Writing a Basic Quickstep Program

October, 2005

Control Technology Corporation, Hopkinton, MA • 800.282.5008 • www.ctc-control.com

> *Note: This is intended to be used as a supplement to the Quickstep Programming manuals and NOT as a replacement.*
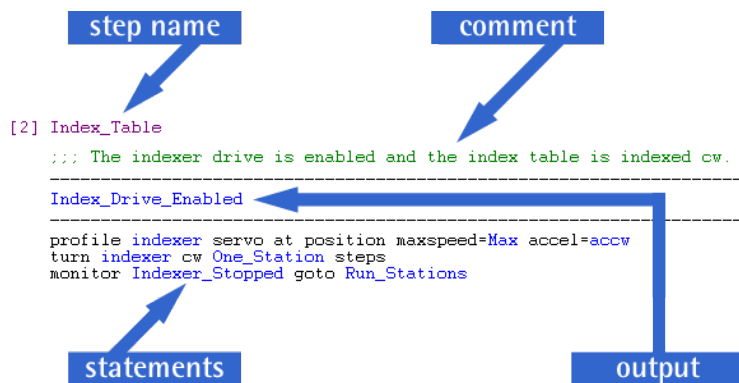
The Basics

Here is what you absolutely need to know to get started:

1. Quickstep Steps have the following format:

   ### The "State Machine" Step
   A Quickstep Programming Step is divided up into four sections:

   - the Step Name section
   - the Comment section
   - the Output section and
   - the Statement section.

2. The following rules apply to Step Execution:

   a. Programmed output changes will always occur first, upon entering a new step, however outputs may be accessed via special-purpose registers and used within the instruction area of a step.
   b. Instructions will be executed in the order in which they are programmed.
   c. "Monitor" and "If" instructions, if satisfied upon first entering a step, will cause any remaining instructions NOT to be executed.
   d. "Monitor" and "If" instructions will continue to scan for the stated condition(s) while a step is active.

e. Instructions that DO NOT include a `goto` (i.e. `Store`) will only be executed once when step execution begins. This means: 1) Math instructions, turn motor instructions, etc. will not be repetitively triggered while your program remains in a step. 2) If you want these type instructions to be repeated you need to include a "`goto samestep`" instruction at the end of your step.

3. Each Input and Output is defined with 2 different symbols, one for each state: on and off for outputs, normally open and closed for inputs.

The Instructions (Quick List)

See the *Specifics on the Most Commonly Used Statements* section of this document or the Quickstep™ Language and Programming Guide for additional information on these Instructions.

Instruction	Description
Cancel (other tasks)	This step will cancel all other executing tasks
Clear (flag)	This statement simply clears a flag
Count	Causes a counter to increase (up) or decrease (down) by one
Delay	Allows you to program in a delay based on a numeric or volatile register value
Disable (Counter)	Stops a counter from counting up or down when its assigned inputs become active
Do (Multi-Tasking)	Starts up to 8 different tasks. Multiple `Do` statements are required to start more then 8. Do must be the only statement in a step.
Done	Stops the task that called the Step that contains this statement. Like the `Do` statement, `Done` must be alone in its own step.
Enable (Counter)	Enables a counter to allow it to count up or down after it's been disabled. Note the `Start` statement is used to initialize a counter.
Goto	Unconditional Jump. Causes the program to start executing another step.
If	Conditional Jump. This is used to look for register based conditions to cause the program to start executing another step.
Monitor	Conditional Jump. This is used to look for binary based conditions (i.e. inputs on or off, or flags set or cleared) to cause the program to start executing another step.
Profile	Initializes the servo
Reset (Counter)	Sets the value of the counter to zero
Rotate (Flags)	Rotates the values of the flags (like a shift register)
Search and Zero	Commands the servo or stepper to execute a home sequence
Set	Sets a flag
Shift	Shifts the values of the flags
Start	This command initializes a counter
Stop (Controller)	This stops the controller from executing its program
Stop (Motor or Servo)	This stops the Servo or Motor from turning. A Soft stop commands a stop using the programmed deceleration rate. A Hard stop commands an immediate stop.
Store	This stores data from one register to another and also allows one math function to be performed per statement
Test and Set (flags)	Checks to see if a flag is set. If the flag is not set it will set the flag and goto the specified step. If the flag is already set the statement will continue to scan until the flag is cleared and then set the flag and goto the specified step. Usually used in Multitasking.
Turn	Commands a servo or motor move. Servo must be stopped before this command can be used.
Zero	Sets a servos current position to zero. Servo must be stopped before this command can be used.

