PROGRAMMING LANGUAGES FOR DDC SYSTEMS

An examination of the current and future state of programming languages and the various options available for programming DDC panels

By THOMAS HARTMAN, PE,
The Hartman Co.,
Seattle, Wash.

A Star Trek movie finds Capt. Kirk, through no fault of his own, mysteriouslv transported back to the primitive present. To correct the situation and return to his proper place in time, he solicits the use of a computer. When shown a powerful computer, one of Kirk's lieutenants immediately tries to initiate a dialog by commanding "Computer!" After a moment of uncomfortable silence, the operator points out the mouse, which Kirk's lieutenant raises to his mouth and again tries to command the computer verbally.

I was reminded of that scene recently at a presentation of a new DDC (direct digital control) product. The vendor was nervously presenting what he thought were advanced programming features to a group of experienced DDC users. The questions and criticisms flew fast and furiously. Finally the exasperated rep asked, "Well, how do you want to program DDC systems?" One of the participants responded immediately, "We want to tell the system what we want it to do, and we want it to understand and do it."

I never had a chance to ask that fellow if he got this idea from the movie, but I suspect it wouldn't make any difference. Virtually everyone who uses computers believes they could be vastly more friendly to use than they are. DDC operators are no exception. Until systems operate as the Star Trek crew expected, the pressure for improvements will continue to be very strong indeed.

In the last few years, the DDC industry has learned a great deal about performance and the elements of successful programming languages. A number of ideas have been tried and many have been very successful. In this article we will examine the current state of programming languages for DDC systems, what the various options are for programming DDC today, and the benefits and drawbacks of each. Finally, we will look to the future and suggest a path for continued improvement of DDC system programming languages.

The beginning

In the early days of computer-based building control, most programming languages offered very few features and even less flexibility. As a result, the notion developed that the system operator should be more a specialist in computers than in HVAC systems. Many systems were supplied with programs written at the factory. These programs were provided in a low-level assembly type language that allowed the operator only a few of what we call *hooks* into the system. For example, typical programs permitted the operator to define occupancy schedules and adjust certain set points, but the sequence of control was fixed and could only be changed by recompiling the program, which usually had to be done off site. Problems that could not be adjusted away with the built-in hooks required elaborate schemes to correct. Operators who were very knowledgeable in the operation of the computer could sometimes adjust certain database parameters, fooling the system into performing more satisfactorily.

However, users became very frustrated by the inflexibility of these systems. Building control problems that seemed quite simple and straightforward often required elaborate measures and a computer specialist to solve. A flurry of activity took place by manufacturers, users, and the building design community to improve the success of computer-based building control systems. Some of the initial solutions proposed (such as tri-services specification) failed because they tried to treat the symptoms and not the cause. But when the dust settled early in the 8Os, two new approaches to building control programming were being offered that allowed operators who were not skilled in computers to support building automation systems more effectively than ever before.

Line *programming*

One of the approaches to improved programming capabilities offered by some manufacturers was the ability to write sequences of operation in standard line-program formats. Line programming had been employed for many years in the general computing industry. The formats of these DDC languages look very much like the high-level general computing languages (such as BASIC) except that certain additional functions are added to permit the language to issue start and stop commands, control outputs to PID (proportional/integral/derivative) algorithms, tie into occupancy schedules, etc. Some languages are compiled and some interpretive; but all offer similar flexible control capabilities, and they can be easily developed and altered on site by the system operator. A sample line program in the form our firm typically employs to calculate the supply air set point for simple fan systems is shown in Fig. 1.

Some line-programming languages contain all the features of powerful high-level languages and include formatting features that make programs written in these languages quite readable. Some newer releases also include additional features, such as full-screen editors and on-line error checking, permitting operators to view, edit, change, and debug virtually any control sequence quickly and easily.

