Fully compatible extensions:

- **Rules for implicit type conversions**
- **Functions for endianness, type conversion**
- **References** (no pointer arithmetic)
- **Object oriented extentions:**

FB / Class: Method, Inheritance, Interface, Polymorphism

- **Namespaces**
- Etc.

Details walkthrough (1/7)

6.1.5	Comments	New: Single line comment // text
6.2.1	Numeric literals ...	New: INT#16#7FFF = decimal value 32767 New: Typed double-byte string and char (using "single quote" character)
6.2.3	Duration literal	New: Nanosec
6.3	Data Types	Clar: Harmonization with variables, Subclause reorganized
6.3.2	Elementary data types	New: (Enum) Type with named values
6.3.3	Generic data type	Corr: Hierarchy
0	User-defined data type Declaration and initialization	New: Enum data type New: Named values New: OVERLAP, OVERLAYED New: Relative location for structure elements New: Directly derived data type New: REF_TO declaration and operations New: Structured data type with relative addressing AT and OVERLAP New: Directly represented components of a structure - partly specified using " * " New: Initialization of a variable with user-defined data type

Details walkthrough (2/7)

6.4.1	Variables – General	Clar: Harmonization with data types Subclauses and tables reorganized New: Variable with user-defined data type
6.4.1	Declaration and initialization	Clar: Declaration and initialization
6.4.3	Variable-length array	New: ARRAY [*, *, *] OF INT; Std FBs for upper and lower bound
6.4.4	Overlaid variables	New: OVERLAYED
6.4.5	Directly repres. variable	Clar.
6.4.6	Retentive variables	Clar: RETAIN
6.5.1	Common features of POU's	New: Overview of common features . Subclauses reorganized Clar: Assignment and expression New: Partial access of ANY-Bit Clar: Call representation (execution sequence) Clar: Execution control EN ENO New: Implicit – explicit; Conversion rules

Details walkthrough (3/7)

6.5.2	Function	<p>Clar: Harmonization of rules with function blocks Subclauses reorganized</p> <p>Clar: Overview, Declaration, call representation, call rules.</p> <p>Clar: Features tables: Function declaration and call</p>
6.5.2.3	Function call	<p>Clar: Results: VAR_EXTERNAL, VAR_IN_OUT</p> <p>New: Function without function result</p> <p>Clar: Feature Table 21 restructured</p> <p>Clar: Initialization of Input and output a feature</p>
6.5.2.4	Typed overloading	<p>Corr: WORD_TO_INT vs. TO_INT</p>
6.5.2.5.2	Type conversion	<p>New: Implicit and explicit conversion</p> <p>New: TRUNC vs. TRUNC_**</p> <p>New: Conversion of Numerical, bit, etc.</p> <p>New: Conversion of Time = implementation dependent</p>
6.5.2.5.7	Functions of time and date	<p>New: Concatenate and split</p>
6.5.2.5.8	Function for endianness	<p>New: Endianness conversion functions</p>
6.5.2.5.10	Functions	<p>New: Validate</p>

Details walkthrough (4/7)

6.5.3	Function block	<p>Clar: Harmonization of rules for function Subclauses reorganized</p> <p>Clar: Overview, Declaration, call representation, call rules.</p> <p>Clar: Features tables: Function block declaration and call</p> <p>New: Error if no value specified for in-out and FB instance</p>
6.5.3.2	FB type declaration	<p>New/Clar: Text and feature table: Rules for FB I/O var usage</p> <p>Clar: Feature table: FB decl. restructured (Harmonization)</p> <p>New: In feature table: Init. of inputs and outputs (was missing)</p> <p>New: FB result like Function</p> <p>New: Feature table: FB instance decl (was missing)</p>
6.5.3.3	FB instance declaration	Clar: Text and feature table
6.5.3.4	FB call	Clar: Textual and graphical repr., feature table (Harmonized with function)
6.5.3.5.2	Bistable elements	New: Additionally long name: SR SET1, RESET... vs. S1, R...
6.5.3.5.4	Counters	Del: Long input names: LOAD, RESET

Details walkthrough (5/7)

