

PicoMite

A Raspberry Pi Pico Running the MMBasic BASIC Interpreter

User Manual

MMBasic Ver 5.08.00

Revision 4

For updates to this manual and more details on MMBasic go to

<http://geoffg.net/picomite.html>

and <http://mmbasic.com>

PicoMite

运行MMBasic BASIC解释器 的树莓派Pico

用户手册

MMBasic版本 5.08.00

修订版 4

有关本手册的更新和MMBasic的更多详细信息，请访问

<http://geoffq.net/picomite.html>

以及<http://mmbasic.com>

About

Peter Mather (matherp on the Back Shed Forum) led the project, ported MMBasic to the Raspberry Pi Pico and wrote the drivers for its hardware features. The MMBasic interpreter and this manual was written by Geoff Graham (<http://geoffg.net>). In addition, many others have supported the project with specialised code, testing and suggestions.

Support

Support questions should be raised on the Back Shed forum (<http://www.thebackshed.com/forum/Microcontrollers>) where there are many enthusiastic MMBasic users who would be only too happy to help. The developers of the PicoMite firmware are also regulars on this forum.

Copyright and Acknowledgments

The PicoMite firmware and MMBasic is copyright 2011-2021 by Geoff Graham and Peter Mather 2016-2021.

1-Wire Support is copyright 1999-2006 Dallas Semiconductor Corporation and 2012 Gerard Sexton.

FatFs (SD Card) driver is copyright 2014, ChaN.

WAV and FLAC file support is copyright 2019 David Reid.

JPG support is thanks to Rich Geldreich

The pico-sdk is copyright 2021 Raspberry Pi (Trading) Ltd.

TinyUSB is copyright tinyusb.org

LittleFS is copyright Christopher Haster

Thomas Williams and Gerry Allardice for MMBasic enhancements

The compiled object code (the .uf2 file) for the PicoMite is free software: you can use or redistribute it as you please. The source code is available from GitHub (<https://github.com/UKTailwind/PicoMite>) and can be freely used subject to some conditions (see the header on the source files).

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

This Manual

The author of this manual is Geoff Graham with input by Peter Mather, Harm de Leeuw, Mick Ames and many others on The Back Shed forum. It is distributed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Australia license (CC BY-NC-SA 3.0)

关于

项目由彼得·马瑟（Back Shed论坛上的matherp）领导，他将MMBasic移植到树莓派Pico并编写了其硬件功能的驱动程序。MMBasic解释器和本手册由Geoff Graham编写（<http://geoffg.net>）。此外，还有许多人通过专业代码、测试和建议支持该项目。

支持

支持问题应在Back Shed论坛上提出（<http://www.thebackshed.com/forum/Microcontrollers>）那里有许多热情的MMBasic用户，他们非常乐意提供帮助。PicoMite固件的开发者也是这个论坛的常客。

版权和致谢

PicoMite固件和MMBasic的版权归Geoff Graham和彼得·马瑟2011-2021年所有。

1-Wire支持的版权归达拉斯半导体公司1999-2006年和Gerard Sexton 2012年所有。

FatFs (SD卡)驱动程序的版权归ChaN 2014年所有。

WAV和FLAC文件支持的版权归David Reid 2019年所有。

JPG支持感谢Rich Geldreich。

pico-sdk的版权归树莓派（交易）有限公司2021年所有。

TinyUSB的版权归tinyusb.org所有。

LittleFS的版权归Christopher Haster所有。

Thomas Williams和Gerry Allardice为MMBasic的增强做出了贡献。

PicoMite的编译对象代码（.uf2文件）是自由软件：您可以随意使用或重新分发。源代码可从GitHub获取（<https://github.com/UKTailwind/PicoMite>），并可以在某些条件下自由使用（请参见源文件的头部）。

本程序的发布希望能对您有所帮助，但不提供任何保证，甚至不包括适销性或特定用途适用性的隐含保证。

本手册

本手册的作者是杰夫·格雷厄姆，彼得·马瑟、哈姆·德·李乌、米克·阿梅斯以及其他许多在The Back Shed论坛上提供意见的人士也参与了编写。本手册根据创意共享署名-非商业性使用-相同方式共享3.0澳大利亚许可证（CCBY-NC-SA 3.0）发布。

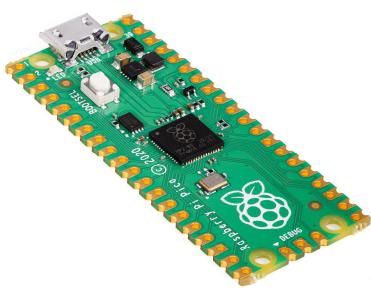
Contents

Introduction.....	4
Getting Started	5
Quick Start Tutorial.....	8
PicoMite Hardware	9
Using MMBasic.....	11
Full Screen Editor.....	15
Program and Data Storage.....	17
Variables and Expressions	24
Subroutines and Functions.....	29
Using the I/O pins.....	32
Sound Output	35
Special Device Support	38
Display Panels.....	45
Touch Support.....	52
Graphics Commands and Functions	54
PicoMite Advanced Graphics	59
Advanced Graphics Programming Techniques	67
MMBasic Characteristics	73
Predefined Read Only Variables	75
Options	79
Commands	86
Functions.....	137
Obsolete Commands and Functions	152
Appendix A – Serial Communications	153
Appendix B – I2C Communications.....	155
Appendix C – 1-Wire Communications.....	158
Appendix D – SPI Communications.....	159
Appendix E – Regex Syntax.....	161
Appendix F – The PIO Programming Package.....	163
Appendix G – Programming in BASIC - A Tutorial	170

目录

介绍.....	4
开始使用.....	5
快速入门教程.....	8
PicoMite 硬件.....	9
使用MMBasic.....	11
全屏编辑器.....	15
程序和数据存储.....	17
变量和表达式.....	24
子程序和函数.....	29
使用I/O引脚.....	32
声音输出.....	35
特殊设备支持.....	38
显示面板.....	45
触摸支持.....	52
图形命令和函数.....	54
PicoMite 高级图形.....	59
高级图形编程技术.....	67
MMBasic特性.....	73
预定义只读变量.....	75
选项.....	79
命令.....	86
函数.....	137
过时的命令和函数.....	152
附录A – 串行通信.....	153
附录B – I2C通信.....	155
附录C – 1-Wire通信.....	158
附录D – SPI通信.....	159
附录E – 正则表达式语法.....	161
附录F – PIO编程包.....	163
附录G – BASIC编程 - 教程.....	170

Introduction



The PicoMite is a Raspberry Pi Pico running the MMBasic interpreter.

MMBasic is a Microsoft BASIC compatible implementation of the BASIC language with floating point, integer and string variables, arrays, long variable names, a built in program editor and many other features.

Using MMBasic you can control the I/O pins and use communications protocols such as I²C or SPI to get data from a variety of sensors. You can display data on low-cost colour LCD displays, measure voltages, detect digital inputs and drive output pins to turn on lights, relays, etc.

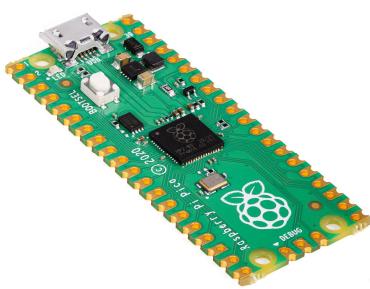
The PicoMite firmware is totally free to download and use.

The emphasis with MMBasic is on ease of use and development. The development cycle is very fast with the ability to instantly switch from edit to run. Errors are listed in plain English and when an error does occur a single keystroke will invoke the built-in editor with the cursor positioned on the line that caused the error.

In summary the features of the PicoMite are:

- **The BASIC interpreter is full featured** with floating point, 64-bit integers and string variables, long variable names, arrays of floats, integers or strings with multiple dimensions, extensive string handling and user defined subroutines and functions. Typically, it will execute a program up to 100,000 lines per second. MMBasic allows the embedding of compiled C programs for high performance functions and the running program can be protected from being listed or modified by a PIN number.
- **Support for all Raspberry Pi Pico input/output pins.** These can be independently configured as digital input or output, analog input, frequency or period measurement and counting. Within MMBasic the I/O pins can be dynamically configured as inputs or outputs. MMBasic commands will generate pulses and can be used to transfer data in parallel. Interrupts can be used to notify when an input pin has changed state. PWM outputs can be used to create various sounds, control servos or generate computer-controlled voltages for driving equipment that uses an analogue input (e.g. motor controllers). In addition, pins that are not exposed on the Raspberry Pi Pico can be accessed using MMBasic allowing it to be used on other modules that use the RP2040 processor.
- **Support for TFT LCD display panels** using parallel, SPI and I²C interfaces allowing the BASIC program to display text and draw lines, circles, boxes, etc in up to 16 million colours. Resistive touch controllers on these panels are also supported allowing them to be used as sophisticated input devices. LCD panels can cost as little as US\$7 and provide a low cost, high tech graphical user interface. For higher speed and greater resolution parallel interface TFT screens are also supported.
- **Flexible program and data storage.** Programs and data can be read/written from an internal file system (approx 688KB) or to an externally connected SD Card up to 32GB formatted as FAT16 or FAT32. This includes opening files for reading, writing or random access and loading and saving programs. The SD Card can also be read/written on personal computers running Windows, Linux or the Mac operating system.
- **Programming and control is done via the USB interface.** All that is needed is a laptop/desktop computer running a VT100 terminal emulator. Once the program has been written and debugged the PicoMite can be instructed to automatically run the program on power up with no user intervention.
- **A full screen editor** is built into the PicoMite and can edit the whole program in one session. It includes advanced features such as colour coded syntax, search and copy, cut and paste to and from a clipboard.
- **Programs can be easily transferred** from a desktop or laptop computer (Windows, Mac or Linux) using the XModem protocol or by streaming the program over the serial console input.
- **A comprehensive range of communications protocols** are implemented including I²C, asynchronous serial, RS232, SPI and 1-Wire. These can be used to communicate with many sensors (temperature, humidity, acceleration, etc) as well as for sending data to test equipment.
- **The PicoMite has built in commands** to directly interface with infrared remote controls, the DS18B20 temperature sensor, LCD display modules, battery backed clock, numeric keypads and more.
- **Power requirements are 2.0 to 5.5 volts** at 10 to 42 mA.

介绍



PicoMite是运行MMBasic解释器的树莓派Pico。

MMBasic是与Microsoft BASIC兼容的BASIC语言实现，支持浮点数、整数和字符串变量、数组、长变量名、内置程序编辑器以及许多其他功能。

使用MMBasic，您可以控制I/O引脚，并使用如I²C或SPI等通信协议从各种传感器获取数据。您可以在低成本彩色LCD显示器上显示数据，测量电压，检测数字输入，并驱动输出引脚以打开灯、继电器等。PicoMite固件完全免费供下载和使用。

MMBasic强调易用性和开发的便利性。开发周期非常快，可以即时从编辑切换到运行。错误以简单英语列出，当发生错误时，单击一个按键即可调用内置编辑器，光标将定位在导致错误的行上。

总之，PicoMite的特点包括：

- **BASIC解释器功能齐全**支持浮点数、64位整数和字符串变量，长变量名，浮点数、整数或字符串的多维数组，广泛的字符串处理以及用户定义的子程序和函数。通常，它可以以每秒高达100,000行的速度执行程序。MMBasic允许嵌入编译的C程序以实现高性能功能，并且运行中的程序可以通过PIN码保护，防止被列出或修改。
- **支持所有树莓派Pico输入/输出引脚。**这些引脚可以独立配置为数字输入或输出、模拟输入、频率或周期测量和计数。在MMBasic中，I/O引脚可以动态配置为输入或输出。MMBasic命令将生成脉冲，并可用于并行传输数据。可以使用中断来通知输入引脚状态的变化。PWM输出可用于创建各种声音、控制伺服电机或生成计算机控制的电压，以驱动使用模拟输入的设备（例如电机控制器）。此外，未在树莓派Pico上暴露的引脚可以通过MMBasic访问，从而使能够用于其他使用RP2040处理器的模块。
- 支持使用并行、SPI和I²C接口的**TFT LCD**显示面板，允许BASIC程序显示文本并绘制线条、圆形、框等，最多可显示1600万种颜色。这些面板上的电阻触控控制器也得到支持，使其可以作为复杂的输入设备使用。
LCD面板的成本低至7美元，提供低成本、高科技的图形用户界面。对于更高的速度和更大的分辨率，也支持并行接口的TFT屏幕。
- **灵活的程序和数据存储。**程序和数据可以从内部文件系统读取/写入（约688KB）或者写入格式为FAT16或FAT32的外部连接SD卡，最大支持32GB。这包括打开文件进行读取、写入或随机访问，以及加载和保存程序。SD卡也可以在运行Windows、Linux或Mac操作系统的个人计算机上进行读写。
- **编程和控制通过USB接口完成。**所需的仅仅是一台运行VT100终端仿真器的笔记本/台式计算机。一旦程序编写并调试完成，PicoMite可以被指示在开机时自动运行该程序，无需用户干预。
- **PicoMite内置了一个全屏编辑器**，可以在一个会话中编辑整个程序。它包括高级功能，如彩色编码语法、搜索以及剪切、复制和粘贴到剪贴板。
- 程序可以通过XModem协议或通过串行控制台输入流式传输，轻松从台式机或笔记本电脑（Windows、Mac或Linux）传输。
- 实现了一系列全面的通信协议，包括I²C、异步串行、RS232、SPI和1-Wire。这些可以用于与许多传感器（温度、湿度、加速度等）进行通信，以及向测试设备发送数据。
- **PicoMite内置命令**可直接与红外遥控器、DS18B20温度传感器、LCD显示模块、电池备份时钟、数字键盘等进行接口。
- **电源要求为2.0至5.5伏**在10至42毫安之间。

Getting Started

Loading the PicoMite Firmware

The Raspberry Pi Pico comes with its own built in firmware loader that is easy to use. Just follow these steps:

- Download the PicoMite firmware from <http://geoffg.net/picomite.html> and unzip the file. Identify the firmware which should be named something like "PicoMiteV5.xx.xx.uf2".
- Using a USB cable plug the Raspberry Pi Pico into your computer (Windows, Linux or Mac) **while holding down the white BOOTSEL button** on the Raspberry Pi Pico.
- The Raspberry Pi Pico should connect to your computer and create a virtual drive (the same as if you had plugged in a USB memory stick) called "RPI-RP2". This drive will contain two files which you can ignore.
- Copy the firmware file (with the extension .uf2) to this virtual drive.
- When the copy has completed the Raspberry Pi Pico will restart and create a virtual serial port on your computer. The LED on the Raspberry Pi Pico will blink slowly indicating that the PicoMite firmware with MMBasic is now running.

While the virtual drive created by the Raspberry Pi Pico looks like a USB memory stick it is not, the firmware file will vanish once copied and if you try copying any other type of file it will be ignored.

Loading the PicoMite firmware will erase the flash memory including the current program, any programs saved in flash memory slots and all saved variables. So make sure that you backup this data before you upgrade the firmware.

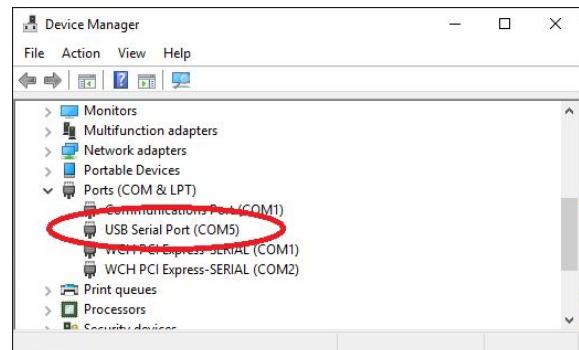
It is possible for the flash memory to be corrupted resulting in unusual and unpredictable behaviour. In that case you should load the firmware file https://geoffg.net/Downloads/picomite/Clear_Flash.uf2 which will reset the Raspberry Pi Pico to its factory fresh state, then you can reload the PicoMite firmware.

Virtual Serial Port

The virtual serial port created by the PicoMite firmware acts like a normal serial port but it operates over USB.

Windows 10 includes a driver for this virtual serial port but with other versions you may have to load a driver to make it work with the operating system (see below).

Once this is done you should note the port number created by your computer for the virtual serial connection. In Windows this can be done by starting Device Manager and checking the "Ports (COM & LPT)" entry for a new COM port as shown on the right.



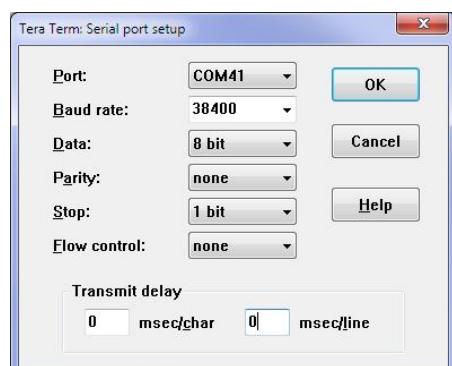
Terminal Emulator

You also need a terminal emulator program on your desktop computer. This program acts like an old fashioned computer terminal where it will display text received from a remote computer and any key presses will be sent to the remote computer over the serial link. The terminal emulator that you use should support VT100 emulation as that is what the editor built into the PicoMite expects.

For Windows users it is recommended that you use Tera Term as this has a good VT100 emulator and is known to work with the XModem protocol which you can use to transfer programs to and from the PicoMite (Tera Term can be downloaded from: <http://tera-term.en.lo4d.com>).

The screen shot on the right shows the setup for Tera Term. Note that the "Port:" setting will vary depending on which USB port your Raspberry Pi Pico was plugged into. The PicoMite ignores the baud rate setting so it can be set to any speed (other than 1200 baud which puts the Pico into firmware upgrade mode).

If you are using Tera Term do not set a delay between characters and if you are using Putty set the backspace key to generate the backspace character.



开始使用

加载PicoMite固件

树莓派Pico配备了易于使用的固件加载程序。只需按照以下步骤操作：

- 从<http://geoffg.net/picomite.html>下载PicoMite固件并解压文件。识别固件，文件名应类似于“Pi coMiteV5.xx.xx.uf2”。
- 使用USB电缆将树莓派Pico连接到您的计算机（Windows、Linux或Mac）同时按住树莓派Pi co上的白色BOOTSEL按钮。
- 树莓派Pico应连接到您的计算机并创建一个虚拟驱动器（就像您插入USB闪存驱动器一样），名为“RPI-RP2”。此驱动器将包含两个文件，您可以忽略它们。
- 将固件文件（扩展名为.uf2）复制到此虚拟驱动器。
- 复制完成后，树莓派Pico将重新启动，并在您的计算机上创建一个虚拟串口。树莓派Pico上的LED将缓慢闪烁，表示运行的PicoMite固件已加载MMBasic。

虽然树莓派Pico创建的虚拟驱动器看起来像一个USB闪存盘，但它并不是，固件文件在复制后将消失，如果您尝试复制任何其他类型的文件，它将被忽略。

加载PicoMite固件将擦除闪存，包括当前程序、保存在闪存插槽中的任何程序以及所有保存的变量。因此，请确保在升级固件之前备份这些数据。

闪存可能会损坏，从而导致异常和不可预测的行为。在这种情况下，您应该加载固件文件https://ge offg.net/Downloads/picomite/Clear_Flash.uf2，这将把树莓派Pico重置为出厂状态，然后您可以重新加载PicoMite固件。

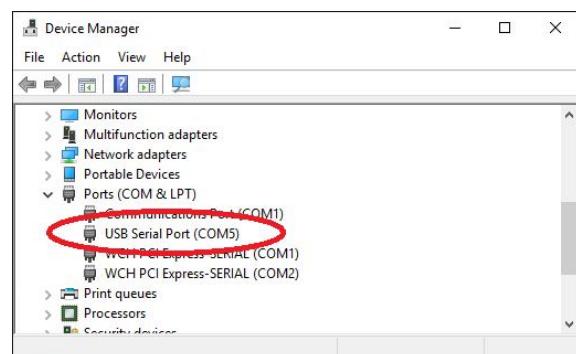
虚拟串口

PicoMite固件创建的虚拟串口

像普通串口一样工作，但它通过USB操作。

Windows 10包含此虚拟串口的驱动程序，但在其他版本中，您可能需要加载驱动程序以使其与操作系统兼容（见下文）。

完成后，您应该注意计算机为虚拟串口连接创建的端口号。在Windows中，可以通过启动设备管理器并检查“端口（COM和LPT）”条目来查看新COM端口，如右侧所示。



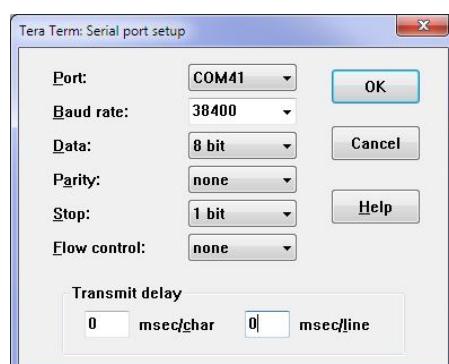
终端仿真器

您还需要在桌面计算机上安装一个终端仿真器程序。该程序像老式计算机终端一样工作，它将显示从远程计算机接收到的文本，任何按键都会通过串行连接发送到远程计算机。您使用的终端仿真器应支持VT100仿真，因为PicoMite内置的编辑器期望这种支持。

对于Windows用户，建议使用Tera Term，因为它有一个良好的VT100仿真器，并且已知可以与XModem协议配合使用，您可以使用该协议将程序传输到PicoMite和从PicoMite传输（Tera Term可以从：<http://tera-term.en.lo4d.com>）下载）。

右侧的屏幕截图显示了Tera Term的设置。请注意，“端口：“设置会根据您的树莓派Pico插入的USB端口而有所不同。PicoMite会忽略波特率设置，因此可以将其设置为任何速度（除了1200波特率，这会将Pico置于固件升级模式）。

如果您使用Tera Term，请不要在字符之间设置延迟；如果您使用Putty，请将退格键设置为生成退格字符。

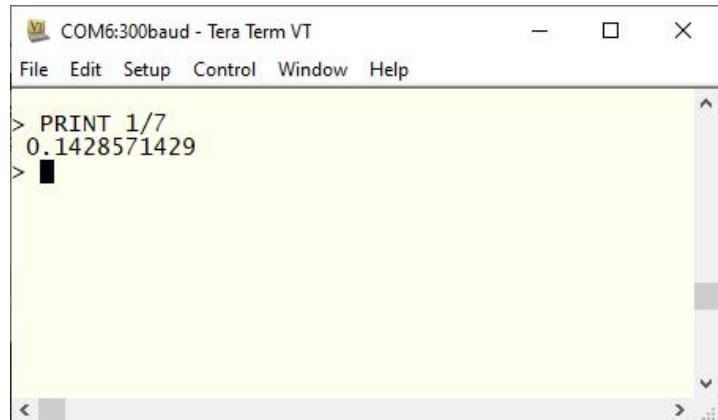


The Console

Once you have identified the virtual serial port and have connected your terminal emulator to it you should be able to press return on your keyboard and see the MMBasic prompt, which is the greater than symbol (e.g. ">").

This is the console and you use it to issue commands to configure the PicoMite, load the BASIC program, edit and run it. MMBasic also uses the console to display error messages.

The console is the only method of communicating with the PicoMite and programming it, so it is important that you can connect to it.



```
File Edit Setup Control Window Help
> PRINT 1/7
0.1428571429
>
```

Some Tests

Here are a few things that you can try out to prove that you have a working PicoMite.

All of these commands should be typed at the command prompt (">"). What you type is shown in bold and the PicoMite's output is shown in normal text.

Try a simple calculation:

```
> PRINT 1/7
```

```
0.1428571429
```

See how much memory you have:

```
> MEMORY
```

Program:

```
OK ( 0% ) Program ( 0 lines )
```

```
80K (100% ) Free
```

RAM:

```
0K ( 0% ) 0 Variables
```

```
0K ( 0% ) General
```

```
112K (100% ) Free
```

What is the current time? Note that the PicoMite's clock starts at midnight on power up.

```
> PRINT TIME$
```

```
00:04:01
```

Set the clock to the current time:

```
> TIME$ = "10:45"
```

Check the time again:

```
> PRINT TIME$
```

```
10:45:09
```

How many milliseconds have elapsed since power up:

```
> PRINT TIMER
```

```
440782.748
```

Count to 20:

```
> FOR a = 1 to 20 : PRINT a; : NEXT a
```

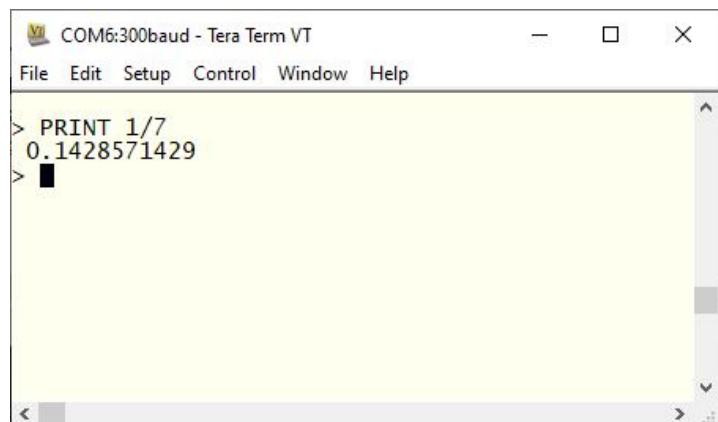
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

控制台

一旦您识别了虚拟串口并将终端仿真器连接到它，您应该能够按下键盘上的回车键，并看到MMBasic提示符，即大于符号（例如“>”）。

这是控制台，你可以使用它来发出命令以配置PicoMite，加载BASIC程序，编辑并运行它。MMBasic还使用控制台显示错误信息。

控制台是与PicoMite进行通信和编程的唯一方法，因此能够连接到它非常重要。



```
File Edit Setup Control Window Help  
> PRINT 1/7  
0.1428571429  
>
```

一些测试

这里有一些你可以尝试的事情，以证明你的PicoMite正常工作。

所有这些命令都应该在命令提示符（">"）下输入。你输入的内容以粗体显示，PicoMite的输出以正常文本显示。

尝试一个简单的计算：

```
> PRINT 1/7
```

```
0.1428571429
```

查看你有多少内存：

```
> MEMORY
```

程序：

```
OK ( 0% ) 程序 ( 0 行 )
```

```
80K ( 100% ) 空闲
```

RAM：

```
OK ( 0% ) 0 变量
```

```
OK ( 0% ) 一般
```

```
112K ( 100% ) 空闲
```

现在几点钟？请注意，PicoMite 的时钟在开机时从午夜开始。

```
> PRINT TIME$
```

```
00:04:01
```

将时钟设置为当前时间：

```
> TIME$ = "10:45"
```

再次检查时间：

```
> PRINT TIME$
```

```
10:45:09
```

自开机以来经过了多少毫秒：

```
> PRINT TIMER
```

```
440782.748
```

数到 20：

```
> FOR a = 1 to 20 : PRINT a; : NEXT a
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Windows 7 and 8.1

The USB serial port uses the CDC protocol and the drivers for this are standard in Windows 10 and 11 and will load automatically.

The Raspberry Pi Foundation lists Windows 7 or 8.1 as “unsupported” however you can use a tool like Zadig (<https://zadig.akeo.ie>) to install a generic driver for a “usbser” device and that should allow these computers to connect. This post describes the process: <https://github.com/raspberrypi/pico-feedback/issues/118>

Apple Macintosh

The Apple Macintosh (OS X) is somewhat easier as it has the device driver and terminal emulator built in. First start the application ‘Terminal’ and at the prompt list the connected serial devices by typing in:

```
ls /dev/tty.*.
```

The USB to serial converter will be listed as something like /dev/tty.usbmodem12345. While still at the Terminal prompt you can run the terminal emulator at 38400 baud by using the command:

```
screen /dev/tty.usbmodem12345 38400
```

By default the function keys will not be correctly defined for use in the PicoMite's built in program editor so you will have to use the control sequences as defined in the section *Full Screen Editor* of this manual. To avoid this you can reconfigure the terminal emulator to generate these codes when the appropriate function keys are pressed.

Documentation for the screen command is here: <https://www.systutorials.com/docs/linux/man/1-screen/>

Linux

For Linux see these posts:

<https://www.thebackshed.com/forum/ViewTopic.php?TID=14157&PID=175474#175474#175466>

and

<https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=16312&LastEntry=Y#213664#213594>

Android

For Android devices see this post:

<https://www.thebackshed.com/forum/ViewTopic.php?TID=16312&PID=213594#213594#213594>

Windows 7 和 8.1

USB 串口使用 CDC 协议，Windows 10 和 11 中的驱动程序是标准的，并会自动加载。

树莓派基金会将Windows 7或8.1列为“未支持”，但是您可以使用像Zadig这样的工具（<https://zadig.akeo.ie>）来安装“usbser”设备的通用驱动程序，这应该允许这些计算机连接。此帖子描述了该过程：<https://github.com/raspberrypi/pico-feedback/issues/118>

苹果 Macintosh

苹果 Macintosh (OS X) 相对容易，因为它内置了设备驱动程序和终端仿真器。

首先启动应用程序‘终端’，然后在提示符下通过输入以下命令列出连接的串行设备：

```
ls /dev/tty.*.
```

USB 到串行转换器将被列为类似于 /dev/tty.usbmodem12345。在终端提示符下，您可以使用以下命令以 38400 波特率运行终端仿真器：

```
screen /dev/tty.usbmodem12345 38400
```

默认情况下，功能键在PicoMite内置程序编辑器中不会被正确定义，因此您需要使用本手册中全屏编辑器部分定义的控制序列。为了避免这种情况，您可以重新配置终端仿真器，以便在按下适当的功能键时生成这些代码。

屏幕命令的文档在这里：<https://www.systutorials.com/docs/linux/man/1-screen/>

Linux

有关Linux，请参见以下帖子：

<https://www.thebackshed.com/forum/ViewTopic.php?TID=14157&PID=175474#175474#175466>

以及

<https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=16312&LastEntry=Y#213664#213594>

Android

有关Android设备，请参见此帖子：

<https://www.thebackshed.com/forum/ViewTopic.php?TID=16312&PID=213594#213594#213594>

Quick Start Tutorial

A Simple Program

To enter a program, you can use the EDIT command which is described later in this manual. However, for the moment, all that you need to know is that anything that you type will be inserted at the cursor, the arrow keys will move the cursor and backspace will delete the character before the cursor.

To get a quick feel for how the PicoMite works, try this sequence (your terminal emulator must be VT100 compatible):

- At the command prompt type EDIT followed by the ENTER key.
- The editor should start up and you can enter this line: PRINT "Hello World"
- Press the F1 key in your terminal emulator (or CTRL-Q which will do the same thing). This tells the editor to save your program and exit to the command prompt.
- At the command prompt type RUN followed by the ENTER key.
- You should see the message: Hello World

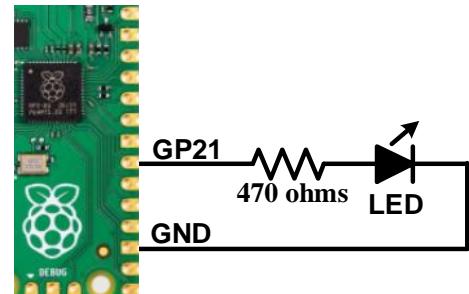
Congratulations. You have just written and run your first program on the PicoMite. If you type EDIT again you will be back in the editor where you can change or add to your program.

Flashing a LED

Connect a LED to pin GP21 (marked on the underside of the board) and a ground pin as shown in the diagram on the right.

Then use the EDIT command to enter the following program:

```
SETPIN GP21, DOUT
DO
    PIN(GP21) = 1
    PAUSE 300
    PIN(GP21) = 0
    PAUSE 300
LOOP
```



When you have saved and run this program you should be greeted by the LED flashing on and off. It is not a great program but it does illustrate how the PicoMite can interface to the physical world via your programming.

The program itself is simple. The first line sets pin GP21 as an output. Then the program enters a continuous loop where the output of that pin is set high to turn on the LED followed by a short pause (300 milliseconds). The output is then set to low followed by another pause. The program then repeats the loop.

If you leave it this way, the PicoMite will sit there forever with the LED flashing. If you want to change something (for example, the speed of flashing) you can interrupt the program by typing CTRL-C on the console and then edit it as needed. This is the great benefit of MMBasic, it is very easy to write and change a program.

If you want this program to automatically start running every time power is applied you can use the command:

```
OPTION AUTORUN ON
```

To test this you can remove the power and then re-apply it. The PicoMite should start up flashing the LED.

Tutorial on Programming in the BASIC Language

If you are new to the BASIC programming language now would be a good time to turn to Appendix G (*Programming in BASIC - A Tutorial*) at the rear of this manual. This is a comprehensive tutorial on the language which will take you through the fundamentals in an easy to read format with lots of examples.

快速入门教程

一个简单的程序

要输入程序，您可以使用本手册后面描述的EDIT命令。然而，目前您需要知道的是，您输入的任何内容都会插入到光标处，箭头键将移动光标，退格键将删除光标前的字符。

要快速了解PicoMite的工作原理，请尝试以下序列（您的终端仿真器必须兼容VT100）：

- 在命令提示符下输入 EDIT，然后按ENTER键。
- 编辑器应该启动，您可以输入这一行： PRINT "Hello World"
- 在你的终端仿真器中按下F1键（或CTRL-Q，这样也可以）。这告诉编辑器保存你的程序并退出到命令提示符。
- 在命令提示符下输入 RUN 然后按下ENTER键。
- 你应该看到消息：你好，世界

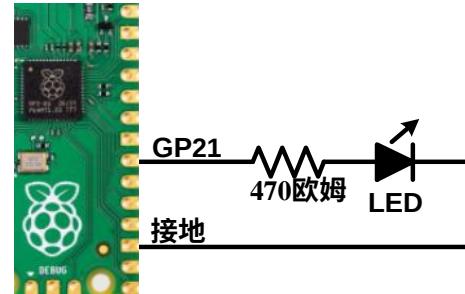
恭喜你。你刚刚在PicoMite上编写并运行了你的第一个程序。如果你再次输入EDIT你将回到编辑器，在那里你可以更改或添加你的程序。

闪烁LED

将LED连接到引脚GP21（在电路板底部标记）以及一个接地引脚，如右侧图示所示。

然后使用EDIT命令输入以下程序：

```
SETPIN GP21, DOUT
DO
    PIN(GP21) = 1
    PAUSE 300
    PIN(GP21) = 0
    PAUSE 300
LOOP
```



当你保存并运行这个程序时，你应该会看到LED闪烁。这不是一个伟大的程序，但它确实展示了PicoMite如何通过你的编程与物理世界进行交互。

程序本身很简单。第一行将引脚 GP21 设置为输出。然后程序进入一个连续循环，在该循环中，该引脚的输出被设置为高电平以点亮 LED，随后短暂暂停（300 毫秒）。

然后输出被设置为低电平，接着又一次暂停。程序随后重复这个循环。

如果你保持这样，PicoMite 将永远保持 LED 闪烁。如果你想更改某些内容（例如，闪烁速度），可以通过在控制台输入 CTRL-C 来中断程序，然后根据需要进行编辑。这就是 MMBasic 的巨大优势，它非常容易编写和更改程序。

如果你希望这个程序在每次通电时自动开始运行，可以使用以下命令：

```
OPTION AUTORUN ON
```

要测试这一点，你可以断开电源，然后重新连接。PicoMite 应该会启动并闪烁 LED。

BASIC语言编程教程

如果你是BASIC编程语言的新手，现在是查看附录G（*BASIC编程教程*）本手册后面的好时机。这是一个全面的语言教程，将以易于阅读的格式带你了解基础知识，并提供大量示例。

PicoMite Hardware

This diagram shows the possible uses within MMBasic for each I/O pin on the Raspberry Pi Pico:

PWM0A	COM1 TX	I2C SDA	SPI RX	GP0	1		VBUS	
PWM0B	COM1 RX	I2C SCL		GP1	2		VSYS	
				GND			GND	
PWM1A		I2C2 SDA	SPI CLK	GP2	4		3V3EN	
PWM1B		I2C2 SCL	SPI TX	GP3	5		3V3	
PWM2A	COM2 TX	I2C SDA	SPI RX	GP4	6		ADC VREF	
PWM2B	COM2 RX	I2C SCL		GP5	7		ADC2	
				GND		34 GP28 SPI2 RX	I2C SDA COM1 TX PWM6A	
PWM3A		I2C2 SDA	SPI CLK	GP6	9		AGND	
PWM3B		I2C2 SCL	SPI TX	GP7	10		ADC1	
PWM4A	COM2 TX	I2C SDA	SPI2 RX	GP8	11		32 GP27 SPI2 TX	I2C2 SCL PWM5B
PWM4B	COM2 RX	I2C SCL		GP9	12		ADC0	31 GP26 SPI2 CLK I2C2 SDA PWM5A
				GND			RUN	
PWM5A		I2C2 SDA	SPI2 CLK	GP10	14		29 GP22	I2C2 SDA PWM3A
PWM5B		I2C2 SCL	SPI2 TX	GP11	15		GND	
PWM6A	COM1 TX	I2C SDA	SPI2 RX	GP12	16		27 GP21	I2C SCL COM2 RX PWM2B
PWM6B	COM1 RX	I2C SCL		GP13	17		26 GP20 SPI RX	I2C SDA COM2 TX PWM2A
				GND			25 GP19 SPI TX	I2C2 SCL PWM1B
PWM7A		I2C2 SDA	SPI2 CLK	GP14	19		24 GP18 SPI CLK	I2C2 SDA PWM1A
PWM7B		I2C2 SCL	SPI2 TX	GP15	20		GND	
							22 GP17	I2C SCL COM1 RX PWM0B
							21 GP16 SPI RX	I2C SDA COM1 TX PWM0A

The notation is as follows:

- GP0 to GP28 Can be used for digital input or output.
- COM1, COM2 Can be used for asynchronous serial I/O (UART0 and UART1 pins on the Pico datasheet).
- I2C, I2C2 Can be used for I²C communications (I2C0 and I2C1 pins on the Pico datasheet).
- SPI, SPI2 Can be used for SPI I/O (see Appendix D). (SPI0 and SPI1 pins on the Pico datasheet).
- PWMnx Can be used for PWM output (see the PWM command).
- GND Common ground.
- VBUS 5V supply directly from the USB port.
- VSYS 5V supply used by the SMPS to provide 3.3V. This can be used as a 5V output or input.
- 3V3EN Enable 3.3V regulator (low = off, high = enabled).
- RUN Reset pin, low will hold the PicoMite in reset.
- ADCn These pins can be used to measure voltage (analog input).
- ADC VREF Reference voltage for voltage measurement.
- AGND Analog ground.

All pins can be used for digital input or output however they are limited to a maximum voltage of 3.6V. This means that level shifting will be required if they are used with devices operating at 5V or higher.

Within the MMBasic program I/O pins can be referred to using the physical pin number (i.e. 1 to 40) or the GP number (i.e. GP0 to GP28). For example, the following refer to the same pin and operate identically:

SETPIN 32, DOUT

and

SETPIN GP27, DOUT

On the PicoMite on-chip functions such as the SPI and I²C interfaces are not allocated to fixed pins, unlike (for example) the Micromite. The PicoMite makes extensive use of the SETPIN command, not only to configure I/O pins but also to configure the pins used for interfaces such as serial, SPI, I²C, etc.

Pins must be allocated according to this drawing. For example, the SPI TX can be allocated to pins GP3, GP7 or GP19 but it cannot be allocated to pin GP11 which can only be allocated to the SPI2 channel. Allocations don't have to be in the same "block" so you could, for example, allocate SPI2 TX to pin GP11 and SPI2 RX to pin GP28.

PicoMite硬件

该图显示了树莓派Pico上每个I/O引脚在MMBasic中的可能用途：

PWM0A	COM1 TX	I2C SDA	SPI RX	GP0	1		VBUS
PWM0B	COM1 RX	I2C SCL		GP1	2		VSYS
				接地			接地
PWM1A		.2C2 SDA	SPI CLK	GP2	4		3V3EN
PWM1B		.2C2 SCL	SPI TX	GP3	5		3V3
PWM2A	COM2 TX	I2C SDA	SPI RX	GP4	6		参考电压
PWM2B	COM2 RX	I2C SCL		GP5	7		ADC VREF
				接地			接地
PWM3A		.2C2 SDA	SPI CLK	GP6	9		ADC2
PWM3B		.2C2 SCL	SPI TX	GP7	10		ADC1
PWM4A	COM2 TX	I2C SDA	SPI2 RX	GP8	11		ADC0
PWM4B	COM2 RX	I2C SCL		GP9	12		运行
				接地			29 GP22 .2C2 数据线
PWM5A		.2C2 SDA	SPI2 CLK	GP10	14		接地
PWM5B		.2C2 SCL	SPI2 TX	GP11	15		27 GP21 .2C 时钟线
PWM6A	COM1 TX	I2C SDA	SPI2 RX	GP12	16		26 GP20 SPI 接收
PWM6B	COM1 RX	I2C SCL		GP13	17		25 GP19 SPI 发送
				接地			24 GP18 SPI 时钟
PWM7A		.2C2 SDA	SPI2 CLK	GP14	19		22 GP17 .2C 时钟线
PWM7B		.2C2 SCL	SPI2 TX	GP15	20		21 GP16 SPI 接收
				接地			2C 数据线 COM1 接收 PWM0B
				接地			2C 数据线 COM1 发送 PWM0A

符号表示如下：

GP0 到 GP28 可以用作数字输入或输出。

COM1, COM2 可以用于异步串行I/O（树莓派数据表上的UART0和UART1引脚）。

I2C, I2C2 可以用于I²C通信（树莓派数据表上的I2C0和I2C1引脚）。

SPI, SPI2 可以用于SPI I/O（见附录D）。（树莓派数据表上的SPI0和SPI1引脚）。

PWMnx 可以用于PWM输出（见PWM命令）。

GND 公共接地。

VBUS 直接来自USB端口的5V电源。

VSYS 用于SMPS提供3.3V的5V电源。可以用作5V输出或输入。

3V3EN 启用3.3V稳压器（低 = 关闭，高 = 启用）。

运行 重置引脚，低电平将使PicoMite保持在重置状态。

ADCn 这些引脚可用于测量电压（模拟输入）。

ADC VREF 电压测量的参考电压。

AGND 模拟接地。

所有引脚都可以用作数字输入或输出，但它们的最大电压限制为3.6V。这意味着如果与工作在5V或更高电压的设备一起使用，则需要进行电平转换。

在MMBasic程序中，I/O引脚可以使用物理引脚编号（即1到40）或GP编号（即GP0到GP28）进行引用。例如，以下两者引用相同的引脚并且操作相同：

SETPIN 32, DOUT

和

SETPIN GP27, DOUT

在PicoMite上，诸如SPI和I2C接口等片上函数并未分配给固定引脚，这与（例如）Micromite不同。PicoMite广泛使用SETPIN命令，不仅用于配置I/O引脚，还用于配置用于串行、SPI、I²C等接口的引脚。

引脚必须根据此图纸进行分配。例如，SPI TX可以分配给引脚GP3、GP7或GP19，但不能分配给引脚GP11，因为它只能分配给SPI2通道。分配不必在同一个“块”中，因此您可以例如将SPI2TX分配给引脚GP11，将SPI2 RX分配给引脚GP28。

Pins that are not exposed on the Raspberry Pi Pico can still be accessed using MMBasic via a pseudo pin number or their GPn number. This allows MMBasic to be used on other modules that use the RP2040 processor. These hidden pins are Pin 41 or GP23, Pin 42 or GP24, Pin 43 or GP25 and Pin 44 or GP29.

On the Raspberry Pi Pico these pins are used for internal functions as follows:

- Pin 41 or GP23 is a digital output set to the value of OPTION POWER. (ON=PWM, OFF=PFM).
- Pin 42 or GP24 is a digital input, which is high when VBUS is present.
- Pin 43 or GP25 is also PWM4B. It is an output connected to the on-board LED.
- Pin 44 or GP29 is also ADC3 which is an analog input reading $\frac{1}{3}$ of VSYS.

I/O Pin Limits

The maximum voltage that can be applied to any I/O pin is 3.6V.

As outputs all I/O pins can individually source or sink a maximum of 12mA. At this load the output voltage will sag to about 2.3V. A more practical load is 5mA where the output voltage would typically be 3V. To drive a red LED at 5mA the recommended resistor is 220Ω . Other colours may require a different value.

The maximum total I/O current load for the entire chip is 50mA.

Power Supply

The Raspberry Pi Pico has a flexible power system.

The input voltage from either the USB or VBUS inputs is connected through a Schottky diode to the buck-boost SMPS (Switch Mode Power Supply) which has an output of 3.3V. The SMPS will accommodate input voltages from 1.8V to 5.5V allowing the PicoMite to run from a wide range of power sources including batteries.

External circuitry can be powered by VBUS (normally 5V) or by the 3V3 (3.3V) output which can source up to 300mA.

For embedded controller applications generally an external power source (other than USB) is required and this can be connected to VSYS via a Schottky diode. This will allow the PicoMite to be powered by whichever supply is producing the highest voltage (USB or VSYS). The diodes will prevent feedback into the lower voltage supply.

To minimize power supply noise it is possible to ground 3V3EN to turn off the SMPS. When shutdown the converter will stop switching, internal control circuitry will be turned off and the load disconnected. You can then power the board via a 3.3V linear regulator feeding into the 3V3 pin.

Clock Speed

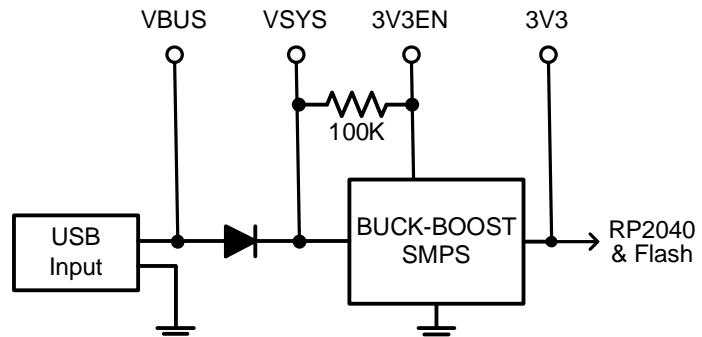
By default the clock speed for the PicoMite is 133MHz which is the recommended maximum for the Raspberry Pi Pico. However, by using the OPTION CPUSPEED command, the CPU can be overclocked up to 378MHz or run slower to a minimum of 48MHz. Nearly all tested Raspberry Pi Picos have worked correctly at 378MHz so overclocking can be useful. If the processor fails to restart at its new clock speed you can reset it by loading this firmware file onto the Pico: https://geoffg.net/Downloads/picomite/Clear_Flash.uf2. The procedure to do this is same as loading any other firmware.

This option is saved and will be reapplied on power up. When changing the clock speed the PicoMite will be reset then rebooted so the USB connection will be disconnected.

Power Consumption

The power consumption is dependent on the clock speed. These are typical readings for the PicoMite and do not include any current sourced or sunk by the I/O pins or the 3V3 pin:

250MHz	43mA
133MHz	21mA
48MHz	10mA.



未在树莓派 Pico 上暴露的引脚仍然可以通过伪引脚编号或其GPn编号使用MMBasic进行访问。这使得MMBasic可以在使用RP2040处理器的其他模块上使用。这些隐藏引脚是引脚41或GP23，引脚42或GP24，引脚43或GP25，以及引脚44或GP29。在树莓派 Pico 上，这些引脚用于内部功能如下：

- 引脚41或GP23是数字输出，设置为OPTION POWER的值。（开启=PWM，关闭=PFM）。
- 引脚42或GP24是一个数字输入，当VBUS存在时为高电平。
- 引脚43或GP25也是PWM4B。它是一个连接到板载LED的输出。
- 引脚44或GP29也是ADC3，它是一个读取 $\frac{1}{3}$ VSYS的模拟输入。

I/O引脚限制

可以施加到任何I/O引脚上的最大电压为3.6V。

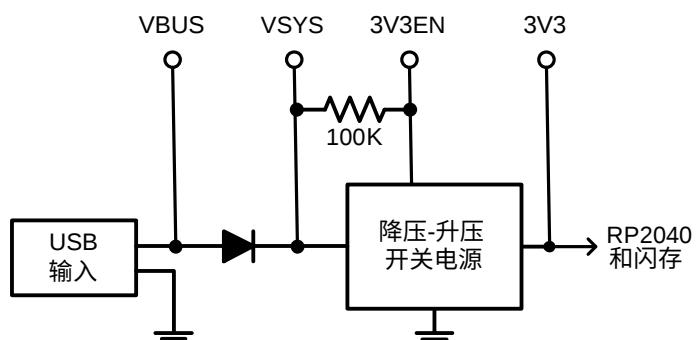
作为输出，所有I/O引脚可以单独提供或吸收最大12mA的电流。在此负载下，输出电压将下降到约2.3V。更实用的负载是5mA，此时输出电压通常为3V。为了以5mA驱动红色LED，推荐的电阻为220Ω。其他颜色可能需要不同的值。

整个芯片的最大总I/O电流负载为50mA。

电源供应

树莓派Pico具有灵活的电源系统。

来自USB或VBUS输入的输入电压通过肖特基二极管连接到具有3.3V输出的升降压开关电源(SMPs)。该SMPs可以适应1.8V到5.5V的输入电压，使PicoMite能够从包括电池在内的广泛电源中运行。



外部电路可以通过VBUS（通常为5V）或3V3（3.3V）输出供电，后者可以提供高达300mA的电流。

对于嵌入式控制器应用，通常需要外部电源（USB以外），可以通过肖特基二极管连接到VSYS。这将允许PicoMite由产生最高电压的电源供电（USB或VSYS）。二极管将防止低电压电源的反馈。

为了最小化电源噪声，可以将3V3EN接地以关闭开关电源。当关闭时，转换器将停止切换，内部控制电路将被关闭，负载将断开。然后可以通过一个3.3V线性稳压器为板子供电，连接到3V3引脚。

时钟速度

默认情况下，PicoMite的时钟速度为133MHz，这是树莓派Pico推荐的最大值。然而，通过使用OPTION CPUSPEED命令，CPU可以超频到378MHz，或降低到最低48MHz。几乎所有测试过的树莓派Pico在378MHz下都能正常工作，因此超频是有用的。如果处理器无法以新的时钟速度重新启动，可以通过将此固件文件加载到Pico上来重置它：https://geoffg.net/Downloads/picomite/Clear_Flash.uf2。执行此操作的过程与加载任何其他固件相同。

此选项会被保存，并将在上电时重新应用。当更改时钟速度时，PicoMite 将重置并重新启动，因此USB连接将被断开。

功耗

功耗取决于时钟速度。这些是 PicoMite 的典型读数，不包括任何由 I/O 引脚或 3V3 引脚提供或消耗的电流：

250MHz	43mA
133MHz	21mA
48MHz	10mA。

Using MMBasic

Commands and Program Input

At the command prompt you can enter a command and it will be immediately run. Most of the time you will do this to tell the PicoMite to do something like run a program or set an option. But this feature also allows you to test out commands at the command prompt.

To enter a program the easiest method is to use the EDIT command. This will invoke the full screen program editor which is built into the PicoMite and is described later in this manual. It includes advanced features such as search and copy, cut and paste to and from a clipboard.

You could also compose the program on your desktop computer using something like Notepad and then transfer it to the PicoMite via the XModem protocol (see the XMODEM command) or by streaming it up the console serial link (see the AUTOSAVE command).

A third and convenient method of writing and debugging a program is to use MMEdit. This is a program running on your Windows computer which allows you to edit your program on your computer then transfer it to the PicoMite with a single click of the mouse. MMEdit was written by Jim Hiley and can be downloaded for free from <https://www.c-com.com.au/MMedit.htm>.

One thing that you cannot do is use the old BASIC way of entering a program which was to prefix each line with a line number. Line numbers are optional in MMBasic so you can still use them if you wish but if you enter a line with a line number at the prompt MMBasic will simply execute it immediately.

Program Structure

A BASIC program starts at the first line and continues until it runs off the end of the program or hits an END command - at which point MMBasic will display the command prompt (>) on the console and wait for something to be entered.

A program consists of a number of statements or commands, each of which will cause the BASIC interpreter to do something (the words statement and command generally mean the same and are used interchangeably). Normally each statement is on its own line but you can have multiple statements in the one line separated by the colon character (:). For example.

```
A = 24.6 : PRINT A
```

Each line can start with a line number. Line numbers were mandatory in the early BASIC interpreters however modern implementations (such as MMBasic) do not need them. You can still use them if you wish but they have no benefit and generally just clutter up your programs. This is an example of a program that uses line numbers:

```
50 A = 24.6
60 PRINT A
```

A line can also start with a label which can be used as the target for a program jump using the GOTO command. For example (the label name is JmpBack):

```
JmpBack: A = A + 1
PRINT A
GOTO JmpBack
```

A label has the same specifications (length, character set, etc) as a variable name but it cannot be the same as a command name. When used to label a line the label must appear at the beginning of a line but after a line number (if used) and be terminated with a colon character (:).

Editing the Command Line

When entering a line at the command prompt the line can be edited using the left and right arrow keys to move along the line, the Delete key to delete a character and the Insert key to switch between insert and overwrite. At any point the Enter key will send the line to MMBasic which will execute it.

The up and down arrow keys will move through a history of previously entered command lines which can be edited and reused.

使用MMBasic

命令和程序输入

在命令提示符下，您可以输入一个命令，它将立即运行。大多数情况下，您这样做是为了告诉Pico Mite执行某个操作，例如运行一个程序或设置一个选项。但这个功能也允许您在命令提示符下测试命令。

输入程序的最简单方法是使用EDIT命令。这将调用内置于PicoMite中的全屏程序编辑器，后面会在本手册中描述。它包括高级功能，如搜索和复制、剪切以及从剪贴板粘贴。

您也可以在桌面计算机上使用类似记事本的程序编写程序，然后通过XModem协议（请参见X MODEM命令）或通过控制台串行链接（请参见AUTOSAVE命令）将其传输到PicoMite。

第三种方便的编写和调试程序的方法是使用MMEdit。这是一个在您的Windows计算机上运行的程序，允许您在计算机上编辑程序，然后通过单击鼠标将其传输到PicoMite。 MMEdit是由吉姆·海利（Jim Hiley）编写的，可以从<https://www.c-com.com.au/MMedit.htm>免费下载。

你不能做的一件事是使用旧的BASIC方式输入程序，即在每行前加上行号。在MMBasic中，行号是可选的，因此如果你愿意仍然可以使用它们，但如果你在命令提示符下输入带有行号的行，MMBasic将立即执行它。

程序结构

一个BASIC程序从第一行开始，直到程序结束或遇到END命令为止——此时MMBasic将在控制台上显示命令提示符（>）并等待输入。

一个程序由多个语句或命令组成，每个语句或命令都会使BASIC解释器执行某些操作（语句和命令通常意味着相同的内容，并可以互换使用）。

通常每个语句占据一行，但你可以在同一行中使用冒号字符（:）分隔多个语句。例如。

```
A = 24.6 : 打印 A
```

每行可以以行号开头。在早期的BASIC解释器中，行号是必需的，但现代实现（如MMBasic）不需要它们。如果你愿意，仍然可以使用它们，但它们没有好处，通常只会使你的程序变得杂乱。这是一个使用行号的程序示例：

```
50 A = 24.6
60 打印 A
```

一行也可以以标签开头，该标签可以用作使用GOTO命令进行程序跳转的目标。例如（标签名称是 JmpBack）：

```
JmpBack: A = A + 1
打印 A
转到 JmpBack
```

标签具有与变量名称相同的规范（长度、字符集等），但不能与命令名称相同。当用于标记一行时，标签必须出现在行的开头，但在行号（如果使用）之后，并以冒号字符（:）结束。

编辑命令行

在命令提示符下输入一行时，可以使用左箭头和右箭头键在行中移动，使用删除键删除字符，使用插入键在插入和覆盖模式之间切换。

在任何时候，按下回车键将把该行发送给MMBasic进行执行。

上下箭头键将浏览之前输入的命令行历史，可以进行编辑和重用。

Shortcut Keys

The function keys on the keyboard or the serial console can be used at the command prompt to automatically enter common commands. These function keys will insert the text followed by the Enter key so that the command is immediately executed:

F2	RUN
F3	LIST
F4	EDIT
F10	AUTOSAVE
F11	XMODEM RECEIVE
F12	XMODEM SEND

Function keys F1, and F5 to F9 can be programmed with custom text. See the OPTION FNKey command.

Interrupting A Running Program

A program is set running by the RUN command. You can interrupt MMBasic and the running program at any time by typing CTRL-C on the console input and MMBasic will return to the command prompt.

Setting Options

Many options can be set by using commands that start with the keyword OPTION. They are listed in their own section of this manual. For example, you can change the CPU clock speed with the command:

```
OPTION CPUSPEED speed
```

Saved Variables

Because the PicoMite does not necessarily have a normal storage system it needs to save data that can be recovered when power is restored. This can be done with the VAR SAVE command which will save the variables listed on its command line in non-volatile flash memory. The space reserved for saved variables is 16KB.

These variables can be restored with the VAR RESTORE command which will add all the saved variables to the variable table of the running program. Normally this command is placed near the start of a program so that the variables are ready for use by the program.

This facility is intended for saving calibration data, user selected options and other items which change infrequently. It should not be used for high-speed saves as you may wear out the flash memory. The flash used for the Raspberry Pi Pico has a high endurance but this can be exceeded by a program that repeatedly saves variables. If you do want to save data often you should add a real time clock chip. The RTC commands can then be used to store and retrieve data from the RTC's battery backed memory. See the RTC command for more details.

Watchdog Timer

The main use for the PicoMite is as an embedded controller. It can be programmed in MMBasic and when the program is debugged and ready for "prime time" the OPTION AUTORUN configuration setting can be turned on. The module will then automatically run its program when power is applied and act as a custom circuit performing some special task. The user need not know anything about what is running inside it.

However, there is the possibility that a fault in the program could cause MMBasic to generate an error and return to the command prompt. This would be of little use in an embedded situation as the PicoMite would not have anything connected to the console. Another possibility is that the BASIC program could get itself stuck in an endless loop for some reason. In both cases the visible effect would be the same... the program would stop running until the power was cycled.

To guard against this the watchdog timer can be used. This is a timer that counts down to zero and when it reaches zero the processor will be automatically restarted (the same as when power was first applied), this will occur even if MMBasic was sitting at the command prompt. Following the restart the automatic variable MM.WATCHDOG will be set to true to indicate that the restart was caused by a watchdog timeout.

The WATCHDOG command should be placed in strategic locations in the program to keep resetting the timer and therefore preventing it from counting down to zero. Then, if a fault occurs, the timer will not be reset, it will count down to zero and the program will be restarted (assuming the AUTORUN option is set).

快捷键

键盘上的功能键或串行控制台可以在命令提示符下用于自动输入常用命令。这些功能键将插入文本并随后按回车键，以便立即执行命令：

F2	运行
F3	列表
F4	编辑
F10	自动保存
F11	XMODEM 接收
F12	XMODEM 发送

功能键F1，以及F5到F9可以编程为自定义文本。查看OPTION FNKey命令。

中断正在运行的程序

通过RUN命令启动程序。您可以随时通过在控制台输入CTRL-C来中断MMBasic和正在运行的程序，MMBasic将返回到命令提示符。

设置选项

许多选项可以通过以关键字OPTION开头的命令进行设置。它们在本手册的专门部分中列出。例如，您可以使用以下命令更改CPU时钟速度：

```
OPTION CPUSPEED speed
```

保存的变量

由于PicoMite不一定具有正常的存储系统，因此需要保存可以在恢复电源时恢复的数据。这可以通过VAR SAVE命令完成，该命令将把命令行中列出的变量保存在非易失性闪存中。为保存的变量保留的空间为16KB。

这些变量可以通过VAR RESTORE命令恢复，该命令将把所有保存的变量添加到正在运行的程序的变量表中。通常，这个命令放置在程序的开头附近，以便变量可以被程序使用。

此功能旨在保存校准数据、用户选择的选项和其他不常更改的项目。它不应被用于高速保存，因为这可能会损坏闪存。用于树莓派 Pico 的闪存具有高耐用性，但如果程序反复保存变量，这种耐用性可能会被超越。如果您确实想要频繁保存数据，您应该添加一个实时钟芯片。然后可以使用 RTC 命令从 RTC 的电池备份内存中存储和检索数据。有关更多详细信息，请参见RTC命令。

看门狗定时器

PicoMite的主要用途是作为嵌入式控制器。它可以用MMBasic 编程，当程序调试完成并准备好进入“正式使用”时，可以开启自动运行选项配置设置。然后模块将在通电时自动运行其程序，并作为执行某些特殊任务的自定义电路。用户无需了解内部运行的任何内容。然而，程序中的故障可能会导致MMBasic生成错误并返回到命令提示符。在嵌入式情况下，这几乎没有用，因为PicoMite不会连接到控制台。另一种可能性是BASIC程序可能由于某种原因陷入无限循环。在这两种情况下，明显的效果都是一样的……程序将停止运行，直到电源被重启。

为了防止这种情况，可以使用看门狗定时器。这是一种倒计时到零的定时器，当它达到零时，处理器将自动重启（与首次通电时相同），即使MMBasic在命令提示符处也会发生这种情况。重启后，自动变量MM.WATCHDOG将被设置为true，以指示重启是由于看门狗超时引起的。

WATCHDOG命令应放置在程序的关键位置，以保持重置定时器，从而防止其倒计时至零。然后，如果发生故障，定时器将不会被重置，它将倒计时至零，程序将重新启动（假设已设置AUTORUN选项）。

PIN Security

Sometimes it is important to keep the data and program in an embedded controller confidential. In the PicoMite this can be done by using the OPTION PIN command. This command will set a pin number (which is stored in flash) and whenever the PicoMite returns to the command prompt (for whatever reason) the user at the console will be prompted to enter the PIN number. Without the correct PIN the user cannot get to the command prompt and their only option is to enter the correct PIN or reboot the PicoMite. When it is rebooted the user will still need the correct PIN to access the command prompt.

Because an intruder cannot reach the command prompt they cannot list or copy a program, they cannot change the program or change any aspect of MMBasic or the PicoMite. Once set the PIN can only be removed by providing the correct PIN as set in the first place. If the number is lost the only method of recovery is to reload the PicoMite firmware (which will erase the program and all options).

There are other time consuming ways of accessing the data (such as using a programmer to examine the flash memory) so this should not be regarded as the ultimate security but it does act as a significant deterrent.

The Library

Using the LIBRARY feature it is possible to create BASIC functions, subroutines and embedded fonts and add them to MMBasic to make them permanent and part of the language. For example, you might have written a series of subroutines and functions that perform sophisticated bit manipulation; these could be stored as a library and become part of MMBasic and perform the same as other built-in functions that are already part of the language. An embedded font can also be added the same way and used just like a normal font.

To install components into the library you need to write and test the routines as you would with any normal BASIC routines. When they are working correctly you can use the LIBRARY SAVE command. This will transfer the routines (as many as you like) to a non-visible part of flash memory where they will be available to any BASIC program but will not show when the LIST command is used and will not be deleted when a new program is loaded or NEW is used. However, the saved subroutines and functions can be called from within the main program and can even be run at the command prompt (just like a built-in command or function).

Some points to note:

- Library routines act exactly like normal BASIC code and can consist of any number of subroutines, functions, embedded C routines and fonts. The only difference is that they do not show when a program is listed and are not deleted when a new program is loaded.
- Library routines can create and access global variables and are subject to the same rules as the main program – for example, respecting OPTION EXPLICIT if it is set.
- When the routines are transferred to the library MMBasic will compress them by removing comments, extra spaces, blank lines and the hex codes in embedded C routines and fonts. This makes the library space efficient, especially when loading large fonts. Following the save the program area is cleared.
- You can use the LIBRARY SAVE command multiple times. With each save the new contents of the program space are appended to the already existing code in the library.
- You can use line numbers in the library but you cannot use a line number on an otherwise empty line as the target for a GOTO, etc. This is because the LIBRARY SAVE command will remove any blank lines.
- You can use READ commands in the library but they will default to reading DATA statements in the main program memory. If you want to read from DATA statements in the library you must use the RESTORE command before the first READ command. This will reset the pointer to the library space.
- The library is saved to program flash memory Slot 3 and this will not be available for storing a program if LIBRARY SAVE is used.
- You can see what is in the library by using the LIBRARY LIST command which will list the contents of the library space.
- The LIBRARY contents can be saved to disk using LIBRARY DISK SAVE fname\$ and restored using LIBRARY DISK LOAD fname\$

To delete the routines in the library space you use the LIBRARY DELETE command. This will clear the space and return the Flash Slot 3 used by the library back to being available for storage for normal programs. The only other way to delete a library is to use OPTION RESET.

引脚安全

有时，保持嵌入式控制器中的数据和程序机密是很重要的。在PicoMite中，可以通过使用OPTION PIN命令来实现。该命令将设置一个引脚号码（存储在闪存中），每当PicoMite返回到命令提示符时（无论出于何种原因），控制台上的用户将被提示输入引脚号码。没有正确的引脚，用户无法进入命令提示符，他们唯一的选择是输入正确的引脚或重启PicoMite。重启后，用户仍然需要正确的引脚才能访问命令提示符。

因为入侵者无法到达命令提示符，他们无法列出或复制程序，无法更改程序或更改MMBasic或PicoMite的任何方面。一旦设置，引脚只能通过提供最初设置的正确引脚来移除。如果引脚丢失，恢复的唯一方法是重新加载PicoMite固件（这将擦除程序和所有选项）。

还有其他耗时的方法可以访问数据（例如使用程序检查闪存），因此这不应被视为最终的安全措施，但它确实起到了显著的威慑作用。

库

使用LIBRARY功能，可以创建BASIC函数、子程序和嵌入式字体，并将它们添加到MMBasic中，使其成为永久性的一部分。例如，您可能已经编写了一系列执行复杂位操作的子程序和函数；这些可以存储为库，并成为MMBasic的一部分，执行与语言中其他内置函数相同的功能。可以以相同的方式添加嵌入式字体，并像普通字体一样使用。

要将组件安装到库中，您需要像处理任何普通BASIC例程一样编写和测试例程。当它们正常工作时，您可以使用LIBRARY SAVE命令。这将把例程（任意数量）转移到闪存的一个不可见部分，在那里它们将对任何BASIC程序可用，但在使用LIST命令时不会显示，并且在加载新程序或使用NEW时不会被删除。然而，保存的子程序和函数可以在主程序中调用，甚至可以在命令提示符下运行（就像内置命令或函数一样）。

需要注意几点：

- 库例程的行为与普通BASIC代码完全相同，可以包含任意数量的子程序、函数、嵌入式C例程和字体。唯一的区别是它们在程序列出时不会显示，并且在加载新程序时不会被删除。
- 库例程可以创建和访问全局变量，并遵循与主程序相同的规则——例如，如果设置了选项显式，则需要遵守。
- 当例程被转移到库时，MMBasic会通过删除注释、额外空格、空行以及嵌入式C例程和字体中的十六进制代码来压缩它们。这使得库空间高效，特别是在加载大型字体时。保存后，程序区域将被清空。
- 您可以多次使用保存库命令。每次保存时，程序空间的新内容将附加到库中已存在的代码上。
- 您可以在库中使用行号，但不能在其他空行上使用行号作为转到的目标等。这是因为保存库命令会删除任何空行。
- 您可以在库中使用读取命令，但它们将默认读取主程序内存中的数据语句。如果您想从库中的数据语句读取，必须在第一个读取命令之前使用恢复命令。这将重置指针到库空间。
- 库被保存到程序闪存内存槽3，如果使用LIBRARY SAVE，这将无法用于存储程序。
- 您可以使用LIBRARY LIST命令查看库中的内容，该命令将列出库空间的内容。
- 库的内容可以使用LIBRARY DISK SAVE fname\$保存到磁盘，并使用LIBRARY DISK LOAD fname\$恢复。

要删除库空间中的例程，您可以使用LIBRARY DELETE命令。这将清除空间并将库使用的闪存槽3返回为可用于存储正常程序。删除库的唯一其他方法是使用OPTION RESET。

Program Initialisation

The library can also include code that is not contained within a subroutine or function. This code (if it exists) will be run automatically before a program starts running (ie, via the RUN command). This feature can be used to initialise constants or setup MMBasic in some way. For example, if you wanted to set some constants you could include the following lines in the library code:

```
CONST TRUE = 1  
CONST FALSE = 0
```

For all intents and purposes, the identifiers TRUE and FALSE have been added to the language and will be available to any program that is run on the PicoMite.

MM.STARTUP

There may be a need to execute some code on initial power up, perhaps to initialise some hardware, set some options or print a custom start-up banner. This can be accomplished by creating a subroutine with the name MM.STARTUP. When the PicoMite is first powered up or reset it will search for this subroutine and, if found, it will be run once.

For example, if the PicoMite has a real time clock attached, the program could contain the following code:

```
SUB MM.STARTUP  
    RTC GETTIME  
END SUB
```

This would cause the internal clock within MMBasic to be set to the current time on every power up or reset.

After the code in MM.STARTUP has been run MMBasic will continue with running the rest of the program in program memory. If there is no other code MMBasic will return to the command prompt.

Note that you should not use MM.STARTUP for general setup of MMBasic (like dimensioning arrays, opening communication channels, etc) before running a program. The reason is that when you use the RUN command MMBasic will clear the interpreter's state ready for a fresh start.

MM.PROMPT

If a subroutine with this name exists it will be automatically executed by MMBasic instead of displaying the command prompt. This can be used to display a custom prompt, set colours, define variables, etc all of which will be active at the command prompt.

Note that MMBasic will clear all variables and I/O pin settings when a program is run so anything set in this subroutine will only be valid for commands typed at the command prompt (i.e. in immediate mode).

As an example the following will display a custom prompt:

```
SUB MM.PROMPT  
    PRINT TIME$ "> " ;  
END SUB
```

Note that while constants can be defined, they will not be visible because a constant defined inside a subroutine is local to a subroutine. However, DIM will create variables that are global that should be used instead.

程序初始化

库还可以包含不在子程序或函数中的代码。这段代码（如果存在）将在程序开始运行之前自动运行（即，通过RUN命令）。此功能可用于初始化常量或以某种方式设置MMBasic。例如，如果您想设置一些常量，可以在库代码中包含以下行：

```
CONST TRUE = 1  
CONST FALSE = 0
```

就所有意图和目的而言，标识符TRUE和FALSE已被添加到语言中，并将对在PicoMite上运行的任何程序可用。

MM.STARTUP

在初始通电时，可能需要执行一些代码，例如初始化某些硬件、设置一些选项或打印自定义启动横幅。这可以通过创建一个名为MM.STARTUP的子程序来实现。当PicoMite首次通电或重置时，它将搜索此子程序，如果找到，将运行一次。

例如，如果PicoMite连接了一个实时钟，程序可以包含以下代码：

```
SUB MM.STARTUP  
    RTC GETTIME  
END SUB
```

这将导致MMBasic内部时钟在每次开机或重置时设置为当前时间。

在MM.STARTUP中的代码运行后，MMBasic将继续运行程序内存中的其余程序。如果没有其他代码，MMBasic将返回到命令提示符。

请注意，在运行程序之前，您不应使用MM.STARTUP进行MMBasic的一般设置（如定义数组、打开通信通道等）。原因是，当您使用RUN命令时，MMBasic将清除解释器的状态，以准备进行全新的开始。

MM.PROMPT

如果存在具有此名称的子程序，MMBasic将自动执行该子程序，而不是显示命令提示符。这可以用于显示自定义提示、设置颜色、定义变量等，所有这些将在命令提示符下生效。

请注意，当运行程序时，MMBasic将清除所有变量和I/O引脚设置，因此在此子程序中设置的任何内容仅对在命令提示符下输入的命令有效（即在立即模式下）。

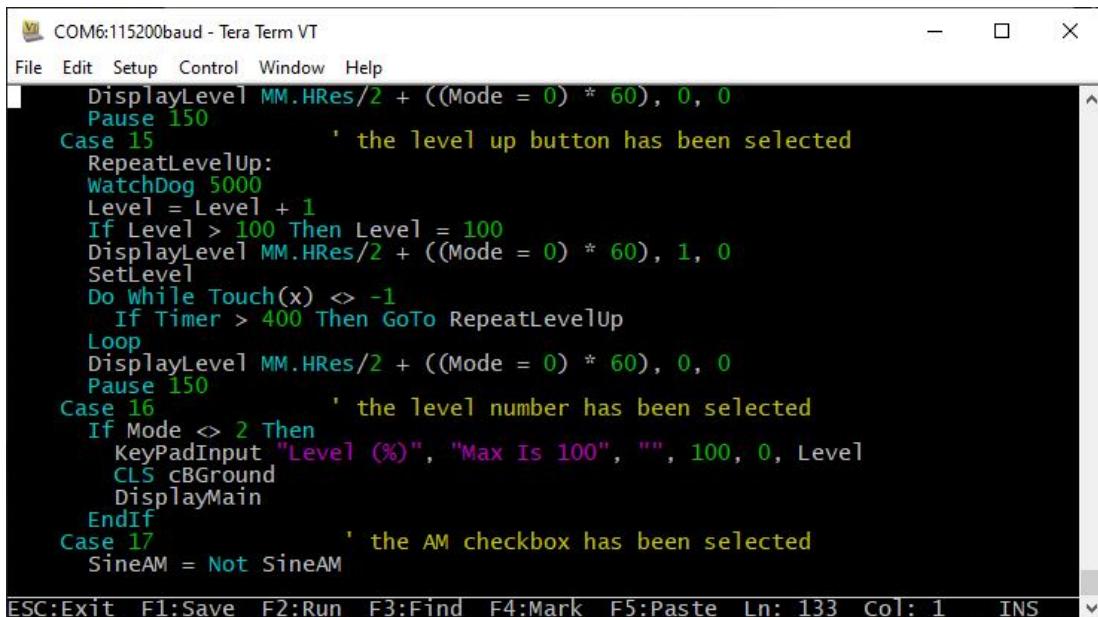
作为示例，以下将显示一个自定义提示：

```
SUB MM.PROMPT  
    PRINT TIME$ "> ";  
END SUB
```

请注意，虽然可以定义常量，但它们将不可见，因为在子程序内部定义的常量是局部的。然而，DIM将创建全局变量，应该使用这些变量。

Full Screen Editor

An important productivity feature is the built-in full screen editor. When running it looks like this:



The screenshot shows a window titled "COM6:115200baud - Tera Term VT". The menu bar includes File, Edit, Setup, Control, Window, and Help. The main area displays a BASIC program with syntax highlighting. The status bar at the bottom shows "ESC:Exit F1:Save F2:Run F3:Find F4:Mark F5:Paste Ln: 133 Col: 1 INS".

```
DisplayLevel MM.HRes/2 + ((Mode = 0) * 60), 0, 0
Pause 150
Case 15           ' the level up button has been selected
    RepeatLevelUp:
    WatchDog 5000
    Level = Level + 1
    If Level > 100 Then Level = 100
    DisplayLevel MM.HRes/2 + ((Mode = 0) * 60), 1, 0
    SetLevel
    Do While Touch(x) <> -1
        If Timer > 400 Then GoTo RepeatLevelUp
    Loop
    DisplayLevel MM.HRes/2 + ((Mode = 0) * 60), 0, 0
    Pause 150
Case 16           ' the level number has been selected
    If Mode <> 2 Then
        KeyPadInput "Level (%)", "Max Is 100", "", 100, 0, Level
        CLS cBGround
        DisplayMain
    EndIf
Case 17           ' the AM checkbox has been selected
    SineAM = Not SineAM
```

When the editor starts up the cursor will be automatically positioned at the last place that you were editing or, if your program had just been stopped by an error, the cursor will be positioned at the line that caused the error. At the bottom of the screen the status line lists details such as the current cursor position and the common functions supported by the editor.

If you have previously used an editor like Windows Notepad you will find that the operation of this editor is familiar. The arrow keys will move the cursor around in the text, home and end will take you to the beginning or end of the line. Page up and page down will do what their titles suggest. The delete key will delete the character at the cursor and backspace will delete the character before the cursor. The insert key will toggle between insert and overtype modes. About the only unusual key combination is that two home key presses will take you to the start of the program and two end key presses will take you to the end.

At the bottom of the screen the status line will list the various function keys used by the editor and their action. In more details these are:

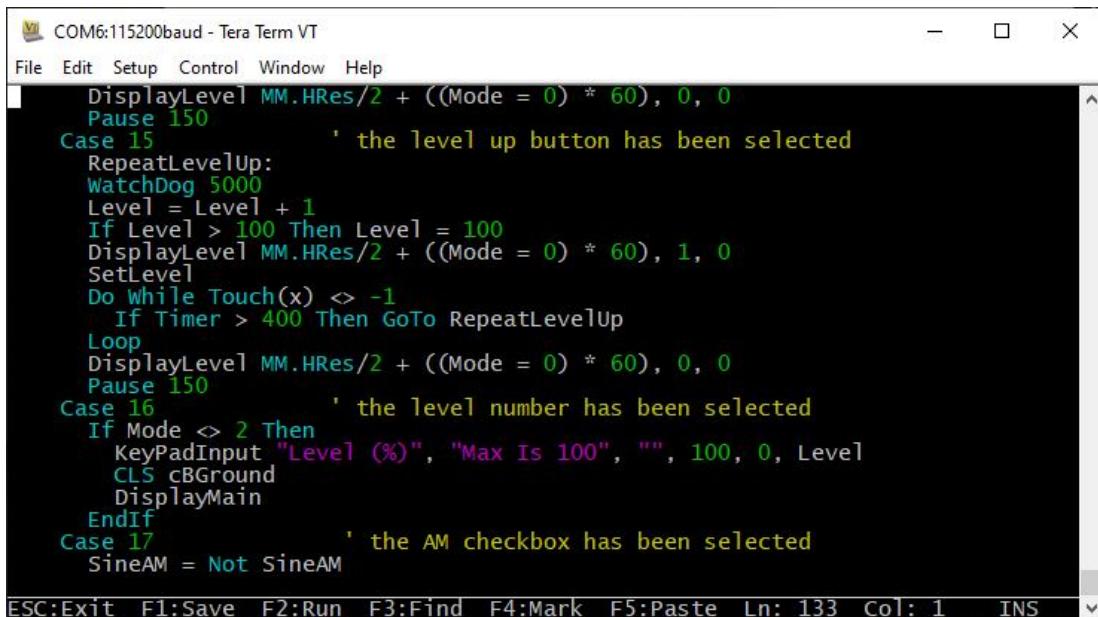
ESC	This will cause the editor to abandon all changes and return to the command prompt with the program memory unchanged. If you have changed the text you will be asked if you really want to abandon your changes.
F1: SAVE	This will save the program to program memory and return to the command prompt.
F2: RUN	This will save the program to program memory and immediately run it.
F3: FIND	This will prompt for the text that you want to search for. When you press enter the cursor will be placed at the start of the first entry found.
SHIFT-F3	Once you have used the search function you can repeat the search by pressing SHIFT-F3.
F4: MARK	This is described in detail below.
F5: PASTE	This will insert (at the current cursor position) the text that had been previously cut or copied (see below).

If you pressed the mark key (F4) the editor will change to the mark mode. In this mode you can use the arrow keys to mark a section of text which will be highlighted in reverse video. You can then delete, cut or copy the marked text. In this mode the status line will change to show the functions of the function keys in the mark mode. These keys are:

ESC	Will exit mark mode without changing anything.
F4: CUT	Will copy the marked text to the clipboard and remove it from the program.
F5: COPY	Will just copy the marked text to the clipboard.
DELETE	Will delete the marked text leaving the clipboard unchanged.

全屏编辑器

一个重要的生产力功能是内置的全屏编辑器。运行时看起来像这样：



The screenshot shows a terminal window titled "COM6:115200baud - Tera Term VT". The window contains a block of assembly-like code with syntax highlighting. The code includes labels like "Case 15", "RepeatLevelUp:", "Case 16", and "Case 17", along with various instructions such as "DisplayLevel", "Pause", "If", "Then", "Else", "EndIf", "KeyPadInput", "CLS", and "DisplayMain". The status bar at the bottom shows keyboard shortcuts: ESC:Exit F1:Save F2:Run F3:Find F4:Mark F5:Paste Ln: 133 Col: 1 INS.

当编辑器启动时，光标将自动定位在您最后编辑的位置，或者如果您的程序因错误而刚刚停止，光标将定位在导致错误的行。

在屏幕底部，状态行列出了当前光标位置和编辑器支持的常用功能等详细信息。

如果您之前使用过像Windows记事本这样的编辑器，您会发现这个编辑器的操作很熟悉。箭头键将光标在文本中移动，Home和End键将带您到行的开头或结尾。向上翻页和向下翻页将执行其标题所暗示的操作。删除键将删除光标处的字符，而退格键将删除光标前的字符。插入键将在插入模式和覆盖模式之间切换。唯一不寻常的键组合是，按两次主页键将带你到程序的开始，按两次结束键将带你到程序的结束。

在屏幕底部，状态行将列出编辑器使用的各种功能键及其操作。

更详细地说，这些是：

ESC 这将导致编辑器放弃所有更改，并在程序内存不变的情况下返回到命令提示符。如果您更改了文本，将会询问您是否真的想放弃更改。

F1: 保存 这将把程序保存到程序内存，并返回到命令提示符。

F2: 运行 这将把程序保存到程序内存并立即运行它。

F3: 查找 这将提示您输入要搜索的文本。当您按下回车键时，光标将放置在找到的第一个条目的开头。

SHIFT-F3 一旦您使用了搜索功能，可以通过按SHIFT-F3重复搜索。

F4: 标记 下面将详细描述。

F5: 粘贴 这将在当前光标位置插入之前剪切或复制的文本
(见下文)。

如果您按下了标记键 (F4)，编辑器将切换到标记模式。在此模式下，您可以使用箭头键标记一段文本，该文本将以反向视频高亮显示。然后，您可以删除、剪切或复制标记的文本。在此模式下，状态行将更改以显示标记模式下功能键的功能。这些键是：

ESC 将退出标记模式而不更改任何内容。

F4: 剪切 将标记的文本复制到剪贴板并从程序中删除。

F5: 复制 将标记的文本复制到剪贴板。

删除 将删除标记的文本，剪贴板保持不变。

You can also use control keys instead of the function keys listed above. These control keystrokes are:

LEFT	Ctrl-S	RIGHT	Ctrl-D	UP	Ctrl-E	DOWN	Ctrl-X
HOME	Ctrl-U	END	Ctrl-K	PageUp	Ctrl-P	PageDn	Ctrl-L
DEL	Ctrl-[INSERT	Ctrl-N	F1	Ctrl-Q	F2	Ctrl-W
F3	Ctrl-R	ShiftF3	Ctrl-G	F4	Ctrl-T	F5	Ctrl-Y

If you are using Tera Term, Putty, MMEdit or GFXterm as the terminal emulator it is also possible to position the cursor by left clicking the PC's mouse in the terminal emulator's window.

The best way to learn how to use the editor is to simply fire it up and experiment.

The editor is a very productive method of writing a program. With the command EDIT you can enter your program then, by pressing the F2 key, you can save and run the program. If your program stops with an error pressing the function key F4 at the command prompt will run the command EDIT and place you back in the editor with the cursor positioned at the line that caused the error. This edit/run/edit cycle is very fast.

Colour Coded Editor Display

The editor can colour code the edited program with keywords, numbers and comments displayed in different colours. This feature can be turned on or off with the command:

```
OPTION COLOURCODE ON or OPTION COLOURCODE OFF
```

This setting requires a compatible terminal emulator like Tera Term and is saved in non-volatile memory and automatically applied on start-up.

您也可以使用控制键代替上述功能键。这些控制键组合是：

左	Ctrl-S	右	Ctrl-D	上	Ctrl-E	下	Ctrl-X
主页	Ctrl-U	结束	Ctrl-K	页面上	Ctrl-P	页面下	Ctrl-L
删除	Ctrl-]	插入	Ctrl-N	F1	Ctrl-Q	F2	Ctrl-W
F3	Ctrl-R	ShiftF3	Ctrl-G	F4	Ctrl-T	F5	如果

您使用 Tera Term、Putty、MMEdit 或 GFXterm 作为终端仿真器，您还可以通过在终端仿真器窗口中左键单击 PC 的鼠标来定位光标。

学习如何使用编辑器的最佳方法是简单地启动它并进行实验。

编辑器是一种非常高效的编写程序的方法。使用命令 EDIT，您可以输入程序，然后按 F2 键即可保存并运行程序。如果您的程序因错误停止，按下功能键 F4 将在命令提示符下运行命令 EDIT，并将您带回编辑器，光标定位在导致错误的行上。这个编辑/运行/编辑循环非常快速。

颜色编码编辑器显示

编辑器可以对编辑的程序进行颜色编码，关键字、数字和注释以不同的颜色显示。此功能可以通过以下命令开启或关闭：

```
OPTION COLOURCODE ON or OPTION COLOURCODE OFF
```

此设置需要兼容的终端仿真器，如 Tera Term，并保存在非易失性内存中，启动时自动应用。

Program and Data Storage

The BASIC program is held in flash memory and is run from there. When a program is edited via EDIT or loaded via the console it will be saved there. Flash memory is non-volatile so the program will not be lost if the power is lost or the processor is reset. The maximum program size is 160KB.

In addition to this program memory there are three other locations where programs can be saved. These are described in detail below and are Flash Slots, the Flash Filesystem and an attached SD Card

Flash Slots

There are four of these which can be used to save completely different programs or previous versions of the program you are working on (in case you need to revert to an earlier version). In addition, MMBasic will allow a BASIC program to load and run another program saved to a numbered flash location while retaining all the variables and settings of the original program – this is called chaining and allows for a much larger program to be run than the amount of program memory would normally allow.

To manage these numbered locations in flash you can use the following commands (note that in the following *n* is a number from 1 to 3):

FLASH SAVE <i>n</i>	Save the program in the program memory to the flash location <i>n</i> .
FLASH LOAD <i>n</i>	Load a program from flash location <i>n</i> into the program memory.
FLASH RUN <i>n</i>	Run a program from flash location <i>n</i> , clears all variables but does not erase or change the program held in the main program memory.
FLASH LIST	Display a list of all flash locations including the first line of the program.
FLASH LIST <i>n</i> [,ALL]	Lists the program held in location <i>n</i> . Use FLASH LIST <i>n</i> ,ALL to list without page breaks
FLASH ERASE <i>n</i>	Erase flash location <i>n</i> .
FLASH ERASE ALL	Erase all flash locations.
FLASH CHAIN <i>n</i>	Load and run a program from flash location <i>n</i> , leaving all variables intact. As with FLASH RUN this command but does not erase or change the program held in the main program memory.
FLASH OVERWRITE <i>n</i>	Erase flash location <i>n</i> and then save the program in the program memory to that location.
FLASH DISK LOAD f\$ [,O]	Loads a flash slot from the binary file specified. Overwrites the slot if the optional “O” is specified.

In addition, the command OPTION AUTORUN can be used to specify a flash program location to be set running when power is applied or the CPU restarted. This option can also be used without specifying a flash location and in that case MMBasic will automatically load and run the program that is in the program memory.

Notes:

- It is recommended that you include a comment describing the program as the first line of the program. This will then be displayed by the FLASH LIST command and will help identify the program.
- All BASIC programs saved to flash may be erased if you upgrade (or downgrade) the PicoMite firmware. So make sure that you backup these first.
- The LIBRARY command uses Slot 3 for saving library data therefore only 2 slots will be available if the library feature is used.

Flash Filesystem

This is an area of the Raspberry Pi Pico’s flash memory which is automatically created by the firmware and will look like a normal disk drive to MMBasic. It is called drive A: and data and programs can be read/written using the normal BASIC file commands (SAVE, RUN, OPEN, etc). In addition, sub directories can be created and deleted and long filenames used.

For example, to run a program:

```
RUN "A:/MyProgram.bas"
```

Open a text file for random access:

```
OPEN "A:/data/database.dat" FOR RANDOM as #4
```

程序和数据存储

BASIC 程序保存在闪存中，并从那里运行。当通过 EDIT 编辑程序或通过控制台加载程序时，它将保存在那里。闪存是非易失性的，因此如果断电或处理器重置，程序不会丢失。最大程序大小为 16 0KB。

除了这个程序内存外，还有三个其他位置可以保存程序。这些将在下面详细描述，包括闪存槽、闪存文件系统和附加的 SD 卡。

闪存槽

这些槽有四个，可以用来保存完全不同的程序或您正在处理的程序的先前版本（以防您需要恢复到早期版本）。此外，MMBasic 允许一个 BASIC 程序加载并运行另一个保存到编号闪存位置的程序，同时保留原始程序的所有变量和设置——这被称为链式调用，允许运行比程序内存通常允许的更大的程序。

要管理闪存中的这些编号位置，您可以使用以下命令（请注意，在以下内容中 *n* 是从 1 到 3 的数字）：

FLASH SAVE <i>n</i>	将程序内存中的程序保存到闪存位置 <i>n</i> 。
FLASH LOAD <i>n</i>	从闪存位置 <i>n</i> 加载程序到程序内存中。
FLASH RUN <i>n</i>	从闪存位置 <i>n</i> 运行程序，清除所有变量，但不会擦除或更改主程序内存中保存的程序。
FLASH LIST	显示所有闪存位置的列表，包括程序的第一行。
FLASH LIST <i>n</i> [,ALL]	列出位置 <i>n</i> 中保存的程序。使用 FLASH LIST <i>n</i> ,ALL 列出而不带分页。
闪存擦除 <i>n</i>	擦除闪存位置 <i>n</i> 。
闪存擦除全部	擦除所有闪存位置。
闪存链 <i>n</i>	从闪存位置 <i>n</i> 加载并运行程序，同时保留所有变量不变。 与闪存运行相同，但不会擦除或更改主程序内存中的程序。
闪存覆盖 <i>n</i>	擦除闪存位置 <i>n</i> ，然后将程序保存到该位置的程序内存中。

闪存磁盘加载 f\$ [,O] 从指定的二进制文件加载闪存槽。如果指定了可选的“O”，则会覆盖该槽。

此外，可以使用命令选项自动运行来指定一个闪存程序位置，以便在通电或中央处理器重启时运行。此选项也可以在不指定闪存位置的情况下使用，在这种情况下，MMBasic 将自动加载并运行程序内存中的程序。

注意：

- 建议您在程序的第一行包含描述程序的注释。这将通过 FLASH LIST 命令显示，并将帮助识别程序。
- 所有保存到闪存的 BASIC 程序在升级（或降级）PicoMite 固件时可能会被擦除。
因此，请确保先备份这些程序。
- LIBRARY 命令使用 Slot3 来保存库数据，因此如果使用库功能，则仅会有 2 个插槽可用。

Flash 文件系统

这是树莓派 Pico 的闪存中的一个区域，由固件自动创建，并且在 MMBasic 中看起来像一个普通的磁盘驱动器。它被称为驱动器 A:，可以使用正常的 BASIC 文件命令（SAVE、RUN、OPEN 等）读取/写入数据和程序。此外，可以创建和删除子目录，并使用长文件名。

例如，要运行一个程序：

```
RUN "A:/MyProgram.bas"
```

打开一个文本文件以进行随机访问：

```
OPEN "A:/data/database.dat" FOR RANDOM as #4
```

Nothing needs to be done to create this drive so it will always be available to the BASIC program.

The system will create and maintain the file "BOOTCOUNT" on the Flash Filesystem. This keeps a count of the number of times the PicoMite has been restarted and can be read with the function MM.INFO(boot count).

SD Cards

An SD Card socket can be connected to the PicoMite and accessed as drive B:. Like the Flash Filesystem the normal BASIC file commands can be used to save/load programs as well as opening data files for read/write.

Cards up to 32 GB formatted in FAT16 or FAT32 are supported and the files created can also be read/written on personal computers running Windows, Linux or the Mac operating system. The PicoMite uses the SPI protocol to talk to the card and this is not influenced by the card type, so all types (Class 4, 10, UHS-1 etc) are supported

The SPI protocol needs to be specifically configured before it can be used. First the "system" SPI port needs to be configured. This is a port that will be used for system use (SD Card, LCD display and the touch controller on an LCD panel).

There are a number of ports and pins that can be used (see the section *PicoMite Hardware*) but this example uses SPI on pins GP18, GP19 and GP16 for Clock, MOSI and MISO.

```
OPTION SYSTEM SPI GP18, GP19, GP16
```

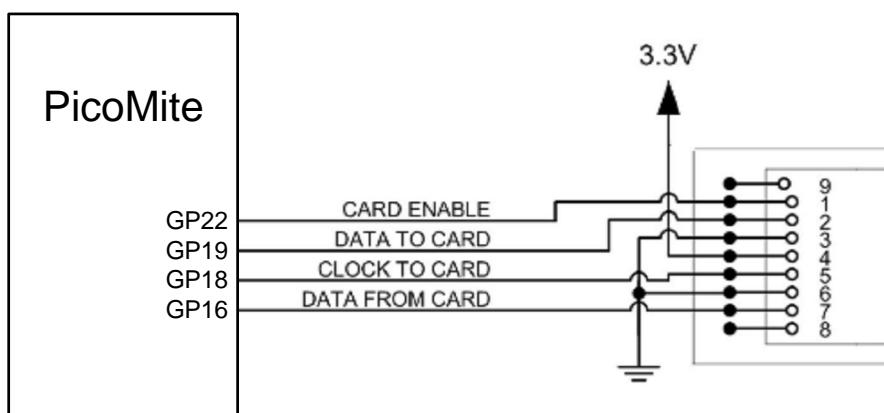
Then MMBasic must be told that there is an SD Card attached and what pin is used for the Chip Select signal:

```
OPTION SDCARD GP22
```

These commands must be entered at the command prompt (not in a program) and will cause the PicoMite to restart. This has the side effect of disconnecting the USB console interface which will need to be reconnected.

When the PicoMite is restarted MMBasic will automatically initialise the SD Card interface. This SPI port will then not be available to BASIC programs (i.e. it is reserved). To verify the configuration, you can use the command OPTION LIST to list all options that have been set including the configuration of the SD Card.

The basic circuit diagram for connecting the SD Card connector using these pin allocations is illustrated below.



Note that you can use many different configurations using various pin allocations – this is just an example based on the configuration commands listed above.

Care must be taken when the SPI port is shared between a number of devices (SD Card, touch, etc). In this case all the Chip Select signals must be configured in MMBasic or alternatively disabled by a permanent connection to 3.3V. If this is not done any floating Chip Select signal lines will cause the wrong controller to respond to commands on the SPI bus.

Where no other devices share the SPI bus the SD Card can be set up with:

```
OPTION SDCARD CSpin, CLKpin, MOSIpin, MISOpin
```

In this case the pins can be assigned completely flexibly and do not need to be capable of SPI operation.

MMBasic Support for Flash and SD Card Filesystems

The MMBasic support for the Flash Filesystem and SD Cards is almost identical. This allows programs to use either filesystem with minimal modification. The Flash Filesystem is referred to as drive A: while the SD Card (when connected) is drive B:. The default drive can be set with the DRIVE command and then the drive prefix is not needed.

In the following note that:

- On startup the active drive (ie, the default) is A: (the Flash Filesystem).

创建此驱动器不需要任何操作，因此它将始终可用于BASIC程序。

系统将在Flash文件系统上创建并维护文件"BOOTCOUNT"。这会记录PicoMite重启的次数，并可以通过函数MM.INFO(boot count)读取。

SD卡

可以将SD卡插槽连接到PicoMite，并作为驱动器B:访问。与Flash文件系统一样，可以使用普通的BASIC文件命令来保存/加载程序，以及打开数据文件进行读/写。

支持格式为FAT16或FAT32的最大32GB的卡，所创建的文件也可以在运行Windows、Linux或Mac操作系统的个人计算机上进行读/写。PicoMite使用SPI协议与卡进行通信，这不受卡类型的影响，因此所有类型（Class 4、10、UHS-1等）均受支持。

在使用之前，SPI协议需要进行特定配置。首先需要配置“系统”SPI端口。这是一个将用于系统用途的端口（SD卡、液晶显示和液晶面板上的触摸控制器）。

有多个端口和引脚可以使用（请参见部分*PicoMite*硬件），但此示例使用引脚GP18、GP19和GP16的SPI作为时钟、MOSI和MISO。

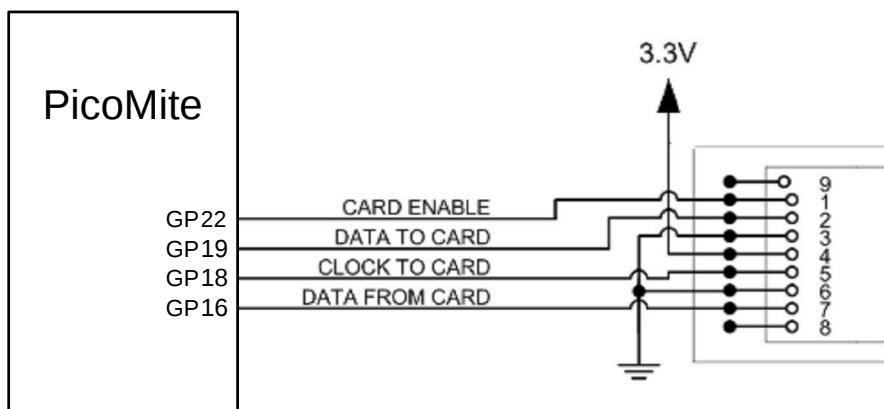
```
OPTION SYSTEM SPI GP18, GP19, GP16
```

然后必须告诉MMBasic有一个SD卡连接，并且使用哪个引脚作为芯片选择信号：

```
OPTION SDCARD GP22
```

这些命令必须在命令提示符下输入（而不是在程序中），并将导致PicoMite重启。这会导致USB控制台接口断开，需要重新连接。

当PicoMite重启时，MMBasic将自动初始化SD卡接口。此SPI端口将不再可用于BASIC程序（即，它是保留的）。要验证配置，可以使用命令OPTION LIST列出所有已设置的选项，包括SD卡的配置。以下是使用这些引脚分配连接SD卡连接器的基本电路图。



请注意，您可以使用多种不同的配置，使用各种引脚分配——这只是基于上述配置命令的一个示例。

当SPI端口在多个设备（SD卡、触摸等）之间共享时，必须小心。在这种情况下，所有的芯片选择信号必须在MMBasic中配置，或者通过永久连接到3.3V来禁用。如果不这样做，任何浮动的芯片选择信号线将导致错误的控制器响应SPI总线上的命令。

如果没有其他设备共享SPI总线，可以将SD卡设置为：

```
OPTION SDCARD CSpin, CLKpin, MOSIpin, MISOpin
```

在这种情况下，引脚可以完全灵活地分配，并且不需要能够进行SPI操作。

MMBasic对Flash和SD卡文件系统的支持

MMBasic对Flash文件系统和SD卡的支持几乎是相同的。这允许程序在最小修改的情况下使用任一文件系统。Flash文件系统被称为驱动器A:，而SD卡（连接时）为驱动器B:。默认驱动器可以通过DRIVE命令设置，此后不需要驱动器前缀。

在以下内容中注意：

- 启动时，活动驱动器（即默认驱动器）为A:（Flash文件系统）。

- Any file path that uses the drive letter must be a full path from the root (ie, “A:/mypath/myfile.txt”).
- Long file/directory names are supported in addition to the old 8.3 format.
- The maximum file/path length is 63 characters.
- Upper/lowercase characters and spaces are allowed. The file system on the SD Card is **NOT** case sensitive however the Flash Filesystem **IS** case sensitive.
- Directory paths are allowed in file/directory strings. (i.e., OPEN "A:\dir1\dir2\file.txt" FOR ...).
- Either forward or back slashes can be used in paths. E.g. \dir\file.txt is the same as /dir/file.txt.
- The current PicoMite time is used for file create and last access times.
- Up to ten files can be simultaneously open and they can be on both the A: drive and the B: drive.
- Except for INPUT, LINE INPUT and PRINT the # in #fnbr is optional and may be omitted.

Programs can be loaded from or saved to the Flash Filesystem and SD Cards using these commands.

- LOAD fname\$ [, R]
Load a BASIC program. The optional suffix ",R" will cause the program to be run after it has been loaded (in this case fname\$ must be a string constant).
- RUN fname\$
Load a BASIC program and run it. fname\$ must be a string constant.
- SAVE fname\$
Save the current program to the Flash Filesystem or SD Card.

These are the basic commands for reading and writing data.

- OPEN fname\$ FOR mode AS #fnbr
Opens a file for reading or writing. 'fname\$' is the file name in 8.3 format. 'mode' can be INPUT, OUTPUT, APPEND or RANDOM. '#fnbr' is the file number (1 to 10).
- PRINT #fnbr, expression [[,]expression] ... etc
Outputs text to the file opened as #fnbr.
- INPUT #fnbr, list of variables
Read a list of comma separated data into the variables specified from the file previously opened as #fnbr.
- LINE INPUT #fnbr, variable\$
Read a complete line into the string variable specified from the file previously opened as #fnbr.
- FLUSH #fnbr
Forces any buffered writes to be written to the Flash Filesystem or SD Card. It is recommended that this command be used regularly where data loss could occur in the event of power loss.
- CLOSE #fnbr [,#fnbr] ...
Close the file(s) previously opened with the file number '#fnbr'.

Basic file and directory manipulation. Most can be done at the command prompt or from within a BASIC program.

- DRIVE drive\$
Sets the active disk drive as ‘drive\$’. ‘drive\$’ can be “A:” or “B:” where A is the flash drive and B is the SD Card (if configured).
- DRIVE "A:/FORMAT"
Reformat the Flash Filesystem (drive A:) to its initial state.
- FILES [wildcard]
Search the current directory and list the files/directories found. Note: Can only be used at the command prompt, not within a program.
- LIST fname\$
List the contents of a program or text file on the console.

- 任何使用驱动器字母的文件路径必须是从根目录开始的完整路径（即，“A:/mypath/myfile.txt”）。
- 除了旧的 8.3 格式外，还支持长文件/目录名称。
- 最大文件/路径长度为 63 个字符。
- 允许使用大小写字母和空格。SD 卡上的文件系统不区分大小写，但 Flash 文件系统是区分大小写的。
- 文件/目录字符串中允许使用目录路径。（即，OPEN "A:\dir1\dir2\file.txt" FOR ...）。
- 路径中可以使用正斜杠或反斜杠。例如 \dir\file.txt 与 /dir/file.txt 是相同的。
- 当前的 PicoMite 时间用于文件创建和最后访问时间。
- 最多可以同时打开十个文件，它们可以位于 A: 驱动器和 B: 驱动器上。
- 除了 INPUT、LINE INPUT 和 PRINT，# 在 #fnbr 中是可选的，可以省略。

程序可以使用这些命令从 Flash 文件系统或 SD 卡加载或保存。

- LOAD fname\$ [, R]
加载一个 BASIC 程序。可选后缀 ",R" 将导致程序在加载后运行（在这种情况下 fname\$ 必须是一个字符串常量）。
- RUN fname\$
加载一个 BASIC 程序并运行它。fname\$ 必须是一个字符串常量。
- 保存 fname\$
将当前程序保存到 Flash 文件系统或 SD 卡。

这些是用于读取和写入数据的基本命令。

- 打开 fname\$ 以 mode 的方式作为 #fnbr
打开一个文件以进行读取或写入。'fname' 是 8.3 格式的文件名。'mode' 可以是 INPUT、OUTPUT、APPEND 或 RANDOM。'#fnbr' 是文件编号（1 到 10）。
- PRINT #fnbr, 表达式 [[,]表达式] ... 等等
将文本输出到以 #fnbr 打开的文件。
- INPUT #fnbr, 变量列表
从之前以 #fnbr 打开的文件中读取一组以逗号分隔的数据到指定的变量中。
- LINE INPUT #fnbr, 变量\$
从之前以 #fnbr 打开的文件中读取完整的一行到指定的字符串变量中。
- FLUSH #fnbr
强制将任何缓冲的写入写入 Flash 文件系统或 SD 卡。建议在可能因断电而导致数据丢失的情况下定期使用此命令。
- 关闭 #fnbr [,#fnbr] ...
关闭之前使用文件编号 '#fnbr' 打开的文件。

基本文件和目录操作。大多数操作可以在命令提示符下或在 BASIC 程序中完成。

- 驱动器 drive\$
将活动磁盘驱动器设置为 'drive\$'。'drive\$' 可以是 "A:" 或 "B:", 其中 A 是闪存驱动器, B 是 SD 卡（如果已配置）。
- 驱动器 "A:/格式"
将闪存文件系统（驱动器 A:）重新格式化为初始状态。
- 文件 [通配符]
搜索当前目录并列出找到的文件/目录。注意：只能在命令提示符下使用，不能在程序中使用。
- 列表 fname\$
在控制台上列出程序或文本文件的内容。

- KILL fname\$**
Delete a file in the current directory on the current drive. See the command reference for more details on wildcard deletes.
- MKDIR dname\$**
Creates a sub directory in the current directory on the current drive.
- CHDIR dname\$**
Change into to the directory \$dname. \$dname can also be "." (dot dot) for up one directory or "\\" for the root directory. The starting point is the current directory on the current drive.
- RMDIR dir\$**
Remove, or delete, the directory 'dir\$' in the current directory on the current drive.
- SEEK #fnbr, pos**
Will position the read/write pointer in a file that has been opened for RANDOM access to the 'pos' byte.
- RENAME fromname\$ AS toname\$**
Will rename the file fromname\$ to have the name toname\$ in the current directory on the current drive
- COPY [mode] fromname\$ TO toname\$**
Will copy the file fromname\$ to have the file toname\$. See the command reference for more details on the optional mode and wildcard copies.

Also there are a number of functions that support the above commands.

- INPUT\$(nbr, #fnbr)**
Will return a string composed of 'nbr' characters read from a file previously opened for INPUT with the file number '#fnbr'. If less than 'nbr' characters are available the function will return with what it has (including an empty string if no characters are available).
- DIR\$(fspec, type)**
Will search for files and return the names of entries found.
- CWD\$**
Will return the current working directory.
- EOF(#fnbr)**
Will return true if the file previously opened for INPUT with the file number '#fnbr' is positioned at the end of the file.
- LOC(#fnbr)**
For a file opened as RANDOM this will return the current position of the read/write pointer in the file.
- LOF(#fnbr)**
Will return the current length of the file in bytes.
- MM.INFO(drive)**
Will return the current active drive – ie, "A:" or "B:"
- MM.INFO(free space)**
Will return how much space is left on the active drive
- MM.INFO(disk size)**
Will return the size of the active drive
- MM.INFO(exists file fname\$)**
Will return true if the file exists
- MM.INFO(exists dir dirname\$)**
Will return true if the directory exists

- 删除 fname\$**
删除当前驱动器上当前目录中的文件。有关通配符删除的更多详细信息，请参见命令参考。
- MKDIR dname\$**
在当前驱动器的当前目录中创建一个子目录。
- CHDIR dname\$**
切换到目录 \$dname。 \$dname 也可以是 "." (点) 表示上一级目录，或 "\" 表示根目录。起始点是当前驱动器上的当前目录。
- RMDIR dir\$**
在当前驱动器的当前目录中删除目录 ‘dir\$’。
- SEEK #fnbr, pos**
将在已打开的随机访问文件中将读/写指针定位到 'pos' 字节。
- RENAME fromname\$ AS toname\$**
将在当前驱动器的当前目录中将文件 fromname\$ 重命名为 toname\$。
- COPY [mode] fromname\$ TO toname\$**
将文件 fromname\$ 复制为文件 toname\$。有关可选模式和通配符复制的更多详细信息，请参阅命令参考。

此外，还有许多函数支持上述命令。

- INPUT\$(nbr, #fnbr)**
将返回由‘nbr’个字符组成的字符串，这些字符是从之前以文件编号‘#fnbr’打开的文件中读取的。如果可用字符少于‘nbr’，该函数将返回它所读取的内容（如果没有可用字符，则返回空字符串）。
- DIR\$(fspec, type)**
将搜索文件并返回找到的条目的名称。
- CWD\$**
将返回当前工作目录。
- EOF(#fnbr)**
如果之前以文件编号‘#fnbr’打开的文件位于文件末尾，则返回真。
- LOC(#fnbr)**
对于以RANDOM方式打开的文件，这将返回读/写指针在文件中的当前位置。
- LOF(#fnbr)**
将返回文件当前的字节长度。
- MM.INFO(drive)**
将返回当前活动驱动器——即，“A:”或“B:”
- MM.INFO(free space)**
将返回活动驱动器上剩余的空间
- MM.INFO(磁盘大小)**
将返回活动驱动器的大小
- MM.INFO(文件存在 fname\$)**
如果文件存在，将返回真
- MM.INFO(目录存在 dirname\$)**
如果目录存在，将返回真

XModem Transfer

In addition to the standard method of XModem transfer which copies to or from the program memory the PicoMite can also copy to and from a file on the Flash Filesystem or SD Card. The syntax is:

```
XMODEM SEND filename$
```

or

```
XMODEM RECEIVE filename$
```

Where ‘filename\$’ is the file to save or send. ‘filename\$’ can be a string expression, variable or constant. If it is a constant the string must be quoted (e.g., XMODEM SEND "prbas.bas"). In the case of receiving a file, any file with the same name will be overwritten.

Load and Save Image

An image can be loaded from the Flash Filesystem or SD Card for display on an attached LCD display panel. This can be used to draw a logo or add a background on the display. The syntax is:

```
LOAD IMAGE filename$ [, StartX, StartY]
```

or

```
LOAD JPG filename$ [, StartX, StartY]
```

Where ‘filename\$’ is the image to load and ‘StartX’/‘StartY’ are the coordinates of the top left corner of the image (these are optional and will default to the top left corner of the display if not specified).

The image must be in the BMP format (for LOAD IMAGE) or JPG format (for LOAD JPG) and MMBasic will add “.BMP” or “.JPG” to the file name if an extension is not specified. All types of the BMP or JPG formats are supported including black and white and true colour 24-bit images.

The current image on a ILI9341 based LCD screen can be saved to a file using the following command:

```
SAVE IMAGE filename$ [,StartX, StartY, width, height]
```

This will save the image, or part of the image, as a 24-bit true colour BMP file (the extension .BMP) will be added if an extension is not supplied.

Example of Sequential I/O

In the example below a file is created and two lines are written to the file (using the PRINT command). The file is then closed.

