# PROGRAMMING OF CONTROL SYSTEMS FOR INDUSTRIAL PROCESSES

**Delia Ungureanu, Francisc Sisak, Ion Truican**

Automatics Department, "Transilvania" University of Brasov, M.Viteazu Street, no.5, 500174, Brasov, Romania, phone/fax: +40 0268 418836,
delia@deltanet.ro, sisak@unitbv.ro, itruican@yahoo.com

*This paper contains some aspects of application programming, based on control systems of industrial processes. The aspects on which we focus are defining types of data specific to the application and adding user functions to the standard function set. We discuss about automation systems belonging to two categories: real-time controls and automation systems. These systems are designed to aid facilities managers and owners to reduce operating, maintenance, and management costs. To efficiently use the hardware components, is very important how the programming of applications is done, so we analyze some aspects of programming a control system for industrial processes. When we design a program for a complex application it is very useful to define structured data types to process jointly several variables even if they have different data types and we can supplement the function block library by specific function blocks. In order to define an automation strategy, we will add new functions for specifically needs. User function blocks can be called from all programming languages: IL, LD, FBD and SFC. We defined some simple functions and based on them we try to define complex functions, such as the implementation of the algorithm for a Smith predictor, anticipation algorithm, functions for neural nets, etc.*

**Keywords:** programming, data types, control systems, Smith Predictor.

## 1. INTRODUCTION

The current automation systems are in two categories:

- *real-time* controls and automation systems, such as direct digital control (DDC) and automation systems, power plant controls and automation systems, industrial control systems, online utilities metering systems, etc.
- *data processing systems*, such as maintenance management systems, production control systems, various office automation systems, engineering systems, drafting and project management systems, and other specific systems.

The above systems are designed to aid facilities managers and owners to reduce operating, maintenance, and management costs. Thus, to efficiently use the hardware components, is very important the manner of applications programming.

That in way we analyze some aspects of programming a control system for industrial processes. We use as support a Freelance 2000 system offered by ABB Automation Systems GmbH. Freelance 2000 is a scalable control system which is divided into an operator level and a process level and the communication between them:

*A. The operator level* - The operator stations use standard PC hardware, either conventional or industrialized, running the Windows NT operating system. One engineering station and several operator stations can be installed at the operator level. The engineering station is used to configure and commission the system. It can be either a standard desktop PC or laptop. The operator level PCs can also be used for configuration tasks.

*B. The process level* - At the process level, the system can contain one or several process stations or Field Controllers that can be extended with I/O units. You have the option of configuring the process station either redundantly (CPU redundancy, field bus module redundancy) or without redundancy. Modular plug-in input/output modules in racks are used in accordance with the type and quantity of process signals.

*C. System communication* - The system employs two industry standard buses. A station bus (based on a CAN bus) is used for the field area and is characterized by enhanced ruggedness and exceptional data security. The operator and process levels communicate with each other via the system bus (based on Ethernet) with TCP/IP protocol, where you can choose between various transmission media such as coaxial cable or fiber optic.

Subsystems, such as weighing systems, PLCs and others, can be connected with Freelance 2000 via the various interfaces and interface modules and a higher level of process control is possible with other process control systems.

The problem on which we focus is building the application program.

The application programs can be built with graphic configuration with high-performance editors in programming languages according to IEC 61131-3: function block diagram (FBD), ladder diagram (LD), instruction list (IL), sequential function chart (SFC).

Among the facilities offered in the programming of the application, which brig in a lot of flexibility, are the possibility of defining data specifically for the application and the completion of the standard function set with user functions.

## 2. USE OF STRUCTURED DATA TYPES

Variables are used for storing and processing information. Various different data types are available in the system, e.g. byte, word, integer, real, data&time.

Along with the standard data types, user-defined structured data types are also available when declaring a variable.

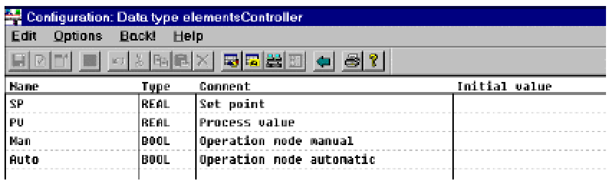
System variables are created every time a new resource, process station, Field Controller or gateway is added. The status details for the resource are stored in these variables. The system enters all the variables for a project in the *variable list*.