6.5.4	Object oriented ext. of FB concept	<p>New: FB with optional OO-extensions</p> <p>New: CLASS as special case of OO FB</p> <p>New: Function block inherits from base class</p> <ul style="list-style-type: none"> - VAR_IN_OUT of a (base) FB type may be assigned an instance of a derived FB type, - A reference to a (base) FB type may be assigned the address of an instance of a derived FB type
	OO FB	Clar: Feature Table restructured
	Classes - Feature	New: CLASS as subset of OO FB and feature table
	Methods	New: METHOD for FB
	Interfaces	New: INTERFACE, method prototypes, FB IMPLEMENTS
	Inheritance	New: EXTENDS for Interface and FB
6.5.7.3.2	FB inheritance	New: FINAL, SUPER
6.5.7.3.6	Type check and cast	New: Type check, derivation check, cast
6.6.5	SFC - Evaluation rules	Corr: Simult. seq. – divergence and convergence
6.8	Namespaces	New: NAMESPACE, INTERNAL ACCESS, PUBLIC ACCESS

Details walkthrough (6/7)

7.2	Instruction list	<p>DEL: Proposal: This language is a) often implementation specifically realised and b) outdated as an assembler like language. Therefore it will not be contained in the next version of this part of the standard.</p> <p>New: Method call with formal and nonm-formal list New: IL FB call – Counter RESET (Long name)</p>
7.2.3	IL function and FB call	New: Harmonized with the general call rules.
7.3.2	Structured Text	<p>New: Feature table New: Statements: “;”, CONTINUE</p>
8.1.2	Representation of lines and blocks	Clar: Feature table > Figure Example (not normative anymore)
8.2.4	Ladder Diagram	New: Contacts for Compare (typed and overloaded)

Details walkthrough (7/7)

Annex A	Syntax	New: updatenew element (open)
Annex B	Delimiter and kexword	New: update new element (open)
Annex C	Implement. dependencies	New: update new element (open)
Annex D	Error conditions	New: update new elements (open)
Annex E	Extension of 3 rd Edition	New: Text, Figure and Table (This table (here) will go to an Annex
Annex	Interoperability with IEC 61499 devices	Del:
Annex	Examples	Del: Move to next edition of part 8 - Guidelines

Some examples

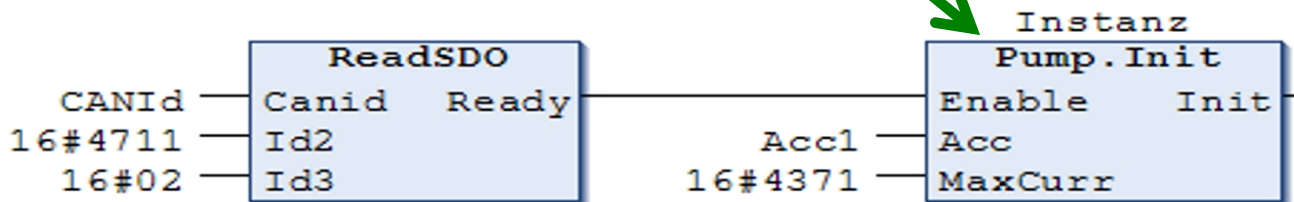
1. Objects

```
PLC_PRG
1 PROGRAM PLC_PRG
2 VAR
3   Pump1:Pump;
4   Pump2:MonitoredPump;
5   (* Instanziierung des FB mit seinen Methoden *)
6   IM1, IM2:InitMove;
7   DriveCAN1:CANDrive;
8   DriveAnalog1:AnalogDrive;
9 END_VAR

1 Pump1.Start(Forward); //Methodenaufruf
2 IF NOT Pump2.HasError() THEN
3   Pump2.Start(Backward); // Alle Methoden gehen!
4 END_IF
5
6 DriveCAN1.SetCANId(12);
```

Creation of Objects:
through declaration of Variables

Usage of Objects:
Call of Methods



2. Classes

Classes define Data and Operations for the relevant instances of the objects

Definitions of the instance data:
Variables of the FB

The screenshot displays the SIMATIC Manager interface. The main editor shows the definition of a Function Block (FB) named 'Pump'. The code is as follows:

```
FUNCTION_BLOCK Pump
VAR
  Enabled: BOOL;
  Dir: Direction;
END_VAR
```

Two method editors are also visible:

- GetState [Pump]:**

```
METHOD GetState : BOOL
VAR_INPUT
END_VAR
VAR
END_VAR
1 GetState:=Enabled;
```
- Start [Pump]:**

```
METHOD Start : BOOL
VAR_INPUT
  WantedDirection: INT;
END_VAR
VAR
END_VAR
1 Enabled := TRUE;
2 Dir := WantedDirection;
```

