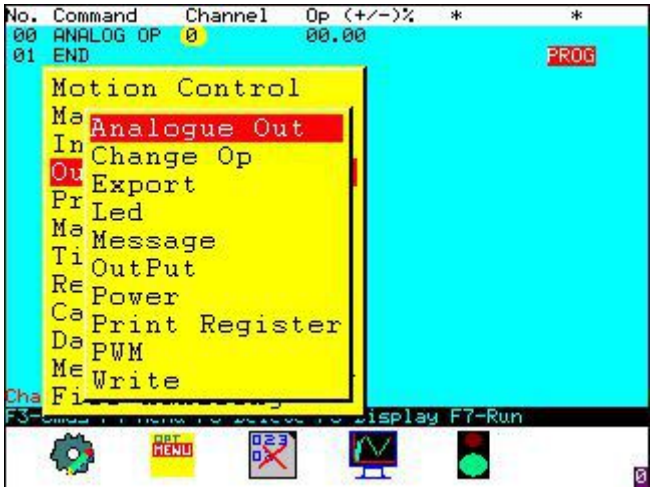
Introduction:

MAP is an easy-to-use programming language, it was originally designed for people that don't do programming, because they thought it just to complicated for them. Now it is the choice of many programmers from beginner to expert. MAP is probably one of the most flexible programming languages in the industrial market. It includes full plc programming capabilities such as driving outputs and checking inputs, also it includes motion control, CNC, data logging and other useful commands that enable CNC, motion control, data logging to be seamlessly integrated with input output commands. MAP started as a simple language for simple motion control applications such as “pick and place”. Customers found it easy to use and started asking for more and more commands. Today there are more than 102 commands some of which have 12 options.

Making a programming language easy-to-use is key in the industrial market. The commands can be written on the controller and there are no license fees. As a language evolves, there are no investments needed for compilers or programming tools. Commands are entered via a drop-down menu, on the controller it's self, each of the commands have a descriptive name and are organised in groups. For example there is an output group which as you would expect controls outputs such as the controllers output drivers, the on-screen LEDs, printing to the screen, control of the pulse with modulators etc. There is a command for each of the above options as shown below:

1. For example to set the voltage on one of the analogue outputs select analogue out from the output group pull down menu, by using the arrow keys and selecting analogue out, then push the enter key. The menu is shown below:



2. All the entries in map are displayed in spreadsheet style, this means that there are rows and columns. As you navigate around the screen, there is a help message explaining what the command does and what input is expected. On the top line of the screen is an explanation of each column that is unique to each command. Therefore often map is so intuitive, people don't read the manual. Using the arrow keys it is easy to navigate round the screen, in this case using the right-hand arrow key in the first column, displays channel and the help message shows that the selection is 0 to 3 (there are four analogue channels to choose from) the next column represents the output voltage and the help message shows the user to enter the voltage they require.

Two years ago TRM gave science classes at Ormskirk School, and after a 20 minute introduction, 12 groups of three students were challenged to draw a square or rectangle on a plotter by writing programs in map. Each group only five minutes complete work as one of 12 experiments automation experiments. The results were impressive all groups succeeded in producing a square or rectangle, one group wanted to produce two perfectly overlapping rectangles, and another produced a hexagon this shows that these 14 and 15 year olds found programming easy with MAP.

Map consists of a command followed by data fields, each command occupies the same space, this is called Fixed Width Command Set (FWCS). The idea of a FWCS is that the programme will run fast and is predictable, an important feature when programming for industrial equipment. Within each of the data fields the map editor provides help to assist the programmer by explaining what each of the data fields holds. As an example of how intuitive MAP is, let us look at a simple program to move, axis to position. Let us assume that we want to do a 3 axis move to position X=100, Y=50 and Z=25 (100,50,25). From the pull-down menu select the Motion Control group, press <ENTER> to reveal the motion control commands. There are two main options that we might want to use these are MOVE and RAPID_MOVE. The Rapid_Move is a full speed move while move obeys the speed or feed command. Let us select MOVE, under the heading “Axis X” we type 100 using the right arrow arrow key the cursor will move to the next column where the enter 50 and move the the axis Z colum and enter 25. Next time we use a motion control move command or rapid move the positions of the last command are already entered to save us time. That is how all commands are entered, MAP is as orthogonal and efficient as possible. To run the program we have created just press F7 the traffic light key.

Some commands have “options” called “Condition Codes” these condition codes have an effect on the way the commands operate. A good example of how this works is the way that the load_reg command which is used to load values into registers. Rather than having 12 different commands to load different conditions (such as a fixed value or another register), there are 12 different conditions or modes available for this command. These options are selected by positioning the cursor over the name of the command and hitting the Enter key. If there are options for this command a pull down menu will appear and the user can select an item from the menu.