```
OPEN "fox.txt" FOR OUTPUT AS #1
PRINT #1, "The quick brown fox"
PRINT #1, "jumps over the lazy dog"
CLOSE #1
```

You can read the contents of the file using the LINE INPUT command. For example:

```
OPEN "fox.txt" FOR INPUT AS #1
LINE INPUT #1,a$
LINE INPUT #1,b$
CLOSE #1
```

LINE INPUT reads one line at a time so the variable a\$ will contain the text "The quick brown fox" and b\$ will contain "jumps over the lazy dog".

Another way of reading from a file is to use the INPUT\$() function. This will read a specified number of characters. For example:

```
OPEN "fox.txt" FOR INPUT AS #1
ta$ = INPUT$(12, #1)
tb$ = INPUT$(3, #1)
CLOSE #1
```

XModem传输

除了标准的XModem传输方法（将数据复制到或从程序内存），PicoMite还可以从Flash文件系统或SD卡的文件中进行复制。语法为：

XMODEM 发送 filename\$

或

XMODEM 接收 filename\$

其中‘filename\$’是要保存或发送的文件。‘filename\$’可以是字符串表达式、变量或常量。如果是常量，则字符串必须用引号括起来（例如，XMODEM 发送 "prbas.bas"）。在接收文件的情况下，任何同名文件将被覆盖。

加载和保存图像

可以从Flash文件系统或SD卡加载图像，以在连接的液晶显示面板上显示。这可以用于在显示器上绘制徽标或添加背景。语法为：

LOAD IMAGE filename\$ [, StartX, StartY]

或

LOAD JPG filename\$ [, StartX, StartY]

其中‘filename\$’是要加载的图像，‘StartX’/‘StartY’是图像左上角的坐标（这些是可选的，如果未指定，将默认为显示器的左上角）。

图像必须是BMP格式（用于LOAD IMAGE）或JPG格式（用于LOAD JPG），如果未指定扩展名，MM Basic将自动添加“.BMP”或“.JPG”到文件名中。支持所有类型的BMP或JPG格式，包括黑白和真彩色24位图像。

可以使用以下命令将当前在基于ILI9341的液晶屏上的图像保存到文件：

保存图像 filename\$ [, StartX, StartY, width, height]

这将保存图像或图像的一部分，如果未提供扩展名，将添加24位真彩BMP文件（扩展名为.BMP）。

顺序输入/输出示例

在下面的示例中，创建了一个文件并向文件中写入了两行（使用PRINT命令）。然后关闭该文件。

```
打开 "fox.txt" 以输出方式作为 #1
打印 #1, "快速的棕色狐狸"
打印 #1, "跳过懒狗"
关闭 #1
```

您可以使用行输入命令读取文件的内容。例如：

```
打开 "fox.txt" 以输入方式作为 #1
行输入 #1,a$
行输入 #1,b$
关闭 #1
```

行输入一次读取一行，因此变量 a\$将包含文本"快速的棕色狐狸"，而 b\$将包含"跳过懒狗"。

从文件读取的另一种方法是使用 INPUT\$() 函数。这将读取指定数量的字符。例如：

```
OPEN "fox.txt" FOR INPUT AS #1
ta$ = INPUT$(12, #1)
tb$ = INPUT$(3, #1)
CLOSE #1
```

The first INPUT\$() will read 12 characters and the second three characters. So the variable ta\$ will contain "The quick br" and the variable tb\$ will contain "own".

Files normally contain just text and the print command will convert numbers to text. So in the following example the first line will contain the line "123" and the second "56789".

```
nbr1 = 123 : nbr2 = 56789
OPEN "numbers.txt" FOR OUTPUT AS #1
PRINT #1, nbr1
PRINT #1, nbr2
CLOSE #1
```

Again you can read the contents of the file using the LINE INPUT command but then you would need to convert the text to a number using VAL().

For example:

```
OPEN "numbers.txt" FOR INPUT AS #1
LINE INPUT #1, a$
LINE INPUT #1, b$
CLOSE #1
x = VAL(a$) : y = VAL(b$)
```

Following this the variable x would have the value 123 and y the value 56789.

Random File I/O

For random access the file should be opened with the keyword RANDOM. For example:

```
OPEN "filename" FOR RANDOM AS #1
```

To seek to a record within the file you would use the SEEK command which will position the read/write pointer to a specific byte. The first byte in a file is numbered one so, for example, the fifth record in a file that uses 64 byte records would start at byte 257. In that case you would use the following to point to it:

```
SEEK #1, 257
```

When reading from a random access file the INPUT\$() function should be used as this will read a fixed number of bytes (i.e. a complete record) from the file. For example, to read a record of 64 bytes you would use:

```
dat$ = INPUT$(64, #1)
```

When writing to the file a fixed record size should be used and this can be easily accomplished by adding sufficient padding characters (normally spaces) to the data to be written. For example:

```
PRINT #1, dat$ + SPACE$(64 - LEN(dat$));
```

The SPACE\$() function is used to add enough spaces to ensure that the data written is an exact length (64 bytes in this example). The semicolon at the end of the print command suppresses the addition of the carriage return and line feed characters which would make the record longer than intended.

Two other functions can help when using random file access. The LOC() function will return the current byte position of the read/write pointer and the LOF() function will return the total length of the file in bytes.

The following program demonstrates random file access. Using it you can append to the file (to add some data in the first place) then read/write records using random record numbers. The first record in the file is record number 1, the second is 2, etc.

```
RecLen = 64
OPEN "test.dat" FOR RANDOM AS #1
DO
    abort: PRINT
    PRINT "Number of records in the file =" LOF(#1)/RecLen
    INPUT "Command (r = read, w = write, a = append, q = quit): ", cmd$
    IF cmd$ = "q" THEN CLOSE #1 : END
    IF cmd$ = "a" THEN
```

第一个 INPUT\$() 将读取 12 个字符，第二个将读取 3 个字符。因此变量 ta\$ 将包含 "The quick br"，而变量 tb\$ 将包含 "own"。

文件通常只包含文本，打印命令会将数字转换为文本。因此在以下示例中，第一行将包含行 "123"，第二行将包含 "56789"。

```
nbr1 = 123 : nbr2 = 56789
OPEN "numbers.txt" FOR OUTPUT AS #1
PRINT #1, nbr1
PRINT #1, nbr2
CLOSE #1
```

同样，您可以使用 LINE INPUT 命令读取文件的内容，但您需要使用 VAL() 将文本转换为数字。

例如：

```
OPEN "numbers.txt" FOR INPUT AS #1
LINE INPUT #1, a$
LINE INPUT #1, b$
CLOSE #1
x = VAL(a$) : y = VAL(b$)
```

因此变量 x 的值将为 123，而 y 的值将为 56789。

随机文件输入/输出

对于随机访问，文件应使用关键字 RANDOM 打开。例如：

```
OPEN "filename" FOR RANDOM AS #1
```

要在文件中查找记录，您可以使用 SEEK 命令，该命令将读/写指针定位到特定字节。文件中的第一个字节编号为 1，因此，例如，使用 64 字节记录的文件中的第五条记录将从字节 257 开始。在这种情况下，您可以使用以下命令指向它：

```
SEEK #1, 25 7
```

从随机访问文件读取时，应使用 INPUT\$() 函数，因为它将从文件中读取固定数量的字节（即完整记录）。例如，要读取一个 64 字节的记录，你可以使用：

```
dat$ = INPUT$(64, #1)
```

在写入文件时，应使用固定的记录大小，这可以通过向要写入的数据添加足够的填充字符（通常是空格）来轻松实现。例如：

```
PRINT #1, dat$ + SPACE$(64 - LEN(dat$));
```

SPACE\$() 函数用于添加足够的空格，以确保写入的数据具有确切的长度（在此示例中为 64 字节）。打印命令末尾的分号抑制了回车和换行字符的添加，这会使记录比预期的更长。

还有两个其他函数可以帮助使用随机文件访问。LOC() 函数将返回当前读/写指针的字节位置，而 LOF() 函数将返回文件的总长度（以字节为单位）。

以下程序演示了随机文件访问。使用它，你可以向文件追加数据（首先添加一些数据），然后使用随机记录号读取/写入记录。文件中的第一个记录是记录编号 1，第二个是 2，依此类推。

```
RecLen = 64
打开 "test.dat" 以随机方式作为 #1
DO
    abort: 打印
    打印 "文件中的记录数 = " LOF(#1) / RecLen
    输入 "命令 (r = 读取, w = 写入, a = 追加, q = 退出)：" , cmd$
    如果 cmd$ = "q" 那么 关闭 #1 : 结束
    如果 cmd$ = "a" 那么
```

```

SEEK #1, LOF(#1) + 1
ELSE
  INPUT "Record Number: ", nbr
  IF nbr < 1 or nbr > LOF(#1)/RecLen THEN PRINT "Invalid record" : GOTO abort
  SEEK #1, RecLen * (nbr - 1) + 1
ENDIF
IF cmd$ = "r" THEN
  PRINT "The record = " INPUT$(RecLen, #1)
ELSE
  LINE INPUT "Enter the data to be written: ", dat$
  PRINT #1,dat$ + SPACE$(RecLen - LEN(dat$));
ENDIF
LOOP

```

Random access can also be used on a normal text file. For example, this will print out a file backwards:

```

OPEN "file.txt" FOR RANDOM AS #1
FOR i = LOF(#1) TO 1 STEP -1
  SEEK #1, i
  PRINT INPUT$(1, #1);
NEXT i
CLOSE #1

```

```
SEEK #1, LOF(#1) + 1
否则
    输入 "记录编号: ", nbr
    如果 nbr < 1 或 nbr > LOF(#1)/RecLen 那么 打印 "无效记录" : 转到 abort
        SEEK #1, RecLen * (nbr - 1) + 1
结束如果
如果 cmd$ = "r" 那么
    打印 "记录 = " 输入$(RecLen, #1)
否则
    行输入 "请输入要写入的数据: ", dat$
    打印 #1, dat$ + 空间$(RecLen - LEN(dat$));
结束如果
循环
```

随机访问也可以用于普通文本文件。例如，这将反向打印文件：

```
打开 "file.txt" 以随机方式作为 #1
对于 i = LOF(#1) 到 1 步长 -1
    寻址 #1, i
    打印 输入$(1, #1);
下一个 i
关闭 #1
```

Variables and Expressions

In MMBasic command names, function names, labels, variable names, file names, etc are not case sensitive, so that "Run" and "RUN" are equivalent and "dOO" and "Doo" refer to the same variable.

Variables

Variables can start with an alphabetic character or underscore and can contain any alphabetic or numeric character, the period (.) and the underscore (_). They may be up to 31 characters long.

A variable name or a label must not be the same as a function or one of the following keywords: THEN, ELSE, GOTO, GOSUB, TO, STEP, FOR, WHILE, UNTIL, LOAD, MOD, NOT, AND, OR, XOR, AS.

E.g. step = 5 is illegal as STEP is a keyword.

MMBasic supports three types of variables:

1. Double Precision Floating Point.

These can store a number with a decimal point and fraction (e.g. 45.386) however they will lose accuracy when more than 14 digits of precision are used. Floating point variables are specified by adding the suffix '! to a variable's name (e.g. i!, nbr!, etc). They are also the default when a variable is created without a suffix (e.g. i, nbr, etc).

2. 64-bit Signed Integer.

These can store positive or negative numbers with up to 19 decimal digits without losing accuracy but they cannot store fractions (i.e. the part following the decimal point). These are specified by adding the suffix '%' to a variable's name. For example, i%, nbr%, etc.

3. A String.

A string will store a sequence of characters (e.g. "Tom"). Each character in the string is stored as an eight bit number and can therefore have a decimal value of 0 to 255. String variable names are terminated with a '\$' symbol (e.g. name\$, s\$, etc). Strings can be up to 255 characters long.

Note that it is illegal to use the same variable name with different types. E.g. using nbr! and nbr% in the same program would cause an error.

Most programs use floating point variables for arithmetic as these can deal with the numbers used in typical situations and are more intuitive than integers when dealing with division and fractions. So, if you are not bothered with the details, always use floating point.

Constants

Numeric constants may begin with a numeric digit (0-9) for a decimal constant, &H for a hexadecimal constant, &O for an octal constant or &B for a binary constant. For example &B1000 is the same as the decimal constant 8. Constants that start with &H, &O or &B are always treated as 64-bit unsigned integer constants.

Decimal constants may be preceded with a minus (-) or plus (+) and may be terminated with 'E' followed by an exponent number to denote exponential notation. For example 1.6E+4 is the same as 16000.

When a constant number is used it will be assumed that it is an integer if a decimal point or exponent is not used. For example, 1234 will be interpreted as an integer while 1234.0 will be interpreted as a floating point number.

String constants must be surrounded by double quote marks (""). E.g. "Hello World".

OPTION DEFAULT

A variable can be used without a suffix (i.e. !, % or \$) and in that case MMBasic will use the default type of floating point. For example, the following will create a floating point variable:

```
Nbr = 1234
```

However, the default can be changed with the OPTION DEFAULT command. For example, OPTION DEFAULT INTEGER will specify that all variables without a specific type will be integer. So, the following will create an integer variable:

```
OPTION DEFAULT INTEGER
Nbr = 1234
```

变量和表达式

在MMBasic命令名称、函数名称、标签、变量名称、文件名称等中不区分大小写，因此 "Run" 和 "RU
N" 是等价的，而 "dOO" 和 "Doo" 指的是同一个变量。

变量

变量可以以字母字符或下划线开头，并且可以包含任何字母或数字字符、句点(.) 和下划线(_)。它们的长度可以达到31个字符。

变量名称或标签不得与函数或以下关键字相同：THEN、ELSE、GOTO、GOSUB、TO、STEP、FOR、WHILE、UNTIL、LOAD、MOD、NOT、AND、OR、XOR、AS。例如，step = 5 是非法的，因为 STEP 是一个关键字。

MMBasic 支持三种类型的变量：

1. 双精度浮点数。

这些可以存储带小数点和分数的数字（例如 45.386），但是当使用超过 14 位的精度时，它们会失去准确性。浮点变量通过在变量名称后添加后缀 '! ' 来指定（例如 i!、nbr! 等）。当创建变量时，如果没有后缀，它们也是默认的（例如 i、nbr 等）。

2. 64 位有符号整数。

这些可以存储最多 19 位小数的正数或负数而不失去准确性，但它们不能存储分数（即小数点后的部分）。这些通过在变量名称后添加后缀 '%' 来指定。例如，i%、nbr% 等。

3. 一个字符串。

字符串将存储一系列字符（例如 "汤姆"）。字符串中的每个字符都作为一个八位数字存储，因此可以具有 0 到 255 的十进制值。字符串变量名称以 ' \$' 符号结束（例如 name\$、s\$ 等）。字符串可以长达 255 个字符。

请注意，使用相同的变量名称但不同类型是非法的。例如，在同一程序中使用 nbr! 和 nbr% 会导致错误。

大多数程序使用浮点变量进行算术运算，因为这些变量可以处理典型情况下使用的数字，并且在处理除法和分数时比整数更直观。因此，如果您不在意细节，请始终使用浮点数。

常量

数字常量可以以数字字符 (0-9) 开始，表示十进制常量，以 &H 开头表示十六进制常量，以 &O 开头表示八进制常量，或以 &B 开头表示二进制常量。例如 &B1000 与十进制常量 8 是相同的。以 &H、&O 或 &B 开头的常量始终被视为 64 位无符号整数常量。

十进制常量可以前面加上负号 (-) 或正号 (+)，并且可以以 'E' 后跟指数数字来表示指数表示法。例如 1.6E+4 与 16000 是相同的。

当使用常量数字时，如果没有使用小数点或指数，则假定它是一个整数。例如，1234 将被解释为整数，而 1234.0 将被解释为浮点数。

字符串常量必须用双引号 ("") 括起来。例如："你好，世界"。

选项 默认

变量可以不带后缀（即 !、% 或 \$）使用，在这种情况下，MMBasic 将使用默认的浮点类型。例如，以下代码将创建一个浮点变量：

```
Nbr = 1234
```

然而，默认值可以通过OPTION DEFAULT命令进行更改。例如，OPTION DEFAULT INTEGER将指定所有没有特定类型的变量为整数。因此，以下代码将创建一个整数变量：

```
OPTION DEFAULT INTEGER
Nbr = 1234
```

The default can be set to FLOAT (which is the default when a program is run), INTEGER, STRING or NONE. In the latter all variables must be specifically typed otherwise an error will occur.

The OPTION DEFAULT command can be placed anywhere in the program and changed at any time but good practice dictates that if it is used it should be placed at the start of the program and left unchanged.

OPTION EXPLICIT

By default MMBasic will automatically create a variable when it is first referenced. So, `Nbr = 1234` will create the variable and set it to the number 1234 at the same time. This is convenient for short and quick programs but it can lead to subtle and difficult to find bugs in large programs. For example, in the third line of this fragment the variable `Nbr` has been misspelt as `Nbrs`. As a consequence the variable `Nbrs` would be created with a value of zero and the value of `Total` would be wrong.

```
Nbr = 1234  
Incr = 2  
Total = Nbrs + Incr
```

The OPTION EXPLICIT command tells MMBasic to not automatically create variables. Instead they must be explicitly defined using the DIM, LOCAL or STATIC commands (see below) before they are used. The use of this command is recommended to support good programming practice. If it is used it should be placed at the start of the program before any variables are used.

DIM and LOCAL

The DIM and LOCAL commands can be used to define a variable and set its type and are mandatory when the OPTION EXPLICIT command is used.

The DIM command will create a global variable that can be seen and used throughout the program including inside subroutines and functions. However, if you require the definition to be visible only within a subroutine or function, you should use the LOCAL command at the start of the subroutine or function. LOCAL has exactly the same syntax as DIM.

If LOCAL is used to specify a variable with the same name as a global variable then the global variable will be hidden to the subroutine or function and any references to the variable will only refer to the variable defined by the LOCAL command. Any variable created by LOCAL will vanish when the program leaves the subroutine. At its simplest level DIM and LOCAL can be used to define one or more variables based on their type suffix or the OPTION DEFAULT in force at the time. For example:

```
DIM nbr%, s$
```

But it can also be used to define one or more variables with a specific type when the type suffix is not used:

```
DIM INTEGER nbr, nbr2, nbr3, etc
```

In this case `nbr`, `nbr2`, `nbr3`, etc are all created as integers. When you use the variable within a program you do not need to specify the type suffix. For example, `MyStr` in the following works perfectly as a string variable:

```
DIM STRING MyStr  
MyStr = "Hello"
```

The DIM and LOCAL commands will also accept the Microsoft practice of specifying the variable's type after the variable with the keyword "AS". For example:

```
DIM nbr AS INTEGER, s AS STRING
```

In this case the type of each variable is set individually (not as a group as when the type is placed before the list of variables).

The variables can also be initialised while being defined. For example:

```
DIM INTEGER a = 5, b = 4, c = 3  
DIM s$ = "World", i% = &H8FF8F  
DIM msg AS STRING = "Hello" + " " + s$
```

The value used to initialise the variable can be an expression including user defined functions.

The DIM or LOCAL commands are also used to define an array and all the rules listed above apply when defining an array. For example, you can use:

```
DIM INTEGER nbr(10), nbr2, nbr3(5,8)
```

默认值可以设置为FLOAT（这是运行程序时的默认值）、INTEGER、STRING或NONE。在后者中，所有变量必须明确指定类型，否则将发生错误。

OPTION DEFAULT命令可以放置在程序的任何位置，并随时更改，但良好的实践是，如果使用它，应该将其放在程序的开头并保持不变。

选项 显式

默认情况下，MMBasic将在第一次引用变量时自动创建该变量。因此，Nbr = 1234将同时创建变量并将其设置为数字1234。这对于短小快速的程序非常方便，但在大型程序中可能会导致微妙且难以发现的错误。例如，在这个片段的第三行中，变量Nbr被错误拼写为Nbrs。因此，变量Nbrs将被创建为零值，而Total的值将是错误的。

```
Nbr = 1234  
Incr = 2  
Total = Nbrs + Incr
```

OPTION EXPLICIT命令告诉MMBasic不要自动创建变量。相反，它们必须在使用之前通过DIM、LOCAL或STATIC命令（见下文）显式定义。建议使用此命令以支持良好的编程实践。如果使用此命令，应将其放在程序的开头，在使用任何变量之前。

DIM和LOCAL

DIM和LOCAL命令可用于定义变量并设置其类型，当使用OPTION EXPLICIT命令时，这是强制性的。

DIM命令将创建一个全局变量，该变量可以在整个程序中被看到和使用，包括在子程序和函数内部。然而，如果您希望定义仅在子程序或函数内可见，则应在子程序或函数的开头使用LOCAL命令。LOCAL的语法与DIM完全相同。

如果使用LOCAL来指定一个与全局变量同名的变量，则全局变量将在子程序或函数中被隐藏，任何对该变量的引用将仅指向由LOCAL命令定义的变量。由LOCAL创建的任何变量将在程序离开子程序时消失。

在最简单的层面上，DIM和LOCAL可以根据它们的类型后缀或当时生效的OPTION DEFAULT来定义一个或多个变量。例如：

```
DIM nbr%, s $
```

但它也可以在未使用类型后缀时定义一个或多个具有特定类型的变量：

```
DIM INTEGER nbr, nbr2, nbr3 等
```

在这种情况下，nbr、nbr2、nbr3等都被创建为整数。当你在程序中使用变量时，不需要指定类型后缀。例如，MyStr在下面的示例中作为字符串变量完美工作：

```
DIM STRING MyStr  
MyStr = "你好"
```

DIM和LOCAL命令也接受微软的做法，即在变量后面使用关键字"AS"来指定变量的类型。例如：

```
DIM nbr AS INTEGER, s AS STRING
```

在这种情况下，每个变量的类型是单独设置的（而不是像将类型放在变量列表前面时那样作为一组设置）。

变量在定义时也可以初始化。例如：

```
DIM INTEGER a = 5, b = 4, c = 3  
DIM s$ = "世界", i% = &H8FF8F  
DIM msg AS STRING = "你好" + " " + s$
```

用于初始化变量的值可以是包括用户定义函数在内的表达式。

DIM或LOCAL命令也用于定义数组，并且在定义数组时适用上述所有规则。例如，你可以使用：

```
DIM INTEGER nbr(10), nbr2, nbr3(5,8)
```

When initialising an array the values are listed as comma separated values with the whole list surrounded by brackets. For example:

```
DIM INTEGER nbr(5) = (11, 12, 13, 14, 15, 16)
```

or

```
DIM days(7) AS STRING = ("", "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
```

STATIC

Inside a subroutine or function it is sometimes useful to create a variable which is only visible within the subroutine or function (like a LOCAL variable) but retains its value between calls to the subroutine or function.

You can do this by using the STATIC command. STATIC can only be used inside a subroutine or function and uses the same syntax as LOCAL and DIM. The difference is that its value will be retained between calls to the subroutine or function (i.e. it will not be initialised on the second and subsequent calls).

For example, if you had the following subroutine and repeatedly called it, the first call would print 5, the second 6, the third 7 and so on.

```
SUB Foo
    STATIC var = 5
    PRINT var
    var = var + 1
END SUB
```

Note that the initialisation of the static variable to 5 (as in the above example) will only take effect on the first call to the subroutine. On subsequent calls the initialisation will be ignored as the variable had already been created on the first call.

As with DIM and LOCAL the variables created with STATIC can be float, integers or strings and arrays of these with or without initialisation. The length of the variable name created by STATIC and the length of the subroutine or function name added together cannot exceed 31 characters.

CONST

Often it is useful to define an identifier that represents a value without the risk of the value being accidentally changed - which can happen if variables were used for this purpose (this practice encourages another class of difficult to find bugs).

Using the CONST command you can create an identifier that acts like a variable but is set to a value that cannot be changed. For example:

```
CONST InputVoltagePin = 31
CONST MaxValue = 2.4
```

The identifiers can then be used in a program where they make more sense to the casual reader than simple numbers. For example:

```
IF PIN(InputVoltagePin) > MaxValue THEN SoundAlarm
```

A number of constants can be created on the one line:

```
CONST InputVoltagePin = 31, MaxValue = 2.4, MinValue = 1.5
```

The value used to initialise the constant is evaluated when the constant is created and can be an expression including user defined functions.

The type of the constant is derived from the value assigned to it; so for example, MaxValue above will be a floating point constant because 2.4 is a floating point number. The type of a constant can also be explicitly set by using a type suffix (i.e. !, % or \$) but it must agree with its assigned value.

Special Characters in Strings

Special, non-printable characters can be inserted in string constants using the backslash (ie, \) as an escape symbol. To enable this facility the command OPTION ESCAPE must be placed at the start of the program.

These are the valid escape sequences:

Hex Value	Description
\a	07 Alert (Beep, Bell)
\b	08 Backspace
\e	1B Escape character

在初始化数组时，值以逗号分隔，并且整个列表用括号括起来。例如：

```
DIM INTEGER nbr (5) = (11, 12, 13, 14, 15, 16)
```

或

```
DIM days(7) AS STRING = (" ", "日", "一", "二", "三", "四", "五", "六")
```

静态

在子程序或函数内部，有时创建一个仅在子程序或函数内可见的变量（类似于局部变量）是有用的，但它在调用子程序或函数之间保留其值。

你可以通过使用 STATIC 命令来实现这一点。STATIC 只能在子程序或函数内部使用，并且使用与 LOCAL 和 DIM 相同的语法。不同之处在于，它的值将在对子程序或函数的调用之间保留（即在第二次及后续调用时不会被初始化）。

例如，如果你有以下子程序并重复调用它，第一次调用将打印5，第二次调用打印6，第三次调用打印7，依此类推。

```
SUB Foo
    STATIC var = 5
    PRINT var
    var = var + 1
END SUB
```

请注意，静态变量初始化为5（如上例所示）只会在第一次调用子程序时生效。在后续调用中，初始化将被忽略，因为该变量在第一次调用时已经被创建。

与DIM和LOCAL一样，使用STATIC创建的变量可以是浮点数、整数或字符串，以及这些类型的数组，可以选择是否初始化。由STATIC创建的变量名称的长度与子程序或函数名称的长度相加不得超过31个字符。

常量

通常，定义一个表示值的标识符是有用的，这样可以避免值被意外更改的风险——如果使用变量来实现这一目的，就可能发生这种情况（这种做法会导致另一类难以发现的错误）。

使用 CONST 命令可以创建一个标识符，它的作用类似于变量，但被设置为一个不可更改的值。例如：

```
CONST InputVoltagePin = 31
CONST MaxValue = 2.4
```

这些标识符可以在程序中使用，对于普通读者来说，它们比简单的数字更有意义。例如：

如果 PIN(InputVoltagePin) > MaxValue 那么 SoundAlarm

可以在一行中创建多个常量：

```
CONST InputVoltagePin = 31, MaxValue = 2.4, MinValue = 1.5
```

用于初始化常量的值在创建常量时被求值，并且可以是包括用户定义函数的表达式。

常量的类型是根据分配给它的值推导出来的；例如，MaxValue将是一个浮点常量，因为2.4是一个浮点数。常量的类型也可以通过使用类型后缀（即!，% 或 \$）显式设置，但它必须与其分配的值一致。

字符串中的特殊字符

可以使用反斜杠（即\）作为转义符在字符串常量中插入特殊的不可打印字符。要启用此功能，命令 OPTION ESCAPE 必须放在程序的开头。

以下是有效的转义序列：

十六进制值描述

\a	07	警报（哔声，铃声）
\b	08	退格
\e	1B	转义字符

\f	0C	Formfeed Page Break
\n	0A	Newline (Line Feed)
\r	0D	Carriage Return
\q	22	Quote symbol
\t	09	Horizontal Tab
\v	0B	Vertical Tab
\\\	5C	Backslash
\nnn	any	The byte whose value is given by nnn interpreted as a decimal number
\&hh	any	The byte whose value is given by hh... interpreted as a hexadecimal number

For example, the following will print the words Hello and World on separate lines:

```
OPTION ESCAPE
PRINT "Hello\r\nWorld"
```

Expressions and Operators

MMBasic will evaluate a mathematical expression using the standard mathematical rules. For example, multiplication and division are performed first followed by addition and subtraction. These are called the rules of precedence and are detailed below.

This means that $2 + 3 * 6$ will resolve to 20, so will $5 * 4$ and also $10 + 4 * 3 - 2$.

If you want to force the interpreter to evaluate parts of the expression first you can surround that part of the expression with brackets. For example, $(10 + 4) * (3 - 2)$ will resolve to 14 not 20 as would have been the case if the brackets were not used. Using brackets does not appreciably slow down the program so you should use them liberally if there is a chance that MMBasic will misinterpret your intention.

The following operators, in order of precedence, are implemented in MMBasic. Operators that are on the same level (for example + and -) are processed with a left to right precedence as they occur on the program line.

Arithmetic operators:

^	Exponentiation (e.g. b^n means b^n)
* / \ MOD	Multiplication, division, integer division and modulus (remainder)
+ -	Addition and subtraction

Shift operators:

$x << y$ $x >> y$	These operate in a special way. $<<$ means that the value returned will be the value of x shifted by y bits to the left while $>>$ means the same only right shifted. They are integer functions and any bits shifted off are discarded and any bits introduced are set to zero.
-------------------	--

Logical operators:

NOT INV	invert the logical value on the right (e.g. NOT a=b is $a <> b$) or bitwise inversion of the value on the right (e.g. a = INV b)
\diamond < > \leq \leq \geq \Rightarrow	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	equality
AND OR XOR	Conjunction, disjunction, exclusive or

For Microsoft compatibility the operators AND, OR and XOR are integer bitwise operators. For example, PRINT (3 AND 6) will output the number 2. Because these operators can act as both logical operators (for example, IF a=5 AND b=8 THEN ...) and as bitwise operators (e.g. $y\% = x\% \text{ AND } \&B1010$) the interpreter will be confused if they are mixed in the same expression. So, always evaluate logical and bitwise expressions in separate expressions.

The other logical operations result in the integer 0 (zero) for false and 1 for true. For example the statement PRINT 4 \geq 5 will print the number zero on the output and the expression A = 3 $>$ 2 will store +1 in A.

\f	0C	换页符
\n	0A	换行符 (行进)
\r	0D	回车符
\q	22	引号符号
\t	09	水平制表符
\v	0B	垂直制表符
\\\	5C	反斜杠
\nnn	任何	值为nnn的字节，按十进制数解释
\&hh	任何	值为hh…的字节，按十六进制数解释

例如，以下代码将在不同的行上打印单词 Hello 和 World：

```
OPTION ESCAPE
PRINT "Hello \r\nWorld"
```

表达式和运算符

MMBasic 将使用标准数学规则求值数学表达式。例如，乘法和除法优先执行，其次是加法和减法。这些被称为优先级规则，详细信息如下。

这意味着 $2 + 3 * 6$ 将计算为 20， $5 * 4$ 也是如此， $10 + 4 * 3 - 2$ 也会如此。

如果您想强制解释器首先求值表达式的某些部分，可以用括号将该部分括起来。例如， $(10 + 4) * (3 - 2)$ 将计算为 14，而不是 20，如果没有使用括号的话。使用括号不会显著减慢程序的运行速度，因此如果有可能导致 MMBasic 误解您的意图，您应该自由使用它们。

以下运算符按优先级顺序在 MMBasic 中实现。处于同一层级的运算符（例如 + 和 -）会按照它们在程序行中出现的顺序从左到右处理。

算术运算符：

^	指数运算（例如 b^n 表示 b^n ）
* / \ MOD	乘法、除法、整数除法和模（余数）
+ -	加法和减法

移位运算符：

$x << y$ $x >> y$	这些运算以特殊方式操作。 $<<$ 表示返回的值是 x 向左移 y 位的值，而 $>>$ 则表示向右移。它们是整数函数，任何移出的位都会被丢弃，而引入的位会被设置为零。
-------------------	--

逻辑运算符：

NOT INV	反转右侧的逻辑值（例如 NOT a=b 表示 $a <> b$ ）或右侧值的按位反转（例如 $a = INV b$ ）
\diamond < > \leq $=\leq$ \geq $=>$	不等式、小于、大于、小于或等于、小于或等于（替代版本）、大于或等于、大于或等于（替代版本）
=	等式
AND OR XOR	连接词、析取词、异或

为了与微软兼容，运算符 AND、OR 和 XOR 是整数位运算符。例如，PRINT (3 AND 6) 将输出数字 2。因为这些运算符可以作为逻辑运算符（例如，如果 $a=5$ AND $b=8$ THEN ...）和位运算符（例如 $y\% = x\% AND \&B1010$ ）使用，如果它们在同一表达式中混合使用，解释器会感到困惑。因此，总是将逻辑表达式和位表达式在单独的表达式中求值。

其他逻辑运算的结果为整数 0（假）和 1（真）。例如，语句 PRINT $4 \geq 5$ 将在输出中打印数字零，而表达式 $A = 3 > 2$ 将在 A 中存储 +1。

The NOT operator will invert the logical value on its right (it is not a bitwise invert) while the INV operator will perform a bitwise invert. Both of these have the highest precedence so they will bind tightly to the next value. For normal use of NOT or INV the expression to be operated on should be placed in brackets. Eg:

IF NOT (A = 3 OR A = 8) THEN ...

String operators:

+	Join two strings
<> < > <= == >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	Equality

String comparisons respect case. For example "A" is greater than "a".

Mixing Floating Point and Integers

MMBasic automatically handles conversion of numbers between floating point and integers. If an operation mixes both floating point and integers (e.g. PRINT A% + B!) the integer will be converted to a floating point number first, then the operation performed and a floating point number returned. If both sides of the operator are integers then an integer operation will be performed and an integer returned.

The one exception is the normal division ("/") which will always convert both sides of the expression to a floating point number and then returns a floating point number. For integer division you should use the integer division operator "\".

MMBasic functions will return a float or integer depending on their characteristics. For example, PIN() will return an integer when the pin is configured as a digital input but a float when configured as an analog input.

If necessary you can convert a float to an integer with the INT() function. It is not necessary to specifically convert an integer to a float but if it was needed the integer value could be assigned to a floating point variable and it will be automatically converted in the assignment.

64-bit Unsigned Integers

MMBasic on the PicoMite supports 64-bit signed integers. This means that there are 63 bits for holding the number and one bit (the most significant bit) which is used to indicate the sign (positive or negative). However it is possible to use full 64-bit unsigned numbers as long as you do not do any arithmetic on the numbers.

64-bit unsigned numbers can be created using the &H, &O or &B prefixes to a number and these numbers can be stored in an integer variable. You then have a limited range of operations that you can perform on these. They are << (shift left), >> (shift right), AND (bitwise and), OR (bitwise or), XOR (bitwise exclusive or), INV (bitwise inversion), = (equal to) and <> (not equal to). Arithmetic operators such as division or addition may be confused by a 64-bit unsigned number and could return nonsense results.

To display 64-bit unsigned numbers you should use the HEX\$(), OCT\$() or BIN\$() functions.

For example, the following 64-bit unsigned operation will return the expected results:

```
X% = &HFFFF0000FFFF0044
Y% = &H800FFFFFFFFFFFFF
X% = X% AND Y%
PRINT HEX$(X%, 16)
```

Will display "800F0000FFFF0044"

NOT运算符将反转其右侧的逻辑值（它不是按位反转），而INV运算符将执行按位反转。这两者的优先级最高，因此它们将紧密绑定到下一个值。对于NOT或INV的正常使用，操作的表达式应放在括号中。例如：

如果 NOT (A = 3 或 A = 8) 那么 ...

字符串操作符：

+	连接两个字符串
<> < > <= =< >= =>	不等式、小于、大于、小于或等于、小于或等于（替代版本） ）、大于或等于、大于或等于（替代版本）
=	相等

字符串比较区分大小写。例如 "A" 大于 "a"。

混合浮点数和整数

MMBasic 自动处理浮点数和整数之间的转换。如果一个操作混合了浮点数和整数（例如 PRINT A% + B!），整数将首先被转换为浮点数，然后执行操作并返回浮点数。如果运算符两边都是整数，则将执行整数操作并返回整数。

唯一的例外是普通除法（"/"），它将始终将表达式两边的值转换为浮点数，然后返回浮点数。对于整数除法，您应该使用整数除法运算符 "\"。

MMBasic 函数将根据其特性返回浮点数或整数。例如，当引脚配置为数字输入时，PIN() 将返回整数，但当配置为模拟输入时，将返回浮点数。

如果需要，可以使用 INT() 函数将浮点数转换为整数。不需要特别将整数转换为浮点数，但如果需要，可以将整数值赋给浮点变量，赋值时会自动转换。

64位无符号整数

PicoMite 上的 MMBasic 支持 64 位有符号整数。这意味着有 63 位用于存储数字，还有一位（最高有效位）用于指示符号（正或负）。然而，只要不对这些数字进行任何算术运算，就可以使用完整的 64 位无符号数字。

可以使用 &H、&O 或 &B 前缀来创建 64 位无符号数字，并且这些数字可以存储在整数变量中。然后，您可以对这些数字执行有限范围的操作。

它们是 <<（左移）、>>（右移）、AND（按位与）、OR（按位或）、XOR（按位异或）、INV（按位取反）、=（等于）和 <>（不等于）。像除法或加法这样的算术运算符可能会被 64 位无符号数字混淆，从而返回无意义的结果。

要显示 64 位无符号数字，您应该使用 HEX\$()、OCT\$() 或 BIN\$() 函数。

例如，以下 64 位无符号运算将返回预期的结果：

```
X% = &HFFFF0000FFFF00 44
Y% = &H800FFFFFFFFFFF
X% = X % AND Y%
PRINT HEX$(X%, 16)
```

将显示 "800F0000FFF0044"

Subroutines and Functions

A program defined subroutine or function 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 or function to the language.

Subroutines

A subroutine acts like a command and it can have arguments (sometimes called a parameter list). 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 hidden by the arguments defined for the subroutine.

When calling a subroutine you can supply less than the required number of values and in that case the missing values will be assumed to be either zero or an empty string. You can also leave out a value in the middle of the list and the same will happen. For example:

```
MYSUB 23, , 55
```

Will result in `arg2$` being set to the empty string `" "`.

Rather than using the type suffix (e.g. the \$ in `arg2$`) you can use the suffix AS `<type>` in the definition of the subroutine argument and then the argument will be known as the specified type, even when the suffix is not used. For example:

```
SUB MYSUB arg1, arg2 AS STRING, arg3
    IF arg2 = "Cat" THEN ...
END SUB
```

Inside a subroutine you can define a variable using LOCAL (which has the same syntax as DIM). This variable will only exist within the subroutine and will vanish when the subroutine exits.

Functions

Functions are similar to subroutines with the main difference being that the function is used to return a value in an expression. The rules for the argument list in a function are similar to subroutines. The only difference is that brackets are required around the argument list when you are calling a function, even if there are no arguments (they are optional when calling a subroutine).

To return a value from the function you assign a value to the function's name within the function. If the function's name is terminated with a \$, a % or a ! the function will return that type, otherwise it will return whatever the OPTION DEFAULT is set to. You can also specify the type of the function by adding AS `<type>` to the end of the function definition.

For example:

```
FUNCTION Fahrenheit(C) AS FLOAT
    Fahrenheit = C * 1.8 + 32
END FUNCTION
```

Passing Arguments by Reference

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

子程序和函数

程序定义的子程序或函数只是包含在模块中的一段编程代码，可以在程序的任何地方调用。这就像您向语言添加了自己的命令或函数一样。

子例程

子程序的作用类似于命令，并且可以有参数（有时称为参数列表）。在子程序的定义中，它们看起来像这样：

```
SUB MYSUB arg1, arg2$, arg3
    <语句>
    <语句>
END SUB
```

当你调用子程序时，可以为参数赋值。例如：

```
MYSUB 23, "猫", 55
```

在子程序内部 arg1 的值将是 23，arg2\$ 的值将是 "猫"，依此类推。参数的作用类似于普通变量，但它们仅存在于子程序中，并将在子程序结束时消失。在主程序中可以有相同名称的变量，它们将被子程序定义的参数隐藏。

调用子程序时，可以提供少于所需数量的值，在这种情况下，缺失的值将被假定为零或空字符串。你也可以在列表中间省略一个值，结果也是一样。例如：

```
MYSUB 23, , 55
```

将导致 arg2\$ 被设置为空字符串 ""。

与其使用类型后缀（例如 arg2\$ 中的 \$），不如在子程序参数的定义中使用后缀 AS <type>，这样即使不使用后缀，该参数也将被视为指定的类型。例如：

```
SUB MYSUB arg1, arg2 AS STRING, arg3
    如果 arg2 = "猫" 那么 ...
    结束子程序
```

在子程序内部，你可以使用 LOCAL 定义一个变量（其语法与 DIM 相同）。这个变量只在子程序内存中，子程序退出时将消失。

函数

函数与子程序类似，主要区别在于函数用于在表达式中返回一个值。函数的参数列表规则与子程序类似。唯一的区别是，当你调用一个函数时，参数列表必须用括号括起来，即使没有参数（调用子程序时参数是可选的）。

要从函数返回一个值，你需要在函数内部将一个值赋给函数的名称。如果函数的名称以 \$、% 或 ! 结尾，函数将返回该类型，否则将返回 OPTION DEFAULT 设置的类型。你还可以通过在函数定义的末尾添加 AS <类型> 来指定函数的类型。

例如：

```
FUNCTION Fahrenheit(C) AS FLOAT
    Fahrenheit = C * 1.8 + 32
    结束函数
```

通过引用传递参数

如果在调用子程序或函数时使用普通变量（即，不是表达式）作为值，则子程序/函数中的参数将指向调用中使用的变量，对参数的任何更改也将反映在提供的变量上。这称为通过引用传递参数。

For example, you might define a subroutine to swap two values, as follows:

```
SUB Swap a, b
    LOCAL t
    t = a
    a = b
    b = t
END SUB
```

In your calling program you would use variables for both arguments:

```
Swap nbr1, nbr2
```

And the result will be that the values of nbr1 and nbr2 will be swapped.

For this to work the type of the variable passed (e.g. nbr1) and the defined argument (e.g. a) must be the same (in the above example both default to float).

Unless you need to return a value via the argument you should not use an argument as a general purpose variable inside a subroutine or function. This is because another user of your routine may unwittingly use a variable in their call and that variable could be "magically" changed by your routine. It is much safer to assign the argument to a local variable and manipulate that instead.

Passing Arrays

Single elements of an array can be passed to a subroutine or function and they will be treated the same as a normal variable. For example, this is a valid way of calling the Swap subroutine (discussed above):

```
Swap dat(i), dat(i + 1)
```

This type of construct is often used in sorting arrays.

You can also pass one or more complete arrays to a subroutine or function by specifying the array with empty brackets instead of the normal dimensions. For example, a(). In the subroutine or function definition the associated parameter must also be specified with empty brackets. The type (i.e., float, integer or string) of the argument supplied and the parameter in the definition must be the same.

In the subroutine or function the array will inherit the dimensions of the array passed and these must be respected when indexing into the array. If required the dimensions of the array could be passed as additional arguments to the subroutine or function so it could correctly manipulate the array. The array is passed by reference which means that any changes made to the array within the subroutine or function will also apply to the supplied array.

For example, when the following is run the words "Hello World" will be printed out:

```
DIM MyStr$(5, 5)
MyStr$(4, 4) = "Hello" : MyStr$(4, 5) = "World"
Concat MyStr$()
PRINT MyStr$(0, 0)

SUB Concat arg$()
    arg$(0,0) = arg$(4, 4) + " " + arg$(4, 5)
END SUB
```

Early Exit

There can be only one END SUB or END FUNCTION for each definition of a subroutine or function. To exit early from a subroutine (i.e., before the END SUB command has been reached) you can use the EXIT SUB command. This has the same effect as if the program reached the END SUB statement. Similarly you can use EXIT FUNCTION to exit early from a function.

Recursion

Recursion is where a subroutine or function calls itself. You can do recursion in MMBasic but there are a number of issues (these are a direct consequence of the limitations of microcontrollers and the BASIC language):

- There is a fixed limit to the depth of recursion. In the PicoMite this is 50 levels.
- If you have many arguments to the subroutine or function and many LOCAL variables (especially strings) you could easily run out of memory before reaching the 50 level limit.
- Any FOR...NEXT loops and DO...LOOPs will be corrupted if the subroutine or function is recursively called from within these loops.

例如，您可以定义一个子程序来交换两个值，如下所示：

```
SUB Swap a, b
    局部 t
    t = a
    a = b
    b = t
    结束子程序
```

在您的调用程序中，您将为两个参数使用变量：

```
Swap nbr1, nbr2
```

结果将是 nbr1 和 nbr2 的值被交换。

为了使其正常工作，传递的变量类型（例如 nbr1）和定义的参数（例如 a）必须相同
(在上述示例中，两者默认都是浮点数)。

除非你需要通过参数返回一个值，否则不应在子程序或函数内部将参数用作通用变量。这是因为你的例程的另一个用户可能在调用中不知情地使用了一个变量，而该变量可能会被你的例程“神奇地”改变。将参数赋值给一个局部变量并操作该变量要安全得多。

传递数组

数组的单个元素可以传递给子程序或函数，它们将被视为普通变量。例如，这是一种有效的调用 Swap 子程序（如上所述）的方法：

```
Swap dat(i), dat(i + 1)
```

这种构造通常用于排序数组。

你还可以通过指定带有空括号的数组来传递一个或多个完整的数组，而不是使用正常的尺寸。例如，`a()`。在子程序或函数定义中，相关参数也必须用空括号指定。提供的参数和定义中的参数类型（即浮点数、整数或字符串）必须相同。

在子程序或函数中，数组将继承传递的数组的尺寸，并且在索引数组时必须遵守这些尺寸。如果需要，数组的尺寸可以作为额外参数传递给子程序或函数，以便它能够正确操作数组。数组是通过引用传递的，这意味着在子程序或函数中对数组所做的任何更改也将适用于提供的数组。

例如，当运行以下代码时，将打印出 "Hello World"：

```
DIM MyStr$(5, 5)
MyStr$(4, 4) = "Hello" : MyStr$(4, 5) = "World"
Concat MyStr$()
PRINT MyStr$(0, 0)

SUB Concat arg$()
    arg$(0, 0) = arg$(4, 4) + " " + arg$(4, 5)
END SUB
```

提前退出

每个子程序或函数的定义只能有一个 END SUB 或 END FUNCTION。要提前退出子程序（即，在达到 END SUB 命令之前），可以使用 EXIT SUB 命令。这与程序达到 END SUB 语句的效果相同。同样，你可以使用 EXIT FUNCTION 提前退出函数。

递归

递归是指子程序或函数调用自身。你可以在 MMBasic 中进行递归，但存在一些问题（这些是微控制器和 BASIC 语言限制的直接结果）：

- 递归的深度有固定限制。在 PicoMite 中，这个限制是 50 层。
- 如果你有许多参数传递给子程序或函数，并且有许多局部变量（尤其是字符串），你可能会在达到 50 层限制之前轻易耗尽内存。
- 如果子程序或函数在这些循环中被递归调用，任何 FOR...NEXT 循环和 DO...LOOP 都会被破坏。

Examples

There is often the need for a special command or function to be implemented in MMBasic but in many cases these can be constructed using an ordinary subroutine or function which will then act exactly the same as a built in command or function.

For example, sometimes there is a requirement for a TRIM function which will trim specified characters from the start and end of a string. The following provides an example of how to construct such a simple function in MMBasic.

The first argument to the function is the string to be trimmed and the second is a string containing the characters to trim from the first string. RTrim\$() will trim the specified characters from the end of the string, LTrim\$() from the beginning and Trim\$() from both ends.

```
' trim any characters in c$ from the start and end of s$  
Function Trim$(s$, c$)  
    Trim$ = RTrim$(LTrim$(s$, c$), c$)  
End Function  
  
' trim any characters in c$ from the end of s$  
Function RTrim$(s$, c$)  
    RTrim$ = s$  
    Do While Instr(c$, Right$(RTrim$, 1))  
        RTrim$ = Mid$(RTrim$, 1, Len(RTrim$) - 1)  
    Loop  
End Function  
  
' trim any characters in c$ from the start of s$  
Function LTrim$(s$, c$)  
    LTrim$ = s$  
    Do While Instr(c$, Left$(LTrim$, 1))  
        LTrim$ = Mid$(LTrim$, 2)  
    Loop  
End Function
```

As an example of using these functions:

```
S$ = " ****23.56700 "  
PRINT Trim$(s$, " ")
```

Will give "****23.56700"

```
PRINT Trim$(s$, " *0")
```

Will give "23.567"

```
PRINT LTrim$(s$, " *0")
```

Will give "23.56700"

示例

在MMBasic中，通常需要实现一个特殊的命令或函数，但在许多情况下，这些可以通过普通的子程序或函数构造，从而完全像内置的命令或函数一样工作。

例如，有时需要一个TRIM函数，该函数将修剪字符串开头和结尾的指定字符。以下提供了如何在MMBasic中构造这样一个简单函数的示例。

该函数的第一个参数是要修剪的字符串，第二个参数是一个包含要从第一个字符串中修剪的字符的字符串。RTrim\$()将从字符串的末尾修剪指定的字符，LTrim\$()从开头修剪，Trim\$()则从两端修剪。

```
' 从s$的开头和结尾修剪c$中的任何字符
函数 Trim$(s$, c$)
    Trim$ = RTrim$(LTrim$(s$, c$), c$)
结束函数

' 从 s$ 的末尾修剪 c$ 中的任何字符
函数 RTrim$(s$, c$)
    RTrim$ = s$
    当 Instr(c$, Right$(RTrim$, 1)) 时
        RTrim$ = Mid$(RTrim$, 1, Len(RTrim$) - 1)
    循环
结束函数

' 从 s$ 的开头修剪 c$ 中的任何字符
函数 LTrim$(s$, c$)
    LTrim$ = s$
    当 Instr(c$, Left$(LTrim$, 1)) 时
        LTrim$ = Mid$(LTrim$, 2)
    循环
结束函数
```

作为使用这些函数的示例：

```
S$ = " ****23.56700   "
打印 Trim$(s$, " ")
```

将得到 "****23.56700"

```
打印 Trim$(s$, " *0")
```

将得到 "23.567"

```
打印 LTrim$(s$, " *0")
```

将得到 "23.56700"

Using the I/O pins

The Raspberry Pi Pico has 26 input/output pins which can be controlled from within the BASIC program with 3 of these supporting a high speed ADC (Analog to Digital Converter).

An I/O pin is referred to by its pin number and this can be the number (e.g., 2) or its GP number (e.g., GP1).

Digital Inputs

A digital input is the simplest type of input configuration. If the input voltage is higher than 2.5V the logic level will be true (numeric value of 1) and anything below 0.65V will be false (numeric value of 0). The inputs use a Schmitt trigger input so anything in between these levels will retain the previous logic level. All pins are limited to a maximum voltage of 3.6V. This means that resistor divider will be required if they are used with input voltages greater than that.

In your BASIC program you would set the input as a digital input and use the PIN() function to get its level. For example:

```
SETPIN GP4, DIN  
IF PIN(GP4) = 1 THEN PRINT "High"
```

The SETPIN command configures pin GP4 as a digital input and the PIN() function will return the value of that pin (the number 1 if the pin is high). The IF command will then execute the command after the THEN statement if the input was high. If the input pin was low the program would just continue with the next line in the program.

The SETPIN command also recognises a couple of options that will connect an internal resistor from the input to either the supply or ground. This is called a "pullup" or "pulldown" resistor and is handy when connecting to a switch as it saves having to install an external resistor to place a voltage across the contacts.

Analog Inputs

Pins marked as ADC can be configured to measure the voltage on the pin. The input range is from zero to 3.3V and the PIN() function will return the voltage. For example:

```
> SETPIN 31, AIN  
> PRINT PIN(31)  
2.345  
>
```

You will need a voltage divider if you want to measure voltages greater than 3.3V. For small voltages you may need an amplifier to bring the input voltage into a reasonable range for measurement.

The measurement uses 3.3V power supply to the CPU as its reference and it is assumed that this is exactly 3.3V. This value can be changed with the OPTION command.

The ADC commands provide an alternate method of recording analog inputs and are intended for high speed recording of many readings into an array.

Counting Inputs

Any four pins can be used as counting inputs to measure frequency, period or just count pulses on the input. The pins used for this function can be configured using the OPTION COUNT command but, if not changed, will default to GP6, GP7, GP8 and GP9.

As an example, the following will print the frequency of the signal on pin GP7:

```
> SETPIN GP7, FIN  
> PRINT PIN(GP7)  
110374  
>
```

In this case the frequency is 110.374 kHz.

By default the gate time is one second which is the length of time that MMBasic will use to count the number of cycles on the input and this means that the reading is updated once a second with a resolution of 1 Hz. By specifying a third argument to the SETPIN command it is possible to specify an alternative gate time between 10ms and 100000 ms. Shorter times will result in the readings being updated more frequently but the value

使用I/O引脚

树莓派 Pico 具有 26 个输入/输出引脚，这些引脚可以在 BASIC 程序中控制，其中 3 个支持高速 ADC（模数转换器）。

输入/输出引脚通过其引脚编号来引用，这可以是数字（例如，2）或其GP编号（例如，GP1）。

数字输入

数字输入是最简单的输入配置类型。如果输入电压高于 2.5V，则逻辑电平为真（数值为 1），而低于 0.65V 则为假（数值为 0）。输入使用施密特触发器输入，因此在这些电平之间的任何内容将保留先前的逻辑电平。所有引脚的最大电压限制为 3.6V。这意味着如果输入电压超过该值，则需要使用电阻分压器。

在您的BASIC程序中，您将输入设置为数字输入，并使用PIN()函数获取其电平。
例如：

```
SETPIN GP4, DIN  
如果 PIN(GP4) = 1 那么 打印 "高"
```

SETPIN 命令将引脚 GP4 配置为数字输入，PIN() 函数将返回该引脚的值（如果引脚为高，则返回数字 1）。如果输入为高，则 IF 命令将执行 THEN 语句后的命令。如果输入引脚为低，程序将继续执行程序中的下一行。

SETPIN 命令还识别几个选项，可以将内部电阻连接到输入的电源或接地。这被称为“上拉”或“下拉”电阻，连接开关时非常方便，因为它省去了安装外部电阻以在触点之间施加电压的需要。

模拟输入

标记为 ADC 的引脚可以配置为测量引脚上的电压。输入范围为零到 3.3V，PIN() 函数将返回电压。例如：

```
> SETPIN 31, AIN  
> 打印 PIN(31)  
2.345  
>
```

如果您想测量大于 3.3V 的电压，则需要一个电压分压器。对于小电压，您可能需要一个放大器将输入电压提升到合理的测量范围。

测量使用 3.3V 电源供电给中央处理器作为参考，并假设这个值正好是 3.3V。这个值可以通过 OPTION 命令进行更改。

ADC 命令提供了一种记录模拟输入的替代方法，旨在以高速将多个读数记录到数组中。

计数输入

任何四个引脚都可以用作计数输入，以测量频率、周期或仅仅计数输入上的脉冲。

用于此功能的引脚可以通过 OPTION COUNT 命令进行配置，但如果未更改，将默认为 GP6、GP7、GP8 和 GP9。

作为示例，以下代码将打印引脚 GP7 上信号的频率：

```
> SETPIN GP7, FIN  
> 打印 PIN(GP7)  
110374  
>
```

在这种情况下，频率为 110.374 kHz。

默认情况下，门时间为一秒，这意味着 MMBasic 将使用这段时间来计数输入的周期，因此读取值每秒更新一次，分辨率为 1Hz。通过为 SETPIN 命令指定第三个参数，可以指定一个介于 10ms 和 1000ms 之间的替代门时间。较短的时间将导致读取值更新得更频繁，但该值

returned will have a lower resolution. The PIN() function will always scale the returned number as the frequency in Hz regardless of the gate time used.

For example, the following will set the gate time to 10ms with a corresponding loss of resolution:

```
> SETPIN GP7, FIN, 10
> PRINT PIN(GP7)
110300
>
```

For accurate measurement of signals less than 10Hz it is generally better to measure the period of the signal. When set to this mode the PicoMite will measure the number of milliseconds between sequential rising edges of the input signal. The value is updated on the low to high transition so if your signal has a period of (say) 100 seconds you should be prepared to wait that amount of time before the PIN() function will return an updated value.

The count pins can also count the number of pulses on their input. When a pin is configured as a counter (for example, SETPIN 7, CIN) the counter will be reset to zero and PicoMite will then count every transition from a low to high voltage. The counter can be reset to zero again by executing PIN(7) = 0.

Counting inputs are accurate up to about 200KHz at the default processor frequency. A minimum pulse width of about 40nS is needed to trigger the counter

Digital Outputs

All I/O pins can be configured as a digital output using the DOUT parameter to the SETPIN command. For example:

```
SETPIN GP15, DOUT
```

This means that when an output pin is set to logic low it will pull its output to zero and when set high it will pull its output to 3.3V. In MMBasic this is done with the PIN command. For example PIN(GP15) = 0 will set pin GP15 to low while PIN(GP15) = 1 will set it high.

Pulse Width Modulation

The PWM (Pulse Width Modulation) command allows the PicoMite to generate square waves with a program controlled duty cycle. By varying the duty cycle you can generate a program controlled voltage output for use in controlling external devices that require an analog input (power supplies, motor controllers, etc). The PWM outputs are also useful for driving servos and for generating a sound output via a small transducer.

The PWM outputs consists of up to 8 channels (numbered 0 to 7) with each channel having two outputs (A and B). For each channel the frequency can be selected and for each output a different duty cycle can be set.

Up to 16 pins can be configured as PWM outputs using the SETPIN command.

Communications Interfaces (Serial, SPI and I²C)

These are described in the appendices at the rear of this manual. Before these interfaces can be used the pins that are to be used for the relevant signals must be configured using the SETPIN command.

Some devices such as an SD Card, LCD panels, touch, etc also use SPI or I2C interfaces and the pins used for these must similarly be configured using the OPTION SYSTEM command before they can be used.

Interrupts

Interrupts are a handy way of dealing with an event that can occur at an unpredictable time. An example is when the user presses a button. In your program you could insert code after each statement to check to see if the button has been pressed but an interrupt makes for a cleaner and more readable program.

When an interrupt occurs MMBasic will execute a special subroutine and when finished return to the main program. The main program is completely unaware of the interrupt and will carry on as normal.

Any I/O pin that can be used as a digital input can be configured to generate an interrupt using the SETPIN command with up to ten interrupts active at any one time. Interrupts can be set up to occur on a rising or falling digital input signal (or both) and will cause an immediate branch to the specified user defined subroutine. The target can be the same or different for each interrupt. Return from an interrupt is via the END SUB or EXIT SUB commands. Note that no parameters can be passed to the subroutine however within the interrupt calls to other subroutines and functions are allowed.

返回的值将具有较低的分辨率。无论使用何种门时间，PIN() 函数始终会将返回的数字缩放为 Hz 中的频率。

例如，以下代码将门时间设置为 10ms，相应地会损失分辨率：

```
> SETPIN GP7, FIN, 10  
> 打印 PIN(GP7)  
110300  
>
```

对于低于 10Hz 的信号，通常更好地测量信号的周期。当设置为此模式时，PicoMite 将测量输入信号连续上升沿之间的毫秒数。该值在低到高的转换时更新，因此如果您的信号周期为（例如）100 秒，您应该准备等待该时间量，才能使 PIN() 函数返回更新的值。

计数引脚也可以计算其输入上的脉冲数量。当引脚被配置为计数器时（例如，SETPIN 7,CIN），计数器将重置为零，PicoMite 将计算每次从低电压到高电压的过渡。可以通过执行 PIN(7) = 0 再次将计数器重置为零。

计数输入在默认处理器频率下的准确度可达约 200KHz。触发计数器需要约 40nS 的最小脉冲宽度。

数字输出

所有 I/O 引脚都可以使用 SETPIN 命令的 DOUT 参数配置为数字输出。例如：

```
SETPIN GP15, DOUT
```

这意味着当输出引脚设置为逻辑低时，它将输出拉至零，当设置为高时，它将输出拉至 3.3V。在 MMBasic 中，这通过 PIN 命令完成。例如 PIN(GP15) = 0 将引脚 GP15 设置为低电平，而 PIN(GP15) = 1 将其设置为高电平。

脉冲宽度调制

PWM（脉冲宽度调制）命令允许 PicoMite 生成具有程序控制占空比的方波。通过改变占空比，您可以生成程序控制的电压输出，用于控制需要模拟输入的外部设备（如电源、电动机控制器等）。 PWM 输出也适用于驱动伺服电机和通过小型换能器生成声音输出。

PWM 输出由最多 8 个通道（编号 0 到 7）组成，每个通道有两个输出（A 和 B）。对于每个通道，可以选择频率，并且每个输出可以设置不同的占空比。最多可以使用 SETPIN 命令将 16 个引脚配置为 PWM 输出。

通信接口（串行、SPI 和 I²C）

这些在本手册后面的附录中有描述。在使用这些接口之前，必须使用 SETPIN 命令配置相关信号所使用的引脚。

一些设备，如 SD 卡、液晶面板、触摸等，也使用 SPI 或 I²C 接口，使用这些接口的引脚必须在使用之前同样通过 OPTION SYSTEM 命令进行配置。

中断

中断是一种处理可能在不可预测时间发生的事件的便捷方式。一个例子是当用户按下按钮时。在你的程序中，你可以在每个语句后插入代码来检查按钮是否被按下，但使用中断可以使程序更简洁、更易读。

当发生中断时，MMBasic 将执行一个特殊的子程序，完成后返回主程序。主程序完全不知道中断的发生，并将正常继续运行。任何可以用作数字输入的输入/输出引脚都可以使用 SETPI

N 命令配置为生成中断，同时最多可以激活十个中断。中断可以设置为在上升或下降的数字输入信号（或两者）上发生，并将立即跳转到指定的用户定义子程序。每个中断的目标可以相同或不同。从中断返回是通过 END SUB 或 EXIT SUB 命令。请注意，无法将参数传递给子程序，但在中断中调用其他子程序和函数是允许的。

If two or more interrupts occur at the same time they will be processed in order of the interrupts as defined below. During the processing of an interrupt all other interrupts are disabled until the interrupt subroutine returns. During an interrupt (and at all times) the value of the interrupt pin can be accessed using the PIN() function.

Interrupts can occur at any time but they are disabled during INPUT statements. Also interrupts are not recognised during some long hardware related operations (e.g. the TEMP() function, LCD drawing commands, and SD access commands) although they will be recognised if they are still present when the operation has finished. When using interrupts the main program is completely unaffected by the interrupt activity unless a variable used by the main program is changed during the interrupt.

Because interrupts run in the background they can cause difficult to diagnose bugs. Keep in mind the following factors when using interrupts:

- Interrupts are only checked by MMBasic at the completion of each command, and they are not latched by the hardware. This means that an interrupt that lasts for a short time can be missed, especially when the program is executing commands that take some time to execute. Most commands will execute in under 15µs however some commands such as the TEMP() function can take up to 200ms so it is possible for an interrupt to occur and vanish within this window and thus not be recognised.
- When inside an interrupt all other interrupts are blocked so your interrupts should be short and exit as soon as possible. For example, never use PAUSE inside an interrupt. If you have some lengthy processing to do you should simply set a flag and immediately exit the interrupt, then your main program loop can detect the flag and do whatever is required.
- The subroutine that the interrupt calls (and any other subroutines or functions called by it) should always be exclusive to the interrupt. If you must call a subroutine that is also used by an interrupt you must disable the interrupt first (you can reinstate it after you have finished with the subroutine).
- Remember to disable an interrupt when you have finished needing it – background interrupts can cause strange and non-intuitive bugs.

In addition to interrupts generated by the change in state of an I/O pin, an interrupt can also be generated by other sections of MMBasic including timers and communications ports and the above notes also apply to them.

The list of all these interrupts (in high to low priority ranking) is:

1. ON KEY individual
2. ON KEY general
3. PIO RX FIFO
4. PIO TX FIFO
5. PIO RX DMA completion
6. PIO TX DMA completion
7. GUI Int Down
8. GUI Int Up
9. ADC completion
10. I2C Slave Rx
11. I2C Slave Tx
12. I2C2 Slave Rx
13. I2C2 Slave Tx
14. WAV Finished
15. COM1: Serial Port
16. COM2: Serial Port
17. IR Receive
18. Keypad
19. Interrupt command/CSub Interrupt
20. I/O Pin Interrupts in order of definition
21. Tick Interrupts (1 to 4 in that order)

As an example: If an ON KEY interrupt occurred at the same time as a COM1: interrupt the ON KEY interrupt subroutine would be executed first and then, when the interrupt subroutine finished, the COM1: interrupt subroutine would then be executed.

如果两个或多个中断同时发生，它们将按照下面定义的中断顺序进行处理。在处理中断期间，所有其他中断都被禁用，直到中断子程序返回。在中断期间（以及所有时间），可以使用 PIN() 函数访问中断引脚的值。

中断可以在任何时间发生，但在 INPUT 语句期间被禁用。此外，在某些长时间的硬件相关操作（例如 TEMPR() 函数、LCD 绘图命令和 SD 访问命令）期间，中断不会被识别，尽管如果在操作完成时仍然存在，它们将被识别。在使用中断时，主程序完全不受中断活动的影响，除非在中断期间更改了主程序使用的变量。

由于中断在后台运行，它们可能会导致难以诊断的错误。使用中断时，请记住以下因素：

- MMBasic 仅在每个命令完成时检查中断，并且硬件不会锁存它们。这意味着持续时间很短的中断可能会被错过，特别是在程序执行需要一些时间的命令时。大多数命令将在 15µs 内执行，但某些命令（例如 TEMPR() 函数）可能需要长达 200ms，因此在这个时间窗口内可能会发生并消失中断，从而未被识别。
- 在中断内部，所有其他中断都被阻塞，因此您的中断应该尽量简短，并尽快退出。例如，绝不要在中断中使用 PAUSE。如果您有一些较长的处理任务，您应该简单地设置一个标志并立即退出中断，然后您的主程序循环可以检测到该标志并执行所需的操作。
- 中断调用的子程序（以及它调用的任何其他子程序或函数）应始终专属于该中断。如果您必须调用一个也被中断使用的子程序，您必须先禁用该中断（在完成子程序后可以重新启用它）。
- 记得在完成对中断的需求后禁用它——后台中断可能会导致奇怪和不直观的错误。

除了由输入/输出引脚状态变化生成的中断外，其他MMBasic部分（包括定时器和通信端口）也可以生成中断，上述说明同样适用于它们。

所有这些中断的列表（按优先级从高到低排序）是：

1. ON KEY 单独
2. ON KEY 一般
3. PIO RX FIFO
4. PIO TX FIFO
5. PIO RX DMA 完成
6. PIO TX DMA 完成
7. GUI 中断向下
8. GUI 中断向上
9. ADC 完成
10. I2C 从属接收
11. I2C 从属发送
12. I2C2 从属接收
13. I2C2 从属 Tx
14. WAV 播放完成
15. COM1: 串行端口
16. COM2: 串行端口
17. 红外接收
18. 键盘
19. 中断命令/子程序中断
20. 按定义顺序的输入/输出引脚中断
21. 滴答中断（按顺序1到4）

例如：如果在同一时间发生了 ON KEY 中断和 COM1: 中断，则会首先执行 ON KEY 中断子程序，然后在中断子程序完成后，再执行 COM1: 中断子程序。

Sound Output

The PicoMite can play stereo FLAC, MOD, or WAV files located on the Flash Filesystem or SD Card or generate precise sine waves using the PLAY command. Note that the switching power regulator on the Raspberry Pi Pico will cause some interference with the output. This can be reduced by disabling the regulator and powering the module via an external linear regulator

Allocating the Output Pins

The audio is created using PWM outputs so before the PLAY commands can be used the PWM output pins to be used must be allocated as audio outputs follows :

This is done using the OPTION AUDIO command as follows:

```
OPTION AUDIO PWM-A-PIN, PWM-B-PIN
```

This command should be entered at the command prompt and will be saved, so it only needs to be run once. Both pins must be on the same PWM channel with PWM-A the left audio channel and PWM-B the right.

For example:

```
OPTION AUDIO GP0, GP1
```

The audio output can also be generated through a connected MCP48n2 DAC (e.g. MCP4822) in which case it is configured using the command

```
OPTION AUDIO SPI CS-PIN, CLK-PIN, MOSI-PIN
```

In this case there is no requirement for a complex low pass filter and a 120ohm resistor connected to the DAC output and with the other end of the resistor connected to GND via a 100nF capacitor will be more than adequate. When a MCP4822 is used the LDAC pin on the DAC should be connected to GND.

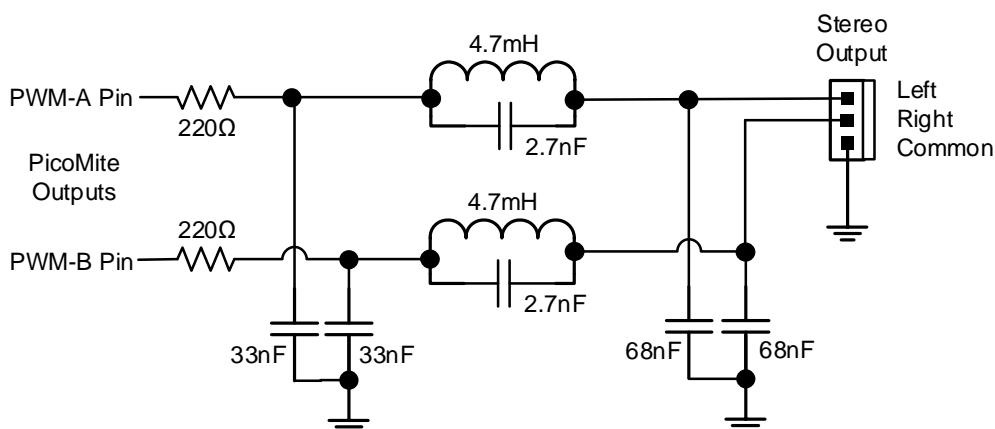
In addition, the audio output can be generated using a VS1053 CODEC module which is configured using the command

```
OPTION AUDIO VS1053 CLKpin, MOSIpin, MISOpin, XCSpin, XDCSpin, DREQpin, XRSTpin
```

This requires no output filtering and can drive 32ohm headphones direct. It also supports additional playback capabilities.

Low Pass Filter

The audio signal is superimposed on a square wave as a pulse width modulated (PWM) signal. This means that a low pass filter, as shown below, is required to recover the audio signal. This circuit is intended to drive an amplifier (not headphones or speakers) and relies on capacitor coupling into the following amplifier (most have this) and has an output level of about 1V peak to peak (650mV RMS).



声音输出

PicoMite 可以播放位于 Flash 文件系统或 SD 卡上的立体声 FLAC、MOD 或 WAV 文件，或使用 PLAY 命令生成精确的正弦波。请注意，树莓派 Pico 上的开关电源调节器会对输出造成一些干扰。通过禁用调节器并通过外部线性调节器为模块供电，可以减少这种干扰。

分配输出引脚

音频是通过 PWM 输出创建的，因此在使用 PLAY 命令之前，必须将要使用的 PWM 输出引脚分配为音频输出，如下所示：

这可以通过以下方式使用 OPTION AUDIO 命令完成：

```
OPTION AUDIO PWM-A-PIN, PWM-B-PIN
```

该命令应在命令提示符下输入并将被保存，因此只需运行一次。

两个引脚必须在同一 PWM 通道上，其中 PWM-A 为左音频通道，PWM-B 为右音频通道。

例如：

```
OPTION AUDIO GP0, GP1
```

音频输出也可以通过连接的 MCP48n2 DAC（例如 MCP4822）生成，在这种情况下，它使用命令配置：

```
OPTION AUDIO SPI CS -PIN, CLK-PIN, MOSI -PIN
```

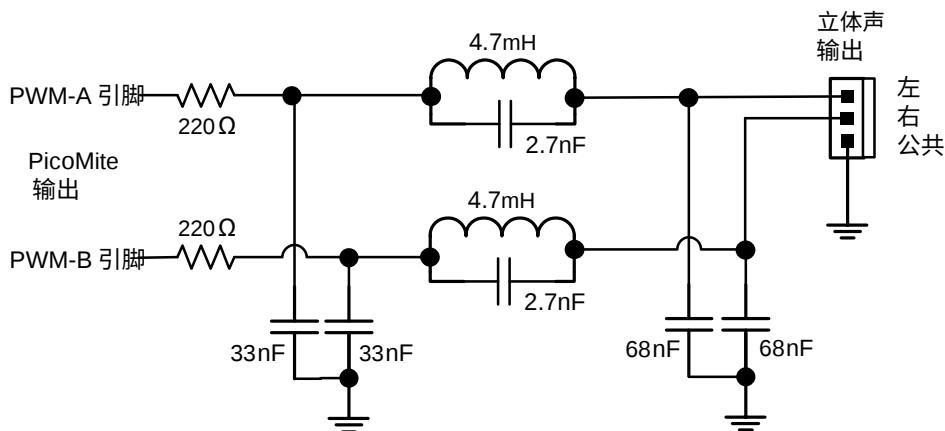
在这种情况下，不需要复杂的低通滤波器，连接到 DAC 输出的 120 欧姆电阻，其另一端通过 100nF 电容器连接到接地，将足够满足要求。当使用 MCP4822 时，DAC 上的 LDAC 引脚应连接到接地。此外，音频输出还可以使用 VS1053 编解码器模块生成，该模块使用命令进行配置：

选项 音频 VS1053 CLK引脚，MOSI引脚，MISO引脚，XC引脚，XDC引脚，DREQ引脚，XRST引脚

这不需要输出滤波，可以直接驱动32欧姆耳机。它还支持额外的播放功能。

低通滤波器

音频信号叠加在方波上，作为脉宽调制（PWM）信号。这意味着需要一个低通滤波器，如下所示，以恢复音频信号。该电路旨在驱动放大器（而不是耳机或扬声器），并依赖于电容耦合到下一个放大器（大多数都有这个），输出电平约为1V峰峰值（650mV RMS）。



Playing WAV FLAC and MOD Files

The PLAY command will play an audio file residing on the Flash Filesystem or SD Card to the sound output. It can be used to provide background music, add sound effects to programs and provide informative announcements.

The commands are:

```
PLAY WAV file$, interrupt  
PLAY FLAC file$, interrupt  
PLAY MODFILE file$, interrupt
```

file\$ is the name of the audio file to play. It must be on the Flash Filesystem or SD Card and the appropriate extension (eg .WAV) will be appended if missing. The audio will play in the background (ie, the program will continue without pause). *interrupt* is optional and is the name of a subroutine which will be called when the file has finished playing.

VS1053 support

If a VS1053 codec is used as the audio output device, additional commands are available:

```
PLAY MP3 file$, interrupt  
PLAY MIDIFILE file$, interrupt  
PLAY MIDI  
PLAY MIDI CMD cmd%, data1% [,data2%]  
PLAY NOTE ON channel, note, velocity  
PLAY NOTE OFF channel, note [,velocity]  
PLAY HALT  
PLAY CONTINUE track$  
PLAY STREAM buffer%(), readpointer%, writepointer%
```

These are explained in more detail in the commands listing section.

Generating Sine Waves

The PLAY TONE command uses the audio output to generate sine waves with selectable frequencies for the left and right channels. This feature is intended for generating attention catching sounds but, because the frequency is very accurate, it can be used for many other applications. For example, signalling DTMF tones down a telephone line or testing the frequency response of loudspeakers.

The syntax of the command is:

```
PLAY TONE left, right, duration, interrupt
```

left and *right* are the frequencies in Hz to use for the left and right channels. The tone plays in the background (the program will continue running after this command) and 'dur' specifies the number of milliseconds that the tone will sound for.

duration is optional and if not specified the tone will continue until explicitly stopped or the program terminates. *interrupt* (if specified) will be triggered when the duration has finished.

The frequency can be from 1 Hz to 20 KHz and is very accurate (it is based on a crystal oscillator). The frequency can be changed at any time by issuing a new PLAY TONE command. Note that the sine wave is generated by stepping through a lookup table so to reduce the distortion the audio output should be passed through a low pass filter.

Specialised Audio Output

The PLAY SOUND command will generate an output based on a mixture of sine, square, etc waveforms. See the details in the command listing.

Using PLAY

It is important to realise that the PLAY command will generate the audio in the background. This allows a program (for example) to play the sound of a bell while continuing with its control function. Without the background facility the whole BASIC program would freeze while the sound was heard.

However, generating the audio in the background has some subtle inferences which can trip up newcomers. For example, take the following program:

```
PLAY TONE 500, 500, 2000  
END
```

播放 WAV、FLAC 和 MOD 文件

PLAY 命令将播放存储在 Flash 文件系统或 SD 卡上的音频文件到声音输出。它可用于提供背景音乐、为程序添加音效以及提供信息公告。

命令如下：

```
PLAY WAV 文件$, 中断  
PLAY FLAC 文件 $, 中断  
PLAY MODFILE 文件 $, 中断
```

*file\$*是要播放的音频文件的名称。它必须位于 Flash 文件系统或 SD 卡上，如果缺少适当的扩展名（例如 .WAV），将会自动添加。音频将在后台播放（即程序将继续运行而不会暂停）。中断是可选的，是一个子程序的名称，当文件播放完成时将被调用。

VS1053 支持

如果使用 VS1053 编解码器作为音频输出设备，则可以使用额外的命令：

```
PLAY MP3 文件 $, 中断  
PLAY MIDIFILE 文件 $, 中断  
播放 MIDI  
播放 MIDI 命令 cmd%, 数据1% [, 数据2%]  
播放音符 开启 通道, 音符, 音量  
播放音符 关闭 通道, 音符 [, 音量]  
暂停播放  
继续播放 track$  
播放流 buffer%(), 读取指针%, 写入指针%
```

这些在命令列表部分有更详细的解释。

生成正弦波

播放音调命令使用音频输出生成可选择频率的正弦波，适用于左通道和右通道。此功能旨在生成引人注意的声音，但由于频率非常准确，它可以用于许多其他应用。例如，通过电话线路发送 DTMF 音调或测试扬声器的频率响应。

命令的语法是：

播放音调 左, 右, 持续时间, 中断

左和右是用于左通道和右通道的频率（单位：赫兹）。音调在后台播放（该程序将在此命令后继续运行），'dur' 指定音调将响起的毫秒数。

持续时间是可选的，如果未指定，音调将持续直到明确停止或程序终止。中断（如果指定）将在持续时间结束时触发。

频率可以从1赫兹到20千赫，非常准确（基于晶体振荡器）。频率可以通过发出新的播放音调命令随时更改。请注意，正弦波是通过查找表逐步生成的，因此为了减少失真，音频输出应通过低通滤波器。

专业音频输出

播放声音命令将基于正弦波、方波等波形的混合生成输出。请参见命令列表中的详细信息。

使用播放

重要的是要意识到播放命令将在后台生成音频。这允许程序（例如）在继续其控制功能的同时播放铃声。如果没有后台功能，整个BASIC程序在听到声音时将会冻结。

然而，在后台生成音频有一些微妙的推论，这可能会让新手感到困惑。例如，考虑以下程序：

```
PLAY TONE 500, 500, 2000  
结束
```

You may expect the 500Hz tone to sound for 2 seconds but in practice it will not make any sound at all. This is because MMBasic will execute the PLAY TONE command (which will start generating the sound in the background) and then it will immediately execute the END command which will terminate the program and the background sound. This will happen so fast that nothing is heard.

Similarly the following program will not work either:

```
PLAY TONE 500, 500, 2000  
PLAY TONE 300, 300, 5000
```

This is because the first command will set a 500Hz tone playing but then the second PLAY command will immediately replace that with a 300Hz tone and following that the program will run off the end terminating the program (and the background audio), resulting in nothing being heard.

If you want MMBasic to wait while the PLAY command is doing its thing you should use suitable PAUSE commands. For example:

```
PLAY TONE 500, 500  
PAUSE 2000  
PLAY TONE 300, 300  
PAUSE 5000  
PLAY STOP
```

This applies to all versions of the PLAY command including PLAY WAV.

Utility Commands

There are a number of commands that can be used to manage the sound output:

PLAY PAUSE	Temporarily halt (pause) the currently playing file or tone.
PLAY RESUME	Resume playing a file or tone that was previously paused.
PLAY NEXT	Play the next WAV or FLAC file in a directory
PLAY PREVIOUS	Play the previous WAV or FLAC file in a directory
PLAY STOP	Terminate the playing of the file or tone. The sound output will also be automatically stopped when the program ends.
PLAY VOLUME L, R	Set the volume to between 0 and 100 with 100 being the maximum volume. The volume will reset to the maximum level when a program is run. A logarithmic scale is used so that PLAY VOLUME 50,50 should sound half as loud as 100,100.

你可能期望500Hz的音调持续2秒，但实际上它根本不会发出任何声音。这是因为MMBasic会执行PLAY TONE命令（这将开始在后台生成声音），然后它会立即执行结束命令，这将终止程序和后台声音。这一切发生得如此之快，以至于什么也听不见。

同样，以下程序也无法正常工作：

```
PLAY TONE 500, 500, 2000  
PLAY TONE 300, 300, 5000
```

这是因为第一个命令会设置一个500Hz的音调在播放，但随后第二个PLAY命令会立即将其替换为300Hz的音调，接着程序将运行到结束，终止程序（和后台音频），导致什么也听不见。

如果你希望MMBasic在PLAY命令执行时等待，你应该使用合适的PAUSE命令。例如：

```
PLAY TONE 500, 500  
PAUSE 2000  
PLAY TONE 300, 300  
PAUSE 5000  
PLAY STOP
```

这适用于所有版本的PLAY命令，包括PLAY WAV。

实用命令

有一些命令可以用来管理声音输出：

PLAY PAUSE	暂时暂停当前播放的文件或音调。
PLAY RESUME	恢复播放之前暂停的文件或音调。
PLAY NEXT	播放目录中的下一个WAV或FLAC文件
PLAY PREVIOUS	播放目录中的上一个WAV或FLAC文件
PLAY STOP	终止文件或音调的播放。当程序结束时，声音输出也会自动停止 。
PLAY VOLUME L, R	将音量设置为0到100之间，100为最大音量。当程序运行时，音量将重置为最大级别。使用对数缩放，因此PLAY VOLUME 50,50的声音应该是100,100的一半。

Special Device Support

To make it easier for a program to interact with the external world the PicoMite includes drivers for a number of common peripheral devices.

These are:

- Infrared remote control receiver and transmitter
- The DS18B20 temperature sensor and DHT22 temperature/humidity sensor
- LCD display modules
- Numeric keypads
- Battery backed clock
- Ultrasonic distance sensor
- WS2812 RGB LEDs
- PS2 Keyboard

Infrared Remote Control Decoder

You can easily add a remote control to your project using the IR command. When enabled this function will run in the background and interrupt the running program whenever a key is pressed on the IR remote control.

It will work with any NEC or Sony compatible remote controls including ones that generate extended messages. Most cheap programmable remote controls will generate either protocol and using one of these you can add a sophisticated flair to your PicoMite based project. The NEC protocol is also used by many other manufacturers including Apple, Pioneer, Sanyo, Akai and Toshiba so their branded remotes can be used.

To detect the IR signal you need an IR receiver. NEC remotes use a 38kHz modulation of the IR signal and suitable receivers tuned to this frequency include the Vishay TSOP4838, Jaycar ZD1952 and Altronics Z1611A. Note that the I/O pins on the PicoMite are only 3.3V tolerant and so the receiver must be powered by a maximum of 3.3V.

Sony remotes use a 40 kHz modulation but receivers for this frequency can be hard to find. Generally 38 kHz receivers will work but maximum sensitivity will be achieved with a 40 kHz receiver.

The IR receiver can be connected to any pin on the PicoMite. This pin must be configured by the program using the command:

```
SETPIN n, IR
```

where *n* is the I/O pin to use for this function.

To setup the decoder you use the command:

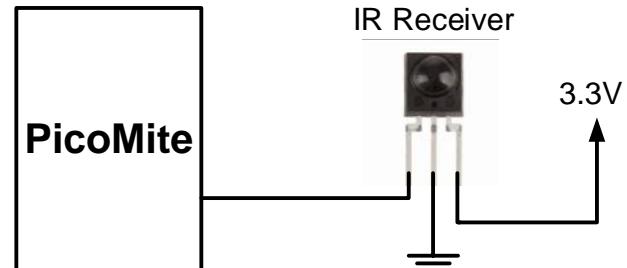
```
IR dev, key, interrupt
```

Where *dev* is a variable that will be updated with the device code and *key* is the variable to be updated with the key code. *Interrupt* is the interrupt subroutine to call when a new key press has been detected. The IR decoding is done in the background and the program will continue after this command without interruption.

This is an example of using the IR decoder connected to the GP6 pin:

```
SETPIN GP6, IR
DIM INTEGER DevCode, KeyCode
IR DevCode, KeyCode, IRInt
DO
    ' < body of the program >
LOOP

SUB IRInt
    ' a key press has been detected
    PRINT "Received device = " DevCode " key = " KeyCode
END SUB
```



特殊设备支持

为了使程序更容易与外部世界交互，PicoMite 包含了许多常见外设设备的驱动程序。

这些包括：

- 红外遥控接收器和发射器
- DS18B20 温度传感器和 DHT22 温度/湿度传感器
- 液晶显示模块
- 数字键盘
- 电池备份时钟
- 超声波距离传感器
- WS2812 RGB LEDs
- PS2 键盘

红外遥控解码器

您可以使用 IR 命令轻松地将遥控器添加到您的项目中。启用后，此功能将在后台运行，并在红外遥控器上按下任何键时中断正在运行的程序。

它可以与任何 NEC 或索尼兼容的遥控器一起使用，包括生成扩展消息的遥控器。大多数便宜的可编程遥控器将生成任一协议，使用其中之一，您可以为基于 PicoMite 的项目增添复杂的风格。NEC 协议也被许多其他制造商使用，包括苹果、先锋、三洋、雅马哈和东芝，因此它们的品牌遥控器可以使用。

要检测红外信号，您需要一个红外接收器。NEC 遥控器使用 38kHz 调制的红外信号，适合此频率的接收器包括

Vishay TSOP4838、Jaycar ZD1952 和 Altronics Z1611A。请注意，PicoMite 上的 I/O 引脚仅耐受 3.3V，因此接收器必须由最大 3.3V 供电。

索尼遥控器使用 40kHz 调制，但这种频率的接收器可能难以找到。一般来说，38kHz 接收器可以工作，但使用 40kHz 接收器将达到最大灵敏度。

红外接收器可以连接到 PicoMite 上的任何引脚。该引脚必须通过程序使用以下命令进行配置：

```
SETPIN n, IR
```

其中 *n* 是用于此功能的输入/输出引脚。

要设置解码器，您使用以下命令：

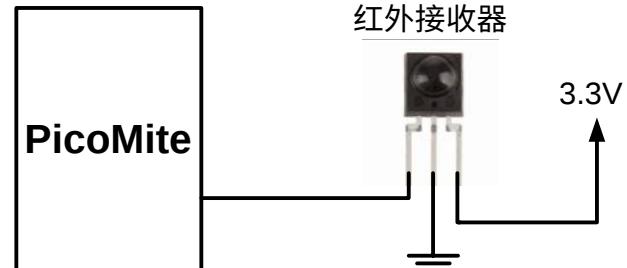
```
IR dev, key, interrupt
```

其中 *dev* 是一个变量，将更新为设备代码，*key* 是将更新为键代码的变量。中断是当检测到新的按键按下时要调用的中断子程序。红外解码在后台进行，程序将在此命令后继续执行而不被中断。

这是一个将红外解码器连接到 GP6 引脚的示例：

```
SETPIN GP6, IR          ' 定义要使用的引脚
DIM INTEGER D evCode, KeyCode   ' 解码器使用的变量
IR DevCode, KeyCode, IR Int    ' 启动红外解码器
DO
    ' <程序主体>
LOOP

子程序 IRInt            ' 检测到按键按下
    打印 "接收到的设备 = " DevCode " 键 = " KeyCode
结束子程序
```



IR remote controls can address many different devices (VCR, TV, etc) so the program would normally examine the device code first to determine if the signal was intended for the program and, if it was, then take action based on the key pressed. There are many different devices and key codes so the best method of determining what codes your remote generates is to use the above program to discover the codes.

Infrared Remote Control Transmitter

Using the IR SEND command you can transmit a 12 bit Sony infrared remote control signal. This is intended for PicoMite to PicoMite or Micromite communications but it will also work with Sony equipment that uses 12 bit codes. Note that all Sony products require that the message be sent three times with a 26 ms delay between each message.

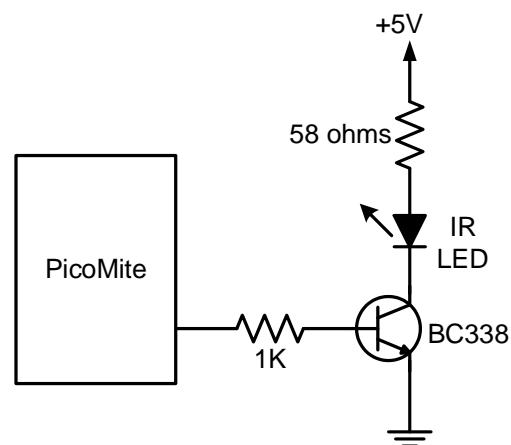
The circuit on the right illustrates what is required. The transistor is used to drive the infrared LED because the output capability of the PicoMite is limited. This circuit provides about 50mA to the LED.

To send a signal you use the command:

```
IR SEND pin, dev, key
```

Where pin is the I/O pin used, dev is the device code to send and key is the key code. Any I/O pin on the PicoMite can be used and you do not have to set it up beforehand (IR SEND will automatically do that).

The modulation frequency used is 38 kHz and this matches the common IR receivers (described in the previous page) for maximum sensitivity when communicating between two PicoMites or with a Micromite.



Measuring Temperature

The TEMP() function will get the temperature from a DS18B20 temperature sensor. This device can be purchased on eBay for about \$5 in a variety of packages including a waterproof probe version.

The DS18B20 can be powered separately by a 3.3V supply or it can operate on parasitic power from the PicoMite as shown on the right. Multiple sensors can be used but a separate I/O pin and a 4.7K pullup resistor is required for each one.

To get the current temperature you just use the TEMP() function in an expression. For example:

```
PRINT "Temperature: " TEMP(pin)
```

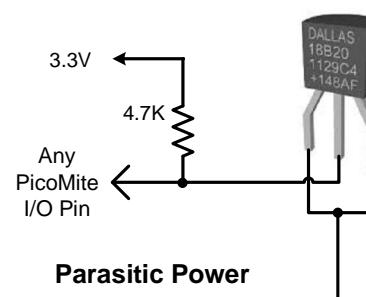
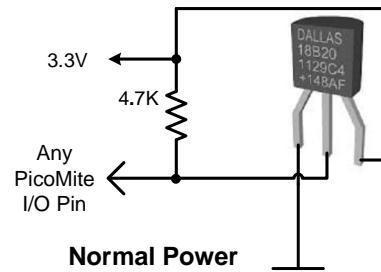
Where 'pin' is the I/O pin to which the sensor is connected. You do not have to configure the I/O pin, that is handled by MMBasic.

The returned value is in degrees C with a resolution of 0.25°C and is accurate to $\pm 0.5^\circ\text{C}$. If there is an error during the measurement the returned value will be 1000.

The time required for the overall measurement is 200ms and the running program will halt for this period while the measurement is being made. This also means that interrupts will be disabled for this period. If you do not want this you can separately trigger the conversion using the TEMP START command then later use the TEMP() function to retrieve the temperature reading. The TEMP() function will always wait if the sensor is still making the measurement.

For example:

```
TEMPR START GP15
< do other tasks >
PRINT "Temperature: " TEMP(GP15)
```



红外遥控器可以控制许多不同的设备（录像机、电视等），因此程序通常会首先检查设备代码，以确定信号是否是针对该程序的，如果是，则根据按下的键采取行动。有许多不同的设备和按键代码，因此确定您的遥控器生成的代码的最佳方法是使用上述程序来发现这些代码。

红外遥控发射器

使用 IRSEND 命令，您可以发送一个 12 位的索尼红外遥控信号。这是为了 PicoMite 与 PicoMite 或 Micromite 之间的通信，但它也适用于使用 12 位代码的索尼设备。请注意，所有索尼产品要求消息发送三次，每条消息之间有 26 毫秒的延迟。

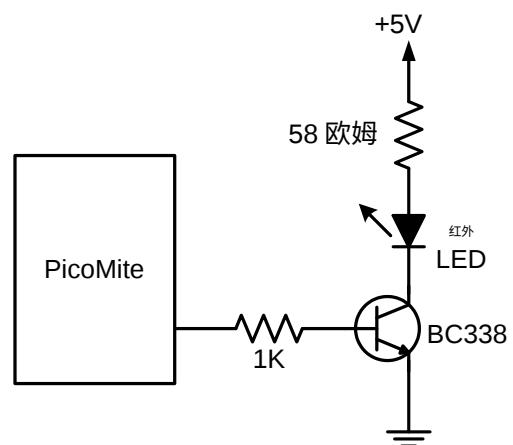
右侧的电路说明了所需的内容。晶体管用于驱动红外LED，因为PicoMite的输出能力有限。该电路为LED提供约50mA的电流。

要发送信号，您可以使用以下命令：

```
IR SEND pin, dev, key
```

其中pin是使用的输入/输出引脚，dev是要发送的设备代码，key是键代码。PicoMite上的任何输入/输出引脚都可以使用，您不必事先进行设置（IR SEND会自动完成此操作）。

使用的调制频率为38kHz，这与常见的红外接收器（在上一页中描述）匹配，以便在两个PicoMite之间或与Micromite通信时达到最大灵敏度。



测量温度

TEMPR()函数将从DS18B20温度传感器获取温度。该设备可以在eBay上以约5美元的价格购买，提供多种包装，包括防水探头版本。

DS18B20可以通过3.3V电源单独供电，或者可以像右侧所示那样从PicoMite获取寄生电源。

可以使用多个传感器，但每个传感器需要一个单独的I/O引脚和一个4.7K的上拉电阻。

要获取当前温度，只需在表达式中使用TEMPR()函数。例如：

```
PRINT "温度: " TEMPTR(pin)
```

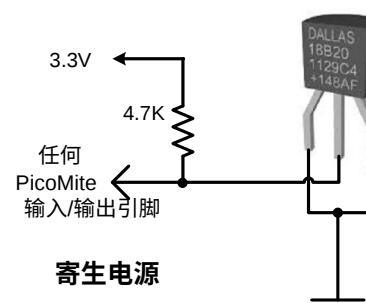
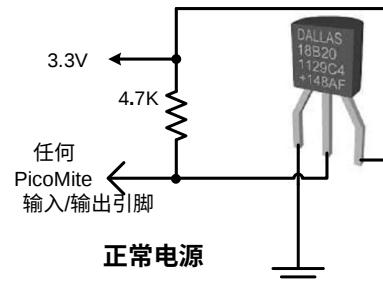
其中'pin'是传感器连接的I/O引脚。您不必配置I/O引脚，这由MM Basic处理。

返回的值以摄氏度表示，分辨率为0.25°C，准确度为±0.5°C。如果测量过程中出现错误，返回的值将为1000。

整体测量所需的时间为200毫秒，运行的程序将在此期间暂停，以进行测量。

这也意味着在此期间中断将被禁用。如果您不想这样，您可以使用TEMPR START命令单独触发转换，然后稍后使用TEMPR()函数来

获取温度读数。如果传感器仍在进行测量，TEMPR()函数将始终等待。



例如：

```
TEMPR START GP15
<执行其他任务>
打印 "温度: " TEMPTR(GP15)
```

Measuring Humidity and Temperature

The DEVICE HUMID command will read the humidity and temperature from a DHT22 humidity/temperature sensor. This device is also sold as the RHT03 or AM2302 but all are compatible and can be purchased on eBay for under \$5. The DHT11 sensor is also supported.

The DHT22 must be powered from 3.3V (the maximum voltage for the PicoMite's I/O pins) and it should have a pullup resistor on the data line as shown. This is suitable for long cable runs (up to 20 meters) but for short runs the resistor can be omitted as the PicoMite also provides an internal weak pullup.

To get the temperature or humidity you use the HUMID command with three arguments as follows:

```
DEVICE HUMID pin, tVar, hVar [,DHT11]
```

Where 'pin' is the I/O pin to which the sensor is connected. The I/O pin will be automatically configured by MMBasic.

'tVar' is a floating point variable in which the temperature is returned and 'hVar' is a second variable for the humidity. The temperature is returned as degrees C with a resolution of one decimal place (e.g. 23.4) and the humidity is returned as a percentage relative humidity (e.g. 54.3).

If the optional DHT11 parameter is set to 1 then the command will use device timings suitable for that device. In this case the results will be returned with a resolution of 1 degree and 1% humidity

For example:

```
DIM FLOAT temp, humidity  
DEVICE HUMID GP15, temp, humidity  
PRINT "The temperature is" temp " and the humidity is" humidity
```

Real Time Clock Interface

Using the RTC GETTIME command it is easy to get the current time from a PCF8563, DS1307, DS3231 or DS3232 real time clock as well as compatible devices such as the M41T11. These integrated circuits are popular and cheap, will keep accurate time even with the power removed and can be purchased for \$2 to \$8 on eBay. Complete modules including the battery can also be purchased on eBay for a little more.

The PCF8563 and DS1307 will keep time to within a minute or two over a month while the DS3231 and DS3232 are particularly precise and will remain accurate to within a minute over a year.

These chips are I²C devices and should be connected to the I²C I/O pins of the PicoMite.

Internal pullup resistors (100KΩ) are applied to the I²C I/O pins so in many cases external resistors are not needed.

In order to enable the RTC you first need to allocate the I²C pins to be used using the command:

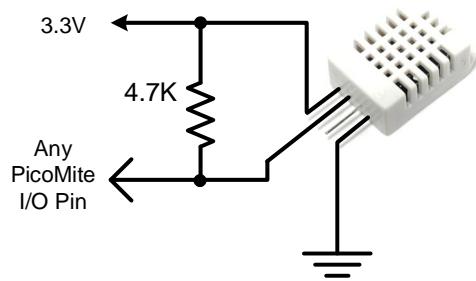
```
OPTION SYSTEM I2C SDApin, SCLpin
```

The time used by the RTC must also be set. That is done with the RTC SETTIME command which uses the format RTC SETTIME year, month, day, hour, minute, second. Note that the hour must be in 24 hour format.

For example, the following will set the real time clock to 4PM on the 10th November 2021:

```
RTC SETTIME 2021, 11, 10, 16, 0, 0
```

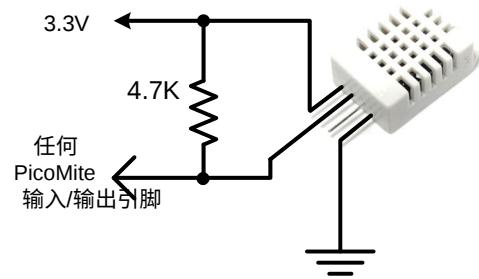
To get the time you use the RTC GETTIME command which will read the time from the real time clock chip and set the clock inside the PicoMiteVGA . Normally this command will be placed at the beginning of the program or in the subroutine MM.STARTUP so that the time is set on power up. The command OPTION RTC AUTO ENABLE can also be used to set an automatic update of the time\$ & date\$ from RTC on boot & every hour.



测量湿度和温度

DEVICE HUMID命令将从DHT22湿度/温度传感器读取湿度和温度。该设备也被称为RHT03或AM2302，但所有设备都是兼容的，可以在eBay上以不到5美元的价格购买。DHT11传感器也受到支持。

DHT22必须从3.3V供电（这是PicoMite的I/O引脚的最大电压），并且数据线应如图所示连接上拉电阻。这适合长电缆运行（最长可达20米），但对于短距离运行，可以省略电阻，因为PicoMite还提供内部弱上拉。



要获取温度或湿度，您可以使用HUMID命令，带有三个参数，如下所示：

```
DEVICE HUMID pin, tVar, hVar [ ,DHT11 ]
```

其中'pin'是传感器连接的I/O引脚。I/O引脚将由MMBasic自动配置。

'tVar'是一个浮点变量，用于返回温度，'hVar'是第二个变量，用于湿度。温度以摄氏度返回，分辨率为一位小数（例如23.4），湿度以相对湿度的百分比返回（例如54.3）。

如果可选的DHT11参数设置为1，则该命令将使用适合该设备的时序。

在这种情况下，结果将以1度和1%的湿度的分辨率返回

例如：

```
DIM FLOAT temp, humidity  
DEVICE HUMID GP15, temp, humidity  
PRINT "温度是" temp ", 湿度是" humidity
```

实时钟接口

使用RTC GETTIME命令，可以轻松从PCF8563、DS1307、DS3231或DS3232实时钟以及兼容设备（如M41T11）获取当前时间。这些集成电路流行且便宜，即使在断电的情况下也能保持准确的时间，价格在eBay上为2到8美元。包括电池在内的完整模块也可以在eBay上以稍高的价格购买。

PCF8563和DS1307在一个月内的时间误差在一两分钟之内，而DS3231和DS3232则特别精确，能够在一年内保持在一分钟之内的准确性。

这些芯片是I²C设备，应连接到PicoMite的I²C I/O引脚。I²C I/O引脚上施加了内部上拉电阻（100KΩ），因此在许多情况下不需要外部电阻。

为了启用RTC，您首先需要使用命令分配要使用的I²C引脚：

```
OPTION SYSTEM I2C SDApin, SCLpin
```

RTC使用的时间也必须设置。这可以通过RTCSETTIME命令完成，该命令使用格式RTC SETTIME 年，月，天，小时，分钟，秒。请注意，小时必须采用24小时制。

例如，以下命令将实时时钟设置为2021年11月10日下午4点：

```
RTC SETTIME 2021, 11, 10, 16, 0, 0
```

要获取时间，可以使用RTC GETTIME命令，该命令将从实时时钟芯片读取时间并设置PicoMiteVGA内部的时钟。通常，该命令会放在程序的开头或子程序MM.STARTUP中，以便在上电时设置时间。命令OPTION RTC AUTO ENABLE也可以用于在启动时和每小时自动更新time\$和date\$。

Measuring Distance

Using a HC-SR04 ultrasonic sensor and the DISTANCE() function you can measure the distance to a target.

This device can be found on eBay for about \$4 and it will measure the distance to a target from 3cm to 3m. It works by sending an ultrasonic sound pulse and measuring the time it takes for the echo to be returned.

Compatible sensors are the SRF05, SRF06, Parallax PING and the DYP-ME007 (which is waterproof and therefore good for monitoring the level of a water tank).

On the PicoMite you use the DISTANCE function as follows:

```
d = DISTANCE(trig, echo)
```

The value returned is the distance in centimetres to the target.

Where trig is the I/O pin connected to the "trig" input of the sensor and echo is the pin connected to the "echo" output of the sensor. You can also use 3-pin devices and in that case only one pin number is specified. The maximum voltage on the PicoMite's I/O pins is 3.3V so a resistor divider will be required to interface the PicoMite to the echo pin of the sensor (which operates on 5V).



LCD Display

The LCD command will display text on a standard LCD module with the minimum of programming effort.

This command will work with LCD modules that use the KS0066, HD44780 or SPLC780 controller chip and have 1, 2 or 4 lines. Typical displays include the LCD16X2 (futurlec.com), the Z7001 (altronics.com.au) and the QP5512 (jaycar.com.au). eBay is another good source where prices can range from \$10 to \$50.

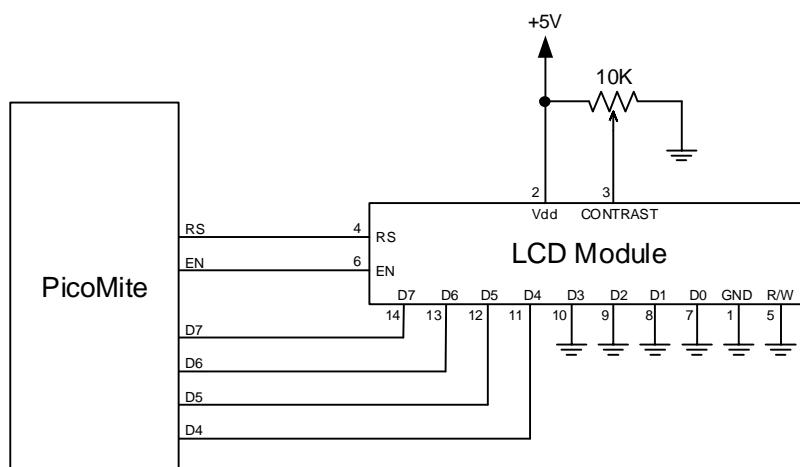
To setup the display you use the DEVICE LCD INIT command:

```
DEVICE LCD INIT d4, d5, d6, d7, rs, en
```



d4, d5, d6 and d7 are the numbers of the I/O pins that connect to inputs D4, D5, D6 and D7 on the LCD module (inputs D0 to D3 and R/W on the module should be connected to ground). 'rs' is the pin connected to the register select input on the module (sometimes called CMD or DAT). 'en' is the pin connected to the enable or chip select input on the module.

Any I/O pins on the PicoMite can be used and you do not have to set them up beforehand (the LCD command automatically does that for you). The following shows a typical set up.



To display characters on the module you use the LCD command:

```
DEVICE LCD line, pos, data$
```

Where line is the line on the display (1 to 4) and pos is the position on the line where the data is to be written (the first position on the line is 1). data\$ is a string containing the data to write to the LCD display. The characters in data\$ will overwrite whatever was on that part of the LCD.

测量距离

使用HC-SR04超声波传感器和DISTANCE()函数，您可以测量到目标的距离。

该设备可以在eBay上以约4美元的价格找到，它可以测量到目标的距离，从3厘米到3米。它通过发送超声波脉冲并测量回声返回所需的时间来工作。

兼容的传感器包括SRF05、SRF06、Parallax PING和DYP-ME007（该传感器防水，因此适合监测水箱的水位）。



在PicoMite上，您可以按如下方式使用DISTANCE函数：

```
d = DISTANCE(trig, echo)
```

返回的值是到目标的距离，单位为厘米。

其中trig是连接到传感器"trig"输入的I/O引脚，echo是连接到传感器"echo"输出的引脚。您还可以使用3引脚设备，在这种情况下只需指定一个引脚编号。PicoMite的I/O引脚上的最大电压为3.3V，因此需要一个电阻分压器将PicoMite与传感器的回声引脚（其工作电压为5V）连接。

液晶显示

LCD命令将在标准液晶模块上显示文本，所需的编程工作量最小。



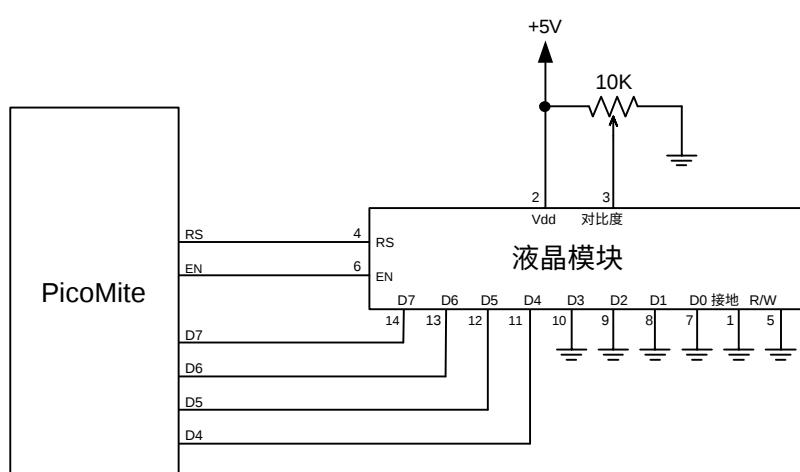
该命令适用于使用KS0066、HD44780或SPLC780控制器芯片的液晶模块，并且具有1、2或4行。典型的显示器包括LCD16X2(futurlec.com)、Z7001(altronics.com.au)和QP5512(jaycar.com.au)。eBay也是一个不错的来源，价格范围从10美元到50美元不等。

要设置显示器，您可以使用DEVICE LCD INIT命令：

```
DEVICE LCD INIT d4, d5, d6, d7, rs, en
```

d4、d5、d6和d7是连接到液晶模块输入D4、D5、D6和D7的I/O引脚编号（模块上的输入D0到D3和R/W应连接到地）。'rs'是连接到模块上寄存器选择输入的引脚（有时称为CMD或DAT）。'en'是连接到模块上启用或芯片选择输入的引脚。

PicoMite上的任何I/O引脚都可以使用，您无需提前设置它们（LCD命令会自动为您完成这项工作）。以下显示了一个典型的设置。



要在模块上显示字符，您可以使用LCD命令：

```
DEVICE LCD line, pos, data$
```

其中line是显示器上的行（1到4），pos是要写入数据的行上的位置（行上的第一个位置为1）。data\$是一个包含要写入液晶显示的字符串。data\$中的字符将覆盖LCD上该部分的任何内容。

The following shows a typical usage where d4 to d7 are connected to pins GP2 to GP4 on the PicoMite, rs is connected to pin GP6 and en to pin GP7.

