MotionTech integral MicroBasic programming for “MT” series BLDC motor controllers

Do More with Scripting

One of the controller’s most powerful and innovative features is the ability for the user to write programs that are permanently saved into, and run from the controller’s Flash Memory. This capability is the equivalent of combining the motor controller functionality and this of a PLC or Single Board Computer directly into the controller.

Script can be simple or elaborate and can be used for various purposes:

- Complex sequences.
  MicroBasic Scripts can be written to chain motion sequences based on the status of analogue/digital inputs, motor position, or other measured parameters. For example, motors can be made to move to different count values based on the status of pushbuttons and the reaching of switches on the path.
- Adapt parameters at runtime.
  MicroBasic Scripts can read and write most of the controller’s configuration settings at runtime. For example, the Amps limit can be made to change during operation based on the measured heatsink temperature.
- Create new functions.
  Scripting can be used for adding functions or operating modes that may be needed for a given application. For example, a script can compute the motor power by multiplying the measured Amps by the measured battery Voltage, and regularly send the result via the serial port for Telemetry purposes.
- Autonomous operation.
  MicroBasic Scripts can be written to perform fully autonomous operations. For example the complete functionality of a line following robot can easily be written and fitted into the controller.

Script Structure and Possibilities

Scripts are written in a Basic-Like computer language. Because of its literal syntax that is very close to the every-day written English, this language is very easy to learn and simple scripts can be written in minutes. The MicroBasic scripting language also includes support for structured programming, allowing fairly sophisticated programs to be written. Several shortcuts borrowed from the C-language (++, +=, ...) are also included in the scripting language and may be optionally used to write shorter programs.

The complete details on the language can be found in the MicroBasic Language Reference available on request.

Source Program and Bytecodes

Programs written in this Basic-like language are interpreted into an intermediate string of Bytecode instructions that are then downloaded and executed in the controller. This two-step structure ensures that only useful information is stored in the controller and results in significantly higher performance execution over systems that interpret Basic code directly. This structure is for the most part entirely invisible to the programmer as the source editing is the only thing that is visible on the PC and the translation is done in the background just prior to downloading to the controller.

The controller can store 8192 Bytecodes. This translates to approximately 1500 lines of MicroBasic source.

Variables Types and Storage

Scripts can store signed 32-bit integer variables and Boolean variable. Integer variables can handle values up to +/- 2,147,483,647. Boolean variables only contain a True or False state. The language also supports single dimensional arrays of integers and Boolean variables.

In total, up to 1024 Integer variables and up to 1024 Boolean variables can be stored in the controller. An array of n variables will take the storage space of n variables.

The language only works with Integer or Boolean values. It is not possible to store or manipulate decimal values. This constraint results in more efficient memory usage and faster script execution. This constraint is usually not a limitation as it is generally sufficient to work with smaller units (e.g. millivolts instead of Volts, or milliamps instead of Amps) to achieve the same precision as when working with decimals.

The language does not support String variables and does not have string manipulation functions. Basic string support is provided for the Print command.

Variable content after Reset

All integer variables are reset to 0 and all Boolean variables are reset to False after the controller is powered up or reset. When using a variable for the first time in a script, its value can be considered as 0 without the need to initialize it. Integer and Boolean variables are also reset whenever a new script is loaded.

When pausing and resuming a script, all variables keep the values they had at the time the script was paused.

Controller Hardware Read and Write Functions

The MicroBasic scripting language includes special functions for reading and writing configuration parameters. Most configuration parameters can be read and changed using the Configuration Tab in the PC utility or using the Configuration serial commands which can be read and changed from within a script. The GetConfig and SetConfig functions are used for this purpose.

The GetValue function is available for reading real-time operating parameters such as Analog/Digital input status, Amps, Speed or Temperature.

