



## Tutorial

# Programming Guidelines

General advice and guidance in the use of Control-C for programming the E-Node range.

# Contents

Introduction.....	3
The Three Laws.....	3
Law 1. Keep It Simple (KIS).....	3
Law 2. Break It Down (BID).....	3
Law 3. Make It Modular (MIM).....	3
Programming Basics.....	4
Interrupts.....	5
Building your Program.....	6
Debugging a Program.....	6
Single Stepping.....	6
Run to Cursor.....	6
Step Over.....	6
User Program, Motion Commands and the Control Loop.....	6
Protecting your program.....	7
Questions.....	7

## **Introduction**

This tutorial assumes the reader is familiar with the basics of the E-Node range and Control-C. It also assumes that Control-C is installed on your computer and that you are familiar with its use. If not please refer to the Etrol web site downloads page where all the documents and software are freely available.

The E-Nodes are a range of small control nodes each with its own facilities. They are all connected together via the built in E-Net network. They are all independently programmable and they share their facilities over the E-Net.

At the heart of the E-Node design philosophy is the concept of 'Modularity'. The node themselves are modules and the software they are programmed with can be created in modules. Control systems using the E-Node are intended to be built in a modular fashion.

The E-Nodes themselves are relatively simple and so is Control-C the programming language. However they can be combined into systems to perform very powerful and complex tasks. The method used to design and build these systems is very important. Imagining the E-Node modules are like house bricks. If you follow good building practice a long lasting and beautiful house may result. However if you ignore good building practice an ugly unsafe monster of a house will probably be the case.

This leads us to the The Three Laws.

## **The Three Laws**

To help you build good quality control systems that are easy to understand, reliable and easy to maintain we offer the following laws.

### **Law 1. Keep It Simple (KIS)**

Whatever you are doing, keep it simple. If it is not simple then use Law 2. and break it down until it becomes clear and understandable.

### **Law 2. Break It Down (BID)**

Break all parts of the control system down into modules with each module having a clear defined function. Here we are referring to both the physical system and the software used to program it.

### **Law 3. Make It Modular (MIM)**

By studying the needs of your proposed control system and applying 'KIS' and 'BID' you should be able to build the entire system from interconnected modules.

If you follow the three laws your control system should be a hierarchy of modules much like a family tree in which the top level is broken down into lower level modules and so on until the bottom level is reached. Ideally the bottom level modules will all be very simple and easy to understand and the module tree will describe the function of the whole control system.

If your control system is simple to start with then the laws may not be needed however its always good to keep them in mind because control system have a habit of growing. When the boss sees what you've created he will always ask what else it can do, how long it will take and how much it will cost.

# Programming Basics

The way the programmer thinks is fundamental to producing good code. Clear ordered thinking will produce clear readable and usable code. We start this tutorial by getting you to think in a manner that makes it easy to understand how a program flows and hence the best way to write it.

The user program, is like a simple journey. It always starts at the beginning 'main()' and works its way through the instructions one at a time until it drops out of the bottom of the curly braces that follow 'main()'. Once its finished, no more action takes place. The program does exactly what its told, following the exact path the programmer describes using the Control-C code.

Using the journey analogy, the first thing you would most likely do is prepare for the journey. In our code this translates to initialising any variables and system elements used, for example motors and their start position and possibly also coordinating with other parts of the control system. We are then ready to start the journey. In most cases the journey actually never ends until power is turned off because the programmer wants control to run permanently so we create a loop that never ends using the 'while()' commands.

Example:

```
init_vars()
{
    // initialise your variables here
}

init_system()
{
    // initialise the system here
}

main()
{
    init_vars();
    init_system();
    while(TRUE)
    {
        // more code goes here
    }
}
```

The 'while(TRUE)' command loops around the code within the following curly braces forever. Generally this code will consist of a series of tests that if found true will execute some other code you've written.

We are now at a very important point in the code creation process. You may be tempted to simply write all the code you need and place the whole of it with in those curly braces following the 'while(TRUE)' command. This will work but you will be wasting a very, very valuable feature of Control-C, 'the subroutine'. This cannot be emphasised enough. Subroutines are the key to good coding. Use them at every opportunity. They should be used to break the code up into understandable blocks. If a subroutine it not easily understood break it up into more subroutine that are. And please use names for your subroutine that convey sensible meaning to the reader. This process takes a little extra time but the rewards are well worth it.

In the following example we have a simple motor running a conveyor on which product travels. A product sensor senses it and we fire a paint gun by pulsing an output. Using long meaningful names clearly conveys the purpose of the program.

