

3.0 CONTROL BLOCK PROGRAMMING

AutoMax Control Block is a high-level language used for programming process control loops. It consists of BASIC language statements and function calls, or blocks, that are commonly used in the design of control systems. These functions have a graphic form that is used during the design and check-out phase of a project, as well as a textual form that is used by the programmer for creating the program. The graphic form includes an abbreviated symbol for each input and output. An arrow pointing into a block indicates a required input parameter. An arrow pointing out of a block indicates a required output parameter.

A file that contains templates of the textual form of all blocks is loaded onto the hard disk of the personal computer when the Executive software is installed. The file is named "TEMPLATE.BLK" and is stored in the AutoMax installation directory. You can access the file when creating Control Block tasks using a text editor.

TEMPLATE.BLK can be used when creating either AutoMax Control Block tasks or UDC Control Block tasks. AutoMax PC3000 Control Block tasks are identical to AutoMax Control Block tasks. Note that some of the blocks listed in TEMPLATE.BLK cannot be used in UDC Control Block tasks. The text editor will not prevent you from using an illegal block, but the compiler will catch the error. The task will not compile successfully until any illegal blocks are removed. Refer to Appendix C for a list of the Control Block functions supported in UDC Control Block tasks.

3.1 Format of Control Blocks

A control block, or function call, consists of the following:

a line number: positive integer in the range 1-32767 inclusive

the keyword CALL

the function name that identifies the function

the parameter list, which contains the inputs and outputs for the function.

Note that there are three types of inputs: optional, required, and default. Optional means that the value or parameter does not need to be entered. Required means that the value/parameter must be entered. Default means that the value/parameter does not need to be entered by the user, but that the value is still required in order to execute the function and, if the user does not enter a value, the default value will be used in executing the function.

3.2 Task Execution

Each AutoMax Control Block task or UDC Control Block task must include a SCAN LOOP block. The SCAN LOOP block specifies the periodic interval at which the task will execute. The SCAN LOOP block must be the first block called in a Control Block task. Any BASIC statement that could cause the task to be suspended is not permitted. The BASIC statement END must be executed at the end of each scan of a Control Block task. See Table 4 and 5, Maximum

Execution Time Summary, for information on execution times for all blocks. Note that UDC Control Block tasks can use only the blocks listed in Table 5. See the AutoMax Programming Executive instruction manual for more information on task execution.

3.3 Variable Definition and Initialization

All variables used in the task must be defined using the BASIC statements COMMON or LOCAL at the beginning of the scan before the SCAN LOOP block. All variables that must be at a known state or at a non-zero state must also be initialized in the task, i.e., a value must be assigned to them, before the SCAN LOOP block. Variables are initialized using the BASIC statement LET or the BASIC relational operator “=”.

At the beginning of each scan, the values of the following are latched, i.e., read into a local buffer for reference throughout the scan: all simple (non-subscripted) common integer, double integer, boolean variables. This ensures that external inputs will not change state during a scan. Reals, strings, and array variables of any data type are not latched.

As the task executes, each block will read/write simple common variables of integer, double integer, and boolean type to this local buffer. Each block that references these variables will always see the value of those variables as currently stored in the buffer, even if another task modifies the actual state of the variables in question. At the end of the Control Block task, all buffered variables are examined to see if they are different from their initial state. If so, the new value is written to the actual variable.

Note that BASIC statements, whether in a BASIC task, an AutoMax Control Block task, or a UDC Control Block task, always reference the current value/state of all common variables. Variables are not buffered for BASIC statements. Instead, as they are referenced via the BASIC statement, they are read from and written to their actual location, whether it is a common memory location or a common I/O point.

For example, Control Block task ABC references variable COMM_VAR% from several control block statements within the task. Task XYZ, which is in the same Processor module or another Processor module in the same rack, writes to COMM_VAR%. At the start of the scan of ABC, the value of COMM_VAR% is 100 and is read and stored in the local buffer of ABC. Task XYZ runs and changes the value of COMM_VAR% to 299 before task ABC has completed its scan. Since the buffered value is used, all control block statements that reference COMM_VAR% will see 100 rather than 299 as the value of COMM_VAR%.

WARNING

IF BASIC STATEMENTS ARE USED WITHIN A CONTROL BLOCK TASK, ENSURE THAT A CONFLICT DOES NOT ARISE BETWEEN A CONTROL BLOCK'S USE OF A PARTICULAR COMMON VARIABLE AND A BASIC STATEMENT'S USE OF THE SAME COMMON VARIABLE. A CONFLICT MAY ARISE IF A BASIC STATEMENT REFERENCES A COMMON VARIABLE ALSO REFERENCED BY A CONTROL BLOCK STATEMENT WITHIN THE TASK. FAILURE TO OBSERVE THESE PRECAUTIONS COULD RESULT IN BODILY INJURY OR IN DAMAGE TO OR DESTRUCTION OF THE EQUIPMENT.

In the example below, if a BASIC statement produces variable `COMM_VAR%`, subsequently referenced as an input to a control block statement, the DIFFERENCE block will not see the new state of `COMM_VAR%` produced by statement 101. Instead, it will see the state produced during the last scan of the task. This is because the DIFFERENCE block will obtain the value of `COMM_VAR%` from the LOCAL buffer and the BASIC statement at line 101 will write the new value for `COMM_VAR%` to its common location. This value will then be read into the task's LOCAL buffer at the start of the next scan.

```
101 COMM_VAR% = LOCAL_A% + LOCAL + C%
```

```
.  
. .  
.
```

```
204 CALL DIFFERENCE(INPUT1 = COMM_VAR%,    &  
                    INPUT2 = FDBK%, OUTPUT = CNTL_ERR%)
```

Control Block tasks in which BASIC statements and control blocks reference the same simple common integer, double integer, and boolean variables must be constructed taking into account the fact that BASIC statements will always reference the actual current value of the variable, whereas control blocks will always reference the value of the variable that is in the buffer.

Potential problems can be avoided if both BASIC statements and control blocks in the task use local variables, or if common variables used in BASIC statements are not used by any control blocks.

Note that if you define both an integer and bits within that integer, the variable values are stored in the Control Block task buffer independently. If you write to the integer variable in the buffer (i.e., with a Control Block statement), you do not affect the integer's bits within the buffer. If you use BASIC statements to write to the integer or bits within the integer, you are referencing the actual variable values, not the values in the buffer. If you want to change the value of an integer and the bits within it, use BASIC statements to do so.

See Appendix B for the BASIC statements that can be used in AutoMax Control Block tasks. See Appendix D for the BASIC statements that can be used in UDC Control Block tasks. See the Enhanced BASIC Language instruction manual, J-3675, for error codes that can occur when application tasks are compiled and when they are put into RUN.