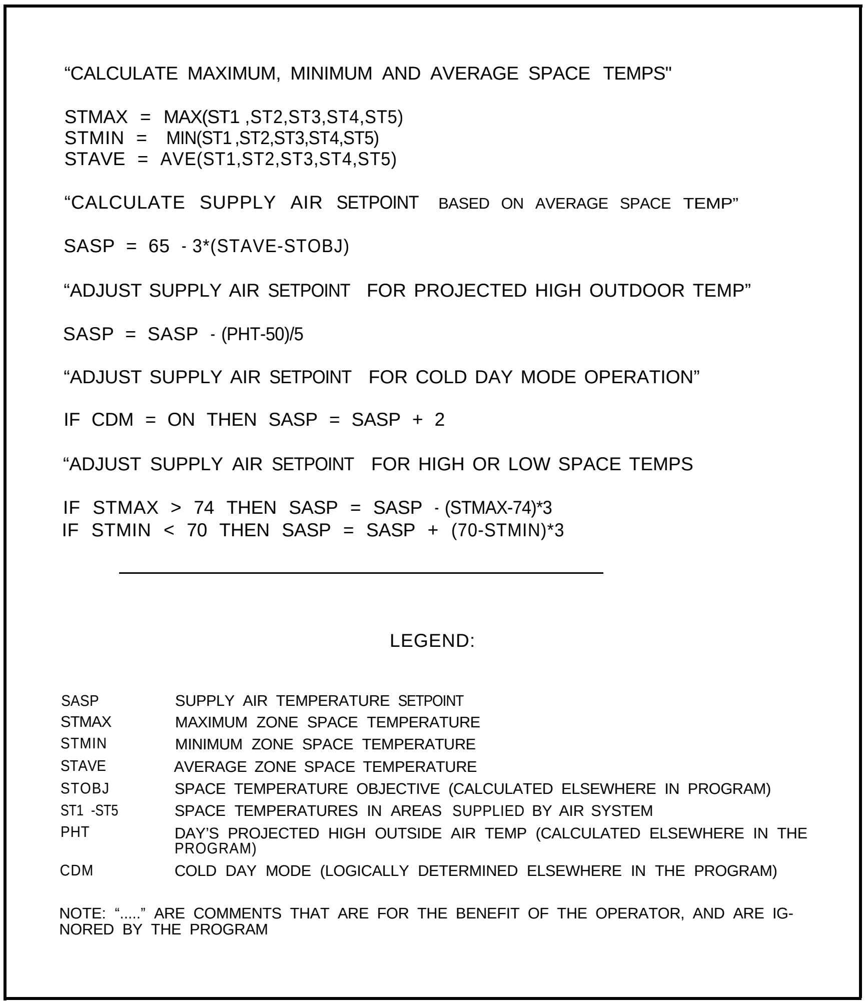
It is important to note, however, that there are wide variations among line-programming languages. Some are crude, inflexible, and very difficult to use, though their suppliers still claim they provide high-level line programming. As with other features of DDC systems, equals in languages do not exist. Users and consultants should make themselves knowledgeable about the features of any programming language to be supplied with a system before it is purchased.

The primary advantage to line programming rests in its power and flexibility. Line-program functions have already proved themselves in solving diverse general computing problems. With the addition of a few special functions for building control, system designers and building operators find they have the tools they need to develop just about any control sequence(s) for particular HVAC system control requirements.

Another advantage of some line programs is that they are self-documenting. When a designer or operator determines that a program change is necessary, a printout of the program provides an accurate and fairly readable description of the new control sequence. And because line programs are in the form of general computing languages, many operators have already had some experience with this type of language at school or home.

The primary disadvantage cited for line programming is that some operators have trouble understanding and writing line programs without specific training. Indeed, some of the line languages have

continued on page 7.1



```
"CALCULATE MAXIMUM, MINIMUM AND AVERAGE SPACE TEMPS"

STMAX = MAX(ST1 ,ST2,ST3,ST4,ST5)
STMIN =  MIN(ST1 ,ST2,ST3,ST4,ST5)
STAVE = AVE(ST1,ST2,ST3,ST4,ST5)

"CALCULATE SUPPLY AIR SETPOINT BASED ON AVERAGE SPACE TEMP"

SASP = 65 - 3*(STAVE-STOBJ)

"ADJUST SUPPLY AIR SETPOINT FOR PROJECTED HIGH OUTDOOR TEMP"

SASP = SASP - (PHT-50)/5

"ADJUST SUPPLY AIR SETPOINT FOR COLD DAY MODE OPERATION"

IF CDM = ON THEN SASP = SASP + 2

"ADJUST SUPPLY AIR SETPOINT FOR HIGH OR LOW SPACE TEMPS

IF STMAX > 74 THEN SASP = SASP - (STMAX-74)*3
IF STMIN < 70 THEN SASP = SASP + (70-STMIN)*3
```

LEGEND:

SASP	SUPPLY AIR TEMPERATURE SETPOINT
STMAX	MAXIMUM ZONE SPACE TEMPERATURE
STMIN	MINIMUM ZONE SPACE TEMPERATURE
STAVE	AVERAGE ZONE SPACE TEMPERATURE
STOBJ	SPACE TEMPERATURE OBJECTIVE (CALCULATED ELSEWHERE IN PROGRAM)
ST1 -ST5	SPACE TEMPERATURES IN AREAS SUPPLIED BY AIR SYSTEM
PHT	DAY'S PROJECTED HIGH OUTSIDE AIR TEMP (CALCULATED ELSEWHERE IN THE PROGRAM)
CDM	COLD DAY MODE (LOGICALLY DETERMINED ELSEWHERE IN THE PROGRAM)

NOTE: "....." ARE COMMENTS THAT ARE FOR THE BENEFIT OF THE OPERATOR, AND ARE IGNORED BY THE PROGRAM

1 **Line program for calculating supply air temperature set point.**

continued from page 70

long lists of rules regarding their use, and operators are often frustrated when they cannot easily make occasional program changes. However, line-based languages with fewer rules, which permit the use of comments and special formatting, are usually supported successfully by building operators.