```
DEVICE LCD INIT GP2, GP3, GP4, GP5, GP6, GP7
DEVICE LCD 1, 2, "Temperature"
DEVICE LCD 2, 6, STR$(TEMPR(GP15))      ' DS18B20 connected to pin GP15
```

Note that this example also uses the TEMP(R) function to get the temperature (described above).

Keypad Interface

A keypad is a low tech method of entering data into a PicoMite based system. The PicoMite supports either a 4x3 keypad or a 4x4 keypad and the monitoring and decoding of key presses is done in the background. When a key press is detected an interrupt will be issued where the program can deal with it.

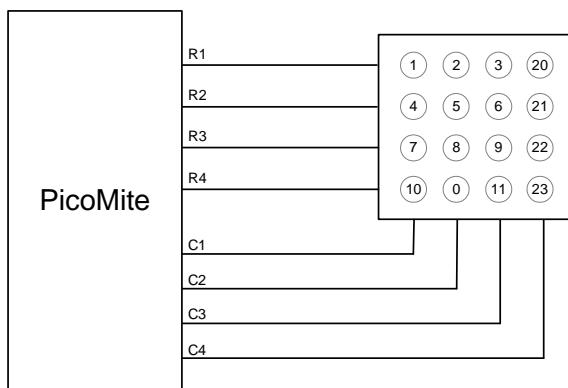
Examples of a 4x3 keypad and a 4x4 keypad are the Altronics S5381 and S5383 (go to www.altronics.com).

To enable the keypad feature you use the command:

```
KEYPAD var, int, r1, r2, r3, r4, c1, c2, c3, c4
```

Where var is a variable that will be updated with the key code and int is the name of the interrupt subroutine to call when a new key press has been detected. r1, r2, r3 and r4 are the pin numbers used for the four row connections to the keypad (see the diagram below) and c1, c2, c3 and c4 are the column connections. c4 is only used with 4x4 keypads and should be omitted if you are using a 4x3 keypad.

Any I/O pins on the PicoMite can be used and you do not have to set them up beforehand, the KEYPAD command will automatically do that for you.



The detection and decoding of key presses is done in the background and the program will continue after this command without interruption. When a key press is detected the value of the variable var will be set to the number representing the key (this is the number inside the circles in the diagram above). Then the interrupt will be called.

For example:

```
Keypad KeyCode, KP_Int,GP2,GP3,GP4,GP5,GP6,GP7,GP8      ' 4x3 keybd
DO
    < body of the program >
LOOP

SUB KP_Int                                         ' a key press has been detected
    PRINT "Key press = " KeyCode
END SUB
```

以下显示了一个典型的用法，其中d4到d7连接到PicoMite上的引脚GP2到GP4，rs连接到引脚GP6，en连接到引脚GP7。

```
设备 LCD 初始化 GP2 , GP3 , GP4 , GP5 , GP6 , GP7  
设备 LCD 1 , 2 , "温度"  
设备 LCD 2 , 6 , STR$(TEMPR(GP15)) ' DS18B20连接到引脚 GP15
```

请注意，此示例还使用了TEMPR()函数来获取温度（如上所述）。

键盘接口

键盘是一种低技术的将数据输入到基于PicoMite的系统的方法。PicoMite支持4x3键盘或4x4键盘，按键的监控和解码在后台进行。当检测到按键按下时，将发出一个中断，程序可以对此进行处理。

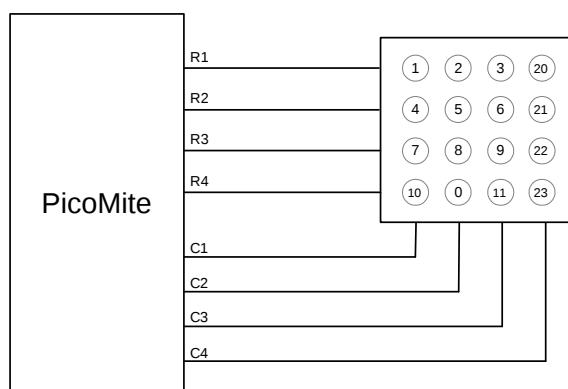
4x3 键盘和 4x4 键盘的示例是 Altronics S5381 和 S5383（请访问 www.altronics.com）。

要启用键盘功能，您可以使用以下命令：

```
KEYPAD var, int, r1, r2, r3, r4, c1, c2, c3, c4
```

其中 var 是一个变量，将更新为按键代码，int 是在检测到新按键按下时要调用的中断子程序的名称。r1、r2、r3 和 r4 是用于连接到键盘的四个行的引脚编号（请参见下图），而 c1、c2、c3 和 c4 是列连接。c4 仅在 4x4 键盘中使用，如果您使用的是 4x3 键盘，则应省略。

PicoMite上的任何I/O引脚都可以使用，您无需提前设置它们，KEYPAD命令会自动为您完成此操作。



按键检测和解码在后台进行，程序将在此命令后继续执行而不会中断。当检测到按键时，变量var的值将被设置为表示该按键的数字（这是上图中圆圈内的数字）。然后将调用中断。

例如：

```
Keypad KeyCode, KP_Int, GP2, GP3, GP4, GP5, GP6, GP7, GP8      ' 4x3键盘
DO
    <程序主体>
LOOP

SUB KP_Int
    PRINT "按键 = " KeyCode
    END SUB
                                ' 检测到按键
```

WS2812 Support

The PicoMite has built in support for the WS2812 multicolour LED chip. This chip needs a very specific timing to work properly and with the DEVICE WS2812 command it is easy to control these devices with minimal effort.

This command will output the required signals needed to drive a chain of WS2812 LED chips connected to the pin specified and set the colours of each LED in the chain. The syntax of the command is:

```
DEVICE WS2812 type, pin, nbr%, colours%[ () ]
```

Note that the pin must be set to a digital output before this command is used. The colours%() array should be sized to have at least the same number of elements as the number of LEDs to be driven (nbr%). Each element in the array should contain the colour in the normal RGB888 format (0 - HFFFFFF). Where a single LED is to be driven then colours% should be a simple variable.

Up to 256 WS2812 chips in a string are supported.

'type' is a single character specifying the type of chip being driven as follows:

O = original WS2812

B = WS2812B

S = SK6812

W = SK6812W (RGBW)

As an example:

```
DIM b% (4)=(RGB(red), Rgb(green), RGB(blue), RGB(Yellow), rgb(cyan))  
SETPIN GP5, DOUT  
DEVICE WS2812 O, GP5, 5, b%()
```

will output the specified colours to an array of five WS2812 LEDs daisy chained off pin GP5.

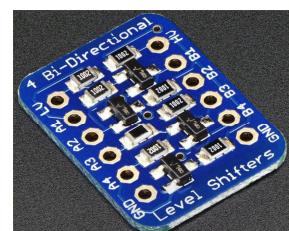
It is possible that a WS2812 will not work reliably with the 3.3V output from the PicoMite. In this case there are a number of solutions:

- Use the WS2812B which will work with a 3.3V supply and inputs.
- Use a level shifter to drive the WS2812.
- Use a single WS2812 powered from 3.3V as a first stage to buffer the input of the first "real" LED in the string. The minimum supply for the WS2812 is 4V but in many cases it will work at 3.3V.

PS2 Keyboard

On the PicoMite you can attach a PS2 keyboard and, display the output of the interpreter on the LCD (see the LCD Display as the Console Output section below). This turns the PicoMite into a completely self contained computer with its own keyboard and display. Using the built in colour coded editor programs can be entered, edited and run without requiring another computer.

Because the PicoMite I/O is specified for a maximum of 3.6V and PS2 keyboards run off 5V level conversion should be used on the CLOCK and DATA pins. A suitable commercially available Adafruit 4 channel bi-directional level converter is pictured on the right. The PS2 CLOCK pin should be connected via the level converter to PicoMite pin 11 (GP8) and the PS2 DATA pin to PicoMite pin 12 (GP9).



Before the keyboard can be used it must first be enabled by specifying the language of the keyboard:

```
OPTION KEYBOARD language [,capslock][,numlock][,repeatstart][,repeatrate]
```

Where 'language' is a two character code such as US for the standard keyboard used in the USA, Australia and New Zealand. Other keyboard layouts that can be specified are United Kingdom (UK), French (FR), German (GR), Belgium (BE), Italian (IT), Brazilian (BR) or Spanish (ES). Note that the non US layouts map some of the special keys present on these keyboards but the corresponding special character will not display as they are not part of the standard PicoMite fonts (another character will be used instead).

This command configures the I/O pins dedicated to the keyboard and initialises it for use. As with similar commands this option will be saved in flash memory and automatically applied on power up. If you need to remove the keyboard you can do this with the OPTION KEYBOARD DISABLE command.

See the OPTION KEYBOARD command for details of the optional parameters.

WS2812支持

PicoMite内置对WS2812多彩LED芯片的支持。这个芯片需要非常特定的时序才能正常工作，使用DEVICE WS2812命令可以轻松控制这些设备，几乎不需要任何努力。

该命令将输出驱动连接到指定引脚的WS2812 LED芯片链所需的信号，并设置链中每个LED的颜色。命令的语法是：

```
DEVICE WS2812 类型, 引脚, 编号%, 颜色%[ () ]
```

请注意，在使用此命令之前，引脚必须设置为数字输出。颜色%()数组的大小应至少与要驱动的LED数量（编号%）相同。数组中的每个元素应包含正常RGB888格式的颜色（0 - HFFFFFF）。如果只驱动一个LED，则颜色%应为一个简单的变量。

支持最多256个WS2812芯片在一条链中。

'类型'是一个单字符，指定要驱动的芯片类型，如下所示：

O = 原始 WS2812

B = WS2812B

S = SK6812

W = SK6812W (RGBW)

作为一个例子：

```
DIM b% (4)=(RGB(红色), Rgb(绿色), RGB(蓝色), RGB(黄色), rgb(青色))
```

```
SETPIN GP5, DOUT
```

```
DEVICE WS2812 O, GP5, 5, b%()
```

将输出指定的颜色到连接在引脚 GP5 上的五个 WS2812 LED 的数组中。可能 WS2812

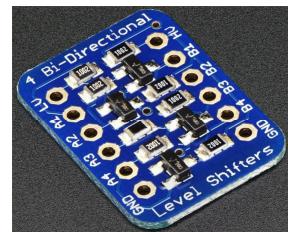
在 PicoMite 的 3.3V 输出下无法可靠工作。在这种情况下，有多种解决方案：

- 使用 WS2812B，它可以在 3.3V 电源和输入下工作。
- 使用电平转换器来驱动 WS2812。
- 使用一个由 3.3V 供电的单个 WS2812 作为第一阶段，以缓冲字符串中第一个 "真实" LED 的输入。WS2812的最小供电为4V，但在许多情况下它可以在3.3V下工作。

PS2键盘

在PicoMite上，您可以连接PS2键盘，并在液晶显示器上显示解释器的输出（请参见下面的液晶显示作为控制台输出部分）。这使PicoMite变成一个完全独立的计算机，配有自己的键盘和显示器。使用内置的颜色编码编辑器，可以输入、编辑和运行程序，而无需其他计算机。

由于PicoMite的I/O规定最大为3.6V，而PS2键盘运行在5V，因此应在时钟和数据引脚上使用电平转换。右侧展示了一款适合的商业可用Adafruit 4通道双向电平转换器。PS2时钟引脚应通过电平转换器连接到PicoMite引脚11（GP8），PS2数据引脚应连接到PicoMite引脚12（GP9）。



在键盘可以使用之前，必须首先通过指定键盘的语言来启用它：

```
OPTION KEYBOARD language [,capslock][,numlock][,repeatstart][,repeatrate]
```

其中'language'是一个两字符代码，例如美国标准键盘使用的US，适用于美国、澳大利亚和新西兰。可以指定的其他键盘布局包括英国（UK）、法语（FR）、德语（GR）、比利时（BE）、意大利（IT）、巴西（BR）或西班牙（ES）。请注意，非美国布局映射了这些键盘上某些特殊键，但相应的特殊字符不会显示，因为它们不是标准PicoMite字体的一部分（将使用其他字符代替）。

此命令配置专用于键盘的I/O引脚并初始化以供使用。与类似命令一样，此选项将保存在闪存中，并在上电时自动应用。如果您需要移除键盘，可以使用OPTION KEYBOARD DISABLE命令。

有关可选参数的详细信息，请参见OPTION KEYBOARD命令。

LCD Display as the Console Output

The keyboard can be used on its own as an alternative input method but it works particularly well when the LCD display panel is used as the console output. The LCD should be one of the SSD1963 versions in the landscape orientation and it must be first configured using OPTION LCDPANEL.

To enable the output to the LCD panel you should use the following command:

```
OPTION LCDPANEL CONSOLE [font [, fc [, bc [, blight]]]]
```

'font' is the default font, 'fc' is the default foreground colour, 'bc' is the default background colour and 'blight' is the default backlight brightness (2 to 100). These settings are saved in flash and are used to configure MMBasic at power up. They are all optional and default to font 2, white, black and 100%.

Colour coding in the editor (see below) is also turned on by this command (OPTION COLOURCODE OFF will turn it off again). To disable using the LCD panel as the console the command is OPTION LCDPANEL NOCONSOLE.

There are three SPI displays that can also be used as console devices. ILI9341 (but only if MISO is connected), ILI9488/86 (connect MISO with a 680Ω series resistor as the controller does not tri-state its output properly) and the 2.8" Waveshare ST7789. Note that with all of these the scrolling of text on the screen will be very slow.

Used with a PS2 keyboard this option turns the PicoMite into a self contained computer with its own keyboard and display. Rather like a modern version of the Maximite (see <http://geoffg.net/maxomite.html>).

OV7670 Camera module

The PicoMite has support for a OV7670 camera module. See the DEVICE CAMERA command for details

将LCD显示器作为控制台输出

键盘可以单独用作替代输入方法，但当LCD显示面板用作控制台输出时，它的效果特别好。LCD应为横屏方向的SSD1963版本之一，并且必须首先使用OPTION LCDPANEL进行配置。要启用LCD面板的输出，您应使用以下命令：

```
OPTION LCDPANEL CONSOLE [font [, fc [, bc [, blight]]]]
```

'font'是默认字体，'fc'是默认前景颜色，'bc'是默认背景颜色，'blight'是默认背光亮度（2到100）。这些设置保存在闪存中，并在上电时用于配置MMBasic。它们都是可选的，默认值为字体2、白色、黑色和100%。

编辑器中的颜色编码（见下文）也通过此命令开启（OPTION COLOURCODE OFF将再次关闭它）。要禁用将LCD面板用作控制台，命令为 OPTION LCDPANEL NOCONSOLE。

有三种SPI显示器也可以用作控制设备。ILI9341（但仅在MISO连接时），ILI9488/86（将MISO与680Ω串联电阻连接，因为控制器无法正确三态其输出）和2.8" Waveshare ST7789。请注意，使用这些显示器时，屏幕上的文本滚动会非常缓慢。

与PS2键盘一起使用时，此选项将PicoMite变成一个独立的计算机，配有自己的键盘和显示器。有点像现代版的Maximite（请参见<http://geoffg.net/maxomite.html>）。

OV7670摄像头模块

PicoMite支持OV7670摄像头模块。有关详细信息，请参见DEVICE CAMERA命令

Display Panels

The PicoMite includes support for many LCD display panels using an SPI, I²C or parallel interface.

These commands must be entered at the command prompt (not in a program) and will cause the PicoMite to restart. This has the side effect of disconnecting the USB console interface which will need to be reconnected.

Note that the maximum voltage on all the PicoMite I/O pins is 3.3V. Level shifting will be required if a display uses 5V levels for signalling.

SPI Based Display Panels

The SPI based display controllers share the SYSTEM SPI channel interface on the PicoMite with the touch controller (if present). An SD Card can also be configured to use the same pins. When this is done the pins allocated to the SYSTEM SPI will not be available to other MMBasic commands. The speed of drawing to SPI based displays will be largely unaffected by the CPU speed.

These panels are configured using the following commands. In all commands the parameters are:

- OR = This is the orientation of the display and it can be LANDSCAPE, PORTRAIT, RLANDSCAPE or RPORTRAIT. These can be abbreviated to L, P, RL or RP. The R prefix indicates the reverse or "upside down" orientation.
- DC = Display Data/Command control pin.
- RESET = Display Reset pin (when pulled low).
- CS = Display Chip Select pin.
- BACKLIGHTPIN = Optional pin used to control the backlight brightness.

Any free pins can be used.

```
OPTION LCDPANEL ILI9341, OR, DC, RESET, CS [,BACKLIGHTPIN]
```

Initialises a TFT display using the ILI9341 controller. This supports 320 * 240 resolution. Displays using this controller are capable of transparent text and will work with the BLIT and BLIT READ commands.

```
OPTION LCDPANEL ILI9163, OR, DC, RESET, CS [,BACKLIGHTPIN]
```

Initialises a TFT display using the ILI9163 controller. This supports 128 * 128 resolution.

```
OPTION LCDPANEL ILI9481, OR, DC, RESET, CS [,BACKLIGHTPIN]
```

Initialises a TFT display using the ILI9481 controller. This supports 480 * 320 resolution.

```
OPTION LCDPANEL ILI9481IPS, OR, DC, RESET, CS [,BACKLIGHTPIN]
```

Initialises an IPS display using the ILI9481 controller. This supports 480 * 320 resolution.

```
OPTION LCDPANEL ILI9488, OR, DC, RESET, CS [,BACKLIGHTPIN]
```

Initialises a TFT display using the ILI9488 controller. This supports 480 * 320 resolution. Note that this controller has an issue with the MISO pin which interferes with the touch controller. For this display to work **the MISO pin must not be connected**.

```
OPTION LCDPANEL ILI9488W, OR, DC, RESET, CS [,BACKLIGHTPIN]
```

Initialises a TFT display using the ILI9488 controller. This supports the Waveshare 3.5" display as used on their Pico Eval board and the normal 3.5" display adapter.

显示面板

PicoMite 支持许多使用 SPI、I²C 或并行接口的 LCD 显示面板。

这些命令必须在命令提示符下输入（而不是在程序中），并将导致 PicoMite 重启。这会导致 USB 控制台接口断开，需要重新连接。

请注意，所有 PicoMite I/O 引脚的最大电压为 3.3V。如果显示器使用 5V 电平进行信号传输，则需要进行电平转换。

基于 SPI 的显示面板

基于 SPI 的显示控制器与触摸控制器（如果存在）共享 PicoMite 上的系统 SPI 通道接口。SD 卡也可以配置为使用相同的引脚。完成此操作后，分配给系统 SPI 的引脚将无法用于其他 MMBasic 命令。绘制到基于 SPI 的显示器的速度在很大程度上不会受到 CPU 速度的影响。

这些面板使用以下命令进行配置。在所有命令中，参数为：

- OR = 这是显示器的方向，可以是横屏、肖像、反向横屏或反向肖像。这些可以缩写为 L、P、R 或 RP。R 前缀表示反向或“倒置”的方向。
- DC = 显示数据/命令控制引脚。
- RESET = 显示重置引脚（当拉低时）。
- CS = 显示芯片选择引脚。
- BACKLIGHTPIN = 可选引脚，用于控制背光亮度。

可以使用任何空闲引脚。

OPTION LCDPANEL ILI9341, OR, DC, RESET, CS [, BACKLIGHTPIN]

使用 ILI9341 控制器初始化 TFT 显示器。此显示器支持 320 * 240 分辨率。使用此控制器的显示器能够显示透明文本，并且可以与 BLIT 和 BLIT READ 命令一起使用。

OPTION LCDPANEL ILI9163, OR, DC, RESET, CS [, BACKLIGHTPIN]

使用 ILI9163 控制器初始化 TFT 显示器。此显示器支持 128 * 128 分辨率。

OPTION LCDPANEL ILI9481, OR, DC, RESET, CS [, BACKLIGHTPIN]

使用 ILI9481 控制器初始化 TFT 显示器。此显示器支持 480 * 320 分辨率。

选项 LCD 面板 ILI9481IPS, 或, 直流, 重置, CS [, 背光引脚]

使用 ILI9481 控制器初始化一个 IPS 显示器。此显示器支持 480 * 320 分辨率。

选项 LCD 面板 ILI9488, 或, 直流, 重置, CS [, 背光引脚]

使用 ILI9488 控制器初始化一个 TFT 显示器。此显示器支持 480 * 320 分辨率。请注意，此控制器在 MISO 引脚上存在问题，会干扰触摸控制器。为了使此显示器正常工作 MISO 引脚必须未连接。

选项 LCD 面板 ILI9488W, 或, 直流, 重置, CS [, 背光引脚]

使用 ILI9488 控制器初始化一个 TFT 显示器。此选项支持 Waveshare 3.5" 显示器，适用于他们的 Pico 评估板和普通 3.5" 显示适配器。

OPTION LCDPANEL N5110, OR, DC, RESET, CS [,contrast]

Initialises a LCD display using the Nokia 5110 controller. This supports 84 * 48 resolution. An additional parameter ‘contrast’ may be specified to control the contrast of the display. Try contrast values between &HA8 and &HD0 to suit your display, default if omitted is &HB1

OPTION LCDPANEL SSD1306SPI, OR, DC, RESET, CS [,offset]

Initialises a OLED display using the SSD1306 controller with an SPI interface. This supports 128 * 64 resolution. An additional parameter ‘offset’ may be specified to control the position of the display. 0.96" displays typically need a value of 0. 1.3" displays typically need a value of 2. Default if omitted is 0.

OPTION LCDPANEL SSD1331, OR, DC, RESET, CS [,BACKLIGHTPIN]

Initialises a colour OLED display using the SSD1331 controller. This supports 96 * 64 resolution.

OPTION LCDPANEL ST7735, OR, DC, RESET, CS [,BACKLIGHTPIN]

Initialises a TFT display using the ST7735 controller. This supports 160 * 128 resolution.

OPTION LCDPANEL ST7735S, OR, DC, RESET, CS [,BACKLIGHTPIN]

Initialises a IPS display using the ST7735S controller. This supports 160 * 80 resolution.

OPTION LCDPANEL ST7735S_W, OR, DC, RESET, CS [,BACKLIGHTPIN]

Initialises a Waveshare 128x128 ST7735S display. This supports 128 * 128 resolution.

OPTION LCDPANEL ST7789, OR, DC, RESET, CS [,BACKLIGHTPIN]

Initialises a IPS display using the 7789 controller. This supports 240 * 240 resolution.

NOTE: display boards without a CS pin are not currently supported on the PicoMite unless modified.

OPTION LCDPANEL ST7789_135, OR, DC, RESET, CS [,BACKLIGHTPIN]

Initialises a IPS display using the 7789 controller. This supports 240 * 135 resolution.

NOTE: display boards without a CS pin are not currently supported on the PicoMite unless modified.

OPTION LCDPANEL ST7789_320, OR, DC, RESET, CS [,BACKLIGHTPIN]

Initialises a IPS display using the 7789 controller. This type supports the 320 * 240 resolution display from Waveshare (<https://www.waveshare.com/wiki/Pico-ResTouch-LCD-2.8>).

These are capable of transparent text and will work with the BLIT and BLIT READ commands.

NOTE: display boards without a CS pin are not currently supported on the PicoMite unless modified.

OPTION LCDPANEL GC9A01, OR, DC, RESET, CS [,BACKLIGHTPIN]

Initialises a IPS display using the GC9A01 controller. This supports 240 * 240 resolution.

OPTION LCDPANEL ST7920, OR, DC, RESET

Initialises a LCD display using the ST7920 controller. This supports 128 * 64 resolution. Note this display does not support a chip select so the SPI bus cannot be shared if this display is used.

I²C Based LCD Panels

The I²C based display controllers use the SYSTEM I2C pins as per the pinout for the specific device. Other I²C devices can share the bus subject to their addresses being unique. To setup the system I²C bus use the command: **OPTION SYSTEM I2C sdapin, sclpin**

选项 LCD面板 N5110, 或, 直流, 重置, CS [, 对比度]

使用诺基亚 5110 控制器初始化一个液晶显示器。这支持 84 * 48 的分辨率。可以指定一个额外的参数‘对比度’来控制显示的对比度。尝试在 &HA8 和 &HD0 之间的对比度值以适应您的显示器，
默认值如果省略为 &HB1

选项 LCD面板 SSD1306SPI, 或, DC, RESET, CS [, offset]

使用 SSD1306 控制器和 SPI 接口初始化 OLED 显示器。这支持 128 * 64 的分辨率。
可以指定一个额外的参数‘偏移量’来控制显示的位置。0.96" 的显示器通常需要值为 0。1.3" 的显示器通常需要值为 2。默认值如果省略为 0。

选项 LCD面板 SSD1331, 或, DC, RESET, CS [, 背光引脚]

使用 SSD1331 控制器初始化彩色 OLED 显示器。这支持 96 * 64 的分辨率。

选项 LCD面板 ST7735, 或, DC, RESET, CS [, 背光引脚]

使用ST7735控制器初始化TFT显示器。此显示器支持160 * 128分辨率。

选项 LCD面板 ST7735S, 或, 直流, 重置, CS [, 背光引脚]

使用ST7735S控制器初始化IPS显示器。此显示器支持160 * 80分辨率。

选项 LCD面板 ST7735S_W, 或, 直流, 重置, CS [, 背光引脚]

初始化Waveshare 128x128 ST7735S显示器。此显示器支持128 * 128分辨率。

选项 LCD面板 ST7789, 或, 直流, 重置, CS [, 背光引脚]

使用7789控制器初始化IPS显示器。此显示器支持240 * 240分辨率。

注意：当前不支持没有CS引脚的显示板在PicoMite上，除非经过修改。

选项 LCD面板 ST7789_135, 或, 直流, 重置, CS [, 背光引脚]

使用7789控制器初始化IPS显示器。此显示器支持240 * 135分辨率。

注意：当前不支持没有CS引脚的显示板在PicoMite上，除非经过修改。

选项 LCD面板 ST7789_320, 或, DC, RESET, CS [, 背光引脚]

使用7789控制器初始化IPS显示器。此类型支持来自Waveshare的320*240分辨率显示器 (<https://www.waveshare.com/wiki/Pico-ResTouch-LCD-2.8>)。

这些能够支持透明文本，并将与BLIT和BLIT READ命令一起工作。

注意：当前不支持没有CS引脚的显示板在PicoMite上，除非经过修改。

选项 LCD面板 GC9A01, 或, DC, RESET, CS [, 背光引脚]

使用GC9A01控制器初始化IPS显示器。此显示器支持240*240分辨率。

选项 LCD面板 ST7920, 或, DC, RESET

使用ST7920控制器初始化液晶显示器。此显示器支持128*64分辨率。注意，此显示器不支持芯片选择，因此如果使用此显示器，SPI总线无法共享。

基于I²C的液晶面板

基于I²C的显示控制器使用特定设备引脚图中的SYSTEM I2C引脚。其他I²C设备可以共享总线，前提是它们的地址是唯一的。要设置系统I²C总线，请使用命令：OPTION SYSTEM I2C sdapin, sc1pin

If an I²C display is configured it will not be necessary to "open" the I²C port for an additional device (I2C OPEN), I2C CLOSE is blocked, and all I²C devices must be capable of 100KHz operation. The I²C bus speed is not affected by changes to the CPU clock speed

These panels are configured using the following commands. In all commands the parameters OR is the orientation of the display and it can be LANDSCAPE, PORTRAIT, RLANDSCAPE or RPORTRAIT. These can be abbreviated to L, P, RL or RP. The R prefix indicates the reverse or "upside down" orientation.

OPTION LCDPANEL SSD1306I2C, OR [,offset]

Initialises a OLED display using the SSD1306 controller with an I²C interface. This supports 128 * 64 resolution. An additional parameter offset may be specified to control the position of the display. 0.96" displays typically need a value of 0. 1.3" displays typically need a value of 2. Default if omitted is 0.

NB many cheap I²C versions of SSD1306 displays do not implement I²C properly due to a wiring error. This seems to be particularly the case with 1.3" variants

OPTION LCDPANEL SSD1306I2C32, OR

Initialises a OLED display using the SSD1306 controller with an I²C interface. This supports 128 * 32 resolution.

8-bit Parallel LCD Panels

In addition to the SPI and I²C based controllers the PicoMite supports LCD displays using the SSD1963 controller (as illustrated) and the ILI9341 controller.

These use a parallel interface, are available in sizes from 2.8" to 9" and have better specifications than the smaller displays. All these displays have an SD Card socket which is fully supported by MMBasic on the PicoMite.

On eBay you can find suitable displays by searching for the controller's name (eg SSD1963).

Because they use a parallel interface the PicoMite can transfer data much faster than an SPI interface resulting in a very quick screen update.

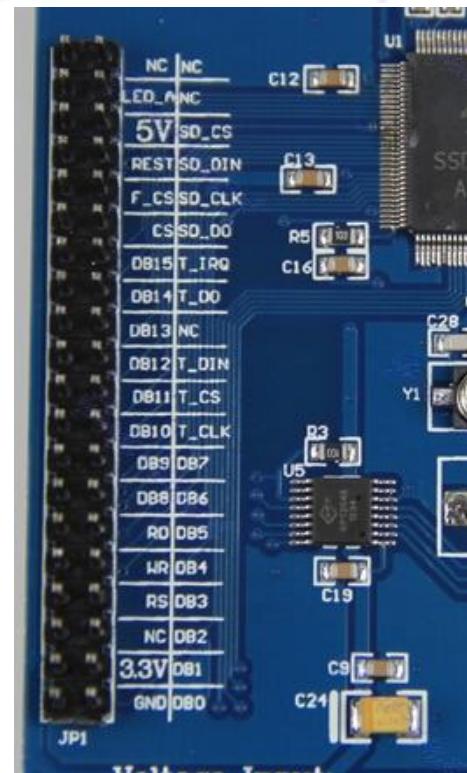
The SSD1963 displays in particular are also much larger, have more pixels and are brighter. MMBasic can drive some of them using 24-bit true colour for a full colour rendition (16 million colours).

The characteristics of these displays are:

- A 2.8, 3.2, 4.3, 5, 7, 8 or 9 inch display
- Resolution of 320 x 240, 480 x 272 pixels (4.3" version) or 800 x 480 pixels (5", 7", 8" or 9" versions).
- An SSD1963 display controller or ILI9341 display controller with a parallel interface (8080 format)
- A touch controller (SPI interface).
- A full sized SD Card socket.

There are a number of different designs using the SSD1963 controller but fortunately most Chinese suppliers have standardised on a single connector as illustrated on the right.

It is strongly recommended that any display purchased has a matching connector – this provides some confidence that the manufacturer has followed the standard that the PicoMite is designed to use.



如果配置了I²C显示器，则不需要为其他设备“打开” I²C端口（I²C OPEN），I²C CLOSE被阻止，所有I²C设备必须能够支持100KHz的操作。I²C总线速度不受CPU时钟速度变化的影响

这些面板使用以下命令进行配置。在所有命令中，参数OR是显示器的方向，可以是LANDSCAPE、PORTRAIT、RLANDSCAPE或RPORTRAIT。这些可以缩写为L、P、RL或RP。R前缀表示反向或“倒置”的方向。

选项 LCD面板 SSD1306I2C，或 [,offset]

使用SSD1306控制器初始化OLED显示器，采用 I²C接口。支持128 * 64分辨率。

可以指定一个额外的参数offset来控制显示器的位置。0.96"的显示器通常需要值为0。1.3"的显示器通常需要值为2。如果省略，默认值为0。

注意，许多便宜的I²C版本的SSD1306显示器由于接线错误而未正确实现I²C。这似乎特别适用于1.3"的变体

选项 LCD面板 SSD1306I2C32，或

使用SSD1306控制器初始化OLED显示器，采用 I²C接口。支持128 * 32分辨率。

8位并行LCD面板

除了基于SPI和I²C的控制器外，PicoMite还支持使用SSD1963控制器（如图所示）和ILI9341控制器的LCD显示器。

这些使用并行接口，尺寸从2.8英寸到9英寸可选，规格优于较小的显示器。所有这些显示器都有一个SD卡插槽，完全支持PicoMite上的MMBasic。

在eBay上，你可以通过搜索控制器的名称（例如SSD1963）找到合适的显示器。

由于它们使用并行接口，PicoMite可以比SPI接口更快地传输数据，从而实现非常快速的屏幕更新。

特别是SSD1963显示器也更大，像素更多，亮度更高。MMBasic可以使用24位真彩色驱动其中一些显示器，实现全色彩呈现（1600万种颜色）。

这些显示器的特点是：

- 2.8、3.2、4.3、5、7、8或9英寸显示器
- 分辨率为320 x 240、480 x 272像素（4.3"版本）或800 x 480像素（5"、7"、8"或9"版本）。
- 一个SSD1963显示控制器或ILI9341显示控制器具有并行接口（8080格式）
- 一个触摸控制器（SPI接口）。
- 一个全尺寸SD卡插槽。

使用SSD1963控制器的不同设计有很多，但幸运的是，大多数中国供应商已经标准化为一个单一的连接器，如右侧所示。

强烈建议购买的任何显示器都具有匹配的连接器——这提供了一定的信心，表明制造商遵循了PicoMite设计所使用的标准。



Connecting an 8-bit parallel LCD Panel

The controller uses a parallel interface while the touch controller and SD Card use an SPI interface. The touch and SD Card features are optional but if they are used they will use the second SPI port (SPI2).

The following table lists the connections required between the display board and the PicoMite to support the 8-bit parallel interface and the LCD display. The touch controller and SD Card interfaces are listed further below.

8-bit parallel Display	Description	PicoMite
DB0	Parallel Data Bus bit 0	Pin 1/GP0
DB1	Parallel Data Bus bit 1	Pin 2/GP1
DB2	Parallel Data Bus bit 2	Pin 4/GP2
DB3	Parallel Data Bus bit 3	Pin 5/GP3
DB4	Parallel Data Bus bit 4	Pin 6/GP4
DB5	Parallel Data Bus bit 5	Pin 7/GP5
DB6	Parallel Data Bus bit 6	Pin 9/GP6
DB7	Parallel Data Bus bit 7	Pin 10/GP7
CS	Chip Select (active low)	Ground (ie, always selected)
WR	Write (active low)	Pin 19/GP14*
RD	Read (active low)	Pin 20/GP15*
DC	Command/Data	Pin 17/GP13*
RESET	Reset the SSD1963	Pin 21/GP16*
LED_A	Backlight control for an unmodified display panel	Configurable see OPTION LCDPANEL
5V	5V power for the backlight on some displays (most displays use the 3.3V supply for this).	
3.3V	Power supply.	
GND	Ground	

* Pins DC, WR, RD, RESET can be allocated to other pins as a block of 4 using the optional parameter DCpin

The following table lists the connections required to support the touch controller interface:

8-bit parallel Display	Description	PicoMite
T_CS	Touch Chip Select	Recommend Pin 24/GP18
T_IRQ	Touch Interrupt	Recommend Pin 25/GP19
T_DIN	Touch Data In (MOSI)	Recommend Pin 15/GP11
T_CLK	Touch SPI Clock	Recommend Pin 14/GP10
T_DO	Touch Data Out (MISO)	Recommend Pin 16/GP12

The following table lists the connections required to support the SD Card connector:

8-bit parallel Display	Description	PicoMite
SD_CS	SD Card Chip Select	Recommend Pin 29/GP22
SD_DIN	SD Card Data In (MOSI)	Recommend Pin 15/GP11
SD_CLK	SD Card SPI Clock	Recommend Pin 14/GP10
SD_DO	SD Card Data Out (MISO)	Recommend Pin 16/GP12

连接8位并行液晶面板

控制器使用并行接口，而触摸控制器和SD卡使用SPI接口。触摸和SD卡功能是可选的，但如果使用，它们将使用第二个SPI端口（SPI2）。

下表列出了显示板与PicoMite之间所需的连接，以支持8位并行接口和液晶显示。触摸控制器和SD卡接口在下面进一步列出。

8位并行显示	描述	PicoMite
DB0	并行数据总线位 0	引脚1/GP0
DB1	并行数据总线位 1	引脚2/GP1
DB2	并行数据总线位 2	引脚4/GP2
DB3	并行数据总线位 3	引脚5/GP3
DB4	并行数据总线位 4	引脚6/GP4
DB5	并行数据总线位5	引脚7/GP5
DB6	并行数据总线位6	引脚9/GP6
DB7	并行数据总线位7	引脚10/GP7
CS	芯片选择（低电平有效）	接地（即，始终被选择）
WR	写入（低电平有效）	引脚19/GP14*
RD	读取（低电平有效）	引脚 20/GP15*
直流	命令/数据	引脚 17/GP13*
重置	重置 SSD1963	引脚 21/GP16*
LED_A	未修改显示面板的背光控制	可配置选项见 OPTION LCD面板
5V	某些显示器的背光电源为5V（大多数显示器使用3.3V电源）。	
3.3V	电源供应。	
接地	接地	

*引脚 DC, WR, RD, RESET 可以作为一个4个引脚的块分配到其他引脚，使用可选参数 DCpin

下表列出了支持触摸控制器接口所需的连接：

8位并行显示	描述	PicoMite
T_CS	触摸芯片选择	推荐引脚 24/GP18
T_IRQ	触摸中断	推荐引脚 25/GP19
T_DIN	触摸数据输入 (MOSI)	推荐引脚 15/GP11
T_CLK	触摸 SPI 时钟	推荐引脚 14/GP10
T_DO	触摸数据输出 (MISO)	推荐引脚 16/GP12

下表列出了支持 SD 卡连接器所需的连接：

8位并行显示	描述	PicoMite
SD_CS	SD 卡芯片选择	推荐引脚 29/GP22
SD_DIN	SD卡数据输入 (MOSI)	推荐引脚 15/GP11
SD_CLK	SD卡SPI时钟	推荐引脚 14/GP10
SD_DO	SD卡数据输出 (MISO)	推荐引脚 16/GP12

Where a PicoMite connection is listed as "Recommend" the specific pin should be specified in the appropriate OPTION command (see below).

Generally 7 inch and larger displays have a separate pin on the connector (marked 5V) for powering the backlight from a 5V supply. If this pin is not provided the backlight power will be drawn from the 3.3V pin. Note that the power drawn by the backlight can be considerable. For example, a 7 inch display will typically draw 330 mA from the 5V pin.

The current drawn by the backlight can cause a voltage drop on the LCD display panel's ground pin which can in turn shift the logic levels as seen by the display controller resulting in corrupted colours or text. An easy way of diagnosing this effect is to reduce the CPU speed to (say) 48MHz. If this fixes the problem it is a strong indication that this is the cause. Soldering power and ground wires direct to the LCD display panel's PCB is one workaround.

Care must be taken with display panels that share the SPI port between a number of devices (SD Card, touch, etc). In this case all the Chip Select signals must be configured in MMBasic or disabled by a permanent connection to 3.3V. If this is not done the pin will float causing the wrong controller to respond to commands on the SPI bus.

On the PicoMite either SPI channel can be used to communicate with the touch controller and the SD Card interface as defined by the OPTION SYSTEM SPI setting. If this is set, that SPI channel will be unavailable to BASIC programs (which can use the other SPI channel).

Configuring an 8-bit parallel LCD Panel

To use the display MMBasic must be configured using the OPTION LCDPANEL command which is normally entered at the command prompt. Every time the PicoMite is restarted MMBasic will automatically initialise the display.

The syntax is:

```
OPTION LCDPANEL controller, orientation [,backlightpin] [,DCpin]
```

Where 'controller' can be either:

- SSD1963_4 For a 4.3 inch display
- SSD1963_5 For a 5 inch display
- SSD1963_5A For an alternative version of the 5 inch display if SSD1963_5 does not work
- SSD1963_7 For a 7 inch display
- SSD1963_7A For an alternative version of the 7 inch display if SSD1963_7 does not work.
- SSD1963_8 For 8 inch or 9 inch displays.
- ILI9341_8 For a 2.8" or 3.2" display

'orientation' can be LANDSCAPE, PORTRAIT, RLandscape or RPORTRAIT. These can be abbreviated to L, P, RL or RP. The R prefix indicates the reverse or "upside down" orientation.

'DCpin' is optional and is the Data/Command pin (previously called the RS pin). If this parameter is omitted the pin assignment will be as above in the table. If it is specified then DC, WR, RD and RESET pins will be assigned sequentially from DC pin.

This command only needs to be run once. From then on MMBasic will automatically initialise the display on startup or reset. In some circumstances it may be necessary to interrupt power to the LCD panel while the PicoMite is running (eg, to save battery power) and in that case the GUI RESET LCDPANEL command can be used to reinitialise the display.

If the LCD panel is no longer required the command OPTION LCDPANEL DISABLE can be used which will return the I/O pins for general use.

To verify the configuration you can use the command OPTION LIST to list all options that have been set including the configuration of the LCD panel.

To test the display you can enter the command GUI TEST LCDPANEL. You should see an animated display of colour circles being rapidly drawn on top of each other. Press the space bar on the console's keyboard to stop the test.

8 and 9 inch Displays

The controller configuration SSD1963_8 has only been tested with the 8 and 9 inch displays made by EastRising (available at www.buydisplay.com). These must be purchased as a TFT LCD panel with 8080 interface, 800x480 pixel LCD, SSD1963 display controller and XPT2046 touch controller. Note that the EastRising panels use a non-standard interface connector pin-out so you will need to refer to their data sheets when connecting these to the

在PicoMite连接被列为“推荐”的地方，特定引脚应在相应的选项命令中指定（见下文）。

通常，7英寸及更大显示器在连接器上有一个单独的引脚（标记为5V），用于从5V电源为背光供电。如果没有提供此引脚，背光电源将从3.3V引脚获取。请注意，背光所需的电流可能相当大。例如，7英寸显示器通常会从5V引脚抽取330mA的电流。

背光所抽取的电流可能会导致液晶显示面板的接地引脚出现电压下降，这可能会导致显示控制器看到的逻辑电平发生变化，从而导致颜色或文本损坏。诊断此效应的一个简单方法是将CPU速度降低到（例如）48MHz。如果这解决了问题，这强烈表明这是原因所在。将电源和接地线直接焊接到液晶显示面板的PCB上是一种解决方法。

对于共享SPI端口的显示面板（如SD卡、触摸等），必须小心。

在这种情况下，所有的芯片选择信号必须在MMBasic中配置，或者通过永久连接到3.3V来禁用。

如果不这样做，引脚将处于浮动状态，导致错误的控制器响应SPI总线上的命令。在PicoMite上，可以使用任一SPI通道与触摸控制器和SD卡接口进行通信，具体由OPTION SYSTEM SPI设置定义。如果设置了该选项，则该SPI通道将无法用于BASIC程序（可以使用其他SPI通道）。

配置8位并行液晶面板

要使用显示器，必须使用OPTION LCDPANEL命令配置MMBasic，该命令通常在命令提示符下输入。每次重新启动PicoMite时，MMBasic将自动初始化显示器。

语法为：

```
OPTION LCDPANEL controller, orientation [,backlightpin] [,DCpin]
```

其中'controller'可以是：

- SSD1963_4 用于4.3英寸显示器
- SSD1963_5 用于5英寸显示器
- SSD1963_5A 用于5英寸显示器的替代版本，如果SSD1963_5无法工作
- SSD1963_7 用于7英寸显示器
- SSD1963_7A 用于7英寸显示器的替代版本，如果SSD1963_7无法工作。
- SSD1963_8 用于8英寸或9英寸显示器。
- ILI9341_8 用于2.8英寸或3.2英寸显示器

'orientation'可以是LANDSCAPE、PORTRAIT、RLANDSCAPE或RPORTRAIT。这些可以缩写为L、P、RL或RP。R前缀表示反向或“倒置”的方向。

‘DC引脚是可选的，它是数据/命令引脚（之前称为RS引脚）。如果省略此参数，则引脚分配将如上表所示。如果指定了该参数，则DC、WR、RD和RESET引脚将按顺序从DC引脚分配。

此命令只需运行一次。从那时起，MMBasic将在启动或重置时自动初始化显示。在某些情况下，可能需要在PicoMite运行时中断LCD面板的电源（例如，为了节省电池电量），在这种情况下，可以使用GUI RESET LCDPANEL命令重新初始化显示。

如果不再需要LCD面板，可以使用命令OPTION LCDPANEL DISABLE，这将使I/O引脚返回用于一般用途。

要验证配置，可以使用命令OPTION LIST列出所有已设置的选项，包括LCD面板的配置。

要测试显示器，您可以输入命令 GUI TEST LCDPANEL。您应该会看到一个动画显示，颜色圆圈快速叠加绘制。按下控制台键盘上的空格键以停止测试。

8英寸和9英寸显示器

控制器配置SSD1963_8仅在东升（EastRising）制造的8英寸和9英寸显示器上进行了测试（可在www.buydisplay.com购买）。这些必须作为具有8080接口的TFT LCD面板、800x480像素LCD、SSD1963显示控制器和XPT2046触摸控制器进行购买。请注意，东升面板使用非标准接口连接器引脚排列，因此在将其连接到

PicoMite. A suitable adapter to convert to the standard 40-pin connection can be purchased from:
<https://www.rictech.nz/micromite-products>

Backlight Control

For the ILI9163, ILI9341, ST7735, ST7735S, SSD1331, ST7789, ILI9481, ILI9488, ILI9488W, ST7789_135 ILI9341_8 and ST7789_320 displays an optional parameter ‘, backlight’ can be added to the end of the configuration parameters which specifies a pin to use to control the brightness of the backlight (LED_A). This will setup a PWM output on that pin with a frequency of 50KHz and an initial duty cycle of 99%.

You can then use the BACKLIGHT command to change the brightness between 0 and 100%. The PWM channel is blocked for normal PWM use and must not conflict with the PWM channel that may be set up for audio.

For example:

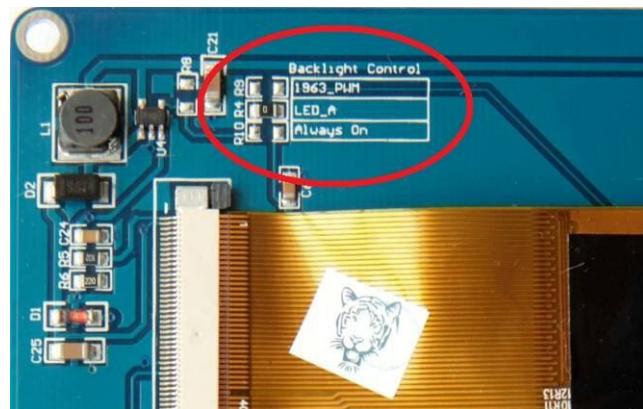
```
OPTION LCDPANEL ILI9341, OR, DC, RESET, CS, GP11
```

The backlight can then be set to 40% with this command:

```
BACKLIGHT 40
```

Most SSD1963 based LCD panels have three pairs of solder pads on the PCB which are grouped under the heading "Backlight Control" as illustrated on the right. Normally the pair marked "LED-A" are shorted together with a zero ohm resistor and this allows control of the backlight's brightness with a PWM (pulse width modulated) signal on the LED-A pin of the display panel's main connector.

However, it is better to use the SSD1963 controller to generate this signal as it frees up one I/O pin. To use the SSD1963 for brightness control the zero ohm resistor should be removed from the pair marked "LED-A" and used to short the nearby pair of solder pads marked "1963-PWM". The PicoMite can then control the brightness via the SSD1963 controller using the BACKLIGHT command.

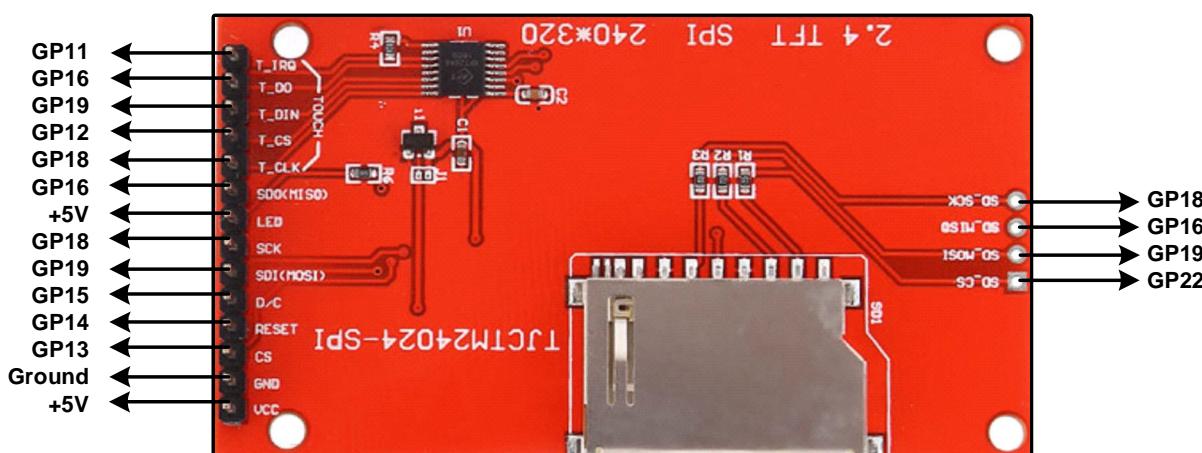


Example SPI LCD Panel Configuration

The following is a summary of how a typical LCD panel using an ILI9341 controller can be connected. This example supports the SD Card socket, the LCD display and the touch interface.

Typical panels can be found on ebay.com and similar sites by searching for the keyword “ILI9341”. Make sure that the connections on the rear of the panel resemble that shown below:

The panel should be connected to the PicoMite as illustrated:



To match the above connections the following configuration commands should be entered, one by one at the command prompt:

```
OPTION SYSTEM SPI GP18, GP19, GP16  
OPTION SDCARD GP22  
OPTION LCDPANEL ILI9341, L, GP15, GP14, GP13  
OPTION TOUCH GP12, GP11
```

PicoMite。可以从以下地址购买适配器以转换为标准40引脚连接：
<https://www.rictech.nz/micromite-products>

背光控制

对于 ILI9163、ILI9341、ST7735、ST7735S、SSD1331、ST7789、ILI9481、ILI9488、ILI9488W、ST7789_135、ILI9341_8 和 ST7789_320 显示器，可以在配置参数的末尾添加一个可选参数‘backlight’，该参数指定用于控制背光亮度的引脚（LED_A）。这将设置一个在该引脚上以50KHz频率和99%的初始占空比输出的PWM信号。

然后你可以使用BACKLIGHT命令将亮度在0%到100%之间进行调整。PWM通道被正常的PWM使用所占用，不能与可能为音频设置的PWM通道冲突。

例如：

选项 LCD面板 ILI9341，或，直流，重置，CS，GP11

然后可以使用以下命令将背光设置为40%：

BACKLIGHT 40

大多数基于SSD1963的液晶面板在PCB上有三对焊接垫，这些焊接垫被归类为“背光控制”，如右侧所示。通常标记为“LED-A”的一对焊接垫通过零欧姆电阻短接在一起，这允许通过显示面板主连接器的LED-A引脚控制背光的亮度，使用PWM（脉宽调制）信号。

然而，使用SSD1963控制器生成此信号更好，因为它释放了一个输入/输出引脚。要使用SSD1963进行亮度控制，应从标记为“LED-A”的对中移除零欧姆电阻，并

用于短接附近标记为“1963-PWM”的焊盘对。然后，PicoMite可以通过SSD1963控制器使用BACKLIGHT命令控制亮度。

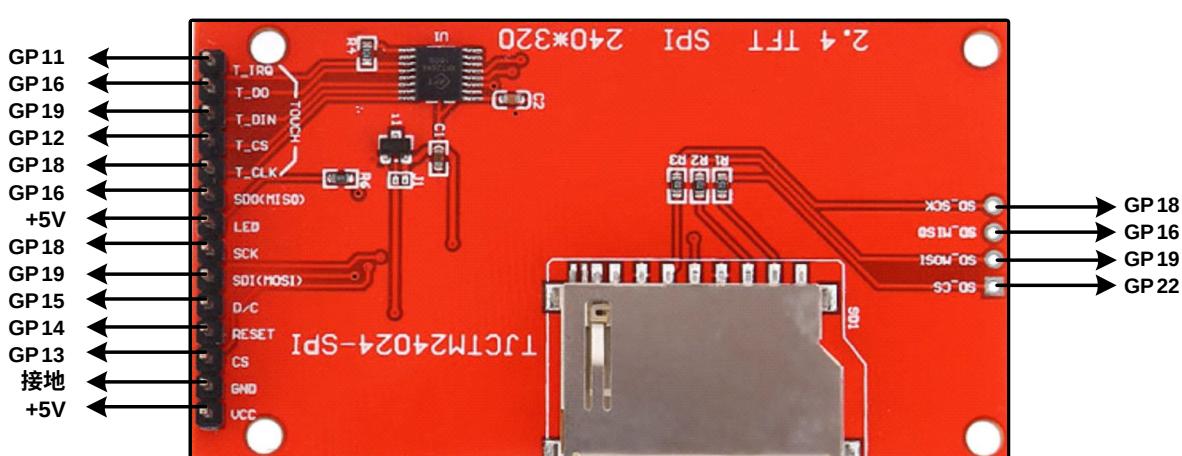


示例SPI液晶面板配置

以下是如何连接使用ILI9341控制器的典型液晶面板的总结。此示例支持SD卡插槽、液晶显示和触摸接口。

可以通过在ebay.com及类似网站上搜索关键词“ILI9341”找到典型面板。确保面板背面的连接与下面所示相似：

面板应按照下图连接到PicoMite：



为了匹配上述连接，应该在命令提示符下逐个输入以下配置命令：

```
OPTION SYSTEM SPI GP18, GP19, GP16
OPTION SDCARD GP22
OPTION LCDPANEL ILI9341, L, GP15, GP14, GP13
OPTION TOUCH GP12, GP11
```

These commands will be remembered and automatically applied on power up. Note that after each command is entered the PicoMite will restart, and the USB connection will be lost and must be reconnected.

Next the touch screen should be calibrated:

GUI CALIBRATE

You can then test the various components. The following will list the files on the SD Card, if it executes without error you can be assured that the SD Card interface is good.

FILES

The following will draw multiple colourful overlapping circles on the LCD screen which will confirm that the LCD is connected correctly:

GUI TEST LCDPANEL

Finally, the following will test the touch interface. When you touch the LCD screen a dot should appear on the screen at the exact point of the touch.

GUI TEST TOUCH

If this is not accurate you may have to run the GUI CALIBRATE command a second time taking greater care.

If you run into trouble getting the display to work it is worth disconnecting everything and clear the options with the command OPTION RESET so that you can start with a clean slate. Then reconnect it one stage at a time and configure and test each new stage as you progress. First OPTION SYSTEM SPI, then the LCD display, the touch interface and finally the SD Card.

Also note that the ILI9341 controller is sensitive to static discharge so, if the panel will not respond, it could be damaged and it would be worth testing with another panel.

这些命令将被记住，并在开机时自动应用。请注意，每输入一个命令，PicoMite 将会重启，USB 连接将会丢失，必须重新连接。

接下来，应该校准触摸屏：

GUI CALIBRATE

然后您可以测试各种组件。以下命令将列出 SD 卡上的文件，如果执行没有错误，您可以确认 SD 卡接口正常。

FILES

以下命令将在液晶屏上绘制多个重叠的彩色圆圈，以确认液晶显示器连接正确：

GUI TEST LCDPANEL

最后，以下命令将测试触摸接口。当您触摸液晶屏时，屏幕上应该会在触摸的确切位置出现一个点。

图形用户界面 测试 触摸

如果这不准确，您可能需要第二次运行 GUI 校准命令，并更加小心。

如果您在使显示器工作时遇到问题，值得断开所有连接，并使用命令选项 重置来清除选项，以便从干净的状态开始。然后逐步重新连接，并在进展中配置和测试每个新阶段。首先是选项 系统 SPI ，然后是液晶显示，触摸接口，最后是SD卡。

还要注意，ILI9341控制器对静电放电敏感，因此，如果面板没有响应，可能已损坏，值得用另一个面板进行测试。

Touch Support

Many LCD panels are supplied with a resistive touch sensitive panel and associated controller chip. MMBasic fully supports this interface on the PicoMite and this allows many of the physical knobs and switches used in a project to be implemented as on-screen controls activated by touch.

Note that the maximum voltage on all the PicoMite I/O pins is 3.3V. Level shifting will be required if a display uses 5V levels for signalling.

Configuring Touch

The touch controller on an LCD panel uses the SPI protocol for communications and this needs to be specifically configured before the panel can be configured. This is the “system” SPI port which is the port that will be used for system use (SD Card, LCD display and the touch controller on a LCD panel). This SPI port will then not be available to BASIC programs (i.e., it is reserved)

There are a number of ports and pins that can be used but these are the same as the configuration used for the example LCD panel interface previously in this manual. This command does not need to be repeated if the system SPI has already been configured:

```
OPTION SYSTEM SPI GP18, GP19, GP16
```

To use the touch facility MMBasic must be told that it is available using the OPTION TOUCH command. This should be done after the LCD display has been configured. This command tells MMBasic what pins are used for the Chip Select and Interrupt signals. For example this sets Chip Select to the GP12 pin and Interrupt to GP11:

```
OPTION TOUCH GP12, GP11
```

These commands must be entered at the command prompt and will cause the PicoMite to restart. This has the side effect of disconnecting the USB console interface which will need to be reconnected.

When the PicoMite is restarted MMBasic will automatically initialise the touch controller. To verify the configuration, you can use the command OPTION LIST to list all options that have been set including the configuration of the display panel and touch.

Note that you can use many different configurations using various pin allocations – this is just an example based on the configuration commands listed above.

Care must be taken when the SPI port is shared between a number of devices (SD Card, touch, etc). In this case all the Chip Select signals must be configured in MMBasic or alternatively disabled.

Calibrating the Touch Screen

Before the touch facility can be used it must be calibrated using the GUI CALIBRATE command.

This command will present a target in the top left corner of the screen. Using a pointy but blunt object (such as a toothpick) press exactly on the centre of the target and hold it down for at least a second. MMBasic will record this location and then continue the calibration by sequentially displaying the target in the other three corners of the screen for touch and calibration.

The calibration routine may warn that the calibration was not accurate. This is just a warning and you can still use the touch feature if you wish but it would be better to repeat the calibration using more care.

Following calibration you can test the touch facility using the GUI TEST TOUCH command. This command will blank the screen and wait for a touch. When the screen is touched a white dot will be placed on the display marking the position on the screen. If the calibration was carried out successfully the dot should be displayed exactly under the location of the stylus on the screen.

To exit the test routine you can press the space bar on the console’s keyboard.

Touch Functions

To detect if and where the screen is touched you can use the following functions in a BASIC program:

- **TOUCH(X)**
Returns the X coordinate of the currently touched location or -1 if the screen is not being touched.
- **TOUCH(Y)**
Returns the Y coordinate of the currently touched location or -1 if the screen is not being touched.

触摸支持

许多液晶面板配备了电阻式触摸敏感面板和相关控制芯片。MMBasic 完全支持 PicoMite 上的此接口，这使得项目中使用的许多物理旋钮和开关可以作为触摸激活的屏幕控件实现。

请注意，所有PicoMite I/O引脚的最大电压为3.3V。如果显示器使用5V电平进行信号传输，则需要进行电平转换。

配置触摸

液晶面板上的触摸控制器使用SPI协议进行通信，这需要在面板配置之前进行特定配置。这是“系统”SPI端口，将用于系统用途（SD卡、液晶显示器和液晶面板上的触摸控制器）。此SPI端口将不再可用于BASIC程序（即，它是保留的）

有多个端口和引脚可以使用，但这些与本手册中之前示例液晶面板接口的配置相同。如果系统SPI已经配置，则无需重复此命令：

```
OPTION SYSTEM SPI GP18, GP19, GP16
```

要使用触摸功能，必须通过OPTION TOUCH命令告知MMBasic它可用。这应该在液晶显示器配置后进行。此命令告诉MMBasic哪些引脚用于芯片选择和中断信号。例如，这将芯片选择设置为GP12引脚，中断设置为GP11：

```
OPTION TOUCH GP12, GP11
```

这些命令必须在命令提示符下输入，将导致PicoMite重启。这会导致USB控制台接口断开，需要重新连接。

当PicoMite重启时，MMBasic将自动初始化触摸控制器。要验证配置，可以使用命令OPTION LIST列出所有已设置的选项，包括显示面板和触摸的配置。

请注意，您可以使用多种不同的配置，使用各种引脚分配——这只是基于上述配置命令的一个示例。

当 SPI 端口在多个设备（SD 卡、触摸等）之间共享时，必须小心。在这种情况下，所有的芯片选择信号必须在MMBasic中配置，或者禁用。

校准触摸屏

在使用触摸功能之前，必须使用GUI CALIBRATE命令进行校准。

该命令将在屏幕的左上角显示一个目标。使用一个尖但钝的物体（例如牙签）准确按压目标的中心，并保持至少一秒钟。MMBasic将记录此位置，然后通过依次在屏幕的其他三个角落显示目标来继续校准。

校准程序可能会警告校准不准确。这只是一个警告，如果您愿意仍然可以使用触摸功能，但最好是更加小心地重复校准。

校准后，您可以使用图形用户界面中的 TEST TOUCH 命令测试触摸功能。此命令将清空屏幕并等待触摸。当屏幕被触摸时，显示器上将放置一个白点标记触摸位置。如果校准成功，白点应准确显示在屏幕上触控笔的位置下方。

要退出测试程序，您可以按控制台键盘上的空格键。

触摸函数

要检测屏幕是否被触摸以及触摸的位置，您可以在 BASIC 程序中使用以下函数：

- **TOUCH(X)**
返回当前触摸位置的 X 坐标，如果屏幕未被触摸，则返回 -1。
- **TOUCH(Y)**
返回当前触摸位置的 Y 坐标，如果屏幕未被触摸，则返回 -1。

- **TOUCH(DOWN)**
Returns true if the screen is currently being touched (this is much faster than TOUCH(X or Y)).
- **TOUCH(UP)**
Returns true if the screen is currently NOT being touched (also faster than TOUCH(X or Y))
- **TOUCH(LASTX)**
Returns the X coordinate of the last location that was touched.
- **TOUCH(LASTY)**
Returns the Y coordinate of the last location that was touched.
- **TOUCH(REF)**
Returns the reference number of the control that is currently being touched or zero if no control is being touched. See the section Advanced Graphics for more details.
- **TOUCH(LASTREF)**
Returns the reference number of the control that was last touched.

The GUI BEEP Command

The Piezo buzzer specified in the OPTION TOUCH command can also be driven by a BASIC program using the command:

```
GUI BEEP msec
```

Where 'msec' is the number of milliseconds that the beeper should be driven. A time of 3ms produces a click while 100ms produces a short beep.

Touch Interrupts with no Advanced GUI controls

An interrupt can be set on the IRQ pin number that was specified when the touch facility was configured. To detect touch down the interrupt should be configured as INTL (i.e., high to low).

For example, if the command OPTION TOUCH 7, 15 was used to configure touch the following program will print out the X and Y coordinates of any touch on the screen:

```
SETPIN 15, INTL, MyInt
DO : LOOP

SUB MyInt
    PRINT TOUCH(X) TOUCH(Y)
END SUB
```

The interrupt can be cancelled with the command SETPIN pin, OFF.

Touch Interrupts with Advanced GUI controls

When the Advanced GUI controls are activated (by setting the number of GUI controls to a non-zero number using OPTION GUI CONTROLS) the GUI INTERRUPT command is used instead to setup a touch interrupt. The syntax is:

```
GUI INTERRUPT down [, up]
```

Where 'down' is the subroutine to call when a touch down has been detected. And optionally 'up' is the subroutine to call when the touch has been lifted from the screen ('up' and 'down' can point to the same subroutine if required).

As an example, the following program will print out the X and Y coordinates of any touch on the screen:

```
GUI INTERRUPT MyInt
DO : LOOP

SUB MyInt
    PRINT TOUCH(X) TOUCH(Y)
END SUB
```

Specifying the number zero (single digit) as the argument will cancel both up and down interrupts. ie:

```
GUI INTERRUPT 0
```

- TOUCH(DOWN)
如果当前屏幕正在被触摸，则返回真（这比TOUCH(X或Y)快得多）。
- TOUCH(UP)
如果当前屏幕没有被触摸，则返回真（也比TOUCH(X或Y)快）。
- TOUCH(LASTX)
返回最后一次触摸位置的X坐标。
- TOUCH(LASTY)
返回最后一次触摸位置的Y坐标。
- TOUCH(REF)
返回当前被触摸的控件的引用编号，如果没有控件被触摸，则返回零。有关更多详细信息，请参见高级图形部分。
- TOUCH(LASTREF)
返回最后一次触摸的控件的引用编号。

图形用户界面BEEP命令

在OPTION TOUCH命令中指定的压电蜂鸣器也可以通过BASIC程序使用以下命令驱动：

```
GUI BEEP msec
```

其中'msec'是蜂鸣器应驱动的毫秒数。3毫秒的时间产生一个点击而100毫秒产生一个短音。

没有高级图形用户界面控件的触摸中断

可以在配置触摸功能时指定的IRQ引脚上设置中断。要检测触摸按下，中断应配置为INTL（即，从高到低）。

例如，如果使用命令OPTION TOUCH 7, 15配置触摸，以下程序将打印出屏幕上任何触摸的X和Y坐标：

```
SETPIN 15, INTL, MyInt
DO : LOOP

SUB MyInt
    PRINT TOUCH(X) TOUCH(Y)
END SUB
```

可以使用命令SETPIN pin, OFF取消中断。

具有高级图形用户界面控件的触摸中断

当激活高级图形用户界面控件时（通过使用OPTION GUI CONTROLS将图形用户界面控件的数量设置为非零），将使用GUI INTERRUPT命令来设置触摸中断。语法为：

图形用户界面中断 down [, up]

其中 'down' 是在检测到触摸按下时调用的子程序。可选的 'up' 是在触摸从屏幕上抬起时调用的子程序（如果需要，'up' 和 'down' 可以指向同一个子程序）。

例如，以下程序将打印出屏幕上任何触摸的 X 和 Y 坐标：

```
图形用户界面中断 MyInt
DO : 循环

子程序 MyInt
    PRINT TOUCH(X) TOUCH(Y)
END SUB
```

将数字零（单个数字）作为参数指定将取消上升和下降中断。即：

```
图形用户界面中断 0
```

Graphics Commands and Functions

Colours

Colour is specified as a true colour 24 bit number where the top eight bits represent the intensity of the red colour, the middle eight bits the green intensity and the bottom eight bits the blue. The easiest way to generate this number is with the `RGB()` function which has the form:

RGB(red, green, blue)

The `RGB()` function also supports a shortcut where you can specify common colours by naming them. For example, `RGB(red)` or `RGB(cyan)`. The colours that can be named using the shortcut form are white, black, blue, green, cyan, red, magenta, yellow, brown, white, orange, pink, gold, salmon, beige, lightgrey and grey (or USA spelling gray/lightgray).

MMBasic will automatically translate all colours to the format required by the individual display controller. For example, in the case of the ILI9341 controller, is 64K colours in the 565 format.

The default for commands that require a colour parameter can be set with the COLOUR command (can also be spelt COLOR). This is handy if your program uses a consistent colour scheme, you can then set the defaults and use the short version of the drawing commands throughout your program.

The COLOUR command takes the format:

COLOUR foreground-colour, background-colour

Fonts

There are eight built in fonts. These are:

Font Number	Size (width x height)	Character Set	Description
1	8 x 12	All 95 ASCII characters plus 7F to FF (hex)	Standard font (default on start-up).
2	12 x 20	All 95 ASCII characters	Medium sized font
3	16 x 24	All 95 ASCII characters	A larger font
4	10x16	All 95 ASCII characters plus 7F to FF (hex)	A font with extended graphic characters. Suitable for high resolution displays
5	24 x 32	All 95 ASCII characters	Extra large font, very clear
6	32 x 50	0 to 9 plus some symbols	Numbers plus decimal point, positive, negative, equals, degree and colon symbols. Very clear.
7	6 x 8	All 95 ASCII characters	A small font useful when low resolutions are used.
8	4 x 6	All 95 ASCII characters	An even smaller font.

In all fonts (including font #6) the back quote character (60 hex or 96 decimal) has been replaced with the degree symbol (°).

Font #1 (the default font) and font #4 have an extended character set covering all characters from CHR\$(32) to CHR\$(255) or 20 to FF (hex) as illustrated on the right.

If required, additional fonts can be embedded in a BASIC program. These fonts work exactly same as the built in font (i.e. selected using the FONT command or specified in the TEXT command).

图形命令和函数

颜色

颜色被指定为真实的 24 位颜色数字，其中最高的八个位表示红色的强度，中间的八个位表示绿色的强度，最低的八个位表示蓝色的强度。生成这个数字的最简单方法是使用 RGB() 函数，其形式为：

RGB(红色, 绿色, 蓝色)

RGB() 函数还支持一种快捷方式，您可以通过命名常见颜色来指定它们。例如，RGB(red) 或 RGB(cyan)。可以使用快捷方式命名的颜色有白色、黑色、蓝色、绿色、青色、红色、品红色、黄色、棕色、白色、橙色、粉色、金色、鲑鱼色、米色、浅灰色和灰色（或美国拼写的 gray/lightgray）。

MMBasic 会自动将所有颜色转换为各个显示控制器所需的格式。

例如，对于 ILI9341 控制器，使用的是 64K 颜色的 565 格式。

需要颜色参数的命令的默认值可以通过 COLOUR 命令设置（也可以拼写为 COLOR）。如果您的程序使用一致的颜色方案，这非常方便，您可以设置默认值，并在整个程序中使用绘图命令的简短版本。

COLOUR 命令的格式为：

COLOUR 前景颜色，背景颜色

字体

内置了八种字体。它们是：

字体 编号	大小 (宽度 x 高度)	字符 集	描述
1	8 x 12	所有95个ASCII字符 加上7F到FF (十六进制)	标准字体 (启动时默认)。
2	12 x 20	所有95个ASCII字符	中等大小字体
3	16 x 24	所有95个ASCII字符	较大的字体
4	10x16	所有95个ASCII字符 加上7F到FF (十六进制)	具有扩展图形字符的字体。适合 高分辨率显示
5	24 x 32	所有95个ASCII字符	超大字体，非常清晰
6	32 x 50	0到9加上一些 符号	数字加小数点，正数，负数， 等于，度和冒号符号。非常清晰。
7	6 x 8	所有95个ASCII字符	在低分辨率下使用时有用的小字体。
8	4 x 6	所有95个ASCII字符	一个更小的字体。

在所有字体（包括字体 #6）中，反引号字符（60 十六进制或 96 十进制）已被度符号（°）替换。

字体 #1（默认字体）和字体 #4 具有扩展字符集，覆盖从 CHR\$(32) 到 CHR\$(255) 或 20 到 FF（十六进制）的所有字符，如右侧所示。

如有需要，可以在 BASIC 程序中嵌入额外的字体。这些字体的工作方式与内置字体完全相同（即使用 FONT 命令选择或在 TEXT 命令中指定）。

The format of an embedded font is:

```
DefineFont #Nbr
    hex [[ hex[...]
    hex [[ hex[...
END DefineFont
```

It must start with the keyword "DefineFont" followed by the font number (which may be preceded by an optional # character). Any font number in the range of 2 to 5 and 8 to 16 can be specified and if it is the same as a built in font it will replace that font. The body of the font is a sequence of 8-digit hex words with each word separated by one or more spaces or a new line. The font definition is terminated by an "End DefineFont" keyword. These can be placed anywhere in a program and MMBasic will skip over it. This format is the same as that used by the Micromite.

Additional fonts and information can be found in the Embedded Fonts folder in the PicoMite firmware download. These fonts cover a wide range of character sets including a symbol font (Dingbats) which is handy for creating on screen icons, etc.

Read Only Variables

All coordinates and measurements on the screen are done in terms of pixels with the X coordinate being the horizontal position and Y the vertical position. The top left corner of the screen has the coordinates X=0 and Y=0 and the values increase as you move down and to the right of the screen.

There are four read only variables which provide useful information about the display currently connected:

- MM. HRES
Returns the width of the display (the X axis) in pixels.
- MM. VRES
Returns the height of the display (the Y axis) in pixels.
- MM.FONTHEIGHT
Returns the height of the current default font (in pixels). All characters in a font have the same height.
- MM.FONTWIDTH
Returns the width of a character in the current font (in pixels). All characters have the same width.

Drawing Commands

There are nine basic drawing commands that you can use within MMBasic programs on the PicoMite to interact with an attached LCD display. There is also a series of more powerful GUI commands for drawing switches, radio buttons, etc. See the next section *Advanced Graphics* for more details.

Most of the basic drawing commands have optional parameters. You can completely leave these off the end of a command or you can use two commas in sequence to indicate a missing parameter. For example, the fifth parameter of the LINE command is optional so you can use this format:

```
LINE 0, 0, 100, 100, , rgb(red)
```

Optional parameters are indicated below by italics, for example: *font*.

In the following commands C is the drawing colour and defaults to the current foreground colour. FILL is the fill colour which defaults to -1 which indicates that no fill is to be used.

The drawing commands are:

- CLS C**
Clears the screen to the colour C. If C is not specified the current default background colour will be used.
- PIXEL X, Y, C**
Illuminates a pixel. If C is not specified the current default foreground colour will be used.
- LINE X1, Y1, X2, Y2, LW, C**
Draws a line starting at X1 and Y1 and ending at X2 and Y2.
LW is the line's width and is only valid for horizontal or vertical lines. It defaults to 1 if not specified or if the line is a diagonal.
- BOX X, Y, W, H, LW, C, FILL**
Draws a box starting at X and Y which is W pixels wide and H pixels high.
LW is the width of the sides of the box and can be zero. It defaults to 1.

嵌入字体的格式为：

```
DefineFont #Nbr
    hex [[ hex[...]
    hex [[ hex[...
END DefineFont
```

它必须以关键字 "定义字体" 开头，后面跟着字体编号（可以在前面加上可选的 # 字符）。可以指定范围在 2 到 5 和 8 到 16 之间的任何字体编号，如果它与内置字体相同，则会替换该字体。字体的主体是一个由 8 位十六进制字组成的序列，每个字之间用一个或多个空格或换行符分隔。字体定义以 "结束 定义字体" 关键字结束。这些可以放置在程序的任何地方，MMBasic 会跳过它。此格式与 Micromite 使用的格式相同。

更多字体和信息可以在 PicoMite 固件下载中的嵌入式字体文件夹中找到。这些字体涵盖了广泛的字符集，包括一个符号字体（Dingbats），非常适合创建屏幕图标等。

只读变量

屏幕上的所有坐标和测量均以像素为单位，X坐标表示水平位置，Y坐标表示垂直位置。屏幕的左上角坐标为X=0和Y=0，随着向下和向右移动，值会增加。

有四个只读变量提供有关当前连接显示器的有用信息：

- MM. HRES
返回显示器的宽度（X轴）以像素为单位。
- MM. VRES
返回显示器的高度（Y轴）以像素为单位。
- MM.FONTHEIGHT
返回当前默认字体的高度（以像素为单位）。字体中的所有字符具有相同的高度。
- MM.FONTWIDTH
返回当前字体中一个字符的宽度（以像素为单位）。所有字符具有相同的宽度。

绘图命令

在PicoMite的MMBasic程序中，有九个基本绘图命令可用于与连接的液晶显示器进行交互。还有一系列更强大的图形用户界面命令用于绘制开关、单选按钮等。有关更多详细信息，请参见下一节高级图形。

大多数基本绘图命令都有可选参数。您可以完全省略这些参数，或者可以使用两个逗号连续表示缺失的参数。例如，LINE命令的第五个参数是可选的，因此您可以使用以下格式：

```
LINE 0, 0, 100, 100, , rgb(red)
```

可选参数在下面以斜体表示，例如： *font*。

在以下命令中，C是绘图颜色，默認為当前前景颜色。FILL是填充颜色，默認為-1，表示不使用填充。

绘图命令如下：

- CLS C
将屏幕清除为颜色C。如果未指定C，则将使用当前默认背景颜色。
- PIXEL X, Y, C
照亮一个像素。如果未指定C，将使用当前默认前景颜色。
- LINE X1, Y1, X2, Y2, LW, C
绘制一条从X1和Y1开始，到X2和Y2结束的线。
LW是线的宽度，仅对水平或垂直线有效。如果未指定或线是对角线，则默認為1。
- BOX X, Y, W, H, LW, C, FILL
绘制一个从X和Y开始，宽W像素，高H像素的框。
LW是框的边的宽度，可以为零。默認為1。

- **RBOX X, Y, W, H, R, C, FILL**
Draws a box with rounded corners starting at X and Y which is W pixels wide and H pixels high. R is the radius of the corners of the box. It defaults to 10.
- **CIRCLE X, Y, R, LW, A, C, FILL**
Draws a circle with X and Y as the centre and a radius R. LW is the width of the line used for the circumference and can be zero (defaults to 1). A is the aspect ratio which is a floating point number and defaults to 1. For example, an aspect of 0.5 will draw an oval where the width is half the height.
- **TEXT X, Y, STRING, ALIGNMENT, FONT, SCALE, C, BC**
Displays a string starting at X and Y. ALIGNMENT is 0, 1 or 2 characters (a string expression or variable is also allowed) where the first letter is the horizontal alignment around X and can be L, C or R for LEFT, CENTER or RIGHT aligned text and the second letter is the vertical alignment around Y and can be T, M or B for TOP, MIDDLE or BOTTOM aligned text. The default alignment is left/top. An additional code letter can be used to rotate the text (see below for the details). FONT and SCALE are optional and default to that set by the FONT command. C is the drawing colour and BC is the background colour. They are optional and default to that set by the COLOUR command.
- **GUI BITMAP X, Y, BITS, WIDTH, HEIGHT, SCALE, C, BC**
Displays the bits in a bitmap starting at X and Y. HEIGHT and WIDTH are the dimensions of the bitmap as displayed on the LCD panel and default to 8x8. SCALE, C and BC are the same as for the TEXT command. The bitmap can be an integer or a string variable or constant and is drawn using the first byte as the first bits of the top line (bit 7 first, then bit 6, etc) followed by the next byte, etc. When the top line has been filled the next line of the displayed bitmap will start with the next bit in the integer or string.
- **POLYGON n, xarray%(), yarray%() [, bordercolour] [, fillcolour]**
Draws a filled or outline polygon with n xy-coordinate pairs in xarray%() and yarray%(). If 'fillcolour' is omitted then just the polygon outline is drawn. If 'bordercolour' is omitted then it will default to the current default foreground colour.
- **ARC x, y, r1, [r2], a1, a2 [, c]**
Draws an arc of a circle with a given colour and width between two radials (defined in degrees). Parameters for the ARC command are the x and y coordinates of the centre of the arc, the inner and outer radii, the start and end angles of the arc and the colour of the arc. The zero degrees reference is at the 12 o'clock position

Rotated Text

As described above the alignment of the text in the TEXT command can be specified by using one or two characters in a string expression for the third parameter of the command. In this string you can also specify a third character to indicate the rotation of the text. This character can be one of:

- N for normal orientation
- V for vertical text with each character under the previous running from top to bottom.
- I the text will be inverted (i.e. upside down)
- U the text will be rotated counter clockwise by 90°
- D the text will be rotated clockwise by 90°

This extra feature applies in the TEXT and GUI CAPTION commands.

As an example, the following will display the text "LCD Display" vertically down the left hand margin of the display panel and centred vertically:

```
TEXT 0, 250, "LCD Display", "LMV", 5
```

Positioning is relative to the top left corner of the character when viewed normally so inverted 100,100 will have the top left pixel of the first character at 100,100 and the text will then be above y=101 and to the left of x=101. Similarly "R" in the alignment string is viewed from the perspective of the character in whatever orientation it is in (not the screen).

Transparent Text

If the display is capable of transparent text the TEXT command will allow the use of -1 for the background colour. This means that the text is drawn over the background with the background image showing through the gaps in the letters. Compatible displays use the SSD1963, ILI9341, ST7789_320, or ILI9488 with MISO connected.

- RBOX X, Y, W, H , R, C, FILL
绘制一个从X和Y开始，宽W像素，高H像素的圆角框。
R是框角的半径。默认为10。
- 圆 X, Y, R, LW, A, C, 填充
绘制一个以 X 和 Y 为中心，半径为 R 的圆。LW 是用于圆周的线宽，可以为零（默认为 1 ）。A 是宽高比，它是一个浮点数，默认为 1。例如，宽高比为 0.5 将绘制一个椭圆，宽度是高度的一半。
- 文本 X, Y, 字符串, 对齐方式, 字体, 缩放, C, BC
在 X 和 Y 开始显示一个字符串。对齐方式是 0、1 或 2 个字符（也允许字符串表达式或变量），其中第一个字母是围绕 X 的水平对齐，可以是 L、C 或 R，分别表示左对齐、居中对齐或右对齐文本，第二个字母是围绕 Y 的垂直对齐，可以是 T、M 或 B，分别表示顶部、中间或底部对齐文本。默认对齐方式为左/上。可以使用额外的代码字母来旋转文本（详见下文）。FONT 和 SCALE 是可选的，默认为 FONT 命令设置的值。C 是绘图颜色，BC 是背景颜色。它们是可选的，默认为 COLOUR 命令设置的值。
- GUI BITMAP X, Y, BITS , WIDTH, HEIGHT, SCALE, C, BC
从 X 和 Y 开始显示位图中的位。HEIGHT 和 WIDTH 是在液晶面板上显示的位图的尺寸，默认为 8x8。SCALE、C 和 BC 与 TEXT 命令相同。位图可以是整数或字符串变量或常量，并使用第一个字节作为顶部行的第一个位（先是位 7，然后是位 6，依此类推），接着是下一个字节，等等。当顶部行填满后，显示的位图的下一行将从整数或字符串中的下一个位开始。
- 多边形 n, xarray%(), yarray%() [, 边框颜色] [, 填充颜色]
绘制一个填充或轮廓多边形，使用 xarray%() 和 yarray%() 中的 n 个 xy 坐标对。如果省略‘填充颜色’，则只绘制多边形的轮廓。如果省略‘边框颜色’，则默认为当前默认前景颜色。
- 弧 x, y, r1, [r2], a1, a2 [, c]
绘制一个具有给定颜色和宽度的圆弧，位于两个径向之间（以度为单位定义）。ARC 命令的参数是弧的中心的 x 和 y 坐标，内半径和外半径，弧的起始和结束角度，以及弧的颜色。零度参考位于 12 点钟位置。

旋转文本

如上所述，TEXT 命令中文本的对齐方式可以通过在命令的第三个参数中使用一个或两个字符的字符串表达式来指定。在这个字符串中，您还可以指定第三个字符以指示文本的旋转。这个字符可以是以下之一：

N 表示正常方向

V 表示垂直文本，每个字符在前一个字符的下方，从上到下排列。

I 表示文本将被翻转（即倒置）

U 表示文本将逆时针旋转90°

D 表示文本将顺时针旋转90°

这个额外功能适用于 TEXT 和 GUI CAPTION 命令。

例如，以下代码将在显示面板的左侧边缘垂直显示文本 "LCD Display"，并垂直居中：

```
TEXT 0, 250, "LCD Display", "LMV", 5
```

定位是相对于正常视图下字符的左上角，因此倒置的 100,100 将使第一个字符的左上像素位于 100, 100，文本将位于 y=101 以上，x=101 左侧。同样，排列字符串中的“R”是从字符所处的任何方向的角度来看（而不是屏幕）。

透明文本

如果显示器支持透明文本，TEXT命令将允许使用-1作为背景颜色。这意味着文本是在背景上绘制的，背景图像通过字母间的空隙显示出来。兼容的显示器使用SSD1963、ILI9341、ST7789_320或ILI9488，并连接MISO。

BLIT Command

If the display is capable of transparent text the BLIT command allows a portion of the image currently showing on the display to be copied to a memory buffer and later copied back to the display. This is useful when something needs to be drawn over the background and later removed without damaging the image in the background. Examples include a game where a character is moving about in front of a landscape or the moving needle of a photorealistic gauge.

The available commands are:

```
BLIT READ #b, x, y, w, h  
BLIT WRITE #b, x, y, w, h  
BLIT LOAD #b, f$, x, y, w, h  
BLIT CLOSE #b
```

#b is the buffer number in the range of 1 to 32. x and y are the coordinates of the top left corner and w and h are the width and height of the image. READ will copy the display image to the buffer, WRITE will copy the buffer to the display and CLOSE will free up the buffer and reclaim the memory used. LOAD will load an image file into the buffer.

BLIT LOAD and BLIT WRITE will work on any display while BLIT and BLIT READ will only work on displays capable of transparent text (i.e. using the SSD1963, ILI9341, ST7789_320, or ILI9488 with MISO connected).

These commands can be used to copy a portion of the display to another location (by copying to a buffer then writing somewhere else) but a simpler method is to use an alternative version of the BLIT command as follows:

```
BLIT x1, y1, x2, y2, w, h
```

This will copy a portion of the image at x1/y1 to the location x2/y2. w and h specify the width and height of the image to be copied. The source and destination areas can overlap and the BLIT command will perform the copy correctly.

This form of the BLIT command is particularly useful for creating graphs that can scroll horizontally or vertically as new data is added.

Load Image

The LOAD IMAGE and LOAD JPG commands can be used to load an image from the Flash Filesystem or SD Card and display it on the LCD display. This can be used to draw a logo or add an ornate background to the graphics drawn on the display.

Example

As an example the following program will draw a simple digital clock on an ILI9341 based LCD display. The program will terminate and return to the command prompt if the display screen is touched.

First the display and touch options must be configured by entering the commands listed at the beginning of this section. The exact format of these will depend on how you have connected the display panel.

Then enter and run the program:

```
CONST DBblue = RGB(0, 0, 128)           ' A dark blue colour  
COLOUR RGB(GREEN), RGB(BLACK)          ' Set the default colours  
FONT 1, 3                             ' Set the default font  
  
BOX 0, 0, MM.HRes-1, MM.VRes/2, 3, RGB(RED), DBblue  
  
DO  
    TEXT MM.HRes/2, MM.VRes/4, TIME$, "CM", 1, 4, RGB(CYAN), DBblue  
    TEXT MM.HRes/2, MM.VRes*3/4, DATE$, "CM"  
    IF TOUCH(X) <> -1 THEN END  
LOOP
```

This program starts by defining a constant with a value corresponding to a dark blue colour and then sets the defaults for the colours and the font. It then draws a box with red walls and a dark blue interior.

BLIT命令

如果显示器支持透明文本，BLIT命令允许将当前显示在屏幕上的图像的一部分复制到内存缓冲区，然后再复制回显示器。这在需要在背景上绘制某些内容并稍后移除而不损坏背景图像时非常有用。示例包括一个角色在风景前移动的游戏或一个逼真的量规的移动指针。

可用的命令有：

```
BLIT READ #b, x, y, w, h  
BLIT WRITE #b, x, y, w, h  
BLIT LOAD #b, f$, x, y, w, h  
BLIT CLOSE #b
```

#b是缓冲区编号，范围为1到32。x和y是左上角的坐标，w和h是图像的宽度和高度。READ将显示图像复制到缓冲区，WRITE将缓冲区复制到显示器，CLOSE将释放缓冲区并回收所用的内存。LOAD将把图像文件加载到缓冲区中。

BLIT LOAD和BLIT WRITE将在任何显示器上工作，而BLIT和BLIT READ仅在能够显示透明文本的显示器上工作（即使用SSD1963、ILI9341、ST7789_320或ILI9488并连接MISO）。

这些命令可以用来将显示器的一部分复制到另一个位置（通过复制到缓冲区然后写入其他地方），但更简单的方法是使用BLIT命令的替代版本，如下所示：

```
BLIT x1, y1, x2, y2, w, h
```

这将把位于x1/y1的图像的一部分复制到位置x2/y2。w和h指定要复制的图像的宽度和高度。源区域和目标区域可以重叠，BLIT命令将正确执行复制。

这种形式的BLIT命令特别适用于创建可以水平或垂直滚动的图表，因为新数据被添加。

加载图像

LOAD IMAGE和LOAD JPG命令可以用来从Flash文件系统或SD卡加载图像并在液晶显示器上显示。这可以用来绘制徽标或为显示器上绘制的图形添加华丽的背景。

示例

作为示例，以下程序将在基于ILI9341的液晶显示器上绘制一个简单的数字时钟。如果触摸显示屏，程序将终止并返回到命令提示符。

首先，必须通过输入本节开头列出的命令来配置显示和触摸选项。这些命令的确切格式将取决于您如何连接显示面板。

然后输入并运行程序：

```
CONST D Blue = RGB(0, 0, 128)           ' 一种深蓝色  
COLOUR RGB(GREEN), RGB(BLACK)          ' 设置默认颜色  
FONT 1, 3                                ' 设置默认字体  
  
BOX 0, 0, MM.HRes -1, MM.VRes/2, 3, RGB(RED), D Blue  
  
DO  
    TEXT MM.HRes/2, MM.VRes/4, TIME $, "CM", 1, 4, RGB(CYAN), D Blue  
    TEXT MM.HRes/2, MM.VRes*3/4, DATE$, "CM"  
    IF TOUCH(X) <> -1 THEN END  
LOOP
```

该程序首先定义了一个常量，其值对应于深蓝色，然后设置颜色和字体的默认值。接着，它绘制了一个红色边框和深蓝色内部的框。

Following this the program enters a continuous loop where it performs three functions:

1. Displays the current time inside the previously drawn box. The string is drawn centred both horizontally and vertically in the middle of the box. Note that the TEXT command overrides both the default font and colours to set its own parameters.
2. Draws the date centred in the lower half of the screen. In this case the TEXT command uses the default font and colours previously set.
3. Checks for a touch on the screen. This is indicated when the TOUCH(X) function returns something other than -1. In that case the program will terminate.

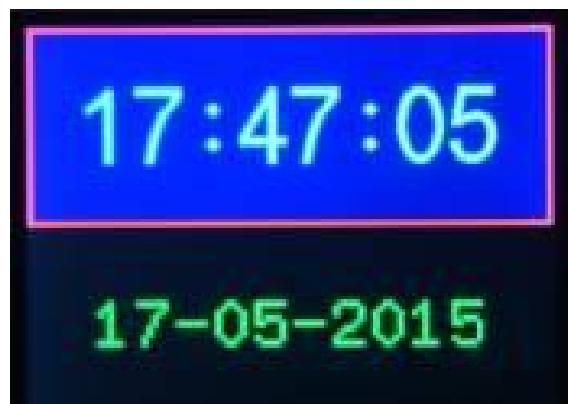
The screen display should look like this (the font used in this illustration is different):



随后，程序进入一个连续循环，执行三个函数：

1. 在之前绘制的框内显示当前时间。字符串在框的中间水平和垂直居中绘制。请注意，TEXT命令覆盖了默认字体和颜色，以设置其自己的参数。
2. 在屏幕的下半部分居中绘制日期。在这种情况下，TEXT命令使用之前设置的默认字体和颜色。
3. 检查屏幕上的触摸。当TOUCH(X)函数返回的值不是-1时，表示有触摸。在这种情况下，程序将终止。

屏幕显示应如下所示（此插图中使用的字体不同）：



PicoMite Advanced Graphics

The PicoMite incorporates a suite of advanced graphic controls that respond to touch, these include on screen switches, buttons, indicator lights, keyboard, etc. MMBasic will draw the control and animate it (i.e. a switch will appear to depress when touched). All that the BASIC program needs to do is invoke a single command to specify the basic details of the control.

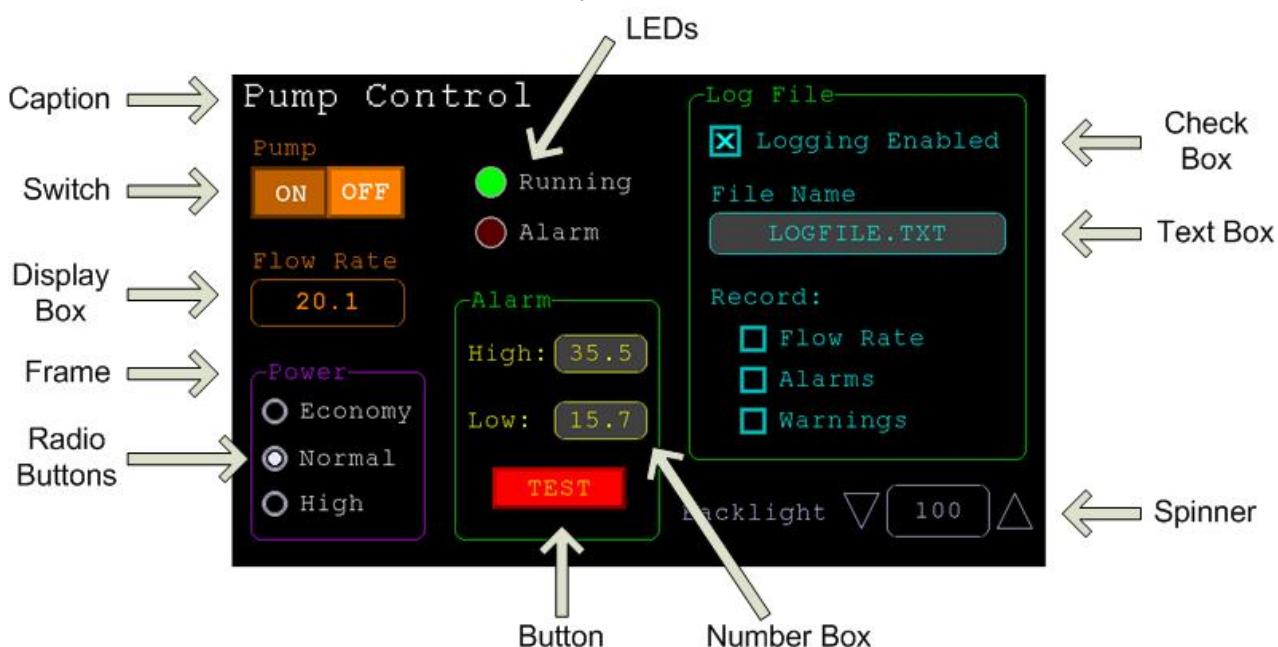
To use the GUI controls in the PicoMite the memory required for the GUI controls must be allocated first by using the command OPTION GUI CONTROLS. Typically you would use the command like this:

```
OPTION GUI CONTROLS 75
```

This will set the maximum number of controls that you can define to 75. This option is permanent (i.e. it will be remembered on power down). By default the maximum number of controls is set to zero and in this case the GUI features will not be available and no memory will be used.

Defining Controls

These are some of the advanced GUI controls that you can use:



Each control has a reference number called '#ref' in the description of the control. This can be any number between 1 and the upper limit set by the OPTION CONTROL command. This reference number is used to identify a control. For example, a check box can be created with a reference number of #10:

```
GUI CHECKBOX #10, "Test", 100, 100, 50, rgb(BLUE)
```

Once created the user can check and uncheck the box using the touch feature of the LCD panel without the running BASIC program being involved. When needed the program can determine the check box value by using its reference number in the CtrlVal() function:

```
IF CtrlVal(#10) THEN ..
```

The # character is optional but serves to remind the programmer that this is not an ordinary number.

In the following commands any arguments that are in italic font (e.g. *Width*, *Height*) are optional and if not specified will take the value of the previous command that did specify them. This means for example, that a number of radio buttons with the same size and colour can be specified with only the first button having to list all the details. Note that with the colour specification this is different to the basic drawing commands which default to the last COLOUR command.

All strings used in GUI controls and the MsgBox can display multiple lines by using the tilde character (~) to separate each line in the string. For example, a push button's caption can be "ALARM~TEST" and this would be displayed as two lines. For all controls the font used for the caption will be whatever is set with the FONT command and the colours will be whatever was set by the last COLOUR command.

If the display is capable of transparent text these commands will allow the use of -1 for the background colour. This means that the text is drawn over the background with the background image showing through the gaps in the letters.

PicoMite高级图形

PicoMite 集成了一套先进的图形控件，这些控件可以响应触摸，包括屏幕上的开关、按钮、指示灯、键盘等。MMBasic 将绘制控件并进行动画处理（即，开关在触摸时会看起来被按下）。BASIC 程序所需做的就是调用一个命令来指定控件的基本细节。

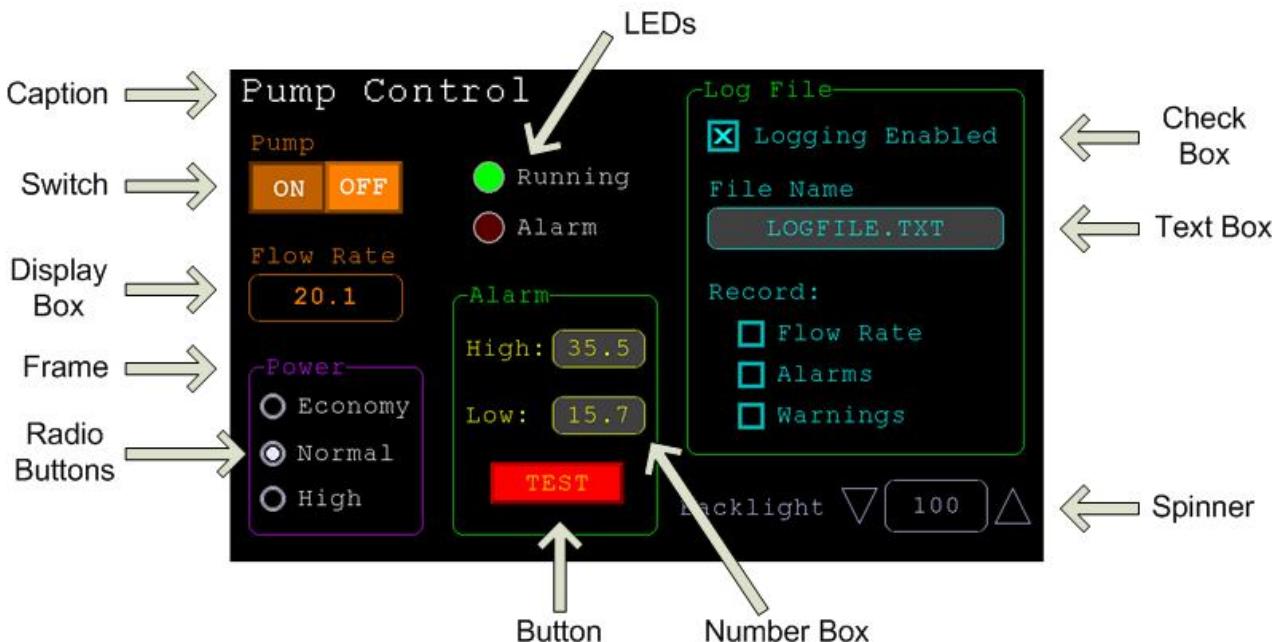
要在 PicoMite 中使用 GUI 控件，必须首先通过使用命令 OPTION GUI CONTROLS 分配所需的内存。通常你会像这样使用该命令：

```
OPTION GUI CONTROLS 75
```

这将设置你可以定义的控件的最大数量为 75。此选项是永久性的（即在断电后会被记住）。默认情况下，控件的最大数量设置为零，在这种情况下，GUI 功能将不可用，并且不会使用任何内存。

定义控件

以下是一些你可以使用的高级 GUI 控件：



每个控件都有一个参考编号，称为 '#ref'，在控件的描述中。这可以是任何数字在1和OPTION CONTROL命令设定的上限之间。这个参考编号用于识别一个控件。例如，可以创建一个参考编号为 #10 的复选框：

图形用户界面复选框 #10, "测试", 100, 100, 50, rgb(蓝色)

一旦创建，用户可以使用液晶面板的触摸功能来勾选和取消勾选该框，而无需运行的BASIC程序参与。在需要时，程序可以通过在 CtrlVal() 函数中使用其参考编号来确定复选框的值：

如果 CtrlVal(#10) 那么 ..

#字符是可选的，但提醒程序员这不是一个普通数字。

在以下命令中，任何以斜体字显示的参数（例如宽度, 高度）都是可选的，如果未指定，将采用之前指定它们的命令的值。这意味着，例如，可以指定多个具有相同大小和颜色的单选按钮，而只有第一个按钮需要列出所有细节。请注意，颜色规格与基本绘图命令不同，后者默认使用最后一个 COLOUR 命令。

所有在图形用户界面控件和消息框中使用的字符串可以通过使用波浪号字符 (~) 来显示多行，以分隔字符串中的每一行。例如，一个按钮的标题可以是 "ALARM~TEST"，这将显示为两行。对于所有控件，标题使用的字体将是通过 FONT 命令设置的字体，颜色将是最后一个 COLOUR 命令设置的颜色。

如果显示器支持透明文本，这些命令将允许使用 -1 作为背景颜色。这意味着文本是在背景上绘制的，背景图像透过字母间的空隙显示出来。

The advanced graphics controls are:

Frame

`GUI FRAME #ref, caption$, StartX, StartY, Width, Height, Colour`

This will draw a frame which is a box with round corners and a caption. A frame does not respond to touch but is useful when a group of controls need to be visually brought together. It can also be used to surround a group of radio buttons and MMBasic will arrange for the radio buttons surrounded by the frame to be exclusive – that is, when one radio button is selected any other button that was selected and within the frame will be deselected.

LED

`GUI LED #ref, caption$, CenterX, CenterY, Diameter, Colour`

This will draw an indicator light (it looks like a panel mounted LED). When its value is set to one it will be illuminated and when it is set to zero it will be off (a dull version of its colour attribute). The LED can be made to flash by setting its value to the number of milliseconds that it should remain on before turning off.

The caption will be drawn to the right of the LED and will use the colours set by the COLOUR command. The LED control is not animated when touched but its reference number can be found using TOUCH(REF) and TOUCH(LASTREF) in the touch interrupts and any required animation can be done in MMBasic.

Check Box

`GUI CHECKBOX #ref, caption$, StartX, StartY, Size, Colour`

This will draw a check box which is a small box with a caption. Both the height and width are specified with the 'Size' parameter. When touched an X will be drawn inside the box to indicate that this option has been selected and the control's value will be set to 1. When touched a second time the check mark will be removed and the control's value will be zero. The caption will be drawn to the right of the Check Box and will use the colours set by the COLOUR command.

Push Button

`GUI BUTTON #ref, caption$, StartX, StartY, Width, Height, FColour, BColour`

This will draw a momentary button which is a square switch with the caption on its face. When touched the visual image of the button will appear to be depressed and the control's value will be 1. When the touch is removed the value will revert to zero. Caption can be a single string with two captions separated by a vertical bar (|) character (e.g. "UP|DOWN"). When the button is up the first string will be used and when pressed the second will be used.

Switch

`GUI SWITCH #ref, caption$, StartX, StartY, Width, Height, FColour, BColour`

This will draw a latching switch with the caption on its face. When touched the visual image of the button will appear to be depressed and the control's value will be 1. When touched a second time the switch will be released and the value will revert to zero. Caption can be a single string with two captions separated by a | character (e.g. "ON|OFF"). When this is used the switch will appear to be a toggle switch with each half of the caption used to label each half of the toggle switch.

Radio Button

`GUI RADIO #ref, caption$, CenterX, CenterY, Radius, Colour`

This will draw a radio button with a caption. When touched the centre of the button will be illuminated to indicate that this option has been selected and the control's value will be 1. When another radio button is selected the mark on this button will be removed and its value will be zero. Radio buttons are grouped together when surrounded by a frame and when one button in the group is selected all others in the group will be deselected. If a frame is not used all buttons on the screen will be grouped together.

The caption will be drawn to the right of the button and will use the colours set by the COLOUR command.

Display Box

`GUI DISPLAYBOX #ref, StartX, StartY, Width, Height, FColour, BColour`

This will draw a box with rounded corners. Any string can be displayed in the box by using the CtrlVal(r)= command. This is useful for displaying text, numbers and messages. This control is not animated when touched but its reference number can be found using TOUCH(REF) and TOUCH(LASTREF) in the touch interrupts and any required animation can be done in MMBasic.

高级图形控件如下：

帧

图形用户界面框架 #ref, 标题\$, 起始x, 起始y, 宽度, 高度, 颜色

这将绘制一个带有圆角和标题的框。帧不响应触摸，但在需要将一组控件视觉上聚集在一起时非常有用。它还可以用来围绕一组单选按钮，MMBasic将确保被框围绕的单选按钮是互斥的——也就是说，当选择一个单选按钮时，任何其他已选择且在框内的按钮将被取消选择。

LED

图形用户界面 LED #ref, 标题\$, 居中 x, 居中 y, 直径, 颜色

这将绘制一个指示灯（看起来像一个面板安装的LED）。当其值设置为1时，它将被点亮，当设置为0时，它将熄灭（其颜色属性的暗淡版本）。通过将其值设置为应保持点亮的毫秒数，可以使LED闪烁。标题将绘制在LED的右侧，并将使用COLOUR命令设置的颜色。当触摸时，LED控制不会动画，但可以使用TOUCH(REF)和TOUCH(LASTREF)在触摸中断中找到其引用编号，任何所需的动画可以在MMBasic中完成。

复选框

图形用户界面复选框 #ref, caption\$, StartX, StartY, 大小, 颜色

这将绘制一个复选框，它是一个带有标题的小框。高度和宽度都通过'大小'参数指定。当被触摸时，框内将绘制一个X，以表示该选项已被选择，控件的值将设置为1。当再次触摸时，勾选标记将被移除控件的值将为零。标题将绘制在复选框的右侧，并将使用COLOUR命令设置的颜色。

按钮

图形用户界面按钮 #ref, caption\$, StartX, StartY, 宽度, 高度, F颜色, B颜色

这将绘制一个瞬时按钮，它是一个带有标题的方形开关。当被触摸时，按钮的视觉图像将看起来被按下，控制的值将为1。当触摸被移除时，值将恢复为零。标题可以是一个单一字符串，包含两个由垂直条（|）字符分隔的标题（例如："UP|DOWN"）。当按钮处于上方时，将使用第一个字符串，当被按下时将使用第二个字符串。

开关

图形用户界面开关 #ref, caption\$, StartX, StartY, 宽度, 高度, F颜色, B颜色

这将绘制一个带有标题的锁存开关。当被触摸时，按钮的视觉图像将看起来被按下，控制的值将为1。当第二次触摸时，开关将被释放，值将恢复为零。标题可以是一个单一字符串，包含两个由|字符分隔的标题（例如："ON|OFF"）。当使用此功能时，开关将显示为一个切换开关，每一半的标题用于标记切换开关的每一半。

单选按钮

GUI RADIO #ref, caption\$, CenterX, CenterY, 半径, 颜色

这将绘制一个带有标题的单选按钮。当触摸时，按钮的中心将被照亮，以指示该选项已被选中，控制的值将为1。当选择另一个单选按钮时，此按钮上的标记将被移除，其值将为零。单选按钮在被框架包围时会被分组，当组中的一个按钮被选中时，组中的所有其他按钮将被取消选择。如果不使用框架，屏幕上的所有按钮将被分组在一起。

标题将绘制在按钮的右侧，并将使用COLOUR命令设置的颜色。

显示框

GUI DISPLAYBOX #ref, StartX, StartY, 宽度, 高度, F颜色, B颜色

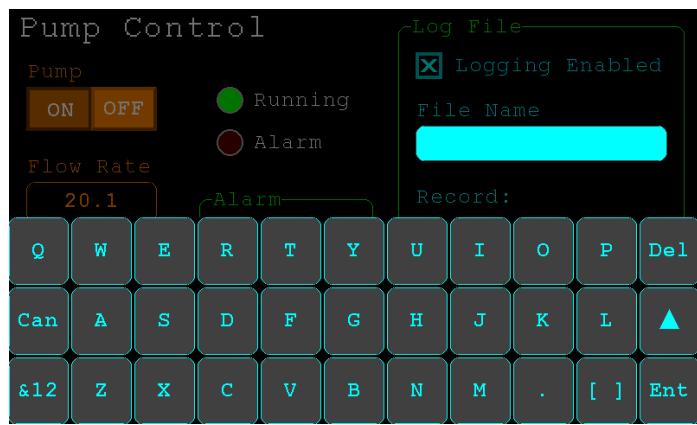
这将绘制一个带圆角的框。通过使用CtrlVal(r)=命令，可以在框中显示任何字符串。这对于显示文本、数字和消息非常有用。此控件在触摸时不会动画，但可以使用TOUCH(REF)和TOUCH(LASTREF)在触摸中断中找到其引用编号，任何所需的动画可以在MMBasic中完成。

Text Box

GUI TEXTBOX #ref, StartX, StartY, Width, Height, FColour, BColour

This will draw a box with rounded corners. When the box is touched a QWERTY keyboard will appear on the screen as shown on the right. Using this virtual keyboard any text can be entered into the box including upper/lower case letters, numbers and any other characters in the ASCII character set. The new text will replace any text previously in the box.

Ent is the enter key, Can is the cancel key and will close the text box and return it to its original state, the triangle is the shift key, the [] key will insert a space and the &12 key will select an alternate key selection with numbers and special characters (there are two sets of special characters and the shift key will switch between them).



The displayed string can be set by assigning a string to the box using the CtrlVal(r) = command. The value of the control can also be set to a string starting with two hash characters (##) and in that case the string (without the leading two hash characters) will be displayed in the box with reduced brightness. This can be used to give the user a hint as to what should be entered (called "ghost text"). Reading the value of the control displaying ghost text will return an empty string. When a key is pressed the ghost text will vanish and be replaced with the entered text.

MMBasic will try to position the virtual keyboard on the screen to not obscure the text box that caused it to appear. A pen down interrupt will be generated just before the keyboard is deployed and a key up interrupt will be generated when the Enter or Cancel keys are touched and the keyboard is hidden.

If necessary the virtual keyboard can be dismissed by the program (same as touching the cancel button) with the command: GUI TEXTBOX CANCEL.

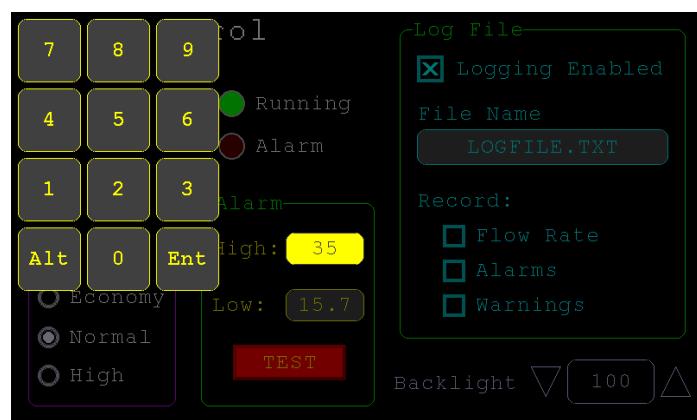
Number Box

GUI NUMBERBOX #ref, StartX, StartY, Width, Height, FColour, BColour

This will draw a box with rounded corners. When the box is touched a numeric keypad will appear on the screen as shown on the right. Using this virtual keypad any number can be entered into the box including a floating point number in exponential format. The new number will replace the number previously in the box.

The Alt key will select an alternative key selection and the other special keys are the same as with the text box.

The displayed number can also be set by assigning a number (float or integer) to the box using the CtrlVal(r) = command.



Similar to the Text Box, the value of the control can set to a literal string with two leading hash characters (e.g. "#Hint") and in that case the string (without the leading two characters) will be displayed in the box with reduced brightness. Reading this will return zero and when a key is pressed the ghost text will vanish.

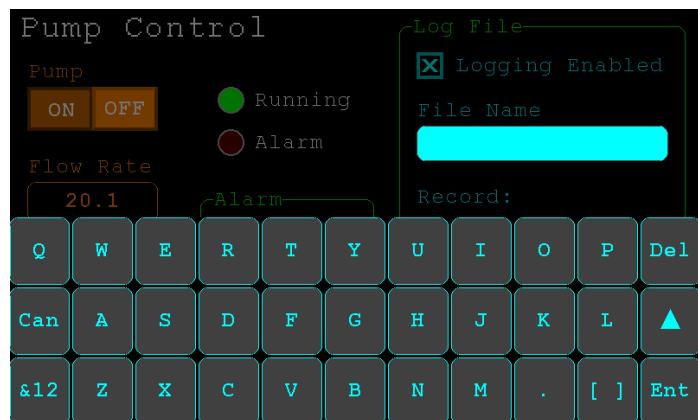
MMBasic will try to position the virtual keypad on the screen to not obscure the number box that caused it to appear. A pen down interrupt will be generated just before the keypad is deployed and a key up interrupt will be generated when the Enter key is touched and the keypad is hidden. Also, when the Enter key is touched the entered text will be evaluated as a number and the NUMBERBOX control redrawn to display this number.

文本框

图形用户界面文本框 #ref, StartX, StartY, 宽度, 高度, F颜色, B颜色

这将绘制一个带圆角的框。当框被触摸时，Q WERTY 键盘将出现在屏幕上，如右侧所示。使用这个虚拟键盘，可以在框中输入任何文本，包括大小写字母、数字和 ASCII 字符集中的其他字符。新文本将替换框中之前的任何文本。

Ent 是回车键，Can 是取消键，将关闭文本框并返回到其原始状态，三角形是 Shift 键，[] 键将插入一个空格，&12 键将选择一个包含数字和特殊字符的备用键选择（有两组特殊字符，Shift 键将在它们之间切换）。



可以通过使用 `CtrlVal(r) =` 命令将字符串分配给文本框来设置显示的字符串。控件的值也可以设置为以两个井号字符 (##) 开头的字符串，在这种情况下，字符串（去掉前两个井号字符）将在文本框中以降低的亮度显示。这可以用来给用户提示应该输入什么（称为“幽灵文本”）。读取显示幽灵文本的控件的值将返回一个空字符串。当按下一个键时，幽灵文本将消失，并被输入的文本替换。

MMBasic 将尝试将虚拟键盘定位在屏幕上，以不遮挡导致其出现的文本框。在键盘部署之前，将生成一个笔下中断，当按下 Enter 或 Cancel 键并隐藏键盘时，将生成一个按键抬起中断。

如果需要，程序可以通过命令：`GUI TEXTBOX CANCEL` 来关闭虚拟键盘（与触摸取消按钮相同）。

数字框

图形用户界面数字框 #ref, StartX, StartY, 宽度, 高度, F颜色, B颜色

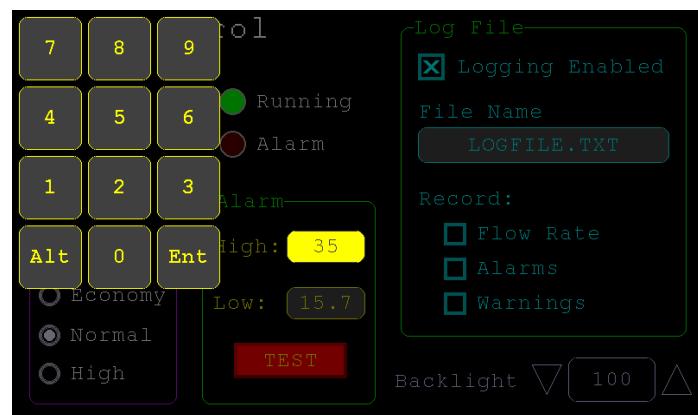
这将绘制一个带圆角的框。当框被触摸时，屏幕上将出现一个数字键盘，如右侧所示。使用这个虚拟键盘，可以在框中输入任何数字，包括以指数格式表示的浮点数。新输入的数字将替换框中之前的数字。

Alt 键将选择替代的键选择，其他特殊键与文本框相同。

显示的数字也可以通过使用 `CtrlVal(r) =` 命令将一个数字（浮点或整数）赋值给框来设置。

与文本框类似，控件的值可以设置为带有两个前导哈希字符的字面字符串（例如 "#Hint"），在这种情况下，字符串（去掉前导的两个字符）将以降低亮度的方式显示在框中。读取此内容将返回零，当按下一个键时，幽灵文本将消失。

MMBasic 将尝试在屏幕上定位虚拟键盘，以不遮挡导致其出现的数字框。在键盘部署之前将生成一个笔下中断，当触摸回车键并隐藏键盘时，将生成一个按键抬起中断。此外，当触摸回车键时，输入的文本将被求值为一个数字，并且NUMBERBOX控件将重新绘制以显示该数字。



Formatted Number Box

GUI FORMATBOX #ref, Format, StartX, StartY, Width, Height, FColour, BColour

This will draw a box with rounded corners. When the box is touched a numeric keypad will appear similar to a Number Box. The difference is that the Formatted Number Box will require the user to enter numbers according to a specific format for dates, time, etc. Invalid keys on the keypad will be disabled and the user will be guided in their entry with guide text. This means that the programmer can be assured that the entry made by the user will always be in a fixed format.

The type of entry is controlled by the 'Format' argument as follows:

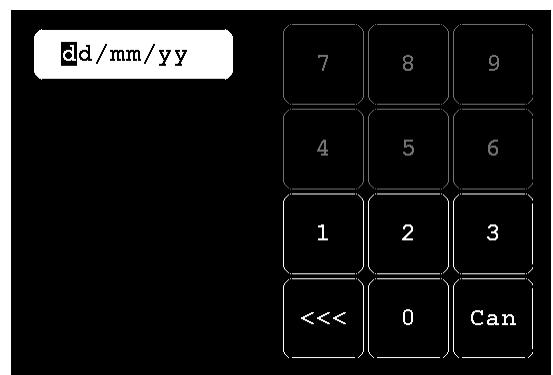
DATE1	Date in UK/Aust/NZ format (dd/mm/yy)
DATE2	Date in USA format (mm/dd/yy)
DATE3	Date in international format (yyyy/mm/dd)
TIME1	Time in 24 hour notation (hh:mm)
TIME2	Time in 24 hour notation with seconds (hh:mm:ss)
TIME3	Time in 12 hour notation (hh:mm AM/PM)
TIME4	Time in 12 hour notation with seconds (hh:mm:ss AM/PM)
DATETIME1	Both date (UK fmt) and time (12 hour) (dd/mm/yy hh:mm AM/PM)
DATETIME2	Both date (UK fmt) and time (24 hour) (dd/mm/yy hh:mm)
DATETIME3	Both date (USA fmt) and time (12 hour) (mm/dd/yy hh:mm AM/PM)
DATETIME4	Both date (USA fmt) and time (24 hour) (mm/dd/yy hh:mm)
LAT1	Latitude in degrees, minutes and seconds (d° mm' ss" N/S)
LAT2	Latitude with seconds to one decimal place (dd° mm' ss.s" N/S)
LONG1	Longitude in degrees, minutes and seconds (ddd° mm' ss" E/W)
LONG2	Longitude with seconds to one decimal place (ddd° mm' ss.s" E/W)
ANGLE1	Angle in degrees and minutes (ddd° mm')

For example:

```
GUI FORMATBOX #1, DATE1, 300, 150, 200, 50
```

would create a data entry box and when it is touched a keypad will appear as shown on the right . Note that:

- The display box is filled with a guide string to prompt the user as to the data required.
- Because the day of the month can only start with a digit from 0 to 3 all other keys are disabled. This also happens with other numbers that have a limited range.
- The value of the control retrieved via CtrlVal(#1) is a string. As an example, if the user entered the date for the 8th of May 2020 the returned string would be "08/05/20" (i.e. the UK/Aust/NZ format as specified by DATE1).



The value of the control can be pulled apart using the string functions or, in some cases, the string can be used directly. For example, if using the above format box to get a date from the user the PicoMite's internal clock could then be directly set as follows:

```
DATE$ = CtrlVal(#1)
```

The RTC SETTIME command will accept a single string argument in the format of dd/mm/yy hh:mm so similarly the RTC time could be set as follows if the formatted box used DATETIME2 for 'Format':

```
RTC SETTIME CtrlVal(#1)
```

You can use the USA style DATETIME4 to get the date/time. In that case you would use this to set the RTC:

```
RTC SETTIME MID$(CtrlVal(#1),4,3) + LEFT$(CtrlVal(#1),2) + RIGHT$(CtrlVal(#1),9)
```

MMBasic will try to position the virtual keypad on the screen so as to not obscure the format box that caused it to appear. A pen down interrupt will be generated when the keypad is deployed and a key up interrupt will be generated when all the required data has been entered and the keypad is hidden.

格式化数字框

图形用户界面格式框 #ref , 格式 , 起始x , 起始y , 宽度 , 高度 , F颜色 , B颜色

这将绘制一个带圆角的框。当触摸该框时，将出现一个类似于数字框的数字键盘。不同之处在于，格式化数字框将要求用户根据特定格式输入数字，例如日期、时间等。键盘上的无效键将被禁用，并且用户在输入时将通过指导文本获得引导。这意味着程序员可以确保用户输入的内容始终采用固定格式。

输入的类型由'格式'参数控制，如下所示：

日期1	英国/澳大利亚/新西兰格式的日期 (dd/mm/yy)
日期2	美国格式的日期 (mm/dd/yy)
日期3	国际格式的日期 (yyyy/mm/dd)
时间1	24小时制的时间 (hh:mm)
时间2	带秒的24小时制时间 (hh:mm:ss)
时间3	12小时制的时间 (hh:mm AM/PM)
时间4	带秒的12小时制时间 (hh:mm:ss AM/PM)
日期时间1	包含日期（英国格式）和时间（12小时制）(dd/mm/yy hh:mm AM/PM)
日期时间2	包含日期（英国格式）和时间（24小时制）(dd/mm/yy hh:mm)
日期时间3	包含日期（美国格式）和时间（12小时制）(mm/dd/yy hh:mm AM/PM)
日期时间4	包含日期（美国格式）和时间（24小时制）(mm/dd/yy hh:mm)
纬度1	纬度以度、分钟和秒表示 (d° mm' ss" N/S)
LAT2	纬度，秒保留一位小数 (dd° mm' ss.s" N/S)
LONG1	经度以度、分钟和秒表示 (ddd°mm' ss" E/W)
LONG2	经度，秒保留一位小数 (ddd° mm' ss.s" E/W)
ANGLE1	角度以度和分钟表示 (ddd° mm')

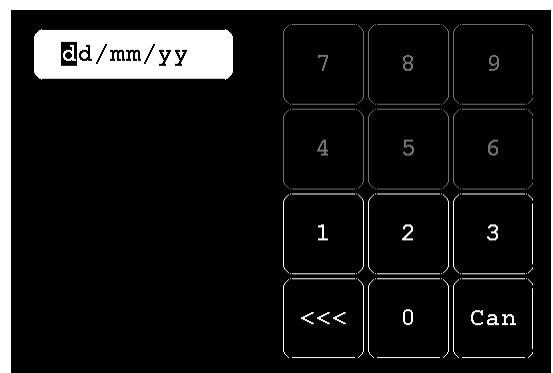
例如：

```
GUI FORMATBOX #1, DATE1, 300, 150, 200, 50
```

将创建一个数据输入框，当触摸时，右侧将出现一个键盘。

请注意：

- 显示框中填充了一个提示字符串，以提示用户所需的数据。
- 由于月份的天数只能以0到3的数字开头，因此其他所有键都被禁用。其他具有有限范围的数字也会出现这种情况。
- 通过 CtrlVal(#1) 获取的控制值是一个字符串。
例如，如果用户输入2020年5月8日的日期，返回的字符串将是"08/05/20"（即按照DATE1指定的英国/澳大利亚/新西兰格式）。



控制值可以通过字符串函数拆分，或者在某些情况下，字符串可以直接使用。例如，如果使用上述格式框从用户获取日期，则PicoMite的内部时钟可以直接设置如下：

```
DATE$ = CtrlVal(#1)
```

RTC SETTIME命令将接受一个字符串参数，格式为dd/mm/yy hh:mm，因此如果格式框使用DATETIME2作为'格式'，RTC时间可以如下设置：

```
RTC SETTIME CtrlVal(#1)
```

您可以使用美国风格的DATETIME4来获取日期/时间。在这种情况下，您将使用此命令设置RTC：

```
RTC SETTIME MID$(CtrlVal(#1), 4, 3) + LEFT$(CtrlVal(#1), 2) + RIGHT$(CtrlVal(#1), 9)
```

MMBasic 将尝试在屏幕上定位虚拟键盘，以避免遮挡导致其出现的格式框。当键盘被部署时，将生成一个笔下中断，当所有必需的数据输入完成并且键盘被隐藏时，将生成一个按键抬起中断。

Spin Box

```
GUI SPINBOX #ref, StartX, StartY, Width, Height, FColour, BColour, Step,  
Minimum, Maximum
```

This will draw a box with up/down icons on either end. When these icons are touched the number in the box will be incremented or decremented by the 'StepValue', holding down the touch will repeat at a fast rate. 'Minimum' and 'Maximum' set a limit on the value that can be entered.

'StepValue', 'Minimum' and 'Maximum' are optional and if not specified 'StepValue' will be 1 and there will be no limit on the number entered. A pen down interrupt will be generated every time up/down is touched or when automatic repeat occurs.

Caption

```
GUI CAPTION #ref, text$, StartX, StartY, Alignment, FColour, BColour
```

This will draw a text string on the screen. It is similar to the basic drawing command TEXT, the difference being that MMBasic will automatically dim this control if a keyboard or number pad is displayed.

'Alignment' is zero to three characters (a string expression or variable is also allowed) where the first letter is the horizontal alignment around X and can be L, C or R for LEFT, CENTER, RIGHT and the second letter is the vertical alignment around Y and can be T, M or B for TOP, MIDDLE, BOTTOM.

A third character can be used to indicate the rotation of the text. This can be 'N' for normal orientation, 'V' for vertical text with each character under the previous running from top to bottom, 'I' the text will be inverted (i.e. upside down), 'U' the text will be rotated counter clockwise by 90° and 'D' the text will be rotated clockwise by 90°. The default alignment is left/top with no rotation.

If the colours are not specified this control will use the colours set by the COLOUR command.

Circular Gauge

```
GUI GAUGE #ref, StartX, StartY, Radius, FColour, BColour, min, max,  
nbrdec, units$, c1, ta, c2, tb, c3, tc, c4
```

This will define a graphical circular analog gauge with a digital display in the centre showing the value and units. If specified the gauge will be coloured to provide a graphical indication of the signal level (e.g. green for OK, yellow for warning, etc).

'StartX' and 'StartY' are the coordinates of the centre of the gauge while 'Radius' is the distance from the centre to the outer edge.

'min' is the value associated with the minimum value of the gauge and 'max' is the maximum value. When CtrlVal() is used to assign a value (floating point or integer) to the gauge the analogue portion of the gauge will be drawn to a length proportional to the range between 'min' and 'max'.

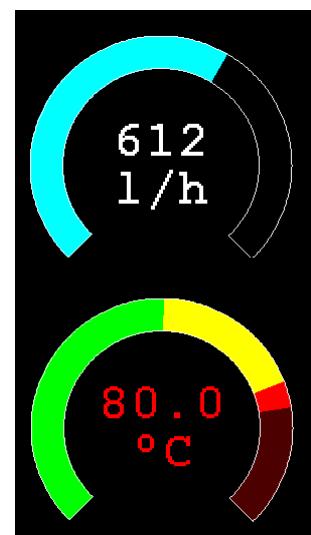
At the same time the digital value will be drawn in the centre of the gauge using the current font settings (set with the FONT command). 'nbrdec' specifies the number of decimal places to be used in this display. Under the digital value the 'units\$' will be displayed (this can be skipped or a zero length string used if not required).

Normally the analog graph is drawn using the colour specified in 'Fcolour' however a multi colour gauge can be created using 'c1' to 'c4' for the colours and 'ta' to 'tc' for the thresholds used to determine when the colour will change.

Specifically, 'c1' is the colour to be used for values up to 'ta'. 'c2' is the colour to be used for values between 'ta' and 'tb', 'c3' is used for values between 'tb' and 'tc' and 'c4' is used for values above 'tc'. Colours and thresholds not required can be left off then list. For example, for a two colour gauge only 'c1', 'ta' and 'c2' need to be specified.

When colours and thresholds are specified the background of the gauge will be drawn with a dull version of the gauge colour at that level ("ghost colouring") so that the user can appreciate how close to the various thresholds the actual value is. Also the digital value displayed in the centre will also change to the colour specified by the current value.

If only one colour is required for the whole analogue graph it can be specified by just using 'c1' and leaving all the following parameters off.



旋转框

```
GUI SPINBOX #ref, StartX, StartY, 宽度, 高度, F颜色, B颜色, 步骤,  
          最小值, 最大值
```

这将绘制一个两端带有上下图标的框。当触摸这些图标时，框中的数字将按 'StepValue' 增加或减少，按住触摸将以快速速度重复。

'最小值' 和 '最大值' 设置可以输入的值的限制。

'步骤值'、'最小值'和'最大值'是可选的，如果未指定，'步骤值'将为1，并且输入的数字没有限制。每次触摸上/下或发生自动重复时，将生成一个笔下中断。

标题

```
图形用户界面标题 #ref, text$, StartX, StartY, 对齐方式, F颜色, B颜色
```

这将在屏幕上绘制一个文本字符串。它类似于基本绘图命令TEXT，不同之处在于如果显示了键盘或数字键盘，MMBasic将自动使该控件变暗。

"对齐方式"是零到三个字符（也允许字符串表达式或变量），第一个字母是围绕X的水平对齐方式，可以是L、C或R，分别表示左、居中、右；第二个字母是围绕Y的垂直对齐方式，可以是T、M或B，分别表示顶部、中间、底部。

第三个字符可以用来指示文本的旋转。这可以是'N'表示正常方向，'V'表示垂直文本，每个字符在上一个字符的下方从上到下排列，'I'表示文本将被翻转（即倒置），'U'表示文本将逆时针旋转90°，'D'表示文本将顺时针旋转90°。默认对齐方式为左/上且没有旋转。

如果未指定颜色，则该控件将使用COLOUR命令设置的颜色。

圆形量规

```
GUI GAUGE #ref, StartX, StartY, Radius, FColour, BColour, min, max,  
nbrdec, units$, c1, ta, c2, tb, c3, tc, c4
```

这将定义一个图形圆形模拟量规，中心显示值和单位的数字显示。如果指定，量规将被着色以提供信号水平的图形指示（例如，绿色表示正常，黄色表示警告等）。

'StartX' 和 'StartY' 是量规中心的坐标，而 'Radius' 是从中心到外边缘的距离。

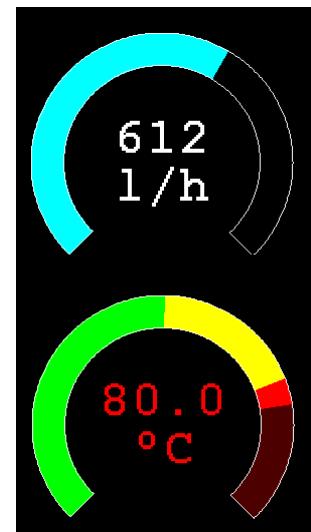
'min' 是与量规最小值相关的值，'max' 是最大值。当使用 CtrlVal() 将一个值（浮点数或整数）赋值给量规时，量规的模拟部分将根据 'min' 和 'max' 之间的范围绘制相应的长度。

同时，数字值将使用当前的字体设置（通过 FONT 命令设置）绘制在量规的中心。'nbrdec' 指定在此显示中使用的小数位数。在数字值下方将显示 'units'（如果不需，可以跳过或使用零长度字符串）。

通常，模拟图形使用 'Fcolour' 中指定的颜色绘制，但可以使用 'c1' 到 'c4' 来创建多色量规，使用 'ta' 到 'tc' 来确定颜色变化的阈值。

'c1' 是用于值小于或等于 'ta' 的颜色。'c2' 是用于值在 'ta' 和 'tb' 之间的颜色，'c3' 用于值在 'tb' 和 'tc' 之间，'c4' 用于值大于 'tc' 的颜色。不需要的颜色和阈值可以省略然后列出。例如，对于一个双色量规，只需指定 'c1'、'ta' 和 'c2'。

当指定颜色和阈值时，量规的背景将以该级别的量规颜色的暗淡版本绘制（"幽灵着色"），以便用户能够欣赏实际值与各个阈值的接近程度。此外，中心显示的数字值也将根据当前值的颜色进行更改。



如果整个模拟图只需要一种颜色，可以仅使用 'c1' 并省略所有后续参数。

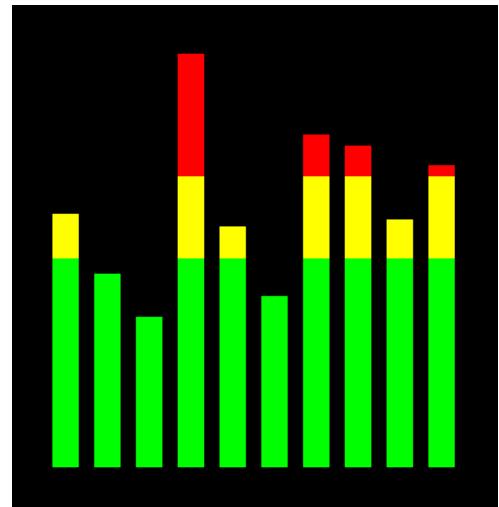
Bar Gauge

```
GUI BARGAUGE #ref, StartX, StartY, width, height, FColour, BColour, min, max, c1, ta, c2, tb, c3, tc, c4
```

This will define either a horizontal or vertical bar gauge. The gauge can be coloured to provide a graphical indication of the signal level (e.g. green for OK, yellow for warning, etc) and many bar graphs can be packed close together so that a number of values can be displayed simultaneously using a small amount of screen space (as shown in the image which consists of ten bar gauges).

If the width is less than the height the bar gauge will be drawn vertically with the analogue graph growing from the bottom towards the top. Otherwise, if the width is more than the height, it will be drawn horizontally with the analogue graph growing from the left towards the right. In both cases 'StartX' and 'StartY' reference the top left coordinate of the bar graph while 'width' is the horizontal width and 'height' the vertical height.

The bar graph does not have a digital display of its value but other than that the parameters are the same as for the circular gauge (described above).



'min' and 'max' specify the range of values for the bar and, if specified, 'c1' to 'c4' and 'ta' to 'tc' specify the colours and thresholds for the analogue bar image. Note that unlike the circular bar gauge a "ghost image" of the colours is not shown in the background.

As with the circular gauge, if only one colour is required for the whole gauge it can be specified by just using 'c1' and leaving all the following parameters off.

Area

```
GUI AREA #ref, StartX, StartY, Width, Height
```

This will define an invisible area of the screen that is sensitive to touch and will set TOUCH(REF) and TOUCH(LASTREF) accordingly when touched or released. It can be used as the basis for creating a custom control which is defined and managed by the BASIC program.

Interacting with Controls

Using the following commands and functions the characteristics of the on screen controls can be changed and their value retrieved.

= CTRLVAL(#ref)

This is a function that will return the current value of a control. For controls like check boxes or switches it will be the number one (true) indicating that the control has been selected by the user or zero (false) if not. For controls that hold a number (e.g. a SPINBOX) the value will be the number (normally a floating point number). For controls that hold a string (e.g. TEXTBOX) the value will be a string. For example:

```
PRINT "The number in the spin box is: " CTRLVAL(#10)
```

CTRLVAL(#ref) =

This command will set the value of a control. For off/on controls like check boxes it will override any touch input and can be used to depress/release switches, tick/untick check boxes, etc. A value of zero is off or unchecked and non zero will turn the control on. For a LED it will cause the LED to be illuminated or turned off. It can also be used to set the initial value of spin boxes, text boxes, etc. For example:

```
CTRLVAL(#10) = 12.4
```

GUI FCOLOUR colour, #ref1 [, #ref2, #ref3, etc]

This will change the foreground colour of the specified controls to 'colour'. This is especially handy for a LED which can change colour.

GUI BCOLOUR colour, #ref1 [, #ref2, #ref3, etc]

This will change the background colour of the specified controls to 'colour'.

= TOUCH(REF)

This is a function that will return the reference number of the control currently being touched. If no control is currently being touched it will return zero.

条形量规

GUI BARGAUGE #ref, StartX, StartY, 宽度, 高度, F颜色, B颜色, 最小, 最大, c1, ta, c2, tb, c3, tc, c4

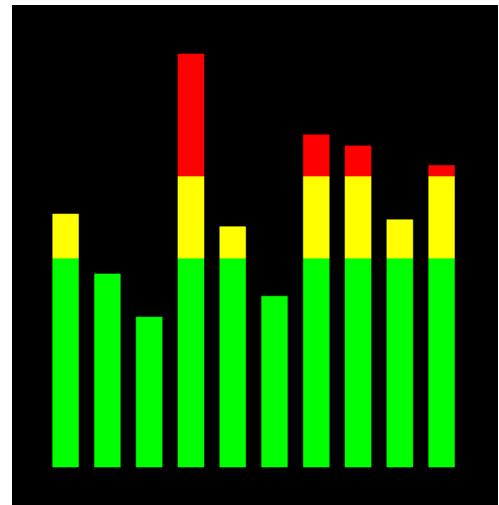
这将定义一个水平或垂直的条形量规。该量规

可以着色以提供信号水平的图形指示

(例如，绿色表示正常，黄色表示警告等)，许多条形图可以紧密排列，以便在较小的屏幕空间内同时显示多个值（如图像中显示的十个条形量规）。如果宽度小于高度，条形量规将垂直绘制，模拟图形从底部向顶部增长。

否则，如果宽度大于高度，它将水平绘制，模拟图形从左向右增长。在这两种情况下，'StartX' 和 'StartY' 引用条形图的左上角坐标，而 '宽度' 是水平宽度，'高度' 是垂直高度。

该条形图没有其值的数字显示，但除此之外，参数与圆形量规（如上所述）相同。



'min' 和 'max' 指定条形的值范围，如果指定，'c1' 到 'c4' 以及 'ta' 到 'tc' 指定模拟条形图像的颜色和阈值。请注意，与圆形条形量规不同，背景中不显示颜色的“幽灵图像”。

与圆形量规一样，如果整个量规只需要一种颜色，可以只使用 'c1' 并省略所有后续参数。

区域

图形用户界面区域 #ref, StartX, StartY, 宽度, 高度

这将定义一个对触摸敏感的屏幕隐形区域，并在触摸或释放时相应地设置 TOUCH(REF) 和 TOUCH(LASTREF)。它可以作为创建自定义控件的基础，该控件由 BASIC 程序定义和管理。

与控件交互

使用以下命令和函数可以更改屏幕上控件的特性并检索其值。

= CTRLVAL(#ref)

这是一个将返回控件当前值的函数。对于复选框或开关等控件，返回值为数字一（真），表示控件已被用户选择；如果未选择，则返回零（假）。对于持有数字的控件（例如 SPINBOX），返回值将是该数字（通常是浮点数）。对于持有字符串的控件（例如 TEXTBOX），返回值将是一个字符串。例如：

PRINT "旋转框中的数字是：" CTRLVAL (#10)

CTRLVAL(#ref) =

此命令将设置控件的值。对于像复选框这样的开/关控件，它将覆盖任何触摸输入，并可用于按下/释放开关、勾选/取消勾选复选框等。值为零表示关闭或未选中，非零值将打开控件。对于 LED，它将使 LED 亮起或熄灭。它还可以用于设置旋转框、文本框等的初始值。例如：

CTRLVAL (#10) = 12.4

GUI FCOLOUR 颜色, #ref1 [, #ref2, #ref3, 等]

这将把指定控件的前景颜色更改为'颜色'。这对于可以改变颜色的LED特别方便。

GUI BCOLOUR 颜色, #ref1 [, #ref2, #ref3, 等]

这将把指定控件的背景颜色更改为'颜色'。

= TOUCH(REF)

这是一个函数，将返回当前被触摸控件的引用编号。如果没有控件被触摸，它将返回零。

- = TOUCH(LASTREF)
This is a function that will return the reference number of the control that was last touched.
- GUI DISABLE #ref1 [, #ref2, #ref3, etc]
This will disable the controls in the list. Disabled controls do not respond to touch and will be displayed dimmed. The keyword ALL can be used as the argument and that will disable all controls on the currently displayed page. For example:
GUI DISABLE ALL
- GUI ENABLE #ref1 [, #ref2, #ref3, etc]
This will undo the effects of GUI DISABLE and restore the controls in the list to normal operation. The keyword ALL can be used as the argument for all controls on the currently displayed page.
- GUI HIDE #ref1 [, #ref2, #ref3, etc]
This will hide the controls in the list. Hidden controls will not respond to touch and will be replaced on the screen with the current background colour. The keyword ALL can be used as the argument.
- GUI SHOW #ref1 [, #ref2, #ref3, etc]
This will undo the effects of GUI HIDE and restore the controls in the list to being visible and capable of normal operation. The keyword ALL can be used as the argument for all controls.
- GUI DELETE #ref1 [, #ref2, #ref3, etc]
This will delete the controls in the list. This includes removing the image of the control from the screen using the current background colour and freeing the memory used by the control. The keyword ALL can be used as the argument and that will cause all controls to be deleted.

MsgBox()

The MsgBox() function will display a message box on the screen and wait for user input. While the message box is displayed all controls will be disabled so that the message box has the complete focus.

The syntax is:

```
r = MsgBox(message$, button1$ [, button2$ [, button3$ [, button4$]]])
```

All arguments are strings. 'message\$' is the message to display. This can contain one or more tilde characters (~) which indicate a line break. Up to 10 lines can be displayed inside the box. 'button1\$' is the caption for the first button, 'button2\$' is the caption for the second button, etc. At least one button must be specified and four is the maximum. Any buttons not included in the argument list will not be displayed.

The font used will be the default font set using the FONT command and the colours used will be the defaults set by the COLOUR command. The box will be automatically sized taking into account the dimensions of the default font, the number of lines to display and the number of buttons specified.

When the user touches a button the message box will erase itself, restore the display (e.g. re enable all controls) and return the number of the button that was touched (the first button will return 1, the second 2, etc). Note that, unlike all other GUI controls the BASIC program will stop running while the message box is displayed, interrupts however will be honoured and acted upon.

To illustrate the usage of a message box will the following program fragment will attempt to open a file and if an error occurs the program will display an error message using the MsgBox() function. The message has two lines and the box has two buttons for retry and cancel.

Do

```
On Error Skip
Open "file.txt" For Input As #1
If MM.ErrNo <> 0 Then
    If MsgBox("Error~Opening file.txt", "RETRY", "CANCEL") = 2 Then Exit Sub
EndIf
Loop While MM.ErrNo <> 0
```

- = TOUCH(LASTREF)
这是一个函数，将返回最后被触摸控件的引用编号。
- GUI DISABLE#ref1 [, #ref2, #ref3, 等]
这将禁用列表中的控件。禁用的控件不响应触摸，并将以暗淡的方式显示。关键字 ALL 可以作为参数使用，这将禁用当前显示页面上的所有控件。例如：

```
GUI DISABLE ALL
```

- GUI ENABLE#ref1 [, #ref2, #ref3, 等等]
这将撤销 GUI DISABLE 的效果，并将列表中的控件恢复到正常操作状态。关键字 ALL 可以作为当前显示页面上所有控件的参数使用。
- GUI HIDE #ref1 [, #ref2, #ref3, 等等]
这将隐藏列表中的控件。隐藏的控件将不响应触摸，并将在屏幕上被当前背景颜色替代。关键字 ALL 可以作为参数使用。
- GUI SHOW #ref1 [, #ref2, #ref3, 等等]
这将撤销 GUI HIDE 的效果，并将列表中的控件恢复为可见并能够正常操作。关键字 ALL 可以作为所有控件的参数使用。
- GUI DELETE #ref1 [, #ref2, #ref3, 等等]
这将删除列表中的控件。这包括使用当前背景颜色从屏幕上移除控件的图像，并释放控件所使用的内存。可以将关键字 ALL 用作参数，这将导致删除所有控件。

MsgBox()

MsgBox() 函数将在屏幕上显示一个消息框，并等待用户输入。当消息框显示时，所有控件将被禁用，以便消息框获得完全的焦点。

语法为：

```
r = MsgBox(message$, button1$ [, button2$ [, button3$ [, button4$]]])
```

所有参数都是字符串。'message' 是要显示的消息。这可以包含一个或多个波浪号字符(~)，表示换行。最多可以在框内显示 10 行。'button1' 是第一个按钮的标题，'button2' 是第二个按钮的标题，等等。必须指定至少一个按钮，最多可以指定四个。未包含在参数列表中的任何按钮将不会显示。

使用的字体将是通过 FONT 命令设置的默认字体，使用的颜色将是通过 COLOUR 命令设置的默认颜色。该框的大小将自动调整，考虑到默认字体的尺寸、要显示的行数以及指定的按钮数量。

当用户触摸一个按钮时，消息框将自动擦除自身，恢复显示（例如，重新启用所有控件），并返回被触摸按钮的编号（第一个按钮返回1，第二个返回2，依此类推）。请注意，与所有其他图形用户界面控件不同，消息框显示时 BASIC 程序将停止运行，但中断将被尊重并执行。

为了说明消息框的用法，以下程序片段将尝试打开一个文件，如果发生错误，程序将使用 MsgBox() 函数显示错误消息。该消息有两行，框中有两个按钮用于重试和取消。

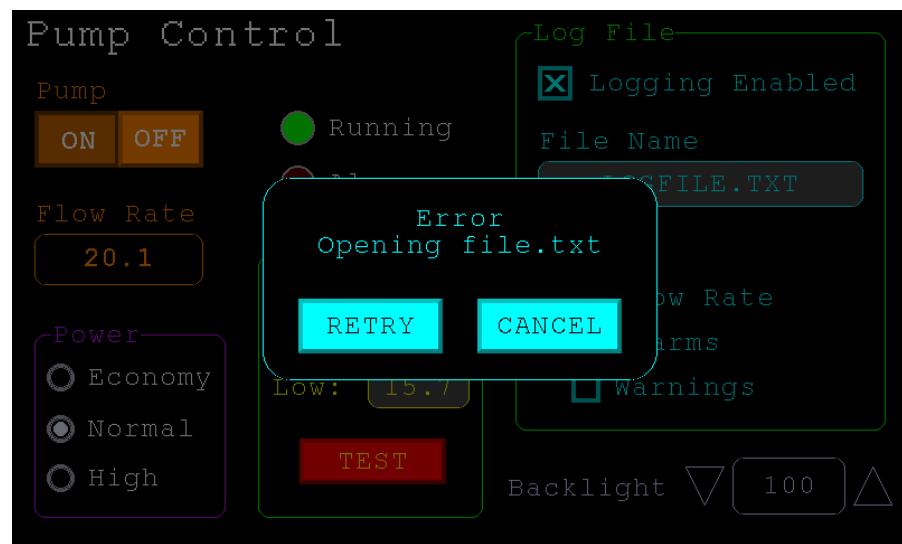
执行

```

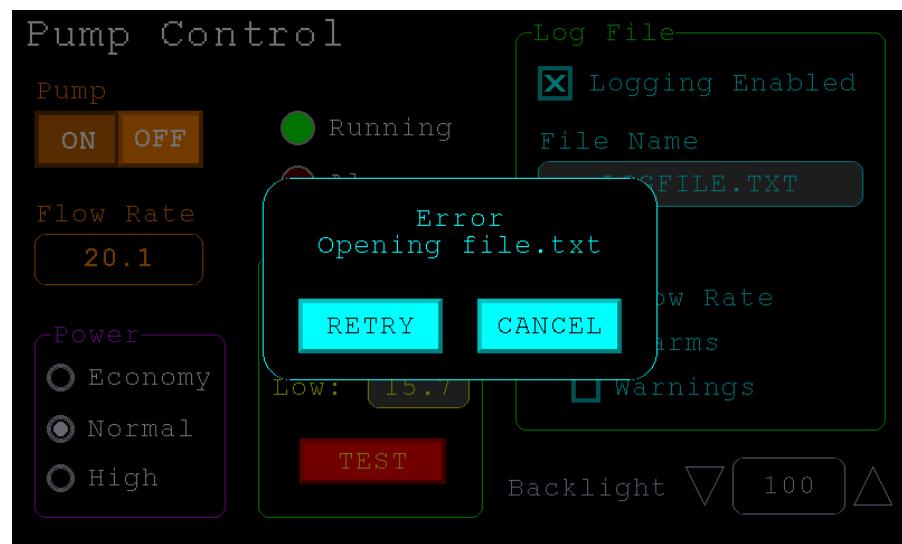
    错误时跳过
    打开 "file.txt" 以输入作为 #1
    如果 MM.ErrNo <> 0 则
        如果 MsgBox( "打开 file.txt 时出错" , "重试" , "取消" ) = 2 则退出子程序
    结束如果
    循环 当 MM.ErrNo <> 0

```

This would be the result if the file "file.txt" did not exist:



如果文件 "file.txt" 不存在，则结果将是：



Advanced Graphics Programming Techniques

When programming using the advanced GUI commands there are a number of hints and techniques to consider that will make it easier to develop and maintain your program.

The User Should Be In Control

Traditional character based programs are normally in control of the interaction with the user. For example, the program may display a menu and prompt the user to select an action. If the user selects an invalid option the program would display an error message and display the menu options again.

However graphical based programs such as that created using the advanced GUI commands are different. Usually the program just starts running doing what it normally does (e.g. control temperature, speed, etc) and it is the user's job to select and change parameters without being prompted. This is a different way of programming and is often hard for the traditional programmer to get used to this different technique.

As an example, consider a program that is to control a cutting device. The traditional program would prompt the user for the speed and cutting time. When both have been entered the program would prompt to start the cutting cycle. However, a graphical based program would display two number boxes where the user could enter the speed and time along with a run button. The number boxes could be filled with default values and the run button would be disabled if the user entered an invalid speed or time. When the run button is touched the cutting cycle would start.

A good example of this type of graphical interface is the dialogue box used on a Windows/IOS/Android computer to set the time and date. It displays a number of boxes where the user can enter the date/time along with an OK button that tells the program to accept the data entered. At no time is the user forced to make a selection from a menu. Also, the current time/date is already displayed in the entry boxes so the user can accept them as the default if they wanted to do so.

If you need some inspiration as to how your graphical program should look and feel check your nearest GUI based operating system to see how they operate.

Program Structure

Typically a program would start by defining the controls (which MMBasic will draw on the screen), then it would set the defaults and finally it would drop into a continuous loop where it would do whatever job it was design to do. For example, take the case of a simple controller for a motor where the user could select the speed and cause the motor to run by pressing an on screen button.

To implement this function the program would look something like this:

```
GUI CAPTION #1, "Speed (rpm)", 200, 50      ' label the number box
GUI NUMBERBOX #2, 200, 100, 150, 40          ' define and draw the number box
CtrlVal(#2) = 100                            ' default value for the speed
GUI BUTTON #3, "RUN", 200, 350, 0, RGB(red)   ' define and draw the RUN button

DO
  IF CtrlVal(#3)<10 OR CtrlVal(#3)>200 THEN
    GUI DISABLE #3
  ELSE
    GUI ENABLE #3
  ENDIF

  IF CtrlVal(#3) = 1 THEN
    SetMotorSpeed CtrlVal(#2)
  ELSE
    SetMotorSpeed 0
  ENDIF
LOOP
```

' this runs in a loop forever
' check the speed setting
' disable RUN if it is invalid
' otherwise
' enable the RUN button

' if the button is pressed
' make the motor run
' otherwise the button is up
' therefore set motor speed to zero

Note that the user is not prompted to do anything; the program just sits in a loop reacting to the changes that the user has made to the controls (i.e. the user is in control).

Disable Invalid Controls

As in the above example, disabling a control will prevent a user from using it and MMBasic will redraw it in a dull colour to indicate that it is not available. This is the equivalent of an error message in a traditional text

高级图形编程技术

在使用高级图形用户界面命令进行编程时，有许多提示和技巧需要考虑，这将使开发和维护程序变得更容易。

用户应该掌控

传统的基于字符的程序通常控制与用户的交互。例如，程序可能会显示一个菜单并提示用户选择一个操作。如果用户选择了无效的选项，程序将显示错误消息并再次显示菜单选项。

然而，使用高级图形用户界面命令创建的基于图形的程序则不同。

通常，程序会开始运行，执行其正常操作（例如控制温度、速度等），用户的任务是选择和更改参数，而无需提示。这是一种不同的编程方式，传统程序员往往很难适应这种不同的技术。

作为一个例子，考虑一个控制切割设备的程序。传统程序会提示用户输入速度和切割时间。当两者都输入后，程序会提示开始切割周期。然而，基于图形的程序会显示两个数字框，用户可以在其中输入速度和时间，并有一个运行按钮。数字框可以填充默认值，如果用户输入了无效的速度或时间，运行按钮将被禁用。当触摸运行按钮时，切割周期将开始。

这种图形界面的一个好例子是用于设置时间和日期的对话框，通常在Windows/iOS/Android计算机上使用。它显示多个框，用户可以在其中输入日期/时间，并有一个确定按钮，告诉程序接受输入的数据。用户在任何时候都不被强迫从菜单中做出选择。此外，当前的时间/日期已经显示在输入框中，因此用户可以选择接受它们作为默认值。

如果您需要一些灵感来设计您的图形程序的外观和感觉，可以查看您最近的基于GUI的操作系统，看看它们是如何操作的。

程序结构

通常，一个程序会首先定义控件（MMBasic将在屏幕上绘制这些控件），然后设置默认值，最后进入一个连续循环，在其中执行其设计的任务。例如，考虑一个简单的电动机控制器，用户可以选择速度并通过按下屏幕上的按钮使电动机运行。

要实现这个功能，程序看起来大致如下：

```
GUI CAPTION #1, "速度 (rpm)", 200, 50          ' 标记数字框
GUI NUMBERBOX #2, 200, 100, 150, 40            ' 定义并绘制数字框
CtrlVal(#2) = 10 0                             ' 默认速度值
GUI BUTTON #3, "运行", 200, 350, 0, RGB(红色) ' 定义并绘制运行按钮

DO
    如果 CtrlVal(#3)<10 或 CtrlVal(#3)>200 那么 ' 检查速度设置
        GUI 禁用 #3
    否则
        GUI 启用 #3
    结束如果

    如果 CtrlVal(#3) = 1 那么 ' 如果按钮被按下
        设置电动机速度 CtrlVal(#2)
    否则
        设置电动机速度 0
    结束如果
循环
```

请注意，用户不会被提示做任何事情；程序只是处于一个循环中，响应用户对控件所做的更改（即用户掌控一切）。

禁用无效控件

如上例所示，禁用控件将阻止用户使用它，MMBasic会将其重新绘制为暗淡的颜色，以指示它不可用。这相当于传统文本程序中的错误消息。

based program and is more user friendly than popping up a message box which must be dismissed before anything else can be done.

There are many times that a control could be invalid, for example when an input is not ready or simply when an option or action does not apply. Later, when the control becomes valid you can use the GUI ENABLE command to return it to use. Another example is when a GUI NUMBERBOX keypad is displayed MMBasic will automatically disable all other controls on the screen so that it is obvious to the user where their input is required.

Disabling a control still leaves it on the screen, so that the user knows that it is there but it will be dimmed and will not respond to touch. Not responding to touch also means that the user cannot change it and an interrupt will not be generated when it is touched. This is handy for you the programmer because you do not have to check if the control is valid before acting on it.

Use Constants for Control Reference Numbers

The advanced controls use a reference number to identify the control. To make it easy to read and maintain your program you should define these numbers as constants with easy to recognise names.

For example, in the following program fragment MAIN_SWITCH is defined as a constant and this constant is used wherever the reference number for that control is required:

```
CONST MAIN_SWITCH = 5
CONST ALARM_LED = 6
'...
GUI SWITCH MAIN_SWITCH, "ON|OFF", 330, 50, 140, 50, RGB(white), RGB(blue)
GUI LED ALARM_LED, 215, 220, 30, RGB(red)
'...
IF CtrlVal(MAIN_SWITCH) = 0 THEN ...    ' for example turn the pump off
IF ALARM THEN CtrlVal(ALARM_LED) = 1
```

It is much easier to remember what MAIN_SWITCH does than remembering what control the number 5 refers to. Also, when you have a lot of controls it is much easier to renumber the controls when all their numbers are defined at the one place at the start of the program.

The reference number must be a number between 1 and the value set with the OPTION CONTROL command. Increasing this number will consume more RAM and decreasing it will recover some RAM.

The Main Program Is Still Running

It is important to realise that your main BASIC program is still running while the user is interacting with the GUI controls. For example, it will continue running even while a user holds down an on screen switch and it will keep running while the virtual keyboard is displayed as a result of touching a TEXTBOX control.

For this reason your main program should not arbitrarily update touch sensitive screen controls, because they might change the on screen image while the user is using them (with undefined results). Normally when a BASIC program using GUI controls starts it will initialise controls such as a SPINBOX, NUMBERBOX and TEXTBOX to some initial value but from then on the main program should just read the value of these controls – it is the responsibility of the user to change these, not your program.

However, if you do want to change the value of such an on-screen control you need some mechanism to prevent both the program and the user making a change at the same time. One method is to set a flag within the key down interrupt to indicate that the control should not be updated during this time. This flag can then be cleared in the key up interrupt to allow the main program to resume updating the control.

Note that this discussion only applies to controls that respond to touch. Controls such as CAPTION and LED can be changed at any time by the main program and often are.

Use Interrupts and SELECT CASE Statements

Everything that happens on a screen using the advanced controls will be signalled by an interrupt, either touch down or touch up. So, if you want to do something immediately when a control is changed, you should do it in an interrupt. Mostly you will be interested in when the touch (or pen) is down but in some cases you might also want to know when it is released.

Because the interrupt is triggered when the pen touches any control or part of the screen you need to discover what control was being touched. This is best performed using the TOUCH(REF) function and the SELECT CASE statement.

For example, in the following fragment the subroutine PenDown will be called when there is a touch and the function TOUCH(REF) will return the reference number of the control being touched. Using the SELECT

并且比弹出一个必须关闭的消息框更友好，这样在关闭之前无法进行其他操作。

控件无效的情况有很多，例如当输入尚未准备好，或当某个选项或操作不适用时。稍后，当控件变为有效时，您可以使用GUI ENABLE命令将其恢复使用。另一个例子是，当显示GUI NUMBERBOX键盘时，MMBasic会自动禁用屏幕上的所有其他控件，以便用户清楚地知道他们的输入在哪里需要。

禁用控件仍然会在屏幕上显示，以便用户知道它的存在，但它会变得暗淡，并且不会响应触摸。不响应触摸也意味着用户无法更改它，并且在触摸时不会生成中断。这对你作为程序员来说很方便，因为你不必在操作之前检查控件是否有效。

使用常量作为控件引用编号

高级控件使用引用编号来识别控件。为了使你的程序易于阅读和维护，你应该将这些编号定义为具有易于识别名称的常量。

例如，在以下程序片段中，MAIN_SWITCH 被定义为常量，并且在需要该控件的引用编号的地方使用了这个常量：

```
CONST MAIN_SWITCH = 5
CONST ALARM_LED = 6
...
GUI SWITCH MAIN_SWITCH, "开|关", 330, 50, 140, 50, RGB(白色), RGB(蓝色)
GUI LED ALARM_LED, 215, 220, 30, RGB(红色)
...
```

如果 CtrlVal(MAIN_SWITCH) = 0 那么 ... ' 例如关闭泵
如果 ALARM 那么 CtrlVal(ALARM_LED) = 1

记住 MAIN_SWITCH 的功能比记住数字 5 所指的控件要容易得多。此外，当你有很多控件时，在程序开始时将所有控件的数字定义在一个地方会使重新编号控件变得更加容易。

参考编号必须是 1 到 OPTION CONTROL 命令设置的值之间的数字。

增加这个数字会消耗更多的 RAM，减少它会恢复一些 RAM。

主程序仍在运行

重要的是要意识到，当用户与 GUI 控件交互时，你的主 BASIC 程序仍在运行。例如，即使用户按住屏幕上的开关，它也会继续运行，并且在触摸 TEXTBOX 控件时虚拟键盘显示时，它也会继续运行。

因此，您的主程序不应随意更新触摸敏感屏幕控件，因为在用户使用这些控件时，它们可能会改变屏幕上的图像（结果未定义）。通常，当使用图形用户界面控件的BASIC程序启动时，它会将控件（如SPINBOX、NUMBERBOX和TEXTBOX）初始化为某个初始值，但从那时起，主程序只需读取这些控件的值——改变这些值的责任在于用户，而不是您的程序。

然而，如果您确实想要更改此类屏幕控件的值，您需要某种机制来防止程序和用户同时进行更改。一种方法是在按键按下中断中设置一个标志，以指示在此期间控件不应被更新。然后可以在按键抬起中断中清除此标志，以允许主程序恢复更新控件。

请注意，这个讨论仅适用于响应触摸的控件。控件如标题和LED可以随时由主程序更改，并且通常会被更改。

使用中断和选择案例语句

在使用高级控件时，屏幕上发生的所有事件都会通过中断信号传递，无论是触摸下还是触摸上。因此，如果您希望在控件更改时立即执行某些操作，您应该在中断中进行。大多数情况下，您会对触摸（或笔）按下时感兴趣，但在某些情况下，您可能还想知道它何时被释放。

因为中断是在笔触碰到任何控件或屏幕的某个部分时触发的，所以您需要确定被触摸的控件是什么。这最好通过使用TOUCH(REF)函数和选择案例语句来完成。

例如，在以下片段中，当发生触摸时，将调用子程序PenDown，而函数TOUCH(REF)将返回被触摸控件的引用编号。使用选择

CASE the alarm LED will be turned on or off depending on which button is touched. The action could be any number of things like raising an I/O pin to turn on a physical siren or printing a message on the console.

```
CONST ALARM_ON = 15
CONST ALARM_OFF = 16
CONST ALARM_LED = 33
GUI INTERRUPT PenDown
' ...
GUI BUTTON ALARM_ON, "ALARM ON ", 330, 50, 140, 50, RGB(white), RGB(blue)
GUI BUTTON ALARM_OFF, "ALARM OFF ", 330, 150, 140, 50, RGB(white), RGB(blue)
GUI LED ALARM_LED, 215, 220, 30, RGB(red)
' ...
DO : LOOP      ' the main program is doing something

' this sub is called when touch is detected
SUB PenDown
    SELECT CASE TOUCH(REF)
        CASE ALARM_ON
            CtrlVal(ALARM_LED) = 1
        CASE ALARM_OFF
            CtrlVal(ALARM_LED) = 0
    END SELECT
END SUB
```

The SELECT CASE can also test for other controls and perform whatever actions are required for them in their own section of the CASE statement.

The important point is that the maintenance of the controls (e.g. responding to the buttons and turning the alarm LED off or on) is done automatically without the main program being involved – it can continue doing something useful like calculating some control response, etc.

Touch Up Interrupt

In most cases you can process all user input in the touch down interrupt. But there are exceptions and a typical example is when you need to change the characteristics of the control that is being touched. For example, if you wanted to change the foreground colour of a button from white to red when it is down. When it is returned to the up state the colour should revert to white.

Setting the colour on the touch down is easy. Just define a touch down interrupt and change the colour in the interrupt when that control is touched. However, to return the colour to white you need to detect when the touch has been removed from the control (i.e. touch up). This can be done with a touch up interrupt.

To specify a touch up interrupt you add the name of the subroutine for this interrupt to the end of the GUI INTERRUPT command. For example:

```
GUI INTERRUPT IntTouchDown, IntTouchUp
```

Within the touch up subroutine you can use the same structure as in the touch down sub but you need to find the reference number of the last control that was touched. This is because the touch has already left the screen and no control is currently being touched. To get the number of the last control touched you need to use the function TOUCH(LASTREF)

The following example shows how you could meet the above requirement and implement both a touch down and a touch up interrupt:

```
SUB IntTouchDown
    SELECT CASE TOUCH(REF)
        CASE ButtonRef
            GUI FCOLOUR RGB(RED), ButtonRef
    END SELECT
END SUB

SUB IntTouchUp
    SELECT CASE TOUCH(LASTREF)
        CASE ButtonRef
            GUI FCOLOUR RGB(WHITE), ButtonRef
    END SELECT
END SUB
```

案例：警报 LED 将根据触摸的按钮被打开或关闭。该操作可以是多种事情，例如提升一个输入/输出引脚以打开物理警报器，或在控制台上打印一条消息。

```
CONST ALARM_ON = 15
CONST ALARM_OFF = 16
CONST ALARM_LED = 33
图形用户界面中断 PenDown
'...
图形用户界面按钮 ALARM_ON, "警报开启", 330, 50, 140, 50, RGB(白色), RGB(蓝色)
图形用户界面按钮 ALARM_OFF, "警报关闭", 330, 150, 140, 50, RGB(白色), RGB(蓝色)
图形用户界面 LED ALARM_LED, 215, 220, 30, RGB(红色)
'...
循环：循环      ' 主程序正在执行某些操作
' 当检测到触摸时调用此子程序
子程序 PenDown
    选择案例 TOUCH(REF)
        案例 ALARM_ON
            Ctrl1值(ALARM_LED) = 1
        案例 ALARM_OFF
            Ctrl1值(ALARM_LED) = 0
    结束选择
结束子程序
```

SELECT CASE 还可以测试其他控件，并在 CASE 语句的各自部分执行所需的操作。

重要的一点是，控件的维护（例如，响应按钮并打开或关闭警报 LED）是自动完成的，而无需主程序参与——它可以继续执行一些有用的操作，比如计算某些控制响应等。

触摸上升中断

在大多数情况下，您可以在触摸按下中断中处理所有用户输入。但也有例外，典型的例子是当您需要更改被触摸控件的特性时。例如，如果您希望在按钮按下时将前景颜色从白色更改为红色。当它返回到上升状态时，颜色应恢复为白色。

在触摸按下时设置颜色很简单。只需定义一个触摸按下中断，并在该控件被触摸时在中断中更改颜色。然而，要将颜色恢复为白色，您需要检测何时从控件上移除触摸（即触摸上升）。这可以通过触摸抬起中断来完成。

要指定触摸抬起中断，您需要将该中断的子程序名称添加到 GUI 中断命令的末尾。例如：

```
GUI 中断 IntTouchDown, IntTouchUp
```

在触摸抬起子程序中，您可以使用与触摸按下子程序相同的结构，但您需要找到最后一个被触摸控件的引用编号。这是因为触摸已经离开屏幕并且当前没有控件被触摸。要获取最后一个被触摸控件的编号，您需要使用函数 TOUCH(LASTREF)

以下示例展示了如何满足上述要求并实现触摸按下和触摸抬起中断：

```
SUB IntTouchDown
    SELECT CASE TOUCH(REF)
        CASE ButtonRef
            GUI F颜色 RGB(RED), ButtonRef
        END SELECT
    END SUB

SUB IntTouchUp
    SELECT CASE TOUCH(LASTREF)
        CASE ButtonRef
            GUI F颜色 RGB(WHITE), ButtonRef
        END SELECT
    END SUB
```

Keep Interrupts Very Short

Because a touch interrupt indicates a request by the user it is tempting to do some extensive programming within an interrupt. For example, if the touch indicates that the user wants to send a message to another controller it sounds logical to put all that code within the interrupt. But this is not a good idea because MMBasic cannot do anything else while your program is processing the interrupt and sending a message could take many milliseconds.

Instead your program should update a global variable to indicate what is requested and leave the actual execution to the main program. For example, if the user did touch the "send a message" button your program could simply set a global variable to true. Then the main program can monitor this variable and if it changes perform the logic and communications required to satisfy the request.

Remember the commandment "Thou shalt not hang around in an interrupt".

Multiple Screens

Your program might need a number of screens with differing controls on each screen. This could be implemented by deleting the old controls and creating new ones when the screen is switched. But another way to do this is to use the GUI SETUP and GUI PAGE commands. These allow you to organise the controls onto pages and with one simple command you can switch pages. All controls on the old page will be automatically hidden and controls on the new page will be automatically shown.

To allocate controls to a page you use the GUI SETUP nn command where nn refers to the page in the range of 1 to 32. When you have used this command any newly created controls will be assigned to that page. You can use GUI SETUP as many times that you want. For example, in the program fragment below the first two controls will be assigned to page 1, the second to page 2, etc.

```
GUI SETUP 1
GUI Caption #1, "Flow Rate", 20, 170,, RGB(brown),0
GUI Displaybox #2, 20, 200, 150, 45

GUI SETUP 2
GUI Caption #3, "High:", 232, 260, LT, RGB(yellow)
GUI Numberbox #4, 318, 6,90, 12, RGB(yellow), RGB(64,64,64)

GUI SETUP 3
GUI Checkbox #5, "Alarms", 500, 285, 25
GUI Checkbox #6, "Warnings", 500, 325, 25
```

By default only the controls setup as page 1 will be displayed and the others will be hidden.

To switch the screen to page 3 all you need do is use the command GUI PAGE 3. This will cause controls #1 and #2 to be automatically hidden and controls #5 and #6 to be displayed. Similarly GUI PAGE 2 will hide all except #3 and #4 which will be displayed.

You can specify multiple pages to display at the one time, for example, GUI PAGE 1, 3 will display both pages 1 and 3 while hiding page 2. This can be useful if you have a set of controls that must be visible all the time. For example, GUI PAGE 1,2 and GUI PAGE 1, 3 will leave the controls on page 1 visible while the others are switched on and off.

It is perfectly legal for a program to modify controls on other pages even though they are not displayed at the time. This includes changing the value and colours as well as disabling or hiding them. When the display is switched to their page the controls will be displayed with their new attributes.

It is possible to place the GUI PAGE commands in the touch down interrupt so that pressing a certain control or part of the screen will switch to another page.

Note that when ALL is used for the list of controls in commands such as GUI ENABLE ALL this only refers to the controls on the pages that are currently selected for display. Controls on other pages will be unaffected.

All programs start with the equivalent of the commands GUI SETUP 1 and GUI PAGE 1 in force. This means that if the GUI SETUP and GUI PAGE commands are not used the program will run as you would expect with all controls displayed.

A typical usage of the GUI PAGE command is shown below.

保持中断非常简短

因为触摸中断表示用户的请求，所以在中断中进行一些复杂的编程是很诱人的。例如，如果触摸表示用户想要向另一个控制器发送消息，将所有代码放在中断中似乎是合乎逻辑的。但这并不是一个好主意，因为在处理中断和发送消息时，MMBasic无法执行其他任何操作，而发送消息可能需要很多毫秒。

相反，您的程序应该更新一个全局变量以指示请求的内容，并将实际执行留给主程序。例如，如果用户确实触摸了“发送消息”按钮，您的程序可以简单地将全局变量设置为真。然后主程序可以监视这个变量，如果它发生变化，就执行满足请求所需的逻辑和通信。

记住这条戒律：“你不应在中断中停留。”

多个屏幕

您的程序可能需要多个屏幕，每个屏幕上都有不同的控件。这可以通过删除旧控件并在切换屏幕时创建新控件来实现。但另一种方法是使用 GUI SETUP 和 GUI PAGE 命令。这些命令允许您将控件组织到页面上，并通过一个简单的命令切换页面。旧页面上的所有控件将自动隐藏，而新页面上的控件将自动显示。

要将控件分配到页面，您可以使用 GUI SETUP nn 命令，其中 nn 指的是范围为 1 到 32 的页面。当您使用此命令时，任何新创建的控件将被分配到该页面。您可以根据需要多次使用 GUI SETUP。例如，在下面的程序片段中，前两个控件将被分配到页面 1，第二个控件将被分配到页面 2，依此类推。

```
GUI SETUP 1  
GUI 标题 #1, "流量", 20, 170,, RGB(棕色), 0  
图形用户界面显示框 #2, 20, 200, 150, 45
```

图形用户界面设置 2

```
图形用户界面标题 #3, "高:", 232, 260, LT, RGB(黄色)  
图形用户界面数字框 #4, 318, 6, 90, 12, RGB(黄色), RGB(64,64,64)
```

图形用户界面设置 3

```
图形用户界面复选框 #5, "警报", 500, 285, 25  
图形用户界面复选框 #6, "警告", 500, 325, 25
```

默认情况下，只有设置为页面 1 的控件会显示，其他控件将被隐藏。

要切换屏幕到页面 3，您只需使用命令 GUIPAGE 3。这将导致控件 #1 和 #2 自动隐藏，而控件 #5 和 #6 将被显示。同样，图形用户界面页面 2 将隐藏所有控件除了 #3 和 #4，这两个控件将被显示。

您可以指定多个页面同时显示，例如，GUI PAGE 1,3 将同时显示页面 1 和 3，同时隐藏页面 2。如果您有一组必须始终可见的控件，这将非常有用。例如，GUI PAGE 1,2 和 GUI PAGE 1,3 将使页面 1 上的控件可见，而其他页面的控件则会被切换开关。

程序可以合法地修改其他页面上的控件，即使它们在当时并未显示。这包括更改值和颜色，以及禁用或隐藏它们。当显示切换到它们的页面时，控件将以其新属性显示。

可以将 GUIPAGE 命令放置在触摸下中断中，以便按下某个控件或屏幕的某个部分将切换到另一个页面。

请注意，当在诸如 GUI ENABLE ALL 的命令中使用 ALL 作为控件列表时，这仅指当前选定显示的页面上的控件。其他页面上的控件将不受影响。

所有程序都以相当于命令 GUI SETUP 1 和 GUIPAGE 1 的状态开始。这意味着如果不使用 GUI SETUP 和 GUIPAGE 命令，程序将按预期运行，所有控件将被显示。

下面显示了 GUIPAGE 命令的典型用法。

Two buttons (which are always displayed) allow the user to select between the first page and the second page. The switch is done in the touch down interrupt.

```
GUI SETUP 1
GUI Button #10, "SELECT PAGE ONE", 50, 100, 150, 30, RGB(yellow), RGB(blue)
GUI Button #11, "SELECT PAGE TWO", 50, 140, 150, 30, RGB(yellow), RGB(blue)

GUI SETUP 2
GUI Caption #1, "Displaying First Page", 20, 20

GUI SETUP 3
GUI Caption #2, "Displaying Second Page", 20, 50

Page 1, 2
GUI INTERRUPT TouchDown
Do
    ' the main program loop
Loop

Sub TouchDown
    If Touch(REF) = 10 Then GUI Page 1, 2
    If Touch(REF) = 11 Then GUI Page 1, 3
End Sub
```

Multiple Interrupts

With many screen pages the interrupt subroutine could get long and complicated. To work around that it is possible to have multiple interrupt subroutines and switch dynamically between them as you wish (normally after switching pages). This is done by redefining the current interrupt routines using the GUI INTERRUPT command.

For example, this program fragment uses different interrupt routines for pages 4 and 5 and they are specified immediately after switching the pages.

```
GUI PAGE 4
GUI INTERRUPT P4keydown, P4keyup
..
GUI PAGE 5
GUI INTERRUPT P5keydown, P5keyup
..
```

Using Basic Drawing Commands

There are two types of objects that can be on the screen. These are the GUI controls and the basic drawing objects (PIXEL, LINE, TEXT, etc). Mixing the two on the screen is not a good idea because MMBasic does not track the position of the basic drawing objects and they can clash with the GUI controls.

As a result, unless you are prepared to do some extra programming, you should use either the GUI controls or the basic drawing objects – but you should not use both. So, for example, do not use TEXT but use GUI CAPTION instead. If you only use GUI controls MMBasic will manage the screen for you including erasing and redrawing it as required, for example when a virtual keyboard is displayed.

Note that the CLS command (used to clear the screen) will automatically set any GUI controls on the screen to hidden (i.e. it does a GUI HIDE ALL before clearing the screen).

The main problem with mixing basic graphics and GUI controls occurs with the Text Box, Formatted Box and Number Box controls which display a virtual keyboard. This can erase any basic graphics and MMBasic will not know to restore them when the keyboard is removed. If you want to mix basic graphics with GUI controls you should:

- Intercept the touch down interrupt for the Text Box, Formatted Box and Number Box controls as that indicates that a virtual keyboard is about to be displayed and that will give you the opportunity to redraw your non GUI basic graphics in anticipation of this event (for example, draw them in a dimmed state to appear as if they are disabled).
- Intercept the touch up interrupt for the same controls as that indicates that the virtual keyboard has been removed and you could then redraw any non GUI graphics in their original state.

两个按钮（始终显示）允许用户在第一页和第二页之间进行选择。

切换是在触摸下中断中完成的。

GUI SETUP 1

GUI 按钮 #10, "选择第一页", 50, 100, 150, 30, RGB(黄色), RGB(蓝色)

GUI 按钮 #11, "选择第二页", 50, 140, 150, 30, RGB(黄色), RGB(蓝色)

图形用户界面设置 2

GUI 标题 #1, "显示第一页", 20, 20

图形用户界面设置 3

GUI 标题 #2, "显示第二页", 20, 50

页面 1, 2

图形用户界面中断 触摸按下

执行

 ' 主程序循环

循环

子程序 触摸按下

 如果 Touch(REF) = 10 则 图形用户界面 页面 1, 2

 如果 Touch(REF) = 11 则 图形用户界面 页面 1, 3

结束子程序

多个中断

对于许多屏幕页面，中断子程序可能会变得冗长和复杂。为了解决这个问题，可以拥有多个中断子程序，并根据需要动态切换它们（通常是在切换页面后）。这可以通过使用图形用户界面中断命令重新定义当前的中断例程来实现。

例如，这段程序片段为页面 4 和 5 使用不同的中断例程，并在切换页面后立即指定它们。

图形用户界面 页面 4

图形用户界面中断 P4keydown, P4keyup

..

图形用户界面 页面 5

图形用户界面中断 P5keydown, P5keyup

..

使用基本绘图命令

屏幕上可以有两种类型的对象。这两种类型分别是图形用户界面控件和基本绘图对象（像素、行、文本等）。将这两者混合在屏幕上并不是一个好主意，因为MMBasic不会跟踪基本绘图对象的位置，它们可能会与图形用户界面控件发生冲突。

因此，除非你准备进行一些额外的编程，否则你应该只使用图形用户界面控件或基本绘图对象——但不应该同时使用两者。例如，不要使用文本，而是使用图形用户界面标题。如果你只使用图形用户界面控件，MMBasic将为你管理屏幕，包括在需要时擦除和重新绘制屏幕，例如当虚拟键盘显示时。

请注意，CLS命令（用于清除屏幕）会自动将屏幕上的任何图形用户界面控件设置为隐藏状态（即在清除屏幕之前执行GUI HIDE ALL）。

混合基本图形和图形用户界面控件的主要问题出现在文本框、格式化框和数字框控件上，这些控件会显示虚拟键盘。这可能会擦除任何基本图形，而MMBasic将不知道在键盘移除时恢复它们。如果你想将基本图形与图形用户界面控件混合，你应该：

- 拦截文本框、格式化框和数字框控件的触摸按下中断，因为这表明虚拟键盘即将显示，这将给你机会在此事件发生之前重新绘制你的非图形用户界面基本图形（例如，将它们绘制为暗淡状态，以看起来像是被禁用）。
- 拦截相同控件的触摸抬起中断，因为这表明虚拟键盘已被移除，然后你可以将任何非图形用户界面图形重新绘制为其原始状态。

Overlapping Controls

Controls can be defined to overlap on the display, this mostly occurs with GUI AREA which, as an example, you might want to capture a touch that was intended for (say) a GUI BUTTON. This will allow you to create your own animation for the button rather than that provided by MMBasic. In this case the control that you wish to respond to the touch (i.e. GUI AREA) should have a lower reference number (i.e. #ref) than the control that it is covering (i.e. the GUI BUTTON). This is because when the screen is touched MMBasic will check the current list of active controls starting with control number 1 and working upwards. When a match is made MMBasic will take the appropriate action and terminate the search. This results in the lower numbered control effectively masking out a higher numbered control covering the same screen area as the touched location.

重叠控件

控件可以定义为在显示上重叠，这主要发生在图形用户界面区域，例如，你可能想捕捉原本打算用于（例如）图形用户界面按钮的触摸。这将允许你为按钮创建自己的动画，而不是使用MMBasic提供的动画。在这种情况下，你希望响应触摸的控件（即图形用户界面区域）应该具有比其覆盖的控件（即图形用户界面按钮）更低的参考编号（即#ref）。这是因为当屏幕被触摸时，MMBasic将从控件编号1开始检查当前活动控件列表，并向上工作。当匹配成功时MMBasic将采取适当的行动并终止搜索。这导致较低编号的控件有效地遮蔽了覆盖相同屏幕区域的较高编号控件，该区域是触摸的位置。

MMBasic Characteristics

Naming Conventions

Command names, function names, labels, variable names, etc are not case sensitive, so that "Run" and "RUN" are equivalent and "dOO" and "Doo" refer to the same variable.

The type of a variable can be specified in the DIM command or by adding a suffix to the end of the variable's name. For example the suffix for an integer is '%' so if a variable called nbr% is automatically created it will be an integer. There are three types of variables:

1. Floating point. These can store a number with a decimal point and fraction (e.g. 45.386) and also very large numbers. However, they will lose accuracy when more than 14 significant digits are stored or manipulated. The suffix is '!' and floating point is the default when a variable is created without a suffix
2. 64-bit integer. These can store numbers with up to 19 decimal digits without losing accuracy but they cannot store fractions (i.e. the part following the decimal point). The suffix for an integer is '%'.
3. Strings. These will store a string of characters (e.g. "Tom"). The suffix for a string is the '\$' symbol (e.g. name\$, s\$, etc) Strings can be up to 255 characters long.

Variable names and labels can start with an alphabetic character or underscore and can contain any alphabetic or numeric character, the period (.) and the underscore (_). They may be up to 31 characters long. A variable name or a label must not be the same as a command or a function or one of the following keywords: THEN, ELSE, TO, STEP, FOR, WHILE, UNTIL, MOD, NOT, AND, OR, XOR, AS. E.g. step = 5 is illegal.

For file names see the section “MMBasic Support for Flash and SD Card Filesystems”

Constants

Numeric constants may begin with a numeric digit (0-9) for a decimal constant, &H for a hexadecimal constant, &O for an octal constant or &B for a binary constant. For example &B1000 is the same as the decimal constant 8. Constants that start with &H, &O or &B are always treated as 64-bit integer constants.

Decimal constants may be preceded with a minus (-) or plus (+) and may be terminated with 'E' followed by an exponent number to denote exponential notation. For example 1.6E+4 is the same as 16000.

If the decimal constant contains a decimal point or an exponent, it will be treated as a floating point constant; otherwise it will be treated as a 64-bit integer constant.

String constants are surrounded by double quote marks (""). E.g. "Hello World".

Implementation Characteristics

Maximum program size (as plain text) is 160KB. Note that MMBasic tokenises the program when it is stored in flash so the final size in flash might vary from the plain text size.

Maximum length of a command line is 255 characters.

Maximum length of a variable name or a label is 31 characters.

Maximum number of dimensions to an array is 6.

Maximum number of arguments to commands that accept a variable number of arguments is 50.

Maximum number of nested FOR...NEXT loops is 20.

Maximum number of nested DO...LOOP commands is 20.

Maximum number of nested GOSUBs, subroutines and functions (combined) is 320.

Maximum number of nested multiline IF...ELSE...ENDIF commands is 20.

Maximum number of user defined labels, subroutines and functions (combined): 224

Maximum number of interrupt pins that can be configured: 10

Numbers are stored and manipulated as double precision floating point numbers or 64-bit signed integers. The range of floating point numbers is 1.797693134862316e+308 to 2.225073858507201e-308.

The range of 64-bit integers (whole numbers) that can be manipulated is ± 9223372036854775807 .

Maximum string length is 255 characters.

Maximum line number is 65000.

Maximum number of background pulses launched by the PULSE command is 5.

Maximum number of global variables and constants is 256

Maximum number of local variables is 256

The maximum number of files that can be listed by the FILES command is 1000

MMBasic特性

命名约定

命令名称、函数名称、标签、变量名称等不区分大小写，因此"Run"和"RUN"是等效的，而"DOO"和"Doo"指的是同一个变量。

变量的类型可以在DIM命令中指定，或者通过在变量名称的末尾添加后缀来指定。例如，整数的后缀是'%'，因此如果自动创建一个名为nbr%的变量，它将是一个整数。变量有三种类型：

1. 浮点。这些可以存储带小数点和分数的数字（例如45.386），以及非常大的数字。然而，当存储或操作超过14个有效数字时，它们将失去精度。后缀是'!'，当创建变量时，如果没有后缀，浮点数是默认值2.64位整数。这些可以存储最多19位小数的数字而不失去精度，但它们不能存储分数（即小数点后的部分）。整数的后缀是'%3.字符串。这些将存储一串字符（例如"汤姆"）。字符串的后缀是符号（

例如 name\$，s\$ 等）。字符串长度可以达到255个字符。

变量名称和标签可以以字母字符或下划线开头，并且可以包含任何字母或数字字符、句点(.) 和下划线(_)。它们的长度可以达到31个字符。变量名称或标签不得与命令、函数或以下关键字相同：THEN、ELSE、TO、STEP、FOR、WHILE、UNTIL、MOD、NOT、AND、OR、XOR、AS。例如 step = 5 是非法的。有关文件名，请参见“MMBasic对Flash和SD卡文件系统的支持”一节。

常量

数字常量可以以数字(0-9)开头表示十进制常量，以&H表示十六进制常量，以&O表示八进制常量或以&B表示二进制常量。例如 &B1000 与十进制常量 8 是相同的。以&H、&O或&B开头的常量始终被视为64位整数常量。

十进制常量可以前面加上负号(-)或正号(+)，并且可以以'E'后跟指数数字来表示指数表示法。例如 1.6E+4 与 16000 是相同的。

如果十进制常量包含小数点或指数，它将被视为浮点常量；否则，它将被视为64位整数常量。

字符串常量用双引号(“)括起来。例如："你好，世界"。

实现特性

最大程序大小（以纯文本形式）为160KB。请注意，MMBasic在程序存储到闪存时会进行标记化，因此闪存中的最终大小可能与纯文本大小有所不同。

命令行的最大长度为255个字符。

变量名称或标签的最大长度为31个字符。

数组的最大维度数为6。

接受可变数量参数的命令的最大参数数量为50。

嵌套的FOR...NEXT循环的最大数量是20。

嵌套的DO...LOOP命令的最大数量是20。

嵌套的GOSUB、子程序和函数（合计）的最大数量是320。

嵌套的多行IF...ELSE...ENDIF命令的最大数量是20。

用户定义的标签、子程序和函数（合计）的最大数量：224

可配置的中断引脚的最大数量：10

数字以双精度浮点数或64位有符号整数的形式存储和处理。浮点数的范围是1.797693134862316e+308到2.225073858507201e-308。可以处理的64位整数（整数）的范围是±9223372036854775807。

字符串的最大长度为255个字符。

最大行号为65000。

PULSE命令启动的背景脉冲的最大数量为5。

全局变量和常量的最大数量为256。

局部变量的最大数量为256

FILES命令可以列出的最大文件数量为1000

The maximum length filename supported is 63 characters

Compatibility

MMBasic implements a large subset of Microsoft's GW-BASIC. There are numerous differences due to physical and practical considerations but most standard BASIC commands and functions are essentially the same. An online manual for GW-BASIC is available at <http://www.antonis.de/qbebooks/gwbasman/index.html> and this provides a more detailed description of the commands and functions.

MMBasic also implements a number of modern programming structures documented in the ANSI Standard for Full BASIC (X3.113-1987) or ISO/IEC 10279:1991. These include SUB/END SUB, the DO WHILE ... LOOP, the SELECT...CASE statements and structured IF . THEN ... ELSE ... ENDIF statements.

支持的最大文件名长度为63个字符

兼容性

MMBasic实现了微软GW-BASIC的大部分子集。由于物理和实际考虑，存在许多差异，但大多数标准BASIC命令和函数基本相同。GW-BASIC的在线手册可在<http://www.antonis.de/qbebooks/gwbasman/index.html>获取，其中提供了命令和函数的更详细描述。

MMBasic还实现了一些在ANSI标准的完整BASIC（X3.113-1987）或ISO/IEC 10279:1991中记录的现代编程结构。这些包括SUB/END SUB、DO WHILE ... LOOP、SELECT...CASE语句和结构化的IF. THEN ... ELSE ... ENDIF语句。

Predefined Read Only Variables

These variables are set by MMBasic and cannot be changed by the running program. Note that they do not do anything on their own, they must be printed or assigned to a variable.

For example:

```
> PRINT MM.VER  
5.0707  
>
```

or, in a program:

```
IF MM.VER < 5.0707 Then Error( "Needs version 5.07.07 or greater" )
```

MM.VER	The version number of the firmware as a floating point number in the form aa.bbcc where aa is the major version number, bb is the minor version number and cc is the revision number. For example version 5.03.00 will return 5.03 and version 5.03.01 will return 5.0301.
MM.CMDLINE\$	This constant variable containing any command line arguments passed to the current program is automatically created when an MMBasic program runs; see RUN and * commands for details. <ul style="list-style-type: none">• Programs run from the Editor or using OPTION AUTORUN will set MM.CMDLINE\$ to the empty string.• If not required this constant variable may be removed from memory using ERASE MM.CMDLINE\$
MM.DEVICE\$	A string representing the device or platform that MMBasic is running on. Currently this variable will contain one of the following: "Maxomite" on the standard Maxomite and compatibles. "Colour Maxomite" on the Colour Maxomite and UBW32. "Colour Maxomite 2" on the Colour Maxomite 2. "DuinoMite" when running on one of the DuinoMite family. "DOS" when running on Windows in a DOS box. "Generic PIC32" for the generic version of MMBasic on a PIC32. "Micromite" on the PIC32MX150/250 "Micromite MkII" on the PIC32MX170/270 "Micromite Plus" on the PIC32MX470 "Micromite Extreme" on the PIC32MZ series "ARMMite H7" on the ARMMiteH7 "ARMMite F407" on the ARMMiteF4 "ARMMite L4" with chip no. and pin count appended on the ARMMiteL4 "PicoMite" on the Raspberry Pi Pico "WebMite" on the Raspberry Pi Pico W "PicoMiteVGA" on the Raspberry Pi Pico VGA Edition "MMBasic for Windows" on the Windows version
MM.ERRNO MM.ERRMSG\$	If a statement caused an error which was ignored these variables will be set accordingly. MM.ERRNO is a number where non zero means that there was an error and MM.ERRMSG\$ is a string representing the error message that would have normally been displayed on the console. They are reset to zero and an empty string by RUN, ON ERROR IGNORE or ON ERROR SKIP.
MM.INFO() MM.INFO\$()	These two versions can be used interchangeably but good programming practice would require that you use the one corresponding to the returned datatype.

预定义只读变量

这些变量由MMBasic设置，无法通过运行的程序更改。请注意，它们本身并不执行任何操作，必须打印或赋值给变量。

例如：