Quick Details about Multi-Tasking

1. The Do statement can start up to eight tasks within that statement. If you need to start more than eight tasks this needs to be done using multiple steps.
2. The number of tasks that can run simultaneously is controller dependent.
3. The Do and Done statement must be the only statement in a Step.
4. The Done statement ends the task that called this statement.
5. The Cancel statement stops all running tasks except the task that called this statement.
6. The Do statement includes a goto statement. The Do statement will goto the step indicated once all of the tasks that were started from the do statement have executed a done statement.
7. Tasks can be finite or infinite (i.e. you can program them to finish with a done statement or you can program them to continuously run forever and never execute a done statement). See Figures 1 and 2.
8. If you program a task to run multiple times it will continue to start over until it sees the Done statement the specified amount of times. See example in Figure 3.
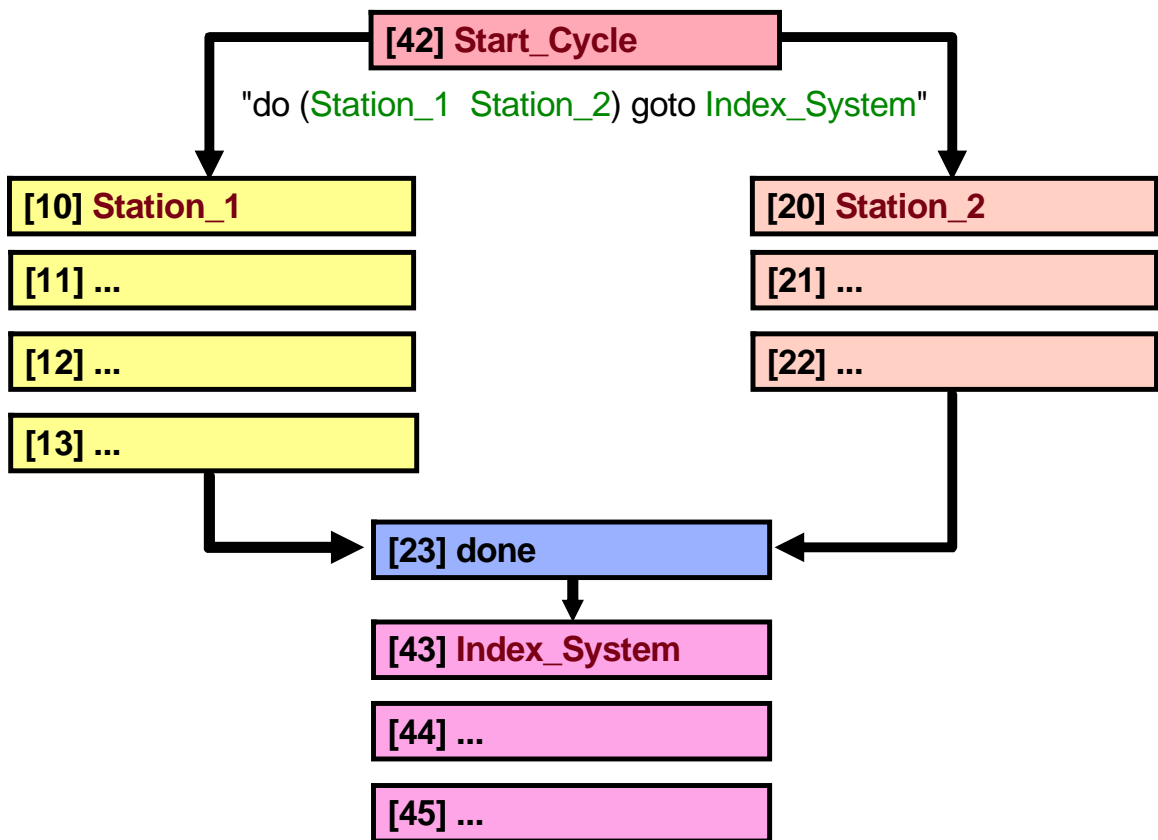
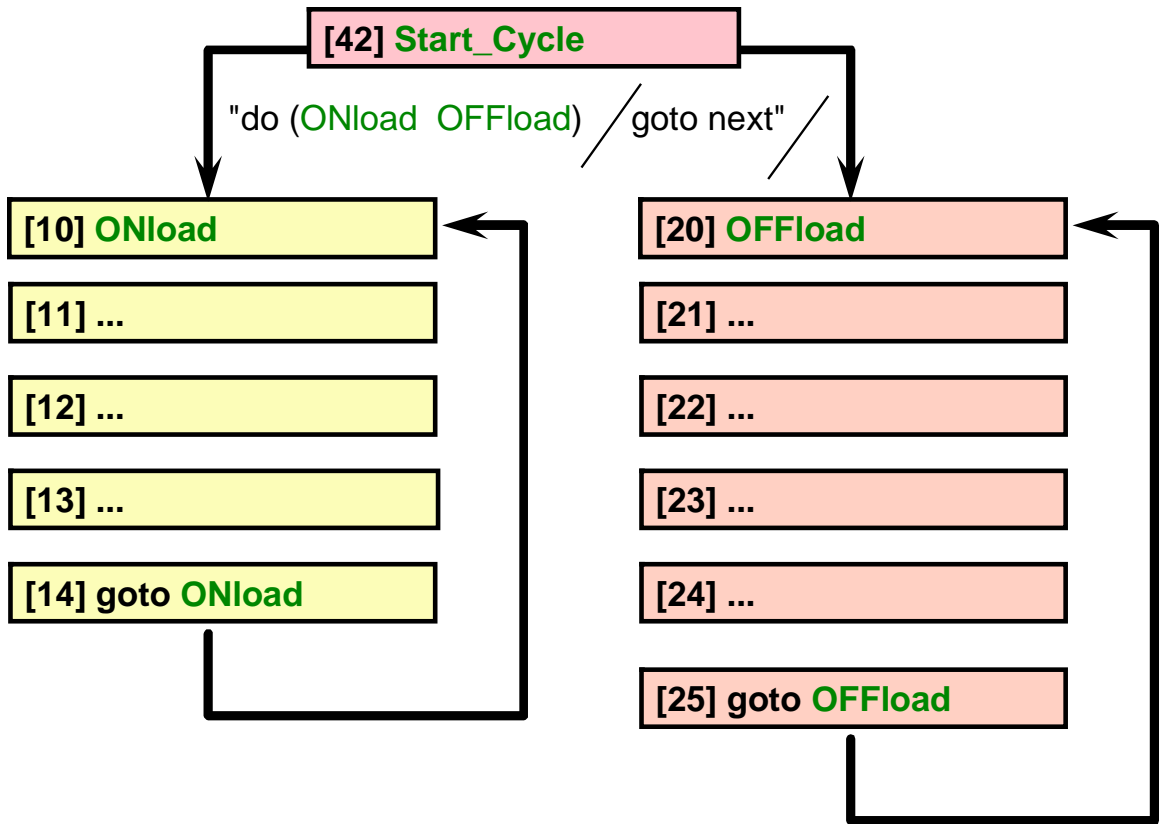*Figure 1: Flow chart depicting two finite tasks*

