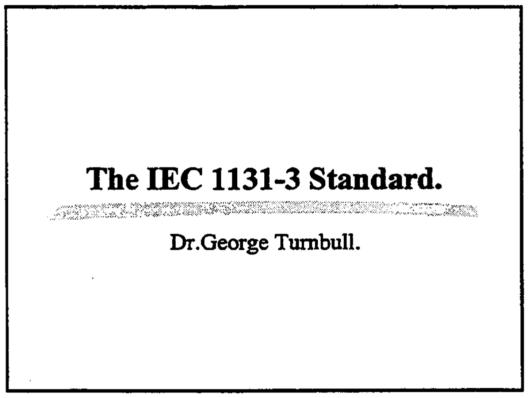
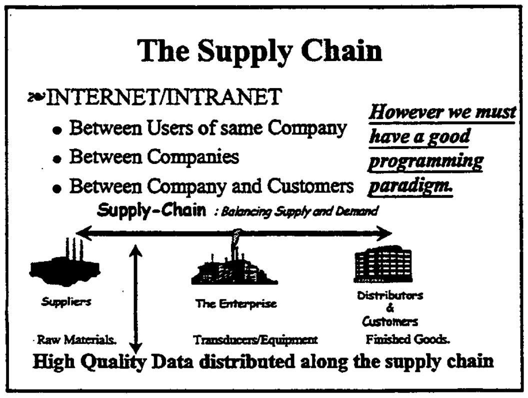


The IEC 1131-3 Standard.

Dr.George Turnbull.

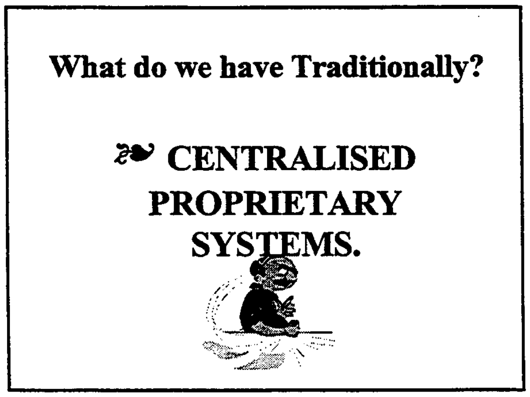


The Supply Chain

- INTERNET/INTRANET
  - Between Users of same Company
  - Between Companies
  - Between Company and Customers

<u>However we must have a good programming paradigm.</u>

High Quality Data distributed along the supply chain



What do we have Traditionally?

- **CENTRALISED PROPRIETARY SYSTEMS.**

What do we Really Need?

❧ **OPEN DISTRIBUTED ACQUISITION AND CONTROL SYSTEMS.**

❧**ODACS**

The Advantages of ODACS.

❧Significant material capital cost savings.
❧Large reductions in engineering time.
❧Better product through tighter control.
❧Less scrap during product changeover.
❧More efficient preventative maintenance.
❧High integrity gives high availability.
❧**NEED OPEN STANDARDS THROUGHOUT.**

For programming at a factory level, IEC 1131 is the only real choice

Now to IEC 1131.

❧This is not just a programming standard.
❧Neither is it a stand alone standard.
❧It came from formalising successful approaches.

- Grafset.
- FBD's

The Full IEC 1131 Standard.

- -2 Environment, Functionality and Safety.
- -3 Graphical and Textural Languages.
- -5 Communication between Controllers.

  - It is a continually expanding set of standards and is effectively an umbrella document for other standards and directives.

The Reason for IEC 1131-3.

- Primarily for People Programming PLC's.
- These are not Computer Programmers.
- However the Problems are Complex.

  - Engineers involved with developing IEC 1131-3 software should be aware of the benefits from using many of the features provided.

Five Different Languages.

- Three Graphical Languages-SFC,FBD,LD.
- Two Textural Languages-ST and IL.
- Can Mix Languages within an Application.

  - It is important to note that each language is suitable for certain types of problem and therefore choose carefully.

Three Graphical Languages.

➢Sequential Function Chart (SFC).
➢Function Block Diagram (FBD).
➢Ladder Diagram (LD).

- Now let us summarise the main strengths of the above languages in the context of where it is optimal to use them.

Main Areas of Use - GL.

➢SFC - Main area is Top-level Sequence.
➢FBD - Principally for Continuous Control.
➢LD - Mostly Consequential Logic-History.