Default values can be assigned to each variable and to the separate elements of a structured variable. These values are assumed after a cold start, or when a station is initialized.

To enable several variables to be processed jointly even if they have different data types, it is possible to define new, *structured data types.*

Application-specific data types can be created, i.e. defined in addition to the structured ones, with the aid of the specific editor. These user-defined data types are included in the data type selection list and can be selected like standard ones. In this way a series of data can be transmitted via a structured variable.

To define a new data type we must insert a new data type name into the list of structured data types. This option is activate from menu (System-Structurated data type-Edit-Insert a new data type)(fig.1).


Configuration: Data type elementsController

Edit Options Back! Help

| Name | Type | Comment | Initial value |
|---|---|---|---|
| SP | REAL | Set point | |
| PU | REAL | Process value | |
| Man | BOOL | Operation node manual | |
| Auto | BOOL | Operation node automatic | |

Fig.1 Creating data type components

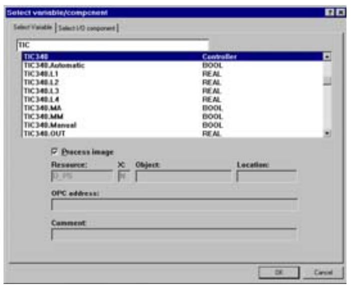
The components of the new defined data type can be entered with:

- *Name* – The name of structured data type (max. 16 characters);
- *Type* - Data type as BOOL or REAL;
- *Comment* – Comment;
- *Initial value* - A default initial value may be entered. When a warm start is made, this value is used as the structured data type for all variables which have that data type.

Corresponding variables may be adopted using the new data type (e.g. *Control*). For example, several controls of the same type may be provided with variables just by opening a variable, e.g. *TC120_V*, with the new *Control* data type.

In the example below the new variable TC120_V is assigned the structured data type *Control*. The following components of variable *TC120_V* are thus available (fig. 2):

| | | |
|---|---|---|
| *TC120_V.X* | REAL | Track value |
| *TC120_V.W* | REAL | Set point |
| *TC120_V0.Y* | REAL | Output value |
| *TC120_V.MM* | BOOL | Manual |
| *TC120_V.MA* | BOOL | Automatic, etc. |


Select variable/component

Select Variable | Select I/O component

TIC

| | |
|---|---|
| TIC340 | Controller |
| TIC340.Automatic | BOOL |
| TIC340.L1 | REAL |
| TIC340.L2 | REAL |
| TIC340.L3 | REAL |
| TIC340.L4 | REAL |
| TIC340.MA | BOOL |
| TIC340.MM | BOOL |
| TIC340.Manual | BOOL |
| TIC340.OUT | REAL |

Process image

Resource: X: Object: Location:

OPC address:

Comment:

OK Cancel

Fig. 2 Creating data type components

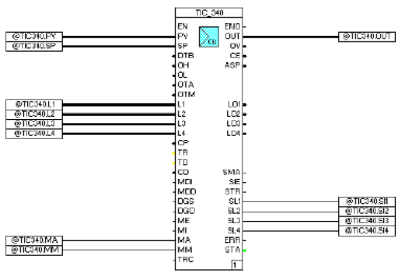


Fig. 3 A variable of structured data type

To use a structured data type in a program, we select a read or write variable in a function block diagram.

In the fig. 3 various components of the structured variable TC120 (data type control) have been used.

## 3. ADDING USER FUNCTIONS

The scope of functions and function blocks provided by the Freelance 2000 process station corresponds to the basic supply defined in IEC 61131-3, in addition to numerous other tested functions and function blocks.

Program execution in the process station is based on a task-oriented, real-time multitasking operating system, leading to a flexible strategy for processing programs.

Available function categories are: analog value processing, digital value processing, loop control, logic control, logic functions, monitoring, acquisition, arithmetic functions, Modbus functions.

While designing the station and during configuration, the processing capacity and speed of the process station can be easily adapted to the demands of the automation task. They are accommodated in a function block library and can be supplemented by user-specific function blocks.