```
> 打印 MM.VER  
5.0707  
>
```

或者，在程序中：

```
如果 MM.VER < 5.0707 那么 错误( "需要版本 5.07.07 或更高" )
```

MM.VER	固件的版本号以浮点数形式表示，格式为 aa.bbcc，其中 aa 是主版本号，bb 是次版本号，cc 是修订号。例如，版本 5.03.00 将返回 5.03，而版本 5.03.01 将返回 5.0301。
MM.CMDLINE\$	<p>此常量变量包含传递给当前程序的任何命令行参数，在 MMBasic 程序运行时自动创建；有关详细信息，请参见 RUN 和 * 命令。</p> <ul style="list-style-type: none">从编辑器运行的程序或使用自动运行选项将 MM.CMDLINE\$ 设置为空字符串。如果不需，此常量变量可以使用擦除 MM.CMDLINE\$ 从内存中移除
MM.DEVICE\$	一个表示MMBasic运行所在设备或平台的字符串。 目前此变量将包含以下之一： "Maxomite" 在标准Maxomite及其兼容设备上。 "颜色最大化" 在颜色最大化和UBW32上。"颜色最大化2" 在颜色最大化2上。 "DuinoMite" 当在DuinoMite系列之一上运行时。 "DOS" 当在Windows的DOS窗口中运行时。 "通用PIC32" 为在PIC32上的MMBasic通用版本。 "Micromite" 在PIC32MX150/250上 "MicromiteMkII" 在PIC32MX170/270上 "MicromitePlus" 在PIC32MX470上 "Micromite Extreme" 在PIC32MZ系列上 "ARMMite H7" 在ARMMiteH7上 "ARMMite F407" 在ARMMiteF4上 "ARMMite L4" 附加了芯片编号和引脚计数在ARMMiteL4上 "PicoMite" 在树莓派 Pico 上 "WebMite" 在树莓派 Pico W 上 "PicoMiteVGA" 在树莓派 Pico VGA 版上 "MMBasic for Windows" 在Windows版本上
MM.ERRNO MM.ERRMSG\$	如果某个语句导致了一个被忽略的错误，这些变量将相应地被设置。MM.ERRNO是一个数字，非零表示发生了错误，而MM.ERRMSG\$是一个字符串，表示通常会在控制台上显示的错误信息。它们被重置为零和一个空字符串通过RUN、ON ERROR IGNORE或ON ERROR SKIP。
MM.INFO() MM.INFO\$()	这两个版本可以互换使用，但良好的编程实践要求您使用与返回的数据类型相对应的版本。

MM.INFO\$(AUTORUN)	Returns the setting of the OPTION AUTORUN command
MM.INFO(ADC)	Returns the number of the buffer currently ready to read when using ADC RUN (1 or 2). Returns 0 if nothing ready.
MM.INFO(ADC DMA)	Returns true (1) if the ADC DMA is active.
MM.INFO(BOOT COUNT)	Returns the number of times the Pico has been restarted since the flash drive was last formatted
MM.INFO\$(CPUSPEED)	Returns the CPU speed as a string.
MM.INFO\$(LCDPANEL)	Returns the name of the configured LCD panel or a blank string.
MM.INFO\$(SDCARD)	Returns the status of the SD Card. Valid returns are: DISABLED, NOT PRESENT, READY, and UNUSED
MM.INFO\$(CURRENT)	Returns the name of the current program or NONE if called after a NEW or EDIT Command.
MM.INFO(PATH)	Returns the path of the current program or NONE if called after a NEW or EDIT Command.
MM.INFO(DISK SIZE)	Returns the capacity of the Flash Filesystem or SD Card, whichever is the active drive, in bytes
MM.INFO(EXISTS FILE fname\$)	Returns 1 if the file specified exists, returns -1 if fname\$ is a directory, otherwise returns 0.
MM.INFO(EXISTS DIR dirname\$)	Returns a Boolean indicating whether the directory specified exists.
MM.INFO(FREE SPACE)	Returns the free space on the Flash Filesystem or SD Card whichever is the active drive.
MM.INFO(FILESIZE file\$)	Returns the size of file\$ in bytes or -1 if not found, -2 if a directory.
MM.INFO\$(MODIFIED file\$)	Returns the date/time that file\$ was modified, Empty string if not found.
MM.INFO(FCOLOUR)	Returns the current foreground colour.
MM.INFO(BCOLOUR)	Returns the current background colour.
MM.INFO(FONT)	Returns the number of the currently active font.
MM.INFO(FONT ADDRESS n)	Returns the address of the memory location with the address of FONT n .
MM.INFO(FONT POINTER n)	Returns a POINTER to the start of FONT n in memory.
MM.INFO(FONTHEIGHT) MM.INFO(FONTWIDTH)	Integers representing the height and width of the current font (in pixels).
MM.INFO(FLASH)	Reports which flash slot the program was loaded from if applicable.
MM.INFO(FLASH ADDRESS n)	Returns the address of the flash slot n.
MM.INFO(HEAP)	Returns the amount of MMbasic Heap memory free. MMBasic heap is used for strings, arrays and various other temporary and permanent buffers (e.g.

MM.INFO\$(AUTORUN)	返回OPTION AUTORUN命令的设置
MM.INFO(ADC)	返回使用ADC时当前准备读取的缓冲区编号 RUN (1或2)。如果没有准备好，返回0。
MM.INFO(ADC DMA)	如果ADC DMA处于活动状态，返回真 (1)。
MM.INFO(BOOT COUNT)	返回自上次格式化闪存驱动器以来，Pico重启的次数。
MM.INFO\$(CPUSPEED)	返回CPU速度作为字符串。
MM.INFO\$(LCDPANEL)	返回配置的液晶面板的名称或一个空字符串。
MM.INFO\$(SDCARD)	返回SD卡的状态。有效的返回值有： 禁用、未插入、准备就绪和未使用
MM.INFO\$(CURRENT)	返回当前程序的名称，如果在 NEW 或 EDIT 命令后调用则返回 NONE。
MM.INFO(PATH)	返回当前程序的路径，如果在 NEW 或 EDIT 命令后调用则返回 NONE。
MM.INFO(DISK SIZE)	返回 Flash 文件系统或 SD 卡（以活动驱动器为准）的容量，单位为字节。
MM.INFO(EXISTS FILE fname\$)	如果指定的文件存在则返回 1，如果 fname\$ 是一个目录则返回 -1，否则返回 0。
MM.INFO(EXISTS DIR dirname\$)	返回一个布尔值，指示指定的目录是否存在。
MM.INFO(FREE SPACE)	返回 Flash 文件系统或 SD 卡（以活动驱动器为准）的可用空间。
MM.INFO(FILESIZE file\$)	返回 file\$ 的大小（单位为字节），如果未找到则返回 -1，如果是目录则返回 -2。
MM.INFO\$(MODIFIED file\$)	返回 file\$ 的修改日期/时间，如果未找到则返回空字符串。
MM.INFO(FCOLOUR)	返回当前的前景颜色。
MM.INFO(BCOLOUR)	返回当前的背景颜色。
MM.INFO(FONT)	返回当前活动字体的编号。
MM.INFO(FONT 地址 n)	返回存储 FONT n 地址的内存位置的地址。
MM.INFO(FONT 指针 n)	返回指向内存中 FONT n 开始位置的指针。
MM.INFO(FONTHEIGHT) MM.INFO(FONTWIDTH)	表示当前字体高度和宽度的整数（以像素为单位）。
MM.INFO(FLASH)	如果适用，报告程序是从哪个闪存槽加载的。
MM.INFO(FLASH 地址 n)	返回闪存槽 n 的地址。
MM.INFO(HEAP)	返回可用的 MMBasic 堆内存量。MMBasic 堆用于字符串、数组以及各种其他临时和永久缓冲区（例如。

	audio)
MM.INFO(HPOS) MM.INFO(VPOS)	The current horizontal and vertical position (in pixels) following the last graphics or print command.
MM.INFO(ID)	Returns the unique ID of the Pico.
MM.INFO(OPTION option)	Returns the current value of a range of options that affect how a program will run. “option” can be one of AUTORUN, BASE, BREAK, DEFAULT, EXPLICIT, KEYBOARD, ANGLE, HEIGHT, WIDTH, FLASH SIZE
MM.INFO\$(PIN pinno)	Returns the status of I/O pin 'pinno'. Valid returns are: INVALID, RESERVED, IN USE, and UNUSED
MM.INFO(PINNO GPnn)	Returns the physical pin number for a given GP number. GPnn can be an unquoted string (GP01), a string literal("GP01") or a string variable. Ie, A\$="GP01": MM.INFO(PINNO A\$).
MM.INFO\$(PLATFORM)	Returns the string previously set with OPTION PLATFORM.
MM.INFO(PS2)	Reports the last raw message received on the PS2 interface if enabled.
MM.INFO\$(SOUND)	Returns the current activity on the audio output (OFF, PAUSED, TONE, WAV, FLAC, SOUND)
MM.INFO(STACK)	Returns the C stack pointer. Complex or recursive Basic code may result in the error "Stack overflow, expression too complex at depth %". This will occur when the stack is below &H 2003f800. Monitoring the stack will allow the programmer to identify simplifications to the Basic code to avoid the error.
MM.INFO(SYSTEM HEAP)	Returns the free space on the System Heap.
MM.INFO(TRACK)	Returns the name of the FLAC, MP3, WAV or MIDI file currently playing on the audio output
MM.INFO\$(TOUCH)	Returns the status of the Touch controller. Valid returns are: “Disabled”, “Not calibrated”, and “Ready”.
MM.INFO(VARCNT)	Returns the number of variables in use in the MMBasic program.
MM.INFO\$(LINE)	Returns the current line number as a string. LIBRARY returned if in the Library and UNKNOWN if not in a program. Assists in diagnostics while unit testing.
MM.INFO(UPTIME)	Returns the time in seconds since booted as a floating point number.
MM.INFO(VERSION)	Returns the version number as a floating point number.
MM.INFO(WRITEBUFF)	Returns the address in memory of the current buffer used for drawing commands.

	音频)
MM.INFO(HPOS)	当前的水平和垂直位置（以像素为单位），在最后一个图形或打印命令之后。
MM.INFO(VPOS)	
MM.INFO(ID)	返回 Pico 的唯一 ID。
MM.INFO(OPTION option)	返回影响程序运行方式的一系列选项的当前值。“option”可以是 AUTORUN、BASE、BREAK、DEFAULT、EXPLICIT、KEYBOARD、ANGLE、HEIGHT、WIDTH、FLASH SIZE 之一。
MM.INFO\$(PIN pinno)	返回 I/O 引脚 'pinno' 的状态。有效的返回值有： 无效、保留、正在使用和未使用
MM.INFO(PINNO GPnn)	返回给定 GP 编号的物理引脚编号。GPnn 可以是一个未加引号的字符串 (GP01)、字符串字面量 (“GP01”) 或字符串变量。即，A\$=”GP01”: MM.INFO(PINNO A\$)。
MM.INFO\$(PLATFORM)	返回之前通过选项 PLATFORM 设置的字符串。
MM.INFO(PS2)	如果启用，将报告在 PS2 接口上接收到的最后一条原始消息。
MM.INFO\$(SOUND)	返回音频输出的当前活动状态（关闭、暂停、音调、WAV、FLAC、声音）。
MM.INFO(STACK)	返回 C 栈指针。复杂或递归的 Basic 代码可能导致错误 "栈溢出，表达式在深度 % 时过于复杂"。当栈低于 &H 2003f800 时，将发生此错误。监控栈将允许程序员识别 Basic 代码中的简化，以避免该错误。
MM.INFO(SYSTEM HEAP)	返回系统堆栈上的可用空间。
MM.INFO(TRACK)	返回当前在音频输出上播放的FLAC、MP3、WAV或MIDI文件的名称。
MM.INFO\$(TOUCH)	返回触摸控制器的状态。有效的返回值为：“禁用”、“未校准”和“准备就绪”。
MM.INFO(VARCNT)	返回MMBasic程序中正在使用的变量数量。
MM.INFO\$(LINE)	返回当前行号作为字符串。如果在库中则返回LIBRARY，如果不在程序中则返回UNKNOWN。在单元测试时协助诊断。
MM.INFO(UPTIME)	返回自启动以来经过的时间（以浮点数表示，单位为秒）。
MM.INFO(VERSION)	返回版本号（以浮点数表示）。
MM.INFO(WRITEBUFF)	返回当前用于绘图命令的缓冲区在内存中的地址。

MM.HRES MM.VRES	Integers representing the horizontal and vertical resolution of the LCD display panel (if configured) in pixels.
MM.FONTHEIGHT MM.FONTWIDTH	Integers representing the height and width of the current font (in pixels) kept for compatibility. NB: these are automatically converted into MM.INFO(FONTHEIGHT) and MM.INFO(FONTWIDTH) by MMBasic.
MM.ONewire	Following a 1-Wire reset function this integer variable will be set to indicate the result of the operation: 0 = Device not found, 1 = Device found.
MM.I2C	Following an I2C write or read command this integer variable will be set to indicate the result of the operation as follows: 0 = The command completed without error. 1 = Received a NACK response 2 = Command timed out
MM.WATCHDOG	An integer which is true if MMBasic was restarted as the result of a Watchdog timeout (see the WATCHDOG command) otherwise false.

MM.HRES MM.VRES	表示液晶显示面板（如果已配置）水平和垂直分辨率的整数（以像素为单位）。
MM.FONTHEIGHT MM.FONTWIDTH	表示当前字体高度和宽度的整数（以像素为单位），以保持兼容性。 注意：这些会自动转换为MM.INFO(FONTHEIGHT)和M.INFO(FONTWIDTH)由MMBasic处理。
MM.ONewire	在1-Wire重置函数之后，此整数变量将被设置以指示操作的结果：0 = 设备未找到，1 = 设备已找到。
MM.I2C	在 I2C 写入或读取命令之后，这个整数变量将被设置以指示操作的结果，如下所示： 0 = 命令成功完成，没有错误。 1 = 收到NACK响应 2 = 命令超时
MM.WATCHDOG	一个整数，如果MMBasic因看门狗超时而重新启动，则为真（请参见WATCHDOG命令），否则为假。

Options

This table lists the various option commands which can be used to configure MMBasic and change the way it operates. Options that are marked as permanent will be saved in non-volatile memory and automatically restored when the PicoMite is restarted. Options that are not permanent will be reset on start-up.

Many OPTION commands will force a restart of the PicoMite and that will cause the USB console interface to be reset. The program in memory will not be lost as it is held in non-volatile flash memory.

Permanent?		
OPTION DISK SAVE fname\$ OPTION DISK LOAD fname\$	✓	These commands let the user save and restore the complete set of options defined to and from a disk file. The file could then be transferred to a host computer using XMODEM allowing additional PicoMites to be easily configured or options recovered after a firmware upgrade
OPTION ANGLE RADIANS DEGREES		This command switches trig functions between degrees and radians. Acts on SIN, COS, TAN, ATN, ATAN2, MATH ATAN3, ACOS, ASIN
OPTION AUDIO PWMnApin, PWMnBpin or OPTION AUDIO DISABLE	✓	Configures one of the PWM channels as an audio output. 'PWMnApin' is the left audio channel, 'PWMnBpin' is the right. Both pins must belong to the same audio channel. Example, OPTION AUDIO GP18, GP19 would use PWM1A and PWM1B on pins 24 and 25 respectively. This option prevents use of these pins in the BASIC program. The audio output is generated using PWM so a low pass filter is necessary on the output. The audio output from the PicoMite is very noisy. Using OPTION POWER and/or supplying power via a separate 3.3V linear regulator can reduce this. This command must be run at the command prompt (not in a program).
OPTION AUDIO SPI CSpin, CLKpin, MOSIpin or OPTION AUDIO DISABLE	✓	Configures the audio output to be directed to a MCP48n2 DAC connected to the specified pins. The LDAC pin on the DAC should be connected to GND.
OPTION AUDIO VS1053 CLKpin, MOSIpin, MISOpin, XCSpin, XDCSpin, DREQpin, XRSTpin or OPTION AUDIO DISABLE	✓	Configures the audio output to be directed to a VS1053 CODEC. This allows MP3 and MIDI playback in addition to the other formats supported and also supports real-time MIDI output. See the PLAY command for more details
OPTION AUTOREFRESH OFF ON		Black and white displays can only be updated a full screen at a time. By using OPTION AUTOREFRESH OFF/ON you can control whether a write command immediately updates the display or not. If AUTOREFRESH is OFF the REFRESH command can be used to trigger the write. This applies to the following displays: N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI, ST7920

选项

此表列出了可以用于配置MMBasic并更改其操作方式的各种选项命令。标记为永久的选项将保存在非易失性内存中，并在PicoMite重新启动时自动恢复。非永久选项将在启动时重置。

许多OPTION命令将强制重启PicoMite，这将导致USB控制台接口被重置。内存中的程序不会丢失，因为它保存在非易失性闪存中。

永久?		
OPTION DISK SAVE fname\$ OPTION DISK LOAD fname\$	✓	这些命令允许用户将定义的完整选项集保存到磁盘文件中并恢复。然后可以使用XMODEM将文件传输到主机计算机，从而轻松配置其他PicoMite或在固件升级后恢复选项。
OPTION ANGLE RADIANS DEGREES		此命令在度和弧度之间切换三角函数。 作用于SIN、COS、TAN、ATN、ATAN2、MATH ATAN3、ACOS、ASIN
OPTION AUDIO PWMnApin, PWMnBpin 或 OPTION AUDIO DISABLE	✓	将其中一个PWM通道配置为音频输出。 ‘PWMnApin’是左音频通道，‘PWMnBpin’是右音频通道。两个引脚必须属于同一个音频通道。 例如，OPTION AUDIO GP18,GP19 将分别使用引脚 24 和 25 上的 PWM1A 和 PWM1B。 此选项防止在 BASIC 程序中使用这些引脚。 音频输出是通过 PWM 生成的，因此输出上需要一个低通滤波器。来自 PicoMite 的音频输出非常嘈杂。使用 OPTION POWER 和/或通过单独的 3.3V 线性稳压器供电可以减少这种情况。 此命令必须在命令提示符下运行（而不是在程序中）。
OPTION AUDIO SPI CSpin, CLK引脚, MOSI引脚 或 OPTION AUDIO DISABLE	✓	将音频输出配置为定向到连接到指定引脚的 MCP48n2 DAC。DAC 上的 LDAC 引脚应连接到接地。
OPTION AUDIO VS1053 CLK引脚, MOSI引脚, MIS引脚, XCS引脚, XDC引脚, DREQ引脚, XRST引脚 或 OPTION AUDIO DISABLE	✓	将音频输出配置为指向VS1053 编解码器。这允许除了支持的其他格式外，还可以播放MP3和MIDI，并支持实时MIDI输出。有关更多详细信息，请参见PLAY命令
选项 自动刷新 关闭 开启		黑白显示器只能一次更新整个屏幕。通过使用选项 自动刷新 关闭/开启，您可以控制写入命令是否立即更新显示。如果自动刷新关闭，则可以使用刷新命令来触发写入。这适用于以下显示器：N5110, SSD1306I2C, SSD1306I2C32, SSD1306 SPI, ST7920

OPTION AUTORUN ON [,NORESET] or OPTION AUTORUN n [,NORESET] or OPTION AUTORUN OFF	✓	Instructs MMBasic to automatically run a program on power up or restart. ON will cause the current program to be run. Specifying ‘n’ will cause that location in flash memory to be run. ‘n’ must be in the range 1 to 3. Specifying the optional parameter “NORESET” will maintain AUTORUN even if the program causes a system error (by default this will cause the firmware to cancel any OPTION AUTORUN setting). OFF will disable the autorun option and is the default for a new program. Entering the break key (default CTRL-C) at the console will interrupt the running program and return to the command prompt.
OPTION BASE 0 1		Set the lowest value for array subscripts to either 0 or 1. This must be used before any arrays are declared and is reset to the default of 0 on power up.
OPTION BAUDRATE nn		Set the baudrate of the serial console (if it is configured).
OPTION BREAK nn		Set the value of the break key to the ASCII value 'nn'. This key is used to interrupt a running program. The value of the break key is set to CTRL-C key at power up but it can be changed to any keyboard key using this command (for example, OPTION BREAK 4 will set the break key to the CTRL-D key). Setting this option to zero will disable the break function entirely.
OPTION CASE LOWER UPPER TITLE	✓	Change the case used for listing command and function names when using the LIST command. The default is TITLE but the old standard of MMBasic can be restored using OPTION CASE UPPER.
OPTION COLOURCODE ON or OPTION COLOURCODE OFF	✓	Turn on or off colour coding for the editor's output. Keywords will be in cyan, comments in yellow, etc. The default is OFF. The keyword COLORCODE (USA spelling) can also be used. This requires a terminal emulator that can interpret the appropriate escape codes (eg, Tera Term). This command must be run at the command prompt (not in a program).
OPTION CPUSPEED speed	✓	Change the CPU clock. ‘speed’ is the CPU clock in KHz in the range of 48000 to 378000. Speeds above 133MHz are regarded as overclocking as that is the specified maximum speed of the standard Raspberry Pi Pico. With no option set the CPU speed will default to 133000. This command must be run at the command prompt (not in a program).
OPTION COUNT pin1, pin2, pin3, pin4	✓	Specifies which pins are to be used as Count inputs. By default these are GP6, GP7, GP8 and GP9. The SETPIN command defines the Counter mode. This command must be run at the command prompt (not in a program).

选项 自动运行开启 [,不重置] 或 选项 自动运行 n [,不重置] 或 选项 自动运行 关闭	✓	<p>指示MMBasic在开机或重启时自动运行程序。</p> <p>ON将导致当前程序被运行。</p> <p>指定'n'将导致闪存中的该位置被运行。'n'必须在1到3的范围内。</p> <p>指定可选参数“NORESET”将保持自动运行，即使程序导致系统错误（默认情况下，这将导致固件取消任何OPTION AUTORUN设置）。</p> <p>OFF将禁用自动运行选项，并且是新程序的默认设置。</p> <p>在控制台输入中断键（默认是CTRL-C）将中断正在运行的程序并返回到命令提示符。</p>
OPTION BASE 0 1		将数组下标的最低值设置为0或1。 这必须在声明任何数组之前使用，并在上电时重置为默认值0。
OPTION BAUDRATE nn		设置串行控制台的波特率（如果已配置）。
OPTION BREAK nn		将中断键的值设置为ASCII值'nn'。此键用于中断正在运行的程序。 中断键的值在开机时设置为CTRL-C键，但可以使用此命令更改为任何键盘键（例如，OPTION BREAK 4将中断键设置为CTRL-D键）。将此选项设置为零将完全禁用中断功能。
OPTION CASE LOWER UPPER TITLE	✓	更改使用LIST命令时列出命令和函数名称的大小写。默认值为TITLE，但可以使用OPTION CASE UPPER恢复MMBasic的旧标准。
OPTION COLOURCODE ON 或 OPTION COLOURCODE OFF	✓	开启或关闭编辑器输出的颜色编码。关键字将显示为青色，注释为黄色，等等。默认值为OFF。也可以使用关键字COLORCODE（美国拼写）。 这需要一个能够解释适当转义代码的终端仿真器（例如，Term a Term）。此命令必须在命令提示符下运行（而不是在程序中）。
选项 CPU速度 speed	✓	更改CPU时钟。 'speed'是CPU时钟，单位为千赫，范围为48000到378000。超过133MHz的速度被视为超频，因为这是标准树莓派Pico的指定最大速度。 如果没有设置选项，CPU速度将默认为133000。此命令必须在命令提示符下运行（而不是在程序中）。
选项 计数引脚1, 引脚2, 引脚3, 引脚4	✓	指定哪些引脚用作计数输入。默认情况下，这些引脚是GP6, GP7, GP8 和 GP9。SETPIN命令定义计数器模式。 此命令必须在命令提示符下运行（而不是在程序中）。

OPTION DEFAULT FLOAT INTEGER STRING NONE		Used to set the default type for a variable which is not explicitly defined. If OPTION DEFAULT NONE is used then all variables must have their type explicitly defined or the error “Variable type not specified” will occur. When a program is run the default is set to FLOAT for compatibility with Microsoft BASIC and previous versions of MMBasic.
OPTION DISPLAY lines [,chars]	✓	Set the characteristics of the display terminal used for the console. Both the LIST and EDIT commands need to know this information to correctly format the text for display. 'lines' is the number of lines on the display and 'chars' is the width of the display in characters. The default is 24 lines x 80 chars and when changed this option will be remembered even when the power is removed. Maximum values are 100 lines and 240chars. This command is not available if the display is being used as a console.
OPTION ESCAPE		Enables the ability to insert escape sequences into string constants. See the section <i>Special Characters in Strings</i> .
OPTION EXPLICIT		Placing this command at the start of a program will require that every variable be explicitly declared using the DIM, LOCAL or STATIC commands before it can be used in the program. This option is disabled by default when a program is run. If it is used it must be specified before any variables are used.
OPTION FAST AUDIO ON/OFF		When using the PLAY SOUND command, changes to sounds, volumes, or frequencies can cause audible clicks in the output. The firmware attempts to mitigate this by ramping the volume down on the channel's previous output before changing the output and ramping it back up again. This significantly improves the audio output but at the expense of a short delay in the PLAY SOUND command (worst case 3mSec). This delay can be avoided using OPTION FAST AUDIO ON in a program. The audible clicks may then re-appear but this is at the programmer's discretion. This is a temporary option that is reset to OFF whenever a program is run.
OPTION FNKey string\$	✓	Define the string that will be generated when a function key is pressed at the command prompt. ‘FNKey’ can be F1, and F5 thru to F9. Example: OPTION F8 “RUN “+chr\$(34)+”myprog” +chr\$(34)+chr\$(13)+chr\$(10). This command must be run at the command prompt (not in a program).
OPTION GUI CONTROLS NbrOfGUIControls	✓	Specifies the maximum number of GUI controls that can be defined. Each control uses 52 bytes and the total memory used must be rounded up to the next 2048 byte multiple. For example, specifying 70 controls will use 4KB of RAM. By default the number of GUI controls is set to zero so this option must be used before any GUI controls are defined.
OPTION HEARTBEAT ON/OFF	✓	Enables or disables the output of the heartbeat on GP25

选项 默认浮点数 整数 字符串 无		用于设置未明确定义的变量的默认类型。 如果使用选项 默认 无，则所有变量必须明确定义其类型，否则将出现错误“变量类型未指定”。 当程序运行时，默认设置为FLOAT，以兼容微软BASIC和之前版本的MMBasic。
OPTION DISPLAY lines [,chars]	✓	设置用于控制台的显示终端的特性。LIST和EDIT命令都需要知道这些信息，以正确格式化文本以供显示。 'lines'是显示的行数，'chars'是显示的字符宽度。默认值为24行 x 80字符，当更改时，此选项将在断电后仍然被记住。最大值为100行和240字符。 如果显示被用作控制台，则此命令不可用。
OPTION ESCAPE		启用将转义序列插入字符串常量的能力。请参见字符串中的特殊字符部分。
选项 显式		将此命令放在程序的开头将要求在程序中使用之前，必须使用DIM、LOCAL或STATIC命令显式声明每个变量。 当程序运行时，此选项默认情况下是禁用的。如果使用此选项，必须在使用任何变量之前指定。
选项 快速音频 开启/关闭		使用播放声音命令时，对声音、音量或频率的更改可能会导致输出中出现可听到的点击声。固件尝试通过在更改输出之前将通道的先前输出音量降低，然后再将其恢复来减轻这一问题。这显著改善了音频输出，但以播放声音命令中的短暂延迟为代价（最坏情况为3毫秒）。在程序中使用选项 快速音频 开启可以避免此延迟。 可听到的点击声可能会再次出现，但这由程序员自行决定。 这是一个临时选项，每当程序运行时会重置为关闭。
选项 功能键字符串 \$	✓	定义在命令提示符下按下功能键时生成的字符串。‘FNKey’ 可以是F1，以及 F5 到 F9。示例： 选项 F8 “RUN “+chr\$(34)+”myprog” +chr\$(34)+chr\$(13)+chr\$(10)。 此命令必须在命令提示符下运行（而不是在程序中）。
选项 图形用户界面 控件 控件数量	✓	指定可以定义的最大图形用户界面控件数量。 每个控件使用 52 字节，所使用的总内存必须向上舍入到下一个 2048 字节的倍数。例如，指定 70 个控件将使用 4KB 的 RAM。 默认情况下，图形用户界面控件的数量设置为零，因此此选项必须在定义任何图形用户界面控件之前使用。
选项 心跳 启用/禁用	✓	启用或禁用在 GP25 上输出心跳信号

OPTION KEYBOARD nn [,capslock] [,numlock] [repeatstart] [repeatrate]	✓	<p>Configure a PS2 keyboard. This can be used for console input and any characters typed will be available via any commands that read from the console (serial over USB).</p> <p>'nn' is a two character code defining the keyboard layout. The choices are US for the standard keyboard layout in the USA, Australia and New Zealand and UK for the United Kingdom, GR for Germany, FR for France, BR for Brazil and ES for Spain.</p> <p>The keyboard must be connected as follows:</p> <ul style="list-style-type: none"> • Connect GP8 to PS2 socket CLOCK pin. • Connect GP9 to PS2 socket DATA pin. • Connect VBUS (5V) or 3V3 (3.3V) to PS2 socket +5V. • Connect GND to PS2 socket GND. <p>Some keyboards will run on 3.3V and in that case the clock and data pins can be directly connected. However, if the keyboard is powered by 5V, level shifting must be used for these connections so that the PicoMite I/O pins are not subjected to more than 3.6V</p> <p>This command can only be run from the command line and will cause a reboot. This setting can be reset with the command OPTION KEYBOARD NO_KEYBOARD.</p> <p>The optional parameters capslock and numlock set the initial state of the keyboard (default 0, 1). The repeatstart defines how long before a character repeats the first time (valid 0-3 = 250mSec, 500mSec, 750mSec, 1S: default 1=500mSec). The repeat rate defines how fast a character repeats after the first repeat (valid 0-31 = 33mSec to 500mSec: default 12=100mSec).</p>
OPTION KEYBOARD I2C	✓	Configures support for the Solderparty bbq20 mini I2C keyboard. Note: OPTION SYSTEM I2C must be set before executing this command
OPTION LCDPANEL VIRTUAL_C or OPTION LCDPANEL VIRTUAL_M	✓	<p>Configures a virtual LCD panel without a physically connected panel.</p> <p>VIRTUAL_C = Colour, 4bit, 320 x 240</p> <p>VIRTUAL_M = Monochrome, 640 x 480</p> <p>Using this feature a program can draw graphical images on this virtual panel and then save them as a BMP file. Useful for creating a graphic image for export without an attached display</p>
OPTION LCDPANEL options or OPTION LCDPANEL DISABLE	✓	<p>Configures the PicoMite to work with an attached LCD panel.</p> <p>See the section <i>LCD Displays</i> for the details.</p> <p>This command must be run at the command prompt (not in a program).</p>
OPTION LCDPANEL CONSOLE [font [, fc [,bc]]] or OPTION LCDPANEL NOCONSOLE	✓	<p>Configures the LCD display panel for use as the console output. The LCD must support transparent text (i.e. the SSD1963_x, ILI9341 or ST7789_320 controllers).</p> <p>'font' is the default font, 'fc' is the default foreground colour, 'bc' is the default background colour. These parameters are optional and default to font 1, white, black and 100%. These settings are applied at power up.</p> <p>Note that for displays other than the SSD1963 scrolling for any console output is very slow so this feature is not recommended for general use.</p> <p>This setting is saved in flash and will be automatically applied on startup.</p> <p>To disable it use the OPTION LCDPANEL NOCONSOLE command.</p> <p>This command must be run at the command prompt.</p>

选项 键盘 nn [,大写锁定] [,数字锁] [重复开始] [重复速率]	<p>✓ 配置 PS2 键盘。这可以用于控制台输入，任何输入的字符将通过任何从控制台读取的命令（USB 串行）可用。</p> <p>‘nn是一个定义键盘布局的两个字符代码。可选项包括美国的标准键盘布局（US）、澳大利亚和新西兰（AU）、英国（UK）、德国（GR）、法国（FR）、巴西（BR）和西班牙（ES）。键盘必须按如下方式连接：</p> <ul style="list-style-type: none"> • 将GP8连接到PS2插座的时钟引脚。 • 将GP9连接到PS2插座的数据引脚。 • 将VBUSS（5V）或3V3（3.3V）连接到PS2插座的+5V。 • 将接地连接到PS2插座的接地。 <p>某些键盘可以在3.3V下运行，在这种情况下，时钟和数据引脚可以直接连接。然而，如果键盘由5V供电，则必须使用电平转换器进行这些连接，以确保PicoMite的输入/输出引脚不受超过3.6V的电压影响。</p> <p>此命令只能从命令行运行，并将导致重启。此设置可以通过命令OPTIONKEYBOARD NO_KEYBOARD重置。</p> <p>可选参数capslock和numlock设置键盘的初始状态（默认值0, 1）。repeatstart定义了字符第一次重复之前的时间（有效值0-3 = 250毫秒, 500毫秒, 750毫秒, 1秒：默认值1=500毫秒）。重复速率定义了字符在第一次重复后重复的速度（有效值0-31 = 33毫秒到500毫秒：默认值12=100毫秒）。</p>
OPTION KEYBOARD I2C	<p>✓ 配置对Solderpartybbq20mini I2C键盘的支持。注意：在执行此命令之前必须设置OPTION SYSTEM I2C</p>
OPTION LCDPANEL VIRTUAL_C 或 OPTION LCDPANEL VIRTUAL_M	<p>✓ 配置一个没有物理连接面板的虚拟LCD面板。</p> <p>VIRTUAL_C = 颜色, 4位, 320 x 240 虚拟_M = 单色, 640 x 480</p> <p>使用此功能，程序可以在此虚拟面板上绘制图形图像，然后将其保存为 BMP 文件。对于创建可导出的图形图像而无需连接显示器非常有用</p>
选项 LCD面板 或 选项 LCD面板 禁用	<p>✓ 配置 PicoMite 以与连接的液晶面板一起工作。</p> <p>有关详细信息，请参见部分液晶显示器。</p> <p>此命令必须在命令提示符下运行（而不是在程序中）。</p>
选项 LCD面板 控制台[字体 [, 前景颜色 [, 背景颜色]]] 或 选项 LCD面板 无控制台	<p>✓ 配置液晶显示面板以用作控制台输出。液晶显示器必须支持透明文本（即 SSD1963_x、ILI9341 或 ST7789_320 控制器）。</p> <p>‘字体’是默认字体，‘前景颜色’是默认前景颜色，‘背景颜色’是默认背景颜色。这些参数是可选的，默认为字体1、白色、黑色和 100%。这些设置在开机时应用。请注意，对于除SSD1963以外的显示器，任何控制台输出的滚动速度都非常慢，因此不建议在一般使用中使用此功能。此设置保存在闪存中，并将在启动时自动应用。</p> <p>要禁用它，请使用OPTION LCDPANEL NOCONSOLE命令。</p> <p>此命令必须在命令提示符下运行。</p>

OPTION LCDPANEL USER hres, vres	✓	Configures a user written display driver in MMBasic. See the file “User Display Driver.txt” in the PicoMite firmware distribution for a description of how to write the driver.
OPTION LEGACY ON or OPTION LEGACY OFF		This will turn on or off compatibility mode with the graphic commands used in the original Colour Maximite. The commands COLOUR, LINE, CIRCLE and PIXEL use the legacy syntax and all drawing commands will accept colours in the range of 0 to 7. Notes: <ul style="list-style-type: none"> Keywords such as RED, BLUE, etc are not implemented so they should be defined as constants if needed. Refer to the Colour Maximite MMBasic Language Manual for the syntax of the legacy commands. This can be downloaded from https://geoffg.net/OriginalColourMaximite.html.
OPTION LIST		This will list the settings of any options that have been changed from their default setting and are the permanent type. OPTION LIST also shows the version number and which firmware is loaded. This command must be run at the command prompt (not in a program).
OPTION MODBUFF ENABLE/DISABLE [sizeinK]	✓	Creates or removes an area of flash memory used for loading and playing .MOD files. If enabled then a mod buffer is created with a size of 128Kbytes. This can be overridden with “sizeinK”. The default is that no mod buffer is available
OPTION PICO ON/OFF	✓	When set to OFF pins GP23, GP24 and GP29 are not set up for normal Pico use and are immediately available to use. Default ON
OPTION PIN nbr		Set 'nbr' as the PIN (Personal Identification Number) for access to the console prompt. 'nbr' can be any non zero number of up to eight digits. Whenever a running program tries to exit to the command prompt for whatever reason MMBasic will request this number before the prompt is presented. This is a security feature as without access to the command prompt an intruder cannot list or change the program in memory or modify the operation of MMBasic in any way. To disable this feature enter zero for the PIN number (i.e. OPTION PIN 0). A permanent lock can be applied by using 99999999 for the PIN number. If a permanent lock is applied or the PIN number is lost the only way to recover is to reload the PicoMite firmware. This command must be run at the command prompt (not in a program).
OPTION PLATFORM name\$	✓	This command allows a user to identify a particular H/W configuration that can then be used to control a program. name\$ can be up to 31 characters long. This is a permanent option. MM.INFO\$(PLATFORM) returns this string.
OPTION POWER PFM PWM	✓	Changes operation of the 3.3V supply switch mode power supply. By default this runs in PFM mode. PWM gives better noise performance but is less power-efficient. Note that under heavy load the system will run in PWM mode regardless of this setting.
OPTION RESET	✓	Reset all saved options to their default values. This command must be run at the command prompt (not in a program).

OPTION LCDPANEL USER hres, vres	✓	配置一个用户编写的MMBasic显示驱动程序。有关如何编写驱动程序的描述，请参见PicoMite固件分发中的“用户显示驱动程序.txt”文件。
OPTION LEGACY ON 或 OPTION LEGACY OFF		<p>这将开启或关闭与原始颜色最大化中使用的图形命令的兼容模式。命令COLOUR、LINE、CIRCLE和PIXEL使用遗留语法，所有绘图命令将接受范围为0到7的颜色。注意：</p> <ul style="list-style-type: none"> 诸如RED、BLUE等关键字未实现，因此如果需要，应将其定义为常量。 请参阅颜色最大化 MMBasic 语言手册以获取遗留命令的语法。可以从https://geoffg.net/OriginalColourMaximite.html下载。
选项列表		<p>这将列出任何已更改为永久类型的选项的设置，这些选项与其默认设置不同。OPTION LIST 还显示版本号和加载的固件。</p> <p>此命令必须在命令提示符下运行（而不是在程序中）。</p>
OPTION MODBUFF ENABLE/DISABLE [sizeinK]	✓	<p>创建或删除用于加载和播放 .MOD 文件的闪存内存区域。如果启用，则创建一个大小为 128K 字节的 mod 缓冲区。可以用“sizeinK”覆盖此设置。</p> <p>默认情况下没有可用的 mod 缓冲区。</p>
OPTION PICO ON/OFF	✓	当设置为 OFF 时，引脚 GP23、GP24 和 GP29 不会被设置为正常的 Pico 使用，并且可以立即使用。默认设置为 ON。
选项 引脚 nbr		<p>将'nbr'设置为访问控制台提示符的引脚（个人识别号码）。'nbr'可以是任何非零的最多八位数字。</p> <p>每当正在运行的程序尝试以任何理由退出到命令提示符时，MMBasic将在提示符出现之前请求此数字。这是一个安全功能，因为没有访问命令提示符，入侵者无法列出或更改内存中的程序或以任何方式修改MMBasic的操作。要禁用此功能，请输入零作为引脚号码（即OPTION PIN 0）。</p> <p>可以通过使用以下方式应用永久锁定99999999 作为引脚号码。如果应用了永久锁定或引脚号码丢失，恢复的唯一方法是重新加载PicoMite固件。</p> <p>此命令必须在命令提示符下运行（而不是在程序中）。</p>
选项 平台 name\$	✓	<p>此命令允许用户识别特定的硬件配置，然后可以用来控制程序。name\$ 最多可以包含 31个字符。这是一个永久选项。</p> <p>MM.INFO\$(PLATFORM) 返回这个字符串。</p>
OPTION POWER PFM PWM	✓	<p>更改 3.3V 供电开关模式电源的操作。</p> <p>默认情况下，这在 PFM 模式下运行。PWM 提供更好的噪声性能但功率效率较低。请注意，在重负载下，系统将无论此设置如何都运行在 PWM 模式。</p>
选项 重置	✓	<p>将所有保存的选项重置为默认值。</p> <p>此命令必须在命令提示符下运行（而不是在程序中）。</p>

OPTION RTC AUTO ENABLE DISABLE	✓	Enable auto-load time\$ & date\$ from RTC on boot & every hour. If enabled and the RTC does not respond then any running program will abort with an error. At the command prompt an information message will be output. This command must be run at the command prompt (not in a program).
OPTION SDCARD CSpin [,CLKpin, MOSIpin, MISOpin] or OPTION SDCARD DISABLE	✓	Specify or disable the I/O pins to use for the SD Card interface. If the optional pins are not specified the SD Card will use the pins specified by OPTION SYSTEM SPI. Note: The pins specified by OPTION SYSTEM SPI must be a valid set of hardware SPI pins (SPI or SPI2), however, the pins specified by OPTION SDCARD can be any pins. The pins specified by OPTION SYSTEM SPI and OPTION SDCARD cannot be the same. This command must be run at the command prompt (not in a program).
OPTION SERIAL CONSOLE uartapin, uartbpin [,B] OPTION SERIAL CONSOLE DISABLE	✓	Specify that the console be accessed via a hardware serial port (instead of virtual serial over USB). 'uartapin' and 'uartbpin' can be any valid pair of rx and tx pins for either COM1 or COM2. The order that they are specified is not important. The speed defaults to 115200 baud but can be changed with OPTION BAUDRATE. Adding the "B" parameter means output will go to "B"oth the serial port and the USB. Revert to the normal the USB console. These commands must be run at the command prompt (not in a program).
OPTION SYSTEM I2C sdapin, sclpin [,SLOW/FAST]	✓	Specify the I2C port and pins for use by system devices (LCD panel, and RTC). The PicoMite uses a specific I2C port for system devices, leaving the other for the programmer. This command specifies which pins are to be used, and hence which of the I2C ports is to be used. The pins allocated to the SYSTEM I2C will not be available for other MMBasic SETPIN settings but can be used for additional I2C devices using the standard I2C command. Note: I2C(2) OPEN and I2C(2) CLOSE are not available in this case. By default the I2C port is opened at a speed of 400KHz and with a 100mSec timeout. The I2C frequency can be set using the optional third parameter which can take the values FAST = 400KHz or SLOW = 100KHzThis command must be run at the command prompt (not in a program).
OPTION SYSTEM SPI CLKpin, MOSIpin, MISOpin or OPTION SYSTEM SPI DISABLE	✓	Specify or disable the SPI port and pins for use by system devices (SD Card, LCD panel, etc). The PicoMite uses a specific hardware SPI port for system devices, leaving the other for the user. This command specifies which pins are to be used, and hence which of the SPI ports is to be used. The pins allocated to the SYSTEM SPI will not be available to other MMBasic commands. This command must be run at the command prompt (not in a program).
OPTION TAB 2 3 4 8	✓	Set the spacing for the tab key. Default is 2.

OPTION RTC AUTO ENABLE DISABLE	✓	启用从 RTC 自动加载 time\$ 和 date\$，在启动时和每小时。如果启用且 RTC 不响应，则任何正在运行的程序将以错误中止。在命令提示符下将输出一条信息消息。 此命令必须在命令提示符下运行（而不是在程序中）。
OPTION SDCARD CSpin [,CLKpin, MOSIpin, MISOpin] 或 OPTION SDCARD DISABLE	✓	指定或禁用用于 SD 卡接口的 I/O 引脚。 如果未指定可选引脚，SD卡将使用OPTION SYSTEM SPI指定的引脚。 注意：OPTION SYSTEM SPI指定的引脚必须是有效的硬件SPI引脚集（SPI或SPI2），然而，OPTION SDCARD指定的引脚可以是任何引脚。由选项系统 SPI 和选项 SD卡指定的引脚不能相同。 此命令必须在命令提示符下运行（而不是在程序中）。
OPTION SERIAL CONSOLE uartapin, uartbpin [,B] OPTION SERIAL CONSOLE 禁用	✓	指定通过硬件串行端口访问控制台（而不是通过USB的虚拟串行）。 ‘uartapin’和‘uartbpin’可以是COM1或COM2的任何有效的接收和发送引脚。指定的顺序并不重要。 速度默认为115200波特率，但可以通过OPTION BAUDRATE更改。添加“B”参数意味着输出将同时发送到串行端口和USB。 恢复到正常的USB控制台。 这些命令必须在命令提示符下运行（而不是在程序中）。
选项 系统 I2C sdapin, sclpin [,SLOW/FAST]	✓	指定供系统设备（液晶面板和RTC）使用的I2C端口和引脚。PicoMite使用特定的I2C端口供系统设备使用，留出其他端口供程序员使用。此命令指定要使用哪些引脚，因此指定了要使用的I2C端口。分配给系统I2C的引脚将无法用于其他MMBasic SETPIN设置，但可以使用标准I2C命令用于额外的I2C设备。注意：在这种情况下，I2C(2) OPEN和I2C(2) CLOSE不可用。默认情况下，I2C端口以400KHz的速度打开，并具有100毫秒的超时。I2C频率可以使用可选的第三个参数进行设置，该参数可以取值为FAST = 400KHz或SLOW = 100KHz。此命令必须在命令提示符下运行（而不是在程序中）。
选项 系统 SPI CLK引脚, MOSI引脚, MISO引脚 或 选项 系统 SPI 禁用	✓	指定或禁用供系统设备（SD卡、液晶面板等）使用的SPI端口和引脚。 PicoMite为系统设备使用特定的硬件SPI端口，将另一个留给用户。此命令指定要使用哪些引脚，因此指定使用哪个SPI端口。分配给系统SPI的引脚将无法用于其他MMBasic命令。 此命令必须在命令提示符下运行（而不是在程序中）。
选项 TAB2 3 4 8	✓	设置制表键的间距。默认值为2。

OPTION TOUCH T_CS pin, T_IRQ pin [, Beep] OPTION TOUCH DISABLE	<input checked="" type="checkbox"/>	<p>Configures MMBasic for the touch sensitive feature of an attached LCD panel.</p> <p>'T_CS pin' and 'T_IRQ pin' are the PicoMite I/O pins to be used for chip select and touch interrupt respectively (any free pins can be used).</p> <p>The remaining pins are connected to those specified using the OPTION SYSTEM SPI command.</p> <p>'Beep' is an optional pin which can be connected to a small buzzer/beeper to generate a "click" or beep sound when an advanced control is touched.</p> <p>This command must be run at the command prompt (not in a program).</p>
OPTION VCC voltage		<p>Specifies the voltage (Vcc) supplied to the PicoMite.</p> <p>When using the ADC pins to measure voltage the PicoMite uses the voltage on the pin marked VREF as its reference. This voltage can be accurately measured using a DMM and set using this command for more accurate measurement.</p> <p>The parameter is not saved and should be initialised either on the command line or in a program. The default if not set is 3.3.</p>

选项 TOUCHT_CS引脚, T_IRQ引脚[, 哔声] 选项 触摸 禁用	<p>✓ 为连接的液晶面板的触摸敏感特性配置MMBasic。</p> <p>'T_CS引脚'和'T_IRQ引脚'是PicoMite的I/O引脚，分别用于芯片选择和触摸中断（可以使用任何空闲引脚）。</p> <p>其余引脚连接到使用选项 系统 SPI命令指定的引脚。</p> <p>‘Beep’是一个可选引脚，可以连接到一个小蜂鸣器/发声器，以在触摸高级控制时产生“点击”或蜂鸣声。</p> <p>此命令必须在命令提示符下运行（而不是在程序中）。</p>
选项 VCC电压	<p>指定供给PicoMite的电压（Vcc）。在使用ADC引脚测量电压时，PicoMite使用标记为VREF的引脚上的电压作为参考。可以使用数字万用表准确测量此电压，并通过此命令设置以获得更准确的测量。</p> <p>该参数不会被保存，应在命令行或程序中初始化。如果未设置，默认值为3.3。</p>

Commands

Square brackets indicate that the parameter or characters are optional.

‘ (single quotation mark)	Starts a comment and any text following it will be ignored. Comments can be placed anywhere on a line.
file	<p>The star/asterisk command is a shortcut for RUN that may only be used at the MMBasic prompt. e.g.</p> <pre> RUN *foo RUN "foo" *"foo bar" RUN "foo bar" *foo -wombat RUN "foo", "--wombat" *foo "wom" RUN "foo", CHR\$(34) + "wom" + CHR\$(34) *foo --wom="bat" RUN "foo", "--wom=" + CHR\$(34) + "bat" + CHR\$(34)</pre> <p>String expressions are not supported/evaluated by this command; any arguments provided are passed as a literal string to the RUN command.</p>
? (question mark)	Shortcut for the PRINT command.
/* */	Start and end of multiline comments. /* and */ must be the first non-space characters at the start of a line and have a space or end-of-line after them (i.e. they are MMBasic commands). Multi-line comments cannot be used inside subroutines and functions. Any characters after */ on a line are also treated as a comment.
A: or B:	Shortcut for DRIVE “A:” and DRIVE “B:” at the command prompt
ADC OPEN freq, n_channels [,interrupt]	<p>This allocates up to 4 ADC channels for use GP26, GP27, GP28, and GP29 and sets them to be converted at the specified frequency. The maximum total frequency is 500KHz (e.g. 125KHz if all four channels are to be sampled). If the number of channels is one then it will always be GP26 used, if two then GP26 and GP27 etc. Sampling of multiple channels is sequential (there is only one ADC). The specified pins are locked to the function when ADC OPEN is active</p> <p>The optional interrupt parameter specifies an interrupt to call when the conversion completes. If not specified then conversion will be blocking</p>
ADC FREQUENCY freq	This changes the sampling frequency of the ADC conversion without having to close and re-open
ADC CLOSE	Releases the pins to normal usage
ADC START array1!() [,array2!()] [,array3!()] [,array4!()] [,Chan4arr!()] [,C1min] [,C1max] [,C2min] [,C2max] [,C3min] [,C3max] [,C4min] [,C4max]	This starts conversion into the specified arrays. The arrays must be floating point and the same size. The size of the arrays defines the number of conversions. Start can be called repeatedly once the ADC is OPEN ‘Cxmin’ and ‘Cxmax’ will scale the readings. For example, C1min=200 and C1max=100 will create values ranging from 200 to 100 for equivalent voltages of 0 - 3.3. If the scaling is not used the results are returned as a voltage between 0 and OPTION VCC (defaults to 3.3V).
ADC RUN array1%(),array2%()	Runs the ADC continuously in double buffered mode. The ADC first fills array1% and then array2% and then back to array1% etc. If more than one ADC channel is specified in the ADC OPEN command the data are interleaved. The data is returned as packed 8-bit values (Use MEMORY UNPACK to convert to a normal array). MM.INFO(ADC) will return the number of the buffer currently available for reading (1 or 2)

命令

方括号表示参数或字符是可选的。

‘ (单引号)	开始一个注释，后面的任何文本将被忽略。注释可以放置在行的任何位置。
*文件	星号/星号命令是RUN的快捷方式，仅可在MMBasic提示符下使用。例如： * 运行 *foo RUN "foo" *"foo bar" RUN "foo bar" *foo -wombat RUN "foo", "--wombat" *foo "wom" RUN "foo", CHR\$(34) + "wom" + CHR\$(34) *foo --wom="bat" RUN "foo", "--wom=" + CHR\$(34) + "bat" + CHR\$(34) 字符串表达式不被此命令支持/求值；提供的任何参数都作为字符串传递给RUN命令。
? (问号)	PRINT命令的快捷方式。
/* */	多行注释的开始和结束。/* 和 */ 必须是行首的第一个非空字符，并且后面必须有空格或行结束符（即它们是MMBasic命令）。多行注释不能在子程序和函数内部使用。行中*/后面的任何字符也被视为注释。
A: 或 B:	在命令提示符下DRIVE “A:” 和 DRIVE “B:” 的快捷方式
ADC OPEN freq, n_channels [,interrupt]	这为 GP26、GP27、GP28 和 GP29 分配最多 4 个 ADC 通道，并将它们设置为以指定的频率进行转换。最大总频率为 500KHz（例如，如果要对所有四个通道进行采样，则为 125KHz）。如果通道数量为一个，则始终使用 GP26；如果为两个，则使用 GP26 和 GP27，依此类推。多个通道的采样是顺序进行的（只有一个 ADC）。指定的引脚在 ADC OPEN 激活时被锁定为该功能。 可选的中断参数指定在转换完成时调用的中断。如果未指定，则转换将是阻塞的。
ADC 频率 freq	这可以在不关闭和重新打开的情况下更改 ADC 转换的采样频率。
ADC 关闭	释放引脚以供正常使用。
ADC 启动 array1!() [,array2!()] [,array3!()] [,array4!()] [,Chan4arr!()] [,C1min] [,C1max] [,C2min] [,C2max] [,C3min] [,C3max] [,C4min] [,C4max]	这开始将转换结果存入指定的数组中。数组必须是浮点类型且大小相同。数组的大小定义了转换的数量。一旦ADC被打开，启动可以被重复调用。 Cxmin'和'Cxmax'将缩放读取值。例如，C1min=200和C1max=100将为0 - 3.3的等效电压创建从200到100的值。如果不使用缩放，结果将作为0到OPTION VCC（默认为3.3V）之间的电压返回。
ADC 运行 array1%(),array2%()	以双缓冲模式连续运行ADC。ADC首先填充array1%，然后填充array2%，然后再返回到array1%等。如果在ADC打开命令中指定了多个ADC通道，数据将交错。数据作为打包的8位值返回（使用MEMORY UNPACK转换为普通数组）。MM.INFO(ADC)将返回当前可供读取的缓冲区编号（1或2）。

ARC x, y, r1, [r2], a1, a2 [, c]	<p>Draws an arc of a circle with a given colour and width between two radials (defined in degrees). Parameters for the ARC command are:</p> <p>x: X coordinate of centre of arc y: Y coordinate of centre of arc r1: inner radius of arc r2: outer radius of arc - can be omitted if 1 pixel wide a1: start angle of arc in degrees a2: end angle of arc in degrees c: Colour of arc (if omitted it will default to the foreground colour) Zero degrees is at the 12 o'clock position.</p>
AUTOSAVE or AUTOSAVE CRUNCH Or AUTOSAVE APPEND	<p>Enter automatic program entry mode. This command will take lines of text from the console serial input and save them to program memory.</p> <p>This mode is terminated by entering Control-Z or F1 which will then cause the received data to be transferred into program memory overwriting the previous program. Use F2 to exit and immediately run the program.</p> <p>The CRUNCH option instructs MMBasic to remove all comments, blank lines and unnecessary spaces from the program before saving. This can be used on large programs to allow them to fit into limited memory. CRUNCH can be abbreviated to the single letter C.</p> <p>The APPEND option will leave the existing program intact and append the new data from the serial input to the end of it.</p> <p>At any time this command can be aborted by Control-C which will leave program memory untouched.</p> <p>This is one way of transferring a BASIC program into the PicoMite. The program to be transferred can be pasted into a terminal emulator and this command will capture the text stream and store it into program memory. It can also be used for entering a small program directly at the console input.</p>
BACKLIGHT n [,DEFAULT]	Sets the display backlight, valid values are 0 to 100. If DEFAULT is specified then the firmware will automatically set the backlight to that level on power-up. This is particularly useful for battery operation where reducing the backlight level can significantly increase battery life
BITBANG	Replaced by the command DEVICE. For compatibility BITBANG can still be used in programs and will be automatically converted to DEVICE
BLIT	Copy one section of the display screen on LCD panel using the SSD19863, ILI9341, ILI9488 (if MISO connected), or ST7789_320 controllers controllers to or from a memory buffer.
BLIT READ [#]b, x, y, w, h	BLIT READ will copy a portion of the display to the memory buffer '#b'. The source coordinate is 'x' and 'y' and the width of the display area to copy is 'w' and the height is 'h'. When this command is used the memory buffer is automatically created and sufficient memory allocated. This buffer can be freed and the memory recovered with the BLIT CLOSE command.
BLIT WRITE [#]b, x, y [,mode]	BLIT WRITE will copy the memory buffer '#b' to the display. The destination coordinate is 'x' and 'y' and the width/height of the buffer to copy is 'w' and 'h'. If omitted w,h will default to the width and height of the blit buffer. The optional 'mode' parameter defaults to 0 and specifies how the stored image data is changed as it is written out. It is the bitwise AND of the following values: &B001 = mirrored left to right &B010 = mirrored top to bottom &B100 = don't copy transparent pixels

ARC x, y, r1, [r2], a1, a2 [, c]	<p>绘制一个给定颜色和宽度的圆弧，位于两个半径之间（以度为单位定义）。ARC命令的参数为：</p> <p>x: 圆弧中心的X坐标 y: 圆弧中心的Y坐标 r1: 圆弧的内半径 r2: 圆弧的外半径 - 如果宽度为1像素可以省略 a1: 圆弧的起始角度（以度为单位） a2: 圆弧的结束角度（以度为单位） c: 圆弧的颜色（如果省略，将默认为前景颜色）</p> <p>零度位于12点钟位置。</p>
自动保存或 自动保存压缩或 自动保存附加	<p>进入自动程序输入模式。该命令将从控制台串行输入中获取文本行并将其保存到程序内存中。</p> <p>此模式通过输入Control-Z或F1终止，这将导致接收到的数据被转移到程序内存中，覆盖之前的程序。使用 F2 退出并立即运行程序。</p> <p>CRUNCH 选项指示 MMBasic 在保存程序之前删除所有注释、空行和不必要的空格。这可以用于大型程序，以使它们适应有限的内存。CRUNCH 可以缩写为单个字母 C。</p> <p>APPEND 选项将保留现有程序不变，并将来自串行输入的新数据附加到其末尾。</p> <p>在任何时候，此命令都可以通过 Control-C 中止，这将使程序内存保持不变。</p> <p>这是一种将 BASIC 程序传输到 PicoMite 的方法。要传输的程序可以粘贴到终端仿真器中，此命令将捕获文本流并将其存储到程序内存中。它也可以用于直接在控制台输入中输入小程序。</p>
BACKLIGHT n [,DEFAULT]	设置显示背光，有效值为 0 到 100。如果指定了 DEFAULT，则固件将在上电时自动将背光设置为该级别。这对于电池供电特别有用，因为降低背光级别可以显著延长电池寿命。
位翻转	由命令DEVICE替代。为了兼容性，BITBANG仍然可以在程序中使用，并将自动转换为DEVICE
位翻转	使用SSD19863、ILI9341、ILI9488（如果MISO连接）或ST7789_320控制器从内存缓冲区复制液晶面板上的显示屏的一个部分。
位翻转 读取 [#]b, x, y, w, h	位翻转 读取将会把显示的一部分复制到内存缓冲区'#b'。源坐标为'x'和'y'，要复制的显示区域的宽度为'w'，高度为'h'。当使用此命令时，内存缓冲区会自动创建并分配足够的内存。此缓冲区可以通过BLIT CLOSE命令释放，并回收内存。
位翻转 写入 [#]b, x, y [,mode]	<p>位翻转 写入将内存缓冲区'#b'复制到显示器上。目标坐标为 'x' 和 'y'，要复制的缓冲区的宽度/高度为 'w' 和 'h'。</p> <p>如果省略，w 和 h 将默认为 blit 缓冲区的宽度和高度。</p> <p>可选的 'mode' 参数默认为 0，指定在写出时如何更改存储的图像数据。它是以下值的按位与：</p> <p style="padding-left: 40px;">&B001 = 左右镜像 &B010 = 上下镜像 &B100 = 不复制透明像素</p>

BLIT LOAD[BMP] [#]b, fname\$ [,x] [,y] [,w] [,h]	<p>BLIT LOAD will load a blit buffer from a 24-bit bmp image file. x,y define the start position in the image to start loading and w,h specify the width and height of the area to be loaded. This command will work on most display panels (not just panels using the ILI9341 controller).</p> <p>e.g.</p> <pre>BLIT LOAD #1,"image1", 50,50,100,100</pre> <p>will load an area of 100 pixels square with the top left had corner at 50,50 from the image image1.bmp</p>
BLIT CLOSE [#]b	<p>BLIT CLOSE will close the memory buffer '#b' to allow it to be used for another BLIT READ operation and recover the memory used.</p> <p>Notes:</p> <ul style="list-style-type: none"> • 32 buffers are available ranging from #1 to #32. • When specifying the buffer number the # symbol is optional. • All other arguments are in pixels.
BLIT MERGE colour, x, y, w, h	<p>Copies an area of the framebuffer defined by the 'x' and 'y' pixel coordinates of the top left and with a width of 'w' and height 'h' to the LCD display. As part of the copy it will overlay the LCD display with pixels from the layer buffer that aren't set to the 'colour' specified. The colour is specified as a number between 0 and 15 representing:</p> <p>Black, Blue, Myrtle, Cobalt, Midgreen, Cerulean, green, cyan, red, magenta, rust, fuschia, brown, lilac, yellow and white</p> <p>Requires both a framebuffer and a layer buffer to have been created to operate. Will automatically wait for frame blanking before starting the copy on ILI9341, ST7789_320 and ILI9488 displays</p>
BLIT FRAMEBUFFER from, to, xin, yin, xout, yout, width, height [,colour]	<p>Copies an area of a specific 'from' framebuffer N, F, or L to another different 'to' framebuffer N, F, or L. 'xin' and 'yin' define the top left of the area of 'width' and 'height' on the source framebuffer to be copied. 'xout' and 'yout' define the top left of the area on the target framebuffer to receive the copy. The optional parameter colour defines a pixel colour on the source which will not be copied. If omitted all pixels are copied. The colour is specified as a number between 0 and 15 representing:</p> <p>Black, Blue, Myrtle, Cobalt, Midgreen, Cerulean, green, cyan, red, magenta, rust, fuschia, brown, lilac, yellow and white</p>
BLIT MEMORY address, x, y [,col]	<p>Copies an area of memory treated as a packed array of colour nibbles to the current graphical output as specified by FRAMEBUFFER WRITE. The colour is specified as a number between 0 and 15 representing:</p> <p>Black, Blue, Myrtle, Cobalt, Midgreen, Cerulean, green, cyan, red, magenta, rust, fuschia, brown, lilac, yellow and white</p> <p>The first word of the area of memory starting at 'address%' must contain the width and height of the area to be copied as 16-bit integers with the width as the bottom 16 bits. The address must be aligned to a word boundary (divisible by 4).</p> <p>If the optional parameter 'col' is specified then that specific colour is not copied.</p> <p>If the top bit of either the width or height is set to 1 then the colour data is treated as compressed (the remaining 15 bits are used as the width and/or height). The compression algorithm is simple, each byte contains a count in the bottom nibble (1-15) and a colour in the top nibble (0-15). In the event that more than 15 pixels are the same colour additional bytes are used for that colour</p>

BLIT LOAD[BMP] [#]b, fname\$ [,x] [,y] [,w] [,h]	<p>BLIT LOAD 将从 24 位 bmp 图像文件加载 blit 缓冲区。x,y 定义图像中开始加载的起始位置，w,h 指定要加载区域的宽度和高度。该命令适用于大多数显示面板（不仅限于使用 ILI9341 控制器的面板）。</p>
BLIT CLOSE [#]b	<p>例如。</p> <pre>BLIT LOAD #1,"image1", 50,50,100,100</pre> <p>将从图像 image1.bmp 中加载一个 100 像素的正方形区域，左上角坐标为 50,50。</p>
BLIT MERGE 颜色, x, y, w, h	<p>BLIT CLOSE 将关闭内存缓冲区 '#b'，以便可以用于另一个 BLIT READ 操作，并回收所使用的内存。</p> <p>注意：</p> <ul style="list-style-type: none"> 可用的缓冲区有 32 个，范围从 #1 到 #32。 指定缓冲区编号时，# 符号是可选的。 所有其他参数均以像素为单位。
BLIT FRAMEBUFFER from, to, xin, yin, xout, yout, width, height [,colour]	<p>将由左上角的 ‘x’ 和 ‘y’ 像素坐标定义的帧缓冲区区域，宽度为 ‘w’ 和高度为 ‘h’，复制到液晶显示器上。作为复制的一部分，它将用来自层缓冲区的未设置为指定‘颜色’的像素覆盖液晶显示。颜色被指定为一个介于0到15之间的数字，代表：</p> <p>黑色、蓝色、紫罗兰色、钴蓝色、中绿色、天蓝色、绿色、青色、红色、品红色、锈色、紫红色、棕色、淡紫色、黄色和白色 需要同时创建帧缓冲区和层缓冲区才能操作。 在ILI9341、ST7789_320和ILI9488显示器上，将自动等待帧消隐后开始复制。</p>
BLIT MEMORY 地址, x, y [,col]	<p>将特定的‘从’帧缓冲区N、F或L的区域复制到另一个不同的‘到’帧缓冲区N、F或L。‘xin’和‘yin’定义了要复制的源帧缓冲区上‘宽度’和‘高度’区域的左上角。‘xout’和‘yout’定义了目标帧缓冲区上接收复制的区域的左上角。可选参数颜色定义了源中的像素颜色，该颜色将不会被复制。如果省略，则所有像素都会被复制。颜色以0到15之间的数字表示，代表：</p> <p>黑色、蓝色、紫罗兰色、钴蓝色、中绿色、天蓝色、绿色、青色、红色、品红色、锈色、紫红色、棕色、淡紫色、黄色和白色</p>
	<p>将内存区域视为压缩的颜色半字节数组，复制到当前图形输出中，具体由帧缓冲区写入指定。颜色以0到15之间的数字表示，代表：</p> <p>黑色、蓝色、紫罗兰色、钴蓝色、中绿色、天蓝色、绿色、青色、红色、品红色、锈色、紫红色、棕色、淡紫色、黄色和白色</p> <p>内存区域的第一个字（word）从‘地址%’开始，必须包含要复制区域的宽度和高度，作为16位整数，其中宽度为底部16位。地址必须对齐到字边界（可被4整除）。</p> <p>如果指定了可选参数‘col’，则该特定颜色将不会被复制。</p> <p>如果宽度或高度的最高位被设置为1，则颜色数据被视为压缩（剩余的15位用于宽度和/或高度）。压缩算法很简单，每个字节的低四位包含一个计数（1-15），高四位包含一个颜色（0-15）。如果有超过15个像素是相同颜色，则会使用额外的字节来表示该颜色。</p>

BLIT COMPRESSED address%, x, y [,col]	Acts the same as BLIT MEMORY but assumes the data is compressed and ignores the top bit in the width and height
BLIT x1, y1, x2, y2, w, h	Copy one section of the display screen to another part of the display. The source coordinate is 'x1' and 'y1'. The destination coordinate is 'x2' and 'y2'. The width of the screen area to copy is 'w' and the height is 'h'. All arguments are in pixels. The display on a PicoMite must use the SSD19863, ILI9341_8, ILI9341, ILI9488 (if MISO connected), or ST7789_320 controllers.
BOX x, y, w, h [,lw] [,c] [,fill]	Draws a box on the attached LCD panel with the top left hand corner at 'x' and 'y' with a width of 'w' pixels and a height of 'h' pixels. 'lw' is the width of the sides of the box and can be zero. It defaults to 1. 'c' specifies the colour and defaults to the default foreground colour if not specified. 'fill' is the fill colour. It can be omitted or set to -1 in which case the box will not be filled. All parameters can be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. 'x', 'y', 'w', and 'h' must all be arrays or all be single variables /constants otherwise an error will be generated. 'lw', 'c', and fill can be either arrays or single variables/constants. See the section <i>Graphics Commands and Functions</i> for a definition of the colours and graphics coordinates.
CALL usersubname\$ [,usersubparameters,...]	This is an efficient way of programmatically calling user defined subroutines (see also the CALL() function). In many case it can allow you to get rid of complex SELECT and IF THEN ELSEIF ENDIF clauses and is processed in a much more efficient way. The “usersubname\$” can be any string or variable or function that resolves to the name of a normal user subroutine (not an in-built command). The “usersubparameters” are the same parameters that would be used to call the subroutine directly. A typical use could be writing any sort of emulator where one of a large number of subroutines should be called depending on some variable. It also allows a way of passing a subroutine name to another subroutine or function as a variable.
CAT S\$, N\$	Concatenates the strings by appending N\$ to S\$. This is functionally the same as S\$ = S\$ + N\$ but operates faster.
CHDIR dir\$	Change the current working directory on the default drive to ‘dir\$’ The special entry “.” represents the parent of the current directory and “..” represents the current directory. “/” is the root directory.
CIRCLE x, y, r [,lw] [,a] [,c] [, fill]	Draw a circle on the video output centred at 'x' and 'y' with a radius of 'r' on the attached LCD panel. 'lw' is optional and is the line width (defaults to 1). 'c' is the optional colour and defaults to the current foreground colour if not specified. The optional 'a' is a floating point number which will define the aspect ratio. If the aspect is not specified the default is 1.0 which gives a standard circle. 'fill' is the fill colour. It can be omitted or set to -1 in which case the box will not be filled. All parameters can be expressed as arrays and the software will plot the number of circles as determined by the dimensions of the smallest array. 'x', 'y' and 'r' must all be arrays or all be single variables/constants otherwise an error will be generated. 'lw', 'a', 'c', and fill can be either arrays or single variables/constants. See the section <i>Graphics Commands and Functions</i> for a definition of the colours and graphics coordinates.

BLIT COMPRESSED address%, x, y [,col]	与BLIT MEMORY的作用相同，但假设数据是压缩的，并忽略宽度和高度的最高位。
BLIT x1, y1, x2, y2, w, h	将显示屏的一部分复制到显示屏的另一部分。 源坐标为'x1'和'y1'。目标坐标为'x2'和'y2'。要复制的屏幕区域的宽度为'w'，高度为'h'。 所有参数均以像素为单位。PicoMite上的显示器必须使用SSD19863、ILI9341_8、ILI9341、ILI9488（如果连接了MISO），或ST7789_320控制器。
BOX x, y, w, h [, lw] [,c] [,fill]	在连接的液晶面板上绘制一个框，左上角位于'x'和'y'，宽度为'w'像素，高度为'h'像素。 'lw'是框的边宽，可以为零。默认为1。 'c'指定颜色，如果未指定，则默认为默认前景颜色。'fill'是填充颜色。可以省略或设置为-1，在这种情况下，框将不会被填充。 所有参数可以表示为数组，软件将根据最小数组的尺寸绘制框的数量。'x'、'y'、'w'和'h'必须全部为数组或全部为单个变量/常量，否则将生成错误。'lw'、'c'和fill可以是数组或单个变量/常量。 请参见章节图形命令和函数以获取颜色和图形坐标的定义。
CALL usersubname\$ [,usersubparameters,...]	这是一种有效的以编程方式调用用户定义子程序的方法（另见CALL()函数）。在许多情况下，它可以让您摆脱复杂的SELECT和IF THEN ELSEIF ENDIF语句，并以更高效的方式处理。 “usersubname\$”可以是任何字符串、变量或解析为普通用户子程序名称（而不是内置命令）的函数。“usersubparameters”是调用子程序时使用的相同参数。一个典型的用法可能是编写任何类型的仿真器，其中应根据某个变量调用大量子程序中的一个。它还允许将子程序名称作为变量传递给另一个子程序或函数。
CAT S\$, N\$	通过将N\$附加到S\$来连接字符串。这在功能上与S\$ = S\$ + N\$相同，但运行速度更快。
CHDIR dir\$	将默认驱动器上的当前工作目录更改为'dir\$'。 特殊条目“..”表示当前目录的父目录，而“.”表示当前目录。"/"是根目录。
CIRCLE x, y, r [,lw] [, a] [,c] [, fill]	在附加的液晶面板上绘制一个以'x'和'y'为中心，半径为'r'的圆。 'lw'是可选的，表示线宽（默认为1）。 'c'是可选的颜色，如果未指定，则默认为当前前景颜色。可选的'a'是一个浮点数，用于定义宽高比。如果未指定宽高比，默认值为1.0，这将生成一个标准圆。 'fill'是填充颜色。可以省略或设置为-1，在这种情况下，框将不会被填充。 所有参数都可以表示为数组，软件将根据最小数组的尺寸绘制圆的数量。 'x'、'y'和'r'必须都是数组，或者都是单个变量/常量，否则将生成错误。 'lw'、'a'、'c'和fill可以是数组或单个变量/常量。 请参见章节图形命令和函数以获取颜色和图形坐标的定义。

CLEAR	Delete all variables and recover the memory used by them. See ERASE for deleting specific array variables.
CLOSE [#]fnbr [,[#]fnbr] ...	Close the file(s) previously opened with the file number '#fnbr'. The # is optional. Also see the OPEN command.
CLS [colour]	Clears the LCD panel's screen. Optionally 'colour' can be specified which will be used for the background when clearing the screen.
COLOUR fore [, back] or COLOR fore [, back]	Sets the default colour for commands (PRINT, etc) that display on the attached LCD panel. 'fore' is the foreground colour, 'back' is the background colour. The background is optional and if not specified will default to black.
CONST id = expression , id = expression] ... etc	Create a constant identifier which cannot be changed once created. 'id' is the identifier which follows the same rules as for variables. The identifier can have a type suffix (!, %, or \$) but it is not required. If it is specified it must match the type of 'expression'. 'expression' is the value of the identifier and it can be a normal expression (including user defined functions) which will be evaluated when the constant is created. A constant defined outside a sub or function is global and can be seen throughout the program. A constant defined inside a sub or function is local to that routine and will hide a global constant with the same name.
CONTINUE	Resume running a program that has been stopped by an END statement, an error, or CTRL-C. The program will restart with the next statement following the previous stopping point. Note that it is not always possible to resume the program correctly – this particularly applies to complex programs with graphics, nested loops and/or nested subroutines and functions.
CONTINUE DO or CONTINUE FOR	Skip to the end of a DO/LOOP or a FOR/NEXT loop. The loop condition will then be tested and if still valid the loop will continue with the next iteration.
COPY fname1\$ TO fname2\$	Copy a file from 'fname1\$' to 'fname2\$'. Both are strings. A directory path can be used in both 'fname\$' and 'fname\$'. If the paths differ the file specified in 'fname\$' will be copied to the path specified in 'fname2\$' with the file name as specified. The filenames can include the drive specification in the case that you are copying to and or from the non-active drive (see the DRIVE command)
COPY fname\$ TO dirname\$	Wildcard copy. the bulk copy is triggered if fname\$ contains a '*' or a '?' character. dirname\$ must be a valid directory name and should NOT end in a slash character
CPU RESTART	Will force a restart of the processor. This will clear all variables and reset everything (e.g. timers, COM ports, I2C, etc) similar to a power up situation but without the power up banner. If OPTION AUTORUN has been set the program in the specified flash location or program memory will restart.
CPU SLEEP n	Will cause the processor to sleep for 'n' seconds. Note that the CPU does not have a true low power sleep so the power saving is limited.

清除	删除所有变量并恢复它们使用的内存。 有关删除特定数组变量，请参见 ERASE。
CLOSE [#]fnbr [,[#]fnbr] ...	关闭之前使用文件编号 '#fnbr' 打开的文件。# 是可选的。另请参见 OPEN 命令。
CLS [颜色]	清除液晶面板的屏幕。可以选择指定 '颜色'，在清除屏幕时将用于背景。
COLOUR 前 [, 后] 或 COLOR 前 [, 后]	设置在附加的液晶面板上显示的命令（如 PRINT 等）的默认颜色。'fore' 是前景颜色，'back' 是背景颜色。背景是可选的，如果未指定，将默认为黑色。
CONST id = 表达式 [,id = 表达式] ... 等等	创建一个常量标识符，一旦创建后无法更改。 'id' 是标识符，其规则与变量相同。标识符可以有类型后缀 (!、% 或 \$)，但不是必需的。如果指定，则必须与 '表达式' 的类型匹配。'表达式' 是标识符的值，可以是一个普通表达式（包括用户定义的函数），在常量创建时将被求值。 在子程序或函数外定义的常量是全局的，可以在整个程序中看到。在子程序或函数内定义的常量是该例程的局部，将隐藏同名的全局常量。
继续	恢复运行一个因结束语句、错误或CTRL-C而停止的程序。程序将从上一个停止点后的下一条语句重新开始。 请注意，并不总是可以正确恢复程序——这尤其适用于具有图形、嵌套循环和/或嵌套子程序和函数的复杂程序。
继续 DO 或 继续 FOR	跳到 DO/LOOP 或 FOR/NEXT 循环的末尾。然后将测试循环条件，如果仍然有效，循环将继续进行下一次迭代。
复制 fname1\$ 到 fname2\$	将文件从 'fname1\$' 复制到 'fname2\$'。两者都是字符串。 在 'fname' 和 'fname2' 中都可以使用目录路径。如果路径不同，指定在 'fname' 中的文件将被复制到 'fname2' 中指定的路径，文件名按指定的方式进行。文件名可以包含驱动器规范，以便在复制到或从非活动驱动器时使用（参见 DRIVE 命令）
复制 fname\$ 到 dirname\$	通配符复制。如果 fname\$ 包含 '*' 或 '?' 字符，则触发批量复制。dirname\$ 必须是有效的目录名称，并且不应以斜杠字符结尾
中央处理器重启	将强制重启处理器。 这将清除所有变量并重置所有内容（例如定时器、COM 端口、I2C 等），类似于电源启动情况，但没有电源启动横幅。如果设置了自动运行选项，则指定闪存位置或程序内存中的程序将重新启动。
CPU SLEEP n	将使处理器睡眠'n'秒。 请注意，CPU 没有真正的低功耗睡眠，因此节能有限。

<pre>CSUB name [type [, type] ...] hex [[hex[...] hex [[hex[... END CSUB</pre>	<p>Defines the binary code for an embedded machine code program module written in C or ARM assembler. The module will appear in MMBasic as the command 'name' and can be used in the same manner as a built-in command. Multiple embedded routines can be used in a program with each defining a different module with a different 'name'.</p> <p>The first 'hex' word is a 32 bit word which is the offset in bytes from the start of the CSUB to the entry point of the embedded routine (usually the function main()). The following hex words are the compiled binary code for the module. These are automatically programmed into MMBasic when the program is saved. Each 'hex' must be exactly eight hex digits representing the bits in a 32-bit word and be separated by one or more spaces or new lines. The command must be terminated by a matching END CSUB. Any errors in the data format will be reported when the program is run.</p> <p>During execution MMBasic will skip over any CSUB commands so they can be placed anywhere in the program.</p> <p>The type of each parameter can be specified in the definition. For example:</p> <pre>CSUB MySub integer, integer, string</pre> <p>This specifies that there will be three parameters, the first two being integers and the third a string.</p> <p>Note:</p> <ul style="list-style-type: none"> Up to ten arguments can be specified ('arg1', 'arg2', etc). If a variable or array is specified as an argument the C routine will receive a pointer to the memory allocated to the variable or array and the C routine can change this memory to return a value to the caller. In the case of arrays, they should be passed with empty brackets e.g. arg(). In the CSUB the argument will be supplied as a pointer to the first element of the array. Constants and expressions will be passed to the embedded C routine as pointers to a temporary memory space holding the value.
<pre>CTRLVAL(#ref) =</pre>	<p>This command will set the value of an advanced control.</p> <p>#ref' is the control's reference number.</p> <p>For off/on controls like check boxes it will override any touch input and can be used to depress/release switches, tick/untick check boxes, etc. A value of zero is off or unchecked and non-zero will turn the control on. For a LED it will cause the LED to be illuminated or turned off. It can also be used to set the initial value of spin boxes, text boxes, etc.</p> <p>For example:</p> <pre>CTRLVAL(#10) = 12.4</pre> <p>All controls expect to be assigned a number (float or integer) except Frame, Caption, Display Box, Text Box and Format Box which expect a string.</p>
<pre>DATA constant[,constant]..</pre>	<p>Stores numerical and string constants to be accessed by READ.</p> <p>In general string constants should be surrounded by double quotes (""). An exception is when the string consists of just alphanumeric characters that do not represent MMBasic keywords (such as THEN, WHILE, etc). In that case quotes are not needed.</p> <p>Numerical constants can also be expressions such as 5 * 60.</p>
<pre>DATE\$ = "DD-MM-YY[YY]" or DATE\$ = "DD/MM/YY[YY]" or DATE\$ = "YYYY-MM-DD" or DATE\$ = "YYYY/MM/DD"</pre>	<p>Set the date of the internal clock/calendar.</p> <p>DD, MM and YY are numbers, for example: DATE\$ = "28-7-2014"</p> <p>With OPTION RTC AUTO ENABLE the picomite starts with the DATE\$ programmed in RTC.</p> <p>Without OPTION RTC AUTO ENABLE the picomite starts with the date set to "01-01-2000" on power up.</p>

<pre>CSUB 名称 [类型 [, 类型] ...] 十六进制 [[十六进制[...] 十六进制 [[十六进制[...] 结束 CSUB</pre>	<p>定义了一个用C或ARM汇编语言编写的嵌入式机器代码程序模块的二进制代码。该模块将在MMBasic中显示为命令'名称'，并可以像内置命令一样使用。</p> <p>可以在一个程序中使用多个嵌入式例程，每个例程定义一个不同的模块，具有不同的'名称'。</p> <p>第一个'十六进制'字是一个32位字，它是从CSUB开始到嵌入式例程入口点（通常是函数main()）的字节偏移量。后续的十六进制字是该模块的编译二进制代码。这些在程序保存时会自动编程到MMBasic中。每个'十六进制'必须恰好是八个十六进制数字，表示32位字中的位数，并且必须用一个或多个空格或换行符分隔。命令必须以匹配的END CSUB结束。数据格式中的任何错误将在程序运行时报告。</p> <p>在执行过程中，MMBasic 将跳过任何 CSUB 命令，因此它们可以放置在程序的任何位置。</p> <p>每个参数的类型可以在定义中指定。例如：</p> <pre>CSUB MySub integer, integer, string</pre> <p>这指定将有三个参数，前两个为整数，第三个为字符串。</p> <p>注意：</p> <ul style="list-style-type: none"> 最多可以指定十个参数 ('arg1', 'arg2', 等等)。 如果指定变量或数组作为参数，C 例程将接收指向分配给变量或数组的内存的指针，C 例程可以更改此内存以将值返回给调用者。在数组的情况下，应使用空括号传递，例如 arg()。在CSUB 中，参数将作为指向数组第一个元素的指针提供。 常量和表达式将作为指向临时内存空间的指针传递，该空间保存值。
<pre>CTRLVAL(#ref) =</pre>	<p>此命令将设置高级控件的值。</p> <p>'#ref' 是控件的参考编号。</p> <p>对于像复选框这样的开关控件，它将覆盖任何触摸输入，并可用于按下/释放开关、勾选/取消勾选复选框等。值为零表示关闭或未勾选，非零值将打开控件。对于LED，它将使LED亮起或关闭。它还可以用于设置旋转框、文本框等的初始值。例如：</p> <pre>CTRLVAL (#10) = 12.4</pre> <p>所有控件都期望被分配一个数字（浮点数或整数），除了帧、标题、显示框、文本框和格式框，它们期望一个字符串。</p>
<pre>DATA 常量[,常量]..</pre>	<p>存储数值和字符串常量，以便通过READ访问。一般来说，字符串常量应被双引号 ("") 包围。一个例外是当字符串仅由不表示MMBasic关键字（如THEN、WHILE等）的字母数字字符组成时。在这种情况下不需要引号。</p> <p>数值常量也可以是表达式，例如 5 * 60。</p>
<pre>DATE\$ = "DD-MM-YY[YY]" 或 DATE\$ = "DD/MM/YY[YY]" 或 DATE\$ = "YYYY-MM-DD" 或 DATE\$ = "YYYY/MM/DD"</pre>	<p>设置内部时钟/日历的日期。</p> <p>DD、MM 和 YY 是数字，例如：DATE\$ = "28-7-2014" 使用选项 OPTION RTC AUTO ENABLE，PicoMite 将以 RTC 中编程的 DATE\$ 启动。</p> <p>如果没有选项 OPTION RTC AUTO ENABLE，PicoMite 将在上电时以日期 "01-01-2000" 启动。</p>

<pre>DEFINEFONT #Nbr hex [[hex[...] hex [[hex[...] END DEFINEFONT</pre>	<p>This will define an embedded font which can be used alongside or to replace the built in font(s) used on an attached LCD panel. These work exactly same as the built in fonts (i.e. selected using the FONT command or specified in the TEXT command).</p> <p>See the <i>Embedded Fonts</i> folder in the PicoMite distribution zip file for a selection of embedded fonts and a full description of how to create them.</p> <p>'#Nbr' is the font's reference number (from 1 to 16). It can be the same number as a built in font and in that case it will replace the built in font.</p> <p>Each 'hex' must be exactly eight hex digits and be separated by spaces or new lines from the next.</p> <ul style="list-style-type: none"> • Multiple lines of 'hex' words can be used with the command terminated by a matching END DEFINEFONT. • Multiple embedded fonts can be used in a program with each defining a different font with a different font number. • During execution MMBasic will skip over any DEFINEFONT commands so they can be placed anywhere in the program. • Any errors in the data format will be reported when the program is saved.
<pre>DEVICE BITSTREAM pinno, n_transitions, array%()</pre>	<p>This command is used to generate an extremely accurate bit sequence on the pin specified. The pin must have previously been set up as an output and set to the required starting level.</p> <p>Notes:</p> <ul style="list-style-type: none"> • The array contains the length of each level in the bitstream in microseconds. The maximum period allowed is 65.5 mSec • The first transition will occur immediately on executing the command. • The last period in the array is ignored other than defining the time before control returns to the program or command line. • The pin is left in the starting state if the number of transitions is even and the opposite state if the number of transitions is odd.
<pre>DEVICE CAMERA OPEN XLKpin, PLKpin, HSpin, VSCpin, RETpin, D0pin</pre>	<p>Command supporting the OV7670 camera module. This command initialises the camera, It outputs a 12MHz clock on XLK (PWM) and checks that it is correctly receiving signals on PLK, VS, and HS. The camera is set to a resolution of 160x120 (QQVGA) which is the maximum achievable within the limits of the RP2040 memory.</p> <p>Enable OPTION SYSTEM I2C on the PicoMite and wire SCL and SDA to the relevant pins (may be labelled SIOC and SIOD on the camera module). These connections must have a pullup to 3.3V - 2K7 recommended)</p> <p>Other pins are wired as per the OPEN command. (NB: VS may be labelled VSYNC, HS may be labelled HREF, PLK may be labelled PCLK, RET may be labelled RESET and XLK may be XCLK on your module)</p> <p>D0pin defines the start of a range of 8 contiguous pins (e.g.GP0 - GP7).</p>
<pre>DEVICE CAMERA CAPTURE [scale, [x , y]]</pre>	<p>This captures a picture from the camera (RGB565) and displays it on an LCD screen. An SPI LCD must be connected and enabled in order for the command to work. (ILI9341 and ST7789_320 recommended).</p> <p>Scale defaults to 1 and x,y each to 0</p> <p>By default a 160x120 image is output on the LCD with the top left at 0,0 on the LCD. Setting scale to 2 will fill a 320x240 display with the image. Setting the x and y parameters will offset the top left of the image on the LCD.</p> <p>Update rate in a continuous loop is 7FPS onto the display at 1:1 scale and 5FPS scaled to 320x240.</p> <p>Of course, assuming the display has MISO wired it is then possible to save the image to disk using the SAVE IMAGE command as used to create the example image above.</p>

<pre>DEFINEFONT #Nbr hex [[hex[...] hex [[hex[...] END DEFINEFONT</pre>	<p>这将定义一个嵌入式字体，可以与附加的液晶面板上使用的内置字体一起使用或替代。这些与内置字体的工作方式完全相同（即通过 FONT 命令选择或在 TEXT 命令中指定）。</p> <p>请查看PicoMite分发压缩文件中的嵌入式字体文件夹，以获取嵌入式字体的选择以及如何创建它们的完整说明。</p> <p>'#Nbr'是字体的参考编号（从1到16）。它可以与内置字体的编号相同，在这种情况下，它将替换内置字体。</p> <p>每个'hex'必须恰好是八个十六进制数字，并且必须与下一个数字用空格或换行符分隔。</p> <ul style="list-style-type: none"> • 可以使用多行'hex'字词，命令以匹配的ENDDEFIN.FONT结束。 • 在程序中可以使用多个嵌入式字体，每个字体定义一个不同的字体编号。 • 在执行过程中，MMBasic将跳过任何DEFIN.FONT命令，因此它们可以放置在程序的任何位置。 • 数据格式中的任何错误将在程序保存时报告。
设备位流引脚编号， 转换次数, 数组%()	<p>此命令用于在指定引脚上生成极其准确的位序列。引脚必须先被设置为输出并设定为所需的起始电平。</p> <p>注意：</p> <ul style="list-style-type: none"> • 数组包含位流中每个电平的长度，以微秒为单位。允许的最大周期为65.5毫秒。 • 第一次转换将在执行命令时立即发生。 • 数组中的最后一个周期被忽略，除了定义控制返回到程序或命令行之前的时间。 • 如果转换次数是偶数，引脚将保持在起始状态；如果转换次数是奇数，则处于相反状态。
设备 摄像头打开 XLK引脚, PLK引脚, HSpin, VSC引脚, RET引脚, D0引脚	<p>支持OV7670摄像头模块的命令。此命令初始化摄像头，它在XLK (PW) 上输出12MHz的时钟，并检查是否正确接收PLK、VS和HS上的信号。摄像头设置为160x120 (QQVGA) 的分辨率，这是在RP2040内存限制内可以达到的最大值。</p> <p>在PicoMite上启用选项 系统 I2C，并将SCL和SDA连接到相关引脚（在相机模块上可能标记为SIOC和SIOD）。这些连接必须有一个上拉到3.3V（推荐2K7）。其他引脚按打开命令连接。（注意：V_S可能标记为VSYNC，HS可能标记为HREF，PLK可能标记为PCLK，RET可能标记为RESET，XLK可能标记为XCLK在您的模块上）D0引脚定义了一系列8个连续引脚的开始（例如，GP0-GP7）。</p>
设备 摄像头 捕获 [缩放, [x , y]]	<p>这将从相机捕获一张图片 (RGB565) 并在液晶显示器上显示。必须连接并启用SPI液晶显示器，以便命令能够工作。（推荐使用ILI9341和ST7789_320）。</p> <p>缩放默认为1，x和y各自默认为0。</p> <p>默认情况下，LCD上输出的图像为160x120，左上角位于LCD的0,0位置。将缩放设置为2将填充320x240的显示器。设置x和y参数将偏移图像在LCD上的左上角。在1:1缩放下，更新率为7FPS，在缩放到320x240时为5FPS。</p> <p>当然，假设显示器已连接MISO，则可以使用SAVE IMAGE命令将图像保存到磁盘，如上面创建示例图像时所用。</p>

DEVICE CAMERA CLOSE	Closes the camera subsystem and frees up all the pins allocated in the OPEN command
DEVICE CAMERA CHANGE image%(),change! [,scale [x ,y]]	<p>The camera firmware is also able to detect motion in the camera's field of view using the command. It does this by operating the camera in YUV mode rather than RGB. This has the advantage that the intensity information and colour information are separated and just one byte is needed for a 256-level greyscale image which is ideal for detecting movement.</p> <p>image% is an array of size 160x120 bytes (DIM image%(160,120/8-1)</p> <p>On calling the command it holds a packed 8-bit greyscale image.</p> <p>The change! variable returns the percentage the image has changed from the previous time the command was called.</p> <p> Optionally if "scale" is set then the image delta is output to the screen i.e. the difference between the previous image and this one. As in the CAPTURE command the delta image can be scaled and positioned as required. If the scale parameter is omitted then the LCD is not updated by this command.</p>
DEVICE CAMERA TEST tnum	The command enables or disables a test signal from the camera. tnum=2 generates colourbars and tnum=0 sets back to the visual input. tnum = 1 and tnum = 3 do something but what?
DEVICE CAMERA REGISTER reg%, data%	Sets the register "reg%" in the camera to the value "data%". When used the command will report to the console the previous value and automatically confirms that the new value has been set as requested. The colour rendition of the camera as initialised is reasonable but could probably be improved further by tuning the various camera registers.
DEVICE HUMID pin, tvar, hvar [,DHT11]	<p>Returns the temperature and humidity using the DHT22 sensor. Alternative versions of the DHT22 are the AM2303 or the RHT03 (all are compatible). 'pin' is the I/O pin connected to the sensor. Any I/O pin may be used.</p> <p>'tvar' is the variable that will hold the measured temperature and 'hvar' is the same for humidity. Both must be present and both must be floating point variables.</p> <p>For example: HUMID 3, TEMP!, HUMIDITY!</p> <p>Temperature is measured in °C and the humidity is percent relative humidity. Both will be measured with a resolution of 0.1. If an error occurs (sensor not connected or corrupt signal) both values will be 1000.0.</p> <p>Normally the DHT22 should be powered by 3.3V to keep its output below 3.6V for the PicoMite and the signal pin should be pulled up by a 1K to 10K resistor (4.7K recommended) to 3.3V.</p> <p>The optional DHT11 parameter modifies the timings to work with the DHT11. Set to 1 for DHT11 and 0 or omit for DHT22.</p>
DEVICE SERIALTX pinno, baudrate, ostring\$	Outputs ostring\$ as a serial datastream on pinno. Baudrate can be between 110 and 230400 (230400 may need CPU to be overclocked). NB command is blocking during transmission.
DEVICE SERIALRX pinno, baudrate, istring\$, timeout_in_ms, status% [,nbr] [,terminators\$]	<p>Inputs serial data on pinno. Baudrate can be between 110 and 230400 (230400 may need CPU to be overclocked). status% returns:</p> <ul style="list-style-type: none"> -1 = timeout (NB: use len(istring\$) to see number received) 2 = number of characters requested satisfied 3 = terminating character satisfied <p>Nbr specifies the number of characters to be received before the command returns. Terminators\$ specifies one or more single characters that can be used to terminate reception. NB command is blocking during reception.</p>

设备 摄像头 关闭	关闭摄像头子系统并释放OPEN命令中分配的所有引脚
设备 摄像头 更改 image%(),change! [,scale [x ,y]]	<p>摄像头固件还能够使用该命令检测摄像头视野中的运动。它通过以YUV模式而不是RGB模式操作相机来实现。这有一个优点，即强度信息和颜色信息是分开的，并且只需要一个字节来表示256级灰度图像，这对于检测运动是理想的。</p> <p>image%是一个大小为160x120字节的数组 (DIM image%(160,120/8-1) 在调用该命令时，它保存一个打包的8位灰度图像。 change!变量返回图像与上次调用该命令时的变化百分比。</p> <p>可选地，如果设置了"scale"，则图像增量将输出到屏幕上，即前一幅图像与当前图像之间的差异。与CAPTURE命令一样，增量图像可以根据需要进行缩放和定位。如果省略scale参数，则该命令不会更新LCD。</p>
设备 摄像头 测试 tnum	该命令启用或禁用来自相机的测试信号。tnum=2生成颜色条，tnum=0则恢复为视觉输入。tnum = 1 和 tnum = 3 做某事，但是什么呢？
设备 摄像头 注册 reg%, data%	将相机中的寄存器 "reg%" 设置为值 "data%"。使用该命令时，将向控制台报告先前的值，并自动确认新值已按请求设置。相机初始化后的颜色还原是合理的，但通过调整各种相机寄存器可能会进一步改善。
设备湿度引脚, tvar, hvar [,DHT11]	<p>使用 DHT22 传感器返回温度和湿度。DHT22 的替代版本是 AM2303 或 RHT03（均兼容）。</p> <p>'引脚' 是连接到传感器的输入/输出引脚。可以使用任何输入/输出引脚。</p> <p>'tvar' 是将保存测量温度的变量，'hvar' 是湿度的相同变量。两者必须存在，并且两者必须是浮点变量。</p> <p>例如：HUMID 3, TEMP!, HUMIDITY! 温度以°C为单位测量，湿度以相对湿度百分比表示。 两者的分辨率均为0.1。如果发生错误（传感器未连接或信号损坏），两个值将为1000.0。 通常，DHT22应由3.3V供电，以保持其输出低于3.6V 对于PicoMite，信号引脚应通过1K到10K的 电阻（推荐使用4.7K）上拉至3.3V。 可选的DHT11参数修改时序以适应DHT11。 设置为1表示DHT11，设置为0或省略表示DHT22。</p>
DEVICE SERIALTX pinno, baudrate, ostring\$	在pinno上将ostring\$作为串行数据流输出。波特率可以在110到230400之间（230400可能需要超频CPU）。NB命令在传输期间是阻塞的。
设备 SERIALRX引脚编号 , 波特率, 字符串\$, 超时 (毫秒) , 状态% [,编号] [,终止符\$]	<p>在引脚编号上输入串行数据。波特率可以在110到230400之间（230400可能需要超频CPU）。状态%返回：</p> <ul style="list-style-type: none"> -1 = 超时（注意：使用len(istring\$)查看接收到的数字） 2 = 满足请求的字符数量 3 = 满足终止字符 <p>编号指定在命令返回之前要接收的字符数量。终止符\$指定一个或多个可以用于终止接收的单个字符。NB命令在接收期间是阻塞的。</p>

DEVICE WS2812 type, pin, nbr, value%[()]	<p>This command outputs the required signals to drive one or more WS2812 LED chips connected to 'pin'. Note that the pin must be set to a digital output before this command is used.</p> <p>'type' is a single character specifying the type of chip being driven:</p> <ul style="list-style-type: none"> O = original WS2812 B = WS2812B S = SK6812 W =SK6812W (RGBW) <p>'nbr' is the number of LEDs in the chain (1 to 256). The 'value%()' array should be an integer array sized to have exactly the same number of elements as the number of LEDs to be driven.</p> <p>For the first three variants each element in the array should contain the colour in the normal RGB888 format (i.e. 0 to &HFFFFFF).</p> <p>For type W use a RGBW value (0-&HFFFFFFF).</p> <p>If only one LED is connected then a single integer should be used for value% (ie, not an array).</p>
DEVICE WII OPEN [,interrupt]	<p>Opens a Wii Classic controller and implements background polling of the device. The Wii Classic must be wired to the pins specified by OPTION SYSTEM I2C which is a prerequisite. Open attempts to talk to the Wii Classic and will return an error if not found. If found the firmware will sample the Wii data in the background at a rate of 50Hz. If an optional user interrupt is specified this will be triggered if any of the buttons changes (both on and off) See the DEVICE function for how to read data from the Wii Classic.</p>
DEVICE WII CLOSE	<p>CLOSE will stop the background polling and disable any interrupt specified</p>
<p>DEVICE LCD INIT d4, d5, d6, d7, rs, en or DEVICE LCD line, pos, text\$ or DEVICE LCD CLEAR or DEVICE LCD CLOSE</p>	<p>Display text on an LCD character display module. This command will work with most 1-line, 2-line or 4-line LCD modules that use the KS0066, HD44780 or SPLC780 controller (however this is not guaranteed).</p> <p>The LCD INIT command is used to initialise the LCD module for use. 'd4' to 'd7' are the I/O pins that connect to inputs D4 to D7 on the LCD module (inputs D0 to D3 should be connected to ground). 'rs' is the pin connected to the register select input on the module (sometimes called CMD). 'en' is the pin connected to the enable or chip select input on the module. The R/W input on the module should always be grounded. The above I/O pins are automatically set to outputs by this command.</p> <p>When the module has been initialised data can be written to it using the LCD command. 'line' is the line on the display (1 to 4) and 'pos' is the character location on the line (the first location is 1). 'text\$' is a string containing the text to write to the LCD display.</p> <p>'pos' can also be C8, C16, C20 or C40 in which case the line will be cleared and the text centred on a 8 or 16, 20 or 40 line display. For example:</p> <pre>LCD 1, C16, "Hello"</pre> <p>LCD CLEAR will erase all data displayed on the LCD and LCD CLOSE will terminate the LCD function and return all I/O pins to the not configured state. See the section <i>Special Hardware Devices</i> for more details.</p>
<p>DEVICE LCD CMD d1 [, d2 [, etc]] or DEVICE LCD DATA d1 [, d2 [, etc]]</p>	<p>These commands will send one or more bytes to an LCD display as either a command (LCD CMD) or as data (LCD DATA). Each byte is a number between 0 and 255 and must be separated by commas. The LCD must have been previously initialised using the LCD INIT command (see above).</p> <p>These commands can be used to drive a non standard LCD in "raw mode" or they can be used to enable specialised features such as scrolling, cursors and custom character sets. You will need to refer to the data sheet for your LCD to find the necessary command and data values.</p>

设备 WS2812type, 引脚, 编号, 值%[()]	<p>此命令输出所需的信号以驱动连接到'引脚'的一个或多个WS2812 LED芯片。请注意，在使用此命令之前，必须将引脚设置为数字输出。</p> <p>'type' 是一个单字符，指定所驱动芯片的类型：</p> <ul style="list-style-type: none"> O = 原始 WS2812 B = WS2812B S = SK6812 W =SK6812W (RGBW) <p>'nbr' 是链中 LED 的数量（1 到 256）。'value%()' 数组应该是一个整数数组，其大小恰好与要驱动的 LED 数量相同。</p> <p>对于前三种变体，数组中的每个元素应包含颜色以正常的 RGB888 格式表示（即 0 到 &HFFFFFFF）。</p> <p>对于类型 W，使用 RGBW 值 (0-&HFFFFFFF).</p> <p>如果只连接一个 LED，则应使用单个整数作为 value%（即，不是数组）。</p>
设备 WII 打开 [, 中断]	<p>打开一个 WII Classic 控制器并实现对设备的后台轮询。Wii Classic 必须按照 OPTIONSYSTEM I2C 指定的引脚进行连接，这是一个前提条件。打开尝试与Wii Classic进行通信如果未找到，将返回错误。如果找到，固件将以50Hz的速率在后台采样Wii数据。如果指定了可选的用户中断，则在任何按钮状态变化（开和关）时将触发此中断。</p> <p>有关如何从Wii Classic读取数据，请参见DEVICE函数。</p>
设备 WII 关闭	<p>CLOSE将停止后台轮询并禁用任何指定的中断。</p>
<p>DEVICE LCD INIT d4, d5, d6, d7, rs, en 或 DEVICE LCD line, pos, text\$ 或 DEVICE LCD CLEAR 或 DEVICE LCD CLOSE</p>	<p>在LCD字符显示模块上显示文本。此命令适用于大多数使用KS0066、HD44780或SPLC780控制器的1行、2行或4行LCD模块（但不保证）。</p> <p>LCD INIT命令用于初始化LCD模块以供使用。'd4'到'd7'是连接到LCD模块输入D4到D7的I/O引脚（输入D0到D3应连接到地）。'rs'是连接到模块上寄存器选择输入的引脚（有时称为 CMD）。'en'是连接到模块上启用或芯片选择输入的引脚。模块上的 R/W 输入应始终接地。上述 I/O 引脚通过此命令自动设置为输出。</p> <p>当模块初始化后，可以使用 LCD 命令向其写入数据。'line' 是显示器上的行（1 到 4），'pos' 是该行上的字符位置（第一个位置为 1）。'text' 是一个包含要写入液晶显示器的文本的字符串。</p> <p>'pos' 也可以是 C8、C16、C20 或 C40，在这种情况下，该行将被清除，文本将居中显示在 8 或 16、20 或 40 行的显示器上。例如：</p> <p style="text-align: center;">LCD 1, C16, "你好"</p> <p>LCD CLEAR 将擦除液晶显示器上显示的所有数据，而 LCD CLOSE 将终止液晶显示器功能并将所有 I/O 引脚返回到未配置状态。</p> <p>请参阅章节特殊硬件设备以获取更多详细信息。</p>
<p>DEVICE LCD CMD d1 [, d2 [, 等等]] 或 DEVICE LCD DATA d1 [, d2 [, 等等]]</p>	<p>这些命令将作为命令（LCD CMD）或数据（LCD DATA）向液晶显示器发送一个或多个字节。每个字节是一个介于 0 和 255 之间的数字，必须用逗号分隔。液晶显示器必须先使用 LCD INIT 命令进行初始化（见上文）。</p> <p>这些命令可以用于驱动非标准液晶显示器的“原始模式”，或者可以用于启用特殊功能，如滚动、光标和自定义字符集。您需要参考液晶显示器的数据手册以找到必要的命令和数据值。</p>

<p>DIM [type] decl [,decl]..</p> <p>where 'decl' is:</p> <p>var [length] [type] [init]</p> <p>'var' is a variable name with optional dimensions</p> <p>'length' is used to set the maximum size of the string to 'n' as in LENGTH n</p> <p>'type' is one of FLOAT or INTEGER or STRING (the type can be prefixed by the keyword AS - as in AS FLOAT)</p> <p>'init' is the value to initialise the variable and consists of: = <expression></p> <p>For a simple variable one expression is used, for an array a list of comma separated expressions surrounded by brackets is used.</p> <p> Examples:</p> <p>DIM nbr(50)</p> <p>DIM INTEGER nbr(50)</p> <p>DIM name AS STRING</p> <p>DIM a, b\$, nbr(100), strn\$(20)</p> <p>DIM a(5,5,5), b(1000)</p> <p>DIM strn\$(200) LENGTH 20</p> <p>DIM STRING strn(200) LENGTH 20</p> <p>DIM a = 1234, b = 345</p> <p>DIM STRING strn = "text"</p> <p>DIM x%(3) = (11, 22, 33, 44)</p>	<p>Declares one or more variables (i.e. makes the variable name and its characteristics known to the interpreter).</p> <p>When OPTION EXPLICIT is used (as recommended) the DIM, LOCAL or STATIC commands are the only way that a variable can be created. If this option is not used then using the DIM command is optional and if not used the variable will be created automatically when first referenced.</p> <p>The type of the variable (i.e. string, float or integer) can be specified in one of three ways:</p> <p>By using a type suffix (i.e. !, % or \$ for float, integer or string). For example:</p> <pre>DIM nbr%, amount!, name\$</pre> <p>By using one of the keywords FLOAT, INTEGER or STRING immediately after the command DIM and before the variable(s) are listed. The specified type then applies to all variables listed (i.e. it does not have to be repeated). For example:</p> <pre>DIM STRING first_name, last_name, city</pre> <p>By using the Microsoft convention of using the keyword "AS" and the type keyword (i.e. FLOAT, INTEGER or STRING) after each variable. If you use this method the type must be specified for each variable and can be changed from variable to variable.</p> <p>For example:</p> <pre>DIM amount AS FLOAT, name AS STRING</pre> <p>Floating point or integer variables will be set to zero when created and strings will be set to an empty string (i.e. ""). You can initialise the value of the variable with something different by using an equals symbol (=) and an expression following the variable definition. For example:</p> <pre>DIM STRING city = "Perth", house = "Brick"</pre> <p>The initialising value can be an expression (including other variables) and will be evaluated when the DIM command is executed. See the section <i>Defining and Using Variables</i> for more examples of the syntax.</p> <p>As well as declaring simple variables the DIM command will also declare arrayed variables (i.e. an indexed variable with a number of dimensions). Following the variable's name the dimensions are specified by a list of numbers separated by commas and enclosed in brackets. For example:</p> <pre>DIM array(10, 20)</pre> <p>Each number specifies the number of elements in each dimension. Normally the numbering of each dimension starts at 0 but the OPTION BASE command can be used to change this to 1.</p> <p>The above example specifies a two dimensional array with 11 elements (0 to 10) in the first dimension and 21 (0 to 20) in the second dimension. The total number of elements is 231 and because each floating point number on the PicoMite requires 8 bytes a total of 1848 bytes of memory will be allocated.</p> <p>Strings will default to allocating 255 bytes (i.e. characters) of memory for each element and this can quickly use up memory when defining arrays of strings. In that case the LENGTH keyword can be used to specify the amount of memory to be allocated to each element and therefore the maximum length of the string that can be stored. This allocation ('n') can be from 1 to 255 characters.</p> <p>For example: DIM STRING s(5, 10) will declare a string array with 66 elements consuming 16,896 bytes of memory while:</p> <pre>DIM STRING s(5, 10) LENGTH 20</pre> <p>Will only consume 1,386 bytes of memory. Note that the amount of memory allocated for each element is n + 1 as the extra byte is used to track the actual length of the string stored in each element.</p> <p>If a string longer than 'n' is assigned to an element of the array an error will be</p>
--	--

<p>DIM [type] decl [,decl].. 'decl' 是： var [长度] [类型] [初始化] 'var' 是一个变量名称，具有可选的尺寸 '长度' 用于设置字符串的最大大小为 'n'，如 LENGTH n '类型' 是 FLOAT、INTEGER 或 STRING 之一（类型可以通过关键字 AS 前缀 - 如 AS FLOAT） '初始化' 是用于初始化变量的值，包含： = <表达式> 对于简单变量，使用一个表达式，对于数组，使用用逗号分隔的表达式列表，并用括号括起来。</p> <p>示例：</p> <pre> DIM nbr(50) DIM INTEGER nbr(50) DIM name AS STRING DIM a, b\$, nbr(100), strn\$(20) DIM a(5,5,5), b(1000) DIM strn\$(200) LENGTH 20 DIM STRING strn(200) LENGTH 20 DIM a = 1234, b = 345 DIM STRING strn = "文本" DIM x%(3) = (11, 22, 33, 44) </pre>	<p>声明一个或多个变量（即使变量名称及其特性为解释器所知）。</p> <p>当使用选项 显式（如推荐的那样）时，DIM、局部或静态命令是创建变量的唯一方式。如果不使用此选项，则使用DIM命令是可选的，如果不使用，变量将在首次引用时自动创建。</p> <p>变量的类型（即字符串、浮点数或整数）可以通过三种方式之一指定：</p> <p>通过使用类型后缀（即！、%或\$表示浮点数、整数或字符串）。例如：</p> <pre>DIM nbr%, amount!, name\$</pre> <p>通过在命令DIM后立即使用关键字FLOAT、INTEGER或STRING，并在列出变量之前。指定的类型适用于所有列出的变量（即不必重复）。</p> <p>例如：</p> <pre>DIM STRING first_name, last_name, city</pre> <p>通过使用微软的约定，在每个变量后使用关键字"AS"和类型关键字（即FLOAT、INTEGER或STRING）。如果您使用这种方法，则必须为每个变量指定类型，并且可以在变量之间更改。</p> <p>例如：</p> <pre>DIM amount AS FLOAT, name AS STRING</pre> <p>浮点或整数变量在创建时将被设置为零，字符串将被设置为空字符串（即""）。您可以通过在变量定义后使用等号（=）和表达式来初始化变量的值。例如：</p> <pre>DIM STRING city = "Perth", house = "Brick"</pre> <p>初始化值可以是一个表达式（包括其他变量），并将在执行DIM命令时进行求值。有关语法的更多示例，请参见章节定义和使用变量。</p> <p>除了声明简单变量外，DIM命令还将声明数组变量（即具有多个维度的索引变量）。</p> <p>在变量名称后，尺寸由用逗号分隔并用括号括起来的数字列表指定。例如：</p> <pre>DIM array(10, 20)</pre> <p>每个数字指定每个维度中的元素数量。通常，每个维度的编号从0开始，但可以使用OPTION BASE命令将其更改为1。</p> <p>上述示例指定了一个二维数组，第一维有11个元素（0到10），第二维有21个元素（0到20）。元素的总数为231，由于PicoMite上的每个浮点数需要8个字节，因此将分配1848个字节的内存。</p> <p>字符串默认为每个元素分配255个字节（即字符）的内存，当定义字符串数组时，这可能会迅速耗尽内存。</p> <p>在这种情况下，可以使用LENGTH关键字指定分配给每个元素的内存量，从而确定可以存储的字符串的最大长度。此分配（'n'）可以是1到255个字符。</p> <p>例如：DIM STRING s(5, 10) 将声明一个包含66个元素的字符串数组，消耗16,896字节的内存，而：</p> <pre>DIM STRING s(5, 10) LENGTH 20</pre> <p>将只消耗1,386字节的内存。请注意，为每个元素分配的内存量为n + 1，因为额外的字节用于跟踪存储在每个元素中的字符串的实际长度。</p> <p>如果将一个长度超过'n'的字符串分配给数组的一个元素，将会产生一个错误。</p>
--	--

	<p>produced. Other than this, string arrays created with the LENGTH keyword act exactly the same as other string arrays. This keyword can also be used with non-array string variables but it will not save any memory.</p> <p>In the above example you can also use the Microsoft syntax of specifying the type <u>after</u> the length qualifier. For example:</p> <pre>DIM s(5, 10) LENGTH 20 AS STRING</pre> <p>Arrays can also be initialised when they are declared by adding an equals symbol (=) followed by a bracketed list of values at the end of the declaration. For example:</p> <pre>DIM INTEGER nbr(4) = (22, 44, 55, 66, 88)</pre> <p>or <pre>DIM s\$(3) = ("foo", "boo", "doo", "zoo")</pre></p> <p>Note that the number of initialising values must match the number of elements in the array including the base value set by OPTION BASE. If a multi dimensioned array is initialised then the first dimension will be initialised first followed by the second, etc.</p> <p>Also note that the initialising values must be after the LENGTH qualifier (if used) and after the type declaration (if used).</p>
DO <statements> LOOP	This structure will loop forever; the EXIT DO command can be used to terminate the loop or control must be explicitly transferred outside of the loop by commands like GOTO or EXIT SUB (if in a subroutine).
DO WHILE expression <statements> LOOP	Loops while ‘expression’ is true (this is equivalent to the older WHILE-WEND loop). If, at the start, the expression is false the statements in the loop will not be executed, not even once.
DO <statements> LOOP UNTIL expression	Loops until the expression following UNTIL is true. Because the test is made at the end of the loop the statements inside the loop will be executed at least once, even if the expression is true.
DRAW3D	The 3D engine includes commands for manipulating 3D images including setting the camera, creating, hiding, rotating, etc. See the document “The CMM2 3D engine” in the PicoMite firmware download for a full description.
DRIVE drive\$	Sets the active disk drive as ‘drive\$’. ‘drive\$’ can be “A:” or “B:” where A is the flash drive and B is the SD Card if configured
EDIT or EDIT fname\$	Invoke the full screen editor. If a filename is specified the editor will load the file from the current disk (A: or B:) to allow editing and on exit with F1 or F2 save it to the disk. If the file does not exist it is created on exit. The current program stored in flash memory is not affected. If editing an existing file a backup with .bak appended to the filename is also created on exit. If fname\$ includes an extension other than .bas then colour coding will be temporarily turned off during the edit. If no extension is specified the firmware will assume .bas Editing a file from disk allows non-Basic files such as html or sprite files to be edited without corruption during the tokenising process that happens when stored to flash. See the section <i>Full Screen Editor</i> for details of how to use the editor.
ELSE	Introduces an optional default condition in a multiline IF statement. See the multiline IF statement for more details.
ELSEIF expression THEN or ELSE IF expression THEN	Introduces an optional secondary condition in a multiline IF statement. See the multiline IF statement for more details.
END	End the running program and return to the command prompt.

	<p>将会产生。除此之外，使用 LENGTH 关键字创建的字符串数组与其他字符串数组的行为完全相同。此关键字也可以与非数组字符串变量一起使用，但不会节省任何内存。</p> <p>在上述示例中，您还可以使用微软的语法，在长度限定符后指定类型。例如：</p> <pre>DIM s(5, 10) LENGTH 20 AS STRING</pre> <p>数组在声明时也可以通过在声明末尾添加等号 (=) 后跟括号中的值列表来初始化。</p> <p>例如：</p> <pre>DIM INTEGER nbr(4) = (22, 44, 55, 66, 88)</pre> <p>或 DIM s\$(3) = ("foo", "boo", "doo", "zoo")</p> <p>请注意，初始化值的数量必须与数组中元素的数量相匹配，包括由 OPTION BASE 设置的基值。如果初始化了多维数组，则将首先初始化第一维，然后是第二维，依此类推。</p> <p>还要注意，初始化值必须在 LENGTH 限定符（如果使用）和类型声明（如果使用）之后。</p>
DO <语句> LOOP	该结构将无限循环；可以使用 EXIT DO 命令终止循环，或者必须通过像 GOTO 或 EXIT SUB（如果在子程序中）这样的命令显式地将控制转移到循环外。
DO WHILE 表达式 <语句> LOOP	当‘表达式’为真时循环（这相当于旧版的 WHILE-WEND 循环）。如果在开始时表达式为假，则循环中的语句将不会执行，甚至一次也不会。
DO <语句> LOOP UNTIL 表达式	循环直到 UNTIL 后面的表达式为真。由于测试在循环的末尾进行，因此循环内部的语句至少会执行一次，即使表达式为真。
DRAW3D	3D 引擎包括用于操作 3D 图像的命令，包括设置相机、创建、隐藏、旋转等。 有关完整描述，请参见 PicoMite 固件下载中的文档“CMM2 3D 引擎”。
DRIVE drive\$	将活动磁盘驱动器设置为‘drive\$’。‘drive\$’ 可以是“A.”或“B.”，其中 A 是闪存驱动器，B 是已配置的 SD 卡。
编辑 或 编辑 fname\$	调用全屏编辑器。 如果指定了文件名，编辑器将从当前磁盘 (A: 中加载该文件或 B:) 以允许编辑，并在退出时使用 F1 或 F2 将其保存到磁盘。如果文件不存在，则在退出时会创建该文件。当前存储在闪存中的程序不受影响。如果编辑现有文件，退出时还会创建一个以 .bak 结尾的备份文件。 如果 fname\$ 包含除 .bas 以外的扩展名，则在编辑期间颜色编码将暂时关闭。 如果未指定扩展名，固件将假定为 .bas 从磁盘编辑文件允许编辑非 Basic 文件，例如 html 或精灵文件，而不会在存储到闪存时发生令牌化过程中的损坏。 有关如何使用编辑器的详细信息，请参见部分全屏编辑器。
否则	在多行 IF 语句中引入可选的默认条件。 有关更多详细信息，请参见多行 IF 语句。
否则如果 expression THEN 或 否则如果 expression THEN	在多行 IF 语句中引入一个可选的次要条件。 有关更多详细信息，请参见多行 IF 语句。
结束	结束正在运行的程序并返回到命令提示符。

END CSUB	Marks the end of a C subroutine. See the CSUB command. Each CSUB must have one and only one matching END CSUB statement.
END FUNCTION	Marks the end of a user defined function. See the FUNCTION command. Each function must have one and only one matching END FUNCTION statement. Use EXIT FUNCTION if you need to return from a function from within its body.
ENDIF or END IF	Terminates a multiline IF statement. See the multiline IF statement for more details.
END SUB	Marks the end of a user defined subroutine. See the SUB command. Each sub must have one and only one matching END SUB statement. Use EXIT SUB if you need to return from a subroutine from within its body.
ERASE variable [,variable]..	Deletes variables and frees up the memory allocated to them. This will work with arrayed variables and normal (non array) variables. Arrays can be specified using empty brackets (e.g. dat()) or just by specifying the variable's name (e.g. dat). Use CLEAR to delete all variables at the same time (including arrays).
ERROR [error_msg\$]	Forces an error and terminates the program. This is normally used in debugging or to trap events that should not occur.
EXECUTE command\$	This executes the Basic command "command\$". Use should be limited to basic commands that execute sequentially for example the GOTO statement will not work properly Things that are tested and work OK include GOSUB, Subroutine calls, other simple statements (like PRINT and simple assignments) Multiple statements separated by : are not allowed and will error The command sets an internal watchdog before executing the requested command and if control does not return to the command, like in a GOTO statement, the timer will expire. In this case you will get the message "Command timeout". RUN is a special case and will cancel the timer allowing you to use the command to chain programs if required.
EXIT DO EXIT FOR EXIT FUNCTION EXIT SUB	EXIT DO provides an early exit from a DO..LOOP EXIT FOR provides an early exit from a FOR..NEXT loop. EXIT FUNCTION provides an early exit from a defined function. EXIT SUB provides an early exit from a defined subroutine. The old standard of EXIT on its own (exit a do loop) is also supported.
FILES [fspec\$] [,sort]	Lists files in any directories on the default Flash Filesystem or SD Card. 'fspec\$' (if specified) can contain a path and search wildcards in the filename. Question marks (?) will match any character and an asterisk (*) will match any number of characters. If omitted, all files will be listed. For example: * Find all entries .TXT Find all entries with an extension of TXT E*. Find all entries starting with E X?X.* Find all three letter file names starting and ending with X mydir/* Find all entries in directory mydir NB: putting wildcards in the pathname will result in an error 'sort' specifies the sort order as follows: size by ascending size time by descending time/date name by file name (default if not specified) type by file extension

结束子程序	标记 C 子程序的结束。请参见 CSUB 命令。 每个 CSUB 必须有且仅有一个匹配的 END CSUB 语句。
结束函数	标记用户定义函数的结束。请参见 FUNCTION 命令。 每个函数必须有且仅有一个匹配的 END FUNCTION 语句。如果需要从函数体内返回，请使用 EXIT FUNCTION。
结束如果 或 结束 IF	终止多行 IF 语句。 有关更多详细信息，请参见多行 IF 语句。
结束子程序	标记用户定义子程序的结束。请参见 SUB 命令。 每个子程序必须有且仅有一个匹配的 END SUB 语句。如果需要从子程序的主体中返回，请使用 EXIT SUB。
擦除变量 [, 变量]..	删除变量并释放分配给它们的内存。这适用于数组变量和普通（非数组）变量。数组可以使用空括号（例如 array()）或仅通过指定变量名称（例如 array）来指定。 使用 CLEAR 可以同时删除所有变量（包括数组）。
错误[错误信息\$]	强制产生错误并终止程序。这通常用于调试或捕获不应发生的事件。
执行命令 \$	这将执行 Basic 命令 "命令\$"。使用应限于顺序执行的基本命令，例如 GOTO 语句将无法正常工作。 经过测试并正常工作的内容包括 GOSUB、子程序调用、其他简单语句（如 PRINT 和简单赋值）。不允许使用冒号分隔的多个语句，这将导致错误。该命令在执行请求的命令之前设置一个内部看门狗，如果控制未返回到命令，例如在 GOTO 语句中，定时器将过期。在这种情况下，您将收到消息“命令超时”。 RUN 是一个特殊案例，将取消定时器，允许您在需要时使用该命令链接程序。
EXIT DO EXIT FOR EXIT FUNCTION EXIT SUB	EXIT DO 提供了从 DO..LOOP 的提前退出。 EXIT FOR 提供了从 FOR..NEXT 循环的提前退出。 EXIT FUNCTION 提供了从定义的函数的提前退出。 EXIT SUB 提供了从定义的子程序的提前退出。 单独使用的旧标准 EXIT（退出 DO 循环）也受到支持。
FILES [fspec\$] [,sort]	列出默认 Flash 文件系统或 SD 卡上任何目录中的文件。 'fspec'（如果指定）可以包含路径并在文件名中搜索通配符。 问号 (?) 将匹配任何字符，星号 (*) 将匹配任意数量的字符。如果省略，将列出所有文件。 例如： * 查找所有条目 .TXT 查找所有扩展名为 TXT 的条目 E??.* 查找所有以 E 开头的条目 X?X.* 查找所有以 X 开头和结尾的三字母文件名 mydir/* 查找目录 mydir 中的所有条目 注意：在路径名中放置通配符将导致错误 'sort' 指定排序顺序如下： 按大小升序排列 按时间/日期降序排列 按文件名排列（如果未指定则为默认） 按文件扩展名排列

FLASH	Manages the storage of programs in the flash memory. Up to three programs can be stored in the flash memory and retrieved as required. Note that these saved programs will be erased with a firmware upgrade. One of these flash memory locations can be automatically loaded and run when power is applied using the OPTION AUTORUN n command. In the following 'n' is a number 1 to 3.
FLASH LIST	Displays a list of all flash locations including the first line of the program.
FLASH LIST n [,all]	List the program saved to slot n. Use ALL to list without page breaks.
FLASH ERASE n	Erase a flash program location.
FLASH ERASE ALL	Erase all flash program locations.
FLASH SAVE n	Save the current program to the flash location specified.
FLASH LOAD n	Load a program from the specified flash location into program memory.
FLASH RUN n	Runs the program in flash location n, clear all variables. Does not change the program memory.
FLASH CHAIN n	Runs the program in flash location n, leaving all variables intact (allows for a program that is much bigger than the program memory). Does not change the program memory.
FLASH OVERWRITE n	Erase a flash program location and then save the current program to the flash location specified.
FLASH DISK LOAD n, fname\$ [,O[OVERWRITE]]	Loads the contents of file fname\$ into flash slot n as a binary image. If the optional parameter OVERWRITE (or O) is specified the content of the flash slot will be overwritten without an error being raised.
FLUSH [#]fnbr	Causes any buffered writes to a file previously opened with the file number '#fnbr' to be written to disk. The # is optional. Using this command ensures that no data is lost if there is a power cut after a write command.
FONT [#]font-number, scaling	This will set the default font for displaying text on an LCD panel. Fonts are specified as a number. For example, #2 (the # is optional). See the section <i>Graphics Commands and Functions</i> for details of the available fonts. 'scaling' can range from 1 to 15 and will multiply the size of the pixels making the displayed character correspondingly wider and higher. E.g. a scale of 2 will double the height and width.
FOR counter = start TO finish [STEP increment]	Initiates a FOR-NEXT loop with the 'counter' initially set to 'start' and incrementing in 'increment' steps (default is 1) until 'counter' is greater than 'finish'. The 'increment' can be an integer or a floating point number. Note that using a floating point fractional number for 'increment' can accumulate rounding errors in 'counter' which could cause the loop to terminate early or late. 'increment' can be negative in which case 'finish' should be less than 'start' and the loop will count downwards. See also the NEXT command.
FRAMEBUFFER	FRAMEBUFFER command for colour SPI displays. This command can be used to avoid screen artefacts when updating SPI displays with moving elements.
FRAMEBUFFER CREATE	Creates a framebuffer "F" with a RGB121 colour space and resolution to match the configured SPI colour display

闪存	管理程序在闪存中的存储。最多可以在闪存中存储三个程序，并根据需要检索。请注意，这些保存的程序将在固件升级时被擦除。
	可以使用OPTION AUTORUN n命令在通电时自动加载并运行这些闪存位置之一。在以下内容中，‘n’是一个数字，范围为1到3。
FLASH LIST	显示所有闪存位置的列表，包括程序的第一行。
闪存 列表 n [,all]	列出保存到槽n的程序。使用ALL可以无分页地列出。
闪存 擦除 n	擦除一个闪存程序位置。
闪存擦除全部	擦除所有闪存程序位置。
闪存 保存 n	将当前程序保存到指定的闪存位置。
闪存 加载 n	从指定的闪存位置加载程序到程序内存中。
闪存 运行 n	运行闪存位置n中的程序，清除所有变量。不会改变程序内存。
闪存 链接 n	运行闪存位置n中的程序，保留所有变量不变（允许程序大于程序内存）。不会改变程序内存。
闪存 覆盖 n	擦除一个闪存程序位置，然后将当前程序保存到指定的闪存位置。
闪存 磁盘加载 n, fname\$ [,O[OVERWRITE]]	将文件fname\$的内容作为二进制映像加载到闪存槽n中。如果指定了可选参数OVERWRITE（或O），则闪存槽的内容将被覆盖，而不会引发错误。
FLUSH [#]fnbr	导致任何先前以文件编号‘#fnbr’打开的文件的缓冲写入被写入磁盘。#是可选的。使用此命令可确保在写入命令后如果发生断电不会丢失数据。
字体 [#]字体编号，缩放	这将设置在液晶面板上显示文本的默认字体。 字体以数字形式指定。例如，#2（#是可选的）。请参见部分图形命令和函数以获取可用字体的详细信息。 ‘缩放’可以在1到15之间变化，将乘以像素的大小，使显示的字符相应地变得更宽和更高。例如，缩放为2将使高度和宽度加倍。
FOR 计数器 = 起始 到 结束 [STEP 增量]	启动一个FOR-NEXT循环，‘计数器’初始设置为‘起始’，并以‘增量’步骤（默认是1）递增，直到‘计数器’大于‘结束’。 ‘增量’可以是一个整数或浮点数。请注意，使用浮点数作为‘增量’可能会在‘计数器’中累积舍入误差，这可能导致循环提前或延迟终止。 ‘增量’可以为负数，在这种情况下，‘结束’应小于‘开始’，循环将向下计数。 另见NEXT命令。
帧缓冲区	用于颜色 SPI 显示器的帧缓冲区命令。此命令可用于在更新带有移动元素的 SPI 显示器时避免屏幕伪影。
帧缓冲区创建	创建一个帧缓冲区“F”，其 RGB121 颜色空间和分辨率与配置的 SPI 颜色显示器相匹配

FRAMEBUFFER LAYER	Creates a framebuffer "L" with a RGB121 colour space and resolution to match the configured SPI colour display
FRAMEBUFFER WRITE where/where	Specifies the target for subsequent graphics commands. "where" can be N, F, or L where N is the actual display or a string variable can be used
FRAMEBUFFER CLOSE [which]	Closes a framebuffer and releases the memory. The optional parameter "which" can be F or L. If omitted closes both.
FRAMEBUFFER COPY from, to [,b]	Does a highly optimised full screen copy of one framebuffer to another. "from" and "to" can be N, F, or L where N is the physical display. You can only copy from N on displays that support BLIT and transparent text. The firmware will automatically compress or expand the RGB resolution when copying to and from unmatched framebuffers. Of course copying from RGB565 to RGB121 loses information but for many applications (e.g. games) 16 colour levels is more than enough. When copying to an LCD display the optional parameter "b" can be used (FRAMEBUFFER COPY F/L, N, B). This instructs the firmware to action the copy using the second processor in the PicoMite and control returns immediately to the Basic program
FRAMEBUFFER WAIT	Pauses processing until the LCD display enters frame blanking. Implemented for ILI9341, ST7789_320 and ILI9488 displays. Used to reduce artefacts when writing to the screen
FRAMEBUFFER MERGE [colour] [,mode] [,updaterate]	Copies the contents of the Layer buffer and Framebuffer onto the LCD display omitting all pixels of a particular colour. Preconditions for the command are that FRAMEBUFFER and LAYERBUFFER are both created FRAMEBUFFER MERGE - writes the contents of the framebuffer to the physical display overwriting any pixels in the framebuffer that are set in the layerbuffer (not zero) FRAMEBUFER MERGE col - writes the contents of the framebuffer to the physical display overwriting any pixels in the framebuffer that are in the layerbuffer not set to the transparent colour "col". The colour is specified as a number between 0 and 15 representing: 0:BLACK,1:BLUE,2:MYRTLE,3:COBALT,4:MIDGREEN,5:CERULEAN,6:GREEN,7:CYAN,8:RED,9:MAGENTA,10:RUST,11:FUCHSIA,12:BROWN,13:LILAC,14:YELLOW,15:WHITE FRAMEBUFFER MERGE col,B - as above except that the transfer to the physical display takes place on the second processor and control returns to Basic immediately FRAMEBUFFER MERGE col,R [,updaterate] - sets the second processor to continuously update the physical display with the merger of the two buffers. Automatically sets FRAMEBUFFER WRITE F if not F or L already set. By default the screen will update as fast as possible (At 133MHz an ILI9341 in SPI mode updates about 13 times a second, in 8-bit parallel mode the ILI9341 achieves 27 FPS) If "updaterate" is set then the screen will update to the rate specified in milliseconds (unless that is less than the fastest achievable on the display) NB: FRAMEBUFFER WRITE cannot be set to N while continuous merged update is active. FRAMEBUFFER MERGE col,A - aborts the continuous updates In addition deleting either the layerbuf or framebuffer, ctrl-C, or END will abort the automatic update as well.
FRAMEBUFFER SYNC	Waits for the latest update on the second processor to complete to allow drawing without tearing

帧缓冲区层	创建一个帧缓冲区“L”，其 RGB121 颜色空间和分辨率与配置的 SPI 颜色显示器相匹配
帧缓冲区写入 位置/位置	指定后续图形命令的目标。 "where" 可以是 N、F 或 L，其中 N 是实际显示，或者可以使用字符串变量
帧缓冲区关闭 [哪个]	关闭帧缓冲区并释放内存。可选参数 "which" 可以是 F 或 L。如果省略，则关闭两个。
帧缓冲区复制 从, 至 [,b]	对一个帧缓冲区到另一个帧缓冲区进行高度优化的全屏复制。 "from" 和 "to" 可以是 N、F 或 L，其中 N 是物理显示。 您只能在支持 BLIT 和透明文本的显示器上从 N 复制。 固件将在复制到和从不匹配的帧缓冲区时自动压缩或扩展 RGB 分辨率。 当然，从 RGB565 复制到 RGB121 会丢失信息，但对于许多应用（例如游戏）来说，16 种颜色级别已经足够。 在复制到液晶显示器时，可以使用可选参数“b”（FRAMEBUFFER COPY F/L, N, B）。这指示固件使用 PicoMite 中的第二处理器执行复制，并立即将控制权返回给 Basic 程序。
帧缓冲区等待	暂停处理，直到液晶显示器进入帧消隐状态。为 ILI9341、ST7789_320 和 ILI9488 显示器实现。用于减少写入屏幕时出现的伪影。
帧缓冲区合并 [颜色] [,模式] [,更新频率]	将层缓冲区和帧缓冲区的内容复制到液晶显示器上，省略特定颜色的所有像素。命令的前提条件是帧缓冲区和层缓冲区都已创建。帧缓冲区合并-将帧缓冲区的内容写入物理显示器，覆盖层缓冲区中设置的任何像素（非零）。 帧缓冲区合并 col-将帧缓冲区的内容写入物理显示器，覆盖层缓冲区中未设置为透明颜色 "col" 的帧缓冲区中的任何像素。颜色被指定为一个介于0到15之间的数字，代表： 0:黑色,1:蓝色,2:海蓝色,3:钴蓝,4:中绿色,5:天蓝色,6:绿色,7:青色,8:红色,9:品红,10:锈色,11:紫红色,12:棕色,13:淡紫色,14:黄色,15:白色 帧缓冲区合并 col,B - 除了上述内容外，物理显示的传输发生在第二个处理器上，控制立即返回到Basic 帧缓冲区合并 col,R [,updaterate] - 设置第二个处理器以持续更新物理显示，合并两个缓冲区。 如果未设置F或L，则自动设置帧缓冲区写入F。默认情况下，屏幕将尽可能快地更新（在133MHz下，ILI9341在SPI模式下每秒更新约13次，在8位并行模式下，ILI9341达到27帧每秒） 如果设置了"updaterate"，则屏幕将以毫秒指定的速率更新（除非低于显示器上可实现的最快速率）注意：在连续合并更新处于活动状态时，帧缓冲区写入不能设置为N。 帧缓冲区合并 col,A - 中止连续更新 此外，删除层缓冲区或帧缓冲区、ctrl-C或结束也将中止自动更新。 等待第二个处理器的最新更新完成，以允许无撕裂绘图
帧缓冲区同步	

<pre>FUNCTION xxx (arg1 [,arg2, ...]) [AS <type>] <statements> <statements> xxx = <return value> END FUNCTION</pre>	<p>Defines a callable function. This is the same as adding a new function to MMBasic while it is running your program.</p> <p>'xxx' is the function name and it must meet the specifications for naming a variable. The type of the function can be specified by using a type suffix (i.e. xxx\$) or by specifying the type using AS <type> at the end of the functions definition. For example:</p> <pre>FUNCTION xxx (arg1, arg2) AS STRING</pre> <p>'arg1', 'arg2', etc are the arguments or parameters to the function (the brackets are always required, even if there are no arguments). An array is specified by using empty brackets. i.e. arg3(). The type of the argument can be specified by using a type suffix (i.e. arg1\$) or by specifying the type using AS <type> (i.e. arg1 AS STRING).</p> <p>The argument can also be another defined function or the same function if recursion is to be used (the recursion stack is limited to 50 nested calls).</p> <p>To set the return value of the function you assign the value to the function's name. For example:</p> <pre>FUNCTION SQUARE(a) SQUARE = a * a END FUNCTION</pre> <p>Every definition must have one END FUNCTION statement. When this is reached the function will return its value to the expression from which it was called. The command EXIT FUNCTION can be used for an early exit.</p> <p>You use the function by using its name and arguments in a program just as you would a normal MMBasic function. For example:</p> <pre>PRINT SQUARE(56.8)</pre> <p>When the function is called each argument in the caller is matched to the argument in the function definition. These arguments are available only inside the function.</p> <p>Functions can be called with a variable number of arguments. Any omitted arguments in the function's list will be set to zero or a null string.</p> <p>Arguments in the caller's list that are a variable and have the correct type will be passed by reference to the function. This means that any changes to the corresponding argument in the function will also be copied to the caller's variable and therefore may be accessed after the function has ended. Arrays are passed by specifying the array name with empty brackets (e.g. arg()) and are always passed by reference and must be the correct type.</p> <p>You must not jump into or out of a function using commands like GOTO, GOSUB, etc. Doing so will have undefined side effects including the possibility of ruining your day.</p>
GOTO target	<p>Branches program execution to the target, which can be a line number or a label.</p>
GUI AREA #ref, startX, startY, width, height	<p>This will define an invisible area of the screen that is sensitive to touch and will generate touch down and touch up interrupts. It can be used as the basis for creating custom controls which are defined and managed by the program.</p> <p>'#ref' is the control's reference number. 'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions.</p>
GUI BCOLOUR colour, #ref1 [, #ref2, #ref3, etc]	<p>This will change the background colour of the specified controls to 'colour' which is an RGB value for the drawing colour.</p> <p>'#ref' is the control's reference number.</p>

<pre>FUNCTION xxx (arg1 [,arg2, ...]) [AS <type>] <statements> <statements> xxx = <返回值> 结束函数</pre>	<p>定义一个可调用的函数。这与在运行程序时向MMBasic添加新函数是相同的。</p> <p>'xxx' 是函数名称，必须符合变量命名的规范。函数的类型可以通过使用类型后缀（即 xxx\$）或在函数定义的末尾使用 AS <type> 来指定。例如：</p> <pre>FUNCTION xxx (arg1, arg2) AS STRING</pre> <p>'arg1'、'arg2' 等是函数的参数（括号始终是必需的，即使没有参数）。数组通过使用空括号来指定。即 arg3()。参数的类型可以通过使用类型后缀（即 arg1\$）或通过使用 AS <type>（即 arg1 AS STRING）来指定。</p> <p>参数也可以是另一个已定义的函数，或者如果要使用递归，则可以是相同的函数（递归栈限制为 50 层嵌套调用）。</p> <p>要设置函数的返回值，您需要将值赋给函数的名称。例如：</p> <pre>FUNCTION SQUARE (a) SQUARE = a * a END FUNCTION</pre> <p>每个定义必须有一个 END FUNCTION 语句。当达到这一点时，函数将返回其值给调用它的表达式。可以使用命令 EXIT FUNCTION 进行提前退出。</p> <p>您可以像使用普通的 MMBasic 函数一样，在程序中使用函数的名称和参数。例如：</p> <pre>PRINT SQUARE (56.8)</pre> <p>当调用函数时，调用者中的每个参数都与函数定义中的参数相匹配。这些参数仅在函数内部可用。</p> <p>函数可以使用可变数量的参数进行调用。在函数列表中省略的任何参数将被设置为零或空字符串。</p> <p>调用者列表中是变量且类型正确的参数将通过引用传递给函数。这意味着在函数中对相应参数的任何更改也会复制到调用者的变量中，因此在函数结束后仍然可以访问。数组通过指定数组名称并使用空括号（例如 arg()）来传递，始终通过引用传递，并且必须是正确的类型。</p> <p>你不能使用 GOTO、GOSUB 等命令跳入或跳出函数。这样做会产生未定义的副作用，包括可能会破坏你的一天。</p>
<p>GOTO 目标</p>	<p>将程序执行分支到目标，目标可以是行号或标签。</p>
<p>图形用户界面区域#ref, startX, startY, 宽度, 高度</p>	<p>这将定义一个屏幕上不可见的区域，该区域对触摸敏感，并将生成触摸按下和触摸抬起的中断。它可以作为创建自定义控件的基础，这些控件由程序定义和管理。</p> <p>#ref 是控件的参考编号。startX 和 startY 是左上角坐标，而 width 和 height 设置尺寸。</p>
<p>GUI BCOLOUR 颜色, #ref1 [, #ref2, #ref3, 等等]</p>	<p>这将把指定控件的背景颜色更改为 '颜色'。这是绘图颜色的 RGB 值。</p> <p>#ref 是控件的参考编号。</p>