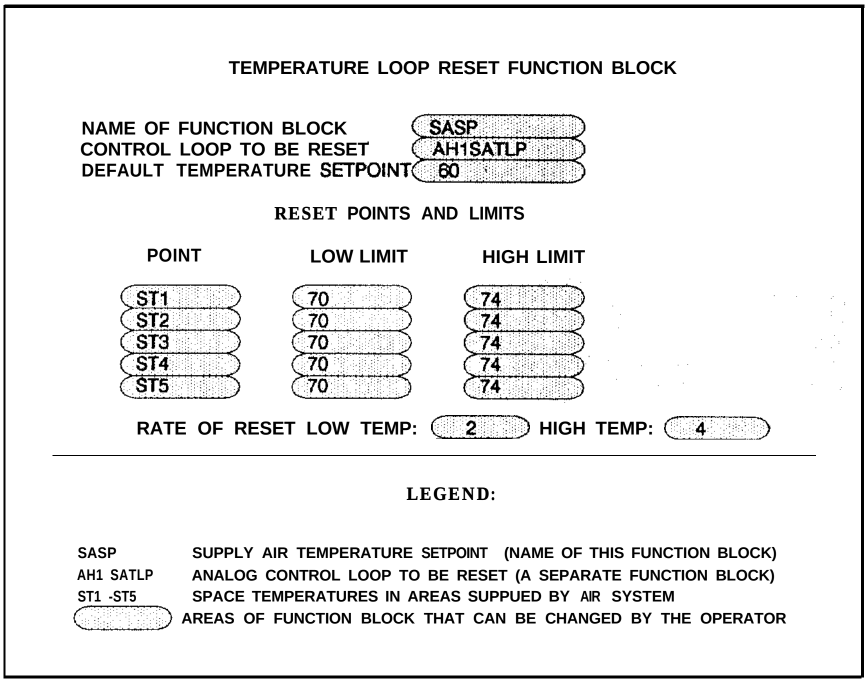
Another disadvantage of line programs is that software development for typical projects can become time consuming by requiring entire programs to be rewritten for multiple systems even though they all may operate very much the same. Fortunately, certain copying features and editing aids mitigate this disadvantage in the more advanced line program languages.

Function-block programming

A second approach taken to improve success with applications software involves refining the preprogrammed approach to give it some additional flexibility in a form that is structured specifically for typical HVAC applications. The manufacturers that adopted this approach decided to break down the factory programmed applications into small program blocks that can be linked together and have parameters assigned by the designer or operator. By assembling these preprogrammed blocks in various combinations and providing some flexibility in assigning points and parameters, manufacturers believe they can satisfy most typical DDC applications while maintaining a simple and easy-to-use program format.

Fig. 2 is a sample function-block program. This function block provides space temperature reset of the supply air temperature set point of a simple fan system. Some in the industry call this “fill in the blanks” programming because only a limited number of fields in each program block need to be entered.

The primary advantage for function-block programming is its simplicity in standard HVAC applications. Indeed, if the control sequence happens to call for the exact functions provided by the function blocks included with the systern, the programming effort is very easy. The disadvantage of the function-block approach is that whenever sequences are required that do not match available function blocks, the programmer must employ custom blocks or employ blocks for purposes other than those for which they were developed. This usually results in more complicated programs and reduced HVAC system performance.



TEMPERATURE LOOP RESET FUNCTION BLOCK

NAME OF FUNCTION BLOCK SASP
CONTROL LOOP TO BE RESET AH1SATLP
DEFAULT TEMPERATURE SETPOINT 60

RESET POINTS AND LIMITS

POINT	LOW LIMIT	HIGH LIMIT
ST1	70	74
ST2	70	74
ST3	70	74
ST4	70	74
ST5	70	74

RATE OF RESET LOW TEMP: 2 HIGH TEMP: 4

LEGEND:

SASP	SUPPLY AIR TEMPERATURE SETPOINT (NAME OF THIS FUNCTION BLOCK)
AH1 SATLP	ANALOG CONTROL LOOP TO BE RESET (A SEPARATE FUNCTION BLOCK)
ST1 -ST5	SPACE TEMPERATURES IN AREAS SUPPUED BY AIR SYSTEM
▨	AREAS OF FUNCTION BLOCK THAT CAN BE CHANGED BY THE OPERATOR

2 Function-block program for supply air reset.

As a result of the relative advantages and disadvantages of the two programming approaches, line programming-based systems are usually far more effective in high-performance building applications that require more in-depth control strategies. Systems with function-block programming are generally limited to applications employing simple or traditional pneumatic strategies.