In order to define an automation strategy, we will add new functions for specifically needs.

The *user function block (UFB)* facility makes it possible for users to create their own function blocks. This means that function blocks can be designed to meet the specific needs of particular sectors.

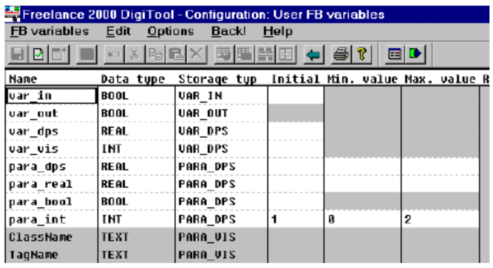
The specifications of a *user-created function block class* will determine the functionality and the appearance of a function block. As such, the class encompasses the user-created program in its entirety, with its functions, function blocks and variables, as well as the faceplate and the parameter mask. Every user function block class receives a class name which can be freely chosen and with which the block can be called from inside other programs.

A class itself cannot be run in a process station. For execution, an instance of the class must first be created. There can be as many instances of a class as desired.

In a function block instance, all local and output variable values are retained from one execution to the next. This means that the function block instance has an internal state, with the result that the same inputs need not always result in the same outputs.

Changes to user function blocks are made to the function block class and affect all function block instances of that class in place. If new pins are added to a block, the function block instances must be replaced and the new pins accounted for.

Any user function block which has already been declared can be used in the declaration of another user function block.



| Name | Data type | Storage typ | Initial | Min. value | Max. value |
|---|---|---|---|---|---|
| var_in | BOOL | VAR_IN | | | |
| var_out | BOOL | VAR_OUT | | | |
| var_dps | REAL | VAR_DPS | | | |
| var_vis | INT | VAR_DPS | | | |
| para_dps | REAL | PARA_DPS | | | |
| para_real | REAL | PARA_DPS | | | |
| para_bool | BOOL | PARA_DPS | | | |
| para_int | INT | PARA_DPS | 1 | 0 | 2 |
| ClassName | TEXT | PARA_VIS | | | |
| TagName | TEXT | PARA_VIS | | | |

Fig. 4 Create a user function block class

A user function block class is made up of the following components: interface, parameter list, text list, program, faceplate.

The variables used in the user function block program must be entered in the Interface editor under User FB variables (fig.4). The parameter mask is created and the text list administered in the Interface editor.

All standardized function blocks and all functions are available in the configuration of a user function blocks.

The desired variables are selected from this list of variables. It is also possible to make entries directly in the input or output fields, but the variable entered must exist in the user FB variables list.

Existing user function blocks can be used in other user function blocks. Recursive calling is not supported for user function blocks.

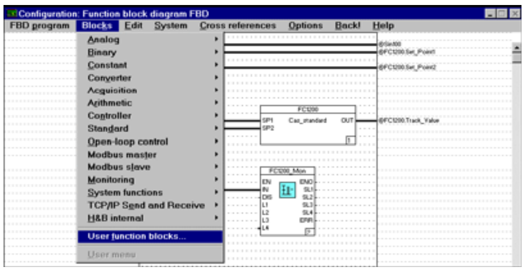


Fig. 5 Create new user function block instance

A function block instance is created by choosing a class out of a list of user function block classes (fig. 5). Instances can only be created from classes which have passed the plausibility check.

After the user function block has been selected from the list, it is positioned in the program and the variables can be connected.

There are some limitations regarding the standardized function blocks that can be used. Particular exceptions are function blocks having an equivalent in DigiVis (such as trend acquisition blocks) or accessing special hardware (such as the Modbus interface block).

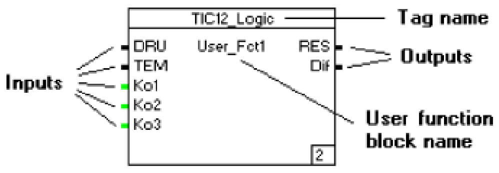


Fig. 12 Example of user function block – FBD/LD program

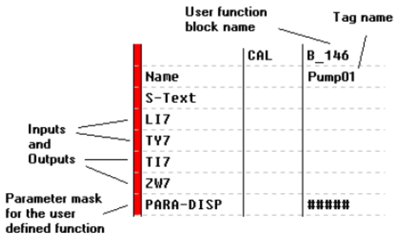


Fig.13 Example of user function block – IL program

## 4. Example of User Function. Smith Predictor

We will now analyze a function which transposes the algorithm for a Smith predictor.

