



Tutorial

E-Node Communications

This short tutorial discusses the various methods available for communications between nodes on an E-Net network.

Table of Contents

Introduction.....	3
Related Documents.....	3
General Addressing Scheme.....	3
Accessing Local Variables.....	3
Global Variables.....	3
Accessing Global Variables.....	3
Assignment Order.....	4
Broadcast Variables.....	5
The Dedicated Messaging system.....	6
Questions.....	6

Introduction

The E-Net network forms the heart of the E-Node system. It enables the E-nodes to share facilities and communicate with ease. Control-C is the programming language that supports the E-Node and the E-Net network. It provides several means for automatic communications together with commands for dedicated communications. These features are detailed in the dedicated Control-C language document but are given further discussion here in a more general format.

Related Documents

All the subjects covered in this tutorial are also mentioned in the following:

- Control-C Language XpXX.pdf
- Control-C Syntax XpXX.pdf

Please have copies of these documents to hand and make reference to the as required.

General Addressing Scheme

Throughout the Control-C language there is a general scheme used to define the address of the node that is being referred to. It takes the form of a colon followed by the address.

Example:

```
time:2 // refers to the variable 'time' on node address '2'
```

Wherever you use this format it means the compiler will automatically link the associated command with the given address and when the program is running it will automatically communicate with the given address. If the address given is actually the node on which the program is running it will still work perfectly. If the address is omitted the compiler will default to using the address of the node on which the program is running.

Accessing Local Variables

Local variables are declared inside subroutines. They are deliberately restricted to use within those subroutines and cannot be accessed by any program outside of the subroutine. Local variable do not have an associated address.

Global Variables

Global variables come in two types 'Pre-Defined' and 'User-Defined'. Both types always have an associated address and can be accessed by any program on the network. This means that you can read and write directly from one program to global variables in any other program. It is a very fast operation and can be a very powerful tool to help you built efficient programs. It is also very simple. BUT, it is quite possible to get the addressing wrong, resulting in all manner of unexpected behaviour. The following is a direct excerpt from the Control-C language document. Please read it carefully?

Accessing Global Variables

► *This is a very important subject. Correct understanding and use of accessing global variables can have a big impact on the simplicity and operation of user programs.*

All global variables (both pre-defined and user-defined) can be accessed by any program running on any node on the E-net network. To achieve this all global variables have an associated network address.

Pre-defined globals automatically take the address of the node on which they reside. To access the variable the programmer types the name followed by a colon and the address of the node on which the variable resides. Alternatively the colon and address can be omitted and the compiler will assume the address given at the top of the program file window.

User-defined globals are assigned an address when the user program is compiled. The programmer informs the compiler of the address to use by following the variable name with a colon followed by the address. Alternatively this step can be omitted and the compiler will assume the address given at the top of the program file window.

```

global example.CCC
Address 0 0 0 Breaks On ListFile On ImageFile On
var
varone, // this variable resides at address = 0
vartwo:2; // this variable resides at address = 2

main()
{
varone = time:3; // reads time from address = 3
varone = time; // reads time from address = 0
}
Ln 11, Col 0 Modified Not Compiled 0% memory used. Insert

```

Assignment Order

When a program is compiled it starts at the top of the file window and works down through the program. As user-defined global variables are encountered, those with the same address are grouped together. Within each group they are each assigned a unique variable number on a first found, first assigned basis. The first found would be assigned a variable number 'x', the next 'x+1' and so on. This gives rule one for assignment:-

► **RULE ONE:** The order in which global variables are declared is VERY IMPORTANT.

To access a global variable declared in one program from another, the accessing program must know two things. Firstly the address and secondly the variable number. The address is given directly as described above. The variable number is given by the compiler according to the assignment order. This gives rule Two for assignment.

► **RULE TWO:** For a program to access a global variable in another program the variable concerned MUST BE DECLARED IN THE SAME ORDER IN BOTH PROGRAMS.