<p>GUI BARGAUGE #ref, StartX, StartY, width, height, FColour, BColour, min, max, c1, ta, c2, tb, c3, tc, c4</p>	<p>Define either a horizontal or vertical analogue bar gauge. '#ref' is the control's reference number. 'StartX' and 'StartY' are the top left coordinates of the bar while 'width' is the horizontal width and 'height' the vertical height. If the width is less than the height the bar gauge will be drawn vertically with the graph growing from the bottom towards the top. Otherwise it will be drawn horizontally with the graph growing from the left towards the right. 'Fcolour' is the colour used for the gauge while 'Bcolour' is the background colour. 'min' is the minimum value of the gauge and 'max' is the maximum value (both floating point). A multi colour gauge can be created using 'c1' to 'c4' for the colours and 'ta' to 'tc' for the thresholds used to determine when the colour will change. 'width', 'height', 'FColour', 'BColour', 'min' and 'max' are optional and will default to the values used in the previous definition of a GUI BARGAUGE. 'c1', 'ta', 'c2', 'tb', 'c3', 'tc' and 'c4' are optional and if not specified the gauge will use less colours. If all are omitted the gauge will be drawn using 'Fcolour'. The section <i>Advanced Graphics</i> has a more detailed description.</p>
<p>GUI BUTTON #ref, caption\$, startX, startY, width, height [, FColour] [,BColour]</p>	<p>This will draw a momentary button which is a square switch with the caption on its face. When touched the visual image of the button will appear to be depressed and the control's value will be 1. When the touch is removed the value will revert to zero. #ref' is the control's reference number. 'caption\$' is the string to display on the face of the button. It can be a single string with two captions separated by a character (e.g. "UP DOWN"). When the button is up the first string will be used and when pressed the second will be used. 'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. ' FColour and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls or set with the COLOUR command.</p>
<p>GUI CAPTION #ref, text\$, startX, startY [,align\$] < b>, FColour] [, BColour]</p>	<p>This will draw a text string on the screen. #ref' is the control's reference number. 'text\$' is the string to display. 'startX' and 'startY' are the top left coordinates. 'align\$' is zero to three characters (a string expression or variable is also allowed) where the first letter is the horizontal alignment around X and can be L, C or R for LEFT, CENTER, RIGHT and the second letter is the vertical alignment around Y and can be T, M or B for TOP, MIDDLE, BOTTOM. A third character can be used in the string to indicate the rotation of the text. This can be 'N' for normal orientation, 'V' for vertical text with each character under the previous running from top to bottom, 'T' the text will be inverted (i.e. upside down), 'U' the text will be rotated counter clockwise by 90° and 'D' the text will be rotated clockwise by 90°. The default alignment is left/top with no rotation. 'FColour and 'BColour' are RGB values for the foreground and background colours. On a display that supports transparent text BColour can be -1 which means that the background will show through the gaps in the characters. FColour and 'BColour' are optional and default to the colours set by the COLOUR command.</p>

<p>GUI BARGAUGE #ref, StartX, StartY, 宽度, 高度, F颜色, B颜色, 最小, 最大, c1, ta, c2, tb, c3, tc, c4</p>	<p>定义水平或垂直的模拟条形量规。 '#ref' 是控件的参考编号。 'StartX' 和 'StartY' 是条形的左上角坐标，而 'width' 是水平宽度， 'height' 是垂直高度。如果宽度小于高度，量规将垂直绘制，图形从底部向顶部增长。否则，它将水平绘制，图形从左向右增长。</p> <p>'F颜色' 是用于量规的颜色，而 'B颜色' 是背景颜色。'min' 是量规的最小值，'max' 是最大值（均为浮点数）。</p> <p>可以使用 'c1' 到 'c4' 来创建多颜色量规，使用 'ta' 到 'tc' 来确定颜色变化的阈值。</p> <p>'宽度'、'高度'、'F颜色'、'B颜色'、'min' 和 'max' 是可选的，默认为之前定义的图形用户界面 BARGAUGE 中使用的值。'c1'、'ta'、'c2'、'tb'、'c3'、'tc' 和 'c4' 是可选的，如果未指定，量规将使用较少的颜色。如果全部省略，量规将使用 'F颜色' 绘制。</p> <p>部分高级图形有更详细的描述。</p>
<p>图形用户界面按钮#ref, 标题\$, 起始X, 起始Y, 宽度, 高度 [, F颜色] [,B颜色]</p>	<p>这将绘制一个瞬时按钮，它是一个带有标题的方形开关。当触摸时，按钮的视觉图像将看起来被按下，控件的值将为1。当触摸被移除时，值将恢复为零。</p> <p>#ref 是控件的参考编号。'标题' 是要显示在按钮面上的字符串。它可能是一个单一字符串，包含两个用 字符分隔的标题（例如："UP DOWN"）。当按钮处于上方时，将使用第一个字符串，按下时将使用第二个。</p> <p>'起始X' 和 '起始Y' 是左上角坐标，而 '宽度' 和 '高度' 设置尺寸。'F颜色' 和 'B颜色' 是前景和背景颜色的RGB值。</p> <p>'宽度'、'高度'、F颜色和B颜色是可选的，默认为之前控件使用的值或通过COLOUR命令设置的值。</p>
<p>图形用户界面标题#ref, text\$, startX, startY [,align\$] [, F颜色] [, B颜色]</p>	<p>这将在屏幕上绘制一个文本字符串。 '#ref' 是控件的参考编号。</p> <p>'text' 是要显示的字符串。'startX' 和 'startY' 是左上角的坐标。</p> <p>'align' 是零到三个字符（也允许字符串表达式或变量），其中第一个字母是围绕X的水平对齐，可以是L、C或R，分别表示左、居中、右；第二个字母是围绕Y的垂直对齐，可以是T、M或B，分别表示顶部、中间、底部。字符串中的第三个字符可以用于指示文本的旋转。</p> <p>这可以是'N'表示正常方向，'V'表示垂直文本，每个字符在上一个字符的下方，从上到下排列，'T'表示文本将被反转（即倒置），'U'表示文本将逆时针旋转90°，'D'表示文本将顺时针旋转90°。默认对齐方式是左/上，没有旋转。</p> <p>'F颜色' 和 'B颜色' 是前景和背景颜色的RGB值。在支持透明文本的显示器上，B颜色可以是-1，这意味着背景将透过字符之间的空隙显示出来。</p> <p>'F颜色' 和 'B颜色' 是可选的，默认为COLOUR命令设置的颜色。</p>

GUI CHECKBOX #ref, caption\$, startX, startY [, size] [, colour]	<p>This will draw a check box which is a small box with a caption. When touched an X will be drawn inside the box to indicate that this option has been selected and the control's value will be set to 1. When touched a second time the check mark will be removed and the control's value will be zero.</p> <p>'#ref' is the control's reference number.</p> <p>The string 'caption\$' will be drawn to the right of the control using the colours set by the COLOUR command.</p> <p>'startX' and 'startY' are the top left coordinates while 'size' set the height and width (the box is square). 'colour' is an RGB value for the drawing colour. 'size' and 'colour' are optional and default to that used in previous controls.</p>														
GUI DELETE #ref1 [,#ref2, #ref3, etc] or GUI DELETE ALL	<p>This will delete the controls in the list. This includes removing the image of the control from the screen using the current background colour and freeing the memory used by the control.</p> <p>'#ref' is the control's reference number. The keyword ALL can be used as the argument and that will delete all controls.</p>														
GUI DISABLE #ref1 [,#ref2, #ref3, etc] or GUI DISABLE ALL	<p>This will disable the controls in the list. Disabled controls do not respond to touch and will be displayed dimmed.</p> <p>'#ref' is the control's reference number. The keyword ALL can be used as the argument and that will disable all controls.</p> <p>GUI ENABLE can be used to restore the controls.</p>														
GUI DISPLAYBOX #ref, startX, startY, width, height, FColour, BColour	<p>This will draw a box with rounded corners that can be used to display a string</p> <p>'#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls.</p> <p>Any text can be displayed in the box by using the CtrlVal(r) = command. This is useful for displaying text, numbers and messages.</p> <p>This control does not respond to touch.</p>														
GUI ENABLE #ref1 [,#ref2, #ref3, etc] or GUI ENABLE ALL	<p>This will undo the effects of GUI DISABLE and restore the control(s) to normal operation.</p> <p>'#ref' is the control's reference number. The keyword ALL can be used as the argument and that will enable all controls.</p>														
GUI FCOLOUR colour, #ref1 [, #ref2, #ref3, etc]	<p>This will change the foreground colour of the specified controls to 'colour' which is an RGB value for the drawing colour.</p> <p>'#ref' is the control's reference number.</p>														
GUI FORMATBOX #ref, Format, startX, startY, width, height, FColour, BColour	<p>This will draw a box with rounded corners that can be used to create a virtual keypad for entry of data using a specific format.</p> <p>'#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls.</p> <p>The 'Format' argument specifies the format of the entry as follows:</p> <table> <tbody> <tr> <td>DATE1</td> <td>Date in UK/Aust/NZ format (dd/mm/yy)</td> </tr> <tr> <td>DATE2</td> <td>Date in USA format (mm/dd/yy)</td> </tr> <tr> <td>DATE3</td> <td>Date in international format (yyyy/mm/dd)</td> </tr> <tr> <td>TIME1</td> <td>Time in 24 hour notation (hh:mm)</td> </tr> <tr> <td>TIME2</td> <td>Time in 24 hour notation with seconds (hh:mm:ss)</td> </tr> <tr> <td>TIME3</td> <td>Time in 12 hour notation (hh:mm AM/PM)</td> </tr> <tr> <td>TIME4</td> <td>Time in 12 hour notation with seconds (hh:mm:ss AM/PM)</td> </tr> </tbody> </table>	DATE1	Date in UK/Aust/NZ format (dd/mm/yy)	DATE2	Date in USA format (mm/dd/yy)	DATE3	Date in international format (yyyy/mm/dd)	TIME1	Time in 24 hour notation (hh:mm)	TIME2	Time in 24 hour notation with seconds (hh:mm:ss)	TIME3	Time in 12 hour notation (hh:mm AM/PM)	TIME4	Time in 12 hour notation with seconds (hh:mm:ss AM/PM)
DATE1	Date in UK/Aust/NZ format (dd/mm/yy)														
DATE2	Date in USA format (mm/dd/yy)														
DATE3	Date in international format (yyyy/mm/dd)														
TIME1	Time in 24 hour notation (hh:mm)														
TIME2	Time in 24 hour notation with seconds (hh:mm:ss)														
TIME3	Time in 12 hour notation (hh:mm AM/PM)														
TIME4	Time in 12 hour notation with seconds (hh:mm:ss AM/PM)														

图形用户界面复选框 #ref, 标题\$, 起始X, 起始Y [, 大小] [, 颜色]	<p>这将绘制一个复选框，它是一个带有标题的小框。当被触摸时，框内将绘制一个X，以指示该选项已被选择，控件的值将被设置为1。当第二次触摸时，选中标记将被移除，控件的值将为零。</p> <p>'#ref' 是控件的参考编号。 字符串 'caption' 将使用 COLOUR 命令设置的颜色绘制在控件的右侧。</p> <p>'startX' 和 'startY' 是左上角的坐标，而 'size' 设置高度和宽度（该框是正方形的）。'colour' 是绘图颜色的 RGB 值。 'size' 和 'colour' 是可选的，默认为之前控件使用的值。</p>														
GUI DELETE #ref1 [,#ref2, #ref3, 等] 或 GUI DELETE ALL	<p>这将删除列表中的控件。这包括使用当前背景颜色从屏幕上移除控件的图像，并释放控件使用的内存。</p> <p>'#ref' 是控件的参考编号。关键字 ALL 可以作为参数使用，这将删除所有控件。</p>														
GUI DISABLE #ref1 [,#ref2, #ref3, 等] 或 GUI DISABLE ALL	<p>这将禁用列表中的控件。禁用的控件不响应触摸，并将以暗淡的方式显示。</p> <p>'#ref' 是控件的参考编号。关键字 ALL 可以作为参数使用，这将禁用所有控件。</p> <p>GUI ENABLE 可用于恢复控件。</p>														
GUI DISPLAYBOX #ref, startX, startY, width, height, FColour, BColour	<p>这将绘制一个带圆角的框，可以用来显示字符串 '#ref' 是控件的参考编号。</p> <p>'起始X' 和 '起始Y' 是左上角坐标，而 '宽度' 和 '高度' 设置尺寸。'FColour' 和 'BColour' 是前景和背景颜色的 RGB 值。'width'、'height'、FColour 和 BColour 是可选的，默认为之前控件使用的值。</p> <p>可以通过使用 CtrlVal(r)= 命令在框中显示任何文本。这对于显示文本、数字和消息非常有用。 此控件不响应触摸。</p>														
GUI ENABLE #ref1 [,#ref2, #ref3 等] 或 GUI ENABLE ALL	<p>这将撤销 GUI DISABLE 的效果，并将控件恢复到正常操作。</p> <p>'#ref' 是控件的参考编号。关键字 ALL 可以作为参数使用，这将禁用所有控件。</p>														
图形用户界面 F颜色颜色, #ref1 [, #ref2, #ref3, 等等]	<p>这将把指定控件的前景颜色更改为'颜色'，这是绘图颜色的RGB值。 '#ref' 是控件的参考编号。</p>														
图形用户界面 格式框 #ref, 格式, 起始X, 起始Y, 宽度, 高度, F颜色, B颜色	<p>这将绘制一个带圆角的框，可以用于创建一个虚拟键盘，以使用特定格式输入数据。 '#ref' 是控件的参考编号。</p> <p>'起始X' 和 '起始Y' 是左上角坐标，而 '宽度' 和 '高度' 设置尺寸。'FColour' 和 'BColour' 是前景和背景颜色的 RGB 值。'width'、'height'、FColour 和 BColour 是可选的，默认为之前控件使用的值。</p> <p>'格式'参数指定输入的格式如下：</p> <table> <tbody> <tr> <td>日期1</td> <td>英国/澳大利亚/新西兰格式的日期 (dd/mm/yy)</td> </tr> <tr> <td>日期2</td> <td>美国格式的日期 (mm/dd/yy)</td> </tr> <tr> <td>日期3</td> <td>国际格式的日期 (yyyy/mm/dd)</td> </tr> <tr> <td>时间1</td> <td>24小时制的时间 (hh:mm)</td> </tr> <tr> <td>时间2</td> <td>带秒的24小时制时间 (hh:mm:ss)</td> </tr> <tr> <td>时间3</td> <td>12小时制的时间 (hh:mm AM/PM)</td> </tr> <tr> <td>时间4</td> <td>12小时制的时间，带秒 (hh:mm:ss AM/PM)</td> </tr> </tbody> </table>	日期1	英国/澳大利亚/新西兰格式的日期 (dd/mm/yy)	日期2	美国格式的日期 (mm/dd/yy)	日期3	国际格式的日期 (yyyy/mm/dd)	时间1	24小时制的时间 (hh:mm)	时间2	带秒的24小时制时间 (hh:mm:ss)	时间3	12小时制的时间 (hh:mm AM/PM)	时间4	12小时制的时间，带秒 (hh:mm:ss AM/PM)
日期1	英国/澳大利亚/新西兰格式的日期 (dd/mm/yy)														
日期2	美国格式的日期 (mm/dd/yy)														
日期3	国际格式的日期 (yyyy/mm/dd)														
时间1	24小时制的时间 (hh:mm)														
时间2	带秒的24小时制时间 (hh:mm:ss)														
时间3	12小时制的时间 (hh:mm AM/PM)														
时间4	12小时制的时间，带秒 (hh:mm:ss AM/PM)														

	<p>DATETIME1 Date (UK fmt) and time (12 hour) (dd/mm/yy hh:mm AM/PM) DATETIME2 Date (UK fmt) and time (24 hour) (dd/mm/yy hh:mm) DATETIME3 Date (USA fmt) and time (12 hour) (mm/dd/yy hh:mm AM/PM) DATETIME4 Date (USA fmt) and time (24 hour) (mm/dd/yy hh:mm) LAT1 Latitude in degrees, minutes and seconds (dd° mm' ss" N/S) LAT2 Latitude with seconds to one decimal place (dd° mm' ss.s" N/S) LONG1 Longitude in degrees, minutes and seconds (ddd° mm' ss" E/W) LONG2 Longitude seconds to one decimal place (ddd° mm' ss.s" E/W) ANGLE1 Angle in degrees and minutes (ddd° mm') For example, this command:</p> <pre>GUI FORMATBOX #1, LAT1, 50, 50, 300, 50</pre> <p>would create a format box which would accept the entry of latitude in the format of dd° mm' ss" N/S. The value of CtrlVal(#1) would be a string which includes the numbers and separating characters. For example an entry of 17 degrees, 32 minutes and 1 second south would result in the string 17° 32' 01" S MMBasic will try to position the virtual keypad on the screen so as to not obscure the format box that caused it to appear. A pen down interrupt will be generated just before the keypad is deployed and a key up interrupt will be generated when the entry is complete and the keypad is hidden. .</p>
GUI FORMATBOX CANCEL	This will dismiss a virtual keypad if it is displayed on the screen. It is the same as if the user touched the cancel key except that the touch up interrupt is not generated. If a keypad is not displayed this command will do nothing.
GUI FRAME #ref, caption\$, startX, startY, width, height, colour	<p>This will draw a frame which is a box with round corners and a caption. '#ref' is the control's reference number. 'caption\$' is a string to display as the caption. 'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'colour' is an RGB value for the drawing colour. 'width', 'height' and 'colour' are optional and default to that used in previous controls.</p> <p>A frame is useful when a group of controls need to be visually brought together. It is also used to surround a group of radio buttons and MMBasic will arrange for the radio buttons surrounded by the frame to be exclusive. i.e. when one radio button is selected any other button that was selected and within the frame will be automatically deselected.</p> <p>A frame does not respond to touch.</p>
GUI GAUGE #ref, StartX, StartY, Radius, FColour, BColour, min, max, nbrdec, units\$, c1, ta, c2, tb, c3, tc, c4	<p>Define a graphical circular analogue gauge with a digital display in the centre. '#ref' is the control's reference number.</p> <p>'StartX' and 'StartY' are the coordinates of the centre of the gauge, 'Radius' is the distance from the centre to the outer edge.</p> <p>'min' is the minimum value of the gauge and 'max' is the maximum value (both floating point).</p> <p>'nbrdec' specifies the number of decimal places to be used when drawing the digital value in the centre of the gauge. Under this 'units\$' will be displayed.</p> <p>'Fcolour' is the colour used for the gauge while 'Bcolour' is the background colour. A multi colour gauge can be created using 'c1' to 'c4' for the colours and 'ta' to 'tc' for the thresholds used to determine when the colour will change. When colours and thresholds are specified the background of the gauge will be drawn with a dull version of the colour at that level. Also the digital value will change to the colour specified by the current value.</p> <p>'Radius', 'FColour', 'BColour', 'min', 'max', 'nbrdec' and 'units\$' are optional and will default to the values used in the previous definition of a GUI GAUGE.</p> <p>'c1', 'ta', 'c2', 'tb', 'c3', 'tc' and 'c4' are optional and if not specified the gauge will use less colours. If all are omitted the gauge will be drawn using 'Fcolour'. The section <i>Advanced Graphics</i> has a more detailed description.</p>

	<p>DATETIME1 日期（英国格式）和时间（12小时制）(dd/mm/yy hh:mm AM/PM) DATETIME2 日期（英国格式）和时间（24小时制）(dd/mm/yy hh:mm) DATETIME3 日期（美国格式）和时间（12小时制）(mm/dd/yy hh:mm AM/PM) DATETIME4 日期（美国格式）和时间（24小时制）(mm/dd/yy hh:mm)</p> <p>LAT1 纬度以度、分钟和秒表示(dd° mm' ss" N/S) LAT2 纬度，秒保留一位小数(dd° mm' ss.s" N/S) LONG1 经度以度、分钟和秒表示(ddd° mm' ss" E/W) LONG2 经度秒数保留一位小数(ddd° mm' ss.s" E/W) ANGLE1 角度以度和分钟表示(dd° mm')</p> <p>例如，这个命令：</p> <pre>GUI FORMATBOX #1, LAT1, 50, 50, 300, 50</pre> <p>将创建一个格式框，接受以 dd° mm' ss" N/S 格式输入的纬度。Ctrl Val(#1) 的值将是一个包含数字和分隔字符的字符串。例如，输入 17 度、32 分和 1 秒南将导致字符串 17° 32' 01" S。MMBasic 将尝试在屏幕上定位虚拟键盘，以避免遮挡导致其出现的格式框。在键盘显示之前，将生成一个笔下中断，而在输入完成并隐藏键盘时，将生成一个按键抬起中断。</p>
图形用户界面格式框 取消	如果虚拟键盘在屏幕上显示，这将使其消失。这与用户触摸取消键是一样的，只是不会生成触摸抬起中断。如果键盘未显示，此命令将无效。
图形用户界面框架#ref, caption\$, startX, startY, width, height, 颜色	<p>这将绘制一个带有圆角和标题的框架。 '#ref' 是控件的参考编号。 ''caption' 是要显示的标题字符串。'startX' 和 'startY' 是左上角坐标，而 'width' 和 'height' 设置尺寸。'颜色' 是绘图颜色的 RGB 值。'width'、'height' 和 '颜色' 是可选的，默认为之前控件使用的值。</p> <p>帧在需要将一组控件视觉上聚集在一起时非常有用。它也用于围绕一组单选按钮，MMBasic 将确保被帧包围的单选按钮是互斥的。即当选择一个单选按钮时，帧内任何其他已选择的按钮将自动取消选择。</p> <p>帧不响应触摸。</p>
GUI GAUGE #ref, StartX, StartY, Radius, FColour, BColour, min, max, nbrdec, units\$, c1, ta, c2, tb, c3, tc, c4	<p>定义一个带有数字显示的图形圆形模拟量规，显示在中心。 '#ref' 是控件的参考编号。 'StartX' 和 'StartY' 是量规中心的坐标，'Radius' 是从中心到外边缘的距离。 'min' 是量规的最小值，'max' 是最大值（均为浮点数）。</p> <p>'nbrdec' 指定在量规中心绘制数字值时使用的小数位数。在此将显示 'units'。</p> <p>'F颜色' 是用于量规的颜色，而 'B颜色' 是背景颜色。可以使用 'c1' 到 'c4' 来创建多颜色量规，颜色和 'ta' 到 'tc' 用于确定颜色变化的阈值。</p> <p>当指定颜色和阈值时，量规的背景将以该级别颜色的暗淡版本绘制。此外，数字值将更改为当前值指定的颜色。</p> <p>'Radius'、'FColour'、'BColour'、'min'、'max'、'nbrdec' 和 'units' 是可选的，默认为上一个 GUI GAUGE 定义中使用的值。 'c1'、'ta'、'c2'、'tb'、'c3'、'tc' 和 'c4' 是可选的，如果未指定，量规将使用较少的颜色。如果全部省略，量规将使用 'F颜色' 绘制。 部分高级图形有更详细的描述。</p>

GUI HIDE #ref1 [,#ref2, #ref3, etc] or GUI HIDE ALL	This will hide the controls in the list. Hidden controls do not respond to touch and will not be visible. '#ref' is the control's reference number. The keyword ALL can be used as the argument and that will hide all controls. GUI SHOW can be used to restore the controls.
GUI INTERRUPT down [, up]	This command will setup an interrupt that will be triggered on a touch on the LCD panel and optionally if the touch is released. 'down' is the subroutine to call when a touch down has been detected. 'up' is the subroutine to call when the touch has been lifted from the screen ('up' and 'down' can point to the same subroutine if required). Specifying the number zero (single digit) as the argument will cancel both of these interrupts. ie: GUI INTERRUPT 0.
GUI LED #ref, caption\$, centerX, centerY, radius, colour	This will draw an indicator light which looks like a panel mounted LED. A LED does not respond to touch. '#ref' is the control's reference number. The string 'caption\$' will be drawn to the right of the control using the colours set by the COLOUR command. 'centerX' and 'centerY' are the coordinates of the centre of the LED and 'radius' is the radius of the LED. 'colour' is an RGB value for the drawing colour. 'radius' and 'colour' are optional and default to that used in previous controls. When a LED's value is set to a value of one it will be illuminated and when it is set to zero it will be off (a dull version of its colour attribute). The LED can be made to flash on then off by setting the value of the LED to a number greater than one which is the time in milliseconds that it should remain on. The colour can be changed with the GUI FCOLOUR command.
GUI NUMBERBOX #ref, startX, startY, width, height, FColour, BColour	This will draw a box with rounded corners that can be used to create a virtual numeric keypad for data entry. '#ref' is the control's reference number. 'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls. When the box is touched a numeric keypad will appear on the screen. Using this virtual keypad any number can be entered into the box including a floating point number in exponential format. The new number will replace the number previously in the box. The value of the control can set to a literal string (not an expression) starting with two hash characters. For example: CtrlVal(nnn) = "#Enter Number" and in that case the string (without the leading two hash characters) will be displayed in the box with reduced brightness. This can be used to give the user a hint as to what should be entered (called "ghost text"). Reading the value of the control displaying ghost text will return zero. When the control is used normally the ghost text will vanish. MMBasic will try to position the virtual keypad on the screen so as to not obscure the number box that caused it to appear. A pen down interrupt will be generated just before the keypad is deployed and a key up interrupt will be generated when the Enter key is touched and the keypad is hidden. Also, when the Enter key is touched the entered number will be evaluated as a number and the NUMBERBOX control redrawn to display this number.
GUI NUMBERBOX CANCEL	This will dismiss a virtual keypad if it is displayed on the screen. It is the same as if the user touched the cancel key except that the touch up interrupt is not generated. If a keypad is not displayed this command will do nothing.
GUI PAGE #n [,#n2, #n3, etc]	This will switch the display to show controls that have been assigned (via the

GUI HIDE #ref1 [,#ref2, #ref3, 等] 或 GUI HIDE ALL	这将隐藏列表中的控件。隐藏的控件不会响应触摸并且将不可见。 '#ref' 是控件的参考编号。关键字 ALL 可以作为参数使用，这将隐藏所有控件。 GUI SHOW 可用于恢复控件。
GUI INTERRUPT down [, up]	此命令将设置一个中断，该中断将在液晶面板上触摸时触发，并可选择在触摸释放时触发。 'down' 是在检测到触摸按下时调用的子程序。'up' 是在触摸从屏幕上抬起时调用的子程序（如果需要，'up' 和 'down' 可以指向同一个子程序）。 将数字零（单个数字）指定为参数将取消这两个中断。即：GUI INTERRUPT 0。
GUI LED #ref, title\$, centerX, centerY, radius, 颜色	这将绘制一个指示灯，看起来像是面板安装的LED。LED不响应触摸。 '#ref' 是控件的参考编号。 字符串 'caption' 将使用 COLOUR 命令设置的颜色绘制在控件的右侧。 'centerX'和'centerY'是LED中心的坐标，'radius'是LED的半径。'colour'是绘图颜色的RGB值。 'radius'和'colour'是可选的，默认为之前控件使用的值。 当LED的值设置为1时，它将被点亮；当设置为0时，它将熄灭（其颜色属性的暗淡版本）。通过将LED的值设置为大于1的数字，可以使LED闪烁，数字表示它应保持点亮的时间（以毫秒为单位）。颜色可以通过GUI FCOLOUR命令进行更改。
GUI NUMBERBOX #ref, startX, startY, width, height, FColour, BColour	这将绘制一个带圆角的框，可以用于创建虚拟数字键盘以进行数据输入。 '#ref' 是控件的参考编号。 '起始X' 和 '起始Y' 是左上角坐标，而 '宽度' 和 '高度' 设置尺寸。'FColour' 和 'BColour' 是前景和背景颜色的 RGB 值。'width'、'height'、'FColour' 和 'BColour' 是可选的，默认为之前控件使用的值。 当触摸该框时，屏幕上将出现一个数字键盘。使用这个虚拟键盘，可以在框中输入任何数字，包括以指数格式表示的浮点数。新输入的数字将替换框中之前的数字。 控件的值可以设置为以两个井号字符开头的字面字符串（而不是表达式）。例如：CtrlVal(nnn) = "##输入数字"，在这种情况下，字符串（去掉前面的两个井号字符）将以减弱的亮度显示在框中。这可以用来给用户提示应该输入什么（称为“幽灵文本”）。读取显示幽灵文本的控件的值将返回零。当控件正常使用时，幽灵文本将消失。 MMBasic将尝试在屏幕上定位虚拟键盘，以避免遮挡导致其出现的数字框。在键盘部署之前，将生成一个笔下中断，当触摸回车键并隐藏键盘时，将生成一个按键抬起中断。此外，当触摸回车键时，输入的数字将被求值为一个数字，并且数字框控件将重新绘制以显示该数字。
图形用户界面数字框取消	如果虚拟键盘在屏幕上显示，这将使其消失。这与用户触摸取消键是一样的，只是不会生成触摸抬起中断。如果键盘未显示，此命令将无效。
图形用户界面页面#n [,#n2, #n3, 等]	这将切换显示以显示已分配的控件（通过

	<p>GUI SETUP command) to the page numbers specified on the command line (#n, #n2, etc). Any controls that were displayed but are not on the current list of pages will be automatically hidden. Any controls on a page that was displayed on the old screen and is also specified in the new PAGE command will remain unaffected.</p> <p>The default when a program starts running is PAGE 1 and GUI SETUP 1. This means that if these commands are not used the program will run as normal showing all GUI controls that have been defined.</p> <p>See also the GUI SETUP command.</p>
GUI RADIO #ref, caption\$, centerX, centerY, radius, colour	<p>This will draw a radio button with a caption. '#ref' is the control's reference number.</p> <p>The string 'caption\$' will be drawn to the right of the control using the colours set by the COLOUR command.</p> <p>'centerX' and 'centerY' are the coordinates of the centre of the button and 'radius' is the radius of the button. 'colour' is an RGB value for the drawing colour. 'radius' and 'colour' are optional and default to that used in previous controls.</p> <p>When touched the centre of the button will be illuminated to indicate that this option has been selected and the control's value will be 1. When another radio button is selected the mark on this button will be removed and its value will be zero. Radio buttons are grouped together when surrounded by a frame and when one button in the group is selected all others in the group will be deselected. If a frame is not used all buttons on the screen will be grouped together.</p>
GUI REDRAW #ref1 [,#ref2, #ref3, etc] or GUI REDRAW ALL	<p>This will redraw the controls on the screen. It is useful if the screen image has somehow been corrupted.</p> <p>'#ref' is the control's reference number. The keyword ALL can be used as the argument and that will first clear the screen then redraw all controls. This is useful if the whole screen needs to be refreshed.</p>
GUI SETUP #n	<p>This will allocate any new controls created to the page '#n'.</p> <p>This command can be used as many times as needed while GUI controls are being defined. The default when a program starts running is GUI SETUP 1.</p> <p>See also the GUI PAGE command.</p>
GUI SHOW #ref1 [,#ref2, #ref3, etc] or GUI SHOW ALL	<p>This will undo the effects of GUI HIDE and restore the control(s) to being visible and capable of normal operation.</p> <p>'#ref' is the control's reference number. The keyword ALL can be used as the argument and that will disable all controls.</p>
GUI SPINBOX #ref, startX, startY, width, height, FColour, BColour, Step, Minimum, Maximum	<p>This will draw a box with up/down icons on either end. When these icons are touched the number in the box will be incremented or decremented. Holding down the up/down icons will repeat the step at a fast rate.</p> <p>'#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls.</p> <p>'Step' sets the amount to increment/decrement the number with each touch. 'Minimum' and 'Maximum' set limits on the number that can be entered. All three parameters can be floating point numbers and are optional. The default for 'Step' is 1 and 'Minimum' and 'Maximum' if omitted will default to no limit.</p>
GUI SWITCH #ref, caption\$, startX, startY, width, height, FColour, BColour	<p>This will draw a latching switch which is a square switch that latches when touched.</p> <p>'#ref' is the control's reference number.</p> <p>'caption\$' is a string to display as the caption on the face of the switch. 'startX'</p>

	<p>图形用户界面设置命令) 到命令行中指定的页面编号 (#n, #n2, 等)。任何已显示但不在当前页面列表中的控件将自动隐藏。任何在旧屏幕上显示的页面中的控件, 如果在新的页面命令中也被指定, 将保持不变。</p> <p>程序开始运行时的默认值是PAGE 1和图形用户界面设置1。这意味着如果不使用这些命令, 程序将正常运行, 显示所有已定义的图形用户界面控件。 另请参见 GUI 设置命令。</p>
图形用户界面单选框#ref, 标题\$, 居中X, 居中Y, 半径, 颜色	<p>这将绘制一个带标题的单选按钮。 '#ref' 是控件的参考编号。 字符串 'caption' 将使用 COLOUR 命令设置的颜色绘制在控件的右侧。</p> <p>'居中X' 和 '居中Y' 是按钮中心的坐标, ' '半径' 是按钮的半径。'颜色' 是绘图颜色的 RGB 值。'半径' 和 '颜色' 是可选的, 默认为之前控件使用的值。</p> <p>当触摸时, 按钮的中心将被照亮, 以指示此选项已被选择, 控件的值 将为 1。当选择另一个单选按钮时, 此按钮上的标记将被移除, 其值将 为零。单选按钮在被框架包围时会被分组, 当组中的一个按钮被选中时 , 组中的所有其他按钮将被取消选择。如果不使用框架, 屏幕上的 所有按钮将被分组在一起。</p>
GUI 重绘#ref1 [,#ref2, #ref3, 等] 或 GUI 重绘全部	<p>这将重新绘制屏幕上的控件。如果屏幕图像以某种方式被损坏, 这将非 常有用。</p> <p>'#ref' 是控件的参考编号。关键字 ALL 可以作为参数使用, 这将首先清 除屏幕, 然后重新绘制所有控件。如果需要刷新整个屏幕, 这将非常 有用。</p>
GUI 设置 #n	<p>这将把创建的任何新控件分配到页面 '#n'。</p> <p>在定义 GUI 控件时, 可以根据需要多次使用此命令。程序开始运行时 的默认值是 GUI 设置 1。</p> <p>另请参见 GUI 页面命令。</p>
GUI 显示#ref1 [,#ref2, #ref3, 等] 或 GUI 显示全部	<p>这将撤销 GUI 隐藏的效果, 并将控件恢复为可见并能够正常操作。</p> <p>'#ref' 是控件的参考编号。关键字 ALL 可以作为参数使用, 这将禁用所 有控件。</p>
GUI 旋转框#ref, startX, startY, 宽度, 高度, F颜色, B颜色, 步长, 最小值, 最大值	<p>这将绘制一个两端带有上下图标的框。当触摸这些图标时, 框中的数字 将增加或减少。按住上下图标将以快速的速度重复步骤。</p> <p>'#ref' 是控件的参考编号。</p> <p>'起始X' 和 '起始Y' 是左上角坐标, 而 '宽度' 和 '高度' 设置尺寸。'FColour' 和 'BColour' 是前景和背景颜色的 RGB 值。'width'、'height'、FColour 和 BColour 是可选的, 默认为之前控件使用的值。</p> <p>'步长' 设置每次触摸时增加/减少数字的量。 '最小值' 和 '最大值' 设置可以输入的数字的限制。这三个参数都可以是 浮点数, 并且是可选的。'步长' 的默认值为 1, 如果省略'最小值'和'最大 值', 则默认为无限制。</p>
图形用户界面开关#ref, caption\$, startX, startY, width, height, F颜色, B颜色	<p>这将绘制一个锁存开关, 这是一个触摸后会锁存的方形开关。</p> <p>'#ref' 是控件的参考编号。</p> <p>"caption"是一个字符串, 用于在开关的表面上显示标题。'startX'</p>

	<p>and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls.</p> <p>When touched the visual image of the button will appear to be depressed and the control's value will be 1. When touched a second time the switch will be released and the value will revert to zero. Caption can consist of two captions separated by a character (e.g. "ON OFF"). When this is used the switch will appear to be a toggle switch with each half of the caption used to label each half of the toggle switch.</p>
GUI TEXTBOX #ref, startX, startY, width, height, FColour, BColour	<p>This will draw a box with rounded corners that can be used to create a virtual keyboard for data entry</p> <p>'#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls. On a display that supports transparent text BColour can be -1 which means that the background will show through the gaps in the characters.</p> <p>When the box is touched a QWERTY keyboard will appear on the screen. Using this virtual keyboard any text can be entered into the box including upper/lower case letters, numbers and any other characters in the ASCII character set. The new text will replace any text previously in the box.</p> <p>The value of the control can set to a string starting with two hash characters. For example: CtrlVal(nnn) = "##Enter Filename" and in that case the string (without the leading two hash characters) will be displayed in the box with reduced brightness. This can be used to give the user a hint as to what should be entered (called "ghost text"). Reading the value of the control displaying ghost text will return an empty string. When the control is used normally the ghost text will vanish.</p> <p>MMBasic will try to position the virtual keyboard on the screen so as to not obscure the text box that caused it to appear. A pen down interrupt will be generated just before the keyboard is deployed and a key up interrupt will be generated when the Enter key is touched and the keyboard is hidden. .</p>
GUI TEXTBOX CANCEL	<p>This will dismiss a virtual keyboard if it is displayed on the screen. It is the same as if the user touched the cancel key except that the touch up interrupt is not generated. If a keyboard is not displayed this command will do nothing.</p>
GUI BITMAP x, y, bits [, width] [, height] [, scale] [, c] [, bc]	<p>Displays the bits in a bitmap on an LCD panel starting at 'x' and 'y' on an attached LCD panel.</p> <p>'height' and 'width' are the dimensions of the bitmap as displayed on the LCD panel and default to 8x8.</p> <p>'scale' is optional and defaults to that set by the FONT command.</p> <p>'c' is the drawing colour and 'bc' is the background colour. They are optional and default to the current foreground and background colours.</p> <p>The bitmap can be an integer or a string variable or constant and is drawn using the first byte as the first bits of the top line (bit 7 first, then bit 6, etc) followed by the next byte, etc. When the top line has been filled the next line of the displayed bitmap will start with the next bit in the integer or string.</p> <p>See the section <i>Graphics Commands and Functions</i> for a definition of the colours and graphics coordinates.</p>
GUI CALIBRATE or GUI CALIBRATE a,b,c,d,d	<p>This command is used to calibrate the touch feature on an LCD panel. It will display a series of targets on the screen and wait for each one to be precisely touched.</p> <p>The command can also be used with five arguments which specify the calibration values and in this case the calibration will be done without</p>

	<p>和'startY'是左上角的坐标，而'width'和'height'设置尺寸。'FColour'和 'BColour' 是前景和背景颜色的 RGB 值。'width'、'height'、FColour 和 BColour 是可选的，默认为之前控件使用的值。</p> <p>当触摸时，按钮的视觉图像将显示为被按下，控件的值将为1。当第二次触摸时，开关将被释放，值将恢复为零。标题可以由两个标题组组成 字符分隔（例如："ON OFF"）。使用此功能时，开关将显示为一个切换开关，每半部分的标题用于标记切换开关的每一半。</p>
图形用户界面文本框#ref, startX, startY, 宽度, 高度, F颜色, B颜色	<p>这将绘制一个具有圆角的框，可以用于创建数据输入的虚拟键盘。'#ref' 是控件的参考编号</p> <p>。</p> <p>'起始X' 和 '起始Y' 是左上角坐标，而 '宽度' 和 '高度' 设置尺寸。'FColour' 和 'BColour' 是前景和背景颜色的 RGB 值。'宽度'、'高度'、F颜色和B颜色是可选的，默认为之前控件使用的值。在支持透明文本的显示器上，B颜色可以为 -1，这意味着背景将透过字符的间隙显示出来。</p> <p>当触摸框时，QWERTY键盘将出现在屏幕上。</p> <p>使用这个虚拟键盘可以在文本框中输入任何文本，包括大小写字母、数字和ASCII字符集中的其他字符。新文本将替换文本框中之前的任何文本。控件的值可以设置为以两个哈希字符开头的字符串</p> <p>。</p> <p>例如：CtrlVal(nnn) = "##输入文件名"，在这种情况下，字符串（去掉前面的两个哈希字符）将以减弱的亮度显示在文本框中。这可以用来给用户一个提示，告诉他们应该输入什么（称为“幽灵文本”）。读取显示幽灵文本的控件的值将返回一个空字符串。当控件正常使用时，幽灵文本将消失。</p> <p>MMBasic将尝试在屏幕上定位虚拟键盘，以避免遮挡导致其出现的文本框。在键盘部署之前将生成一个笔下中断，当按下回车键并隐藏键盘时将生成一个按键抬起中断。</p>
图形用户界面文本框 取消	如果虚拟键盘在屏幕上显示，这将关闭它。这与用户触摸取消键是一样的，只是不会生成触摸抬起中断。如果键盘未显示，此命令将无效。
图形位图 x, y, 位数 [, 宽度] [, 高度] [, 缩放] [, 颜色] [, 背景色]	<p>在连接的液晶面板上，从 'x' 和 'y' 开始显示位图中的位。</p> <p>'高度' 和 '宽度' 是在液晶面板上显示的位图的尺寸，默认为 8x8。</p> <p>'缩放' 是可选的，默认为 FONT 命令设置的值。</p> <p>'颜色' 是绘图颜色，'背景色' 是背景颜色。它们是可选的，默认为当前的前景和背景颜色。</p> <p>位图可以是整数或字符串变量或常量，并使用第一个字节作为顶部行的第一个位（先是位 7，然后是位 6，依此类推），接着是下一个字节，等等。当顶部行填满后，显示的位图的下一行将从整数或字符串中的下一个位开始。</p> <p>请参见章节图形命令和函数以获取颜色和图形坐标的定义。</p>
图形用户界面 校准 或 图形用户界面 校准 a,b,c,d,d	<p>此命令用于校准液晶面板上的触摸功能。它将在屏幕上显示一系列目标，并等待每个目标被精确触摸。</p> <p>该命令还可以使用五个参数，这些参数指定校准值，在这种情况下，校准将不进行</p>

	displaying any targets or requiring an input from the user. To discover the values use the OPTION LIST after calibrating the display normally. Note that these values are specific to that display and can vary considerably.
GUI RESET LCDPANEL	Will reinitialise the configured LCD panel. Initialisation is automatically done when the PicoMite starts up but in some circumstances it may be necessary to interrupt power to the LCD panel (e.g. to save battery power) and this command can then be used to reinitialise the display.
GUI TEST LCDPANEL or GUI TEST TOUCH	Will test the display or touch feature on an LCD panel. With GUI TEST LCDPANEL an animated display of colour circles will be rapidly drawn on top of each other. With GUI TEST TOUCH the screen will blank and wait for a touch which will cause a white dot to be placed on the display marking the touch position on the screen. Any character entered at the console will terminate the test.
I2C OPEN speed, timeout	Enables the first I ² C module in master mode. ‘speed’ is the clock speed (in KHz) to use and must be one of 100, 400 or 1000. ‘timeout’ is a value in milliseconds after which the master send and receive commands will be interrupted if they have not completed. The minimum value is 100. A value of zero will disable the timeout (though this is not recommended).
I2C WRITE addr, option, sendlen, senddata [,senddata ..]	Send data to the I ² C slave device. ‘addr’ is the slave’s I ² C address. ‘option’ can be 0 for normal operation or 1 to keep control of the bus after the command (a stop condition will not be sent at the completion of the command) ‘sendlen’ is the number of bytes to send. ‘senddata’ is the data to be sent - this can be specified in various ways (all values sent will be between 0 and 255). Notes: <ul style="list-style-type: none">• The data can be supplied as individual bytes on the command line. Example: I2C WRITE &H6F, 0, 3, &H23, &H43, &H25• The data can be in a one dimensional array specified with empty brackets (i.e. no dimensions). ‘sendlen’ bytes of the array will be sent starting with the first element. Example: I2C WRITE &H6F, 0, 3, ARRAY() The data can be a string variable (not a constant). Example: I2C WRITE &H6F, 0, 3, STRING\$
I2C READ addr, option, rcvlen, rcvbuf	Get data from the I2C slave device. ‘addr’ is the slave’s I ² C address. ‘option’ can be 0 for normal operation or 1 to keep control of the bus after the command (a stop condition will not be sent at the completion of the command) ‘rcvlen’ is the number of bytes to receive. ‘rcvbuf’ is the variable or array used to save the received data - this can be: <ul style="list-style-type: none">• A string variable. Bytes will be stored as sequential characters.• A one dimensional array of numbers specified with empty brackets. Received bytes will be stored in sequential elements of the array starting with the first. Example: I2C READ &H6F, 0, 3, ARRAY() A normal numeric variable (in this case rcvlen must be 1). Disables the master I ² C module. This command will also send a stop if the bus is still held.
I2C SLAVE	See Appendix B
I2C2	The same set of commands as for I2C (above) but applying to the second I ² C channel.

	显示任何目标或需要用户输入。要发现这些值，请在正常校准显示后使用选项列表。请注意，这些值是特定于该显示器的，可能会有很大差异。
GUI 重置 液晶面板	将重新初始化配置的液晶面板。初始化在 PicoMite 启动时会自动完成，但在某些情况下，可能需要中断液晶面板的电源（例如，为了节省电池电量），然后可以使用此命令重新初始化显示。
图形用户界面测试液晶面板 或 图形用户界面测试触摸	<p>将测试液晶面板上的显示或触摸功能。</p> <p>使用图形用户界面测试液晶面板，将快速绘制颜色圆圈的动画显示在彼此之上。</p> <p>使用图形用户界面测试触摸，屏幕将变为空白并等待触摸，这将导致在显示器上放置一个白点，标记触摸位置。</p> <p>在控制台输入的任何字符将终止测试。</p>
I2C 打开速度, 超时 I2C 写入 addr, 选项, 发送长度, 发送数据 [,发送数据 ..]	<p>启用第一个I²C模块以主控模式运行。‘速度’是要使用的时钟速度（以千赫为单位），必须为100、400或1000之一。</p> <p>‘超时’是一个以毫秒为单位的值，如果主控发送和接收命令未完成，则将在此值之后中断。最小值为100。值为零将禁用超时（尽管不推荐这样做）。</p> <p>将数据发送到从设备²C。‘addr’是从设备的 I²C地址。</p> <p>‘option’可以为0表示正常操作，或1表示在命令后保持对总线的控制（在命令完成时不会发送停止条件）‘sendlen’是要发送的字节数。‘senddata’是要发送的数据 - 这可以通过多种方式指定（所有发送的值将在0到255之间）。</p> <p>注意：</p> <ul style="list-style-type: none"> • 数据可以作为单个字节在命令行上提供。 示例：I2C WRITE &H6F, 0, 3, &H23, &H43, &H25 • 数据可以是一个用空括号指定的一维数组（即没有尺寸）。‘sendlen’字节的数组将从第一个元素开始发送。 示例：I2C WRITE &H6F, 0, 3, ARRAY() <p>数据可以是一个字符串变量（不是常量）。</p> <p>示例：I2C WRITE &H6F, 0, 3, STRING\$</p> <p>从I2C从设备获取数据。‘addr’是从设备的I²C地址。‘option’可以为0表示正常操作，或1表示在命令完成后保持对总线的控制（在命令完成时不会发送停止条件）。‘rcvlen’是要接收的字节数。</p> <p>‘rcvbuf’是用于保存接收数据的变量或数组 - 这可以是：</p> <ul style="list-style-type: none"> • 一个字符串变量。字节将作为连续字符存储。 • 一个用空括号指定的一维数字数组。 接收到的字节将存储在数组的连续元素中，从第一个开始。 <p>示例：I2C READ &H6F, 0, 3, ARRAY()</p> <p>一个普通的数字变量（在这种情况下，rcvlen必须为1）。</p>
I2C 关闭	禁用主控 I ² C 模块。如果总线仍然被占用，此命令还会发送一个停止信号。
I2C 从属	见附录 B
I2C2	与 I2C（上述）相同的一组命令，但适用于第二个 I ² C 通道。

<p>IF expr THEN stmt [: stmt] or IF expr THEN stmt ELSE stmt</p>	<p>Evaluates the expression 'expr' and performs the statement following the THEN keyword if it is true or skips to the next line if false. If there are more statements on the line (separated by colons (:)) they will also be executed if true or skipped if false. The ELSE keyword is optional and if present the statement(s) following it will be executed if 'expr' resolved to be false.</p> <p>The 'THEN statement' construct can be also replaced with: GOTO linenumber label'.</p> <p>This type of IF statement is all on one line.</p>
<p>IF expression THEN <statements> [ELSEIF expression THEN <statements>] [ELSE <statements>] ENDIF</p>	<p>Multiline IF statement with optional ELSE and ELSEIF cases and ending with ENDIF. Each component is on a separate line.</p> <p>Evaluates 'expression' and performs the statement(s) following THEN if the expression is true or optionally the statement(s) following the ELSE statement if false. The ELSEIF statement (if present) is executed if the previous condition is false and it starts a new IF chain with further ELSE and/or ELSEIF statements as required.</p> <p>One ENDIF is used to terminate the multiline IF.</p>
<p>INC var [,increment]</p>	<p>Increments the variable "var" by either 1 or, if specified, the value in increment. "increment" can have a negative. This is functionally the same as var = var + increment but is processed much faster</p>
<p>INPUT ["prompt\$";] var1 [,var2 [, var3 [, etc]]]</p>	<p>Will take a list of values separated by commas (,) entered at the console and will assign them to a sequential list of variables.</p> <p>For example, if the command is: INPUT a, b, c And the following is typed on the keyboard: 23, 87, 66 Then a = 23 and b = 87 and c = 66</p> <p>The list of variables can be a mix of float, integer or string variables. The values entered at the console must correspond to the type of variable.</p> <p>If a single value is entered a comma is not required (however that value cannot contain a comma).</p> <p>'prompt\$' is a string constant (not a variable or expression) and if specified it will be printed first. Normally the prompt is terminated with a semicolon (;) and in that case a question mark will be printed following the prompt. If the prompt is terminated with a comma (,) rather than the semicolon (;) the question mark will be suppressed.</p>
<p>INPUT #nbr, list of variables</p>	<p>Same as above except that the input is read from a serial port or file previously opened for INPUT as 'nbr'. See the OPEN command.</p>
<p>INTERRUPT [myint]</p>	<p>This command triggers a software interrupt. The interrupt is set up using INTERRUPT 'myint' where 'myint' is the name of a subroutine that will be executed when the interrupt is triggered.</p> <p>Use INTERRUPT 0 to disable the interrupt</p> <p>Use INTERRUPT without parameters to trigger the interrupt.</p> <p>NB: the interrupt can also be triggered from within a CSUB</p>
<p>IR dev, key , int or IR CLOSE</p>	<p>Decodes NEC or Sony infrared remote control signals.</p> <p>An IR Receiver Module is used to sense the IR light and demodulate the signal. It can be connected to any pin however this pin must be configured in advanced using the command: SETPIN n, IR</p> <p>The IR signal decode is done in the background and the program will continue after this command without interruption. 'dev' and 'key' should be numeric variables and their values will be updated whenever a new signal is received ('dev' is the device code transmitted by the remote and 'key' is the key pressed). 'int' is a user defined subroutine that will be called when a new key press is received or when the existing key is held down for auto repeat. In the interrupt</p>

<p>如果 expr 那么 stmt [: stmt] 或 如果 expr 那么 stmt 否则 stmt</p>	<p>求值表达式 'expr'，如果为真则执行 THEN 关键字后面的语句，如果为假则跳到下一行。如果行上还有更多语句（用冒号（:）分隔），如果为真也会执行，如果为假则跳过。ELSE 关键字是可选的，如果存在，则在 'expr' 解析为假时将执行其后面的语句。</p> <p>‘THEN 语句’结构也可以替换为： 转到 行号 标签。 这种类型的 IF 语句全部在一行上。</p>
<p>如果 表达式 那么 <语句> [否则 如果 表达式 那么 <语句>] [否则 <语句>] 结束</p>	<p>多行 IF 语句，带有可选的 ELSE 和 ELSEIF 案例，并以结束 结束。每个组件都在单独的一行上。</p> <p>求值 '表达式'，如果表达式为真，则执行 THEN 后面的语句，或者如果为假，则可选地执行 ELSE 语句后面的语句。如果前一个条件为假，则执行 ELSEIF 语句（如果存在），并根据需要开始一个新的 IF 链，后面可以有更多的 ELSE 和/或 ELSEIF 语句。</p> <p>一个 结束 用于终止多行 IF。</p>
<p>增量变量 [, 增量]</p>	<p>将变量 “变量” 增加 1，或者如果指定，则增加增量中的值。“增量” 可以是负数。这在功能上与变量 = 变量 + 增量相同，但处理速度更快。</p>
<p>输入["提示\$"]; 变量1 [, 变量2 [, 变量3 [, 等等]]]]</p>	<p>将接受在控制台输入的以逗号（,）分隔的值，并将它们分配给一系列顺序变量。</p> <p>例如，如果命令是：INPUT a, b, c 而在键盘上输入以下内容：23, 87, 66 那么 a = 23, b = 87, c = 66 变量列表可以是浮点数、整数或字符串变量的混合。在控制台输入的值必须与变量的类型相对应。 如果只输入一个值，则不需要逗号（但该值不能包含逗号）。</p> <p>‘prompt\$’ 是一个字符串常量（不是变量或表达式），如果指定，它将首先被打印。通常提示以分号（;）结束，在这种情况下，提示后将打印一个问号。如果提示以逗号（,）而不是分号（;）结束，则问号将被抑制。</p>
<p>输入 #nbr, 变量列表</p>	<p>与上述相同，只是输入是从之前以‘nbr’打开的串行端口或文件中读取的。请参见打开命令。</p>
<p>中断 [myint]</p>	<p>此命令触发软件中断。中断是通过使用中断 ‘myint’ 设置的，其中 ‘myint’ 是将在中断触发时执行的子程序的名称。</p> <p>使用中断 0 禁用中断 使用中断不带参数触发中断。 注意：中断也可以在 CSUB 内部触发</p>
<p>红外dev, key , int 或 红外 关闭</p>	<p>解码 NEC 或索尼红外遥控信号。 红外接收模块用于感应红外光并解调信号。它可以连接到任何引脚，但该引脚必须提前使用命令配置：SETPIN n, IR</p> <p>红外信号解码在后台进行，程序将在此命令后继续执行而不受干扰。‘dev’ 和 ‘key’ 应该是数字变量，它们的值将在接收到新信号时更新（‘dev’ 是遥控器传输的设备代码，‘key’ 是按下的键）。</p> <p>‘int’ 是一个用户定义的子程序，当接收到新的按键按下事件或现有按键被按住以进行自动重复时将被调用。在中断</p>

	<p>subroutine the program can examine the variables 'dev' and 'key' and take appropriate action.</p> <p>The IR CLOSE command will terminate the IR decoder.</p> <p>Note that for the NEC protocol the bits in 'dev' and 'key' are reversed. For example, in 'key' bit 0 should be bit 7, bit 1 should be bit 6, etc. This does not affect normal use but if you are looking for a specific numerical code provided by a manufacturer you should reverse the bits. This describes how to do it: http://www.thebackshed.com/forum/forum_posts.asp?TID=8367</p> <p>See the section <i>Special Hardware Devices</i> for more details.</p>
IR SEND pin, dev, key	<p>Generate a 12-bit Sony Remote Control protocol infrared signal.</p> <p>'pin' is the I/O pin to use. This can be any I/O pin which will be automatically configured as an output and should be connected to an infrared LED. Idle is low with high levels indicating when the LED should be turned on.</p> <p>'dev' is the device being controlled and is a number from 0 to 31, 'key' is the simulated key press and is a number from 0 to 127.</p> <p>The IR signal is modulated at about 38KHz and sending the signal takes about 25mS during which program execution is paused.</p>
KEYPAD var, int, r1, r2, r3, r4, c1, c2, c3 [, c4] or KEYPAD CLOSE	<p>Monitor and decode key presses on a 4x3 or 4x4 keypad.</p> <p>Monitoring of the keypad is done in the background and the program will continue after this command without interruption. 'var' should be a numeric variable and its value will be updated whenever a key press is detected.</p> <p>'int' is a user defined subroutine that will be called when a new key press is received. In the interrupt subroutine the program can examine the variable 'var' and take appropriate action.</p> <p>r1, r2, r3 and r4 are pin numbers used for the four row connections to the keypad and c1, c2, c3 and c4 are the column connections. c4 is optional and is only used with 4x4 keypads. This command will automatically configure these pins as required.</p> <p>On a key press the value assigned to 'var' is the number of a numeric key (e.g. '6' will return 6) or 10 for the * key and 11 for the # key. On 4x4 keypads the number 20 will be returned for A, 21 for B, 22 for C and 23 for D.</p> <p>The KEYPAD CLOSE command will terminate the keypad function and return the I/O pin to a not configured state.</p> <p>See the section <i>Special Hardware Devices</i> for more details.</p>
KILL file\$ [,all]	<p>Deletes the file specified by 'file\$'. Any extension must be specified.</p> <p>Bulk erase is triggered if fname\$ contains a '*' or a '?' character</p> <p>If the optional 'all' parameter is used then you will be prompted for a single confirmation. If 'all' is not specified you will be prompted on each file.</p>
LET variable = expression	<p>Assigns the value of 'expression' to the variable. LET is automatically assumed if a statement does not start with a command. For example:</p> <pre>Var = 56</pre>
LIBRARY SAVE or LIBRARY DELETE or LIBRARY LIST Or LIBRARY LIST ALL Or LIBRARY DISK SAVE fname\$	<p>The library is a special segment of program memory that can contain program code such as subroutines, functions and CFunctions. These routines are not visible to the programmer but are available to any program running on the Micromite and act the same as built in commands and functions in MMBasic.</p> <p>Any code in the library that is not contained within a subroutine or function will be executed immediately before a program is run. This can be used to initialise constants, set options, etc. See the heading "The Library" in this manual for a full explanation.</p> <p>The library is stored in program memory Flash Slot 3 which will then not be available for saving a program (slots 1 to 2 will still be available).</p> <p>LIBRARY SAVE will take whatever is in normal program memory, compress it (remove redundant data such as comments) and append it to the library area</p>

	<p>子程序中，程序可以检查变量 'dev' 和 'key' 并采取适当的行动。</p> <p>IR CLOSE 命令将终止 IR 解码器。</p> <p>请注意，对于 NEC 协议，'dev' 和 'key' 中的位是反向的。例如，在 'key' 中，位 0 应该是位 7，位 1 应该是位 6，等等。这不会影响正常使用，但如果您在寻找制造商提供的特定数字代码，则应反转位。这描述了如何做到这一点：</p> <p>http://www.thebackshed.com/forum/forum_posts.asp?TID=8367</p> <p>请参阅部分特殊硬件设备以获取更多详细信息。</p>
IR 发送引脚, 设备, 键	<p>生成一个12位索尼遥控协议红外信号。'引脚'是要使用的输入/输出引脚。这可以是任何输入/输出引脚，系统会自动将其配置为输出，并应连接到红外LED。空闲时为低电平，高电平表示LED应开启。'设备'是被控制的设备，范围为0到31，'键'是模拟的按键，范围为0到127。</p> <p>红外信号以约38KHz调制，发送信号大约需要25毫秒，在此期间程序执行会暂停。</p>
键盘变量, 整数, r1, r2, r3, r4, c1, c2, c3 [, c4] 或 键盘关闭	<p>监控并解码4x3或4x4键盘上的按键。</p> <p>键盘的监控在后台进行，程序将在此命令后继续运行而不受干扰。'var' 应该是一个数字变量，当检测到按键时，其值将被更新。</p> <p>'int' 是一个用户定义的子程序，当接收到新的按键时将被调用。在中断子程序中，程序可以检查变量 'var' 并采取适当的行动。</p> <p>r1、r2、r3 和 r4 是用于连接键盘四行的引脚编号，c1、c2、c3 和 c4 是列连接。c4 是可选的，仅在4x4键盘中使用。此命令将自动配置这些引脚以满足要求。</p> <p>在按键时，分配给 'var' 的值是数字键的编号（例如，'6' 将返回 6），* 键返回 10，# 键返回 11。在4x4键盘上，A 返回 20，B 返回 21，C 返回 22，D 返回 23。KEYPAD CLOSE 命令将终止键盘功能，并将 I/O 引脚返回到未配置状态。</p> <p>请参阅章节特殊硬件设备以获取更多详细信息。</p>
KILL file\$ [,all]	<p>删除由 'file\$' 指定的文件。任何扩展名必须被指定。</p> <p>如果 fname\$ 包含 '*' 或 '?' 字符，则触发批量擦除。如果使用可选的 'all' 参数，则会提示您进行单次确认。如果未指定 'all'，则会对每个文件进行提示。</p>
LET 变量 = 表达式	<p>将 ‘表达式’ 的值赋给变量。如果语句不以命令开头，则自动假定为LET。例如：</p> <pre>Var = 56</pre>
LIBRARY SAVE 或 LIBRARY DELETE 或 LIBRARY LIST 或 LIBRARY LIST ALL 或 LIBRARY DISK SAVE fname\$	<p>库是程序内存的一个特殊段，可以包含程序代码，例如子例程、函数和 C 函数。这些例程对程序员不可见，但对在 Micromite 上运行的任何程序可用，并且与 MMBasic 中的内置命令和函数的作用相同。库中不包含在子例程或函数内的任何代码将在程序运行之前立即执行。这可用于初始化常量、设置选项等。有关详细说明，请参见本手册中的“库”标题。</p> <p>库存储在程序内存的闪存槽3中，这样将无法用于保存程序（槽1到2仍然可用）。</p> <p>LIBRARY SAVE 将会把正常程序内存中的内容压缩（去除冗余数据，如注释）并附加到库区域。</p>

Or LIBRARY DISK LOAD fname\$	(main program memory is then empty). The code in the library will not show in LIST or EDIT and will not be deleted when a new program is loaded or NEW is used. LIBRARY DELETE will remove the library and return Flash Slot 3 for normal use (OPTION RESET will do the same). LIBRARY LIST will list the contents of the library. Use ALL to list without page confirmations. LIBRARY DISK SAVE fname\$ will save the current library as a binary file allowing a subsequent call to LIBRARY DISK LOAD fname\$ to restore the library. Together, these allow libraries specific for individual programs to be stored and restored easily and distributed. Other than using version specific functionality in the library (WEB, VGA, GUI) libraries can be shared between versions.
LINE x1, y1, x2, y2 [, LW [, C]]	On an attached LCD display this command will draw a line starting at the coordinates ‘x1’ and ‘y1’ and ending at ‘x2’ and ‘y2’. ‘LW’ is the line’s width and is only valid for horizontal or vertical lines. It defaults to 1 if not specified or if the line is a diagonal. ‘C’ is an integer representing the colour and defaults to the current foreground colour. All parameters can be expressed as arrays and the software will plot the number of lines as determined by the dimensions of the smallest array. ‘x1’, ‘y1’, ‘x2’, and ‘y2’ must all be arrays or all be single variables /constants otherwise an error will be generated. ‘lw’ and ‘c’ can be either arrays or single variables/constants.
LINE AA x1, y1, x2, y2 [, LW [, C]]	Draws a line with anti-aliasing . The parameters are as per the LINE command above. However this version will use variable intensity values of the specified colour to reduce the “staggered” quality of diagonal lines. In addition this version can draw diagonal lines of any width. Note that it does not accept arrays as parameters.
LINE GRAPH x(),y(),colour	This command generates a line graph of the coordinate pairs specified in “x()” and “y()”. The graph will have n-1 segments where there are n elements in the x and y arrays.
LINE INPUT [prompt\$], string-variable\$	Reads an entire line from the console input into ‘string-variable\$’. ‘prompt\$’ is a string constant (not a variable or expression) and if specified it will be printed first. A question mark is not printed unless it is part of ‘prompt\$’. Unlike INPUT, this command will read a whole line, not stopping for comma delimited data items.
LINE INPUT #nbr, string-variable\$	Same as above except that the input is read from a serial communications port or a file previously opened for INPUT as ‘nbr’. See the OPEN command.
LINE PLOT ydata() [,nbr] [,xstart] [,xinc] [,ystart] [,yinc] [,colour]	Plots a line graph from an array of y-axis data points. ‘ydata’ is an array of floats or integers to be plotted ‘nbr’ is the number of line segments to be plotted - defaults to the lesser of the array size and MM.HRES-2 if omitted ‘xstart’ is the x-coordinate to start plotting - defaults to 0 ‘xinc’ is the increment along the x-axis to plot each coordinate - defaults to 1 ‘ystart’ is the location in ydata to start the plot - defaults to the array start ‘yinc’ is the increment in ydata to add for each point to be plotted ‘colour’ is the colour to draw the line
LIST [fname\$] or LIST ALL [fname\$]	List a program on the console. LIST on its own will list the program with a pause at every screen full. LIST ALL will list the program without pauses. This is useful if you wish to transfer the program to a terminal emulator on a PC that has the ability to capture its input stream to a file. If the optional ‘fname\$’ is specified then that

或 LIBRARY DISK LOAD fname\$	(主程序内存随后为空)。库中的代码不会在LIST或EDIT中显示，并且在加载新程序或使用NEW时不会被删除。 LIBRARY DELETE将删除库并将闪存槽3返回正常使用 (OPTION REST也会执行相同操作)。 LIBRARY LIST将列出库的内容。使用ALL可以在不进行页面确认的情况下列出。 LIBRARY DISK SAVE fname\$ 将当前库保存为二进制文件，允许后续调用 LIBRARY DISK LOAD fname\$ 来恢复库。这些功能结合在一起，使得特定于单个程序的库能够轻松存储、恢复和分发。除了使用库中的版本特定功能 (WEB、VGA、GUI) 外，库可以在不同版本之间共享。
LINE x1, y1, x2, y2 [, LW [, C]]	在连接的液晶显示器上，此命令将绘制一条从坐标‘x1’和‘y1’开始，到‘x2’和‘y2’结束的线。 ‘LW’是线的宽度，仅对水平或垂直线有效。如果未指定或线为对角线，则默认为1。‘C’是表示颜色的整数，默认为当前前景颜色。 所有参数都可以表示为数组，软件将根据最小数组的尺寸绘制相应数量的线。'x1','y1','x2' 和 'y2' 必须全部为数组，或全部为单个变量/常量否则将生成错误。'lw' 和 'c' 可以是数组或单个变量/常量。
LINE AA x1, y1, x2, y2 [, LW [, C]]	绘制一条带有抗锯齿的线。参数与上述的 LINE 命令相同。然而，这个版本将使用指定颜色的变量强度值，以减少对角线的“锯齿”质量。此外，这个版本可以绘制任意宽度的对角线。请注意，它不接受数组作为参数。
LINE GRAPH x(),y(),colour	此命令生成由“x()”和“y()”中指定的坐标对构成的折线图。图形将有 n-1 段，其中 x 和 y 数组中有 n 个元素。
LINE INPUT [prompt\$,] string-variable\$	从控制台输入中读取整行到 ‘string-variable\$’。 ‘prompt\$’ 是一个字符串常量 (不是变量或表达式)，如果指定，它将首先被打印。问号不会被打印，除非它是 ‘prompt\$’ 的一部分。 与 INPUT 不同，此命令将读取整行，而不是在逗号分隔的数据项处停止。
LINE INPUT #nbr, string-variable\$	与上述相同，只是输入是从串行通信端口或之前以 'nbr' 打开的文件中读取的。请参见 OPEN 命令。
LINE PLOT ydata() [,nbr] [,xstart] [,xinc] [,ystart] [,yinc] [,颜色]	从 y 轴数据点的数组绘制折线图。 ‘ydata’是要绘制的浮点数或整数数组。 ‘编号’是要绘制的线段数量 - 如果省略，则默认为数组大小和 MM.HRES-2 中的较小值。 ‘xstart’是开始绘制的 x 坐标 - 默认为 0。 ‘xinc’是沿 x 轴绘制每个坐标的增量 - 默认为 1。 ‘ystart’是在 ydata 中开始绘制的位置 - 默认为数组起始位置。 ‘yinc’是在 ydata 中为每个要绘制的点添加的增量。 ‘‘颜色’是绘制线条的颜色
LIST [fname\$] 或 LIST ALL [fname\$]	在控制台上列出一个程序。 单独使用 LIST 将在每个屏幕满时暂停列出程序。 LIST ALL 将不暂停地列出程序。如果您希望将程序传输到能够将其输入流捕获到文件的 PC 上的终端仿真器，这将非常有用。如果指定了可选的 ‘fname\$’，那么

	file on the Flash Filesystem or SD Card will be listed.
LIST COMMANDS or LIST FUNCTIONS	Lists all valid commands or functions
LOAD file\$ [,R]	Loads a program called 'file\$' from the Flash Filesystem or SD Card into program memory. If the optional suffix ,R is added the program will be immediately run without prompting (in this case 'file\$' must be a string constant). If an extension is not specified ".BAS" will be added to the file name.
LOAD IMAGE file\$ [, x] [, y]	Load a bitmapped image from the Flash Filesystem or SD Card and display it on the LCD panel. "file\$" is the name of the file and 'x' and 'y' are the screen coordinates for the top left hand corner of the image. If the coordinates are not specified the image will be drawn at the top left hand position on the screen. If an extension is not specified ".BMP" will be added to the file name. All types of the BMP format are supported including black and white and true colour 24-bit images.
LOAD JPG file\$ [, x] [, y]	Load a jpg image from the Flash Filesystem or SD Card and display it on the LCD panel. "file\$" is the name of the file and 'x' and 'y' are the screen coordinates for the top left hand corner of the image. If the coordinates are not specified the image will be drawn at the top left hand position on the screen. If an extension is not specified ".JPG" will be added to the file name. Progressive jpg images are not supported.
LOCAL variable [, variables] See DIM for the full syntax.	Defines a list of variable names as local to the subroutine or function. This command uses exactly the same syntax as DIM and will create variables that will only be visible within the subroutine or function. They will be automatically discarded when the subroutine or function exits.
LONGSTRING	The LONGSTRING commands allow for the manipulation of strings longer than the normal MMBasic limit of 255 characters. Variables for holding long strings must be defined as single dimensioned integer arrays with the number of elements set to the number of characters required for the maximum string length divided by eight. The reason for dividing by eight is that each integer in an MMBasic array occupies eight bytes. Note that the long string routines do not check for overflow in the length of the strings. If an attempt is made to create a string longer than a long string variable's size the outcome will be undefined.
LONGSTRING APPEND array%(), string\$	Append a normal MMBasic string to a long string variable. array%() is a long string variable while string\$ is a normal MMBasic string expression.
LONGSTRING CLEAR array%()	Will clear the long string variable array%(). i.e. it will be set to an empty string.
LONGSTRING COPY dest%(), src%()	Copy one long string to another. dest%() is the destination variable and src%() is the source variable. Whatever was in dest%() will be overwritten.
LONGSTRING CONCAT dest%(), src%()	Concatenate one long string to another. dest%() is the destination variable and src%() is the source variable. src%() will be added to the end of dest%() (the destination will not be overwritten).
LONGSTRING LCASE array%()	Will convert any uppercase characters in array%() to lowercase. array%() must be long string variable.
LONGSTRING LEFT dest%(), src%(), nbr	Will copy the left hand 'nbr' characters from src%() to dest%() overwriting whatever was in dest%(). i.e. copy from the beginning of src%(). src%() and dest%() must be long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING LOAD array%(), nbr, string\$	Will copy 'nbr' characters from string\$ to the long string variable array%() overwriting whatever was in array%().

	Flash 文件系统或 SD 卡上的文件将被列出。
LIST 命令 或 LIST 函数	列出所有有效的命令或函数
LOAD file\$ [,R]	从 Flash 文件系统或 SD 卡加载名为 ‘file\$’ 的程序到程序内存中。如果添加了可选后缀 ,R，程序将立即运行而不提示（在这种情况下 ‘file\$’ 必须是字符串常量）。 如果未指定扩展名，将会在文件名后添加 “.BAS”。
加载图像file\$ [, x] [, y]	从 Flash 文件系统或 SD 卡加载位图图像并在液晶面板上显示。“file” 是文件的名称，‘x’ 和 ‘y’ 是图像左上角的屏幕坐标。如果未指定坐标，图像将绘制在屏幕的左上角位置。 如果未指定扩展名，将在文件名后添加“.BMP”。 支持所有类型的 BMP 格式，包括黑白和真彩色 24 位图像。
加载 JPGfile\$ [, x] [, y]	从 Flash 文件系统或 SD 卡加载 JPG 图像并在液晶面板上显示。“file” 是文件的名称，‘x’ 和 ‘y’ 是图像左上角的屏幕坐标。如果未指定坐标，图像将绘制在屏幕的左上角位置。 如果未指定扩展名，将在文件名后添加“.JPG”。 不支持渐进式 JPG 图像。
局部变量 [, 变量] 有关完整语法，请参见 DIM。	将变量名称列表定义为子程序或函数的局部变量。此命令的语法与DIM完全相同，将创建仅在子程序或函数内可见的变量。当子程序或函数退出时，这些变量将自动被丢弃。
长字符串	LONGSTRING 命令允许操作超过正常MMBasic限制（255个字符）的字符串。 用于保存长字符串的变量必须定义为单维整数数组，元素数量设置为所需的最大字符串长度除以八的字符数。除以八的原因是MMBasic数组中的每个整数占用八个字节。请注意，长字符串例程不会检查字符串长度的溢出。如果尝试创建一个超过长字符串变量大小的字符串，结果将是未定义的。
LONGSTRING APPEND array%(), string\$	将一个普通的MMBasic字符串附加到一个长字符串变量。array%() 是一个长字符串变量，而 string\$ 是一个普通的 MMBasic 字符串表达式。
LONGSTRING CLEAR array%()	将清除长字符串变量 array%()，即它将被设置为空字符串。
LONGSTRING COPY dest%(), src%()	将一个长字符串复制到另一个长字符串。dest%() 是目标变量，src%() 是源变量。dest%() 中的内容将被覆盖。
LONGSTRING CONCAT dest%(), src%()	将一个长字符串连接到另一个长字符串。dest%() 是目标变量，src%() 是源变量。src%() 将被添加到 dest%() 的末尾（目标不会被覆盖）。
LONGSTRING LCASE array%()	将 array%() 中的任何大写字符转换为小写。array%() 必须是长字符串变量。
LONGSTRING LEFT dest%(), src%(), nbr	将从 src%() 中复制左侧的 ‘nbr’ 个字符到 dest%()，覆盖 dest%() 中的内容。即从 src%() 的开头开始复制。src%() 和 dest%() 必须是长字符串变量。‘nbr’ 必须是一个整数常量或表达式。
LONGSTRING LOAD array%(), nbr, string\$	将从 string\$ 复制 ‘nbr’ 个字符到长字符串变量 array%()，覆盖 array%() 中的内容。

LONGSTRING MID dest%, src%, start, nbr	Will copy 'nbr' characters from src%() to dest%() starting at character position 'start' overwriting whatever was in dest%(). i.e. copy from the middle of src%(). 'nbr' is optional and if omitted the characters from 'start' to the end of the string will be copied src%() and dest%() must be long string variables. 'start' and 'nbr' must be an integer constants or expressions.
LONGSTRING PRINT [#n,] src%	Prints the longstring stored in 'src%' to the file or COM port opened as '#n'. If '#n' is not specified the output will be sent to the console.
LONGSTRING REPLACE array%, string\$, start	Will substitute characters in the normal MMBasic string string\$ into an existing long string array%() starting at position 'start' in the long string.
LONGSTRING RESIZE addr%, nbr	Sets the size of the longstring to nbr. This overrides the size set by other longstring commands so should be used with caution. Typical use would be in using a longstring as a byte array.
LONGSTRING RIGHT dest%, src%, nbr	Will copy the right hand 'nbr' characters from src%() to dest%() overwriting whatever was in dest%. i.e. copy from the end of src%. src%() and dest%() must be long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING SETBYTE addr%, nbr, data	Sets byte nbr to the value "data", nbr respects OPTION BASE
LONGSTRING TRIM array%, nbr	Will trim 'nbr' characters from the left of a long string. array%() must be a long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING UCASE array%	Will convert any lowercase characters in array%() to uppercase. array%() must be long string variable.
LOOP [UNTIL expression]	Terminates a program loop: see DO.
MATH	The math command performs many simple mathematical calculations that can be programmed in BASIC but there are speed advantages to coding looping structures in the firmware and there is the advantage that once debugged they are there for everyone without re-inventing the wheel. Note: 2 dimensional maths matrices are always specified DIM matrix(n_columns, n_rows) and of course the dimensions respect OPTION BASE. Quaternions are stored as a 5 element array w, x, y, z, magnitude.
MATH RANDOMIZE [n]	Seeds the Mersenne Twister algorithm. If n is not specified the seed is the time in microseconds since boot The Mersenne Twister algorithm gives a much better random number than the C-library inbuilt function
Simple array arithmetic	
MATH SET nbr, array()	Sets all elements in array() to the value nbr. Note this is the fastest way of clearing an array by setting it to zero.
MATH SCALE in(), scale ,out()	This scales the matrix in() by the scalar scale and puts the answer in out(). Works for arrays of any dimensionality of both integer and float and can convert between. Setting b to 1 is optimised and is the fastest way of copying an entire array.
MATH ADD in(), num ,out()	This adds the value 'num' to every element of the matrix in() and puts the answer in out(). Works for arrays of any dimensionality of both integer and float and can convert between. Setting num to 0 is optimised and is a fast way of copying an entire array. in() and out() can be the same array.
MATH INTERPOLATE in1(), in2(), ratio, out()	This command implements the following equation on every array element: out = (in2 - in1) * ratio + in1 Arrays can have any number of dimensions and must be distinct and have the

LONGSTRING MID dest%, src%, start, nbr	将从 src%() 复制 'nbr' 个字符到 dest%(), 从字符位置 'start' 开始, 覆盖 dest%() 中的内容。即从 src%() 的中间复制。'nbr' 是可选的, 如果省略, 将从 'start' 到字符串的末尾复制字符, src%() 和 dest%() 必须是长字符串变量。 'start' 和 'nbr' 必须是整数常量或表达式。
LONGSTRING PRINT [#n,] src%	将存储在 'src%' 中的长字符串打印到作为 '#n' 打开的文件或 COM 端口。 如果未指定 '#n', 输出将发送到控制台。
LONGSTRING 替换 array%, string\$, start	将普通MMBasic字符串string\$中的字符替换到现有的长字符串array%()中, 从长字符串的'start'位置开始。
LONGSTRING 调整大小 addr%, nbr	将长字符串的大小设置为nbr。这将覆盖其他长字符串命令设置的大小, 因此应谨慎使用。典型用法是在将长字符串用作字节数组时。
LONGSTRING 右侧 dest%, src%, nbr	将src%()中的右侧 'nbr' 个字符复制到dest%(), 覆盖 dest%()中原有的内容。即从src%()的末尾复制。src%()和dest%()必须是长字符串变量。'nbr'必须是整数常量或表达式。
LONGSTRING 设置字节 addr%, nbr, data	将字节 nbr 设置为值“data”， nbr 遵循 OPTION BASE
LONGSTRING TRIM array%, nbr	将从长字符串的左侧修剪 'nbr' 个字符。array%() 必须是长字符串变量。'nbr'必须是整数常量或表达式。
LONGSTRING UCASE array%	将 array%() 中的任何小写字符转换为大写。array%() 必须是长字符串变量。
LOOP [直到 expression]	终止程序循环：参见 DO。
数学	数学命令执行许多简单的数学计算, 这些计算可以用 BASIC 编程, 但在固件中编码循环结构有速度优势, 并且一旦调试完成, 它们就可以供所有人使用, 而无需重新发明轮子。注意：二维数学矩阵总是指定为 DIM matrix(n_columns, n_rows), 当然, 尺寸遵循 OPTION BASE 。四元数存储为 5 元素数组 w, x, y, z, magnitude。
MATH RANDOMIZE [n]	为梅森旋转算法设置种子。 如果未指定 n, 则种子为自启动以来的微秒时间。梅森旋转算法生成的随机数比 C 库内置函数要好得多。
简单数组算术	
MATH SET nbr, array()	将数组 array() 中的所有元素设置为值 nbr。请注意, 这种方法是将数组清零的最快方式。
MATH SCALE in(), scale ,out()	将矩阵 in() 按标量 scale 进行缩放, 并将结果放入 out() 中。 适用于任何维度的整数和浮点数组, 并且可以进行转换。将 b 设置为 1 是经过优化的, 这是复制整个数组的最快方法。
MATH ADD in(), num ,out()	将值 'num' 添加到矩阵 in() 的每个元素, 并将结果放入 out() 中。适用于任何维度的整数和浮点数组, 并且可以进行转换。将 num 设置为 0 是经过优化的, 这是复制整个数组的快速方法。in() 和 out() 可以是相同的数组。
MATH INTERPOLATE in1(), in2(), ratio, out()	此命令在每个数组元素上实现以下方程： $\text{out} = (\text{in2} - \text{in1}) * \text{ratio} + \text{in1}$ 数组可以具有任意数量的维度, 并且必须是不同的, 并且具有

	<p>same number of total elements. The command works with both integer and floating point arrays in any mixture</p>
MATH WINDOW in(), minout, maxout, out() [,minin, maxin]	<p>This command takes the “in” array and scales it between “minout” and “maxout” returning the answer in “out”. Optionally, it can also return the minimum and maximum values found in the original data (“minin” and “minout”).</p> <p>Note: “minout” can be greater than “maxout” and in this case the data will be both scaled and inverted.</p> <p>e.g</p> <pre>DIM IN(2)=(1,2,3) DIM OUT(2) MATH WINDOW IN(),7,3,OUT(),LOW,HIGH Will return OUT(0)=7, OUT(1)=5,OUT(2)=3,LOW=1,HIGH=3 This command can massively simplify scaling data for plotting etc.</pre>
MATH SLICE sourcearray(), [d1] [,d2] [,d3] [,d4] [,d5], destinationarray()	<p>This command copies a specified set of values from a multi-dimensional array into a single dimensional array. It is much faster than using a FOR loop. The slice is specified by giving a value for all but one of the source array indices and there should be as many indices in the command, including the blank one, as there are dimensions in the source array</p> <p>e.g.</p> <pre>OPTION BASE 1 DIM a(3,4,5) DIM b(4) MATH SLICE a(), 2, , 3, b() Will copy the elements 2,1,3 and 2,2,3 and 2,3,3 and 2,4,3 into array b()</pre>
MATH INSERT targetarray(), [d1] [,d2] [,d3] [,d4] [,d5], sourcearray()	<p>This is the opposite of MATH SLICE, has a very similar syntax, and allows you, for example, to substitute a single vector into an array of vectors with a single instruction</p> <p>e.g.</p> <pre>OPTION BASE 1 DIM targetarray(3,4,5) DIM sourcearray(4)=(1,2,3,4) MATH INSERT targetarray(), 2, , 3, sourcearray() Will set elements 2,1,3 = 1 and 2,2,3 = 2 and 2,3,3 = 3 and 2,4,3 = 4</pre>
MATH SHIFT inarray%(), nbr, outarray%() [,U]	<p>This command does a bit shift on all elements of inarray%() and places the result in outarray%() (may be the same as inarray%()). nbr can be between -63 and 63. Positive numbers are a left shift (multiply by power of 2). Negative number are a right shift. The optional parameter ,U will force an unsigned shift.</p>
Matrix arithmetic	
MATH M_INVERSE array!(), inversearray!()	<p>This returns the inverse of array!() in inversearray!(). The array must be square and you will get an error if the array cannot be inverted (determinant=0). array!() and inversearray!() cannot be the same.</p>
MATH M_PRINT array()	<p>Quick mechanism to print a 2D matrix one row per line.</p>
MATH M_TRANSPOSE in(), out()	<p>Transpose matrix in() and put the answer in matrix out(), both arrays must be 2D but need not be square. If not square then the arrays must be dimensioned in(m,n) out(n,m)</p>

	相同数量的总元素。该命令适用于任何混合的整数和浮点数组
数学窗口 in(), minout, maxout, out() [,minin, maxin]	<p>此命令将“in”数组缩放到“minout”和“maxout”之间，并将答案返回到“out”。可选地，它还可以返回原始数据中找到的最小值和最大值（“minin”和“minout”）。</p> <p>注意：“minout”可以大于“maxout”，在这种情况下，数据将同时被缩放和反转。</p> <p>例如</p> <pre>DIM IN(2)=(1,2,3) DIM OUT(2) MATH WINDOW IN(),7,3,OUT(),LOW,HIGH 将返回 OUT(0)=7, OUT(1)=5, OUT(2)=3, LOW=1, HIGH=3 此命令可以大大简化数据的缩放以便绘图等。</pre>
数学切片 sourcearray(), [d1] [,d2] [,d3] [,d4] [,d5], destinationarray()	<p>此命令将指定的一组值从多维数组复制到一维数组中。它比使用 FOR 循环要快得多。通过为源数组索引中的所有但一个提供值来指定切片，并且命令中应包含与源数组的维度数量相同的索引，包括空白的那个，例如。</p> <p>基数选项 1 维度 a(3, 4, 5) 维度 b(4) 数学 切片 a(), 2, , 3, b() 将复制元素 2,1,3 和 2,2,3 和 2,3,3 和 2,4,3 到数组 b()</p>
数学插入 targetarray(), [d1] [,d2] [,d3] [,d4] [,d5], sourcearray()	<p>这是数学切片的相反操作，语法非常相似，并允许您例如用单个指令将单个向量替换为向量数组，例如：</p> <p>基数选项 1 维度 目标数组(3, 4, 5) 维度 源数组(4)=(1, 2, 3, 4) 数学 插入 目标数组(), 2, , 3, 源数组() 将设置元素 2,1,3 = 1 和 2,2,3 = 2 和 2,3,3 = 3 和 2,4,3 = 4</p>
数学移位 inarray%(), nbr, outarray%() [,U]	<p>此命令对 inarray%() 的所有元素进行位移，并将结果放入 outarray%()（可能与 inarray%() 相同）。编号可以在 -63 和 63 之间。正数是左移（乘以 2 的幂）。负数是右移。可选参数 ,U 将强制进行无符号移位。</p>
矩阵算术	
数学 M_INVERSE array!(), inversearray!()	<p>这将返回 array!() 的反向在 inversearray!() 中。数组必须是方阵。如果数组无法反转（行列式=0），您将收到错误。 array!() 和 inversearray!() 不能是相同的。</p>
数学 M_PRINT array()	快速机制以每行打印一个 2D 矩阵。
数学 M_TRANSPOSE in(), out()	转置矩阵 in() 并将答案放入矩阵 out()，两个数组必须是 2D，但不需要是方阵。如果不是方阵，则数组必须被定义为 in(m,n) out(n,m)

MATH M_MULT in1(), in2(), out()	Multiply the arrays in1() and in2() and put the answer in out(). All arrays must be 2D but need not be square. If not square then the arrays must be dimensioned in1(m,n) in2(p,m) ,out(p,n)
Vector arithmetic	
MATH V_PRINT array()	Quick mechanism to print a small array on a single line
MATH V_NORMALISE inV(), outV()	Converts a vector inV() to unit scale and puts the answer in outV() $(\text{sqr}(x*x + y*y + \dots))=1$ There is no limit on number of elements in the vector
MATH V_MULT matrix(), inV(), outV()	Multiplies matrix() and vector inV() returning vector outV(). The vectors and the 2D matrix can be any size but must have the same cardinality.
MATH V_CROSS inV1(), inV2(), outV()	Calculates the cross product of two three element vectors inV1() and inV2() and puts the answer in outV()
MATH V_ROTATE x, y, a, xin(), yin(), xout(), yout()	This command rotates the coordinate pairs in “xin()” and “yin()” around the centre point defined by “x” and “y” by the angle “a” and puts the results in “xout()” and “yout()”. NB: the input and output arrays can be the same and the rotation angle is, by default, in radians but this can be changed using the OPTION ANGLE command.
Quaternion arithmetic	
MATH Q_INVERT inQ(), outQ()	Invert the quaternion in inQ() and put the answer in outQ()
MATH Q_VECTOR x, y, z, outVQ()	Converts a vector specified by x , y, and z to a normalised quaternion vector outVQ() with the original magnitude stored
MATH Q_CREATE theta, x, y, z, outRQ()	Generates a normalised rotation quaternion outRQ() to rotate quaternion vectors around axis x,y,z by an angle of theta. Theta is specified in radians.
MATH Q_EULER yaw, pitch, roll, outRQ()	Generates a normalised rotation quaternion outRQ() to rotate quaternion vectors as defined by the yaw, pitch and roll angles With the vector in front of the “viewer” yaw is looking from the top of the vector and rotates clockwise, pitch rotates the top away from the camera and roll rotates around the z-axis clockwise. The yaw, pitch and roll angles default to radians but respect the setting of OPTION ANGLE
MATH Q_MULT inQ1(), inQ2(), outQ()	Multiplies two quaternions inQ1() and inQ2() and puts the answer in outQ()
MATH Q_ROTATE , RQ(), inVQ(), outVQ()	Rotates the source quaternion vector inVQ() by the rotate quaternion RQ() and puts the answer in outVQ()
MATH C_ADD array1%(), array2%(), array3%() MATH C_SUB array1%(), array2%(), array3%() MATH C_MUL array1%(), array2%(), array3%() MATH C_DIV array1%(), array2%(), array3%() MATH C_ADD array1!(), array2!(), array3!() MATH C_SUB array1!(), array2!(), array3!() MATH C_MUL array1!(), array2!(), array3!() MATH C_DIV array1!(), array2!(), array3!()	These commands do cell by cell operations (hence C_) on identically sized arrays. There are no restrictions on the number of dimensions and no restrictions on using the same array twice or even three times in the parameters. The datatype must be the same for all the arrays. e.g. MATH C_MULT a%(),a%(),a%() will square all the values in the array a%()

数学 M_MULT in1(), in2(), out()	将数组 in1() 和 in2() 相乘，并将结果放入 out()。所有数组必须是二维的，但不需要是方形的。如果不是方形，则数组必须定义为 in1(m,n) in2(p,m), out(p,n)
向量算术	
数学 V_PRINT array()	快速机制在单行上打印小数组
数学 V_NORMALISE inV(), outV()	将向量 inV() 转换为单位缩放，并将答案放入 outV() $(\text{sqr}(x*x + y*y + \dots))=1$ 向量中的元素数量没有限制
数学 V_MULT matrix(), inV(), outV()	将矩阵 matrix() 和向量 inV() 相乘，返回向量 outV()。向量和二维矩阵可以是任意大小，但必须具有相同的基数。
数学 V_CROSS inV1(), inV2(), outV()	计算两个三元素向量 inV1() 和 inV2() 的叉积，并将答案放入 outV()
数学 V_ROTATE x, y, a, xin(), yin(), xout(), yout()	此命令将“xin()”和“yin()”中的坐标对围绕由“x”和“y”定义的中心点按角度“a”旋转，并将结果放入“xout()”和“yout()”。注意：输入和输出数组可以相同，旋转角度默认以弧度为单位，但可以使用 OPTION ANGLE 命令进行更改。
四元数运算	
数学 Q_INVERT inQ(), outQ()	反转 inQ() 中的四元数，并将答案放入 outQ()
数学 Q_VECTOR x, y, z, outVQ()	将由 x、y 和 z 指定的向量转换为标准化的四元数向量 outVQ()，并存储原始大小
数学 Q_CREATE theta, x, y, z, outRQ()	生成一个标准化的旋转四元数 outRQ()，以在 x、y、z 轴上按角度 theta 旋转四元数向量。Theta 以弧度表示。
数学 Q_EULER yaw, pitch, roll, outRQ()	生成一个标准化的旋转四元数 outRQ()，以根据偏航、俯仰和滚动角度旋转四元数向量 在“观察者”面前，偏航是从向量的顶部看并顺时针旋转，俯仰是将顶部远离相机，滚动是围绕 z 轴顺时针旋转。
数学 Q_MULT inQ1(), inQ2(), outQ()	偏航、俯仰和滚动角度默认为弧度，但遵循 OPTION ANGLE 的设置
数学 Q_ROTATE , RQ(), inVQ(), outVQ()	将两个四元数 inQ1() 和 inQ2() 相乘，并将结果放入 outQ()
通过旋转四元数 RQ() 旋转源四元数向量 inVQ()，并将结果放入 outVQ()	通过旋转四元数 RQ() 旋转源四元数向量 inVQ()，并将结果放入 outVQ()
MATH C_ADD array1%(), array2%(), array3%() MATH C_SUB array1%(), array2%(), array3%() MATH C_MUL array1%(), array2%(), array3%() MATH C_DIV array1%(), array2%(), array3%() MATH C_ADD array1!(), array2!(), array3!() MATH C_SUB array1!(), array2!(), array3!() MATH C_MUL array1!(), array2!(), array3!() MATH C_DIV array1!(), array2!(), array3!()	这些命令对相同尺寸的数组进行逐个单元的操作（因此称为 C_）。对维度的数量没有限制，也没有限制在参数中使用同一个数组两次或三次。 所有数组的数据类型必须相同。 例如。 MATH C_MULT a%(),a%(),a%() 将对数组 a%() 中的所有值进行平方运算。

MATH FFT signalarray!(), FFTarray!()	<p>Performs a fast fourier transform of the data in "signalarray!". "signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero)</p> <p>"FFTarray" must be floating point and have dimension 2*N where N is the same as the signal array (e.g. f(1,1023) assuming OPTION BASE is zero)</p> <p>The command will return the FFT as complex numbers with the real part in f(0,n) and the imaginary part in f(1,n)</p>
MATH FFT INVERSE FFTarray!(), signalarray!()	<p>Performs an inverse fast fourier transform of the data in "FFTarray!".</p> <p>"FFTarray" must be floating point and have dimension 2*N where N must be a power of 2 (e.g. f(1,1023) assuming OPTION BASE is zero) with the real part in f(0,n) and the imaginary part in f(1,n).</p> <p>"signalarray" must be floating point and the single dimension must be the same as the FFT array.</p> <p>The command will return the real part of the inverse transform in "signalarray".</p>
MATH FFT MAGNITUDE signalarray!(), magnitudearray!()	<p>Generates magnitudes for frequencies for the data in "signalarray!"</p> <p>"signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero)</p> <p>"magnitudearray" must be floating point and the size must be the same as the signal array</p> <p>The command will return the magnitude of the signal at various frequencies according to the formula:</p> <p>frequency at array position N = $N * \text{sample_frequency} / \text{number_of_samples}$</p>
MATH FFT PHASE signalarray!(), phasearray!()	<p>Generates phases for frequencies for the data in "signalarray!".</p> <p>"signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero). "phasearray" must be floating point and the size must be the same as the signal array</p> <p>The command will return the phase angle of the signal at various frequencies according to the formula above.</p>
MATH SENSORFUSION type ax, ay, az, gx, gy, gz, mx, my, mz, pitch, roll, yaw [,p1] [,p2]	<p>Type can be MAHONY or MADGWICK</p> <p>Ax, ay, and az are the accelerations in the three directions and should be specified in units of standard gravitational acceleration.</p> <p>Gx, gy, and gz are the instantaneous values of rotational speed which should be specified in radians per second.</p> <p>Mx, my, and mz are the magnetic fields in the three directions and should be specified in nano-Tesla (nT)</p> <p>Care must be taken to ensure that the x, y and z components are consistent between the three inputs. So, for example, using the MPU-9250 the correct input will be ax, ay, az, gx, gy, gz, my, mx, -mz based on the reading from the sensor.</p> <p>Pitch, roll and yaw should be floating point variables and will contain the outputs from the sensor fusion.</p> <p>The SENSORFUSION routine will automatically measure the time between consecutive calls and will use this in its internal calculations.</p> <p>The Madwick algorithm takes an optional parameter p1. This is used as beta in the calculation. It defaults to 0.5 if not specified</p> <p>The Mahony algorithm takes two optional parameters p1, and p2. These are used as Kp and Ki in the calculation. If not specified these default to 10.0 and 0.0 respectively.</p> <p>A fully worked example of using the code is given on the BackShed forum at: https://www.thebackshed.com/forum/ViewTopic.php?TID=13459&PID=166962#166962</p>

MATH FFT signalarray!(), FFTarray!()	对“signalarray!”中的数据执行快速傅里叶变换。“signalarray”必须是浮点数，且大小必须是2的幂（例如，假设基数选项为零时 s(1023)） “FFTarray”必须是浮点数，且维度为 2*N，其中 N 与信号数组相同（例如，假设基数选项为零时 f(1,1023)）。该命令将返回 FFT 作为复数，实部在 f(0,n) 中，虚部在 f(1,n) 中。
MATH FFT 反向 FFTarray!(), signalarray!()	对“FFTarray!”中的数据执行反向快速傅里叶变换。 “FFTarray”必须是浮点数，且维度为 2*N，其中 N 必须是2的幂（例如，假设基数选项为零时 f(1,1023)），实部在 f(0,n) 中，虚部在 f(1,n) 中。 “signalarray”必须是浮点数，且单维度必须与 FFT 数组相同。 该命令将返回 “signalarray” 中反变换的实部。
MATH FFT MAGNITUDE signalarray!(),magnitudearray!()	为 “signalarray!” 中的数据生成频率的幅度。“signalarray”必须是浮点数，且大小必须是 2 的幂（例如，s(1023)，假设 OPTION BASE 为零）。 “magnitudearray”必须是浮点数，且大小必须与信号数组相同。 该命令将根据以下公式返回信号在不同频率下的幅度： 频率在数组位置 $N = N * \text{sample_frequency} / \text{number_of_samples}$
MATH FFT PHASE signalarray!(), phasearray!()	为 “signalarray!” 中的数据生成频率的相位。 “signalarray”必须是浮点数，且大小必须是 2 的幂（例如，s(1023)，假设 OPTION BASE 为零）。“phasearray”必须是浮点类型，并且大小必须与信号数组相同 该命令将根据上述公式返回信号在不同频率下的相位角。
MATH SENSORFUSION type ax, ay, az, gx, gy, gz, mx, my, mz, pitch, roll, yaw [,p1] [,p2]	类型可以是 MAHONY 或 MADGWICK Ax、ay 和 az 是三个方向上的加速度，应该以标准重力加速度的单位来指定。 Gx、gy 和 gz 是瞬时旋转速度的值，应该以每秒弧度为单位来指定。 Mx、my 和 mz 是三个方向上的磁场，应该以纳特斯拉 (nT) 为单位来指定。 必须确保 x、y 和 z 分量在三个输入之间是一致的。因此，例如，使用 MPU-9250，正确的输入将是 ax、ay、az、gx、gy、gz、my、mx、-mz，基于传感器的读数。 音高、滚动和偏航应为浮点变量，并将包含传感器融合的输出。 SENSORFUSION 例程将自动测量连续调用之间的时间，并将在其内部计算中使用此时间。 Madwick 算法接受一个可选参数 p1。它在计算中用作 beta。如果未指定，则默认为 0.5。 Mahony 算法接受两个可选参数 p1 和 p2。这些在计算中用作 Kp 和 Ki。如果未指定，这两个参数分别默认为 10.0 和 0.0。 在 BackShed 论坛上提供了使用该代码的完整示例： https://www.thebackshed.com/forum/ViewTopic.php?TID=13459&PID=166962#166962

MEMORY	<p>List the amount of memory currently in use. For example:</p> <p>Program:</p> <pre>OK (0%) Program (0 lines) 160K (100%) Free</pre> <p>RAM:</p> <pre>OK (0%) 0 Variables OK (0%) General 112K (100%) Free</pre> <p>Notes:</p> <ul style="list-style-type: none"> • Memory usage is rounded to the nearest 1K byte. • General memory is used by serial I/O buffers, etc.
MEMORY SET address, byte, numberofbytes	<p>This command will set a region of memory to a value.</p> <p>BYTE = One byte per memory address.</p>
MEMORY SET BYTE address, byte, numberofbytes	<p>SHORT = Two bytes per memory address.</p>
MEMORY SET SHORT address, short, numberofshorts	<p>WORD = Four bytes per memory address.</p>
MEMORY SET WORD address, word, numberofwords	<p>FLOAT = Eight bytes per memory address.</p> <p>‘increment’ is optional and controls the increment of the ‘address’ pointer as the operation is executed. For example, if increment=3 then only every third element of the target is set. The default is 1.</p>
MEMORY SET INTEGER address, integervalue ,numberofintegers [,increment]	
MEMORY SET FLOAT address, floatingvalue ,numberoffloats [,increment]	
MEMORY COPY sourceaddress, destinationaddress, numberofbytes	<p>This command will copy one region of memory to another.</p> <p>COPY INTEGER and FLOAT will copy eight bytes per operation.</p> <p>‘sourceincrement’ is optional and controls the increment of the ‘sourceaddress’ pointer as the operation is executed. For example, if sourceincrement=3 then only every third element of the source will be copied. The default is 1.</p> <p>‘destinationincrement’ is similar and operates on the ‘destinationaddress’ pointer.</p>
MEMORY COPY INTEGER sourceaddress, destinationaddress, numberofintegers ,[sourceincrement][,destination increment]	
MEMORY COPY FLOAT sourceaddress, destinationaddress, numberoffloats ,[sourceincrement][,destination increment]	

MEMORY	<p>列出当前使用的内存量。例如：</p> <p>程序：</p> <pre>OK (0%) 程序 (0行) 160K (100%) 空闲</pre> <p>RAM：</p> <pre>OK (0%) 0 变量 OK (0%) 一般 112K (100%) 空闲</pre> <p>注意：</p> <ul style="list-style-type: none"> • 内存使用量四舍五入到最近的1K字节。 • 通用内存用于串行I/O缓冲区等。
MEMORY SET 地址, 字节, 字节数	<p>此命令将内存区域设置为一个值。</p> <p>BYTE = 每个内存地址一个字节。</p> <p>SHORT = 每个内存地址两个字节。</p> <p>WORD = 每个内存地址四个字节。</p> <p>FLOAT = 每个内存地址八个字节。</p>
MEMORY SET BYTE 地址, 字节, 字节数	<p>‘增量’是可选的，控制在执行操作时‘地址’指针的增量。例如，如果增量=3，则仅设置目标的每第三个元素。默认值为1。</p>
MEMORY SET SHORT 地址, 短整型, 短整型数量	
MEMORY SET WORD 地址, 字, 字数量	
MEMORY SET INTEGER 地址, 整数值 , 整数数量 [,增量]	
MEMORY SET FLOAT 地址, 浮点值 , 浮点数量 [,增量]	
内存复制 源地址, 目标地址, 字节数	<p>此命令将一个内存区域复制到另一个区域。</p> <p>复制整数和浮点数每次操作将复制八个字节。</p> <p>‘源增量’是可选的，控制在执行操作时‘源地址’指针的增量。例如，如果源增量=3，则仅每第三个元素将被复制。默认值为1。</p>
内存复制 整数 源地址, 目 标地址, 整数 数量 [,源增量][, 目标增量]	<p>‘目标增量’类似，并作用于‘目标地址’指针。</p>
内存复制 浮点数 源地址, 目标地址, 浮 点数数量 [,源增 量][,目标增量]	

MEMORY PRINT #]fnbr , nbr, address%/array()	These commands save or read 'nbr' of data bytes from or to memory from or to an open disk file.
MEMORY INPUT [#]fnbr , nbr, address%/array()	The memory to be saved can be specified as an integer array in which case the nbr of bytes to be saved or read is checked against the array size. Alternatively, a memory address can be used in which case no checking can take place and user errors could result in a crash of the firmware..
MEMORY PACK source%()/sourceaddress%, dest%()/destaddress%, number, size	Memory pack and unpack allow integer values from one array to be compressed into another or uncompressed from one to the other. The two arrays are always normal integer arrays but the packed array can have 2, 4, 8, 16 or 64 values “packed” into them. Thus a single integer array element could store 2 off 32-bit words, 4 off 16 bit values, 8 bytes, 16 nibbles, or 64 booleans (bits). “number specifies the number of values to be packed or unpacked and “size” specifies the number of bits (1,4,8,16,or 32) Alternatively, memory address(es) can be used in which case no checking can take place and user errors could result in a crash of the firmware.
MEMORY UNPACK source%()/sourceaddress%, dest%()/destaddress%, number, size	
MKDIR dir\$	Make, or create, the directory ‘dir\$’ on the default Flash Filesystem or SD Card.
MID\$(str\$, start, num) = str2\$	The ‘num’ characters in 'str\$', beginning at position 'start', are replaced by the characters in 'str2\$'.
NEW	Clears the the program memory and all variables including saved variables. This command also clears the backup copy of the program which is held in flash memory.
NEXT [counter-variable] [, counter-variable], etc	NEXT comes at the end of a FOR-NEXT loop; see FOR. The ‘counter-variable’ specifies exactly which loop is being operated on. If no ‘counter-variable’ is specified the NEXT will default to the innermost loop. It is also possible to specify multiple counter-variables as in: <code>NEXT x, y, z</code>
ON ERROR ABORT or ON ERROR IGNORE or ON ERROR SKIP [nn] or ON ERROR CLEAR	This controls the action taken if an error occurs while running a program and applies to all errors discovered by MMBasic including syntax errors, wrong data, missing hardware, etc. ON ERROR ABORT will cause MMBasic to display an error message, abort the program and return to the command prompt. This is the normal behaviour and is the default when a program starts running. ON ERROR IGNORE will cause any error to be ignored. ON ERROR SKIP will ignore an error in a number of commands (specified by the number 'nn') executed following this command. 'nn' is optional, the default if not specified is one. After the number of commands has completed (with an error or not) the behaviour of MMBasic will revert to ON ERROR ABORT. If an error occurs and is ignored/skipped the read only variable MM.ERRNO will be set to non zero and MM.ERRMSG\$ will be set to the error message that would normally be generated. These are reset to zero and an empty string by ON ERROR CLEAR. They are also cleared when the program is run and when ON ERROR IGNORE and ON ERROR SKIP are used. ON ERROR IGNORE can make it very difficult to debug a program so it is strongly recommended that only ON ERROR SKIP be used.
ON KEY target or ON KEY ASCIIcode, target	The first version of the command sets an interrupt which will call 'target' user defined subroutine whenever there is one or more characters waiting in the serial console input buffer. Note that all characters waiting in the input buffer should be read in the interrupt subroutine otherwise another interrupt will be automatically generated as soon as the program returns from the interrupt.

内存打印#[fnbr , nbr, 地址%/数组()]	这些命令将‘nbr’个数据字节从内存保存到或从打开的磁盘文件读取。
内存输入[#]fnbr , nbr, 地址%/数组()	要保存的内存可以指定为一个整数数组，在这种情况下，保存或读取的字节数会与数组大小进行检查。或者，可以使用一个内存地址，在这种情况下不会进行检查，用户错误可能导致固件崩溃。
内存打包 source%()/sourceaddress%, dest%()/destaddress%, 编号, 大小 内存解包 source%()/sourceaddress%, dest%()/destaddress%, 编号, 大小	内存打包和解包允许将一个数组中的整数值压缩到另一个数组中，或从一个数组解压缩到另一个数组中。 这两个数组始终是普通的整数数组，但打包数组可以包含2、4、8、16或64个“打包”值。因此，单个整数数组元素可以存储2个32位字、4个16位值、8个字节、16个半字节或64个布尔值（位）。 “number指定要打包或解包的值的数量，而“size”指定位数（1、4、8、16或32） 或者，可以使用内存地址，在这种情况下无法进行检查，用户错误可能导致固件崩溃。
MKDIR dir\$	在默认的Flash文件系统或SD卡上创建目录‘dir\$’。
MID\$(str\$, start, num) = str2\$	在’str中，从位置’start’开始的‘num’个字符被’str2中的字符替换。
NEW	清除程序内存和所有变量，包括已保存的变量。 此命令还清除保存在闪存中的程序的备份副本。
NEXT [counter-variable] [, counter-variable]等	NEXT出现在FOR-NEXT循环的末尾；见FOR。 ‘counter-variable’精确指定正在操作的循环。如果没有‘如果指定了‘计数变量’，则NEXT将默认为最内层循环。也可以指定多个计数变量，如下所示： NEXT x, y, z
错误时中止 或 错误时忽略 或 错误时跳过 [nn] 或 错误时清除	这控制了在运行程序时发生错误时采取的行动，并适用于MMBasic发现的所有错误，包括语法错误、错误数据、缺失硬件等。 错误时中止将导致MMBasic显示错误信息，中止程序并返回到命令提示符。这是正常行为，并且是程序开始运行时的默认设置。 错误时忽略将导致任何错误被忽略。 错误时跳过将忽略在此命令之后执行的一定数量的命令（由数字‘nn’指定）。‘nn’是可选的，如果未指定，默认值为1。在命令数量完成后（无论是否有错误），MMBasic的行为将恢复为错误时中止。如果发生错误并被忽略/跳过，只读变量MM.ERRNO将被设置为非零，MM.ERRMSG\$将被设置为通常会生成的错误信息。这些在使用ON ERROR CLEAR时会重置为零和空字符串。当程序运行时以及使用ON ERROR IGNORE和ON ERROR SKIP时，它们也会被清除。 ON ERROR IGNORE 可能会使调试程序变得非常困难，因此强烈建议仅使用ON ERROR SKIP。
ON KEY target 或 ON KEY ASCIIcode, target	该命令的第一个版本设置了一个中断，当串行控制台输入缓冲区中有一个或多个字符等待时，将调用‘target’用户定义的子程序。 请注意，输入缓冲区中等待的所有字符应在中断子程序中读取，否则程序从中断返回后将自动生成另一个中断。

	<p>The second version allows you to associate an interrupt routine with a specific key press. This operates at a low level for the serial console and if activated the key does not get put into the input buffer but merely triggers the interrupt. It uses a separate interrupt from the simple ON KEY command so can be used at the same time if required.</p> <p>In both variants, to disable the interrupt use numeric zero for the target, i.e.: ON KEY 0. or ON KEY ASCIIcode, 0</p>
ON PS2 target	<p>This triggers an interrupt whenever the PicoMite sees a message from the PS2 interface.</p> <p>Use MM.info(PS2) to report the raw message received. This allows the programmer to trap both keypress and release.</p> <p>See https://wiki.osdev.org/PS/2_Keyboard for the scan codes (Set 2).</p>
ONEWIRE RESET pin or ONEWIRE WRITE pin, flag, length, data [, data...] or ONEWIRE READ pin, flag, length, data [, data...]	<p>Commands for communicating with 1-Wire devices.</p> <p>ONEWIRE RESET will reset the 1-Wire bus</p> <p>ONEWIRE WRITE will send a number of bytes</p> <p>ONEWIRE READ will read a number of bytes</p> <p>'pin' is the I/O pin (located in the rear connector) to use. It can be any pin capable of digital I/O.</p> <p>'flag' is a combination of the following options:</p> <ul style="list-style-type: none"> 1 - Send reset before command 2 - Send reset after command 4 - Only send/recv a bit instead of a byte of data 8 - Invoke a strong pullup after the command (the pin will be set high and open drain disabled) <p>'length' is the length of data to send or receive</p> <p>'data' is the data to send or variable to receive. The number of data items must agree with the length parameter.</p> <p>See also <i>Appendix C</i>.</p>
OPEN fname\$ FOR mode AS [#]fnbr	<p>Opens a file for reading or writing.</p> <p>'fname' is the filename with an optional extension separated by a dot (.). Long file names with upper and lower case characters are supported.</p> <p>A directory path can be specified with the backslash as directory separators. The parent of the current directory can be specified by using a directory name of “..” (two dots) and the current directory with “.” (a single dot).</p> <p>For example OPEN ".\dir1\dir2\filename.txt" FOR INPUT AS #1</p> <p>'mode' is INPUT, OUTPUT, APPEND or RANDOM.</p> <p>INPUT will open the file for reading and throw an error if the file does not exist. OUTPUT will open the file for writing and will automatically overwrite any existing file with the same name.</p> <p>APPEND will also open the file for writing but it will not overwrite an existing file; instead any writes will be appended to the end of the file. If there is no existing file the APPEND mode will act the same as the OUTPUT mode (i.e. the file is created then opened for writing).</p> <p>RANDOM will open the file for both read and write and will allow random access using the SEEK command. When opened the read/write pointer is positioned at the end of the file.</p> <p>'fnbr' is the file number (1 to 10). The # is optional. Up to 10 files can be open simultaneously. The INPUT, LINE INPUT, PRINT, WRITE and CLOSE commands as well as the EOF() and INPUT\$() functions all use 'fnbr' to identify the file being operated on.</p> <p>See also ON ERROR and MM.ERRNO for error handling.</p>

	<p>第二个版本允许您将中断例程与特定的按键按下关联。这在串行控制台的低级别操作，如果激活，按键不会被放入输入缓冲区，而只是触发中断。它使用一个独立的中断，与简单的 ON KEY 命令不同，因此可以在需要时同时使用。</p> <p>在这两种变体中，要禁用中断，请为目标使用数字零，即： ON KEY 0. 或 ON KEY ASCIIcode, 0</p>
ON PS2 target	<p>每当 PicoMite 从 PS2 接口看到消息时，这会触发一个中断。</p> <p>使用 MM.info(PS2) 来报告接收到的原始消息。这允许程序员捕获按键和释放事件。 请参见 https://wiki.osdev.org/PS/2_Keyboard 以获取扫描码（设置 2）。</p>
ONEWIRE RESET 引脚 或 ONEWIRE WRITE 引脚, 标志, 长度, 数据 [, 数据...] 或 ONEWIRE READ 引脚, 标志, 长度, 数据 [, 数据...]	<p>与 1-Wire 设备通信的命令。</p> <p>ONEWIRE RESET 将重置 1-Wire 总线</p> <p>ONEWIRE WRITE 将发送多个字节</p> <p>ONEWIRE READ 将读取多个字节</p> <p>'pin' 是要使用的输入/输出引脚（位于后部连接器）。它可以是任何能够进行数字输入/输出的引脚。</p> <p>'flag' 是以下选项的组合：</p> <ul style="list-style-type: none"> 1 - 在命令之前发送重置 2 - 在命令之后发送重置 4 - 仅发送/接收一个位而不是一个字节的数据 8 - 在命令后调用强上拉（引脚将被设置为高电平，开漏禁用） <p>'length' 是要发送或接收的数据长度</p> <p>'data' 是要发送的数据或要接收的变量。数据项的数量必须与长度参数一致。</p> <p>另见 附录 C .</p>
OPEN fname\$ FOR mode AS [#]fnbr	<p>打开一个文件以进行读取或写入。</p> <p>'fname' 是文件名，带有可选的扩展名，用点（.）分隔。 支持带有大小写字符的长文件名。</p> <p>可以使用反斜杠作为目录分隔符来指定目录路径。 当前目录的父目录可以通过使用目录名称“..”（两个点）来指定，当前目录使用“.”（一个点）。</p> <p>例如，使用 OPEN ".\dir1\dir2\filename.txt" FOR INPUT AS #1 ，‘模式’可以是 INPUT、OUTPUT、APPEND 或 R ANDOM。INPUT 将打开文件以进行读取，如果文件不存在则会抛出错误。OUTPUT 将打开文件以进行写入，并会自动覆盖任何同名的现有文件。</p> <p>APPEND 也将打开文件以进行写入，但不会覆盖现有文件；相反，任何写入将附加到文件的末尾。如果没有现有文件，APPEND 模式将与 OUTPUT 模式的行为相同（即创建文件然后打开以进行写入）。</p> <p>RANDOM 将同时打开文件以进行读写，并允许使用 SEEK 命令进行随机访问。打开时，读/写指针位于文件的末尾。</p> <p>'fnbr' 是文件编号（1 到 10）。# 是可选的。最多可以同时打开 10 个文件。INPUT、LINE INPUT、PRINT、WRITE 和 CLOSE 命令，以及 EOF() 和 INPUT\$() 函数都使用 'fnbr' 来识别正在操作的文件。</p> <p>另请参见 ON ERROR 和 MM.ERRNO 以进行错误处理。</p>

OPEN comspec\$ AS [#]fnbr	Will open a serial communications port for reading and writing. Two ports are available (COM1: and COM2:) and both can be open simultaneously. For a full description with examples see Appendix A. Using ‘fnbr’ the port can be written to and read from using any command or function that uses a file number.
OPEN comspec\$ AS GPS [,timezone_offset] [,monitor]	Will open a serial communications port for reading from a GPS receiver. See the GPS function for details. The sentences interpreted are GPRMC, GNRMC, GPGGA and GNGGA. The timezone_offset parameter is used to convert UTC as received from the GPS to the local timezone. If omitted the timezone will default to UTC. The timezone_offset can be any number between -12 and 14 allowing the time to be set correctly even for the Chatham Islands in New Zealand (UTC +12:45). If the monitor parameter is set to 1 then all GPS input is directed to the console. This can be stopped by closing the GPS channel.
OPTION	See the section Options earlier in this manual.
PAUSE delay	Halt execution of the running program for ‘delay’ ms. This can be a fraction. For example, 0.2 is equal to 200 µs. The maximum delay is 2147483647 ms (about 24 days). Note that interrupts will be recognised and processed during a pause.
PIN(pin) = value	For a ‘pin’ configured as digital output this will set the output to low (‘value’ is zero) or high (‘value’ non-zero). You can set an output high or low before it is configured as an output and that setting will be the default output when the SETPIN command takes effect. See the function PIN() for reading from a pin and the command SETPIN for configuring it. Refer to the section <i>Using the I/O pins</i> for a general description of the PicoMite’s input/output capabilities.
PIO	The RP2040 chip used in the PicoMite contains a programmable I/O system with two identical PIO devices (pio%=0 or pio%=1) acting like specialised CPU cores. See the Appendix for a more detailed description.
PIO assemble pio,linedata\$	This command will assemble and load text based PIO assembler code including labels for jumps Use: PIO assemble pio,".program anything" to initialise the assembler Use: PIO assemble pio,".side_set n [opt] [pindirs]" if using side set. This is mandatory in order to correctly construct the op-codes if one or more side set pins are used. It does not load the pinctrl register as this is specific to the state-machine. Also note the "opt" parameter changes the op-code on instructions that have a side parameter Use: PIO assemble pio,".line n" to assemble starting from a line other than 1 - this is optional Use: PIO assemble pio,".end program [list]" to terminate the assembly and program the pio. The optional parameter LIST causes a hex dump of the op-codes to the terminal. Use: PIO assemble pio,"label:" to define a label. This must appear as a separate command. Use: PIO assemble ““wrap target”” to specify where the program will wrap to. See PIO(.wrap target) for how to use this. Use: PIO assemble “.wrap” to specify where the program should wrap back to from “.wrap target” . See PIO(.wrap) for how to use this. Use: PIO assemble pio "instruction [parameters]" to define the actual PIO instructions that will be converted to machine code

OPEN comspec\$ AS [#]fnbr	<p>将打开一个用于读取和写入的串行通信端口。提供两个端口（COM1: 和 COM2:），并且可以同时打开这两个端口。有关完整描述和示例，请参见附录 A。</p> <p>使用 ‘fnbr’，可以使用任何使用文件编号的命令或函数对端口进行写入和读取。</p>
OPEN comspec\$ AS GPS [,timezone_offset] [,monitor]	<p>将打开一个用于从 GPS 接收器读取的串行通信端口。有关详细信息，请参见GPS功能。被解释的句子是GPRMC、GNRMC、GPGGA 和 GNG GA。</p> <p>timezone_offset 参数用于将从 GPS 接收到的 UTC 转换为本地时区。如果省略，时区将默认为 UTC。timezone_offset 可以是 -12 到 14 之间的任何数字，这样即使是新西兰的查塔姆群岛（UTC +12:45）也能正确设置时间。</p> <p>如果 monitor 参数设置为 1，则所有 GPS 输入将被定向到控制台。可以通过关闭 GPS 通道来停止此操作。</p>
选项	请参见本手册前面的选项部分。
PAUSE 延迟	<p>暂停正在运行的程序执行，持续 ‘延迟’ 毫秒。这个值可以是一个小数。例如，0.2 等于 200μs。最大延迟为 2147483647 毫秒（约 24 天）。</p> <p>请注意，在暂停期间会识别和处理中断。</p>
引脚(pin) = 值	<p>对于配置为数字输出的‘引脚’，这将把输出设置为低（‘值’为零）或高（‘值’非零）。您可以在将其配置为输出之前设置输出为高或低，并且该设置将在SETPIN命令生效时成为默认输出。</p> <p>请参阅函数PIN()以从引脚读取数据，以及命令SETPIN以进行配置。有关PicoMite输入/输出能力的一般描述，请参阅章节使用I/O引脚。</p>
PIO	PicoMite中使用的RP2040芯片包含一个可编程I/O系统，具有两个相同的PIO设备（pio%=0或pio%=1），它们像专用的中央处理器核心一样工作。有关更详细的描述，请参阅附录。
PIO assemble pio,linedata\$	<p>此命令将汇编并加载基于文本的PIO汇编代码，包括跳转的标签。</p> <p>使用：PIO assemble pio,".program anything" 来初始化汇编器。使用：PIO assemble pio,".side_set n [opt] [pindirs]" 如果使用侧面设置。这是强制性的，以便在使用一个或多个侧面设置引脚时正确构造操作码。</p> <p>它不会加载引脚控制寄存器，因为这与状态机特定。</p> <p>还要注意，“opt”参数会改变具有侧面参数的指令的操作码。</p> <p>使用：PIO assemble pio,".linen" 从除1以外的行开始汇编——这是可选的。</p> <p>使用：PIO assemble pio,".end program [list]" 来终止汇编并编程pio。可选参数LIST会将操作码的十六进制转储到终端。</p> <p>使用：PIO assemble pio,"label:" 来定义一个标签。这必须作为一个单独的命令出现。</p> <p>使用：PIO assemble “wrap target” 来指定程序将换行到哪里。请参见 PIO(.wrap target) 了解如何使用此功能。</p> <p>使用：PIO assemble “.wrap” 来指定程序应该从 “.wrap target” 回环到哪里。有关如何使用此功能，请参见 PIO(.wrap)。</p> <p>使用：PIO assemble pio "instruction [parameters]" 来定义将转换为机器代码的实际 PIO 指令。</p>

PIO DMA RX pio, sm, nbr, data%() [,completioninterrupt] [,transfersize] [,loopbackcount]	<p>Sets up DMA transfers from PIO to MMBasic memory pio specifies which of the two pio instances to use (0 or 1) sm specifies which of the state machine to use (0-3) nbr specifies how many 32-bit words to transfer. See below for the special case of setting nbr to zero. data%() is the array that will either supply or receive the PIO data</p> <p>The optional parameter completioninterrupt is the name of a MMBasic subroutine that will be called when the DMA completes and in the case of DMA_OUT the FIFO has been emptied.</p> <p>If the optional interrupt is not used then the status of the DMA can be checked using the functions</p> <p>MM.INFO(PIO RX DMA) MM.INFO(PIO TX DMA)</p> <p>The optional parameter transfersize allows the user to override the normal 32-bit transfers and select 8, 16, or 32.</p> <p>The optional parameter loopbackcount specifies how many data items are to be read or written before the DMA starts again at the beginning of the buffer</p> <p>The parameter must be a power of 2 between 2 and 32768</p> <p>Due to a limitation in the RP2040 if loopbackcount is used the MMBasic array must be aligned in memory to the number of bytes in the loop</p> <p>Thus if the array is 64 integers long which is 512 bytes then the array must be aligned to a 512byte boundary in memory</p> <p>All MMBasic arrays are aligned to a 256 byte boundary but to create an array which is guaranteed to be aligned to a 512 byte boundary or greater the PIO MAKE RING BUFFER command must be used</p> <p>If loopbackcount is set then "nbr" can be set to 0. In this case the transfer will run continuously repeatedly filling the buffer until explicitly stopped</p>
PIO DMA RX OFF PIO DMA TX OFF	Aborts a running DMA
PIO INTERRUPT pio, sm [,RXinterrupt] [,TXinterrupt]	<p>Sets Basic interrupts for PIO activity.</p> <p>Use the value 0 for RXinterrupt or TXinterrupt to disable an interrupt</p> <p>Omit values not needed</p> <p>The RX interrupt triggers whenever a word has been "pushed" by the PIO code into the specified FIFO. The data MUST be read in the interrupt to clear it.</p> <p>The TX interrupt triggers whenever the specified FIFO has been FULL and the PIO code has now "pulled" it</p>
PIO INIT MACHINE pio%, statemachine%, clockspeed [,pinctrl] [,execctrl] [,shiftctrl] [,startinstruction]	<p>PIO interrupts have priority over keyboard interrupts but before anything else. As with all interrupts interrupt conditions are processed one at a time.</p> <p>Initialises PIO 'pio%' with state machine 'statemachine%'. 'clockspeed' is the clock speed of the state machine in kHz. The four optional arguments are variables holding initialising values of the state machine registers and the address of the first instruction to execute (defaults to zero). These decide how the PIO will operate.</p>
PIO EXECUTE pio, state_machine, instruction%	<p>It is anticipated that eventually the PIO assembler will be able to generate the register values for the user along with the program array based on the defined assembler directives.</p> <p>Immediately executes the instruction on the pio and state machine specified.</p>

<p>PIO DMA RX pio, sm, nbr, data%() [,completioninterrupt] [,transfersize] [,loopbackcount]</p> <p>PIO DMA TX pio, sm, nbr, data%() [,completioninterrupt] [,transfersize] [,loopbackcount]</p>	<p>设置从 PIO 到 MMBasic 内存的 DMA 传输 pio 指定要使用的两个 pio 实例中的哪一个（0 或 1） sm 指定要使用的状态机（0-3） nbr 指定要传输多少个32位字。请参见下面将nbr设置为零的特殊案例。</p>
	<p>data%() 是将提供或接收PIO数据的数组。可选参数completioninterrupt是一个MMBasic子程序的名称，当DMA完成时会被调用，在DMA_OUT的情况下，FIFO已被清空。</p>
	<p>如果不使用可选中断，则可以使用函数MM.INFO(PIO RX DMA)检查DMA的状态。</p>
	<p>MM.INFO(PIO TX DMA) 可选参数transfersize允许用户覆盖正常的32位传输，并选择8、16或32。 可选参数loopbackcount指定在DMA再次从缓冲区开始之前要读取或写入多少个数据项。该参数必须是2的幂，范围在2到32768之间。</p>
	<p>由于RP2040的限制，如果使用loopbackcount，则MMBasic数组必须在内存中对齐到循环中的字节数。因此，如果数组长度为64个整数，即512字节，则数组必须在内存中对齐到512字节边界。</p>
	<p>所有MMBasic数组都对齐到256字节边界，但要创建一个保证对齐到512字节边界或更大的数组，必须使用PIO MAKE RING BUFFER命令</p>
	<p>如果设置了loopbackcount，则“nbr”可以设置为0。在这种情况下，传输将持续运行，重复填充缓冲区，直到明确停止</p>
<p>PIO DMA RX 关闭 PIO DMA TX 关闭</p>	<p>中止正在运行的DMA</p>
<p>PIO 中断 pio, sm [,RXinterrupt] [,TXinterrupt]</p>	<p>为PIO活动设置基本中断。 使用值0来禁用RXinterrupt或TXinterrupt 省略不需要的值</p>
	<p>当PIO代码将一个字“推送”到指定的FIFO时，RX中断会触发。数据必须在中断中读取以清除它。TX中断在指定的FIFO已满且PIO代码现在“拉取”它时触发</p>
<p>PIO 初始化机器 pio%, 状态机%, 时钟速度 [,引脚控制] [,执行控制] [,移位控制] [,startinstruction]</p>	<p>PIO中断优先于键盘中断，但在其他任何中断之前。 与所有中断一样，中断条件一次处理一个。 初始化PIO 'pio%' 与状态机 'statemachine%'。'时钟速度' 是状态机的时钟速度，以千赫为单位。四个可选参数是持有状态机寄存器初始化值的变量，以及要执行的第一条指令的地址（默认为零）。这些决定了PIO 的操作方式。</p>
	<p>预计最终 PIO 汇编器将能够根据定义的汇编指令为用户生成寄存器值以及程序数组。</p>
<p>PIO EXECUTE pio, state_machine, instruction%</p>	<p>立即在指定的 pio 和状态机上执行指令。</p>

PIO WRITE pio, state_machine, count, data0 [,data1..]	Writes the data elements to the pio and state machine specified. The write is blocking so the state machine needs to be able to take the data supplied NB: this command will probably need additional capability in future releases
PIO READ pio, state_machine, count, data%[0]	Reads the data elements from the pio and state machine specified. The read is non-blocking so the state machine needs to be able to supply the data requested. When count is one then an integer can be used to receive the data, otherwise an integer array should be specified. NB: this command will probably need additional capability in future releases
PIO START pio, statemachine	Start a given state machine on pio
PIO STOP pio, statemachine	Stop a given state machine on pio
PIO CLEAR pio	This stops the pio specified on all statemachines and clears the control registers for the statemachines PINCTRL, EXECTRL, and SHIFTCTRL to defaults
PIO PROGRAM LINE pio, line, instruction	Programs just the specified line in a PIO program
PIXEL x, y [,c]	Set a pixel on an attached LCD panel to a colour. 'x' is the horizontal coordinate and 'y' is the vertical coordinate of the pixel. 'c' is a 24 bit number specifying the colour. 'c' is optional and if omitted the current foreground colour will be used. All parameters can be expressed as arrays and the software will plot the number of pixels as determined by the dimensions of the smallest array. 'x' and 'y' must both be arrays or both be single variables /constants otherwise an error will be generated. 'c' can be either an array or a single variable or constant. See the section <i>Graphics Commands and Functions</i> for a definition of the colours and graphics coordinates.
PLAY	This command will generate a variety of audio outputs. See the OPTION AUDIO command for setting the I/O pins to be used for the output. The audio is a pulse width modulated signal so a low pass filter is required to remove the carrier frequency.
PLAY TONE left [, right [, dur] [,interrupt]]]	Generates two separate sine waves on the sound output left and right channels. 'left' and 'right' are the frequencies in Hz to use for the left and right channels. The tone plays in the background (the program will continue running after this command) and 'dur' specifies the number of milliseconds that the tone will sound for. If the duration is not specified the tone will continue until explicitly stopped or the program terminates. 'interrupt' is an optional subroutine which will be called when the play terminates. The frequency can be from 1Hz to 20KHz and is very accurate (it is based on a crystal oscillator). The frequency can be changed at any time by issuing a new PLAY TONE command.
PLAY FLAC file\$ [, interrupt]	Will play a FLAC file on the sound output. 'file\$' is the FLAC file to play (the extension of .flac will be appended if missing). The sample rate can be up to 48kHz in stereo (96kHz if the Pico is overclocked) The FLAC file is played in the background. 'interrupt' is optional and is the name of a subroutine which will be called when the file has finished playing. If file\$ is a directory the Pico will play all of the files in that directory in turn.
PLAY WAV file\$ [, interrupt]	Will play a WAV file on the sound output. 'file\$' is the WAV file to play (the extension of .wav will be appended if

PIO WRITE pio, state_machine, count, data0 [,data1 ..]	将数据元素写入指定的 pio 和状态机。写入是阻塞的，因此状态机需要能够接收提供的数据 注意：此命令可能在未来版本中需要额外的功能
PIO READ pio, state_machine, count, data%[0]	从指定的 pio 和状态机读取数据元素。读取是非阻塞的，因此状态机需要能够提供请求的数据。当计数为一时，可以使用一个整数来接收数据，否则应指定一个整数数组。
PIO 启动 pio, 状态机	注意：此命令可能在未来的版本中需要额外的功能 在 pio 上启动给定的状态机
PIO 停止 pio, 状态机	在 pio 上停止给定的状态机
PIO 清除 pio	这会停止所有状态机上指定的 pio，并将状态机的控制寄存器 PINCTRL、EXECTRL 和 SHIFTCTRL 清除为默认值
PIO 程序 行 pio, 行, 指令	仅在 PIO 程序中编程指定的行
像素 x, y [,c]	<p>将连接的液晶面板上的一个像素设置为一种颜色。'x' 是像素的水平坐标，'y' 是像素的垂直坐标。'c' 是一个 24 位数字指定颜色。'c' 是可选的，如果省略，将使用当前的前景颜色。所有参数都可以表示为数组，软件将根据最小数组的尺寸绘制像素数量。'x' 和 'y' 必须都是数组，或者都是单个变量/常量否则将生成错误。'c' 可以是数组，也可以是单个变量或常量。</p> <p>请参见章节图形命令和函数以获取颜色和图形坐标的定义。</p>
播放	此命令将生成多种音频输出。 请参阅 OPTION AUDIO 命令以设置用于输出的 I/O 引脚。音频是脉宽调制信号，因此需要低通滤波器来去除载波频率。
PLAY TONE left [, right [, dur] [,interrupt]]]	<p>在声音输出的左通道和右通道上生成两个独立的正弦波。</p> <p>'left' 和 'right' 是用于左通道和右通道的频率（单位：赫兹）。音调在后台播放（该程序将在此命令后继续运行），'dur' 指定音调将持续的毫秒数。如果未指定持续时间，音调将继续播放，直到明确停止或程序终止。</p> <p>'中断'是一个可选的子程序，当播放结束时将被调用。</p> <p>频率可以从1赫兹到20千赫，且非常准确（基于晶体振荡器）。频率可以通过发出新的播放音调命令随时更改。</p>
播放 FLAC 文件\$ [, 中断]	<p>将在声音输出上播放 FLAC 文件。</p> <p>'文件'是要播放的 FLAC 文件（如果缺少扩展名，将自动添加 .flac）。采样率可以高达48千赫（如果树莓派超频，则为96千赫）。</p> <p>FLAC 文件在后台播放。'中断'是可选的，是一个子程序的名称，当文件播放完毕时将被调用。</p> <p>如果文件\$是一个目录，树莓派将依次播放该目录中的所有文件。</p>
播放 WAV 文件\$ [, 中断]	<p>将在声音输出上播放 WAV 文件。</p> <p>'file'是要播放的WAV文件（如果缺少，将附加扩展名 .wav）</p>

	<p>missing). The WAV file must be PCM encoded in mono or stereo with 8 or 16-bit sampling. The sample rate can be up to 48kHz in stereo (96kHz if the Pico is overclocked).</p> <p>The WAV file is played in the background. 'interrupt' is optional and is the name of a subroutine which will be called when the file has finished playing.</p>
PLAY MODFILE file\$ [,interrupt]	<p>Will play a MOD file on the sound output. 'file\$' is the MOD file to play (the extension of .mod will be appended if missing).</p> <p>The MOD file is played in the background and will play continuously in a loop. If the optional 'interrupt' is specified This will be called when the file has played once through the sequence and playback will then be terminated.</p>
PLAY MODSAMPLE Sampelenum, channel [,volume]	<p>Plays a specific sample in the mod file on the channel specified. The volume is optional and can be between 0 and 64. This command can only be used when there is a mod file already playing and allows sound effects to be output whilst the background music is still playing.</p>
PLAY LOAD SOUND array%()	<p>Loads a 1024 element array comprising 4096 16-bit values between 0 and 4095. This provides the data for any arbitrary waveform that can be played by the PLAY SOUND command. You can use the MEMORY PACK command to create the arrays from a normal 40956 element integer array.</p>
PLAY SOUND soundno, channelno, type [,frequency] [,volume]	<p>Play a series of sounds simultaneously on the audio output. 'soundno' is the sound number and can be from 1 to 4 allowing for four simultaneous sounds on each channel.</p> <p>'channelno' specifies the output channel. It can be L (left speaker), R (right speaker), B (both speakers) or M (mono output with the right channel inverted compared to the left).</p> <p>'type' specifies the wave form It can be S (Sine wave), Q (square wave), T (triangular wave), W (saw tooth), O (Null output), P (periodic noise), N (random noise) or U (user defined using the PLAY LOAD sound command).to be used.</p> <p>'frequency' is the frequency from 1 to 20000 (Hz) and it must be specified except when type is O.</p> <p>'volume' is optional and must be between 1 and 25. It defaults to 25</p> <p>The first time PLAY SOUND is called all other audio usage will be blocked and will remain blocked until PLAY STOP is called. Output can be stopped temporarily using PLAY PAUSE and PLAY RESUME.</p> <p>Calling SOUND on an already running 'soundno' will immediately replace the previous output. Individual sounds are turned off using type "O"</p> <p>Running 4 sounds simultaneously on both channels of the audio output consumes about 23% of the CPU.</p>
PLAY PAUSE	PLAY PAUSE will temporarily halt the currently playing file or tone.
PLAY RESUME	PLAY RESUME will resume playing a sound that was paused.
PLAY STOP	PLAY STOP will terminate the playing of the file or tone. When the program terminates for whatever reason the sound output will also be automatically stopped.
PLAY VOLUME left, right	Will adjust the volume of the audio output. 'left' and 'right' are the levels to use for the left and right channels and can be between 0 and 100 with 100 being the maximum volume. There is a linear relationship between the specified level and the output. The volume defaults to maximum when a program is run.
