Introduction to MMBasic V3.X

Version 3.0 introduced a revolutionary change in MMBasic compared to version 2.7B. In addition to dozens of small improvements and bug fixes version 3.0 also implemented an updated core of the BASIC interpreter which provides the basis for many significant changes that would have previously been impossible to implement.

Originally MMBasic was written to emulate the early BASICs like Microsoft's MBASIC. This was not a problem if you were over 50 and grew up in the days of the Tandy TRS-80 and Commodore 64 as you were used to the languages of that era and their idiosyncrasies. But, with the huge popularity of MMBasic, it needed to move beyond the days of the TRS-80 to match modern programming standards.

This has been done with a rewrite of the interpreter core that enables features such as speed improvements, the removal of line numbers and the addition of labels, user defined subroutines/functions and more.

Line Numbers

The most significant change in version 3.X is that line numbers are now no longer required.

For example:

MMBasic 2.X

```
                ...
324 ' GET THE INPUTS
330 IF Q=0 THEN GOTO 700
331 A=A+Q : S=S-Y*Q : C=0
334 GOTO 400
336 ' GET THE SELL STATUS
340 PRINT "DO YOU WISH TO SELL";
341 INPUT Q
345 IF Q<0 THEN GOTO 340
350 A=A-Q : S=S+Y*Q : C=0
400 PRINT
410 PRINT "HOW MANY";
411 INPUT Q
420 IF Q<=S THEN GOTO 324
430 S=S-Q : C=1 : PRINT
                ...
```

MMBasic 3.X

```
                ...
      ' Get the inputs
GetI: If Q=0 Then GoTo Abort
      A=A+Q : S=S-Y*Q : C=0
      GoTo GetQ

      ' Get the sell status
GetQ: Print "DO YOU WISH TO SELL";
      Input Q
      IF Q<0 Then GoTo GetQ
      A=A-Q : S=S+Y*Q : C=0
      Print
      Print "HOW MANY";
      Input Q
      IF Q<=S Then GoTo GetI
      S=S-Q : C=1 : Print
                ...
```

Removing line numbers makes it much easier to write clear and meaningful programs. You can indent lines for clarity and the ability to insert blank lines allows you space out the program so that it is not one solid block of text.

Version 3.X is fully backward compatible with 2.X so you can still use line numbers if you need to and old programs written for 2.X will run exactly as before.

Labels

Instead of using line numbers as the target for GOTO, GOSUB, etc in version 3.X you can use a label.

In the above example GetQ is a label. A label is used exactly like a line number but it is much more useful to anyone who is reading the program. For example:

```
GOTO ErrorHandler
```

Is more meaningful than:

```
GOTO 1290
```

When used to mark a line the label must be terminated with the colon (:) character. For example:

```
ErrorHandler:  Print "Error …
```

A label has the same specifications as a variable (up to 32 characters including letters, numbers, the underscore and period). The only caveat is that a label cannot be the same as a command because that could confuse the interpreter as the colon character (:) is used to both terminate a label and as a separator between commands.

Version 3.X includes a cache which remembers the location of a label in the program and, as a result, jumping to a label is much faster than jumping to a line number. For that reason labels are preferred and should be used instead of line numbers.

Mixing Line Numbers and Labels

Old programs that used line numbers will operate as before.

For example:

```
100 ' old style program
200 PRINT nbr
400 nbr = nbr + 1
500 if nbr < 5 GOTO 200
```

You can still use line numbers as the target for GOTO, GOSUB, etc even if you are not otherwise using line numbers.

For example:

```
' using line numbers as a target
200   Print nbr
      nbr = nbr + 1
      if nbr < 5 GoTo 200
```