► If rule Two is not followed the variable numbers will be different for the different programs. When a access is made from one program to another the wrong variable will be accessed.

If you wish to access a global variable or number of global variables from another program, group them together at the very start of the variable declaration process. For example if you have 5 variables in a program declare all 5 at the start of every program from which you wish to access them. It is also recommended that all addresses are explicitly given using the colon + address process. This will help to avoid confusion.

```

prog1.CCC
Address 0 0 1 Breaks On ListFile On
var
prog1a:1, // address = 1
prog1b:1, // address = 1

prog2a:2, // address = 2
prog2b:2[10]; // address = 2
Ln 0, Col 0 Saved Not Compiled % memory u

prog2.CCC
Address 0 0 2 Breaks On ListFile On
var
prog2a:2, // address = 2
prog2b:2[10]; // address = 2
Ln 0, Col 0 Saved Not Compiled % memory u

```

Global (and arrays) declarations over two programs

In the example above prog1.CCC can access the two declared variables of prog2.CCC by simply typing in the full name and address, e.g. prog2a:2. However prog2.CCC has no knowledge of the variables declared in prog1.CCC and cannot access them.

► *Following these guidelines provides the programmer with the ability to access information from anywhere on the network by simply typing in the name and address of the required variable.*

Broadcast Variables

Using broadcast variables is a very efficient way of sharing a limited quantity of data between nodes in a group on the E-Net network. Groups can be linked together using gateway nodes that can be programmed to pass or block broadcast variables. A maximum of two broadcast variables can be set up for regular transmission across all nodes in the group or linked groups. Any node within the group or linked groups can be set up to receive.

When a node is set up to broadcast a variable it will do so once for every 'bcv_interval' (broadcast variable interval), range 1 millisecond to 10 seconds. If the value of the variable has not changed since the last broadcast it is not sent. This keeps receiving nodes up to date with changed data but saves communications bandwidth when the data is unchanged.

To configure a node for transmission the pre-defined variable 'tx_bcv0' or 'tx_bcv1' needs to be assigned the number of the variable to be transmitted (see pre-defined variables above).

To configure a node to receive a transmission the pre-defined variable 'rx_bcv0' or 'rx_bcv1' needs to be assigned the number of the variable to receive the data. (see pre-defined variables above).

An example for the use of broadcast variables would be the regular transmission of a command variable to a large number of nodes. All 48 bits of the variable could be used as control flags or the variable could contain a control code.

Another regular use for broadcast variables is the sharing of a master encoder value across several nodes. The receiving nodes can then perform gear locking or remote position related moves.

Example:-

```
// transmit node setup

bcv_interval = 1;           // transmit every 1 millisec
tx_bcv0 = varnum(enc0);    // transmit encoder 0 value

// receive node setup

rx_bcv0 = varnum(enc1);    // receive value into variable encoder 1
```

The Dedicated Messaging system

Control-C also provides a dedicated messaging system. The scheme employs several commands that send and receive byte sized (8 bits) messages directly from a program to an addressed node. The received messages are stored in the nodes message buffer and can be retrieved on a first in first out basis. The buffer is of a fixed size and an overflow error will result if the buffer is overfilled.

Here are examples of the commands used with the message system:-

```
msg('A':2);           // sends message 'A' to the buffer on address 2

msgin(2);             // used to test if a message is in the buffer on address 2
                    // returns TRUE is a message is present

getmsg(2);            // used to get a message from the buffer on address 2
                    // returns the message if one exists and null (0)
                    // if the timeout expired
```

► *Note: Avoid using a message with a numerical value of zero. If the getmsg() command times out it returns zero automatically.*

Using the dedicated messaging system has the advantage over writing directly to global variables in that all messages are queued in the buffer and the programmer thus has direct control of the order in which message are sent and received. The disadvantage is that additional coding is required and as a result it is slightly slower.

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