The SetCommand function is used to send motor commands or to activate the Digital Outputs. Practically all controller parameters can be access using these 4 commands, typically by adding the command name as defined in the Serial (RS-232/USB) Operation of the User Manual, preceded with the "_" character. For example, reading the Amps limit configuration for channel 1 is done using getvalue(_ALIM, 1).

See the MicroBasic Language Reference for details on these functions and how to use them.

Timers and Wait

The language supports four 32-bit Timer registers. Timers are counters that can be loaded with a value using a script command. The timers are then counting down every millisecond independently of the script execution status. Functions are included in the language to load a timer, read its current count value, pause/resume count, and check if it has reached 0. Timers are very useful for implementing time-based motion sequences.

A wait function is implemented for suspending script execution for a set amount of time. When such an instruction is encountered, script execution immediately stops and no more time is allocated to script execution until the specified amounts of milliseconds have elapsed. Script execution resumes at the instruction that follows the wait.

Execution Time Slot and Execution Speed

MicroBasic scripts are executed in the free time that is available every 1ms, after the controller has completed all its motion control processing. The available time can therefore vary depending on the functions that are enabled or disabled in the controller configuration. For example more time is available for scripting if the controller is handling a single motor in open loop than if two motors are operated in closed loop with encoders. At the end of the allocated time, the script execution is suspended, motor control functions are performed, and scripts resumed. An execution speed averaging 50,000 lines of MicroBasic code, or higher, per second can be expected in most cases.

Protections

No protection against user error is performed at execution time. For example, writing or reading in an array variable with an index value that is beyond the 1024 variables available in the controller may cause malfunction or system crash. Nesting more than 64 levels of subroutines (i.e. subroutines called from subroutines) will also cause potential problems. It is up to the programmer to carefully check the script's behaviour in all conditions.

Print Command Restrictions

A print function is available in the language for outputting script results onto the serial or USB port. Since script execution is very fast, it is easy to send more date to the serial or USB port than can actually be output physically by these ports. The print command is therefore limited to 32 characters per 1ms time slot. Printing longer strings will force a 1ms pause to be inserted in the program execution every 32 characters.

Editing Scripts

An editor is available for scripting in the Plus PC utility.

The edit window resembles this of a typical IDE editor with, most noticeably, changes in the fonts and colours depending on the type of entry that is recognized as it is entered. This capability makes code easier to read and provides a first level of error checking.

Code is entered as free-form text with no restriction in term of length, indents use, or other.

Building Scripts

Building is the process of converting the Basic source code in the intermediate Bytecode language that is understood by the controller. This step is nearly instantaneous and normally transparent to the user, unless errors are detected in the program.

Build is called automatically when clicking on the “Download to Device” or “Simulate” buttons.

Building can be called independently by clicking on the “Build” button. This step is normally not necessary but it can be useful in order to compare the memory usage efficiency of one script compared to another.

Simulating Scripts

Scripts can be ran directly on the PC in simulation mode. Clicking on the Simulate button will cause the script to be built and launch a simulator in which the code is executed. This feature is useful for writing, testing and debugging scripts. The simulator works exactly the same way as the controller with the following exceptions.

- Execution speed is different.
- Controller configurations and operating parameters are not accessible from the simulator
- Controller commands cannot be sent from the simulator
- The four Timers operate differently in the simulator

In the simulator, any attempt to read a Controller configuration (example Amps limit) or a Controller Runtime parameter (e.g. Volts, Temperature) will cause a prompt to be displayed for the user to enter a value. Entering no value and hitting Enter, will cause the same value that was entered last for the same parameter to be used. If this is the first time the user is prompted for a given parameter, 0 will be entered if hitting Enter with no data.

When a function in the simulator attempts to write a configuration or a command, then the console displays the parameter name and parameter value in the console.

Script execution in the simulator starts immediately after clicking on the Simulate button and the console window opens.

Simulated scripts are stopped when closing the simulator console.

Downloading MicroBasic Scripts to the controller

The Download to Device button will cause the MicroBasic script to be built and then transferred into the controller’s flash memory where it will remain permanently unless overwritten by a new script.