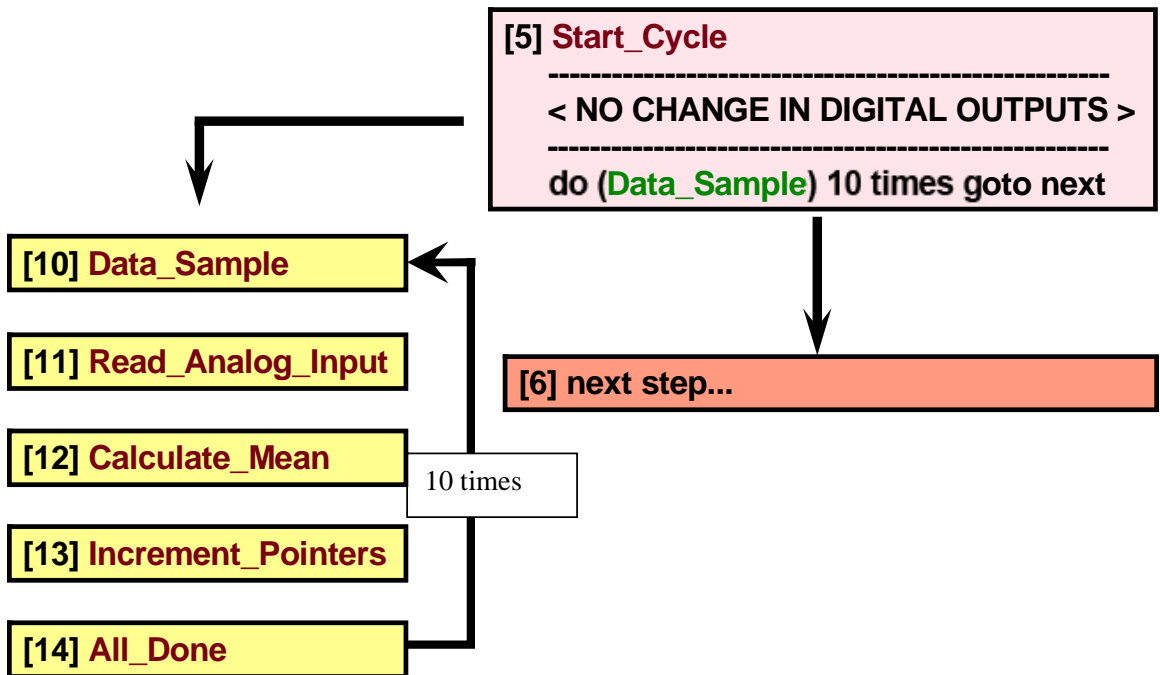*Figure 2: Flow Chart depicting two infinite tasks*

Figure 3: Flow chart using the Do statement to run just one task multiple times.

www.ctc-control.com   Control Technology Corporation

Specifics on the most commonly used statements

Monitor - The `monitor` instruction allows you to check the state of flags, inputs, motors, and servos either individually or combined in a Boolean statement, allowing the programmer to make a decision based on the state of controller resources.