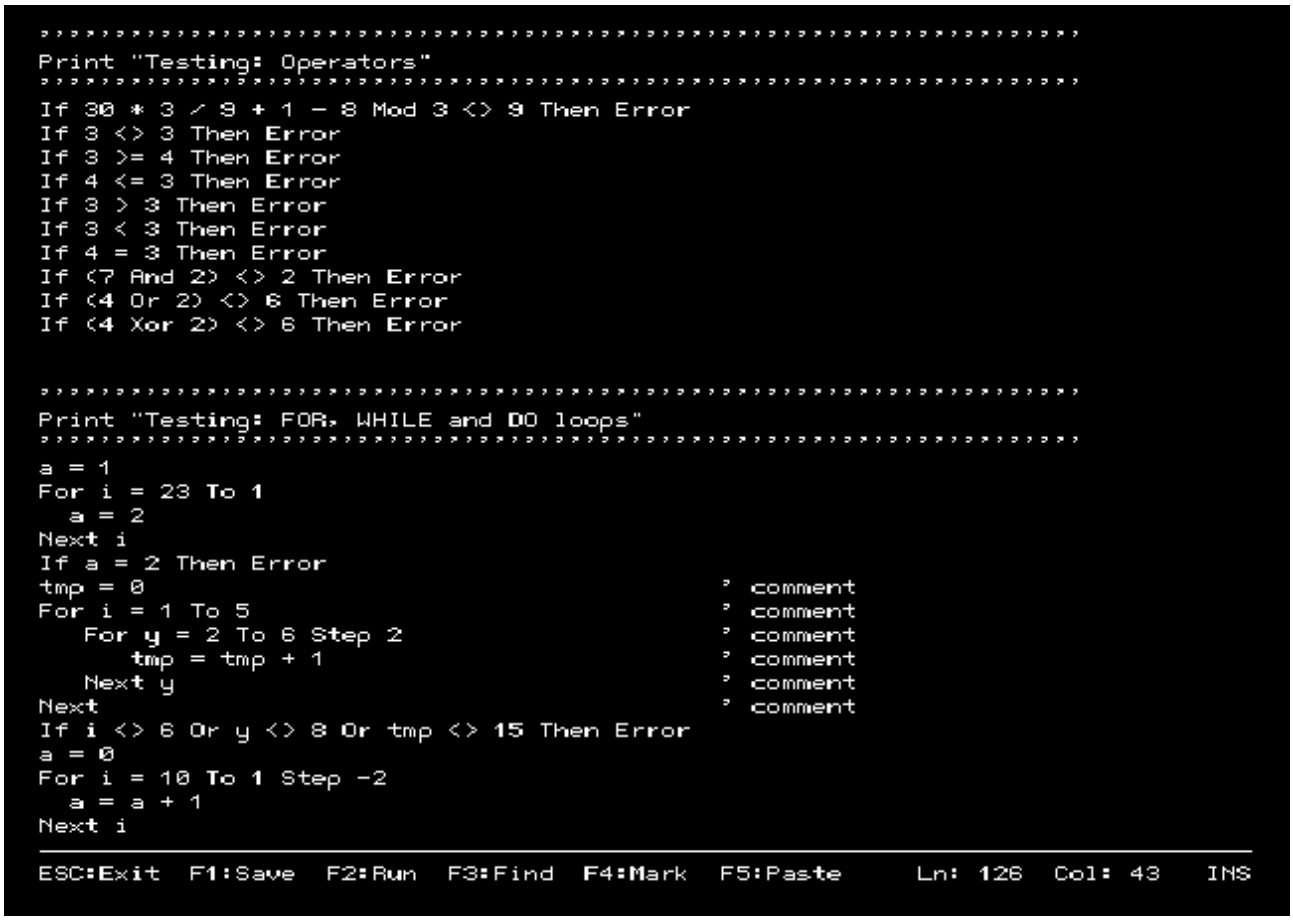
You can also mix labels and line numbers.

For example:

```
100 ' old style program that uses labels
200 Label: Print nbr
400 nbr = nbr + 1
500 if nbr < 5 GoTo Label
```

Full Screen Editor

If you are using line numbers you can still enter programs at the command prompt the old way by prefixing the command with a line number. However, it is much easier to use the full screen editor in version 3.X.



The editor is invoked with the EDIT command. If you run it on its own it will automatically start editing the program currently in memory. If you run it with a file name (eg, EDIT "FILE.DAT") it will edit that file while leaving the program in memory untouched. This last feature is very handy for examining data files or editing font files while you are developing a program.

When you are in the editor the various editing keys will work the way that you would expect. For example, the Up Arrow key will move up a line, the Delete key will delete a character and so on. You can quickly move through the text from the beginning to the end using the Home/End and Page Up/Down keys.

Using the full screen editor makes programming in MMBasic a much more fun experience. You can run the editor, make your changes and press the F2 key. The program will be saved back to program memory and automatically run. If it fails with an error you can run the editor again and the editor will automatically position the cursor at the line that caused the error.

As a result the edit-run-edit cycle is very fast and productive.

The full screen editor includes many other productivity aids such as cut and paste, searching, automatic indenting and the ability to insert code from another file.

The editor will also work with a vt100 compatible terminal emulator over USB. So, if you are using the mini Maximite or the UBW32 you can still conveniently edit the program held in memory. It has been tested with Terra Term and this is the recommended terminal emulation software. Note that Terra Term must be configured for an 80x36 sized screen.

Other Changes

There are many more features introduced in 3.0 including:

- The ability to use full path names when opening files and changing directories (for example: RUN "\BASIC\TEST\FILE1.BAS").
- New commands and functions such as DIR$(), COPY, DEG(), RAD(), etc.
- Better memory management that provides more general memory (used by arrays, etc) when the video is turned off or a mode such as composite is selected.

These and others are listed in the Change Log and are fully documented in the updated User Manual.

Future Additions

The new BASIC core introduced in version 3.0 also provides the basis for implementing future improvements that will further modernise MMBasic. One example is user defined commands and functions. For example in the future you might program:

```
x = Square(y)
    ...
Def Function Square(arg)
    If arg < 0 Error
    Square = arg * arg
    Return
```

These features have not been defined or implemented at this time but they are something to look forward to.