To see the differences in the two approaches, consider the programs in Figs. 1 and 2. The line program in Fig. 1 and the reset-function block in Fig. 2 are both intended to reset the supply air temperature set point of a simple fan system as space conditions change. However, the line program in Fig. 1 also includes outdoor weather factors and could easily be changed to accommodate different relationships or additional factors-all in this one program. By contrast, the function block cannot implement a number of the factors employed in Fig. 1. Function-block programs usually do permit linking blocks together for additional factors in calculations. However, linking causes the calculation to be broken into a number of small relationships that do not appear together on a single screen and are therefore difficult for the operator to follow.

DDC language trends

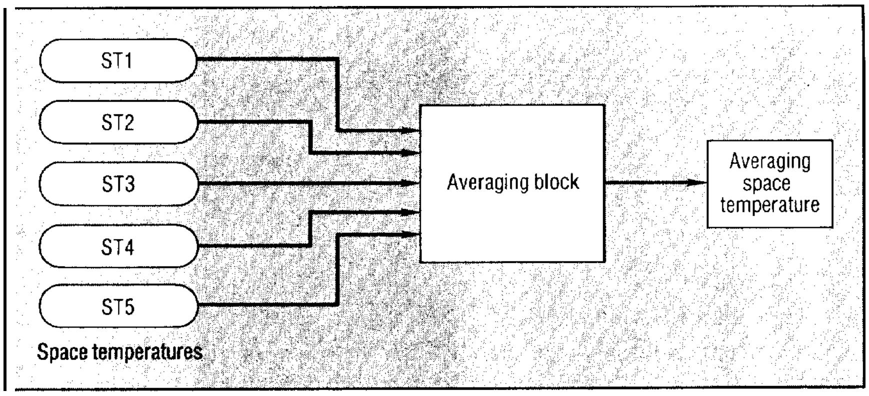
Because our firm focuses its efforts on high-performance building-control applications, we have favored DDC systems employing line programming. Line programs offer greater power and expanded functions that are necessary in our high-performance DDC projects. With our experience, we have long understood the problems and shortcomings of line-programming languages in certain applications.

Over the years, we have worked with users and manufacturers to improve the ease with which line programs can be applied to building control. We promote the concept of output-oriented programs wherein every calculation and command that directly affects an output is installed in one (and only one) section of the program. With output-oriented programming, the operator can quickly trace any control path and adjust the program easily when a point or mechanical system is not operating as desired.

In 1988, our firm released a guideline for line-based program languages called the operators' control language (OCL). The OCL guide was intended as a functional specification for features our firm and our clients desired in line-program languages. A discussion of the merits of good operators' control languages appeared in the September 1990 issue of *HPAC*. "OCL Spells Freedom," by Ken Sinclair, concluded that advanced line program-based DDC systems offer greater flexibility and are easier to use than other approaches.

Our firm's experience agrees with Sinclair's conclusion. Building operators are easily trained in the operation of most line programs and are able to use many of the advanced features to make their maintenance and troubleshooting duties easier to perform. This is possible because most line program-based systems have recently provided updates that simplify the operation of these languages by consolidating and simplifying their rules of application. These improvements are making line programs simpler to apply without compromising the power and flexibility that are inherent in their architecture.

Meanwhile, to improve their range of applications, function block-based systems continue to add to their libraries of blocks. But large libraries reduce their simplicity, the primary advantage of function-block programs. These trends are eroding the few advantages that function-block programs can offer when compared to the more advanced line programs.



3 Typical graphic program expression.

New approaches

The inevitable demand for systems that operate as Capt. Kirk expected continues to drive manufacturers to develop innovative approaches to building-control applications programming. Several DDC system manufacturers have committed themselves to releasing DDC systems that employ a new approach to the applications programming language-graphics programming. This approach utilizes the powerful graphics capabilities of modern PCs to permit the programmer to sketch out a flow chart for the control sequence desired. A program in the PC is then employed to translate this sketch into a program (usually written in a high-level line language), which is down-loaded to the selected DDC stand-alone panel.

An example of a screen containing a graphics program is shown in Fig. 3. Note that the entire figure would be built by the operator with system points, variables, and a library of mathematical and logical functions. If the operator desired an additional space temperature for the calculation in Fig. 3, he could easily add it by choosing the appropriate system point and sketching a connection to the calculation block.