Example:

```
init_vars()
{
    // initialise your variables here
}

init_system()
{
    // initialise the system here
}

startpressed(){ return(input[0]==1); }
runmotor(){ output[1]=1; }
stopmotor(){ output[1]=0; }
productseen(){ return(input[2]==1); }
firepaintgun(){ pulse(3,500); }

main()
{
    init_vars();
    init_system();
    while(TRUE)
    {
        if(startpressed()) runmotor() else stopmotor();
        if(productseen()) firepaintgun();
        // more code goes here
    }
}
```

► Please note: in the example above `return(input[0]==1);` is the same as `return(input[0]);` we have used the longer form for clarity.

In general your program will be much larger than this example but it should not appear any more complex. If you follow the three laws above its function should be clear as you read through it. If you find that it is not, invoke law 2 and break it down until it is.

Programs written using subroutines and meaningful names are easy to understand, modify and maintain and the subroutines can be reused again in other code. This leads to the collection of subroutines in a library that can be pulled out and reused many times over.

## Interrupts

Interrupts are covered in more detail in another document. Here we discuss the use of interrupts in general terms. As mentioned above a program follows exactly the path the programmer describes in the code. Interrupts allow the programmer to respond to errors, changes in digital I/O and COMMS ports without continually testing for them. When interrupts are enabled and when the chosen condition happens the code currently running is suspended and the code in the interrupt subroutine is run. When the interrupt code finishes the suspended code continues on from the point it was stopped.

It is important to understand that the code currently running is suspended when an interrupt occurs. Generally interrupt code will be a high priority but short task that must be performed quickly with little impact on the main program.

Control-C interrupts are prioritised. If two occur simultaneously the highest priority interrupt is run first and the next will be run immediately the first finishes.

## **Building your Program**

You should now be in a position to build your program based on the simple template above. Using the journey method think your way through the path you want the program to follow. Create a series of subroutines to call as you follow this path. Add these subroutines to the code and if possible test them as you go. In this way you can build the program on step at a time and your confidence in its correct operation will grow as you progress. If it does not work as you expect the process of using subroutines will have broken the program into blocks and the area that is not working as expected will be clear to see and therefore easier to correct.

## **Debugging a Program**

To assist you with verification of your programs correct operation Control-C offers several features. Firstly when you compile the program any syntax errors will be reported for correction. When successfully compiled (with breaks on enabled) and downloaded to the target E-Node you can execute the program live in different ways to follow its path and verify its operation.

### **Single Stepping**

Single stepping (F9 key in the Control-C IDE) is the simplest way to see how your program works. If you press F9 directly after downloading the program you can see how the program works. For every F9 press, one line of the program is executed and the next line to be executed is highlighted in the file window.

### **Run to Cursor**

With Run to Cursor (F8 key in the Control-C IDE) the programmer places the text cursor on the line he/she wishes the program to stop. When F8 is pressed the program will run at full speed from its current position until it reaches the break point line. The line is then highlighted and the program is halted. The programmer can then chose the next move. Perhaps placing the cursor on the next line of interest and pressing F8 again.

### **Step Over**

Step over is identical to 'Single Stepping' except that subroutines are run at full speed.

## **User Program, Motion Commands and the Control Loop**

The use of Motion Commands needs a special mention. These commands are used by Motion capable nodes like the 'AXIS' to generate moves on electric motors or similar motive devices. The software that controls the move is special and separate from the User Program. When an User Program executes a Motion Command it takes the command and places it in a special Motion Command Queue. It then continues with the remainder of the program. Motion Commands in the Motion Command Queue are executed by the Control Loop. It will take the commands from the head of the Queue and execute then in turn. The motion command must be fully complete or deliberately stopped before the next command will run.

We give this behaviour special mention because when debugging a program containing Motion Commands and stepping through it might appear to give incorrect results. When stepping through a program it is halted on the highlighted line. However if one of the last executed lines contained a Motion Command it will have been passed to the Motion Command Queue and will be automatically run by the Control Loop. This means the user program will be halted but moves may start or still be running and will continue until completed.

► *Please note: All moves can be cancelled abruptly by issuing a 'Fast Stop All Moves' command from the Control-C IDE.*

► *Please refer to 'E-Nodes Technical.pdf' for detailed information on the Control Loop software.*

## Protecting your program

Having successfully written and debugged your program it should be saved and protected for the future. We recommend the following steps.

1. Always save a copy of the user program on a backup medium remote from your work computer.
2. Always save a copy of the user program on the node running the compiled version. Use the 'Node' menu in the Control-C IDE to do this.
3. After finally programming the node Lock it to prevent unauthorised access and re-programming. Use the 'Node' menu in the Control-C IDE to do this.

## Questions

If you have any questions regarding either this tutorial or any other aspect of using the E-Node range please contact the staff at Etrol Ltd.

Tele: (+44) 01730 816893  
email: [sales@etrol.co.uk](mailto:sales@etrol.co.uk)