- With no history in ladders, a good methodology is to construct the continuous control first in FBD and then add SFC for sequencing.

Two Textural Languages.

➢Structured Text - similar Structure to Pascal
➢Instruction List - similar to Assembler.
➢IL and LD are closely related - early PLC's.

- ST must be used with complex arithmetic and where the data is held in complex data types and without history, for all programs.

Main Areas of Use - TL.

- ST for FB internals, e.g. PID Control.
- IL for elementary logic, LD Translation.
- There is no need to use LD but history.

  - The examples in the back of the standard are extremely useful in comparing how the same problem can be done with FB/LD and ST/IL.

Main Advantages of IEC 1131

- Software Structured with strong data typing.
- Complex Sequence Behavior/Exec Control.
- Provides Encapsulation and Info Hiding.

  - Although these sound like "computer-ease", the concepts are actually quite simple and the features are extremely important.

Software is Structured.

- Programme Organisation Units - POU's.
- Large Program is Broken Down/Hierarchy.
- Libraries of Functions, FB's,and Programs.

  - Another way of looking at this is that one can achieve: -
    - **RE-USABLE SOFTWARE.**

Strong Data Typing.

- Data protection, unlike conventional PLC's.
- Erroneous Operations detected by compiler.
- This is a characteristic of Pascal.

  - Note the emphasis is on ease of use but with built in features which greatly minimise the possibility of run-time errors.

Complex Sequential Behavior.

- Support for Complex Sequences via SFC's.
- Define Steps, Actions and Transitions.
- Can use hierarchy - drill into Major Phases.

  - This aspect is very important, for example in stopping or starting machines. Emergency shut-down possible with certain precautions.

Execution Control.

- Parts of Program need different Speeds.
- Continuous PID typically 100ms or greater.
- Shut-down Logic often 1ms or less.

  - Tasks can be assigned to run at different scan times so that the responsiveness of the program can be tuned to match the application.

Encapsulation and Info Hiding.

- Encapsulation = Collection as Single Entity.
- Info Hiding = Can only Change Data.
- Example is only having FB behavior.
  - Danger of using packaged control strategy with conflicting or inconsistent data and control signals is very much reduced.

Some Advice.

- General Guidelines on using IEC1131-3.
- Application Decomposition/FB Design.
- Coding and Layout Standards.
  - In addition there are some notes on a pair of possible problem areas - Cold start/Warm start initialisation and special SFC considerations.

GUIDELINES ON USING IEC 1131-3 LANGUAGES

Creating an IEC 1131-3 application requires the following main design steps, as outlined by R. Lewis [4]:

1. Identification of the external interfaces to the control system, i.e. definition of all inputs from the sensors and outputs to actuators such as valves and switchgear.
2. Definition of the main signals exchanged between the control system and the rest of the plant.
3. Definition of all operator interactions, overrides and supervisory data.
4. Analysis of the control problem broken down from the top level into the logical partitions.
5. Definition of required POUs, e.g. programs and function blocks, this includes selecting the appropriate IEC language. In some cases, existing POU definitions available from libraries may be used.
6. Definition of scans cycle time requirements for the different parts of the application.
7. Configuration of the system by defining resources, linking programs with physical inputs and outputs and assigning programs and function blocks to tasks.

GUIDELINES ON USING IEC 1131-3 LANGUAGES

Creating an IEC 1131-3 application requires the following main design steps, as outlined by R. Lewis [4]:

1. Identification of the external interfaces to the control system, i.e. definition of all inputs from the sensors and outputs to actuators such as valves and switchgear.
2. Definition of the main signals exchanged between the control system and the rest of the plant.
3. Definition of all operator interactions, overrides and supervisory data.
4. Analysis of the control problem broken down from the top level into the logical partitions.
5. Definition of required POUs, e.g. programs and function blocks, this includes selecting the appropriate IEC language. In some cases, existing POU definitions available from libraries may be used.
6. Definition of scans cycle time requirements for the different parts of the application.
7. Configuration of the system by defining resources, linking programs with physical inputs and outputs and assigning programs and function blocks to tasks.

Application decomposition

The decomposition of the application into logical partitions (design step 4) is particularly important. Each partition should deal with one aspect of the application and will become an IEC 1131-3 Program Organisation Unit, i.e. a Program, if it exists at the top level, or a Function Block or Function.