The idea behind graphics programming is to find a way to provide the power and flexibility of line programming with the simplicity of function-block programming. Essentially, the programmer can develop custom function blocks to meet the needs of any particular project application. The idea is enticing, but there are some potential problems with the concept, including:

- *Program execution uncertainty--Once* a programmer has developed a graphic, a translator program has to be employed to convert the picture into an executable program to be down-loaded to the appropriate stand-alone panel. Experienced programmers have long realized that there are some special considerations required when writing programs to ensure that they execute as expected. For example, assume the following simple sequence of operation is desired to start supply and return fans.

If the weekly schedule is on, start the return fan, and after a 30-sec delay, start the supply fan.

Fig. 4a shows how a line program can be written to execute that logic sequence. If the data point RETURN_FAN is turned on only after the entire block is executed, the program will execute properly. However, if RETURN_FAN is turned on as soon as that line is executed, it is clear that SUPPLY_FAN may be started an instant after RETURN_FAN is started. A better way to write the program is

CONTROL SEQUENCE: If the weekly schedule Is on, start the return fan, and after a 30 second delay start the supply fan

FIGURE 4a: UNCERTAIN EXECUTION ORDER

```
 IF WEEKLY_SCHEDULE = ON THEN BEGIN
    DOEVERY. SO SEC
       START RETURN_FAN
       IF RETURN_FAN ON THEN START SUPPLY_FAN
    ENDDO
END.
```

FIGURE 4b: RELIABLE EXECUTION ORDER

```
 IF WEEKLY-SCHEDULE = ON THEN BEGIN
     DOEVERY 30 SEC
       IF RETURN_FAN ON THEN START SUPPLY-FAN
       START        RETURN_FAN
    ENDDO
END
```

LEGEND

WEEKLY_SCHEDUIE	WEEKLY SCHEDULE (SET UP ELSE WHERE)
DOEVERY 30 SEC	DO LOOP THAT IS EXECUTED ONCE EVERY 30 SECONDS
RETURN-FAN	DIGITAL OUTPUT THAT STARTS THE RETURN FAN
SUPPLY-FAN	DIGITAL OUTPUT THAT OPERATES THE SUPPLY FAN

NOTE: The underline Character Is used to join words to make single terms for point names or variables. This follows normal programming convention.

4 Line programs to execute simple tan start sequence.

shown in Fig. 4b. Although the order of items in the program is reversed from what we might expect, it is clear that SUPPLY-FAN will never be turned on less than 30 sec after RETURN-FAN.

While programmers can expect translators to offer some degree of protection against such translation errors, they can never be entirely certain that the translator is not responsible for operational errors. What is a programmer to do if a program does not appear to be executing as pictured in the graphic? The programmer will inevitably be required to inspect and troubleshoot the translated program if the error cannot be found by reviewing the graphic screens. The line program that is developed by the translator is likely to be very difficult to review because such programs do not have the form and logical flow that programmers typically provide in their line programs. Nor are such programs likely to have comments or formatting devices that make them easy to read.

Graphic programming may seem to be much more straightforward than line-based software, but any debugging effort can become much more complicated, especially if the operator desires to write more complex control algorithms that make the fullest possible use of the energy and comfort-enhancing capabilities of DDC.

- *Display limitations*-Another problem with graphic representation of control programs is they are bulky to display. Note that the averaging calculation of Fig. 3 requires only a single line to represent in the line program of Fig. 1. When several pages or more of graphics are required to represent a control sequence, the sequence can become very difficult to review because the operator cannot look at the whole program at once.

Recently, a client of ours experimented with a prototype of a new graphic program-based DDC system. He translated a line DDC program used to control his building's air systems to see if the graphic representation offered any advantages. To his surprise, he found the graphic program required eight screens of graphic representation and seven pages of constants and gains to duplicate a line program that occupied (with comments) only one and one-half pages. The resulting program was far more difficult for him to review than the original line program.

- *Operator interface cost*-Most line-based programming languages can be operated directly or over phone lines with a simple PC and low-cost software. However, the complex hardware and software required to create, test, translate, and compile graphics programs can add substantially to the cost of each such terminal. Many users have developed system-support mechanisms that include off-site access to the system via telephone modem by the engineer or several operators (at night). These multiple-terminal operational schemes can be much more costly to implement because simple PCs may not be capable of the performance needed to accomplish graphic programming.

Furthermore, the extensive proprietary software required for graphics programming can become costly, and it is possible a copy will have to be purchased for every computer that may be used. Such costs could substantially impact the flexible DDC operating schemes employed by many users.