PLAY NEXT	Stops playback of the current audio file and starts the next one in the directory

	<p>WAV文件必须是单声道或立体声的PCM编码，采样位数为8位或16位。采样率在立体声中可以高达48kHz（如果树莓派超频，则为96kHz）</p> <ul style="list-style-type: none"> ◦ WAV文件在后台播放。'中断'是可选的，是一个子程序的名称，当文件播放完毕时将被调用。
播放 MODFILE file\$ [,interrupt]	<p>将在声音输出上播放MOD文件。 'file' 是要播放的MOD文件（如果缺少，将附加扩展名 .mod）。</p> <p>MOD文件在后台播放，并将持续循环播放。如果指定了可选的 'interrupt'，则在文件播放完一次后将调用此函数，播放将被终止。</p>
播放 MODSAMPLE 样本编号, 通道 [,音量]	<p>在指定的通道上播放mod文件中的特定样本。音量是可选的，可以在0到64之间。此命令仅在已有mod文件播放时使用，并允许在背景音乐播放时输出音效。</p>
播放 加载 声音 array%()	<p>加载一个包含4096个16位值（范围在0到4095之间）的1024元素数组。这为可以通过播放声音命令播放的任意波形提供数据。您可以使用内存包命令从一个正常的40956元素整数数组创建这些数组。</p>
播放 声音 soundno, channelno, type [,frequency] [,volume]	<p>在音频输出上同时播放一系列声音。 'soundno' 是声音编号，可以从1到4，允许每个通道同时播放四个声音。 'channelno' 指定输出通道。可以是L（左扬声器）、R（右扬声器）、B（两个扬声器）或M（单声道输出，右通道与左通道相反）。</p> <p>'type' 指定波形类型，可以是 S（正弦波）、Q（方波）、T（三角波）、W（锯齿波）、O（无输出）、P（周期噪声）、N（随机噪声）或 U（使用 PLAY LOAD 声音命令定义的用户自定义声音）。</p> <p>'frequency' 是频率范围从 1 到 20000（赫兹），必须指定除非类型为 O。</p> <p>'volume' 是可选的，必须在 1 到 25 之间。默认值为 25 第一次调用 PLAY SOUND 时，所有其他音频使用将被阻止并将保持阻止状态，直到调用 PLAY STOP。可以使用 PLAY PAUSE 和 PLAY RESUME 暂时停止输出。</p> <p>在已经运行的 'soundno' 上调用 SOUND 将立即替换之前的输出。使用类型“O”可以关闭单个声音同时在两个音频输出通道上运行 4 个声音大约消耗 23% 的中央处理器。</p>
播放 暂停 播放 恢复 播放 停止	<p>播放 暂停将暂时停止当前正在播放的文件或音调。 播放 恢复将恢复播放已暂停的声音。 播放 停止将终止文件或音调的播放。无论因何原因程序终止，声音输出也将自动停止。</p>
播放 音量 左, 右	<p>将调整音频输出的音量。 '左'和'右'是用于左通道和右通道的音量级别，可以在0到100之间，100为最大音量。指定的音量级别与输出之间存在线性关系。当程序运行时，音量默认为最大值。</p>
PLAY NEXT	<p>停止当前音频文件的播放，并开始播放目录中的下一个文件。</p>

PLAY PREVIOUS	Stops playback of the current audio file and starts the previous one in the directory
PLAY MP3 file\$ [, interrupt]	<p>VS1053 specific PLAY commands</p> <p>Will play a MP3 file on the sound output. 'file\$' is the MP3 file to play (the extension of .mp3 will be appended if missing). The sample rate should be 44100Hz stereo.</p> <p>The MP3 file is played in the background. 'interrupt' is optional and is the name of a subroutine which will be called when the file has finished playing. If file\$ is a directory the Pico will play all of the files in that directory in turn.</p>
PLAY HALT	This command works when a MP3 file is playing. It stops playback and records the current file position to allow playback to be resumed from the same point. This command is specifically designed to support for mp3 audio books
PLAY CONTINUE track\$	<p>Resumes playback of the MP3 track specified. "track\$" will be the name of the file that was playing when halted with all file attributes removed</p> <p>e.g.</p> <pre>PLAY MP3 "B:/mp3/mymp3.mp3" sometime later PLAY HALT later again PLAY CONTINUE "mymp3"</pre>
PLAY MIDIFILE file\$ [, interrupt]	<p>Will play a MIDI file on the sound output. 'file\$' is the MIDI file to play (the extension of .mid will be appended if missing).</p> <p>The MIDI file is played in the background. 'interrupt' is optional and is the name of a subroutine which will be called when the file has finished playing. If file\$ is a directory the Pico will play all of the files in that directory in turn.</p>
PLAY MIDI	Initiates the real-time midi mode. In this mode midi instructions can be sent to the VS1053 to select which instruments to play on which channels, turn notes on, and turn them off in real timer
PLAY MIDI CMD cmd%, data1%, data2%	<p>Sends a midi command when in real time midi mode. An example would be to allocate an instrument to a channel. E.g.</p> <pre>PLAY MIDI CMD &B11000001,4 'set channel 1 to instrument 4'</pre>
PLAY MIDI TEST n	Plays a MIDI test sequence, n=0 to 3, 0 = normal realtime, the others play note and instrument samples
PLAY NOTE ON channel%, note%, velocity%	Turns on the note on the channel specified when in real time MIDI mode
PLAY NOTE OFF channel%, note% [,velocity%]	Turns off the note on the channel specified when in real time MIDI mode
PLAY STREAM buffer%(), readpointer%, writepointer%	Sends data to the VS1053 CODEC from the circular buffer "buffer%". This command initiates a background output stream where the VS1053 is sent anything in the buffer between the readpointer and the write pointer, updating the readpointer as it goes. Can be used for arbitrary waveform output.
POKE BYTE addr%, byte or POKE SHORT addr%, short% Or	<p>Will set a byte or a word within the virtual memory space.</p> <p>POKE BYTE will set the byte (i.e. 8 bits) at the memory location 'addr%' to 'byte'. 'addr%' should be an integer.</p> <p>POKE SHORT will set the short integer (i.e. 16 bits) at the memory location 'addr%' to 'word%'. 'addr%' and 'short%' should be integers.</p>

PLAY PREVIOUS	停止当前音频文件的播放，并开始播放目录中的上一个文件。
播放 MP3 文件\$ [, 中断]	<p>VS1053 特定播放命令</p> <p>将在声音输出上播放 MP3 文件。 'file' 是要播放的 MP3 文件（如果缺少扩展名，将自动添加 .mp3）。采样率应为 44100 赫兹立体声。 MP3 文件在后台播放。'中断'是可选的，是一个子程序的名称，当文件播放完毕时将被调用。 如果文件\$是一个目录，树莓派将依次播放该目录中的所有文件。</p>
暂停播放	此命令在 MP3 文件播放时有效。它停止播放并记录当前文件位置，以便从同一点恢复播放。此命令专门设计用于支持 MP3 有声书。
继续播放 track\$	恢复指定的 MP3 曲目的播放。"track\$" 将是暂停时播放的文件名称，所有文件属性将被移除，例如：
	<p>播放 MP3 "B:/mp3/mymp3.mp3" 过了一段时间 播放停止 再过一会儿 播放继续 "mymp3"</p>
播放 MIDI 文件 file\$ [, 中断]	<p>将在声音输出上播放 MIDI 文件。 'file' 是要播放的 MIDI 文件（如果缺少扩展名，将自动添加 .mid）。</p> <p>MIDI 文件在后台播放。'中断'是可选的，是一个子程序的名称，当文件播放完毕时将被调用。 如果文件\$是一个目录，树莓派将依次播放该目录中的所有文件。</p>
播放 MIDI	启动实时 MIDI 模式。在此模式下，可以向 VS1053 发送 MIDI 指令，以选择在特定通道上播放哪些乐器，实时开启和关闭音符。
播放 MIDI 命令 cmd%，数据1%，数据2%	在实时 MIDI 模式下发送 MIDI 命令。一个例子是将乐器分配给通道。例如： PLAY MIDI CMD &B11000001,4 ‘将通道 1 设置为乐器 4’
播放 MIDI 测试 n	播放 MIDI 测试序列，n=0 到 3，0 = 正常实时，其他则播放音符和乐器样本。
播放音符开启 通道%，音符%，速度%	在实时 MIDI 模式下，开启指定通道上的音符。
播放音符关闭 通道%，音符% [,速度%]	在实时 MIDI 模式下，关闭指定通道上的音符。
播放流缓冲区%(), 读取指针%, 写入指针%	从循环缓冲区“buffer%”向 VS1053 编解码器发送数据。此命令启动一个后台输出流，将缓冲区中读取指针和写入指针之间的内容发送到 VS1053，并在此过程中更新读取指针。可用于任意波形输出。
POKE BYTE addr%, byte 或 POKE SHORT addr%, short% 或	<p>将在虚拟内存空间中设置一个字节或一个字。 POKE BYTE 将把内存位置'addr%'的字节（即8位）设置为'byte'。'addr%'应为整数。 POKE SHORT 将把内存位置'addr%'的短整数（即16位）设置为'word%'。 'addr%' 和 'short%' 应为整数。</p>

<p>POKE WORD addr%, word% or POKE INTEGER addr%, int% or POKE FLOAT addr%, float! or POKE VAR var, offset, byte or POKE VARTBL, offset, byte or POKE DISPLAY command [,data1] [,data2] [,datan]</p>	<p>POKE WORD will set the word (i.e. 32 bits) at the memory location 'addr%' to 'word%'. 'addr%' and 'word%' should be integers. POKE INTEGER will set the MMBasic integer (i.e. 64 bits) at the memory location 'addr%' to int%. 'addr%' and int% should be integers. POKE FLOAT will set the word (i.e. 32 bits) at the memory location 'addr%' to 'float!'. 'addr%' should be an integer and 'float!' a floating point number. POKE VAR will set a byte in the memory address of 'var'. 'offset' is the ±offset from the address of the variable. An array is specified as var(). POKE VARTBL will set a byte in MMBasic's variable table. 'offset' is the ±offset from the start of the variable table. Note that a comma is required after the keyword VARTBL.</p> <p>This command sends commands and associated data to the display controller for a connected display. This allows the programmer to change parameters of how the display is configured. e.g. POKE DISPLAY &H28 will turn off an SSD1963 display and POKE DISPLAY &H29 will turn it back on again. Works for all displays except the ST7790.</p>
<p>POKE DISPLAY HRES n POKE DISPLAY VRES n</p>	<p>These commands change the stored value of MM.HRES and MM.VRES allowing the programmer to configure non-standard displays.</p>
<p>POLYGON n, xarray%(), yarray%() [, bordercolour] [, fillcolour] POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()] POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour]</p>	<p>Draws a filled or outline polygon with n xy-coordinate pairs in xarray%() and yarray%(). If 'fillcolour' is omitted then just the polygon outline is drawn. If 'bordercolour' is omitted then it will default to the current default foreground colour. If the last xy-coordinate pair is not the same as the first the firmware will automatically create an additional xy-coordinate pair to complete the polygon. The size of the arrays should be at least as big as the number of x,y coordinate pairs. 'n' can be an array and the colours can also optionally be arrays as follows: POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour() POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour] The elements of array n() define the number of xy-coordinate pairs in each of the polygons. e.g. DIM n(1)=(3,3) would define that 2 polygons are to be drawn with three vertices each. The size of the n array determines the number of polygons that will be drawn unless an element is found with the value zero in which case the firmware only processes polygons up to that point. The x,y-coordinate pairs for all the polygons are stored in xarray%() and yarray%(). The xarray%() and yarray%() parameters must have at least as many elements as the total of the values in the n array. Each polygon can be closed with the first and last elements the same. If the last element is not the same as the first the firmware will automatically create an additional x,y-coordinate pair to complete the polygon. If fill colour is omitted then just the polygon outlines are drawn. The colour parameters can be a single value in which case all polygons are drawn in the same colour or they can be arrays with the same cardinality as n. In this case each polygon drawn can have a different colour of both border and/or fill. For example, this will draw 3 triangles in yellow, green and red:</p> <pre>DIM c%(2)=(3,3,3) DIM x%(8)=(100,50,150,100,50,150,100,50,150) DIM y%(8)=(50,100,100,150,200,200,250,300,300) DIM fc%(2)=(rgb(yellow),rgb(green),rgb(red)) POLYGON c%,x%,y%,fc%,fc%</pre>

<p>POKE WORD addr%, word% 或 POKE INTEGER addr%, int% 或 POKE FLOAT addr%, float! 或 POKE VAR var, offset, byte 或 POKE VARTBL, offset, byte 或 POKE DISPLAY 命令 [,数据1] [,数据2] [,数据n]</p>	<p>POKE WORD 将在内存位置 'addr%' 设置字（即 32 位）为 'word%'。'addr%' 和 'word%' 应为整数。 POKE INTEGER 将在内存位置 'addr%' 设置 MMBasic 整数（即 64 位）为 'int%'。'addr%' 和 'int%' 应为整数。 POKE FLOAT 将在内存位置 'addr%' 设置字（即 32 位）为 'float!'。'addr%' 应为整数，'float!' 应为浮点数。 POKE VAR 将在 'var' 的内存地址中设置一个字节。'offset' 是变量地址的 \pm偏移量。数组被指定为 var()。 POKE VARTBL 将在 MMBasic 的变量表中设置一个字节。'offset' 是变量表起始位置的 \pm偏移量。请注意，关键字 VARTBL 后需要一个逗号。 此命令将命令和相关数据发送到连接显示器的显示控制器。这允许程序员更改显示配置的参数。例如，POKE DISPLAY &H28 将关闭 SSD1963 显示器，而 POKE DISPLAY &H29 将重新打开它。 适用于除 ST7790 之外的所有显示器。</p>
<p>POKE DISPLAY HRES n POKE DISPLAY VRES n</p> <p>POLYGON n, xarray%(), yarray%() [, bordercolour] [, fillcolour] POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()] POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour]</p>	<p>这些命令更改 MM.HRES 和 MM.VRES 的存储值，允许程序员配置非标准显示器。</p> <p>绘制一个填充或轮廓多边形，包含 n 个 xy 坐标对在 xarray%() 和 yarray%() 中。如果省略了‘填充颜色’，则只绘制多边形的轮廓。如果省略了‘边框颜色’，则默认为当前的默认前景颜色。 如果最后一个xy坐标对与第一个不同，固件将自动创建一个额外的xy坐标对以完成多边形。 数组的大小应至少与x,y坐标对的数量相同。 'n'可以是一个数组，颜色也可以选择性地是数组，如下所示： POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour] POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour]] 数组n()的元素定义每个多边形中的xy坐标对的数量。例如，DIM n(1)= (3,3)将定义绘制2个多边形，每个多边形有三个顶点。n数组的大小决定将绘制的多边形数量，除非找到值为零的元素，在这种情况下，固件仅处理到该点的多边形。所有多边形的 x,y 坐标对存储在 xarray%() 和 yarray%() 中。 xarray%() 和 yarray%() 参数必须至少与 n 数组中的值总数相等。 每个多边形可以通过使第一个和最后一个元素相同来闭合。如果最后一个元素与第一个不同，固件将自动创建一个额外的 x,y 坐标对以完成多边形。如果省略填充颜色则只绘制多边形的轮廓。 颜色参数可以是单个值，在这种情况下，所有多边形都以相同的颜色绘制，或者它们可以是与 n 具有相同基数的数组。在这种情况下，每个绘制的多边形可以具有不同的边框和/或填充颜色。例如，这将绘制 3 个三角形，颜色分别为黄色、绿色和红色： DIM c% (2)=(3,3,3) DIM x% (8)=(100,50,150,100,50,150,100,50,150) DIM y% (8)=(50,100,100,150,200,200,250,300,300) DIM fc% (2)=(rgb(yellow),rgb(green),rgb(red)) POLYGON c%(),x%(),y%(),fc%(),fc%()</p>

<p>PORT(start, nbr [,start, nbr]...) = value</p>	<p>Set a number of I/O pins simultaneously (i.e. with one command). 'start' is an I/O pin number and the lowest bit in 'value' (bit 0) will be used to set that pin. Bit 1 will be used to set the pin 'start' plus 1, bit 2 will set pin 'start'+2 and so on for 'nbr' number of bits. I/O pins used must be numbered consecutively and any I/O pin that is invalid or not configured as an output will cause an error. The start/nbr pair can be repeated if an additional group of output pins needed to be added.</p> <p>For example; PORT(15, 4, 23, 4) = &B10000011 Will set eight I/O pins. Pins 15 and 16 will be set high while 17, 18, 23, 24 and 25 will be set to a low and finally 26 will be set high.</p> <p>This command can be used to conveniently communicate with parallel devices like LCD displays. Any number of I/O pins (and therefore bits) can be used from 1 to the number of I/O pins on the chip.</p> <p>See the PORT function to simultaneously read from a number of pins.</p>
<p>PRINT expression [[,]expression] ... etc</p>	<p>Outputs text to the serial console followed by a carriage return/newline pair. Multiple expressions can be used and must be separated by either a:</p> <ul style="list-style-type: none"> • Comma (,) which will output the tab character • Semicolon (;) which will not output anything (it is just used to separate expressions). • Nothing or a space which will act the same as a semicolon. <p>A semicolon (;) or a comma (,) at the end of the expression list will suppress the output of the carriage return/newline pair at the end of a print statement. When printed, a number is preceded with a space if positive or a minus (-) if negative but is not followed by a space. Integers (whole numbers) are printed without a decimal point while fractions are printed with the decimal point and the significant decimal digits. Large or small floating point numbers are automatically printed in scientific number format.</p> <p>The function TAB() can be used to space to a certain column and the STR\$() function can be used to justify or otherwise format strings.</p>
<p>PRINT #nbr, expression [[,]expression] ... etc</p>	<p>Same as above except that the output is directed to a serial communications port or a file opened for OUTPUT or APPEND with a file number of 'nbr'. See the OPEN command.</p>
<p>PRINT #GPS, expression [[,]expression] ... etc</p>	<p>Outputs a NMEA string to an opened GPS device. The string must start with a \$ character and end with a * character. The checksum is automatically calculated and appended to the string together with the CR/LF characters.</p>
<p>PRINT @(x [, y]) expression Or PRINT @x, [y], m) expression</p>	<p>Works on terminal console on an attached computer or the display if OPTION LCDPANEL CONSOLE is enabled.</p> <p>Same as the standard PRINT command except that the cursor is positioned at the coordinates x, y expressed in pixels. If y is omitted the cursor will be positioned at "x" on the current line.</p> <p>Example: PRINT @(150, 45) "Hello World"</p> <p>The @ function can be used anywhere in a print command.</p> <p>Example: PRINT @(150, 45) "Hello" @(150, 55) "World"</p> <p>The @(x,y) function can be used to position the cursor anywhere on or off the screen. For example, PRINT @(-10, 0) "Hello" will only show "llo" as the first two characters could not be shown because they were off the screen.</p> <p>The @(x,y) function will automatically suppress the automatic line wrap normally performed when the cursor goes beyond the right screen margin.</p> <p>If 'm' is specified the mode of the video operation will be as follows:</p> <p>m = 0 Normal text (white letters, black background) m = 1 The background will not be drawn (ie, transparent) m = 2 The video will be inverted (black letters, white background) m = 5 Current pixels will be inverted (transparent background)</p>

PORT(start, nbr [,start, nbr]...) = 值	<p>同时设置多个I/O引脚（即使用一个命令）。</p> <p>'start' 是一个I/O引脚编号，'value' 中的最低位（位 0）将用于设置该引脚。位 1 将用于设置引脚 'start' 加 1，位 2 将设置引脚 'start' + 2，以此类推，直到 'nbr' 位数。使用的I/O引脚必须连续编号，任何无效或未配置为输出的I/O引脚将导致错误。如果需要添加额外的输出引脚，可以重复使用 start/nbr 对。</p> <p>例如； PORT(15, 4, 23, 4) = &B10000011 将设置八个I/O引脚。引脚15和16将被设置为高电平，而17、18、23、24和25将被设置为低电平，最后26将被设置为高电平。</p> <p>此命令可用于方便地与并行设备（如液晶显示器）进行通信。可以使用任意数量的I/O引脚（因此也可以是任意数量的位），范围从1到芯片上的I/O引脚数量。</p> <p>请参阅PORT函数以同时读取多个引脚。</p>
PRINT expression [[,]expression] ... 等等	<p>将文本输出到串行控制台，后面跟着一个回车/换行对。</p> <p>可以使用多个表达式，必须用以下任一方式分隔：</p> <ul style="list-style-type: none"> • 逗号 (,)，将输出制表符 • 分号 (;)，将不输出任何内容（仅用于分隔表达式）。 • 什么都不输入或一个空格将与分号的作用相同。 <p>在表达式列表末尾的分号 (;) 或逗号 (,) 将抑制打印语句末尾的回车/换行对的输出。</p> <p>当打印时，正数前面会有一个空格，负数前面会有一个负号 (-)，但后面不会有空格。整数（整数）将不带小数点打印，而分数则会带有小数点和有效的小数位。大或小的浮点数会自动以科学计数法格式打印。</p> <p>函数 TAB() 可用于对齐到特定列，而 STR\$() 函数可用于调整或格式化字符串。</p>
PRINT #nbr, 表达式 [[,]表达式] ... 等等	<p>与上述相同，只是输出被定向到串行通信端口或以 OUTPUT 或 APPEND 打开的文件，文件编号为 'nbr'。</p> <p>查看 OPEN 命令。</p>
PRINT #GPS, 表达式 [[,]表达式] ... 等等	<p>将 NMEA 字符串输出到已打开的 GPS 设备。字符串必须以 \$ 字符开头，并以 * 字符结尾。校验和会自动计算并与 CR/LF 字符一起附加到字符串中。</p>
PRINT @(x [, y]) 表达式 或 PRINT @(x, [y], m) 表达式	<p>在连接的计算机的终端控制台或启用 OPTION LCDPANEL CONSOLE 的显示器上工作。</p> <p>与标准 PRINT 命令相同，除了光标定位在以像素表示的坐标 x, y 处。如果省略 y，光标将定位在当前行的“x”处。</p> <p>示例：PRINT @(150, 45) "Hello World"</p> <p>@ 函数可以在打印命令中的任何地方使用。</p> <p>示例：PRINT @(150, 45) "Hello" @(150, 55) "World" @(x, y) 函数可以用于将光标定位在屏幕上的任何位置。例如，PRINT @(-10, 0) "Hello" 只会显示 "llo"，因为前两个字符无法显示，因为它们超出了屏幕范围。</p> <p>@(x,y) 函数将自动抑制光标超出右侧屏幕边缘时通常执行的自动换行。</p> <p>如果指定了 'm'，视频操作的模式如下：</p> <ul style="list-style-type: none"> m = 0 正常文本（白色字母，黑色背景） m = 1 背景将不被绘制（即，透明） m = 2 视频将被反转（黑色字母，白色背景） m = 5 当前像素将被反转（透明背景）

PULSE pin, width	<p>Will generate a pulse on 'pin' with duration of 'width' ms. 'width' can be a fraction. For example, 0.01 is equal to 10µs and this enables the generation of very narrow pulses.</p> <p>The generated pulse is of the opposite polarity to the state of the I/O pin when the command is executed. For example, if the output is set high the PULSE command will generate a negative going pulse. Notes:</p> <ul style="list-style-type: none"> • 'pin' must be configured as an output. • For a pulse of less than 3 ms the accuracy is $\pm 1 \mu\text{s}$. • For a pulse of 3 ms or more the accuracy is $\pm 0.5 \text{ ms}$. • A pulse of 3 ms or more will run in the background. Up to five different and concurrent pulses can be running in the background and each can have its time changed by issuing a new PULSE command or it can be terminated by issuing a PULSE command with zero for 'width'.
PWM channel, frequency, [dutyA] [,dutyB][,phase][,defer]	<p>There are 8 separate PWM frequencies available (channels 0 to 7) and up to 16 outputs with individually controlled duty cycle. You can output on either PWMnA or PWMnB or both for each channel - no restriction. Duty cycles are specified as a percentage and you can use a negative value to invert the output (-100.0 <= duty <=100.0).</p> <p>Minimum frequency = $(cpuspeed + 1) / (2^{24}) \text{ Hz}$. Maximum speed is OPTION CPUSPEED/4. At very fast speeds the duty cycles will be increasingly limited.</p> <p>Phase is a parameter that causes the waveforms to be centred such that a waveform with a shorter duty cycle starts and ends equal times from a longer one. Use 1 to enable this mode and 0 (or omit) to run as normal</p> <p>The parameter "deferredstart" when set to 1 configures the PWM channels as but does not start the output running. They can be started using the PWM SYNC command. This can be used to avoid any undesirable startup artefacts</p> <p>The PWM command is also capable of driving servos as follows:</p> <pre>PWM 1,50,(position_as_a_percentage * 0.05 + 5)</pre> <p>This initiates the PWM on channels where a deferred start was defined or just syncs existing running channels. However, the power comes in the ability to offset the channels one to another (defined as a percentage of the time period as per the duty cycle - can be a float)</p> <p>You can use an offset of -1 to omit a channel from the synch</p> <p>Stop output on 'channel'</p>
PWM channel, OFF	
RANDOMIZE nbr	<p>Seed the random number generator with 'nbr'.</p> <p>On power up the random number generator is seeded with zero and will generate the same sequence of random numbers each time. To generate a different random sequence each time you must use a different value for 'nbr' (the TIMER function is handy for that).</p>
REFRESH	<p>Initiates an update of the screen for certain black and white displays.</p> <p>These can only be updated a full screen at a time and if OPTION AUTOREFRESH is OFF this command can be used to trigger the write. This applies to the following displays: N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI, ST7920.</p>

PULSE pin, width	<p>将在 'pin' 上生成持续时间为 'width' 毫秒的脉冲。'width' 可以是一个小数。例如，0.01 等于 $10\mu\text{s}$，这使得可以生成非常窄的脉冲。</p> <p>生成的脉冲与执行命令时 I/O 引脚的状态极性相反。例如，如果输出设置为高，PULSE 命令将生成一个负脉冲。注意：</p> <ul style="list-style-type: none"> • '引脚' 必须配置为输出。 • 对于小于 3ms 的脉冲，精度为 $\pm 1\mu\text{s}$。 • 对于 3ms 或更长的脉冲，精度为 $\pm 0.5\text{ms}$。 • 3ms 或更长的脉冲将在后台运行。最多可以同时运行五个不同的脉冲，每个脉冲的时间可以通过发出新的 PULSE 命令进行更改，或者可以通过发出宽度为零的 PULSE 命令终止。
PWM 通道, 频率, [占空比A] [,占空比B][,相位][,延迟]	<p>可用 8 个独立的 PWM 频率（通道 0 到 7），最多可提供 16 个具有单独控制占空比的输出。您可以在每个通道的 PWMnA 或 PWMnB 上输出，或者同时在两者上输出 - 没有限制。占空比以百分比表示，您可以使用负值来反转输出 ($-100.0 \leq \text{duty} \leq 100.0$)。</p> <p>最小频率 = $(\text{cpu速度} + 1) / (2^{24})$ 赫兹。最大速度为选项 CPU 速度/4。在非常快的速度下，占空比将受到越来越多的限制。</p> <p>相位是一个参数，使波形居中，以便占空比较短的波形与较长的波形在开始和结束时相等。使用 1 启用此模式，使用 0（或省略）以正常方式运行。当参数 "deferredstart" 设置为 1 时，配置 PWM 通道，但不启动输出。它们可以通过 PWM SYNC 命令启动。这可以用来避免任何不希望的启动伪影 PWM 命令还能够驱动伺服电机，如下所示：</p> <pre>PWM 1, 50, (position_as_a_percentage * 0.05 + 5)</pre> <p>这会在定义了延迟启动的通道上初始化 PWM，或者仅仅同步现有的运行通道。然而，强大之处在于能够将通道彼此偏移（定义为占空比所对应的周期的百分比 - 可以是浮点数）。</p> <p>您可以使用 -1 的偏移量来省略一个通道的同步。</p> <p>停止‘通道’的输出</p>
PWM 同步 s0 [,s1][,s2][,s3][,s4][,s5][,s6][,s7]	
PWM 通道, 关闭	
RANDOMIZE nbr	<p>用 'nbr' 为随机数生成器设定种子。</p> <p>在上电时，随机数生成器的种子为零，每次都会生成相同的随机数序列。要每次生成不同的随机序列，您必须为 'nbr' 使用不同的值（TIMER 函数对此非常方便）。</p>
刷新	<p>为某些黑白显示器启动屏幕更新。</p> <p>这些只能一次更新整个屏幕，如果 OPTION AUTOREFRESH 为关闭状态，则可以使用此命令触发写入。这适用于以下显示器：N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI, ST7920。</p>

RBOX x, y, w, h [,r] [,c] [,fill]	<p>Draws a box with rounded corners on an attached LCD panel starting at 'x' and 'y' which is 'w' pixels wide and 'h' pixels high.</p> <p>'r' is the radius of the corners of the box. It defaults to 10.</p> <p>'c' specifies the colour and defaults to the default foreground colour if not specified. 'fill' is the fill colour. It can be omitted or set to -1 in which case the box will not be filled.</p> <p>All parameters can be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. 'x', 'y', 'w', and 'h' must all be arrays or all be single variables /constants otherwise an error will be generated. 'r', 'c', and 'fill' can be either arrays or single variables/constants.</p> <p>See the section <i>Graphics Commands and Functions</i> for a definition of the colours and graphics coordinates.</p>
READ variable[, variable]..	<p>Reads values from DATA statements and assigns these values to the named variables. Variable types in a READ statement must match the data types in DATA statements as they are read.</p> <p>Arrays can be used as variables (specified with empty brackets, e.g. a()) and in that case the size of the array is used to determine how many elements are to be read. If the array is multidimensional then the leftmost dimension will be the fastest moving.</p> <p>See also DATA and RESTORE.</p>
READ SAVE or READ RESTORE	<p>READ SAVE will save the virtual pointer used by the READ command to point to the next DATA to be read. READ RESTORE will restore the pointer that was previously saved.</p> <p>This enables subroutines to READ data and then restore the read pointer so as not to disturb other parts of the program that may be reading the same data statements. These commands can be nested.</p>
REM string	<p>REM allows remarks to be included in a program.</p> <p>Note the Microsoft style use of the single quotation mark ('') to denote remarks is also supported and is preferred.</p>
RENAME old\$ AS new\$	<p>Rename a file or a directory from 'old\$' to 'new\$'. Both are strings.</p> <p>A directory path can be used in both 'old\$' and 'new\$'. If the paths differ the file specified in 'old\$' will be moved to the path specified in 'new\$' with the file name as specified.</p>
RESTORE [line]	<p>Resets the line and position counters for the READ statement.</p> <p>If 'line' is specified the counters will be reset to the beginning of the specified line. 'line' can be a line number or label or a variable with these values.</p> <p>If 'line' is not specified the counters will be reset to the start of the program.</p>
RMDIR dir\$	Remove, or delete, the directory 'dir\$' on the default Flash Filesystem or SD Card.
RTC GETTIME	RTC GETTIME will get the current date/time from a PCF8563, DS1307 or DS3231 real time clock and set the internal MMBasic clock accordingly. The date/time can then be retrieved with the DATE\$ and TIME\$ functions.
RTC SETTIME year, month, day, hour, minute, second	RTC SETTIME will set the time in the clock chip. 'hour' must use 24 hour notation. 'year' can be two or four digits. The RTC SETTIME command will also accept a single string argument in the format of dd/mm/yy hh:mm. This means the date/time could be entered by the user using a GUI FORMATBOX with the DATETIME2 format.
RTC SETREG reg, value RTC GETREG reg, var	The RTC SETREG and GETREG commands can be used to set or read the contents of registers within the chip. 'reg' is the register's number, 'value' is the number to store in the register and 'var' is a variable that will receive the number read from the register. These commands are not necessary for normal

RBOX x, y, w, h [, r] [,c] [,fill]	<p>在连接的液晶面板上绘制一个圆角矩形，起始点为 'x' 和 'y'，宽度为 'w' 像素，高度为 'h' 像素。 'r' 是矩形角的半径，默认为 10。 'c' 指定颜色，如果未指定，则默认为默认前景颜色。'fill' 是填充颜色。可以省略或设置为 -1，在这种情况下，框将不会被填充。</p> <p>所有参数可以表示为数组，软件将根据最小数组的尺寸绘制框的数量。'x'、'y'、'w' 和 'h' 必须都是数组或都是单个变量/常量，否则将生成错误。'r'、'c' 和 'fill' 可以是数组或单个变量/常量。</p> <p>请参见章节图形命令和函数以获取颜色和图形坐标的定义。</p>
READ 变量[, 变量]..	<p>从 DATA 语句中读取值，并将这些值分配给命名的变量。READ 语句中的变量类型必须与 DATA 语句中的数据类型匹配。</p> <p>数组可以用作变量（用空括号指定，例如 a()），在这种情况下，数组的大小用于确定要读取多少个元素。如果数组是多维的，则最左边的维度将是移动最快的。</p> <p>另见 DATA 和 RESTORE。</p>
READ SAVE 或 READ RESTORE	<p>READ SAVE 将保存 READ 命令使用的虚拟指针，以指向下一个要读取的数据。READ RESTORE 将恢复之前保存的指针。</p> <p>这使得子例程可以读取数据，然后恢复读取指针，以免干扰程序中可能读取相同数据语句的其他部分。这些命令可以嵌套使用。</p>
REM 字符串	<p>REM 允许在程序中包含备注。 请注意，微软风格使用单引号 (‘) 表示备注也被支持，并且是首选。</p>
RENAME old\$ AS new\$	<p>将文件或目录从 'old\$' 重命名为 'new\$'。两者都是字符串。 目录路径可以在 'old' 和 'new' 中使用。如果路径不同，'old' 中指定的文件将被移动到 'new' 中指定的路径，文件名保持不变。</p>
RESTORE [line]	<p>重置 READ 语句的行和位置计数器。 如果指定了 'line'，计数器将重置为指定行的开头。'line' 可以是行号、标签或具有这些值的变量。 如果未指定 'line'，计数器将重置为程序的开始。</p>
RMDIR dir\$	<p>删除默认 Flash 文件系统或 SD 卡上的目录 'dir\$'。</p>
RTC GETTIME RTC SETTIME 年, 月, 日 , 小时, 分钟, 秒	<p>RTC GETTIME 将从 PCF8563、DS1307 或 DS3231 实时时钟获取当前日期/时间，并相应地设置内部 MMBasic 时钟。然后可以使用 DATE\$ 和 TIME\$ 函数检索日期/时间。</p> <p>RTC SETTIME 将设置时钟芯片中的时间。'小时' 必须使用 24 小时制表示法。'年' 可以是两位或四位数字。RTC SETTIME 命令还可以接受一个字符串参数，格式为 <i>dd/mm/yy hh:mm</i>。这意味着用户可以通过使用 DATETIME2 格式的图形用户界面格式框输入日期/时间。</p>
RTC SETREG reg, value RTC GETREG reg, var	<p>RTC SETREG 和 GETREG 命令可用于设置或读取芯片内寄存器的内容。'reg' 是寄存器的编号，'value' 是要存储在寄存器中的数字，'var' 是一个变量，将接收从寄存器读取的数字。这些命令在正常操作中不是必需的</p>

	<p>operation but they can be used to manipulate special features of the chip (alarms, output signals, etc). They are also useful for storing temporary information in the chip's battery backed RAM.</p> <p>These chips are I²C devices and must be connected to the two I²C pins as specified by OPTION SYSTEM I2C with appropriate pullup resistors.</p> <p>Also see the command OPTION RTC AUTO ENABLE.</p>
RUN or RUN [file\$] [, cmdline\$]	<p>Run a program.</p> <p>If file\$ is not supplied then run the program currently held in program memory.</p> <p>If file\$ is supplied then run the named file from the Flash or SD Card filesystem; if file\$ does not contain a '.BAS' extension then one will be automatically added.</p> <p>If cmdline\$ is supplied then pass its value to the MM.CMDLINE\$ constant of the program when it runs.</p> <p>If cmdline\$ is not supplied then an empty string value is passed to MM.CMDLINE\$.</p> <ul style="list-style-type: none"> • Both file\$ and cmdline\$ may be supplied as string expressions. • Use FLASH RUN n to run a program stored in a Flash Slot. .Use FLASH RUN n to run a program stored in a Flash Slot.
SAVE file\$	<p>Saves the program in the current working directory of the Flash Filesystem or SD Card as 'file\$'. Example: SAVE "TEST.BAS"</p> <p>If an extension is not specified ".BAS" will be added to the file name.</p> <p>See also FLASH SAVE n</p>
SAVE IMAGE file\$ [,x, y, w, h] or SAVE COMPRESSED IMAGE file\$ [,x, y, w, h]	<p>Save the current image on the current LCD panel as a BMP file. The panel must be capable of being read, for example, a ILI9341 based panel or a VIRTUAL_M or VIRTUAL_ panel.</p> <p>'file\$' is the name of the file. If an extension is not specified ".BMP" will be added to the file name. The image is saved as a true colour 24-bit image.</p> <p>'x', 'y', 'w' and 'h' are optional and are the coordinates (x and y are the top left coordinate) and dimensions (width and height) of the area to be saved. If not specified the whole screen will be saved.</p> <p>SAVE COMPRESSED IMAGE will work the same except that RLE compression will be used to reduce the file size..</p>
SEEK [#]fnbr, pos	<p>Will position the read/write pointer in a file that has been opened on the Flash Filesystem or SD Card for RANDOM access to the 'pos' byte.</p> <p>The first byte in a file is numbered one so SEEK #5,1 will position the read/write pointer to the start of the file.</p>
SELECT CASE value CASE testexp [, testexp] ...] <statements> <statements> CASE ELSE <statements> <statements> END SELECT	<p>Executes one of several groups of statements, depending on the value of an expression. 'value' is the expression to be tested. It can be a number or string variable or a complex expression.</p> <p>'testexp' is the value that is to be compared against. It can be:</p> <ul style="list-style-type: none"> • A single expression (i.e. 34, "string" or PIN(4)*5) to which it may equal • A range of values in the form of two single expressions separated by the keyword "TO" (i.e. 5 TO 9 or "aa" TO "cc") • A comparison starting with the keyword "IS" (which is optional). For example: IS > 5, IS <= 10. <p>When a number of test expressions (separated by commas) are used the CASE statement will be true if any one of these tests evaluates to true.</p> <p>If 'value' cannot be matched with a 'testexp' it will be automatically matched to the CASE ELSE. If CASE ELSE is not present the program will not execute any <statements> and continue with the code following the END SELECT.</p> <p>When a match is made the <statements> following the CASE statement will be</p>

	<p>但它们可以用于操作芯片的特殊功能（闹钟、输出信号等）。它们也用于在芯片的电池备份内存中存储临时信息。</p> <p>这些芯片是I²C设备，必须按照OPTION SYSTEM I2C的规定连接到两个I²C引脚，并使用适当的上拉电阻。</p> <p>另请参见命令OPTION RTC 自动启用。</p>
运行 或 运行 [file\$] [, cmdline\$]	<p>运行一个程序。</p> <p>如果未提供file\$，则运行当前保存在程序内存中的程序。</p> <p>如果提供了file\$，则从Flash或SD卡文件系统中运行指定的文件；如果file\$不包含'.BAS'扩展名，则会自动添加一个。</p> <p>如果提供了cmdline\$，则在程序运行时将其值传递给MM.CMDLINE\$常量。</p> <p>如果未提供cmdline\$，则将一个空字符串值传递给MM.CMDLINE\$。</p> <ul style="list-style-type: none"> • file\$和cmdline\$都可以作为字符串表达式提供。 • 使用FLASH RUN n来运行存储在Flash插槽中的程序。 使用FLASH RUN n 运行存储在Flash插槽中的程序。
SAVE file\$	<p>将程序保存到Flash文件系统或SD卡的当前工作目录中，文件名为‘file\$’。示例：SAVE “TEST.BAS”如果未指定扩展名，将会在文件名后添加“.BAS”。</p> <p>另见 FLASH SAVE n</p>
SAVE IMAGE file\$ [,x, y, w, h] 或 SAVE COMPRESSED IMAGE file\$ [,x, y, w, h]	<p>将当前液晶面板上的图像保存为BMP文件。该面板必须能够被读取，例如，基于ILI9341的面板或VIRTUAL_M或VIRTUAL_panel。</p> <p>‘file’是文件的名称。如果未指定扩展名，将会在文件名后添加“.BMP”。图像将以真实颜色的24位图像格式保存。</p> <p>‘x’，‘y’，‘w’和‘h’是可选的，分别表示要保存区域的坐标（x和y为左上角坐标）和尺寸（宽度和高度）。如果未指定，则整个屏幕将被保存。</p> <p>SAVE COMPRESSED IMAGE 的工作方式相同，只是将使用RLE压缩来减少文件大小。</p>
SEEK [#]fnbr, pos	<p>将在Flash文件系统或SD卡上打开的文件中定位读/写指针，以便随机访问‘pos’字节。</p> <p>文件中的第一个字节编号为一，因此SEEK #5,1 将把读/写指针定位到文件的开始。</p>
SELECT CASE value CASE testexp [, testexp] ...] <statements> <statements> CASE ELSE <statements> <statements> END SELECT	<p>根据表达式的值执行多个语句组中的一个。‘value’是要测试的表达式。它可以是一个数字或字符串变量或一个复杂的表达式。</p> <p>‘testexp’是要进行比较的值。它可以是：</p> <ul style="list-style-type: none"> • 一个单一的表达式（即34，“字符串”或PIN(4)*5），它可能相等。 • 一系列值的范围，形式为两个单一表达式，由关键字“TO”分隔（即5 TO 9或“aa” TO “cc”） • 以关键字“IS”开始的比较（这是可选的）。例如：IS > 5, IS < = 10。 <p>当使用多个测试表达式（用逗号分隔）时，如果这些测试中的任何一个求值为真，则CASE语句将为真。</p> <p>如果‘value’无法与‘testexp’匹配，它将自动匹配到CASE ELSE。如果没有CASE ELSE，程序将不会执行任何<statements>，并继续执行END SELECT后的代码。当匹配成功时，<statements>将在CASE语句后执行。</p>

	<p>executed until END SELECT or another CASE is encountered when the program will then continue with the code following the END SELECT.</p> <p>An unlimited number of CASE statements can be used but there must be only one CASE ELSE and that should be the last before the END SELECT.</p> <p>Example:</p> <pre>SELECT CASE nbr% CASE 4, 9, 22, 33 TO 88 statements CASE IS < 4, IS > 88, 5 TO 8 statements CASE ELSE statements END SELECT</pre> <p>Each SELECT CASE must have one and one only matching END SELECT statement. Any number of SELECT...CASE statements can be nested inside the CASE statements of other SELECT...CASE statements.</p>												
SETPIN pin, cfg [, option]	<p>Will configure an external I/O pin. Refer to the section <i>Using the I/O pins</i> for a general description of the PicoMite's input/output capabilities.</p> <p>'pin' is the I/O pin to configure, 'cfg' is the mode that the pin is to be set to and 'option' is an optional parameter. 'cfg' is a keyword and can be any one of the following:</p> <table> <tbody> <tr> <td>OFF</td><td>Not configured or inactive</td></tr> <tr> <td>AIN</td><td>Analog input (i.e. measure the voltage on the input).</td></tr> <tr> <td>DIN</td><td>Digital input If 'option' is omitted the input will be high impedance If 'option' is the keyword "PULLUP" a simulated resistor will be used to pull up the input pin to 3.3V If the keyword "PULLDOWN" is used the pin will be pulled down to zero volts. The pull up/down is a constant current of about 50µA.</td></tr> <tr> <td>FIN</td><td>Frequency input 'option' can be used to specify the gate time (the length of time used to count the input cycles). It can be any number between 10 ms and 100000 ms. Note that the PIN() function will always return the frequency correctly scaled in Hz regardless of the gate time used. If 'option' is omitted the gate time will be 1 second. The pins can be GP6, GP7, GP8 or GP9 (can be changed with OPTION COUNT).</td></tr> <tr> <td>PIN</td><td>Period input 'option' can be used to specify the number of input cycles to average the period measurement over. It can be any number between 1 and 10000. Note that the PIN() function will always return the average period of one cycle correctly scaled in ms regardless of the number of cycles used for the average. If 'option' is omitted the period of just one cycle will be used. The pins can be GP6, GP7, GP8 or GP9 (can be changed with OPTION COUNT).</td></tr> <tr> <td>CIN</td><td>Counting input 'option' can be used to specify which edge triggers the count and if any pullup or pulldown is enabled 1 specifies a rising edge with pulldown, 2 specifies a falling edge with pullup, 3 specifies that both a falling and rising edge will trigger a count with no pullup or pulldown applied, 4 specifies both edges but with a pulldown and 5 specifies both edges but with a pullup applied.</td></tr> </tbody> </table>	OFF	Not configured or inactive	AIN	Analog input (i.e. measure the voltage on the input).	DIN	Digital input If 'option' is omitted the input will be high impedance If 'option' is the keyword "PULLUP" a simulated resistor will be used to pull up the input pin to 3.3V If the keyword "PULLDOWN" is used the pin will be pulled down to zero volts. The pull up/down is a constant current of about 50µA.	FIN	Frequency input 'option' can be used to specify the gate time (the length of time used to count the input cycles). It can be any number between 10 ms and 100000 ms. Note that the PIN() function will always return the frequency correctly scaled in Hz regardless of the gate time used. If 'option' is omitted the gate time will be 1 second. The pins can be GP6, GP7, GP8 or GP9 (can be changed with OPTION COUNT).	PIN	Period input 'option' can be used to specify the number of input cycles to average the period measurement over. It can be any number between 1 and 10000. Note that the PIN() function will always return the average period of one cycle correctly scaled in ms regardless of the number of cycles used for the average. If 'option' is omitted the period of just one cycle will be used. The pins can be GP6, GP7, GP8 or GP9 (can be changed with OPTION COUNT).	CIN	Counting input 'option' can be used to specify which edge triggers the count and if any pullup or pulldown is enabled 1 specifies a rising edge with pulldown, 2 specifies a falling edge with pullup, 3 specifies that both a falling and rising edge will trigger a count with no pullup or pulldown applied, 4 specifies both edges but with a pulldown and 5 specifies both edges but with a pullup applied.
OFF	Not configured or inactive												
AIN	Analog input (i.e. measure the voltage on the input).												
DIN	Digital input If 'option' is omitted the input will be high impedance If 'option' is the keyword "PULLUP" a simulated resistor will be used to pull up the input pin to 3.3V If the keyword "PULLDOWN" is used the pin will be pulled down to zero volts. The pull up/down is a constant current of about 50µA.												
FIN	Frequency input 'option' can be used to specify the gate time (the length of time used to count the input cycles). It can be any number between 10 ms and 100000 ms. Note that the PIN() function will always return the frequency correctly scaled in Hz regardless of the gate time used. If 'option' is omitted the gate time will be 1 second. The pins can be GP6, GP7, GP8 or GP9 (can be changed with OPTION COUNT).												
PIN	Period input 'option' can be used to specify the number of input cycles to average the period measurement over. It can be any number between 1 and 10000. Note that the PIN() function will always return the average period of one cycle correctly scaled in ms regardless of the number of cycles used for the average. If 'option' is omitted the period of just one cycle will be used. The pins can be GP6, GP7, GP8 or GP9 (can be changed with OPTION COUNT).												
CIN	Counting input 'option' can be used to specify which edge triggers the count and if any pullup or pulldown is enabled 1 specifies a rising edge with pulldown, 2 specifies a falling edge with pullup, 3 specifies that both a falling and rising edge will trigger a count with no pullup or pulldown applied, 4 specifies both edges but with a pulldown and 5 specifies both edges but with a pullup applied.												

	<p>将执行直到遇到 END SELECT 或另一个 CASE，此时程序将继续执行 END SELECT 后的代码。可以使用无限数量的 CASE 语句，但必须只有一个 CASE ELSE，并且它应在 END SELECT 之前的最后一个。示例：</p> <pre> 选择案例 nbr% 案例 4, 9, 22, 33 到 88 语句 案例 小于 4, 大于 88, 5 到 8 语句 案例 否则 语句 结束选择 </pre> <p>每个选择案例必须有且仅有一个匹配的结束选择语句。任何数量的选择...案例语句可以嵌套在其他选择...案例语句的案例语句中。</p>												
设置引脚引脚, cfg [, 选项]	<p>将配置一个外部 I/O 引脚。请参阅章节使用 <i>I/O</i> 引脚以获取 PicoMite 输入/输出能力的一般描述。</p> <p>'引脚' 是要配置的 I/O 引脚，'cfg' 是要设置的引脚模式，'选项' 是一个可选参数。'cfg' 是一个关键字，可以是以下任意一个：</p> <table> <tr> <td>关闭</td><td>未配置或不活动</td></tr> <tr> <td>模拟输入</td><td>模拟输入（即测量输入上的电压）。</td></tr> <tr> <td>DIN</td><td>数字输入 如果省略'选项'，输入将处于高阻抗状态。如果'选项'为关键字"PULLUP"，将使用模拟电阻将输入引脚拉高至3.3V。如果使用关键字"PULLDOWN"，引脚将被拉低至零伏。上拉/下拉的恒定电流约为50μA。</td></tr> <tr> <td>FIN</td><td>频率输入 '选项'可用于指定门时间（用于计数输入周期的时间长度）。它可以是10毫秒到100000毫秒之间的任何数字。请注意，PIN()函数将始终正确返回以赫兹为单位缩放的频率，无论使用的门时间如何。如果省略'选项'，门时间将为1秒。 引脚可以是GP6、GP7、GP8或GP9（可以通过OPTION COUNT更改）。</td></tr> <tr> <td>引脚</td><td>周期输入 '选项' 可用于指定用于平均周期测量的输入周期数量。它可以是1到10000之间的任何数字。请注意，PIN()函数将始终返回正确缩放为毫秒的一个周期的平均周期，无论用于平均的周期数量如何。如果省略‘选项’，将使用仅一个周期的周期。 引脚可以是GP6、GP7、GP8或GP9（可以通过选项计数进行更改）。</td></tr> <tr> <td>CIN</td><td>计数输入 '选项'可用于指定哪个边缘触发计数以及是否启用了任何上拉或下拉 1指定带下拉的上升沿， 2指定带上拉的下降沿， 3指定在不应用上拉或下拉的情况下，下降沿和上升沿都将触发计数，4指定两个边缘，但带下拉。 5指定了两个边缘，但应用了上拉电阻。</td></tr> </table>	关闭	未配置或不活动	模拟输入	模拟输入（即测量输入上的电压）。	DIN	数字输入 如果省略'选项'，输入将处于高阻抗状态。如果'选项'为关键字"PULLUP"，将使用模拟电阻将输入引脚拉高至3.3V。如果使用关键字"PULLDOWN"，引脚将被拉低至零伏。上拉/下拉的恒定电流约为50μA。	FIN	频率输入 '选项'可用于指定门时间（用于计数输入周期的时间长度）。它可以是10毫秒到100000毫秒之间的任何数字。请注意，PIN()函数将始终正确返回以赫兹为单位缩放的频率，无论使用的门时间如何。如果省略'选项'，门时间将为1秒。 引脚可以是GP6、GP7、GP8或GP9（可以通过OPTION COUNT更改）。	引脚	周期输入 '选项' 可用于指定用于平均周期测量的输入周期数量。它可以是1到10000之间的任何数字。请注意，PIN()函数将始终返回正确缩放为毫秒的一个周期的平均周期，无论用于平均的周期数量如何。如果省略‘选项’，将使用仅一个周期的周期。 引脚可以是GP6、GP7、GP8或GP9（可以通过选项计数进行更改）。	CIN	计数输入 '选项'可用于指定哪个边缘触发计数以及是否启用了任何上拉或下拉 1指定带下拉的上升沿， 2指定带上拉的下降沿， 3指定在不应用上拉或下拉的情况下，下降沿和上升沿都将触发计数，4指定两个边缘，但带下拉。 5指定了两个边缘，但应用了上拉电阻。
关闭	未配置或不活动												
模拟输入	模拟输入（即测量输入上的电压）。												
DIN	数字输入 如果省略'选项'，输入将处于高阻抗状态。如果'选项'为关键字"PULLUP"，将使用模拟电阻将输入引脚拉高至3.3V。如果使用关键字"PULLDOWN"，引脚将被拉低至零伏。上拉/下拉的恒定电流约为50μA。												
FIN	频率输入 '选项'可用于指定门时间（用于计数输入周期的时间长度）。它可以是10毫秒到100000毫秒之间的任何数字。请注意，PIN()函数将始终正确返回以赫兹为单位缩放的频率，无论使用的门时间如何。如果省略'选项'，门时间将为1秒。 引脚可以是GP6、GP7、GP8或GP9（可以通过OPTION COUNT更改）。												
引脚	周期输入 '选项' 可用于指定用于平均周期测量的输入周期数量。它可以是1到10000之间的任何数字。请注意，PIN()函数将始终返回正确缩放为毫秒的一个周期的平均周期，无论用于平均的周期数量如何。如果省略‘选项’，将使用仅一个周期的周期。 引脚可以是GP6、GP7、GP8或GP9（可以通过选项计数进行更改）。												
CIN	计数输入 '选项'可用于指定哪个边缘触发计数以及是否启用了任何上拉或下拉 1指定带下拉的上升沿， 2指定带上拉的下降沿， 3指定在不应用上拉或下拉的情况下，下降沿和上升沿都将触发计数，4指定两个边缘，但带下拉。 5指定了两个边缘，但应用了上拉电阻。												

	<p>If 'option' is omitted a rising edge will trigger the count and a pulldown is enabled. The pins can be GP6, GP7, GP8 or GP9 (can be changed with OPTION COUNT).</p> <p>DOUT Digital output 'option' is not used in this mode.</p> <p>The functions PIN() and PORT() can also be used to return the value on one or more output pins. See the function PIN() for reading inputs and the statement PIN()= for setting an output. See the command below if an interrupt is configured.</p>								
SETPIN pin, cfg, target [, option]	<p>Will configure 'pin' to generate an interrupt according to 'cfg'. Any I/O pin capable of digital input can be configured to generate an interrupt with a maximum of ten interrupts configured at any one time.</p> <p>'cfg' is a keyword and can be any one of the following:</p> <table> <tr> <td>OFF</td><td>Not configured or inactive</td></tr> <tr> <td>INTH</td><td>Interrupt on low to high input</td></tr> <tr> <td>INTL</td><td>Interrupt on high to low input</td></tr> <tr> <td>INTB</td><td>Interrupt on both (i.e. any change to the input)</td></tr> </table> <p>'target' is a user defined subroutine which will be called when the event happens. Return from the interrupt is via the END SUB or EXIT SUB commands. 'option' can be the keywords "PULLUP" or "PULLDOWN" as specified for a normal input pin (SETPIN pin DIN). If 'option' is omitted the input will be high impedance.</p> <p>This mode also configures the pin as a digital input so the value of the pin can always be retrieved using the function PIN().</p> <p>Refer to the section <i>Using the I/O pins</i> for a general description of the PicoMite's input/output capabilities.</p>	OFF	Not configured or inactive	INTH	Interrupt on low to high input	INTL	Interrupt on high to low input	INTB	Interrupt on both (i.e. any change to the input)
OFF	Not configured or inactive								
INTH	Interrupt on low to high input								
INTL	Interrupt on high to low input								
INTB	Interrupt on both (i.e. any change to the input)								
SETPIN GP25, DOUT HEARTBEAT	<p>This version of SETPIN controls the on-board LED.</p> <p>If it is configured as DOUT then it can be switched on and off under program control.</p> <p>If configured as HEARTBEAT then it will flash 1s on, 1s off continually while powered. This is the default state and will be restored to this when the user program stops running.</p>								
SETPIN p1[, p2 [, p3]], device	<p>These commands are used for the pin allocation for special devices.</p> <p>Pins must be chosen from the pin designation diagram and must be allocated before the devices can be used. Note that the pins (e.g. rx, tx, etc) can be declared in any order and that the pins can be referred to by using their pin number (e.g. 1, 2) or GP number (e.g. GP0, GP1).</p>								
SETPIN rx, tx, COM1	<p>Allocate the pins to be used for serial port COM1.</p> <p>Valid pins are RX: GP1, GP13 or GP17 TX: GP0, GP12, GP16 or GP28</p>								
SETPIN rx, tx, COM2	<p>Allocate the pins to be used for serial port COM2.</p> <p>Valid pins are RX: GP5, GP9 or GP21 TX: GP4, GP8 or GP20</p>								
SETPIN rx, tx, clk, SPI	<p>Allocate the pins to be used for SPI port SPI.</p> <p>Valid pins are RX: GP0, GP4, GP16 or GP20 TX: GP3, GP7 or GP19 CLK: GP2, GP6 or GP18</p>								
SETPIN rx, tx, clk, SPI2	<p>Allocate the pins to be used for SPI port SPI2.</p> <p>Valid pins are RX: GP8, GP12 or GP28 TX: GP11, GP15 or GP27</p>								

	<p>如果省略‘选项’，则上升沿将触发计数，并启用下拉电阻。引脚可以是GP6、GP7、GP8或GP9(可以通过选项计数进行更改)。</p> <p>DOUT 数字输出 在此模式下不使用‘选项’。 函数PIN()和PORT()也可以用于返回一个或多个输出引脚上的值。请参阅函数PIN()以读取输入，以及语句PIN()=以设置输出。如果配置了中断，请参阅下面的命令。</p>								
SETPIN 引脚, cfg, 目标 [, 选项]	<p>将配置‘引脚’以根据‘cfg’生成中断。任何能够进行数字输入的I/O引脚都可以配置为生成中断，同时最多可以配置十个中断。</p> <p>'cfg'是一个关键字，可以是以下任意一个：</p> <table> <tr> <td>关闭</td><td>未配置或未激活</td></tr> <tr> <td>INTH</td><td>低到高输入的中断</td></tr> <tr> <td>INTL</td><td>高到低输入的中断</td></tr> <tr> <td>INTB</td><td>在两者上中断（即输入的任何变化）‘target’ 是一个用户定义的子程序，当事件发生时将被调用。从中断返回是通过END SUB或EXIT SUB命令。'option'可以是关键字"PULLUP"或"PULLDOWN"，如为普通输入引脚（SETPIN pin DIN）所指定。如果省略'option'，输入将为高阻抗。</td></tr> </table> <p>此模式还将引脚配置为数字输入，因此引脚的值始终可以使用函数PIN()检索。 请参阅部分使用I/O引脚以获取PicoMite输入/输出能力的一般描述。</p>	关闭	未配置或未激活	INTH	低到高输入的中断	INTL	高到低输入的中断	INTB	在两者上中断（即输入的任何变化）‘target’ 是一个用户定义的子程序，当事件发生时将被调用。从中断返回是通过END SUB或EXIT SUB命令。'option'可以是关键字"PULLUP"或"PULLDOWN"，如为普通输入引脚（SETPIN pin DIN）所指定。如果省略'option'，输入将为高阻抗。
关闭	未配置或未激活								
INTH	低到高输入的中断								
INTL	高到低输入的中断								
INTB	在两者上中断（即输入的任何变化）‘target’ 是一个用户定义的子程序，当事件发生时将被调用。从中断返回是通过END SUB或EXIT SUB命令。'option'可以是关键字"PULLUP"或"PULLDOWN"，如为普通输入引脚（SETPIN pin DIN）所指定。如果省略'option'，输入将为高阻抗。								
SETPIN GP25, DOUT 心跳	<p>此版本的SETPIN控制板载LED。 如果配置为DOUT，则可以在程序控制下开关。</p> <p>如果配置为HEARTBEAT，则在通电时将持续闪烁1秒开，1秒关。这是默认状态，当用户程序停止运行时将恢复到此状态。</p>								
SETPIN p1[, p2 [, p3]], 设备	<p>这些命令用于特殊设备的引脚分配。 引脚必须从引脚指定图中选择，并且必须在设备使用之前进行分配。请注意，引脚（例如rx、tx等）可以按任何顺序声明，并且可以通过使用其引脚编号（例如1、2）或GP编号（例如GP0、GP1）来引用引脚。</p>								
SETPIN rx, tx, COM1	<p>分配用于串行端口COM1的引脚。</p> <table> <tr> <td>有效引脚为</td> <td>RX: GP1, GP13 或 GP17</td> </tr> <tr> <td></td> <td>TX: GP0, GP12, GP16 或 GP28</td> </tr> </table>	有效引脚为	RX: GP1, GP13 或 GP17		TX: GP0, GP12, GP16 或 GP28				
有效引脚为	RX: GP1, GP13 或 GP17								
	TX: GP0, GP12, GP16 或 GP28								
SETPIN rx, tx, COM2	<p>分配用于串行端口COM2的引脚。</p> <table> <tr> <td>有效引脚为</td> <td>RX: GP5, GP9 或 GP21</td> </tr> <tr> <td></td> <td>TX: GP4, GP8 或 GP20</td> </tr> </table>	有效引脚为	RX: GP5, GP9 或 GP21		TX: GP4, GP8 或 GP20				
有效引脚为	RX: GP5, GP9 或 GP21								
	TX: GP4, GP8 或 GP20								
SETPIN rx, tx, clk, SPI	<p>分配用于SPI端口SPI的引脚。</p> <table> <tr> <td>有效引脚为</td> <td>RX: GP0, GP4, GP16 或 GP20</td> </tr> <tr> <td></td> <td>TX: GP3, GP7 或 GP19</td> </tr> <tr> <td></td> <td>CLK: GP2, GP6 或 GP18</td> </tr> </table>	有效引脚为	RX: GP0, GP4, GP16 或 GP20		TX: GP3, GP7 或 GP19		CLK: GP2, GP6 或 GP18		
有效引脚为	RX: GP0, GP4, GP16 或 GP20								
	TX: GP3, GP7 或 GP19								
	CLK: GP2, GP6 或 GP18								
SETPIN rx, tx, clk, SPI2	<p>分配用于SPI端口SPI2的引脚。</p> <table> <tr> <td>有效引脚为</td> <td>RX: GP8, GP12 或 GP28</td> </tr> <tr> <td></td> <td>TX: GP11, GP15 或 GP27</td> </tr> </table>	有效引脚为	RX: GP8, GP12 或 GP28		TX: GP11, GP15 或 GP27				
有效引脚为	RX: GP8, GP12 或 GP28								
	TX: GP11, GP15 或 GP27								

	CLK: GP10, GP14 or GP26
SETPIN sda, scl, I2C	Allocate the pins to be used for the I ² C port I2C. Valid pins are SDA: GP0, GP4, GP8, GP12, GP16, GP20 or GP28 SCL: GP1, GP5, GP9, GP13, GP17 or GP21
SETPIN sda, scl, I2C2	Allocate the pins to be used for the I ² C port I2C2. Valid pins are SDA: GP2, GP6, GP10, GP14, GP18, GP22 or GP26 SCL: GP3, GP7, GP11, GP15, GP19 or GP27
SETPIN pin, PWM[nx]	Allocate pin to PWMnx 'n' is the PWM number (0 to 7) and 'x' and is the channel (A or B). n and x are optional. The setpin can be changed until the PWM command is issued. At that point the pin becomes locked to PWM until PWMn,OFF is issued.
SETPIN pin, IR	Allocate pins for InfraRed (IR) communications (can be any pin).
SETPIN pin, PIOn	Reserve pin for use by a PIO0 or PIO1 (see Appendix F for PIO details).
SETTICK period, target [, nbr]	This will setup a periodic interrupt (or "tick"). Four tick timers are available ('nbr' = 1, 2, 3 or 4). 'nbr' is optional and if not specified timer number 1 will be used. The time between interrupts is 'period' milliseconds and 'target' is the interrupt subroutine which will be called when the timed event occurs. The period can range from 1 to 2147483647 ms (about 24 days). These interrupts can be disabled by setting 'period' to zero (i.e. SETTICK 0, 0, 3 will disable tick timer number 3).
SETTICK PAUSE, target [, nbr] or SETTICK RESUME, target [, nbr]	Pause or resume the specified timer. When paused the interrupt is delayed but the current count is maintained.
SORT array() [,indexarray()] [,flags] [,startposition] [,elementstosort]	This command takes an array of any type (integer, float or string) and sorts it into ascending order in place. It has an optional parameter 'indexarray%()'. If used this must be an integer array of the same size as the array to be sorted. After the sort this array will contain the original index position of each element in the array being sorted before it was sorted. Any data in the array will be overwritten. This allows connected arrays to be sorted. See the section Sorting Data in the tutorial Programming with the Colour Maximite 2 for an example. The 'flag' parameter is optional and valid flag values are: bit0: 0 (default if omitted) normal sort - 1 reverse sort bit1: 0 (default) case dependent - 1 sort is case independent (strings only). The optional 'startposition' defines which element in the array to start the sort. Default is 0 (OPTION BASE 0) or 1 (OPTION BASE 1) The optional 'elementstosort' defines how many elements in the array should be sorted. The default is all elements after the startposition. Any of the optional parameters may be omitted so, for example, to sort just the first 50 elements of an array you could use: <code style="padding-left: 40px;">SORT array(), , , , 50</code>
SPI OPEN speed, mode, bits or SPI READ nbr, array() or SPI WRITE nbr, data1, data2,	Communications via an SPI channel. See Appendix D for the details. 'nbr' is the number of data items to send or receive 'data1', 'data2', etc can be float or integer and in the case of WRITE can be a constant or expression. If 'string\$' is used 'nbr' characters will be sent.

设置引脚 sda, scl, I2C	CLK: GP10, GP14 或 GP26 分配用于 I ² C 端口 I2C 的引脚。 有效引脚为 SDA: GP0, GP4, GP8, GP12, GP16, GP20 或 GP28 SCL: GP1, GP5, GP9, GP13, GP17 或 GP21
设置引脚 sda, scl, I2C2	分配用于 I ² C 端口 I2C2 的引脚。 有效引脚为 SDA: GP2, GP6, GP10, GP14, GP18, GP22 或 GP26 SCL: GP3, GP7, GP11, GP15, GP19 或 GP27
设置引脚 pin, PWM[nx]	分配引脚到 PWM _n x 'n' 是 PWM 数字 (0 到 7) , 'x' 是通道 (A 或 B) 。 n 和 x 是可选的。 在发出PWM命令之前，可以更改setpin。此时，引脚将锁定为PWM，直到发出PWM _n ,OFF命令。
SETPIN 引脚, IR SETPIN 引脚, PIO _n	为红外 (IR) 通信分配引脚 (可以是任何引脚) 。 保留引脚供PIO0或PIO1使用 (有关PIO详细信息，请参见附录F) 。
SETTICK 周期, 目标 [, 编号]	这将设置一个周期性中断 (或称为“滴答”)。 可用的滴答定时器有四个 ('编号'=1, 2, 3或4) 。 '编号'是可选的，如果未指定，将使用定时器编号1。 中断之间的时间为‘周期’毫秒，‘目标’是当定时事件发生时将被调用的中断子程序。 周期可以在1到2147483647毫秒之间 (约24天)。 通过将‘周期’设置为零，可以禁用这些中断 (即SETTICK 0, 0, 3将禁用滴答定时器编号3) 。
SETTICKPAUSE, 目标 [, 编号] 或 SETTICKRESUME, 目标 [, 编号]	暂停或恢复指定的定时器。当暂停时，中断被延迟，但当前计数保持不变。
SORT array() [,indexarray()][,flags] [,startposition] [待排序元素]	此命令接受任何类型 (整数、浮点数或字符串) 的数组，并将其就地排序为升序。 它有一个可选参数‘indexarray%()’。如果使用，则必须是与要排序的数组大小相同的整数数组。排序后，此数组将包含在排序之前每个元素在被排序数组中的原始索引位置。数组中的任何数据将被覆盖。这允许连接的数组被排序。请参见教程中关于排序数据的部分与颜色最大化 2 编程的示例。 ‘标志’参数是可选的，有效的标志值为： bit0: 0 (如果省略则为默认值) 正常排序 - 1 反向排序 bit1: 0 (默认) 区分大小写 - 1 排序不区分大小写 (仅限字符串)。 可选的‘startposition’定义了在数组中从哪个元素开始排序。 默认值为 0 (OPTION BASE 0) 或 1 (OPTION BASE 1) 可选的‘elementstosort’定义了数组中应该排序多少个元素。默认是从 startposition 开始的所有元素。 任何可选参数都可以省略，因此，例如，要仅排序数组的前 50 个元素，可以使用： SORT array(), , , , 50
SPI OPEN speed, mode, bits 或 SPI READ nbr, array() 或 SPI WRITE nbr, data1, data2,	通过 SPI 通道进行通信。有关详细信息，请参见附录D。 'nbr' 是要发送或接收的数据项数量 'data1', 'data2' 等可以是浮点数或整数，在 WRITE 的情况下可以是常量或表达式。 如果使用 'string'，将发送 'nbr' 个字符。

data3, ... etc or SPI WRITE nbr, string\$ or SPI WRITE nbr, array() or SPI CLOSE	<p>'array' must be a single dimension float or integer array and 'nbr' elements will be sent or received.</p>
SPI2	<p>The same set of commands as for SPI (above) but applying to the second SPI channel.</p>
STATIC variable [, variables] See DIM for the full syntax.	<p>Defines a list of variable names which are local to the subroutine or function. These variables will retain their value between calls to the subroutine or function (unlike variables created using the LOCAL command).</p> <p>This command uses exactly the same syntax as DIM. The only difference is that the length of the variable name created by STATIC and the length of the subroutine or function name added together cannot exceed 31 characters.</p> <p>Static variables can be initialised to a value. This initialisation will take effect only on the first call to the subroutine (not on subsequent calls).</p>
SUB xxx (arg1 [,arg2, ...]) <statements> <statements> END SUB	<p>Defines a callable subroutine. This is the same as adding a new command to MMBasic while it is running your program.</p> <p>'xxx' is the subroutine name and it must meet the specifications for naming a variable.</p> <p>'arg1', 'arg2', etc are the arguments or parameters to the subroutine. An array is specified by using empty brackets. i.e. arg3(). The type of the argument can be specified by using a type suffix (i.e. arg1\$) or by specifying the type using AS <type> (i.e. arg1 AS STRING).</p> <p>Every definition must have one END SUB statement. When this is reached the program will return to the next statement after the call to the subroutine. The command EXIT SUB can be used for an early exit.</p> <p>You use the subroutine by using its name and arguments in a program just as you would a normal command. For example: MySub a1, a2</p> <p>When the subroutine is called each argument in the caller is matched to the argument in the subroutine definition. These arguments are available only inside the subroutine. Subroutines can be called with a variable number of arguments. Any omitted arguments in the subroutine's list will be set to zero or a null string.</p> <p>Arguments in the caller's list that are a variable and have the correct type will be passed by reference to the subroutine. This means that any changes to the corresponding argument in the subroutine will also be copied to the caller's variable and therefore may be accessed after the subroutine has ended. Arrays are passed by specifying the array name with empty brackets (e.g. arg()) and are always passed by reference. Brackets around the argument list in both the caller and the definition are optional.</p>
SYNC time% [,period] or SYNC	<p>The SYNC command allows the user to implement very precisely timed repeated actions (1-2 microseconds accuracy).</p> <p>To enable this the command is first called with the parameter time%. This sets up a repeating clock for time% microseconds. The optional parameter 'period' modifies the time and can be "U" for microseconds, "M" for milliseconds or "S" for seconds.</p> <p>Once the clock is set up the program is synchronised to it using the SYNC command without parameters. This waits for the clock period to expire. For periods below 2milliseconds this is non-interruptible. Above two milliseconds the program will respond to Ctrl-C but not any MMBasic interrupts.</p> <p>Typical use is to set the clock outside of a loop and then at the top of the loop</p>

data3, ... 等 或 SPI WRITE nbr, string\$ 或 SPI WRITE nbr, array() 或 SPI CLOSE	'array' 必须是单维的浮点数或整数数组，将发送或接收 'nbr' 个元素。
SPI2	与 SPI (上述) 相同的一组命令，但适用于第二个 SPI 通道。
STATIC 变量 [, 变量] 请参见 DIM 以获取完整语法。	定义一组变量名称，这些名称在子程序或函数中是局部的。 这些变量将在对子程序或函数的调用之间保留其值（与使用 LOCAL 命令创建的变量不同）。 此命令使用与 DIM 完全相同的语法。唯一的区别是由 STATIC 创建的变量名称的长度与添加的子程序或函数名称的长度之和不能超过 31 个字符。 静态变量可以初始化为一个值。此初始化仅在第一次调用子程序时生效（而不是在后续调用时）。
SUB xxx (arg1 [,arg2, ...]) <语句> <语句> 结束子程序	定义一个可调用的子程序。这与在MMBasic运行程序时添加一个新命令是相同的。 'xxx' 是子程序名称，必须符合变量命名的规范。 'arg1'、'arg2' 等是传递给子程序的参数。数组通过使用空括号来指定，即 arg3()。参数的类型可以通过使用类型后缀（即 arg1\$）或通过指定类型来定义 AS <类型>（即 arg1 AS STRING）。 每个定义必须有一个结束子程序语句。当达到此处时，程序将返回到调用子程序后的下一个语句。命令 EXIT SUB 可用于提前退出。 在程序中使用子程序时，您可以像使用普通命令一样使用其名称和参数。例如：MySub a1, a2 当调用子程序时，调用者中的每个参数与子程序定义中的参数相匹配。这些参数仅在子程序内部可用。子程序可以使用可变数量的参数进行调用。子程序列表中省略的任何参数将被设置为零或空字符串。 调用者列表中的变量参数及其类型正确的参数将通过引用传递给子程序。这意味着对子程序中相应参数的任何更改也将复制到调用者的变量中，因此在子程序结束后仍然可以访问。数组通过指定数组名称和空括号（例如 arg()）进行传递，并且始终通过引用传递。调用者和定义中的参数列表括号是可选的。
SYNC time% [,period] 或 SYNC	SYNC命令允许用户实现非常精确的定时重复操作（1-2微秒的精度）。 要启用此功能，首先使用参数time% 调用该命令。这会设置一个重复的时钟，持续time%微秒。可选参数‘period’修改时间，可以是“U”表示微秒，“M”表示毫秒，或“S”表示秒。 一旦时钟设置完成，程序将使用不带参数的SYNC命令与之同步。这将等待时钟周期到期。对于低于2毫秒的周期，这是不可中断的。在两毫秒以上，程序将响应Ctrl-C，但不会响应任何MMBasic中断。 典型用法是在循环外设置时钟，然后在循环的顶部

	<p>call the SYNC command without parameters. This means the contents of the loop will be executed exactly once for each clock period set.</p> <p>For example, the following would drive a servo with the required precise 50Hz timing:</p> <pre>SYNC 20, M DO SYNC PULSE GP0, n LOOP</pre>
TEMPR START pin [, precision]	<p>This command can be used to start a conversion running on a DS18B20 temperature sensor connected to 'pin'.</p> <p>Normally the TEMP() function alone is sufficient to make a temperature measurement so usage of this command is optional.</p> <p>This command will start the measurement on the temperature sensor. The program can then attend to other duties while the measurement is running and later use the TEMP() function to get the reading. If the TEMP() function is used before the conversion time has completed the function will wait for the remaining conversion time before returning the value.</p> <p>Any number of these conversions (on different pins) can be started and be running simultaneously.</p> <p>'precision' is the resolution of the measurement and is optional. It is a number between 0 and 3 meaning:</p> <ul style="list-style-type: none"> 0 = 0.5°C resolution, 100 ms conversion time. 1 = 0.25°C resolution, 200 ms conversion time (this is the default). 2 = 0.125°C resolution, 400 ms conversion time. 3 = 0.0625°C resolution, 800 ms conversion time.
TEXT x, y, string\$ [,alignment\$] [, font] [, scale] [, c] [, bc]	<p>Displays a string on an attached LCD panel starting at 'x' and 'y'.</p> <p>'string\$' is the string to be displayed. Numeric data should be converted to a string and formatted using the Str\$() function.</p> <p>'alignment\$' is a string expression or string variable consisting of 0, 1 or 2 letters where the first letter is the horizontal alignment around 'x' and can be L, C or R for LEFT, CENTER, RIGHT and the second letter is the vertical alignment around 'y' and can be T, M or B for TOP, MIDDLE, BOTTOM. The default alignment is left/top.</p> <p>For example. "CM" will centre the text vertically and horizontally.</p> <p>The 'alignment\$' string can be a constant (e.g. "CM") or it can be a string variable. For backwards compatibility with earlier versions of MMBasic the string can also be unquoted (e.g. CM).</p> <p>In the PicoMite a third letter can be used in the alignment string to indicate the rotation of the text. This can be 'N' for normal orientation, 'V' for vertical text with each character under the previous running from top to bottom, 'I' the text will be inverted (i.e. upside down), 'U' the text will be rotated counter clockwise by 90° and 'D' the text will be rotated clockwise by 90°</p> <p>'font' and 'scale' are optional and default to that set by the FONT command.</p> <p>'c' is the drawing colour and 'bc' is the background colour. They are optional and default to the current foreground and background colours.</p> <p>See the section <i>Graphics Commands and Functions</i> for a definition of the colours and graphics coordinates.</p>

	<p>调用没有参数的SYNC命令。这意味着循环的内容将在每个设置的时钟周期内执行一次。例如，以下代码将以所需的精确50 Hz时序驱动伺服：</p> <pre>SYNC 20, M DO SYNC PULSE GP0, n LOOP</pre>
TEMPR START 引脚 [, 精度]	<p>此命令可用于启动连接到'引脚'的DS18B20温度传感器上的转换。</p> <p>通常，单独使用TEMPR()函数就足以进行温度测量，因此使用此命令是可选的。</p> <p>此命令将启动温度传感器上的测量。程序可以在测量进行时处理其他任务，稍后使用TEMPR()函数获取读数。如果在转换时间完成之前使用TEMPR()函数，该函数将等待剩余的转换时间，然后再返回值。</p> <p>可以同时启动任意数量的这些转换（在不同的引脚上）并使其运行。</p> <p>'精度'是测量的分辨率，属于可选项。它是一个介于0和3之间的数字，表示：</p> <ul style="list-style-type: none"> 0 = 0.5°C 分辨率，100毫秒转换时间。 1 = 0.25°C 分辨率，200毫秒转换时间（这是默认值）。 2 = 0.125°C 分辨率，400毫秒转换时间。 3 = 0.0625°C 分辨率，800毫秒转换时间。
TEXT x, y, string\$ [,alignment\$] [, font] [, scale] [, c] [, bc]	<p>在连接的液晶面板上显示一个字符串，从 'x' 和 'y' 开始。</p> <p>'string\$' 是要显示的字符串。数值数据应转换为字符串，并使用 Str\$() 函数格式化。</p> <p>'alignment\$' 是一个字符串表达式或字符串变量，由0、1或2个字母组成，第一个字母表示在 'x' 周围的水平对齐方式，可以是 L、C 或 R，分别代表左、居中、右；第二个字母表示在 'y' 周围的垂直对齐方式，可以是 T、M 或 B，分别代表顶部、中间、底部。默认对齐方式为左/上。</p> <p>例如。“CM”将文本垂直和水平居中。</p> <p>'alignment' 字符串可以是常量（例如“CM”），也可以是字符串变量。为了与早期版本的MMBasic向后兼容，该字符串也可以不加引号（例如CM）。</p> <p>在PicoMite中，排列字符串中可以使用第三个字母来指示文本的旋转。这可以是'N'表示正常方向，'V'表示垂直文本，每个字符在上一个字符的下方，从上到下排列，'T'表示文本将被反转（即倒置），'U'表示文本将逆时针旋转90°，'D'表示文本将顺时针旋转90°，'font'和'scale'是可选的，默認為FONT命令设置的值。</p> <p>'颜色'是绘图颜色，'背景色'是背景颜色。它们是可选的，默認為当前的前景和背景颜色。</p> <p>请参见章节图形命令和函数以获取颜色和图形坐标的定义。</p>

TIME\$ = "HH:MM:SS" or TIME\$ = "HH:MM" or TIME\$ = "HH"	Sets the time of the internal clock. MM and SS are optional and will default to zero if not specified. For example TIME\$ = "14:30" will set the clock to 14:30 with zero seconds. With OPTION RTC AUTO ENABLE the picomite starts with the TIME\$ programmed in RTC. Without OPTION RTC AUTO ENABLE the picomite starts with TIME\$="00:00:00"
TIMER = msec	Resets the timer to a number of milliseconds. Normally this is just used to reset the timer to zero but you can set it to any positive number. See the TIMER function for more details.
TRACE ON or TRACE OFF or TRACE LIST nn	TRACE ON/OFF will turn on/off the trace facility. This facility will print the number of each line (counting from the beginning of the program) in square brackets as the program is executed. This is useful in debugging programs. TRACE LIST will list the last 'nn' lines executed in the format described above. MMBasic is always logging the lines executed so this facility is always available (i.e. it does not have to be turned on).
TRIANGLE X1, Y1, X2, Y2, X3, Y3 [, C [, FILL]]	Draws a triangle on the LCD display panel with the corners at X1, Y1 and X2, Y2 and X3, Y3. 'C' is the colour of the triangle and defaults to the current foreground colour. 'FILL' is the fill colour and defaults to no fill (it can also be set to -1 for no fill). All parameters can be expressed as arrays and the software will plot the number of triangles as determined by the dimensions of the smallest array unless X1 = Y1 = X2 = Y2 = X3 = Y3 = -1 in which case processing will stop at that point 'x1', 'y1', 'x2', 'y2', 'x3', and 'y3' must all be arrays or all be single variables /constants otherwise an error will be generated 'c' and 'fill' can be either arrays or single variables/constants.
TRIANGLE SAVE [#]n, x1,y1,x2,y2,x3,y3	Saves a triangular area of the screen to buffer #n.
TRIANGLE RESTORE [#]n	Restores a saved triangular region of the screen and deletes the saved buffer.
UPDATE FIRMWARE	Causes the PicoMite to enter the firmware update mode (the same as applying power while holding down the BOOTSEL button). Loading the PicoMite firmware will erase the flash memory including the current program, any programs saved in flash memory slots and all saved variables. So make sure that you backup this data before you upgrade the firmware. A firmware load will also reset all options to their defaults.
VAR SAVE var [, var]... or VAR RESTORE or VAR CLEAR	VAR SAVE will save one or more variables to non-volatile flash memory where they can be restored later (normally after a power interruption). 'var' can be any number of numeric or string variables and/or arrays. Arrays are specified by using empty brackets. For example: var() VAR RESTORE will retrieve the previously saved variables and insert them (and their values) into the variable table. The VAR SAVE command can be used repeatedly. Variables that had been previously saved will be updated with their new value and any new variables (not previously saved) will be added to the saved list for later restoration. VAR CLEAR will erase all saved variables. Also, the saved variables will be automatically cleared by a firmware upgrade, by the NEW command or when a new program is loaded via AUTOSAVE, XMODEM, etc. This command is normally used to save calibration data, options, and other data which does not change often but needs to be retained across a power interruption. Normally the VAR RESTORE command is placed at the start of the program so that previously saved variables are restored and immediately

TIME\$ = "HH:MM:SS" 或 TIME\$ = "HH:MM" 或 TIME\$ = "HH"	设置内部时钟的时间。MM和SS是可选的，如果未指定将默认为零。例如 TIME\$ = "14:30" 将把时钟设置为 14:30并且秒数为零。 使用选项 OPTION RTC 自动启用，PicoMite 将以 RTC 中编程的 TIME\$ 启动。 如果没有选项 OPTION RTC 自动启用，PicoMite 将以 TIME\$="00:00:00" 启动。
TIMER = 毫秒	将定时器重置为指定的毫秒数。通常这只是用于将定时器重置为零，但您可以将其设置为任何正数。 有关更多详细信息，请参见 TIMER 函数。
TRACE ON 或 TRACE OFF 或 TRACE LIST nn	TRACE ON/OFF 将启用/禁用跟踪功能。此功能将在程序执行时以方括号打印每行的编号（从程序开始计数）。这在调试程序时非常有用。 TRACE LIST 将以上述格式列出最后执行的 'nn' 行。MMBasic始终记录执行的行，因此此功能始终可用（即不必开启）。
三角形X1, Y1, X2, Y2, X3, Y3 [, C [, FILL]]	在液晶显示面板上绘制一个三角形，角点位于X1, Y1和X2, Y2以及X3, Y3。'C'是三角形的颜色，默认为当前前景颜色。'FILL'是填充颜色，默认为无填充（也可以设置为-1表示无填充）。 所有参数可以表示为数组，软件将根据最小数组的尺寸绘制三角形的数量，除非X1 = Y1 = X2 = Y2 = X3 = Y3 = -1，在这种情况下处理将在该点停止。'x1', 'y1', 'x2', 'y2', 'x3'和'y3'必须全部为数组或全部为单个变量/常量，否则将生成错误。'c'和'fill'可以是数组或单个变量/常量。
三角形 保存 [#]n, x1,y1,x2,y2,x3,y3	将屏幕的三角形区域保存到缓冲区 #n。
三角形 恢复 [#]n	恢复已保存的屏幕三角形区域并删除已保存的缓冲区。
更新固件	使PicoMite进入固件更新模式（与在按住BOOTSEL按钮的同时通电相同）。 加载PicoMite固件将擦除闪存，包括当前程序、任何保存在闪存插槽中的程序以及所有保存的变量。因此，请确保在升级固件之前备份这些数据。固件加载还将重置所有选项为默认值。
VAR 保存 var [, var]... 或 VAR 恢复 或 VAR 清除	VAR 保存将一个或多个变量保存到非易失性闪存中，以便稍后恢复（通常在断电后）。 'var' 可以是任意数量的数字或字符串变量和/或数组。数组通过使用空括号来指定。例如：var() VAR 恢复将检索之前保存的变量并将它们（及其值）插入变量表中。 VAR 保存命令可以重复使用。之前保存的变量将会用它们的新值更新，任何新的变量（未曾保存的）将被添加到保存列表中以便后续恢复。 VAR 清除将擦除所有保存的变量。此外，保存的变量将在固件升级、NEW 命令或通过自动保存、XMODEM 等加载新程序时自动清除。此命令通常用于保存校准数据、选项和其他不常更改但需要在断电时保留的数据。通常，VAR 恢复命令放在程序的开始，以便在程序启动时恢复之前保存的变量并立即

	<p>available to the program when it starts. Notes:</p> <ul style="list-style-type: none"> • The storage space available to this command is 16KB. • Using VAR RESTORE without a previous save will have no effect and will not generate an error. • If, when using RESTORE, a variable with the same name already exists its value will be overwritten. • Saved arrays must be declared (using DIM) before they can be restored. • Be aware that string arrays can rapidly use up all the memory allocated to this command. The LENGTH qualifier can be used when a string array is declared to reduce the size of the array (see the DIM command). This is not needed for ordinary string variables.
WATCHDOG timeout or WATCHDOG OFF Or WATCHDOG HW timeout Or WATCHDOG HW OFF	<p>Starts the watchdog timer which will automatically restart the processor when it has timed out. This can be used to recover from some event that disabled the running program (such as an endless loop or a programming or other error that halts a running program). This can be important in an unattended control situation. The timeout can either be processed in the system timer interrupt or as a true H/W watchdog.</p> <p>'timeout' is the time in milliseconds (ms) before a restart is forced. This command should be placed in strategic locations in the running BASIC program to constantly reset the watchdog timer (to 'timeout') and therefore prevent it from counting down to zero. If the H/W watchdog is used the timer has a maximum of 8.3 seconds. No such limitation exists for the software watchdog.</p> <p>If the timer count does reach zero (perhaps because the BASIC program has stopped running) the PicoMite will be automatically restarted and the automatic variable MM.WATCHDOG will be set to true (i.e. 1) indicating that an error occurred. On a normal startup MM.WATCHDOG will be set to false (i.e. 0). Note that OPTION AUTORUN must be specified for the program to restart.</p> <p>WATCHDOG OFF can be used to disable the watchdog timer (this is the default on a reset or power up). The timer is also turned off when the break character (CTRL-C) is used on the console to interrupt a running program.</p>
XMODEM SEND or XMODEM SEND file\$ or XMODEM RECEIVE or XMODEM RECEIVE file\$ or XMODEM CRUNCH	<p>Transfers a BASIC program to or from a remote computer using the XModem protocol. The transfer is done over the USB console connection.</p> <p>XMODEM SEND will send the current program held in the PicoMite's program memory to the remote device.</p> <p>XMODEM RECEIVE will accept a program sent by the remote device and save it into the PicoMite's the program memory overwriting the program currently held there.</p> <p>In both cases you can also specify 'file\$' which will transfer the data to/from a file on the Flash Filesystem or SD Card. If the file already exists it will be overwritten when receiving a file.</p> <p>Note that the data is buffered in RAM which limits the maximum transfer size. This command also creates a backup of the program in flash memory which will be automatically retrieved if the CPU is reset or the power is lost.</p> <p>The CRUNCH option works like RECEIVE but will remove all comments, blank lines and unnecessary spaces from the program before saving. This can be used on large programs to allow them to fit into limited memory.</p> <p>SEND, RECEIVE and CRUNCH can be abbreviated to S, R and C.</p> <p>The XModem protocol requires a cooperating software program running on the remote computer and connected to its serial port. It has been tested on Tera Term running on Windows and it is recommended that this be used.</p> <p>After running the XMODEM command in MMBasic select: File -> Transfer -> XMODEM -> Receive/Send</p>

	<p>可供程序使用。注意：</p> <ul style="list-style-type: none"> 此命令可用的存储空间为16KB。 在没有先前保存的情况下使用VAR恢复将没有效果，并且不会生成错误。 如果在使用恢复时，具有相同名称的变量已经存在，则其值将被覆盖。 保存的数组必须在恢复之前声明（使用DIM）。 请注意，字符串数组可能会迅速耗尽分配给此命令的所有内存。在声明字符串数组时，可以使用长度限定符来减少数组的大小（请参见DIM命令）。这对于普通字符串变量不是必需的。
看门狗超时 或 看门狗关闭 或 看门狗硬件超时 或 看门狗硬件关闭	<p>启动看门狗定时器，当其超时时将自动重启处理器。这可以用于从某些事件中恢复，这些事件禁用了正在运行的程序（例如无限循环或编程错误或其他错误导致正在运行的程序停止）。在无人值守的控制情况下，这可能很重要。超时可以在系统定时器中断中处理，或者作为真正的硬件看门狗。</p> <p>'timeout' 是在强制重启之前的毫秒（ms）时间。此命令应放置在运行的 BASIC 程序中的战略位置，以不断重置看门狗定时器（到 'timeout'），从而防止其倒计时到零。如果使用硬件看门狗，定时器的最大值为 8.3 秒。软件看门狗没有这样的限制。</p> <p>如果定时器计数确实达到零（可能是因为 BASIC 程序已停止运行），PicoMite 将自动重启，并且自动变量 MM.WATCHDOG 将被设置为 true（即 1），表示发生了错误。在正常启动时，MM.WATCHDOG 将被设置为 false（即 0）。请注意，必须指定自动运行选项才能使程序重启。</p> <p>WATCHDOG OFF 可用于禁用看门狗定时器（这是重置或上电时的默认设置）。当在控制台上使用中断字符（CTRL-C）中断正在运行的程序时，定时器也会被关闭。</p>
XMODEM 发送 或 XMODEM 发送 file\$ 或 XMODEM 接收 或 XMODEM 接收 file\$ 或 XMODEM 压缩	<p>使用 XModem 协议将 BASIC 程序传输到远程计算机或从远程计算机传输。传输通过 USB 控制台连接进行。</p> <p>XMODEM 发送将把 PicoMite 的程序内存中当前保存的程序发送到远程设备。</p> <p>XMODEM 接收将接受远程设备发送的程序，并将其保存到 PicoMite 的程序内存中，覆盖当前保存的程序。</p> <p>在这两种情况下，您还可以指定 'file'，这将把数据传输到/从 Flash 文件系统或 SD 卡上的文件。如果文件已存在，接收文件时将被覆盖。</p> <p>请注意，数据缓存在 RAM 中，这限制了最大传输大小。 此命令还会在闪存中创建程序的备份，如果中央处理器重置或断电，将自动恢复该备份。</p> <p>CRUNCH 选项的工作方式类似于 RECEIVE，但在保存之前会删除程序中的所有注释、空行和不必要的空格。这可以用于大型程序，以使其适应有限的内存。</p> <p>SEND、RECEIVE 和 CRUNCH 可以缩写为 S、R 和 C。XModem 协议需要在远程计算机上运行的配合软件程序，并连接到其串行端口。它已在 Windows 上运行的 Tera Term 中进行了测试，建议使用此软件。</p> <p>在 MMBasic 中运行 XMODEM 命令后，选择： 文件 -> 传输 -> XMODEM -> 接收/发送</p>

from the Tera Term menu to start the transfer.

The transfer can take up to 15 seconds to start and if the XMODEM command fails to establish communications it will return to the MMBasic prompt after 60 seconds and leave the program memory untouched.

Download Tera Term from <http://ttssh2.sourceforge.jp/>

从Tera Term菜单中开始传输。
传输可能需要最多15秒才能开始，如果XMODEM命令未能建立通信，
它将在60秒后返回到MMBasic提示符，并保持程序内存不变。

从<http://ttssh2.sourceforge.jp/>下载Tera Term

Functions

Note that the functions related to communications functions (I²C, 1-Wire, and SPI) are not listed here but are described in the appendices at the end of this document.

Square brackets indicate that the parameter or characters are optional.

ABS(number)	Returns the absolute value of the argument 'number' (i.e. any negative sign is removed and a positive number is returned).																				
ACOS(number)	Returns the inverse cosine of the argument 'number' in radians.																				
ASC(string\$)	Returns the ASCII code (i.e. byte value) for the first letter in 'string\$'.																				
ASIN(number)	Returns the inverse sine value of the argument 'number' in radians.																				
ATN(number)	Returns the arctangent of the argument 'number' in radians.																				
ATAN2(y, x)	Returns the arc tangent of the two numbers x and y as an angle expressed in radians. It is similar to calculating the arc tangent of y / x, except that the signs of both arguments are used to determine the quadrant of the result.																				
BIN\$(number [, chars])	Returns a string giving the binary (base 2) value for the 'number'. 'chars' is optional and specifies the number of characters in the string with zero as the leading padding character(s).																				
BIN2STR\$(type, value [,BIG])	Returns a string containing the binary representation of 'value'. 'type' can be: <table><tr><td>INT64</td><td>signed 64-bit integer converted to an 8 byte string</td></tr><tr><td>UINT64</td><td>unsigned 64-bit integer converted to an 8 byte string</td></tr><tr><td>INT32</td><td>signed 32-bit integer converted to a 4 byte string</td></tr><tr><td>UINT32</td><td>unsigned 32-bit integer converted to a 4 byte string</td></tr><tr><td>INT16</td><td>signed 16-bit integer converted to a 2 byte string</td></tr><tr><td>UINT16</td><td>unsigned 16-bit integer converted to a 2 byte string</td></tr><tr><td>INT8</td><td>signed 8-bit integer converted to a 1 byte string</td></tr><tr><td>UINT8</td><td>unsigned 8-bit integer converted to a 1 byte string</td></tr><tr><td>SINGLE</td><td>single precision floating point number converted to a 4 byte string</td></tr><tr><td>DOUBLE</td><td>double precision floating point number converted to a 8 byte string</td></tr></table> <p>By default the string contains the number in little-endian format (i.e. the least significant byte is the first one in the string). Setting the third parameter to 'BIG' will return the string in big-endian format (i.e. the most significant byte is the first one in the string). In the case of the integer conversions, an error will be generated if the 'value' cannot fit into the 'type' (e.g. an attempt to store the value 400 in a INT8).</p> <p>This function makes it easy to prepare data for efficient binary file I/O or for preparing numbers for output to sensors and saving to flash memory.</p> <p>See also the function STR2BIN</p>	INT64	signed 64-bit integer converted to an 8 byte string	UINT64	unsigned 64-bit integer converted to an 8 byte string	INT32	signed 32-bit integer converted to a 4 byte string	UINT32	unsigned 32-bit integer converted to a 4 byte string	INT16	signed 16-bit integer converted to a 2 byte string	UINT16	unsigned 16-bit integer converted to a 2 byte string	INT8	signed 8-bit integer converted to a 1 byte string	UINT8	unsigned 8-bit integer converted to a 1 byte string	SINGLE	single precision floating point number converted to a 4 byte string	DOUBLE	double precision floating point number converted to a 8 byte string
INT64	signed 64-bit integer converted to an 8 byte string																				
UINT64	unsigned 64-bit integer converted to an 8 byte string																				
INT32	signed 32-bit integer converted to a 4 byte string																				
UINT32	unsigned 32-bit integer converted to a 4 byte string																				
INT16	signed 16-bit integer converted to a 2 byte string																				
UINT16	unsigned 16-bit integer converted to a 2 byte string																				
INT8	signed 8-bit integer converted to a 1 byte string																				
UINT8	unsigned 8-bit integer converted to a 1 byte string																				
SINGLE	single precision floating point number converted to a 4 byte string																				
DOUBLE	double precision floating point number converted to a 8 byte string																				

函数

请注意，与通信功能相关的函数（I²C、1-Wire和SPI）未在此列出，但在本文档末尾的附录中进行了描述。

方括号表示参数或字符串是可选的。

ABS(数字)	返回参数 '数字' 的绝对值（即去掉任何负号，返回一个正数）。																
ACOS(数字)	返回参数 '数字' 的反余弦值（以弧度表示）。																
ASC(字符串\$)	返回‘字符串\$’中第一个字母的ASCII码（即字节值）。																
ASIN(数字)	返回参数 '数字' 的反正弦值（以弧度表示）。																
ATN(数字)	返回参数 '数字' 的反正切值（以弧度表示）。																
ATAN2(y, x)	返回两个数字 x 和 y 的反正切值，以弧度表示的角度。 这类似于计算 y / x 的反正切，只是两个参数的符号用于确定结果的象限。																
BIN\$(数字 [, 字符])	返回一个字符串，给出 '数字' 的二进制（基数 2）值。 '字符' 是可选的，指定字符串中字符的数量，零作为前导填充字符。																
BIN2STR\$(类型, 值 [,BIG])	返回一个包含 '值' 的二进制表示的字符串。 '类型' 可以是： <table><tr><td>INT64</td><td>转换为 8 字节字符串的有符号 64 位整数</td></tr><tr><td>UINT64</td><td>转换为 8 字节字符串的无符号 64 位整数</td></tr><tr><td>INT32</td><td>转换为 4 字节字符串的有符号 32 位整数</td></tr><tr><td>UINT32</td><td>转换为 4 字节字符串的无符号 32 位整数</td></tr><tr><td>INT16</td><td>转换为 2 字节字符串的有符号 16 位整数</td></tr><tr><td>UINT16</td><td>转换为 2 字节字符串的无符号 16 位整数</td></tr><tr><td>INT8</td><td>有符号8位整数转换为1字节字符串</td></tr><tr><td>UINT8</td><td>无符号8位整数转换为1字节字符串 单精度 单精度浮点数转换为4字节字符串 双精度浮点数转换为8字节字符串</td></tr></table> 默认情况下，字符串包含以小端格式表示的数字（即最低有效字节是字符串中的第一个字节）。将第三个参数设置为‘BIG’将返回以大端格式表示的字符串（即最高有效字节是字符串中的第一个字节）。在整数转换的情况下，如果‘值’无法适应‘类型’（例如，尝试将值400存储在INT8中），将会生成错误。 此函数使准备数据以进行高效的二进制文件I/O或准备数字以输出到传感器并保存到闪存中变得简单。 另请参见函数 STR2BIN	INT64	转换为 8 字节字符串的有符号 64 位整数	UINT64	转换为 8 字节字符串的无符号 64 位整数	INT32	转换为 4 字节字符串的有符号 32 位整数	UINT32	转换为 4 字节字符串的无符号 32 位整数	INT16	转换为 2 字节字符串的有符号 16 位整数	UINT16	转换为 2 字节字符串的无符号 16 位整数	INT8	有符号8位整数转换为1字节字符串	UINT8	无符号8位整数转换为1字节字符串 单精度 单精度浮点数转换为4字节字符串 双精度浮点数转换为8字节字符串
INT64	转换为 8 字节字符串的有符号 64 位整数																
UINT64	转换为 8 字节字符串的无符号 64 位整数																
INT32	转换为 4 字节字符串的有符号 32 位整数																
UINT32	转换为 4 字节字符串的无符号 32 位整数																
INT16	转换为 2 字节字符串的有符号 16 位整数																
UINT16	转换为 2 字节字符串的无符号 16 位整数																
INT8	有符号8位整数转换为1字节字符串																
UINT8	无符号8位整数转换为1字节字符串 单精度 单精度浮点数转换为4字节字符串 双精度浮点数转换为8字节字符串																

BOUND(array() [,dimension])	<p>This returns the upper limit of the array for the dimension requested. The dimension defaults to one if not specified. Specifying a dimension value of 0 will return the current value of OPTION BASE. Unused dimensions will return a value of zero.</p> <p>For example:</p> <pre>DIM myarray(44,45) BOUND(myarray(),2) will return 45</pre>
CALL(userfunname\$, [,userfunparameters,...])	<p>This is an efficient way of programmatically calling user defined functions. (See also the CALL command). In many cases it can be used to eliminate complex SELECT and IF THEN ELSEIF ENDIF clauses and is processed in a much more efficient manner.</p> <p>'userfunname\$' can be any string or variable or function that resolves to the name of a normal user function (not an in-built command).</p> <p>'userfunparameters' are the same parameters that would be used to call the function directly.</p> <p>A typical use for this command could be writing any sort of emulator where one of a large number of functions should be called depending on a some variable. It also provides a method of passing a function name to another subroutine or function as a variable.</p>
CHOICE(condition, ExpressionIfTrue, ExpressionIfFalse)	<p>This function allows you to do simple either/or selections more efficiently and faster than using IF THEN ELSE ENDIF clauses.</p> <p>The condition is anything that will resolve to nonzero (true) or zero (false).</p> <p>The expressions are anything that you could normally assign to a variable or use in a command and can be integers, floats or strings.</p> <p>Examples:</p> <pre>PRINT CHOICE(1, "hello","bye") will print "Hello" PRINT CHOICE (0, "hello","bye") will print "Bye" a=1 : b=1 : PRINT CHOICE (a=b, 4, 5) will print 4</pre>
CHR\$(number)	Returns a one-character string consisting of the character corresponding to the ASCII code (i.e. byte value) indicated by argument 'number'.
CINT(number)	<p>Round numbers with fractional portions up or down to the next whole number or integer.</p> <p>For example, 45.47 will round to 45 45.57 will round to 46 -34.45 will round to -34 -34.55 will round to -35</p> <p>See also INT() and FIX().</p>
COS(number)	Returns the cosine of the argument 'number' in radians.
CTRLVAL(#ref)	<p>Returns the current value of an advanced control.</p> <p>'#ref' is the control's reference. For controls like check boxes or switches it will be the number one (true) indicating that the control has been selected by the user or zero (false) if not. For controls that hold a number (e.g. a SPINBOX) the value will be the number (normally a floating point number). For controls that hold a string (e.g. TEXTBOX) the value will be a string.</p>

BOUND(array() [,dimension])	<p>这将返回请求维度的数组上限。</p> <p>如果未指定，维度默认为一。指定维度值为 0 将返回当前的 OPTION BASE 值。未使用的维度将返回零。</p> <p>例如：</p> <pre>DIM myarray(44,45) BOUND(myarray(),2) 将返回 45</pre>
CALL(userfunname\$, [,userfunparameters,...])	<p>这是一种有效的以编程方式调用用户定义函数的方法。(另请参见 CALL 命令)。在许多情况下，它可以用来消除复杂的 SELECT 和 IF THEN ELSEIF ENDIF 子句，并以更高效的方式处理。</p> <p>‘userfunname\$’可以是任何字符串、变量或函数，解析为普通用户函数的名称（不是内置命令）。</p> <p>‘userfunparameters’是调用该函数时直接使用的相同参数。</p> <p>此命令的典型用法可能是编写任何类型的仿真器，其中应根据某个变量调用大量函数中的一个。它还提供了一种将函数名称作为变量传递给另一个子程序或函数的方法。</p>
CHOICE(条件, 如果为真时的表达式, 如果为假时的表达式)	<p>此函数允许您以比使用 IF THEN ELSE ENDIF 子句更高效、更快速的方式进行简单的选择。</p> <p>条件是任何可以解析为非零（真）或零（假）的内容。</p> <p>表达式是您通常可以赋值给变量或在命令中使用的任何内容，可以是整数、浮点数或字符串。</p> <p>示例：</p> <pre>PRINT CHOICE(1, "hello","bye") 将打印 "你好" PRINT CHOICE (0, "hello","bye") 将打印 "再见" a=1 : b=1 : PRINT CHOICE (a=b, 4, 5) 将打印 4</pre>
CHR\$(数字)	返回一个由与参数'number'对应的ASCII码（即字节值）所表示的字符组成的单字符字符串。
CINT(number)	<p>将带有小数部分的数字四舍五入到下一个整数或整数。</p> <p>例如，45.47将四舍五入为45 45.57将四舍五入为46 -34.45将四舍五入为-34 -34.55将四舍五入为-35</p> <p>另见INT()和FIX()。</p>
COS(number)	返回参数'number'的余弦值（以弧度为单位）。
CTRLVAL(#ref)	<p>返回高级控件的当前值。</p> <p>'#ref'是控件的引用。对于像复选框或开关这样的控件，值将是数字1（真），表示控件已被用户选择，或者0（假），表示未被选择。对于保存数字的控件（例如SPINBOX），值将是数字（通常是浮点数）。对于保存字符串的控件（例如文本框），值将是一个字符串。</p>

CWDS	The current working directory on the Flash Filesystem or SD Card. Invalid for exFAT format. The format is: A:/dir1/dir2.
DATE\$	Returns the current date based on MMBasic's internal clock as a string in the form "DD-MM-YYYY". For example, "28-07-2012". The internal clock/calendar will keep track of the time and date including leap years. To set the date use the command DATE\$ =.
DATETIME\$(n)	Returns the date and time corresponding to the epoch number n (number of seconds that have elapsed since midnight GMT on January 1, 1970). The format of the returned string is "dd-mm-yyyy hh:mm:ss". Use the text NOW to get the current datetime string, i.e. ? DATETIME\$(NOW)
DAY\$(date\$)	Returns the day of the week for a given date as a string "Monday", "Tuesday" etc. The format for date\$ is "DD-MM-YY", "DD-MM-YYYY", or "YYYY-MM-DD". Use NOW to get the day for the current date, e.g. PRINT DAY\$(NOW)
DEG(radians)	Converts 'radians' to degrees.
DEVICE(WII funct)	Returns data from a Wii Classic controller. 'funct' is a 1 or 2 letter code indicating the information to return as follows: LX returns the position of the analog left joystick x axis LY returns the position of the analog left joystick y axis RX returns the position of the analog right joystick x axis RY returns the position of the analog right joystick y axis L returns the position of the analog left button R returns the position of the analog right button B returns a bitmap of the state of all the buttons. A bit will be set to 1 if the button is pressed. T returns the ID code of the controller - should be hex &HA4200101 The button bitmap is as follows: BIT 0: Button R BIT 1: Button start BIT 2: Button home BIT 3: Button select BIT 4: Button L BIT 5: Button down cursor BIT 6: Button right cursor BIT 7: Button up cursor BIT 8: Button left cursor BIT 9: Button ZR BIT 10: Button x BIT 11: Button a BIT 12: Button y BIT 13: Button b BIT 14: Button ZL

CWDS	Flash 文件系统或 SD 卡上的当前工作目录。对于 exFAT 格式无效。 格式为：A:/dir1/dir2。
日期\$	返回基于 MMBasic 内部时钟的当前日期，格式为字符串 "DD-MM-YY YY"。例如，"28-07-2012"。 内部时钟/日历将跟踪时间和日期，包括闰年。要设置日期，请使用命令 DATE\$ =。
DATETIME\$(n)	返回与纪元编号 n (自1970年1月1日午夜 GMT 起经过的秒数) 对应的日期和时间。返回字符串的格式为"dd-mm-yyyy hh:mm:ss"。使用文本 NOW 获取当前日期时间字符串，即 ?DATETIME\$(NOW)
DAY\$(date\$)	返回给定日期的星期几作为字符串，例如“星期一”、“星期二”等。date\$ 的格式为 "DD-MM-YY"、"DD-MM-YYYY" 或 "YYYY-MM-DD"。使用 NOW 获取当前日期的星期几，例如 PRINT DAY\$(NOW)
DEG(弧度)	将 '弧度' 转换为度数。
DEVICE(WII funct)	从 Wii Classic 控制器返回数据。 'funct' 是一个 1 或 2 个字母的代码，指示要返回的信息，如下所示： LX 返回模拟左摇杆 x 轴的位置 LY 返回模拟左摇杆 y 轴的位置 RX 返回模拟右摇杆 x 轴的位置 RY 返回模拟右摇杆 y 轴的位置 L 返回模拟左按钮的位置 R 返回模拟右按钮的位置 B 返回所有按钮状态的位图。如果按钮被按下，位将被设置为1。 T 返回控制器的 ID 代码 - 应为十六进制 &HA4200101 按钮位图如下： 位 0: 按钮 R 位 1: 按钮开始 位 2: 按钮主页 位 3: 按钮选择 位 4: 按钮 L 位 5: 按钮下光标 位 6: 按钮右光标 位 7: 按钮上光标 位 8: 按钮左光标 位 9: 按钮 ZR 位 10: 按钮 x 位 11: 按钮 a 位 12: 按钮 y 位 13: 按钮 b 位 14: 按钮 ZL

<p>DIR\$(fspec, type) or DIR\$(fspec) or DIR\$()</p>	<p>Will search the default Flash Filesystem or SD Card for files and return the names of entries found. 'fspec' is a file specification using wildcards the same as used by the FILES command. E.g. "./*" will return all entries, ".TXT" will return text files. Note that the wildcard "./*" does not find files or folders without an extension. 'type' is the type of entry to return and can be one of: VOL Search for the volume label only DIR Search for directories only FILE Search for files only (the default if 'type' is not specified) The function will return the first entry found. To retrieve subsequent entries use the function with no arguments. i.e. DIR\$(). The return of an empty string indicates that there are no more entries to retrieve. This example will print all the files in a directory: <pre>f\$ = DIR\$("*.*", FILE) DO WHILE f\$ <> "" PRINT f\$ f\$ = DIR\$() LOOP</pre> You must change to the required directory before invoking this command.</p>
<p>DISTANCE(trigger, echo) or DISTANCE(trig-echo)</p>	<p>Measure the distance to a target using the HC-SR04 ultrasonic distance sensor. Four pin sensors have separate trigger and echo connections. 'trigger' is the I/O pin connected to the "trig" input of the sensor and 'echo' is the pin connected to the "echo" output of the sensor. Three pin sensors have a combined trigger and echo connection and in that case you only need to specify one I/O pin to interface to the sensor. Note that any I/O pins used with the HC-SR04 should be 5V capable as the HC-SR04 is a 5V device. The I/O pins are automatically configured by this function and multiple sensors can be used on different I/O pins. The value returned is the distance in centimetres to the target or -1 if no target was detected or -2 if there was an error (i.e. sensor not connected).</p>
<p>EOF([#]fnbr)</p>	<p>Will return true if the file previously opened on the Flash Filesystem or SD Card for INPUT with the file number '#fnbr' is positioned at the end of the file. The # is optional. Also see the OPEN, INPUT and LINE INPUT commands and the INPUT\$ function.</p>
<p>EPOCH(DATETIME\$)</p>	<p>Returns the epoch number (number of seconds that have elapsed since midnight GMT on January 1, 1970) for the supplied DATETIME\$ string. The format for DATETIME\$ is "dd-mm-yyyy hh:mm:ss", "dd-mm-yy hh:mm:ss", or "yyyy-mm-dd hh:mm:ss". Use NOW to get the epoch number for the current date and time, i.e. PRINT EPOCH(NOW)</p>
<p>EVAL(string\$)</p>	<p>Will evaluate 'string\$' as if it is a BASIC expression and return the result. 'string\$' can be a constant, a variable or a string expression. The expression can use any operators, functions, variables, subroutines, etc that are known at the time of execution. The returned value will be an integer, float or string depending on the result of the evaluation. For example: S\$ = "COS(RAD(30)) * 100" : PRINT EVAL(S\$) Will display: 86.6025</p>
<p>EXP(number)</p>	<p>Returns the exponential value of 'number', i.e. e^x where x is 'number'.</p>

<p>DIR\$(fspec, type) 或 DIR\$(fspec) 或 DIR\$()</p>	<p>将搜索默认的 Flash 文件系统或 SD 卡中的文件，并返回找到的条目的名称。 'fspec' 是一个文件规范，使用与 FILES 命令相同的通配符。例如："*.*" 将返回所有条目，"*.TXT" 将返回文本文件。 请注意，通配符 ".*" 不会找到没有扩展名的文件或文件夹。 'type' 是要返回的条目的类型，可以是以下之一：</p> <table border="0"> <tr><td>VOL</td><td>仅搜索卷标</td></tr> <tr><td>DIR</td><td>仅搜索目录</td></tr> <tr><td>FILE</td><td>仅搜索文件（如果未指定 'type'，则为默认值）</td></tr> </table> <p>该函数将返回找到的第一个条目。要检索后续条目请使用不带参数的函数，即 DIR\$()。返回空字符串表示没有更多条目可检索。</p> <p>此示例将打印目录中的所有文件：</p> <pre>f\$ = DIR\$("*.*", FILE) DO WHILE f\$ <> "" PRINT f\$ f\$ = DIR\$() LOOP</pre> <p>在调用此命令之前，您必须切换到所需的目录。</p>	VOL	仅搜索卷标	DIR	仅搜索目录	FILE	仅搜索文件（如果未指定 'type'，则为默认值）
VOL	仅搜索卷标						
DIR	仅搜索目录						
FILE	仅搜索文件（如果未指定 'type'，则为默认值）						
<p>DISTANCE(trigger, echo) 或 DISTANCE(trig-echo)</p>	<p>使用 HC-SR04 超声波距离传感器测量到目标的距离。 四引脚传感器具有独立的触发和回声连接。'trigger' 是连接到传感器的 "trig" 输入的 I/O 引脚，而 'echo' 是连接到传感器的 "echo" 输出的引脚。 三引脚传感器具有合并的触发和回声连接，在这种情况下，您只需指定一个 I/O 引脚与传感器接口。 请注意，任何与 HC-SR04 一起使用的 I/O 引脚应能够承受 5V，因为 HC-SR04 是一个 5V 设备。这些 I/O 引脚由此函数自动配置，并且可以在不同的 I/O 引脚上使用多个传感器。 返回的值是到目标的距离（以厘米为单位），如果未检测到目标则为 -1，如果发生错误（即传感器未连接）则为 -2。</p>						
<p>EOF([#]fnbr)</p>	<p>如果之前在 Flash 文件系统或 SD 卡上以文件编号 '#fnbr' 打开用于输入的文件位于文件末尾，则返回真。 井号是可选的。另请参阅 OPEN、INPUT 和 LINE INPUT 命令以及 INPUT\$ 函数。</p>						
<p>纪元(DATETIME\$)</p>	<p>返回提供的 DATETIME\$ 字符串的纪元号（自 1970 年 1 月 1 日午夜 GMT 起经过的秒数）。DATETIME\$ 的格式为“dd-mm-yyyy hh:mm:ss”、“dd-mm-yy hh:mm:ss”或“yyyy-mm-dd hh:mm:ss”。使用 NOW 获取当前日期和时间的纪元号，即 PRINT EPOCH(NOW)</p>						
<p>EVAL(string\$)</p>	<p>将 'string' 作为 BASIC 表达式进行求值并返回结果。 'string' 可以是常量、变量或字符串表达式。该表达式可以使用在执行时已知的任何运算符、函数、变量、子例程等。返回的值将是整数、浮点数或字符串，具体取决于求值的结果。 例如：S\$ = "COS(RAD(30)) * 100" : PRINT EVAL(S\$) 将显示：86.6025</p>						
<p>EXP(数字)</p>	<p>返回'数字'的指数值，即 e^x，其中 x 是'数字'。</p>						

FIELD\$(string1, nbr, string2 [, string3])	<p>Returns a particular field in a string with the fields separated by delimiters. 'nbr' is the field to return (the first is nbr 1). 'string1' is the string to search and 'string2' is a string holding the delimiters (more than one can be used). 'string3' is optional and if specified will include characters that are used to quote text in 'string1' (ie, quoted text will not be searched for a delimiter).</p> <p>For example:</p> <pre>S\$ = "foo, boo, zoo, doo" r\$ = FIELD\$(s\$, 2, ",")</pre> <p>will result in r\$ = "boo". While:</p> <pre>s\$ = "foo, 'boo, zoo', doo" r\$ = FIELD\$(s\$, 2, ",", "")</pre> <p>will result in r\$ = "boo, zoo".</p>
FIX(number)	<p>Truncate a number to a whole number by eliminating the decimal point and all characters to the right of the decimal point.</p> <p>For example 9.89 will return 9 and -2.11 will return -2.</p> <p>The major difference between FIX() and INT() is that FIX() provides a true integer function (i.e. does not return the next lower number for negative numbers as INT() does). This behaviour is for Microsoft compatibility.</p> <p>See also CINT() .</p>
FORMAT\$(nbr [, fmt\$])	<p>Will return a string representing 'nbr' formatted according to the specifications in the string 'fmt\$'.</p> <p>The format specification starts with a % character and ends with a letter. Anything outside of this construct is copied to the output as is.</p> <p>The structure of a format specification is:</p> <pre>% [flags] [width] [.precision] type</pre> <p>Where 'flags' can be:</p> <ul style="list-style-type: none"> - Left justify the value within a given field width 0 Use 0 for the pad character instead of space + Forces the + sign to be shown for positive numbers space Causes a positive value to display a space for the sign. Negative values still show the – sign <p>'width' is the minimum number of characters to output, less than this the number will be padded, more than this the width will be expanded.</p> <p>'precision' specifies the number of fraction digits to generate with an e, or f type or the maximum number of significant digits to generate with a g type and defaults to 4 digits. If specified, the precision must be preceded by a dot (.).</p> <p>'type' can be one of:</p> <ul style="list-style-type: none"> g Automatically format the number for the best presentation. f Format the number with the decimal point and following digits e Format the number in exponential format <p>If uppercase G or F is used the exponential output will use an uppercase E. If the format specification is not specified "%g" is assumed.</p> <p>Examples: format\$(45) will return 45 format\$(45, "%g") will return 45</p>

<p>FIELD\$(字符串1, 编号, 字符串2 [, 字符串3])</p>	<p>返回字符串中特定的字段，字段由分隔符分隔。 '编号'是要返回的字段（第一个是编号）。'字符串1'是要搜索的字符串，'字符串2'是包含分隔符的字符串（可以使用多个分隔符）。 '字符串3'是可选的，如果指定，将包括用于引用'字符串1'中的文本的字符（即，引用的文本将不会被搜索分隔符）。</p> <p>例如：</p> <pre>S\$ = "foo, boo, zoo, doo" r\$ = FIELD\$(s\$, 2, ",")</pre> <p>将导致 r\$ = "boo"。而：</p> <pre>s\$ = "foo, 'boo, zoo', doo" r\$ = FIELD\$(s\$, 2, ",", "")</pre> <p>将导致 r\$ = "boo, zoo"。</p>						
<p>FIX(number)</p>	<p>通过消除小数点及其右侧的所有字符，将数字截断为整数。</p> <p>例如，9.89 将返回 9，而 -2.11 将返回 -2。</p> <p>FIX() 和 INT() 之间的主要区别在于 FIX() 提供了一个真正的整数函数（即对于负数，不像 INT() 返回下一个较小的数字）。这种行为是为了与微软兼容。另见 CINT()。</p>						
<p>FORMAT\$(nbr [, fmt\$])</p>	<p>将返回一个字符串，表示根据字符串 'fmt\$' 中的规范格式化的 'nbr'。</p> <p>格式规范以 % 字符开头，并以字母结尾。 任何不在此结构中的内容都将原样复制到输出中。</p> <p>格式规范的结构是：</p> <pre>% [标志] [宽度] [.精度] 类型</pre> <p>其中‘标志’可以是：</p> <ul style="list-style-type: none"> - 在给定的字段宽度内左对齐值 0 使用0作为填充字符而不是空格 + 强制显示正数的+号 <p>空格使正值显示空格作为符号。负数值仍然显示-符号</p> <p>‘宽度’是输出的最小字符数，少于此值时数字将被填充，多于此值时宽度将被扩展。</p> <p>‘精度’指定生成的分数位数，适用于e或f类型，或生成的最大有效数字数，适用于g类型，默认为4位数字。如果指定，精度必须以点(.) 开头。</p> <p>‘类型’可以是以下之一：</p> <table border="0"> <tr> <td style="padding-right: 20px;">g</td> <td>自动格式化数字以获得最佳展示效果。</td> </tr> <tr> <td style="padding-right: 20px;">f</td> <td>以小数点和后续数字格式化数字</td> </tr> <tr> <td style="padding-right: 20px;">e</td> <td>以指数格式格式化数字</td> </tr> </table> <p>如果使用大写的 G 或 F，指数输出将使用大写的 E。如果未指定格式规范，则假定为“%g”。</p> <p>示例： format\$(45) 将返回 45 format\$(45, "%g") 将返回 45</p>	g	自动格式化数字以获得最佳展示效果。	f	以小数点和后续数字格式化数字	e	以指数格式格式化数字
g	自动格式化数字以获得最佳展示效果。						
f	以小数点和后续数字格式化数字						
e	以指数格式格式化数字						

GPS()	The GPS functions are used to return data from a serial communications channel opened as GPS. The function GPS(VALID) should be checked before any of these functions are used to ensure that the returned value is valid.
GPS(ALTITUDE)	Returns current altitude (if sentence GGA is enabled).
GPS(DATE)	Returns the normal date string corrected for local time e.g. "12-01-2020".
GPS(DOP)	Returns DOP (dilution of precision) value (if sentence GGA is enabled).
GPS(FIX)	Returns non zero (true) if the GPS has a fix on sufficient satellites and is producing valid data.
GPS(GEOID)	Returns the geoid-ellipsoid separation (if sentence GGA is enabled).
GPS(LATITUDE)	Returns the latitude in degrees as a floating point number, values are negative for South of equator
GPS(LONGITUDE)	Returns the longitude in degrees as a floating point number, values are negative for West of the meridian.
GPS(SATELLITES)	Returns number of satellites in view (if sentence GGA is enabled).
GPS(SPEED)	Returns the ground speed in knots as a floating point number.
GPS(TIME)	Returns the normal time string corrected for local time e.g. "12:09:33".
GPS(TRACK)	Returns the track over the ground (degrees true) as a floating point number.
GPS(VALID)	Returns: 0=invalid data, 1=valid data
HEX\$(number [, chars])	Returns a string giving the hexadecimal (base 16) value for the 'number'. 'chars' is optional and specifies the number of characters in the string with zero as the leading padding character(s).
INKEY\$	Checks the console input buffer and, if there is one or more characters waiting in the queue, will remove the first character and return it as a single character in a string. If the input buffer is empty this function will immediately return with an empty string (i.e. "").
INPUT\$(nbr, [#fnbr])	Will return a string composed of 'nbr' characters read from a serial communications port opened as 'fnbr'. This function will return as many characters as are waiting in the receive buffer up to 'nbr'. If there are no characters waiting it will immediately return with an empty string. #0 can be used which refers to the console's input buffer. The # is optional. Also see the OPEN command.

GPS()	GPS 函数用于从作为 GPS 打开的串行通信通道返回数据。 在使用这些函数之前，应检查函数 GPS(VALID)，以确保返回的值有效。
GPS(ALTITUDE)	返回当前高度（如果启用了 GGA 语句）。
GPS(DATE)	返回正常的日期字符串，已根据当地时间校正，例如“12-01-2020”。
GPS(DOP)	返回 DOP（精度稀释）值（如果启用了 GGA 语句）。
GPS(FIX)	如果 GPS 已经锁定足够的卫星并生成有效数据，则返回非零（真）。
GPS(GEOID)	返回大地水准面与椭球体的分离（如果启用了GGA语句）。
GPS(纬度)	返回以浮点数表示的纬度值，南半球的值为负数。
GPS(经度)	返回以浮点数表示的经度值，西经的值为负数。
GPS(卫星)	返回可见卫星的数量（如果启用了GGA语句）。
GPS(速度)	返回以浮点数表示的地面速度（单位：节）。
GPS(时间)	返回经过本地时间修正的正常时间字符串，例如“12:09:33”。
GPS(航向)	返回以浮点数表示的地面向（真实度数）。
GPS(有效性)	返回：0=无效数据，1=有效数据
HEX\$(数字 [,字符])	返回一个字符串，给出'数字'的十六进制（基数16）值。 '字符'是可选的，指定字符串中字符的数量，零作为前导填充字符。
INKEY\$	检查控制台输入缓冲区，如果队列中有一个或多个字符等待，将移除第一个字符并作为字符串中的单个字符返回。 如果输入缓冲区为空，此函数将立即返回一个空字符串（即 ""）。
INPUT\$(nbr, [#]fnbr)	将返回由从作为 'fnbr' 打开的串行通信端口读取的 'nbr' 个字符组成的字符串。此函数将返回接收缓冲区中等待的字符，最多为 'nbr' 个。如果没有等待的字符，它将立即返回一个空字符串。 #0 可以被使用，表示控制台的输入缓冲区。 井号是可选的。另请参见 OPEN 命令。

INSTR([start-position,] string-searched\$, string-pattern\$ [,size])	Returns the position at which 'string-pattern\$' occurs in 'string-searched\$', beginning at 'start-position'. If 'start-position' is not provided it will default to 1. Both the position returned and 'start-position' use 1 for the first character, 2 for the second, etc. The function returns zero if 'string-pattern\$' is not found. If the optional parameter "size" is specified the "string-pattern" is treated as a regular expression. See Appendix E for the details.
INT(number)	Truncate an expression to the next whole number less than or equal to the argument. For example 9.89 will return 9 and -2.11 will return -3. This behaviour is for Microsoft compatibility, the FIX() function provides a true integer function. See also CINT() .
LCASE\$(string\$)	Returns 'string\$' converted to lowercase characters.
LCOMPARE(array1%(), array2%())	Compare the contents of two long string variables array1%() and array2%(). The returned is an integer and will be -1 if array1%() is less than array2%(). It will be zero if they are equal in length and content and +1 if array1%() is greater than array2%(). The comparison uses the ASCII character set and is case sensitive.
LEFT\$(string\$, nbr)	Returns a substring of 'string\$' with 'nbr' of characters from the left (beginning) of the string.
LEN(string\$)	Returns the number of characters in 'string\$'.
LGETBYTE(array%(), n)	Returns the numerical value of the 'n'th byte in the LONGSTRING held in 'array%()'. This function respects the setting of OPTION BASE in determining which byte to return.
LGETSTR\$(array%(), start, length)	Returns part of a long string stored in array%() as a normal MMBasic string. The parameters start and length define the part of the string to be returned.
LINSTR(array%(), search\$ [,start] [,size]))	Returns the position of a search string in a long string. The returned value is an integer and will be zero if the substring cannot be found. array%() is the string to be searched and must be a long string variable. Search\$ is the substring to look for and it must be a normal MMBasic string or expression (not a long string). The search is case sensitive. Normally the search will start at the first character in 'str' but the optional third parameter allows the start position of the search to be specified. If the optional parameter "size" is specified the "string-pattern" is treated as a regular expression. See Appendix E for the details.
LLEN(array%())	Returns the length of a long string stored in array%()
LOC([#]fnbr)	For a serial communications port opened as 'fnbr' this function will return the number of bytes received and waiting in the receive buffer to be read. #0 can be used which refers to the console's input buffer. The # is optional.

INSTR([起始位置,] 被搜索字符串\$, 字符串模式\$ [,大小])	<p>返回 '字符串模式' 在 '被搜索字符串' 中出现的位置，从 '起始位置' 开始。如果没有提供 'start-position'，则默认为 1。</p> <p>返回的位置和 'start-position' 都使用 1 表示第一个字符，2 表示第二个字符，依此类推。</p> <p>如果未找到 'string-pattern'，则函数返回零。</p> <p>如果指定了可选参数“size”，则“string-pattern”将被视为正则表达式。详细信息请参见附录 E。</p>
INT(数字)	<p>将表达式截断为小于或等于参数的下一个整数。例如，9.89 将返回 9，而 -2.11 将返回 -3。</p> <p>此行为是为了与微软兼容，FIX() 函数提供了真正的整数函数。另见 CINT()。</p>
LCASE\$(字符串\$)	返回转换为小写字符的 '字符串\$'。
LCOMPARE(数组1%(), 数组2%())	<p>比较两个长字符串变量 array1%() 和 array2%() 的内容。</p> <p>返回值是一个整数，如果 array1%() 小于 array2%()，则返回 -1。如果它们在长度和内容上相等，则返回零；如果 array1%() 大于 array2%()，则返回 +1。比较使用 ASCII 字符集，并且区分大小写。</p>
LEFT\$(string\$, nbr)	返回 'string' 的子字符串，包含从字符串左侧（开头）开始的 'nbr' 个字符。
LEN(string\$)	返回 'string' 中的字符数。
LGETBYTE(array%(), n)	返回 'array%' 中 LONGSTRING 的第 'n' 个字节的数值。此函数在确定返回哪个字节时遵循 OPTION BASE 的设置。
LGETSTR\$(array%(), start, length)	将存储在 array%() 中的长字符串的一部分作为普通的 MMBasic 字符串返回。参数 start 和 length 定义要返回的字符串部分。
LINSTR(array%(), search\$ [,start] [,size])	<p>返回搜索字符串在长字符串中的位置。返回的值是一个整数，如果找不到子字符串，则为零。array%() 是要搜索的字符串必须是一个长字符串变量。Search\$ 是要查找的子字符串，必须是一个普通的 MMBasic 字符串或表达式（不是长字符串）。搜索是区分大小写的。</p> <p>通常搜索将从 'str' 的第一个字符开始，但可选的第三个参数允许指定搜索的起始位置。</p> <p>如果指定了可选参数“size”，则“string-pattern”将被视为正则表达式。详细信息请参见附录 E。</p>
LLEN(array%())	返回存储在 array%() 中的长字符串的长度。
LOC([#]fnbr)	<p>对于作为 'fnbr' 打开的串行通信端口，此函数将返回接收缓冲区中接收到的并等待读取的字节数。</p> <p>#0 可以被使用，表示控制台的输入缓冲区。</p> <p># 是可选的。</p>

LOF([#]fnbr)	For a serial communications port opened as 'fnbr' this function will return the space (in characters) remaining in the transmit buffer. Note that when the buffer is full MMBasic will pause when adding a new character and wait for some space to become available. The # is optional.
LOG(number)	Returns the natural logarithm of the argument 'number'.
MATH Simple functions	The math function performs many simple mathematical calculations that can be programmed in Basic but there are speed advantages to coding looping structures in C and there is the advantage that once debugged they are there for everyone without re-inventing the wheel.
MATH(ATAN3 x,y)	Returns ATAN3 of x and y
MATH(COSH a)	Returns the hyperbolic cosine of a
MATH(LOG10 a)	Returns the base 10 logarithm of a
MATH(SINH a)	Returns the hyperbolic sine of a
MATH(TANH a)	Returns the hyperbolic tan of a
MATH(CRCn data [,length] [,polynome] [,startmask] [,endmask] [,reverseIn] [,reverseOut]	Calculates the CRC to n bits (8, 12, 16, 32) of "data". "data" can be an integer or floating point array or a string variable. "Length" is optional and if not specified the size of the array or string length is used. The defaults for startmask, endmask reverseIn, and reversOut are all zero. reverseIn, and reversOut are both Booleans and take the value 1 or 0. The defaults for polynomials are CRC8=&H07, CRC12=&H80D, CRC16=&H1021, crc32=&H04C11DB7 e.g. for crc16_CCITT use MATH(CRC16 array(), n,, &FFFF)
MATH(RAND)	Returns a random number 0.0 <= n < 1.0 using the "Mersenne Twister algorithm. If not seeded with MATH RANDOMIZE the first usage seeds with the time in microseconds since boot
 Simple Statistics	
MATH(CHI a())	Returns the Pearson's chi-squared value of the two dimensional array a()
MATH(CHI_p a())	Returns the associated probability in % of the Pearson's chi-squared value of the two dimensional array a()
MATH(CROSSING array() [,level] [,direction]	This returns the array index at which the values in the array pass the "level" in the direction specified. level defaults to 0. Direction defaults to 1 (valid values are -1 or 1)
MATH(CORREL a(), a())	Returns the Pearson's correlation coefficient between arrays a() and b()
MATH(MAX a() [,index%])	Returns the maximum of all values in the a() array, a() can have any number of

LOF([#]fnbr)	对于以 'fnbr' 打开的串行通信端口，此函数将返回传输缓冲区中剩余的空间（以字符为单位）。 请注意，当缓冲区满时，MMBasic 在添加新字符时会暂停，并等待一些空间变得可用。 # 是可选的。
LOG(数字)	返回参数 '数字' 的自然对数。
数学	数学函数执行许多简单的数学计算，这些计算可以用Basic编程，但在C语言中编写循环结构具有速度优势，并且一旦调试完成，它们就可以供所有人使用，而无需重新发明轮子。
简单函数	
MATH(ATAN3 x,y)	返回x和y的ATAN3值
MATH(COSH a)	返回a的双曲余弦
MATH(LOG10 a)	返回a的以10为底的对数
MATH(SINH a)	返回a的双曲正弦
MATH(TANH a)	返回a的双曲正切
MATH(CRCn 数据 [,长度] [,多项式] [,起始掩码] [,结束掩码] [,反向输入] [,反向输出]	计算“data”的n位（8、12、16、32）CRC。“data”可以是一个整数或浮点数组或字符串变量。“Length”是可选的，如果未指定，则使用数组的大小或字符串长度。startmask、endmask、reverseIn和reverseOut的默认值均为零。reverseIn和reverseOut都是布尔值，取值为1或0。多项式的默认值为 CRC8=&H07, CRC12=&H80D, CRC16=&H1021, crc32=&H04C11DB7 例如，对于 crc16_CCITT 使用 MATH(CRC16 array(), n,, &xFFFF)
MATH(RAND)	使用 "梅森旋转算法" 返回一个随机数 $0.0 \leq n < 1.0$ 。如果没有使用 MATH RANDOMIZE 进行初始化，第一次使用将以自启动以来的微妙时间作为种子
简单统计	
MATH(CHI a())	返回二维数组 a() 的皮尔逊卡方值
MATH(CHI_p a())	返回二维数组 a() 的皮尔逊卡方值的相关概率 (%)
MATH(CROSSING 数组() [,水平] [,方向]	返回数组中值通过指定方向的 "level" 的数组索引。默认 level 为 0。方向默认为 1（有效值为 -1 或 1）
MATH(CORREL a(), a())	返回数组 a() 和 b() 之间的皮尔逊相关系数
MATH(MAX a() [,index%])	返回数组 a() 中所有值的最大值，a() 可以具有任意数量的

	dimensions. If the integer variable is specified then it will be updated with the index of the maximum value in the array. This is only available on one-dimensional arrays
MATH(MEAN a())	Returns the average of all values in the a() array, a() can have any number of dimensions
MATH(MEDIAN a())	Returns the median of all values in the a() array, a() can have any number of dimensions
MATH(MIN a(), [index%])	Returns the minimum of all values in the a() array, a() can have any number of dimensions. If the integer variable is specified then it will be updated with the index of the maximum value in the array. This is only available on one-dimensional arrays.
MATH(SD a())	Returns the standard deviation of all values in the a() array, a() can have any number of dimensions
MATH(SUM a())	Returns the sum of all values in the a() array, a() can have any number of dimensions
Vector Arithmetic	
MATH(MAGNITUDE v())	Returns the magnitude of the vector v(). The vector can have any number of elements
MATH(DOTPRODUCT v1(), v2())	Returns the dot product of two vectors v1() and v2(). The vectors can have any number of elements but must have the same cardinality
Matrix Arithmetic	
MATH(M_DETERMINANT array!())	Returns the determinant of the array. The array must be square.
Creation complex% = MATH(C_CPLX r!, i!) complex% = MATH(C_POLAR radius!, angle!) Floating returns real! = MATH(C_REAL complex%) imag! = MATH(C_IMAG complex%) arg! = MATH(C_ARG complex%) mod! = MATH(C_MOD complex%) phase! = MATH(C_PHASE complex%) Unary functions complex1% = MATH(C_CONJ complex2%) complex1% = MATH(C_SIN complex2%) complex1% = MATH(C_COS complex2%) complex1% = MATH(C_TAN complex2%)	MMBasic supports a full range of functions to allow the manipulation of complex numbers. In this implementation complex numbers have a 32-bit real and 32-bit imaginary part and to make this work in MMBasic, it uses integers (64-bit) to hold these.

	尺寸。如果指定了整数变量，则它将更新为数组中最大值的索引。这仅适用于一维数组
MATH(MEAN a())	返回数组 a() 中所有值的平均值， a() 可以具有任意数量的维度
MATH(MEDIAN a())	返回数组 a() 中所有值的中位数， a() 可以具有任意数量的维度
MATH(MIN a(), [index%])	返回数组 a() 中所有值的最小值， a() 可以具有任意数量的维度。如果指定了整数变量，则它将更新为数组中最大值的索引。这仅适用于一维数组。
MATH(SD a())	返回数组 a() 中所有值的标准差， a() 可以具有任意数量的维度。
MATH(SUM a())	返回数组 a() 中所有值的总和， a() 可以具有任意数量的维度。
向量算术	
MATH(MAGNITUDE v())	返回向量 v() 的大小。向量可以具有任意数量的元素。
MATH(DOTPRODUCT v1(), v2())	返回两个向量 v1() 和 v2() 的点积。向量可以具有任意数量的元素，但必须具有相同的基数。
矩阵算术	
MATH(M_DETERMINANT array!())	返回数组的行列式。数组必须是方阵。
创建	
complex% = MATH(C_CPLX r!, i!)	MMBasic 支持一整套函数，以便对复数进行操作。在此实现中，复数具有 32 位的实部和 32 位的虚部，为了在 MMBasic 中实现这一点，它使用整数（64 位）来存储这些值。
complex% = MATH(C_POLAR radius!, angle!)	
浮点返回	
real! = MATH(C_REAL complex%) imag! = MATH(C_IMAG complex%) arg! = MATH(C_ARG complex%) mod! = MATH(C_MOD complex%) phase! = MATH(C_PHASE complex%)	
一元函数	
complex1% = MATH(C_CONJ complex2%) complex1% = MATH(C_SIN complex2%) complex1% = MATH(C_COS complex2%) complex1% = MATH(C_TAN complex2%)	