The download process requires no other particular attention. There is no warning that a script may already be present in Flash. A progress bar will appear for the duration of the transfer which can be from a fraction of a second to a few seconds. When the download is completed successfully, no message is displayed, and control is returned to the editor.

An error message will appear only if the controller is not ready to receive or if an error occurred during the download phase.

Downloading a new script while a script is already running will cause the running script to stop execution. All variables will also be cleared when a new script is downloaded.

Executing MicroBasic Scripts

Once stored in the Controller's Flash memory, scripts can be executed either "Manually" or automatically every time the controller is started.

Manual launch is done by sending commands via the Serial or USB port. When connected to the PC running the PC utility, the launch command can be entered from the Console tab. The commands for running as stopping scripts are:

- !r : Start or Resume Script
- !r 0: Pause Script execution
- !r 1: Resume Script from pause point. All integer and Boolean variables have values they had at the time the script was paused.
- !r 2: Restarts Script from start. Set all integer variables to 0, sets all Boolean variables to False. Clears and stops the 4 timers.

If the controller is connected to a microcomputer, it is best to have the microcomputer start script execution by sending the !r command via the serial port or USB.

Scripts can be launched automatically after controller power up or after reset by setting the Auto Script configuration to Enable in the controller configuration memory. When enabled, if a script is detected in Flash memory after reset, script execution will be enabled and the script will run as when the !r command is manually entered. Once running, scripts can be paused and resumed using the commands above.

Important Warning

Prior to set a script to run automatically at start-up, make sure that your script will not include errors that can make the controller processor crash. Once set to automatically start, a script will always start running shortly after power up. If a script contains code that causes system crash, the controller will never reach a state where it will be possible to communicate with it to stop the script and/or load a new one. If this ever happens, the only possible recovery is to connect the controller to a PC via the serial port and run the terminal emulation software. Immediately after receiving the Firmware ID, type and send !r 0 to stop the script before it is actually launched. Alternatively, you may reload the controller's firmware.

Script Command Priorities

When sending a Motor or Digital Output command from the script, it will be interpreted by the controller the same way as a serial command (RS232 or USB). This means that the RS232 watchdog timer will trigger in if no commands are sent from the script within the watchdog timeout. If a serial command is received from the serial/USB port at the same time a command is sent from the script, both will be accepted and this can cause conflicts if they are both relating to the same channel. Care must be taken to keep to avoid, for example, cases where the script commands one motor to go to a set level while a serial command is received to set the motor to a different level. To avoid this problem when using the PC utility, click on the mute button to stop commands sending from the PC.

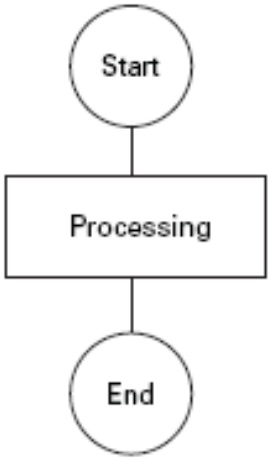
Script commands also share the same priority level as Serial commands. Use the Command Priority Setting to set the priority of commands issued from the script vs commands received from the Pulse Inputs or Analog Inputs.

MicroBasic Scripting Techniques

Writing scripts for the controllers is similar to writing programs for any other computer. Scripts can be called to run once and terminate when done. Alternatively, scripts can be written so that they run continuously.