Ideally, the interface between each POU should be as simple as possible (i.e. the number of variables that link POUs should be minimised). This is also called 'loose coupling' and is important in improving software quality. As S. McConnell [3] recommends for mainstream software - "The goal is to create routines with internal integrity and small, direct, visible and flexible relations to other routines".

For example, consider the design of a control system for a fermentation process, as depicted in figure 2. The control can be broken down into a number of logical partitions as follows:

- **Main sequence** concerned with the principal fermentation process steps e.g. *filling, heating, agitating, fermenting, harvesting, cleaning*.
- **Valve control** concerned with operating valves used to fill and empty the fermentation vessel and checking that they are in the correct state.
- **Temperature control** concerned with monitoring the temperature of the vessel and modulating the heater to maintain the correct temperature for the fermentation process.
- **Agitator control** concerned with activating the agitator motor when required by the main process sequence and ensuring that the agitator speed is within process limits.
- **pH Control** concerned with monitoring the acidity of the fermentation contents, and adding acidic or alkali reagents to modify the pH level.

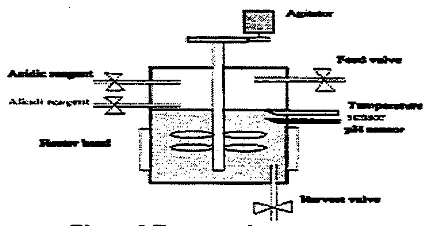


Figure 2 Fermentation system

The top level structure of this system can be represented as a number of linked function blocks as shown in the following figure:

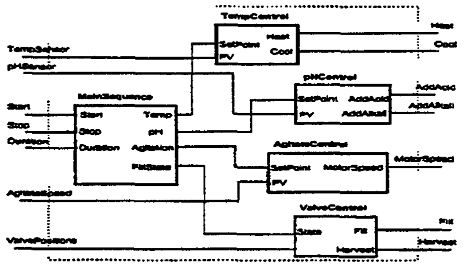


Figure 3 Fermentation Control Program

All of these function blocks are encapsulated within the Fermentation Control Program POU. The most appropriate IEC language for each of these POUs is listed in the following table:

POU	Language
MainSequence	*Sequential Function Chart* - This POU deals mainly with process states and events.
TempControl	*Function Block Diagram* - A simple PID control loop can be depicted graphically using the FBD language.
pHControl	*Structured Text* - The control of pH involves mathematical expressions, which are best defined using the ST language.
AgitateControl	*Instruction List* - Very simple control strategy to regulate the Agitation speed can be defined using the IL language; ST could also be used.
ValveControl	*Ladder Diagram* - Control involves digital logic for the valve interlocks can be defined graphically using the LD language.

Table 2 Examples of appropriate IEC Language

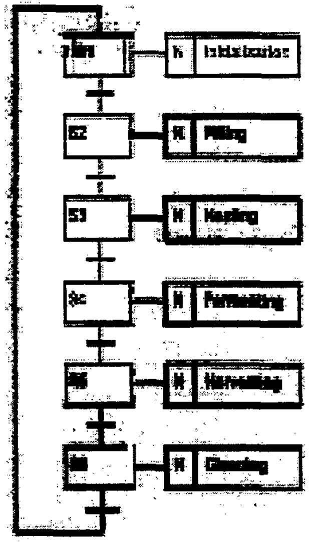


Figure 4 Outline SFC for the MainSequence Function Block

Function Block design recommendations

When designing function blocks the following aspects of the design should always be carefully considered: Identify all the failure modes and ensure that the behaviour is tolerant to various extreme input values. The use of 'defensive programming' is particularly important in control systems where the expected range of values may change as applications are modified over time.

IEC 1131-3 does not define any form of exception handling. It is therefore important to check that all values that may cause run-time errors are detected. For example, always check that divisors are greater than zero to avoid any possibility of 'division by zero' run-time errors.

Where appropriate, provide facilities for testing the function block. This will particularly apply to function blocks that deal with hardware or specific I/O signals.

Consider providing inputs and outputs for operator control. For example, a PID control function block requires an AUTO/MANUAL input to switch between automatic and manual control.