Graphic programming is another serious attempt to provide DDC system operators, whose primary training and knowledge are in mechanical systems, with the ability to write and adjust high-performance custom DDC applications programs. Today, DDC system users and manufacturers alike understand the need to implement DDC systems that are both functional and easy to use. This represents an important change in operations philosophy from a few years ago when most manufacturers (and many users) believed DDC system operators should not be permitted direct access to control programs.

Looking to the future

While Capt. Kirk might not be impressed with the improvements in DDC programming capabilities that have been made in the last few years, DDC system operators should be. Comparing today's DDC programming features with those only a few years old makes one realize the enormous strides that the industry has made. Recently, I discussed program specifics for a replacement DDC system with a very knowledgeable operator of an older system. His ideas and concerns suggested to me that however brilliantly this operator had employed his old system to provide comfort and energy efficiency, his mode of thinking was now limited by the operational capabilities of that system. It is likely he will be able to utilize fully the capabilities of the new DDC system only after he has developed an understanding of them through experience once the new system is installed.

This operator's problem is probably universal to the building design industry-and to other industries that utilize digital technologies as well. None of us is particularly adept at understanding how effective new digital technologies can be until we have some experience working with them. This makes it difficult to look very far in the future with clarity. However, we can look at the issues that need to be resolved to continue to improve the success of DDC programming languages, and this may provide some answers for the most likely next improvements.

Combined line and graphics

Our firm, and many of our clients, continue to believe that line-based programming languages provide the best method to develop DDC programs that execute effective energy and comfort-enhancing control strategies. However, we believe it is possible that combining certain features of both line-programming and graphics-programming techniques may produce a format for control programming that has advantages over the all-programming techniques generally available today.

Earlier in this article, it was illustrated that line-based programs can be the most effective way to represent many kinds of mathematical and logical expressions because they can make such expressions clearly and concisely. However, a problem with line programs is the lack of clarity in representing major logic sequences. Fig. 5 shows the system-level logic that might be employed to control a simple fan system.

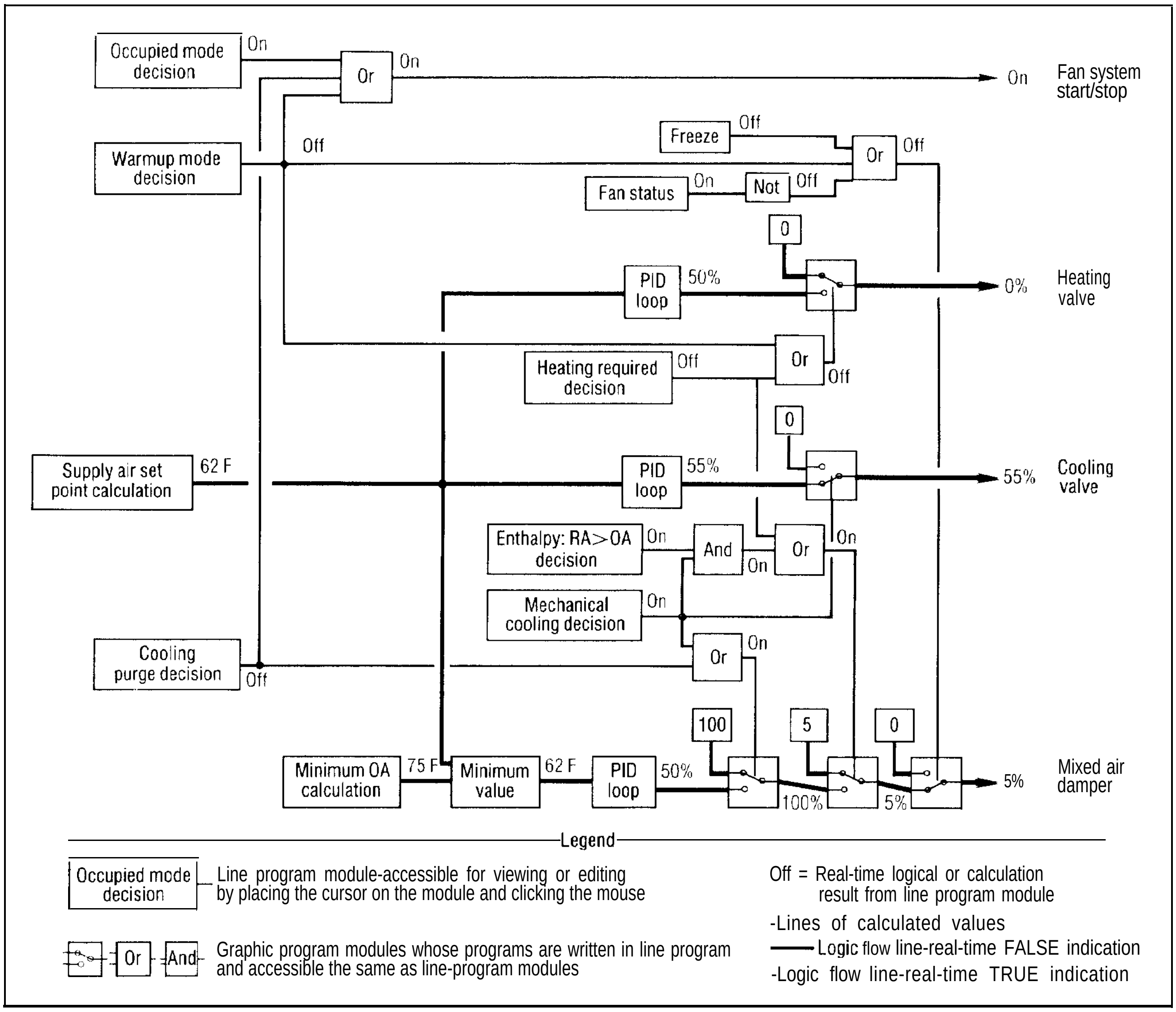
```
"FAN ON/OFF CONTROL'

DOEVERY 1 MIN
   IF OCCUPIED_MODE  = ON OR COOLING_PURGE  = ON OR
   WARMUP-MODE = ON THEN START SUPPLY-FAN ELSE STOP SUPPLY-FAN
ENDDO

"HEATING VALVE CONTROL"

IF WARMUP-MODE = ON OR HEATING_REQD = ON THEN BEGIN
   PID_HTG:SETPOINT  = SUPPLY_SETPOINT_CALC
   HEAT-VALVE = PID_HTG
END
ELSE HEAT-VALVE = 0

"COOLING VALVE CONTROL"

IF MECH_CLG  = ON THEN BEGIN
   PID_CLG:SETPOINT  = SUPPLY_SETPOINT_CALC
   COOL-VALVE = PID_CLG
END
ELSE COOL-VALVE = 0

"MIXED AIR DAMPER CONTROL'

IF SUPPLY FAN = OFF THEN MIXED-DAMPER = 0 ELSE
   IF MECH_CLG  = ON THEN IF ENTHALPY_RA = ON THEN MIXED-DAMPER = 5
      ELSE MIXED DAMPER = 100 ELSE
         IF COOLING_PURGE = ON THEN MIXED-DAMPER = 100 ELSE BEGIN
            IF MINIMUM_OA  < SUPPLY_SETPOINT_CALC  THEN
               PID_MAD:SETPOINT = MINIMUM_OA ELSE BEGIN
                  PID_MAD:SETPOINT  = SUPPLY_SETPOINT_CALC
                  MIXED-DAMPER = PID_MAD
               END
         END
```