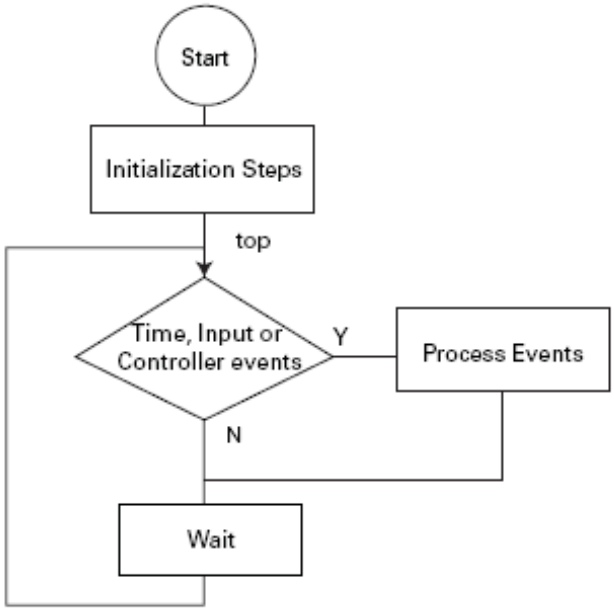
Single Execution Scripts

These scripts are programs that perform a number of functions and eventually terminate. These kinds of scripts can be summarized in the flow chart below. The amount of processing can be simple or very complex but the script has a clear begin and end.



Continuous Scripts

More often, scripts will be active permanently, reacting differently based on the status of analogue/ digital inputs, or operating parameters (Amps, Volts, Temperature, Speed, Count, ...), and continuously updating the motor power and/or digital outputs. These scripts have a beginning but no end as they continuously loop back to the top. A typical loop construction is shown in the flow chart below.



Often, some actions must be done only once when script starts running. This could be setting the controller in an initial configuration or computing constants that will then be used in the script's main execution loop.

The main element of a continuous script is the scanning of the input ports, timers, or controller operating parameters. If specific events are detected, then the script jumps to steps related to these events. Otherwise, no action is taken.

Prior to looping back to the top of the loop, it is highly recommended to insert a wait time. The wait period should be only as short as it needs to be in order to avoid using processing resources unnecessarily. For example, a script that monitors the battery and triggers an output when the battery is low does not need to run every millisecond. A wait time of 100ms would be adequate and keep the controller from allocating unnecessary time to script execution.

Optimizing Scripts for Integer Math

Scripts only use integer values as variables and for all internal calculation. This leads to very fast execution and lower computing resource usage. However, it does also cause limitation. These can easily be overcome with the following techniques.

First, if precision is needed, work with smaller units. In this simple Ohm-law example, whereas 10V divided by 3A results in 3 Ohm, the same calculation using different units will give a higher precision result: 10000mV divided by 3A results in 3333 mOhm

Second, the order in which terms are evaluated in an expression can make a very big difference. For example (10 / 20) * 1000 will produce a result of 0 while (10 * 1000)/20 produces 5000. The two expressions are mathematically equivalent but not if numbers can only be integers.

Script Examples

Below is a continuous script that checks the heat sink temperature at both sides of the controller enclosure and lowers the amps limit to 50A when the average temperature exceeds 50°C. The current limit is set at 100A when temperature is below 50°C. Notice that as temperature is changing slowly, the loop update rate has been set at a relatively slow 50ms rate.

```
print("starting ... \r") ' Lets user know script is starting
top:
' Label marking the beginning of the script.
' Read the 2 temperature sensors and compute average
Temperature1 = getvalue(_TEMP,1)
Temperature2 = getvalue(_TEMP,2)
TempAvg = (Temperature1 + Temperature2) / 2
' If temperature is higher than 50o, set limit to 50A on each channel
if TempAvg > 50 then
        setconfig(_ALIM, 1, 500)
        setconfig(_ALIM, 2, 500)
else
' Otherwise, set limit back to 100A
        setconfig(_ALIM, 1, 1000)
        setconfig(_ALIM, 2, 1000)
end if ' Pause the script for 50ms
wait(50)
' Repeat the script from the start
goto top
```


Distributors for Australia & New Zealand

MOTION TECHNOLOGIES PTY LTD

24/22-30 Northumberland Road
Caringbah NSW 2229 Australia
Phone: (02) 9524 4782
Fax: (02) 9525 3878

sales@motintech.com.au
www.motiontech.com.au

© 09/10/17