Coding and layout standards

As with all large software projects, it is recommended that coding standards are adopted when creating IEC 1131-3 applications. To aid readability and therefore help improve software maintenance consider:

- Having a consistent signal naming policy, e.g. use a signal name prefix such as PRS_, for pressures, TMP_, for temperatures. Some variation on the Hungarian naming convention adopted by C and C++ programmers may be useful [3]; this defines standard variable prefixes for different types of data.
- Name function block and program type definitions consistently, e.g. PIDSimple, PIDHeatCool, PIDAutoTune.
- Functions should be named after the type of value they return, e.g. MaximumSpeed(), ConvertToPressure()
- Ensure that textual languages are indented correctly to aid readability.
- Keep the definitions of all POUs to a moderate size by encapsulating complex behaviour in lower level function blocks and functions. On the other hand, avoid having a high number of very small POUs as this may result in applications that are difficult to understand.
- Ensure that graphical languages are laid out in a way that allows signals to be traced easily; of signal cross-overs if possible.
- If there are a large number of signals passing between POUs, consider packaging the structure, i.e. like a signal bus. It is then only necessary to pass a single value.
- Always comment graphical and textual languages so that the purpose and functionality is absolutely clear.

Cold start/ warm start initialisation

PLC applications generally need to be programmed to behave correctly when either started from cold (i.e. started for the first time) or from warm (i.e. when the states of some internal variables have been retained in some form of non-volatile memory). Warm starts occur after a PLC has lost power for a short period so that the program can continue from its last state.

Cold starting applications will normally be fairly straight forward because all variables must have user defined or default initial values. Generally, every application will need to assess the initial controlled plant state and then reset the plant into a defined state as part of the cold start initialisation.

Warm starts may be more problematic and will generally need careful consideration. It is important to identify variables that need to be retained when the PLC loses power; identified by the RETAIN keyword. On a warm re-start the IEC 1131-3 program will continue from the point of power failure. The programmer will need to consider whether the plant is still in a valid state for the program to continue. For example, when controlling a heat treatment furnace, it may be possible to re-continue the process if the power outage has been for a short time. If the furnace temperature has dropped, the product may have been irreversibly damaged and the process will need to be aborted.

Sequential Function Charts, special considerations

There are a number of aspects to the design of sequential function charts that need to be considered.

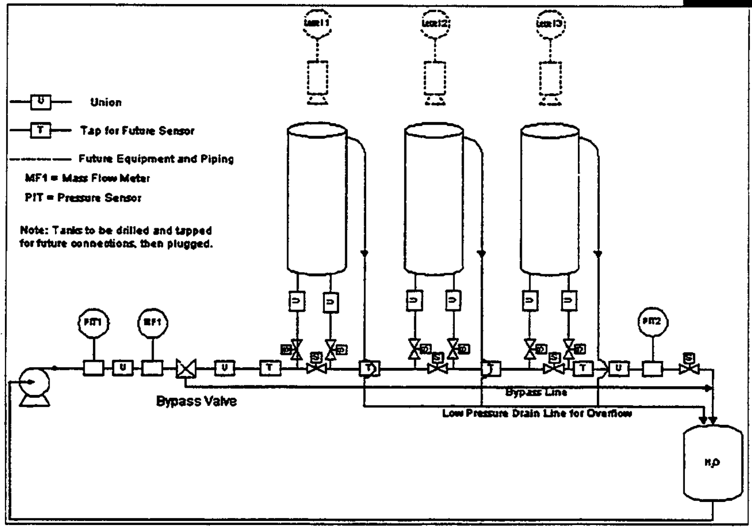
- Care should be taken to ensure that all stored actions are terminated when the higher level parent SFC is aborted, for example, on an error condition.
- Ensure that the SFC design does not allow erroneous steps to become active. This can occur when simultaneous sequences are initiated but are not terminated correctly. Note that many IEC 1131-3 programming stations are able to analyse SFC design and identify faulty constructs.
- Having more than one active action interacting with the same output signals should be avoided. Otherwise, unpredictable behaviour may result.

IEC 1499.

- Extends the function block model in 2 areas
- Allows for multiple nodes - i.e. networking.
- Allows for event driven / triggered models .

  - These are two very important extensions which can be "got around" in 1131 using the principles of 1499 - e.g. OA Control.

The Tank Control System.


Union
Tap for Future Sensor
Future Equipment and Piping
MF1 = Mass Flow Meter
PIT = Pressure Sensor
Note: Tanks to be drilled and tapped for future connections, then plugged.
PIT1
MF1
PIT2
U
T
S
Bypass Valve
Bypass Line
Low Pressure Drain Line for Overflow
H_2O

CONCLUSION.

- **IEC 1131-3 is the only practical way forward but be aware of IEC 1499.**