The Smith Predictor was developed for dealing with dead-time problems common to industrial process where feedback from the processes is continuous, i.e. in each control cycle a new delayed feedback is available.

The advantage of using the Smith predictor is that the according of the controller (usually PID) is done as if the process wouldn't have dead times, so the amplification can be increased and the answer can be accelerated. The output of the predictor is the result of adding up the process output and the output of the delay free model, of which we subtract the output with the delay model. The output of the algorithm is directly connected to the process input of the PID algorithm from downstream. When there is no time delay in the process, the variable input of PREDICTOR goes direct from the process to the process input of the precedent PID. The internal model used is a order 2 dynamic with pure time delay (dead time).

The long delay time will be formed by a cascade chain of delays obtained with an algorithm named TRANSPORT. The number of delay steps is determined by the dead time divided by the sampling time, and rounding the result to an integer value.

The structure of the controlling system can be represented by fig. 6 In the dotted rectangle we have the PREDICTOR algorithm.

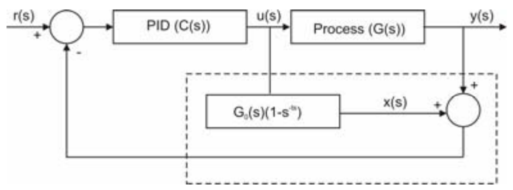


Fig.6 The structure of the control system with PREDICTOR

where : *y(s)* – process variable ;
*u(s)* – control system output;
*r(s)* – reference;
*t* – dead time.

In the ideal case:

$$G(s) = G_{0(s)}e^{-ts} = \frac{K \cdot e^{-ts}}{T_2 s^2 + T_1 s + 1} \qquad (1)$$

And the transfer function of the control loop is:

$$Gc(s) = \frac{y(s)}{r(s)} = \frac{C(s) \cdot G_0(s)e^{-ts}}{1 + C(s) \cdot G_0(s)} \qquad (2)$$

By creating a bilinear transform for the discretisation of the continuous model of the process, the final output of the predictor, OUT, for any time sample K is:

$$OUT(k) = A_{1u}(k-1) + A_{2x}(k-2) + A_{3u}(k) + A_{4u}(k-1) + A_{5u}(k-2) + A_{6u}(k-d) + A_{7u}(k-d-1) + A_{8u}(k-d-2) \qquad (3)$$

where: *d* = int(t/sampling time);
$A_{is}$ = intermediate components

A controlling configuration example can look like in fig. 7.

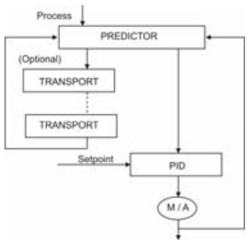


Fig.7 Example of configuration using PREDICTOR

## 5. CONCLUSIONS

Besides the suitable hardware, a high-performance, user-friendly and convenient engineering tool is indispensable for simple planning, programming, testing and commissioning of an automation application.

In this paper we discussed about some aspects of programming a process control system. We used the facilities offered by the system to define new data types and new functions specifically to our applications. We defined some simple functions and we try so to define complex functions, such as the implementation of the algorithm for a Smith predictor. We will continue adding new functions, e.g. anticipation algorithm, etc.

## 6. REFERENCES

[1] Boed V., *Networking and of Facilities Automation Systems,* CRC Press LLC, 2000 Corporate Blvd., N.W., Boca Raton, Florida

[2] Freyer H., H. Spalt, *ABB Automation System Basic Course N110*".

[3] *** *S800 I/O Profibus FCI, Memory Maps for CI830,* Reference Manual.

[4] *** *Freelance Select. Short Instructions Installation and Commissioning*

[5] ***„*http://www.abb.com/product_guide/control_systems/*", Freelance 2000 Product Guide"