NOTE: OCCUPIED-MODE, COOLING_PURGE, WARMUP_MODE. HEATING_REOD, MECH_CLG, SUPPLY_SETPOINY_CALC, ENTHALPY _RA, MINIMUM_OA are all variables representing logical decisions or calculations that are not shown in the program. The program represents the logic flow of a simple fan system that is represented graphically In figure 6. The underline character Is used lo join words to make single terms for point names or variables. This follows normal programming convention.

5 Line program showing control logic for Simple fan System.

In Fig. 5, the calculations and lower-level logic can be considered to have been made elsewhere and are represented by their resulting variables. Note that the logic controlling the supply fan is very readable in this format. However, the logic for the mixed air dampers is not so simple and therefore somewhat difficult to follow even though it is concise.

Function-block and graphics programming as they exist today are also weak in representing system-level logic because they are not concise. In these programs, logic


Occupied mode decision
On
Or
On
On
Fan system start/stop
Freeze
Off
Warmup mode decision
Off
Or
Off
Fan status
On
Not
Off
0
PID loop
50%
0%
Heating valve
Heating required decision
Off
Or
Off
0
Supply air set point calculation
62 F
PID loop
55%
55%
Cooling valve
Enthalpy: RA>OA decision
On
And
On
Or
On
Mechanical cooling decision
On
Cooling purge decision
Off
Or
On
100
5
0
Minimum OA calculation
75 F
Minimum value
62 F
PID loop
50%
100%
5%
5%
Mixed air damper
Legend
Occupied mode decision
Line program module-accessible for viewing or editing by placing the cursor on the module and clicking the mouse
Or
And
Graphic program modules whose programs are written in line program and accessible the same as line-program modules
Off = Real-time logical or calculation result from line program module
-Lines of calculated values
Logic flow line-real-time FALSE indication
-Logic flow line-real-time TRUE indication

6 **System logic for simple fan system.**

paths are provided, but each component represents a very small' portion of the overall program. Therefore, the overall system logic can be represented only after one has pieced together a number of individual blocks. The DDC industry has not yet found an effective means to represent system-level logic. This is a shortcoming of every programming format in wide use today.