This approach makes it very easy to program there is no complex symbols each command is explained by the help messages and the command itself may be written in English or Spanish (in future languages are planned). MAP is extremely efficient programming language and unlike other industrial languages automatically takes care of processing emergency stops etc. Even homing axis in many cases is dealt with automatically as are errors and emergency stops. Each command is checked on entry, for example if you typed 10,000 on the above example and the axis was only 2000 mm long the system would refuse to accept the command an an error message would inform you of the axis limits.

After you have written your program all you need to do is select the traffic light symbol using the F7 key to run your program the map interpreter will automatically check your program and if it finds errors it will inform you why you cannot run. The intention is to make map as safe as possible. Although it is not possible to make a language completely safe, after all if you tell map to make a move that with hit something on a milling machine table for instance, map cannot protect you since it does not know what has been left on the table however it will not let you go outside the limits of the machine so it is more difficult to cause damage.

Data types:

If you enter a value is a data field such as as 123 this is treated as a **constant** which means that the data is fixed and can only be changed by the programmer. There are many times where you might well need to calculate the position of an axis, rather than having a constant position, this allows tool wear to be compensated for example. To achieve this we need to use a **variable,** in computers this is a term to describe some memory that can hold variable at data. The smallest data in computing terms is the "**Bit**" this is like a single switch it can be On or Off. The next type of variable that is commonly used is the "**Byte**" this is 8 bits, Word is 16 bits and long is 32 bits please see the table below:

Name	Width bits	Comment
Bit	1	Can hold 0 to 1 or 0 to -1
Byte	8	Can be signed or unsigned
Word	16	Can be signed or unsigned
Long	32	Can be signed or unsigned
Long Long	64	Not supported on MAP

Figure 2 common integer data types

Integers are a type of variable that hold a whole number, it does not have a fractional part. For example 21 is a whole number and can be stored in an integer. Often we need to use a number that can hold fractional numbers such as 21.35 or 0.2, an variable that can hold this type of value is called a "**floating point**". A floating point variable is typically 32 bits, there are also 64 bit double floating point, these offer more precision, but in most industrial applications these are not needed. Printable charters normally stored in "**strings**". A string is a collection (called an array) of bytes stored at consecutive memory locations. A string can hold any digital data!

H	e	l	l	o	space	M	A	P	0

Figure 3: An example of a string

In the above example, Hello MAP is shown. The memory offset of the string's first charter "H" is 0 and the last character is "P" is 8. Note that the end of the string has a 0, this indicates that the string is terminated and this techniques is called a "Null terminator".

MAP Data fields and registers:

MAP makes available 256 registers, a register is a 4 byte wide variable and some of you may have noticed that with the exception of double floating point all integer and floating point types could fit into a 4 byte block. In MAP the registers can hold the following types:

Type	Width bits	Comment
Unsigned long	32	0 to 4294967296
Long	32	−2147483647, +2147483647
Float	32	$(1-2^{-24})\times 2^{128}\approx 3.402823\times 10^{38}$
String	32 (4 bytes each register)	(usually 4 registers) 16 chars
Pointer	32	Used to locate variables in memory

Figure 4: MAP data fields types:

Why did we need to tell you the above? The answer is that the registers are actually just variables that can be programmed to be different variable types and most commands will accept data from registers or constant data. Constant data is data entered by the programmer which is not intended to change, for example in our above move example we positioned to 100, 50, 25. We could however positioned to registers. Map has 256 registers, we can write the value into a register and position to the content of the register. This has very many advantages. For example it is possible to change positions easily by writing value into a register. The major advantage of this approach is that your program can use the maths command to calculate a position which to move, for example on a grinding wheel it is possible to automatically compensate for wear on the wheel by adding a factor to a register each time a grinder makes a pass over a target material. Using registers gives complete flexibility and yet it is still easy to understand.

When you want to write a program to position an axis using registers, it is better to use floating point registers rather than integer. Initialising the registers as you want them is easy as well there is a command to assign a type to a register or registers. By default all registers are initialised as long that is 32 bit signed integer. The command used to set up registers is the Register_type command, this command uses start to end parameters and a condition code that is menu driven to select the variable type. Good programming states that you should plan your register use so you might decide you need 8 floating point registers, and you might decide to set aside 0 to 9 as floating point (giving you 2 spare just in case you need more than you thought. Always use at least one register as a temporary for calculations, then you will not accidentally erase a register that you needed to save. If we look at the same example we used before but using two registers for X and Y we can write:

Load_register	R000	100	R001	50
Move	R000	R001	25	0

In this example we did not need to use floating point so we loaded 100 into register 0 and 50 into register 1, then we used a Move command to position to Register 0 (for x), Register 1(for Y) and the Z axis was positioned to a constant 25. Note that the registers are written in blue to make the screen easier to read.

On the next newsletter we will cover some more programming examples using registers and what registers can be used for. Please feel free to send in enquiries about your particular programming issues and we will attempt to get back to you as soon as possible.