Definition of the Operations:
Method = Action of the FB with own Variables + Return value

3. Sub (1): Inheritance

The screenshot shows two code blocks in a PLCopen IDE. The top block is 'MonitoredPump' and the bottom block is 'HasError [MonitoredPump]'. Green arrows point from text annotations to specific parts of the code.

```
FUNCTION_BLOCK MonitoredPump EXTENDS Pump
VAR
    MonitoredState:State;
END_VAR

METHOD HasError : BOOL
VAR
END_VAR

HasError:=MonitoredState<>OK;
```

Annotations:

- Spezialfall von Pump (points to EXTENDS)
- zusätzliches Datum (points to MonitoredState)
- zusätzliche Operation (points to HasError)
- bei unverändertem Start u. GetState (points to HasError)

- ❑ EXTENDS deklariert, dass Spezialfall vorliegt
- ❑ Neuer FB hat zumindest alle Daten u. Operationen des alten *durch Vererbung*
- ❑ Erweitern und anpassen (statt Copy&Modify)

3. Sub(2): Interfaces

```

iDrive
1  INTERFACE iDrive
2
-----
HasError [iDrive]
1  METHOD HasError : BOOL
2  VAR_INPUT
3  END_VAR
4
-----
Home [iDrive]
1  METHOD Home : BOOL
2  VAR_INPUT
3  END_VAR
4
-----
MoveAbsolute[iDrive]
1  METHOD MoveAbsolute : BOOL
2  VAR_INPUT
3    Pos:INT;
4  END_VAR
5

```

- ❑ **INTERFACE = Oberklasse, so allgemein, dass weder Daten noch Op.-Implementierungen feststehen (*rein abstrakte Klasse*)**
- ❑ **Völlig unabhängig definierte FBs können Spezialfälle davon sein → *eine* Hierarchie**
- ❑ **Interface definiert die Methodenschnittstelle, FBs implementieren diese (IMPLEMENTS)**

3. Sub(3): Polymorphie

The image displays two windows of PLC code. The top window, titled 'DriveManager', contains the following code:

```
1 FUNCTION_BLOCK DriveManager
2 VAR
3   DriveArray : ARRAY[0..3] OF iDRIVE :=[DriveCAN1,
4     DriveAnalog1,
5     DriveAnalog2,
6     DriveCAN2];
7   DriveCAN1, DriveCAN2 : CANDrive;
8   DriveAnalog1, DriveAnalog2 : AnalogDrive;
9   I: INT;
10  Init: BOOL;
11 END_VAR
```

The bottom window, titled 'InitMove', contains the following code:

```
1 FUNCTION_BLOCK InitMove
2 VAR_INPUT
3   D:iDrive;
4   POS:INT;
5 END_VAR
6 VAR_OUTPUT
7   Done: BOOL;
8 END_VAR
9 IF D.Home () THEN
10  IF D.MoveAbsolute (POS) THEN
11    Done:=TRUE;
12  END_IF
13 END_IF
```

Annotations and callouts:

- A grey box at the top right contains the text: **Interface als Typ**
→ **Referenz auf FB-Instanz**. A green arrow points from this box to the `iDRIVE` type definition in the `DriveManager` code.
- A blue box on the left contains the text: **ein Array / eine POE für Instanzen verschiedener FBs**. A green arrow points from this box to the `DriveArray` declaration in the `DriveManager` code.
- Two green arrows point from the `DriveArray` declaration to the `im1(DriveCAN1)` and `im1(DriveAnalog1)` calls in the `InitMove` code.
- A green box at the bottom left contains the text: **einheitlicher Aufruf der jeweils passenden Implementierung (dynamisches Binden)**. A green arrow points from this box to the `DriveArray[I].Home()` call in the `InitMove` code.

Comparison

Spracheigenschaften	IEC	3 rd Edition	C++	Java	C#
Multi languages	+	+	-	-	-
OOP/procedural mixed	-	+	+	-	-
Classes	~ (FB)	+	+	+	+
Methods	~ (Aktionen)	+	+	+	+
Interfaces	-	+	-	+	+
Polymorphi	-	+	+/-	+	+
„Referenzsemantik“	-	+ (Interfaces)	-	+	+
Constructor / Destructor	-	+	+	+	+
Properties	-	+	-	-	+
Dyn. Memory („new“)	-	-	+	+	+
Access control	~ (Variablen)	~ (Variablen)	+	+	+

