UBM

DesignNews.com

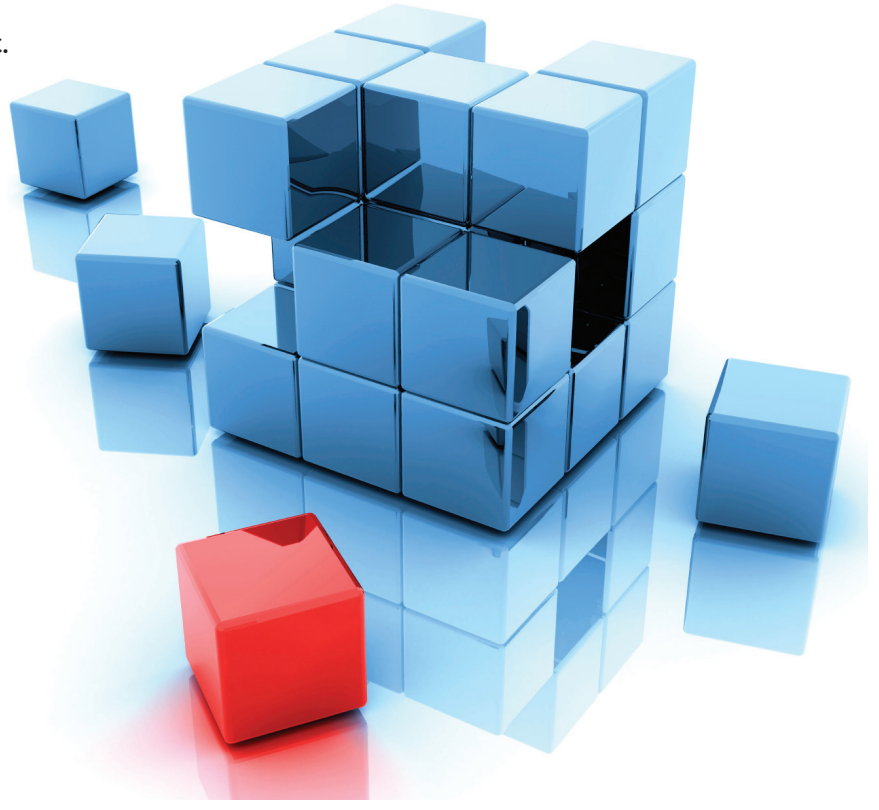
Accelerating Engineering Innovation

ELECTRONICALLY REPRINTED FROM APRIL 2011

IEC61131-3 as an Integration Tool

Understanding the benefits of IEC61131-3 and why its programming methods are becoming preferred for mechatronic designs.

BY MATT PELLETIER, YASKAWA AMERICA INC.



SOURCE: ISTOCKPHOTO.COM/RZDEB

IEC61131-3 may sound like a complex specification, but in practical use it comes down to just a few concepts that promote good programming techniques and code reusability.

The most fundamental of IEC concepts is program execution behavior. Each controller or PLC in an automation system is called a resource, and each resource executes one or more tasks. Multiple tasks may be updated at different update rates. The IEC code in each task is further divided into Program Organization Units (POUs), which are executed within the task in the order specified. The POU may exist as a program type and run in the associated task. Alternatively, it may be a mathematical function type POU or a Function Block type of POU, both of which run when used within the program type of POU.

Much like the familiar programming subroutine, customized functions and function blocks allow the programmer to wrap up a particular piece of code for easy organization, for reuse, for protection of intellectual property, and/or for revision control.

VARIABLES AND DATA TYPES

The terminology and mechanics of variables, data types and literals is defined in IEC61131-3 — and these definitions must be followed.

The character set is limited to alphanumeric and the underscore with no spaces or special characters allowed. Programmers must therefore get used to working with variables with names like "FeedSpeed" or "Feed_Speed." Capital letters are preserved for display purposes, but capitalization differences are ignored in program execution.

Most programmers are likely to find a new set of standard terms to get used to in IEC61131-3. Besides the previously mentioned acronym POU, a constant hard-coded value is called a literal. A variable with one bit of information is called a BOOL. And when there's a decimal point, it's called a REAL or LREAL data type.

This data type terminology manifests itself in both literals and variables. A literal, hard-coded value of "5" is expressed in the code with a data type prefix such as LREAL#5.0, INT#5, WORD#5, or TIME#5s.

This provides a visible double-check for the programmer to avoid unintentional mixing of data types. The same holds true with variables.

A variable is assigned a data type, and the initial value associated with that variable must match the data type. So if a variable is named "FeedSpeed" and is given the data type LREAL, then it can have an initial value of 5.0, but not 5 or 5s.