With the features now generally available in PCs, it may soon be possible to combine line-program blocks with graphic representations of system logic to provide a format for improved DDC control representations. Fig. 6 shows the logic for the simple fan-control program of Fig. 5 in graphic form. Note that the graphic overview is effective in representing the system logic. The circumstances under which the dampers and heating or cooling valves are operated can be reasonably deduced from the diagram. The labeled rectangular blocks are line-program modules.

Imagine that this graphic representation can display real-time result(s) for each line-program module with the current lines of control logic highlighted in special colors for true or false. Assume further that the contents of any of the line-program modules can be called up for review or editing simply by clicking on the chosen module. With such a programming technique, the operator could quickly isolate and troubleshoot the exact areas of logic in effect when problems develop. Such a programming scheme as represented in Fig. 6 may provide advantages over both line and graphics programming while mitigating many of their disadvantages.

Future languages

The programming concepts shown above may be a natural continuation for recent improvements in DDC system programming languages, but a wide variety of other options are possible as well. Whatever the next steps in programming languages may be, they will be successful only if they work toward solving the following current problems:

- *Concise representation of ef-*

fective DDC control strategies- The key to success with DDC controls is not to emulate traditional pneumatic controls but to use the power and flexibility of DDC systems to provide new, in-depth modes of control that result in enhanced comfort and energy-efficient operation.

Such modes of control can be supported only if they are provided in programs that can be understood and diagnosed by system operators. It is not enough to break the program into such small pieces that the overall concept is difficult to determine because the operator has trouble assembling all the pieces together at once. Any language must include representation(s) that show very clearly *both* the overall concept and the calculation/logic pieces that constitute that concept.

- *Real-time indications of program calculations and logic paths*-One of the most powerful tools available to a DDC system operator is the ability to watch programs as they execute and see the results as they are calculated. This programming tool is typically available only with interpretive languages. However it is accomplished, languages must be developed that enhance the operator's ability to view real-time calculations and logic while the DDC system is operating. This feature allows the operator to check programs easily when their operation is suspect.

- *Few and simple rules to govern the language*-The more rules that govern how a programming language can be applied, the more difficulties the operator has trying to support the programs. Early applications program languages had many rules governing everything from the use of integers and floating point numbers to the use of math in conditional statements. Manufacturers have done a good job issuing revisions that have simplified language rules for many existing DDC languages. More needs to be done, however, and new languages, whether graphic- or line-based, should be as free of restrictive rules as possible.

- *Low cost*- Improvements in programming languages cannot be permitted to reverse the trend toward lower-cost DDC systems. Manufacturers should consider the enormous market potential for their products when they have combined sufficient function and ease of operation in a package that competes with pneumatics on first cost. Full DDC systems are usually 10 to 30 times larger (in terms of system points) than the DDC systems that go into many buildings today. Retrofit opportunities are even greater.

Energy costs are now high

enough that most building owners can find a very attractive rate of return in an investment of between $1 and $2 per sq ft for complete HVAC and lighting control retrofit. If full DDC systems that provide the comfort enhancement and energy reductions of advanced control strategies can be implemented at these costs, the industry will experience an enormous growth in volume over the next few years that will help pay for some of the development requirements.

Summary

The DDC industry has made substantial steps over the last few years to improve the power and flexibility of the control languages supplied with their systems. This is a primary reason DDC systems are better accepted today than ever before.

The types of control languages commonly available for programming DDC systems today include line-based programming languages, function-block programming languages, and now graphics programming. Each of these approaches has certain advantages when compared to the others for specific applications, but it is clear further improvements are still needed, particularly improved representations of system logic.

When considering programming language improvements, manufacturers should work toward approaches that permit the more in-depth strategies possible with DDC control to be represented clearly and concisely and provide methods whereby real-time logic paths and calculation results can be displayed and reviewed as the program is operating.

As discussed in last month's article, the acceptance of full DDC systems has become a reality by users who wish to have their buildings perform more effectively than they can with traditional controls. However, to ensure that this higher level of performance can be installed and will be maintained, better performing and more easily supported applications languages need to be developed. As the power, flexibility, and ease of implementing DDC applications programs grow, the demand for DDC products will grow also. Ω

The Hartman Co. intends to update its operators' control language (OCL) guide to DDC system programming function in 1991. We request comments from all in the industry who are experienced in high-performance DDC systems on the subject of DDC control languages and related items. We will be happy to make a copy of the current OCL guide available to such individuals at no cost. Please write to: OCL Guide, The Hartman Co., 1016 North 36th St., Seattle, WA 98103.