Examples:
```
monitor Ejector:set goto Eject_Part (flag)
monitor Punch_Extended goto Retract_Punch (input)
monitor X_Axis:Stopped goto Move_Servo (servo)
monitor (and Stamp_Retracted Ejector:set) goto Eject_Part (input &
flag)
```

If - The `If` instruction allows you to check the value of one resource against the value of another. Based on whether the statement is true or not, a decision is made in your program. Logical operators include $>$, $<$, $>=$, $<=$, $=$, $<>$ and allow the comparison of analog inputs, data table elements, integer numbers, registers, counters, servo motor position or error.

Example: `if reg_15 >= reg_16 goto next`

Store - Data manipulation is carried out with the `Store` command. By utilizing this command, accessibility is gained to controller resources, such as analog inputs, registers, and servo error & position. These resources can then be massaged, converted, and stored to other resources.

Data Manipulation Examples: arithmetic, Boolean, modulo, rotate left or right & masking.
```
store Part_Count + 1 to Part_Count
store reg_11 mod 2 to reg_15
store reg_75 rol 4 reg_76
```

Controller Resources Examples: Ain, Aout, Data Table, Reg., Servo Parameters
```
store servo_1:position /reg_501 to reg_122
store ain_1 * reg_22 to aout_1
store servo_1:error to reg_123
```

Delay - The delay instruction is used when it is desired to dwell in a step for a set period of time before continuing on to a new step. The delay time can be in hours, min., sec., & ms.

Examples:
```
delay 2 sec 30ms goto next
delay reg_22 / 100 sec goto next
```
The counter instructions control the eight internal counters in a controller.

Start - The start instruction starts or initializes a counter. In the example below counter_1 will increment when an input is seen on input _5, it will count down when an input is made on input_6 and will be reset to zero when an input is made on input_7. This will all be done in the controller's background routine while the program is running.

Example:
```
start counter_1 up (input_5) down (input_6) reset (input_7)
```
The other counter instructions include Disable, Enable, Reset, and Count (up and down), and are explained in the Instruction quick list section.

Servo Statements

Profile - Both stepper and servo motors need to have operating parameters defined for them before you can command motion. Therefore the `profile` command must be executed before any turn command. The parameters in the `profile` command can be either hard coded directly into the profile instruction, as shown above, or they can be referenced to a controller resources, such as a register.

Examples:
```
profile Motor_1(half) basespeed=50 maxspeed=1000 accel=500 decel=500
profile Servo_1 at position maxspeed=10000 accel=5000 P=5 I=10 D=200
```
The older 2215 stepper cards use the profile motor command. The newer 2216 stepper cards use the Profile Servo command.

Search and Zero - The `Search and Zero` instruction starts the motor turning at a slow speed until the controller's motor module senses a contact closure on the home limit switch input. At that time, the motor stops and the current position is set as the zero point.

Example: `search and zero Servo_1`
The direction of home and the homing target are set using dedicated registers. See the Register Reference Guide for your specific controller for more details.

Turn - The `turn` instruction initiates a new motor motion. The controller must have executed a `Profile` command to define the motion parameters. Both the Stepper and the Servo can define their moves as Absolute (turns the motor a calculated number of steps based on the distance from a predetermined zero position) or Relative (turns the motor cw or ccw a specified number of steps from the motors current position. The Servo can also make a Velocity move (begin continuous cw or ccw motion). When this instruction is executed a stop command is needed to terminate motion.

Examples:
```
turn Motor_1 ccw 750 steps (relative)
turn Motor_1 to 1500 (absolute)
turn Servo_1 ccw 750 steps (relative)
turn Servo _1 to 1500 (absolute)
turn Servo _1 ccw (velocity)
```

Stop - The `stop` servo instruction brings the servo to a halt. You can choose the Soft Stop -(causes the servo to stop at the deceleration rate specified in the last profile instruction) or the Hard Stop - (causes the controller to attempt to stop the servo instantly) to stop the servo or stepper. If you choose the hard stop please note that because of momentum the servo will not stop instantly and consequently the absolute position may be lost and instability may result.

Examples:
```
stop (soft) Servo_1
stop (hard) Servo_1
```