Introduction to Defined Subroutines

MMBasic version 3.1 introduced defined subroutines. This is an important feature in organising programs so that they are easy to modify and read.

A defined subroutine is simply a block of programming code that is contained within a module and can be called from anywhere within your program. It is the same as if you have added your own command to the language.

For example, assume that you would like to have the command FLASH added to MMBasic, its job would be to flash the power light on the Maximite. You could ask the author of the language to add it or you could define a subroutine like this:

```
Sub FLASH
  Pin(0) = 1
  Pause 100
  Pin(0) = 0
End Sub
```

Then, in your program you just use the command FLASH to flash the power LED. For example:

```
IF A <= B THEN FLASH
```

If the FLASH subroutine was in program memory you could even try it out at the command prompt, just like any command in MMBasic.

The definition of the FLASH subroutine can be anywhere in the program but typically it is at the start or end. If MMBasic runs into the definition while running your program it will simply skip over it.

Arguments

Defined subroutines can have arguments (sometimes called parameter lists). In the definition of the subroutine they look like this:

```
Sub MYSUB (arg1, arg2$, arg3)
  <statements>
  <statements>
End Sub
```

And when you call the subroutine you can assign values to the arguments. For example:

```
MYSUB  23, "Cat", 55
```

Inside the subroutine `arg1` will have the value `23`, `arg2$` the value of `"Cat"`, and so on. The arguments act like ordinary variables but they exist only within the subroutine and will vanish when the subroutine ends. You can have variables with the same name in the main program and they will be different from arguments defined for the subroutine (at the risk of making debugging harder).

When calling a subroutine you can supply less than the required number of values. For example:

```
MYSUB  23
```

In that case the missing values will be assumed to be either zero or an empty string. For example, in the above case `arg2$` will be set to `""` and `arg3` will be set to zero. This allows you to have optional values and, if the value is not supplied by the caller, you can take some special action.

Local Variables

Inside a subroutine you will need to use variables for various tasks. In portable code you do not want the name you chose for such a variable to clash with a variable of the same name in the main program.

To this end MMBasic (3.1 and later) allows you to define a variable as LOCAL. For example, this is our FLASH subroutine but this time we have extended it to take an argument (nbr) that specifies how many times to flash the LED.

```
Sub FLASH ( nbr )
  Local count
  For count = 1 To nbr
    Pin(0) = 1
    Pause 100
    Pin(0) = 0
    Pause 150
  Next count
End Sub
```

The counting variable (`count`) is declared as local which means that (like the argument list) it only exists within the subroutine and will vanish when the subroutine exits. You can have a variable called `count` in your main program and it will be different from the variable `count` in your subroutine.

If you do not declare the variable as local it will be created within your program and be visible in your main program and subroutines, just like a normal variable.

You can define multiple items with the one LOCAL command. If an item is an array the LOCAL command will also dimension the array (ie, you do not need the DIM command). For example:

```
LOCAL NBR, STR$, ARR(10, 10)
```

You can also use local variables in the target for GOSUBs. For example:

```
       ...
  GOSUB MySub
       ...
MySub:
   LOCAL X, Y
   FOR X = 1 TO ...
   FOR Y = 5 TO ...
  <statements>
   RETURN
```

The variables X and Y will only be valid until the RETURN statement is reached and will be different from variables with the same name in the main body of the program.

Passing Arguments by Reference

If you use an ordinary variable (ie, not an expression) as the value when calling a subroutine, the argument within the subroutine will point back to the variable used in the call and any changes to the argument in the subroutine will also be made to the supplied variable. This is called passing arguments by reference.

For example, you might define a subroutine to read the temperature from a sensor, as follows:

```
Sub GetTemp ( PinNbr, Value )
  <statements>
  Value = <code to get the temperature>
End Sub
```

In your calling program you would use a variable for the second argument as follows:

```
GetTemp 5, Temp
Print "The temperature is:" Temp
```

The change to `Value` in the subroutine was copied to the variable `Temp` in the calling subroutine and was therefore available after the subroutine exited. This provides an easy way for a subroutine to pass values back to the caller without having to resort to normal variables which are awkward to use and non portable,

If the argument is an expression and not simply a variable the subroutine will not be able to change its value. For example:

```
GetTemp 5, Temp + 1
```

or

```
GetTemp 5, (Temp)
```

Either will prevent the subroutine from changing the value of `Temp`.

Another Example

The latest update to the MMBasic Library contains a version of the LCD driver written using subroutines. The original code has been used by many people but it was rather messy because defined subroutines were not available.

The new version using subroutines is much easier to use. This is how you would use it:

```
InitLCD                        ' setup the display
PrintLCD 1, "Hello World"      ' write to the first line
PrintLCD 2, "Maximite LCD"     ' write to the second line
```

You do not have to worry about what is going on inside the subroutines and you do not have to be concerned about variables in a subroutine affecting your main program. It is exactly the same as if an LCD driver had been added to the language.

Additional Notes

- For what it is worth, the MMBasic implementation of defined subroutines substantially meets the specifications in the ANSI Standard for Full BASIC (X3.113-1987) or ISO/IEC 10279:1991.
- You cannot use arrays in the subroutine's argument list (although the caller can use them). The MMBasic Library has a good example of this in the BBLSORT.BAS program.
- Brackets around the argument list in both the caller and the definition are optional.
- There can be only one END SUB for each SUB definition. To exit a subroutine early you should use EXIT SUB.
- The defined subroutine is intended to be a portable lump of code that you can insert into any program. This is why the full screen editor has the CTRL-F keys for inserting another file. The idea is that you can keep your defined subroutines in a file and whenever you need them you can quickly insert them using CTRL-F.