Often in programming, the output of one part of the code is in one data type, and it must be used in another part of the code as another data type. For example, an input value may be received from an HMI as a WORD#5, to be written to the variable "FeedSpeed," with the LREAL data type. This requires a data conversion step.

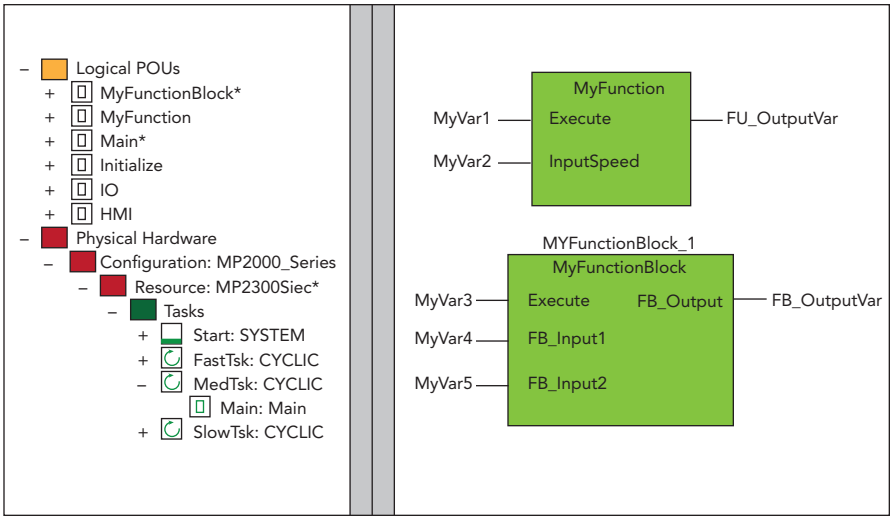
IEC61131-3 good programming practice insists that any conversion step not be hidden from view, so the programmer must specifically define the conversion step. Fortunately, data type conversion function blocks between any two combinations of the 20 standard data types are provided so that data type conversion can be tightly controlled.

MULTIPLE PROGRAMMING LANGUAGES

IEC uses and combines five of the most common programming languages: Ladder Diagram (LD), Structured Text (ST), Function Block Diagram (FBD), Instruction List (IL) and Sequential Function Chart (SFC).

The programmer decides the best language for a particular POU based on personal preference and on the purpose of that particular POU. A function block POU written in one language can be used within another POU of a different language.

For example, a complex mathematical formula may be best programmed using the ST language. I/O logic is usually more easily programmed in LD, while FBD may be best for a motion control sequence. SFC is effective to control the states of the machine. Under the IEC61131-3 standard, the strengths of each language can be used simultaneously.



SOURCE: YASKAWA

A Function POU and Function Block POU run in a Program POU titled "Main."

HIGH-LEVEL PROGRAMMING

IEC61131-3 goes beyond fundamental programming techniques — defining several high-level programming concepts such as Enumerated Types, Arrays and Data Structures.

Enumerated Types give a name to a number so that when a numerical selection is required, the name can be used in its place. This makes the code more intelligible and aids in troubleshooting.

For example, the PLCopen function block MC_MoveAbsolute includes a direction input so that rotary machines can locate the position in four different ways, including "shortest_way." Instead of assigning the direction input to 0, 1, 2, or 3, the "shortest_way" text can be used.

Arrays are familiar to many programmers and allow a large amount of data to be accessed by indexing the address within the array. The data structure is a critical programming tool for wrapping up an assembly of different types of data into one variable.

One example of the usefulness of data structures can be found in Yaskawa's PLCopen Toolbox user library for the MP2000iec series controllers. Each axis in a system will typically have the same data associated with it such as a jog speed, run speed, position, parameters, etc.

Traditionally, the programmer must create separate variables for each of these axis-specific pieces of data, a tedious task. But with IEC61131-3, it's possible to define a data structure that contains all of the axis-specific data once and for all, creating a type of template.

The programmer then creates a single variable (FeedAxis) for the axis and chooses this new structure as the data type (Axis Struct). This provides the equivalent result to having created tens or even hundreds of variables all at once.

The data can be accessed as if it were a variable using the dot notation such as FeedAxis.JogSpeed or FeedAxis.Prm. Additional variables with this same data structure can be defined for the other axes in the system, quickly creating an orderly and consistent definition of variables associated with each axis.

The IEC61131-3 standard provides a host of benefits to programmers of PLCs and controllers for integrating mechatronic applications. Although it takes some time up front to learn specific terminology and techniques, overall efficiency is increased. The resulting programs can then be used and interpreted by others familiar with the standard, creating common ground for other programmers and for maintenance and operations personnel.

Matt Pelletier is senior product training engineer at Yaskawa America Inc.

For more information, go to http://dn.hotims.com/34931-500.