```

complex1% = MATH(C_ASIN complex2%)
complex1% = MATH(C_ACOS complex2%)
complex1% = MATH(C_ATAN complex2%)
complex1% = MATH(C_SINH complex2%)
complex1% = MATH(C_COSH complex2%)
complex1% = MATH(C_TANH complex2%)
complex1% = MATH(C_ASINH complex2%)
complex1% = MATH(C_ACOSH complex2%)
complex1% = MATH(C_ATANH complex2%)
complex1% = MATH(C_PROJ complex2%)

```

Basic Arithmetic

```

complex1% = MATH(C_ADD complex2%,complex3%)
complex1% = MATH(C_SUB complex2%,complex3%)
complex1% = MATH(C_MUL complex2%,complex3%)
complex1% = MATH(C_DIV complex2%,complex3%)
complex1% = MATH(C_POW complex2%,complex3%)
complex1% = MATH(C_AND complex2%,complex3%)
complex1% = MATH(C_OR complex2%,complex3%)
complex1% = MATH(C_XOR complex2%,complex3%)

```

<p>MAX(arg1 [, arg2 [, ...]]) or MIN(arg1 [, arg2 [, ...]])</p>	<p>Returns the maximum or minimum number in the argument list. Note that the comparison is a floating point comparison (integer arguments are converted to floats) and a float is returned.</p>
<p>MID\$(string\$, start) or MID\$(string\$, start, nbr)</p>	<p>Returns a substring of 'string\$' beginning at 'start' and continuing for 'nbr' characters. The first character in the string is number 1. If 'nbr' is omitted the returned string will extend to the end of 'string\$'</p>
<p>MSGBOX (msg\$, b1\$ [,b2\$... b4\$])</p>	<p>This function will display a message box on the screen with one to four touch sensitive buttons. All other controls will be disabled until the user touches one of the buttons. The message box will then be erased, the previous controls will be restored and the function will return the number of the button touched (the first button is number one) 'msg\$' is the message to display. This can contain one or more tilde characters (~) which indicate a line break. Up to 10 lines can be displayed inside the box. 'b1\$' is the caption for the first button, 'b2\$' is the caption for the second button, etc. At least one button must be specified and four is the maximum. Any buttons not included in the argument list will not be displayed.</p>
<p>OCT\$(number [, chars])</p>	<p>Returns a string giving the octal (base 8) representation of 'number'. 'chars' is optional and specifies the number of characters in the string with zero as the leading padding character(s).</p>
<p>PEEK(BYTE addr%) or PEEK(SHORT addr%) or PEEK(WORD addr%) or PEEK(INTEGER addr%) or PEEK(FLOAT addr%)</p>	<p>Will return a byte or a word within the PIC32 virtual memory space. BYTE will return the byte (8-bits) located at 'addr%' SHORT will return the short integer (16-bits) located at 'addr%' WORD will return the word (32-bits) located at 'addr%' INTEGER will return the integer (64-bits) located at 'addr%' FLOAT will return the floating point number (32-bits) located at 'addr%'</p>

```

complex1% = MATH(C_ASIN complex2%)
complex1% = MATH(C_ACOS complex2%)
complex1% = MATH(C_ATAN complex2%)
complex1% = MATH(C_SINH complex2%)
complex1% = MATH(C_COSH complex2%)
complex1% = MATH(C_TANH complex2%)
complex1% = MATH(C_ASINH complex2%)
complex1% = MATH(C_ACOSH complex2%)
complex1% = MATH(C_ATANH complex2%)
complex1% = MATH(C_PROJ complex2%)

```

基本算术

```

complex1% = MATH(C_ADD complex2%,complex3%)
complex1% = MATH(C_SUB complex2%,complex3%)
complex1% = MATH(C_MUL complex2%,complex3%)
complex1% = MATH(C_DIV complex2%,complex3%)
complex1% = MATH(C_POW complex2%,complex3%)
complex1% = MATH(C_AND complex2%,complex3%)
complex1% = MATH(C_OR complex2%,complex3%)
complex1% = MATH(C_XOR complex2%,complex3%)

```

MAX(arg1 [, arg2 [, ...]]) 或 MIN(arg1 [, arg2 [, ...]])	<p>返回参数列表中的最大或最小数字。 请注意，比较是浮点比较（整数参数被转换为浮点数），并返回一个浮点数。</p>
MID\$(string\$, start) 或 MID\$(string\$, start, nbr)	<p>返回从‘string\$’开始的子字符串，起始于‘start’，并持续‘nbr’个字符。 字符串中的第一个字符是编号1。 如果省略‘nbr’，返回的字符串将延伸到‘string\$’的末尾。</p>
MSGBOX (msg\$, b1\$ [,b2\$... b4\$])	<p>此函数将在屏幕上显示一个消息框，带有一到四个触摸敏感按钮。在用户触摸其中一个按钮之前，所有其他控件将被禁用。然后消息框将被擦除，之前的控件将被恢复，函数将返回被触摸按钮的编号（第一个按钮是编号一）。</p> <p>'msg' 是要显示的消息。这可以包含一个或多个波浪号字符(~)，表示换行。最多可以在框内显示10行。'b1' 是第一个按钮的标题，'b2' 是第二个按钮的标题，等等。必须指定至少一个按钮，最多可以指定四个。未包含在参数列表中的任何按钮将不会显示。</p>
OCT\$(数字 [, 字符])	<p>返回一个字符串，给出‘数字’的八进制（基数8）表示。 ‘字符’是可选的，指定字符串中字符的数量，零作为前导填充字符。</p>
PEEK(BYTE addr%) 或 PEEK(SHORT addr%) 或 PEEK(WORD addr%) 或 PEEK(INTEGER addr%) 或 PEEK(FLOAT addr%)	<p>将返回PIC32虚拟内存空间中的一个字节或一个字。 BYTE将返回位于‘addr%’的字节（8位） SHORT将返回位于‘addr%’的短整数（16位） WORD将返回位于‘addr%’的字（32位） INTEGER将返回位于‘addr%’的整数（64位） FLOAT将返回位于‘addr%’的浮点数（32位）</p>

or PEEK(VARADDR var) or PEEK(CFUNADDR cfun) or PEEK(VAR var, ±offset) or PEEK(VARTBL, ±offset) or PEEK(PROGMEM, ±offset)	<p>VARADDR will return the address (32-bits) of the variable 'var' in memory. An array is specified as var().</p> <p>CFUNADDR will return the address (32-bits) of the CFunction 'cfun' in memory. This address can be passed to another CFunction which can then call it to perform some common process.</p> <p>VAR, will return a byte in the memory allocated to 'var'. An array is specified as var().</p> <p>VARTBL, will return a byte in the memory allocated to the variable table maintained by MMBasic. Note that there is a comma after the keyword VARTBL.</p> <p>PROGMEM, will return a byte in the memory allocated to the program. Note that there is a comma after the keyword PROGMEM.</p> <p>Note that 'addr%' should be an integer.</p> <p>peek(bp n%) ' returns the byte at address n% and increments n% to point to the next byte</p> <p>peek(sp n%) ' returns the short at address n% and increments n% to point to the next short</p> <p>peek(wp n%) ' returns the word at address n% and increments n% to point to the next word</p>
PI	Returns the value of pi.
PIN(pin)	<p>Returns the value on the external I/O 'pin'. Zero means digital low, 1 means digital high and for analogue inputs it will return the measured voltage as a floating point number.</p> <p>Frequency inputs will return the frequency in Hz. A period input will return the period in milliseconds while a count input will return the count since reset (counting is done on the positive rising edge). The count input can be reset to zero by resetting the pin to counting input (even if it is already so configured).</p> <p>This function will also return the state of a pin configured as an output or a PIO pin.</p> <p>Also see the SETPIN and PIN() = commands. Refer to the section <i>Using the I/O pins</i> for a general description of the PicoMite's input/output capabilities.</p>
PIN(TEMP)	Returns the temperature of the RP2040 chip (see the RP2040 data sheet for the details)
PIO(DMA RX POINTER) PIO(DMA TX POINTER) PIO (SHIFTCTRL push_threshold <code>[,pull_threshold] [,autopush]</code> <code>[,autopull] [,in_shiftdir]</code> <code>[,out_shiftdir] [,fjoin_rx]</code> <code>[,fjoin_tx])</code>	<p>Returns the current data item being written or read by the PIO</p> <p>helper function to calculate the value of shiftctrl for the INIT MACHINE command</p>

或 PEEK(VARADDR var) 或 PEEK(CFUNADDR cfun) 或 PEEK(VAR var, \pm 偏移量) 或 PEEK(VARTBL, \pm offset) 或 PEEK(PROGMEM, \pm offset)	VARADDR 将返回变量 'var' 在内存中的地址（32 位）。 数组被指定为 var()。 CFUNADDR 将返回 CFunction 'cfun' 在内存中的地址（32 位）。 该地址可以传递给另一个 CFunction，然后可以调用它以执行一些常见的过程。 VAR 将返回分配给 'var' 的内存中的一个字节。数组被指定为 var()。 VARTBL 将返回分配给 MMBasic 维护的变量表中的一个字节。请注意，关键字 VARTBL 后面有一个逗号。 PROGMEM 将返回分配给程序的内存中的一个字节。注意关键字 PROGMEM 后面有一个逗号。 注意 'addr%' 应该是一个整数。
PEEK(BP, n%)	peek(bp n%)' 返回地址 n% 处的字节，并将 n% 增加到指向下一个字节
PEEK(SP,n%)	peek(sp n%)' 返回地址 n% 处的短整型，并将 n% 增加到指向下一个短整型 peek(wp n%)' 返回地址 n% 处的字，并将 n% 增加到指向下一个字
π	返回 π 的值。
PIN(pin)	返回外部 I/O ‘引脚’ 的值。零表示数字低，1 表示数字高，对于模拟输入，它将返回测量的电压作为浮点数。 频率输入将返回以赫兹为单位的频率。周期输入将返回以毫秒为单位的周期，而计数输入将返回自重置以来的计数（计数在正上升沿进行）。计数输入可以通过将引脚重置为计数输入来重置为零（即使它已经配置为如此）。 此函数还将返回配置为输出或 PIO 引脚的引脚状态。 另请参见 SETPIN 和 PIN()= 命令。请参阅章节使用 I/O 引脚以获取 Pic oMite 输入/输出能力的一般描述。
PIN(TEMP)	返回 RP2040 芯片的温度（有关详细信息，请参见 RP2040 数据手册）
PIO(DMA RX POINTER) PIO(DMA TX POINTER) PIO (SHIFTCTRL push_threshold [,pull_threshold] [,autopush] [,autopull] [,in_shiftdir] [,out_shiftdir] [,fjoin_rx] [,fjoin_tx])	返回当前由 PIO 写入或读取的数据项 计算 INIT MACHINE 命令的 shiftctrl 值的辅助函数

PIO (PINCTRL no_side_set_pins [,no_set_pins] [,no_out_pins] [,IN base] [,side_set_base] [,set_base][, out_base])	helper function to calculate the value of pinctrl for the INIT MACHINE command. Note: The pin parameters must be formatted as GPn.
PIO (EXECCTRL jmp_pin ,wrap_target, wrap ,side_pindir] [,side_en])	helper function to calculate the value of execctrl for the INIT MACHINE command
PIO (FDEBUG pio)	returns the value of the FSDEBUG register for the pio specified
PIO (FSTAT pio)	returns the value of the FSTAT register for the pio specified
PIO (FLEVEL pio)	returns the value of the FLEVEL register for the pio specified PIO(FLEVEL pio)
PIO(FLEVEL pio ,sm, DIR)	dir can be RX or TX. Returns the level of the specific fifo
PIO(.WRAP) PIO(.WRAP TARGET)	returns the location of the .wrap directive in PIO ASSEMBLE returns the location of the .wrap target directive in PIO ASSEMBLE. These can be used in the PIO(EXECCTRL function as follows: PIO (EXECCTRL jmp_pin PIO(.WRAP TARGET), PIO(.WRAP) [,side_pindir] [,side_en])
PORT(start, nbr [,start, nbr]...)	Returns the value of a number of I/O pins in one operation. 'start' is an I/O pin number and its value will be returned as bit 0. 'start'+1 will be returned as bit 1, 'start'+2 will be returned as bit 2, and so on for 'nbr' number of bits. I/O pins used must be numbered consecutively and any I/O pin that is invalid or not configured as an input will cause an error. The start/nbr pair can be repeated up to 25 times if additional groups of input pins need to be added. This function will also return the state of a pin configured as an output. It can be used to conveniently communicate with parallel devices like memory chips. Any number of I/O pins (and therefore bits) can be used from 1 to the number of I/O pins on the chip. See the PORT command to simultaneously output to a number of pins.
PIXEL(x, y)	Returns the colour of a pixel on an LCD display. 'x' is the horizontal coordinate and 'y' is the vertical coordinate of the pixel. The display must use one of the SSD1963, ILI9341, ILI9488, or ST7789_320 controllers.
PULSIN(pin, polarity) or PULSIN(pin, polarity, t1) or PULSIN(pin, polarity, t1, t2)	Measures the width of an input pulse from 1µs to 1 second with 0.1µs resolution. 'pin' is the I/O pin to use for the measurement, it must be previously configured as a digital input. 'polarity' is the type of pulse to measure, if zero the function will return the width of the next negative pulse, if non zero it will measure the next positive pulse. 't1' is the timeout applied while waiting for the pulse to arrive, 't2' is the timeout used while measuring the pulse. Both are in microseconds (µs) and are optional. If 't2' is omitted the value of 't1' will be used for both timeouts. If both 't1' and 't2' are omitted then the timeouts will be set at 100000 (i.e.

PIO (PINCTRL no_side_set_pins [,no_set_pins] [,no_out_pins] [,IN base] [,side_set_base] [,set_base][, out_base])	计算 INIT MACHINE 命令的 pinctrl 值的辅助函数。注意：引脚参数必须格式化为 GPn。
PIO (EXECCTRL jmp_pin ,wrap_target, wrap ,side_pmdir) [,side_en])	辅助函数，用于计算 INIT MACHINE 命令的 execctrl 值
PIO (FDEBUG pio)	返回指定 pio 的 FSDEBUG 寄存器的值
PIO (FSTAT pio)	返回指定 pio 的 FSTAT 寄存器的值
PIO (FLEVEL pio)	返回指定 pio 的 FLEVEL 寄存器的值 PIO(FLEVEL pio)
PIO(FLEVEL pio ,sm, DIR)	dir 可以是 RX 或 TX。返回特定 fifo 的电平
PIO(.WRAP) PIO(.WRAP TARGET)	返回 PIO ASSEMBLE 中 .wrap 指令的位置 返回 PIO ASSEMBLE 中 .wrap 目标指令的位置。 这些可以在 PIO(EXECCTRL 函数中使用，如下所示： PIO (EXECCTRL jmp_pin PIO(.WRAP TARGET), PIO(.WRAP) [,side_pmdir] [,side_en])
端口(起始, 编号 [,起始, 编号]...)	返回多个I/O引脚的值，进行一次操作。 '起始'是一个I/O引脚编号，其值将作为位0返回。'起始'+1将作为位1返回，'起始'+2将作为位2返回，以此类推，直到'nbr'位数。使用的I/O引脚必须连续编号，任何无效或未配置为输入的I/O引脚将导致错误。如果需要添加额外的输入引脚组，'起始/编号'对可以重复最多25次。 此函数还将返回配置为输出的引脚的状态。它可以方便地与并行设备（如内存芯片）进行通信。 可以使用任意数量的I/O引脚（因此位数）从1到芯片上的I/O引脚数量。 。 请参见PORT命令以同时输出到多个引脚。
PIXEL(x, y)	返回液晶显示器上一个像素的颜色。'x' 是像素的水平坐标，'y' 是像素的垂直坐标。显示器必须使用SSD1963、ILI9341、ILI9488或ST789_320控制器之一。
PULSIN(pin, polarity) 或 PULSIN(pin, polarity, t1) 或 PULSIN(pin, polarity, t1, t2)	测量输入脉冲的宽度，范围从1μs到1秒，分辨率为0.1μs。 'pin'是用于测量的输入/输出引脚，必须事先配置为数字输入。'polarity'是要测量的脉冲类型，如果为零，函数将返回下一个负脉冲的宽度；如果非零，则测量下一个正脉冲。 't1'是在等待脉冲到达时应用的超时，'t2'是在测量脉冲时使用的超时。两者均以微秒（μs）为单位，并且是可选的。如果省略't2'，则将使用't1'的值作为两个超时的值。如果省略't1'和't2'，则超时将设置为100 000（即）。

	<p>100ms).</p> <p>This function returns the width of the pulse in microseconds (μs) or -1 if a timeout has occurred. The measurement is accurate to $\pm 0.5\%$ and $\pm 0.5\mu$s. Note that this function will cause the running program to pause while the measurement is made and interrupts will be ignored during this period.</p>
RAD(degrees)	Converts 'degrees' to radians.
RGB(red, green, blue) or RGB(shortcut)	<p>Generates an RGB true colour value.</p> <p>'red', 'blue' and 'green' represent the intensity of each colour. A value of zero represents black and 255 represents full intensity.</p> <p>'shortcut' allows common colours to be specified by naming them. The colours that can be named are white, black, blue, green, cyan, red, magenta, yellow, brown, white, orange, pink, gold, salmon, beige, lightgrey and grey (or USA spelling gray/lightgray). For example, RGB(red) or RGB(cyan).</p> <p>Note that the value returned is an integer and, if it is to be saved, the variable should be declared as an integer to retain the accuracy of the number.</p>
RIGHT\$(string\$, number-of-chars)	Returns a substring of 'string\$' with 'number-of-chars' from the right (end) of the string.
RND(number) or RND	Returns a pseudo-random number in the range of 0 to 0.999999. The 'number' value is ignored if supplied. The RANDOMIZE command reseeds the random number generator.
SGN(number)	Returns the sign of the argument 'number', +1 for positive numbers, 0 for 0, and -1 for negative numbers.
SIN(number)	Returns the sine of the argument 'number' in radians.
SPACE\$(number)	Returns a string of blank spaces 'number' characters long.
SPI (data) or SPI2 (data)	<p>Send and receive data using an SPI channel.</p> <p>A single SPI transaction will send data while simultaneously receiving data from the slave. 'data' is the data to send and the function will return the data received during the transaction. 'data' can be an integer or a floating point variable or a constant.</p>
SQR(number)	Returns the square root of the argument 'number'.
STR\$(number) or STR\$(number, m) or STR\$(number, m, n) or STR\$(number, m, n, c\$)	<p>Returns a string in the decimal (base 10) representation of 'number'.</p> <p>If 'm' is specified sufficient spaces will be added to the start of the number to ensure that the number of characters before the decimal point (including the negative or positive sign) will be at least 'm' characters. If 'm' is zero or the number has more than 'm' significant digits no padding spaces will be added.</p> <p>If 'm' is negative, positive numbers will be prefixed with the plus symbol and negative numbers with the negative symbol. If 'm' is positive then only the negative symbol will be used.</p> <p>'n' is the number of digits required to follow the decimal place. If it is zero the string will be returned without the decimal point. If it is negative the output will always use the exponential format with 'n' digits resolution. If 'n' is not specified the number of decimal places and output format will vary.</p>

	<p>100毫秒)。</p> <p>此函数返回脉冲的宽度，以微秒 (μs) 为单位，如果发生超时，则返回-1。测量精度为$\pm 0.5\%$和$\pm 0.5\mu\text{s}$。请注意，此函数将在测量期间导致正在运行的程序暂停，并且在此期间将忽略中断。</p>
RAD(度数)	将'度数'转换为弧度。
RGB(红色, 绿色, 蓝色) 或 RGB(快捷方式)	<p>生成一个RGB真彩色值。</p> <p>'红色'、'蓝色'和'绿色'表示每种颜色的强度。值为零表示黑色，255表示全强度。</p> <p>'shortcut' 允许通过命名来指定常见颜色。可以命名的颜色有白色、黑色、蓝色、绿色、青色、红色、品红色、黄色、棕色、白色、橙色、粉色、金色、鲑鱼色、米色、浅灰色和灰色（或美国拼写的 gray/lightgray）。例如，RGB(red) 或 RGB(cyan)。请注意，返回的值是一个整数，如果要保存，该变量应声明为整数以保持数字的准确性。</p>
RIGHT\$(string\$, number-of-chars)	返回‘string\$’的子字符串，长度为‘number-of-chars’，从字符串的右侧（末尾）开始。
RND(number) 或 RND	返回范围在 0 到 0.999999 之间的伪随机数。如果提供了‘number’值，则会被忽略。RANDOMIZE 命令重新设置随机数生成器的种子。
SGN(数字)	返回参数 '数字' 的符号，正数返回 +1，0 返回 0，负数返回 -1。
SIN(数字)	返回参数 '数字' 的正弦值（以弧度为单位）。
SPACE\$(数字)	返回一个长度为 '数字' 个字符的空格字符串。
SPI (数据) 或 SPI2 (数据)	<p>使用 SPI 通道发送和接收数据。</p> <p>单个 SPI 事务将在发送数据的同时从从属设备接收数据。‘数据’是要发送的数据，函数将返回在事务期间接收到的数据。‘数据’可以是整数、浮点变量或常量。</p>
SQR(数字)	返回参数 '数字' 的平方根。
STR\$(数字) 或 STR\$(数字, m) 或 STR\$(数字, m, n) 或 STR\$(数字, m, n, c\$)	<p>返回一个数字的十进制（基数10）表示的字符串。</p> <p>如果指定了'm'，将会在数字的开头添加足够的空格，以确保小数点前的字符数（包括负号或正号）至少为'm'个字符。如果'm'为零或数字的有效数字超过'm'，则不会添加填充空格。</p> <p>如果'm'为负数，正数将以加号作为前缀，负数将以负号作为前缀。如果'm'为正数，则只会使用负号。</p> <p>'n'是小数点后所需的数字位数。如果为零，则字符串将返回而不带小数点。如果为负数，输出将始终使用指数格式，精度为'n'位。如果未指定'n'，小数位数和输出格式将会有所不同。</p>

	<p>automatically according to the number.</p> <p>'c\$' is a string and if specified the first character of this string will be used as the padding character instead of a space (see the 'm' argument).</p> <p>Examples:</p> <table> <tbody> <tr><td>STR\$(123.456)</td><td>will return " 123 . 456 "</td></tr> <tr><td>STR\$(-123.456)</td><td>will return "-123 . 456 "</td></tr> <tr><td>STR\$(123.456, 1)</td><td>will return " 123 . 456 "</td></tr> <tr><td>STR\$(123.456, -1)</td><td>will return "+123 . 456 "</td></tr> <tr><td>STR\$(123.456, 6)</td><td>will return " 123 . 456 "</td></tr> <tr><td>STR\$(123.456, -6)</td><td>will return "+123 . 456 "</td></tr> <tr><td>STR\$(-123.456, 6)</td><td>will return "-123 . 456 "</td></tr> <tr><td>STR\$(-123.456, 6, 5)</td><td>will return "-123 . 45600 "</td></tr> <tr><td>STR\$(-123.456, 6, -5)</td><td>will return "-1 . 23456e+02 "</td></tr> <tr><td>STR\$(53, 6)</td><td>will return " 53 "</td></tr> <tr><td>STR\$(53, 6, 2)</td><td>will return " 53 . 00 "</td></tr> <tr><td>STR\$(53, 6, 2, "*")</td><td>will return " ****53 . 00 "</td></tr> </tbody> </table>	STR\$(123.456)	will return " 123 . 456 "	STR\$(-123.456)	will return "-123 . 456 "	STR\$(123.456, 1)	will return " 123 . 456 "	STR\$(123.456, -1)	will return "+123 . 456 "	STR\$(123.456, 6)	will return " 123 . 456 "	STR\$(123.456, -6)	will return "+123 . 456 "	STR\$(-123.456, 6)	will return "-123 . 456 "	STR\$(-123.456, 6, 5)	will return "-123 . 45600 "	STR\$(-123.456, 6, -5)	will return "-1 . 23456e+02 "	STR\$(53, 6)	will return " 53 "	STR\$(53, 6, 2)	will return " 53 . 00 "	STR\$(53, 6, 2, "*")	will return " ****53 . 00 "
STR\$(123.456)	will return " 123 . 456 "																								
STR\$(-123.456)	will return "-123 . 456 "																								
STR\$(123.456, 1)	will return " 123 . 456 "																								
STR\$(123.456, -1)	will return "+123 . 456 "																								
STR\$(123.456, 6)	will return " 123 . 456 "																								
STR\$(123.456, -6)	will return "+123 . 456 "																								
STR\$(-123.456, 6)	will return "-123 . 456 "																								
STR\$(-123.456, 6, 5)	will return "-123 . 45600 "																								
STR\$(-123.456, 6, -5)	will return "-1 . 23456e+02 "																								
STR\$(53, 6)	will return " 53 "																								
STR\$(53, 6, 2)	will return " 53 . 00 "																								
STR\$(53, 6, 2, "*")	will return " ****53 . 00 "																								
STR2BIN(type, string\$ [,BIG])	<p>Returns a number equal to the binary representation in 'string\$'.</p> <p>'type' can be:</p> <p>INT64 converts 8 byte string representing a signed 64-bit integer to an integer</p> <p>UINT64 converts 8 byte string representing an unsigned 64-bit integer to an integer</p> <p>INT32 converts 4 byte string representing a signed 32-bit integer to an integer</p> <p>UINT32 converts 4 byte string representing an unsigned 32-bit integer to an integer</p> <p>INT16 converts 2 byte string representing a signed 16-bit integer to an integer</p> <p>UINT16 converts 2 byte string representing an unsigned 16-bit integer to an integer</p> <p>INT8 converts 1 byte string representing a signed 8-bit integer to an integer</p> <p>UINT8 converts 1 byte string representing an unsigned 8-bit integer to an integer</p> <p>SINGLE converts 4 byte string representing single precision float to a float</p> <p>DOUBLE converts 8 byte string representing single precision float to a float</p> <p>By default the string must contain the number in little-endian format (i.e. the least significant byte is the first one in the string). Setting the third parameter to 'BIG' will interpret the string in big-endian format (i.e. the most significant byte is the first one in the string).</p> <p>This function makes it easy to read data from binary data files, interpret numbers from sensors or efficiently read binary data from flash memory chips.</p> <p>An error will be generated if the string is the incorrect length for the conversion requested</p> <p>See also the function BIN2STR\$</p>																								
STRING\$(nbr, ascii) or STRING\$(nbr, string\$)	Returns a string 'nbr' bytes long consisting of either the first character of string\$ or the character representing the ASCII value 'ascii' which is an integer or float number in the range of 0 to 255.																								
TAB(number)	Outputs spaces until the column indicated by 'number' has been reached on the console output.																								

	<p>根据数字自动调整。 'c' 是一个字符串，如果指定，则该字符串的第一个字符将用作填充字符，而不是空格（请参见 'm' 参数）。</p> <p>示例：</p> <table border="0"> <tbody> <tr><td>STR\$(123.456)</td><td>将返回 "123.456"</td></tr> <tr><td>STR\$(-123.456)</td><td>将返回 "-123.456"</td></tr> <tr><td>STR\$(123.456, 1)</td><td>将返回 "123.456"</td></tr> <tr><td>STR\$(123.456, -1)</td><td>将返回 "+123.456"</td></tr> <tr><td>STR\$(123.456, 6)</td><td>将返回 " 123.456"</td></tr> <tr><td>STR\$(123.456, -6)</td><td>将返回 "+123.456"</td></tr> <tr><td>STR\$(-123.456, 6)</td><td>将返回 "-123.456"</td></tr> <tr><td>STR\$(-123.456, 6, 5)</td><td>将返回 "-123.45600"</td></tr> <tr><td>STR\$(-123.456, 6, -5)</td><td>将返回 "-1.23456e+02"</td></tr> <tr><td>STR\$(53, 6)</td><td>将返回 " 53"</td></tr> <tr><td>STR\$(53, 6, 2)</td><td>将返回 " 53.00"</td></tr> <tr><td>STR\$(53, 6, 2, "*")</td><td>将返回 "****53.00"</td></tr> </tbody> </table>	STR\$(123.456)	将返回 "123.456"	STR\$(-123.456)	将返回 "-123.456"	STR\$(123.456, 1)	将返回 "123.456"	STR\$(123.456, -1)	将返回 "+123.456"	STR\$(123.456, 6)	将返回 " 123.456"	STR\$(123.456, -6)	将返回 "+123.456"	STR\$(-123.456, 6)	将返回 "-123.456"	STR\$(-123.456, 6, 5)	将返回 "-123.45600"	STR\$(-123.456, 6, -5)	将返回 "-1.23456e+02"	STR\$(53, 6)	将返回 " 53"	STR\$(53, 6, 2)	将返回 " 53.00"	STR\$(53, 6, 2, "*")	将返回 "****53.00"
STR\$(123.456)	将返回 "123.456"																								
STR\$(-123.456)	将返回 "-123.456"																								
STR\$(123.456, 1)	将返回 "123.456"																								
STR\$(123.456, -1)	将返回 "+123.456"																								
STR\$(123.456, 6)	将返回 " 123.456"																								
STR\$(123.456, -6)	将返回 "+123.456"																								
STR\$(-123.456, 6)	将返回 "-123.456"																								
STR\$(-123.456, 6, 5)	将返回 "-123.45600"																								
STR\$(-123.456, 6, -5)	将返回 "-1.23456e+02"																								
STR\$(53, 6)	将返回 " 53"																								
STR\$(53, 6, 2)	将返回 " 53.00"																								
STR\$(53, 6, 2, "*")	将返回 "****53.00"																								
STR2BIN(type, string\$ [,BIG])	<p>返回一个等于‘string\$’中二进制表示的数字。</p> <p>‘type’可以是：</p> <p>INT64 将8字节字符串表示的有符号64位整数转换为整数 UINT64 将8字节字符串表示的无符号64位整数转换为整数</p> <p>INT32 将4字节字符串表示的有符号32位整数转换为整数 UINT32 将4字节字符串表示的无符号32位整数转换为整数</p> <p>INT16 将2字节字符串表示的有符号16位整数转换为整数 UINT16 将2字节字符串表示的无符号16位整数转换为整数</p> <p>INT8 将1字节字符串表示的有符号8位整数转换为整数 UINT8 将1字节字符串表示的无符号8位整数转换为整数</p> <p>SINGLE 将表示单精度浮点数的 4 字节字符串转换为浮点数，DOUBLE 将表示单精度浮点数的 8 字节字符串转换为浮点数。默认情况下，字符串必须包含以小端格式表示的数字（即，最低有效字节是字符串中的第一个字节）。将第三个参数设置为 ‘BIG’ 将以大端格式解释字符串（即，最高有效字节是字符串中的第一个字节）。</p> <p>此函数使从二进制数据文件读取数据、从传感器解释数字或高效地从闪存芯片读取二进制数据变得简单。</p> <p>如果字符串的长度不正确以进行请求的转换，将生成错误。另请参见函数 BIN2S TR\$。</p>																								
STRING\$(nbr, ascii) 或 STRING\$(nbr, string\$)	返回一个长度为 'nbr' 字节的字符串，该字符串由 string\$ 的第一个字符或表示 ASCII 值 'ascii' 的字符组成，'ascii' 是一个范围在 0 到 255 之间的整数或浮点数。																								
TAB(数字)	输出空格，直到控制台输出达到‘数字’所指示的列。																								

TAN(number)	Returns the tangent of the argument 'number' in radians.
TEMPR(pin)	<p>Return the temperature measured by a DS18B20 temperature sensor connected to 'pin' (which does not have to be configured).</p> <p>The returned value is degrees C with a default resolution of 0.25°C. If there is an error during the measurement the returned value will be 1000.</p> <p>The time required for the overall measurement is 200ms and interrupts will be ignored during this period. Alternatively the TEMPR START command can be used to start the measurement and your program can do other things while the conversion is progressing. When this function is called the value will then be returned instantly assuming the conversion period has expired. If it has not, this function will wait out the remainder of the conversion time before returning the value.</p> <p>The DS18B20 can be powered separately by a 3V to 5V supply or it can operate on parasitic power from the PicoMite.</p> <p>See the section <i>Special Hardware Devices</i> for more details.</p>
TIME\$	<p>Returns the current time based on MMBasic's internal clock as a string in the form "HH:MM:SS" in 24 hour notation. For example, "14:30:00".</p> <p>To set the current time use the command TIME\$ = .</p>
TIMER	<p>Returns the elapsed time in milliseconds (e.g. 1/1000 of a second) since reset.</p> <p>The timer is reset to zero on power up or a CPU restart and you can also reset it by using TIMER as a command. If not specifically reset it will continue to count up forever (it is a 64 bit number and therefore will only roll over to zero after 200 million years).</p>
TOUCH(X) or TOUCH(Y)	<p>Will return the X or Y coordinate of the location currently touched on an LCD panel.</p> <p>If the screen is not being touched the function will return -1.</p>
UCASE\$(string\$)	Returns 'string\$' converted to uppercase characters.
VAL(string\$)	<p>Returns the numerical value of the 'string\$'. If 'string\$' is an invalid number the function will return zero.</p> <p>This function will recognise the &H prefix for a hexadecimal number, &O for octal and &B for binary.</p>

TAN(数字)	返回参数'数字'的正切值（以弧度为单位）。
TEMPr(引脚)	<p>返回连接到'引脚'的DS18B20温度传感器测量的温度（该引脚无需配置）。</p> <p>返回的值为摄氏度， 默认分辨率为0.25°C。如果测量过程中出现错误，返回值将为1000。</p> <p>整体测量所需的时间为200毫秒，在此期间将忽略中断。或者，可以使用TEMPr START命令来启动测量，您的程序可以在转换进行时执行其他操作。当调用此函数时，假设转换周期已过，将立即返回值。如果尚未过期，此函数将在返回值之前等待剩余的转换时间。</p> <p>DS18B20可以通过3V到5V的电源单独供电，或者可以从PicoMite获取寄生电源。有关更多详细信息，请参见章节特殊硬件设备。</p>
TIME\$	<p>返回基于MMBasic内部时钟的当前时间，格式为字符串"HH:MM:SS"，采用24小时制。例如，"14:30:00"。</p> <p>要设置当前时间，请使用命令TIME\$ =。</p>
定时器	<p>返回自重置以来经过的时间（以毫秒为单位，例如1/1000秒）。</p> <p>定时器在上电或CPU重启时重置为零，您也可以通过使用TIMER作为命令来重置它。如果没有特别重置，它将继续无限计数（这是一个64位数字，因此在2亿年后才会回滚到零）。</p>
TOUCH(X) 或 TOUCH(Y)	<p>将返回当前在液晶面板上触摸位置的 X 或 Y 坐标。</p> <p>如果屏幕没有被触摸，该函数将返回 -1。</p>
UCASE\$(string\$)	返回转换为大写字母的 'string\$'。
VAL(string\$)	<p>返回 'string\$' 的数值。如果 'string' 是无效数字，该函数将返回零。</p> <p>该函数将识别十六进制数字的 &H 前缀，八进制的 &O 前缀和二进制的 &B 前缀。</p>

Obsolete Commands and Functions

These commands and functions are mostly included to assist in converting programs written for Microsoft BASIC. For new programs the corresponding modern commands in MMBasic should be used.

Note that these commands may be removed in the future to recover memory for other features.

BITBANG	Replaced by the command DEVICE. For compatibility BITBANG can still be used in programs and will be automatically converted to DEVICE
GOSUB target	Initiates a subroutine call to the target, which can be a line number or a label. The subroutine must end with RETURN. New programs should use defined subroutines (i.e. SUB...END SUB).
IF condition THEN linenbr	For Microsoft compatibility a GOTO is assumed if the THEN statement is followed by a number. A label is invalid in this construct. New programs should use: IF condition THEN GOTO linenbr label
IRETURN	Returns from an interrupt when the interrupt destination was a line number or a label. New programs should use a user defined subroutine as an interrupt destination. In that case END SUB or EXIT SUB will cause a return from the interrupt.
ON nbr GOTO GOSUB target[,target, target,...]	ON either branches (GOTO) or calls a subroutine (GOSUB) based on the rounded value of 'nbr'; if it is 1, the first target is called, if 2, the second target is called, etc. Target can be a line number or a label. New programs should use SELECT CASE.
POS	For the console, returns the current cursor position in the line in characters.
PAGE	Replaced with “GUI PAGE”
RETURN	RETURN concludes a subroutine called by GOSUB and returns to the statement after the GOSUB.

过时的命令和函数

这些命令和函数主要是为了帮助转换为微软 BASIC 编写的程序。对于新程序，应使用 MMBasic 中相应的现代命令。

请注意，这些命令可能会在未来被移除，以释放内存用于其他功能。

位翻转	被命令 DEVICE 替代。为了兼容性，BITBANG 仍然可以在程序中使用，并将自动转换为 DEVICE。
GOSUB target	发起对目标的子程序调用，目标可以是行号或标签。 子程序必须以 RETURN 结束。 新程序应使用定义的子程序（即 SUB...END SUB）。
如果条件 THEN 行编号	为了与微软兼容，如果 THEN 语句后跟一个数字，则假定为 GOTO。 在此结构中，标签无效。 新程序应使用：IF 条件 THEN GOTO 行编号 标签
IRETURN	当中断目标是行编号或标签时，从中断返回。 新程序应使用用户定义的子程序作为中断目标。 在这种情况下，END SUB 或 EXITSUB 将导致从中断返回。
ON 编号 GOTO GOSUB target[,target, target,...]	ON 根据 'nbr' 的四舍五入值选择分支 (GOTO) 或调用子程序 (GO SUB)；如果是 1，则调用第一个目标，如果是 2，则调用第二个目标，依此类推。目标可以是行编号或标签。 新程序应使用 SELECT CASE。
位置	对于控制台，返回当前光标在行中的字符位置。
页面	已替换为“图形用户界面页面”
返回	RETURN 结束由 GOSUB 调用的子程序，并返回到 GOSUB 之后的语句。

Appendix A

Serial Communications

Two serial interfaces are available for asynchronous serial communications. They are labelled COM1: and COM2:.

I/O Pins

Before a serial interface can be used the I/O pins must be defined using the following command for the first channel (referred as COM1):

```
SETPIN rx, tx, COM1
```

Valid pins are	RX: GP1, GP13 or GP17
	TX: GP0, GP12, GP16 or GP28

And the following command for the second channel (referred to as COM2):

```
SETPIN rx, tx, COM2
```

Valid pins are	RX: GP5, GP9 or GP21
	TX: GP4, GP8 or GP20

TX is data from the PicoMite and RX is data to it.

The signal polarity is standard for devices running at TTL voltages. Idle is voltage high, the start bit is voltage low, data uses a high voltage for logic 1 and the stop bit is voltage high. These signal levels allow you to directly connect to devices like GPS modules (which generally use TTL voltage levels).

Commands

After being opened the serial port will have an associated file number and you can use any commands that operate with a file number to read and write to/from it. A serial port can be closed using the CLOSE command.

The following is an example:

```
SETPIN GP13, GP16, COM1      ' assign the I/O pins for the first serial port
OPEN "COM1:4800" AS #5       ' open the first serial port with a speed of 4800 baud
PRINT #5, "Hello"            ' send the string "Hello" out of the serial port
dat$ = INPUT$(20, #5)         ' get up to 20 characters from the serial port
CLOSE #5                      ' close the serial port
```

The OPEN Command

A serial port is opened using the command:

```
OPEN comspec$ AS #fnbr
```

'fnbr' is the file number to be used. It must be in the range of 1 to 10. The # is optional.

'comspec\$' is the communication specification and is a string (it can be a string variable) specifying the serial port to be opened and optional parameters. The default is 9600 baud, 8 data bits, no parity and one stop bit.

It has the form "COMn: baud, buf, int, int-trigger, EVEN, ODD, S2, 7BIT" where:

- 'n' is the serial port number for either COM1: or COM2:.
- 'baud' is the baud rate. This can be any number from 1200 to 921600. Default is 9600.
- 'buf' is the receive buffer size in bytes (default size is 256). The transmit buffer is fixed at 256 bytes.
- 'int' is interrupt subroutine to be called when the serial port has received some data.
- 'int-trigger' is the number of characters received which will trigger an interrupt.

All parameters except the serial port name (COMn:) are optional. If any one parameter is left out then all the following parameters must also be left out and the defaults will be used.

Five options can be added to the end of 'comspec\$'. These are:

- 'S2' specifies that two stop bits will be sent following each character transmitted.
- EVEN specifies that an even parity bit will be applied, this will result in a 9-bit transfer unless 7BIT is set.
- ODD specifies that an odd parity bit will be applied, this will result in a 9-bit transfer unless 7BIT is set
- 7BIT specifies that there a 7bits of data. This is normally used with EVEN or ODD
- INV specifies that the output signals will be inverted and input assumed to be inverted

A 串行通信

串行通信

提供两个串行接口用于异步串行通信。它们标记为 COM1: 和 COM2:。

I/O 引脚

在使用串行接口之前，必须使用以下命令定义 I/O 引脚，针对第一个通道（称为 COM1）：

```
SETPIN rx, tx, COM1
```

有效引脚为 RX: GP1, GP13 或 GP17
 TX: GP0, GP12, GP16 或 GP28

以及针对第二个通道（称为 COM2）的以下命令：

```
SETPIN rx, tx, COM2
```

有效引脚为 RX: GP5, GP9 或 GP21
 TX: GP4, GP8 或 GP20

TX 是来自 PicoMite 的数据，RX 是发送给它的数据。

信号极性是标准的，适用于运行在 TTL 电压下的设备。空闲时电压为高，起始位电压为低，数据使用高电压表示逻辑 1，停止位电压为高。这些信号电平允许您直接连接到像 GPS 模块这样的设备（通常使用 TTL 电压电平）。

命令

串行端口打开后将有一个关联的文件编号，您可以使用任何与文件编号操作的命令来读取和写入数据。可以使用 CLOSE 命令关闭串行端口。

以下是一个示例：

```
SETPIN GP13, GP16, COM1      ' 为第一个串行端口分配 I/O 引脚
OPEN "COM1:4800" AS #5       ' 以 4800 波特率打开第一个串行端口
PRINT #5, "你好"             ' 将字符串 "你好" 发送到串行端口
dat$ = INPUT$( 20, #5)        ' 从串行端口获取最多 20 个字符
CLOSE #5                      ' 关闭串行端口
```

打开命令

使用以下命令打开串行端口：

```
OPEN comspec $ AS #fnbr
```

'fnbr' 是要使用的文件编号。它必须在 1 到 10 的范围内。# 是可选的。

'comspec\$' 是通信规范，是一个字符串（可以是字符串变量），指定要打开的串行端口和可选参数。默认值为 9600 波特率，8 位数据，无奇偶校验和一个停止位。其形式为 "COMn: baud, buf, int, int-trigger, EVEN, ODD, S2, 7BIT" 其中：

- 'n' 是串行端口编号，可以是 COM1: 或 COM2:。
- 'baud' 是波特率。这个值可以是从 1200 到 921600 的任何数字。默认值是 9600。
- 'buf' 是接收缓冲区的大小（默认大小为 256 字节）。传输缓冲区固定为 256 字节。
- 'int' 是当串行端口接收到一些数据时要调用的中断子程序。
- 'int-trigger' 是接收到的字符数量，这将触发一个中断。

除了串行端口名称 (COMn:) 外，所有参数都是可选的。如果省略任何一个参数，则所有后续参数也必须省略，并将使用默认值。

可以在 'comspec' 的末尾添加五个选项。这些选项是：

- 'S2' 指定在每个传输的字符后将发送两个停止位。
- EVEN 指定将应用偶数校验位，这将导致 9 位传输，除非设置为 7BIT。
- ODD 指定将应用奇数校验位，这将导致 9 位传输，除非设置为 7BIT。
- 7BIT 指定有 7 位数据。这通常与偶数或奇数一起使用。
- INV 指定输出信号将被反转，输入假定为反转。

Examples

Opening a serial port using all the defaults:

```
OPEN "COM1:" AS #2
```

Opening a serial port specifying only the baud rate (4800 bits per second):

```
OPEN "COM1:4800" AS #1
```

Opening a serial port specifying the baud rate (9600 bits per second) and receive buffer size (1KB):

```
OPEN "COM2:9600, 1024" AS #8
```

The same as above but with two stop bits enabled:

```
OPEN "COM2:9600, 1024, S2" AS #8
```

An example specifying everything including an interrupt, an interrupt level, and two stop bits:

```
OPEN "COM2:19200, 1024, ComIntLabel, 256, S2" AS #5
```

Reading and Writing

Once a serial port has been opened you can use any command or function that uses a file number to read from and write to the port. Data received by the serial port will be automatically buffered in memory by MMBasic until it is read by the program and the INPUT\$() function is the most convenient way of doing that. When using the INPUT\$() function the number of characters specified will be the maximum number of characters returned but it could be less if there are less characters in the receive buffer. In fact the INPUT\$() function will immediately return an empty string if there are no characters available in the receive buffer.

The LOC() function is also handy; it will return the number of characters waiting in the receive buffer (i.e. the maximum number characters that can be retrieved by the INPUT\$() function). Note that if the receive buffer overflows with incoming data the serial port will automatically discard the oldest data to make room for the new data.

The PRINT command is used for outputting to a serial port and any data to be sent will be held in a memory buffer while the serial port is sending it. This means that MMBasic will continue with executing the commands after the PRINT command while the data is being transmitted. The one exception is if the output buffer is full and in that case MMBasic will pause and wait until there is sufficient space before continuing. The LOF() function will return the amount of space left in the transmit buffer and you can use this to avoid stalling the program while waiting for space in the buffer to become available.

If you want to be sure that all the data has been sent (perhaps because you want to read the response from the remote device) you should wait until the LOF() function returns 256 (the transmit buffer size) indicating that there is nothing left to be sent.

Serial ports can be closed with the CLOSE command. This will wait for the transmit buffer to be emptied then free up the memory used by the buffers and cancel the interrupt (if set). A serial port is also automatically closed when commands such as RUN and NEW are issued.

Interrupts

The interrupt subroutine (if specified) will operate the same as a general interrupt on an external I/O pin (see the section *Using the I/O pins* for a description).

When using interrupts you need to be aware that it will take some time for MMBasic to respond to the interrupt and more characters could have arrived in the meantime, especially at high baud rates. For example, if you have specified the interrupt level as 200 characters and a buffer of 256 characters then quite easily the buffer will have overflowed by the time the interrupt subroutine can read the data. In this case the buffer should be increased to 512 characters or more.

示例

使用所有默认设置打开串行端口：

```
OPEN "COM1:" AS #2
```

打开串行端口，仅指定波特率（4800位每秒）：

```
OPEN "COM1:4800" AS #1
```

打开串行端口，指定波特率（9600位每秒）和接收缓冲区大小（1KB）：

```
OPEN "COM2:9600, 1024" AS #8
```

与上述相同，但启用了两个停止位：

```
OPEN "COM2:9600, 1024, S2" AS #8
```

一个示例，指定所有内容，包括中断、中断级别和两个停止位：

```
OPEN "COM2:19200, 1024, ComIntLabel, 256, S2" AS #5
```

读取和写入

一旦串行端口被打开，您可以使用任何使用文件编号的命令或函数来读取和写入该端口。通过串行端口接收的数据将由MMBasic自动缓冲在内存中，直到程序读取它，而INPUT\$()函数是最方便的方式来做到这一点。使用INPUT\$()函数时，指定的字符数将是返回的最大字符数，但如果接收缓冲区中的字符较少，返回的字符数可能会更少。实际上，如果接收缓冲区中没有可用字符，INPUT\$()函数将立即返回一个空字符串。

LOC()函数也很方便；它将返回接收缓冲区中等待的字符数（即INPUT\$()函数可以检索的最大字符数）。请注意，如果接收缓冲区因接收数据而溢出，串行端口将自动丢弃最旧的数据，以为新数据腾出空间。

PRINT命令用于向串行端口输出，任何要发送的数据将在串行端口发送时保留在内存缓冲区中。这意味着MMBasic将在数据传输时继续执行PRINT命令后的命令。唯一的例外是如果输出缓冲区已满，在这种情况下，MMBasic将暂停并等待，直到有足够的空间再继续。LOF() 函数将返回传输缓冲区中剩余的空间量，您可以使用它来避免在等待缓冲区可用空间时程序停滞。

如果您想确保所有数据都已发送（可能是因为您想读取远程设备的响应），您应该等待直到 LOF() 函数返回 256（传输缓冲区大小），这表示没有剩余数据需要发送。

可以使用 CLOSE 命令关闭串行端口。这将等待传输缓冲区被清空，然后释放缓冲区使用的内存并取消中断（如果已设置）。当发出 RUN 和 NEW 等命令时，串行端口也会自动关闭。

中断

中断子程序（如果指定）将与外部 I/O 引脚上的一般中断相同（请参见使用 I/O 引脚部分以获取描述）。

使用中断时，您需要意识到MMBasic对中断的响应需要一些时间在此期间可能会有更多字符到达，尤其是在高波特率下。例如，如果您将中断级别指定为200个字符，并且缓冲区为256个字符，那么在中断子程序能够读取数据时，缓冲区很容易就会溢出。在这种情况下，缓冲区应该增加到512个字符或更多。

Appendix B

I²C Communications

There are two I²C channels. They can operate in master or slave mode.

I/O Pins

Before the I²C interface can be used the I/O pins must be defined using the following command for the first channel (referred as I2C):

SETPIN sda, scl, I2C

Valid pins are SDA: GP0, GP4, GP8, GP12, GP16, GP20 or GP28
 SCL: GP1, GP5, GP9, GP13, GP17 or GP21

And the following command for the second channel (referred to as I2C2):

SETPIN sda, scl, I2C2

Valid pins are SDA: GP2, GP6, GP10, GP14, GP18, GP22 or GP26
 SCL: GP3, GP7, GP11, GP15, GP19 or GP27

When running the I²C bus at above 100kHz the cabling between the devices becomes important. Ideally the cables should be as short as possible (to reduce capacitance) and the data and clock lines should not run next to each other but have a ground wire between them (to reduce crosstalk).

If the data line is not stable when the clock is high, or the clock line is jittery, the I²C peripherals can get "confused" and end up locking the bus (normally by holding the clock line low). If you do not need the higher speeds then operating at 100 kHz is the safest choice.

I²C Master Commands

There are four commands that can be used for the first channel (I2C) in master mode as follows.

The commands for the second channel (I2C2) are identical except that the command is I2C2

I2C OPEN speed, timeout	Enables the I ² C module in master mode. The I2C command refers to channel 1 while the command I2C2 refers to channel 2 using the same syntax. 'speed' is the clock speed (in KHz) to use and must be either 100 or 400. 'timeout' is a value in milliseconds after which the master send and receive commands will be interrupted if they have not completed. The minimum value is 100. A value of zero will disable the timeout (though this is not recommended).
I2C WRITE addr, option, sendlen, senddata [,senddata ..]	Send data to the I ² C slave device. The I2C command refers to channel 1 while the command I2C2 refers to channel 2 using the same syntax. 'addr' is the slave's I ² C address. 'option' can be 0 for normal operation or 1 to keep control of the bus after the command (a stop condition will not be sent at the completion of the command) 'sendlen' is the number of bytes to send. 'senddata' is the data to be sent - this can be specified in various ways (all data sent will be sent as bytes with a value between 0 and 255): <ul style="list-style-type: none">• The data can be supplied as individual bytes on the command line. Example: I2C WRITE &H6F, 0, 3, &H23, &H43, &H25• The data can be in a one dimensional array specified with empty brackets (i.e. no dimensions). 'sendlen' bytes of the array will be sent starting with the first element. Example: I2C WRITE &H6F, 0, 3, ARRAY()• The data can be a string variable (not a constant). Example: I2C WRITE &H6F, 0, 3, STRING\$

B I²C通信

有两个I²C通道。它们可以在主控或从属模式下工作。

I/O 引脚

在使用I²C接口之前，必须使用以下命令为第一个通道（称为I2C）定义I/O引脚：

SETPIN sda, scl, I2C

有效引脚为 SDA: GP0, GP4, GP8, GP12, GP16, GP20或GP28
 SCL: GP1, GP5, GP9, GP13, GP17或GP21

以及以下命令用于第二个通道（称为I2C2）：

设置引脚 sda, scl, I2C2

有效引脚为 SDA: GP2, GP6, GP10, GP14, GP18, GP22 或 GP26
 SCL: GP3, GP7, GP11, GP15, GP19 或 GP27

当I²C总线运行在100kHz以上时，设备之间的布线变得重要。理想情况下，电缆应尽可能短（以减少电容），数据线和时钟线不应并排运行，而应在它们之间有一根接地线（以减少串扰）。

如果数据线在时钟高电平时不稳定，或者时钟线抖动，I²C外设可能会变得“混乱”，最终导致总线锁定（通常是通过将时钟线保持在低电平）。如果您不需要更高的速度，那么以100kHz运行是最安全的选择。

I²C 主控命令

在主控模式下，有四个命令可以用于第一个通道（I2C），如下所示。

第二个通道（I2C2）的命令是相同的，只是命令为I2C2。

I2C 打开速度,
超时

启用 I²C 模块在主控模式下。I2C 命令指的是通道 1
而命令 I2C2 指的是通道 2，使用相同的语法。
‘速度’是要使用的时钟速度（以千赫为单位），必须是 100 或 400。
‘超时’是一个以毫秒为单位的值，如果主控发送和接收命令未完成，则将在此值之后中断。最小值是100。值为零将禁用超时（尽管这不推荐）。

I2C 写入 addr,
选项, 发送长度,
发送数据 [,sendata ..]

将数据发送到从设备²C。I2C 命令指的是通道 1，而命令 I2C2 指的是通道 2
，使用相同的语法。
‘addr’是从设备的I²C地址。
‘选项’可以是 0 表示正常操作，或 1 表示在命令后保持对总线的控制（
在命令完成时不会发送停止条件）‘发送长度’是要发送的字节数。

‘发送数据’是要发送的数据 - 这可以通过多种方式指定（所有发送的数据将作为值在 0 到 255 之间的字节发送）：

- 数据可以作为单个字节在命令行上提供。
示例：I2C WRITE&H6F, 0, 3, &H23, &H43, &H25
- 数据可以是一个一维数组，使用空括号指定（即没有尺寸）。将发送‘sendlen’字节的数组，从第一个元素开始。示例：I2C WRITE&H6F, 0, 3, ARRAY()
- 数据可以是一个字符串变量（而不是常量）。
示例：I2C WRITE&H6F, 0, 3, STRING\$

I2C READ addr, option, rcvlen, rcvbuf	<p>Get data from the I²C slave device. The I2C command refers to channel 1 while the command I2C2 refers to channel 2 using the same syntax.</p> <p>‘addr’ is the slave’s I²C address.</p> <p>‘option’ can be 0 for normal operation or 1 to keep control of the bus after the command (a stop condition will not be sent at the completion of the command)</p> <p>‘rcvlen’ is the number of bytes to receive.</p> <p>‘rcvbuf’ is the variable or array used to save the received data - this can be:</p> <ul style="list-style-type: none"> • A string variable. Bytes will be stored as sequential characters in the string. • A one dimensional array of numbers specified with empty brackets. Received bytes will be stored in sequential elements of the array starting with the first. Example: I2C READ &H6F, 0, 3, ARRAY() • A normal numeric variable (in this case rcvlen must be 1).
I2C CLOSE	Disables the master I ² C module and returns the I/O pins to a "not configured" state. This command will also send a stop if the bus is still held.
I²C Slave Commands	
I2C SLAVE OPEN addr, send_int, recv_int	<p>Enables the I²C module in slave mode. The I2C command refers to channel 1 while the command I2C2 refers to channel 2 using the same syntax.</p> <p>‘addr’ is the slave I²C address.</p> <p>‘send_int’ is the subroutine to be invoked when the module has detected that the master is expecting data.</p> <p>‘recv_int’ is the subroutine to be called when the module has received data from the master. Note that this is triggered on the first byte received so your program might need to wait until all the data is received.</p>
I2C SLAVE WRITE sendlen, senddata [,senddata ..]	<p>Send the data to the I²C master. The I2C command refers to channel 1 while the command I2C2 refers to channel 2 using the same syntax.</p> <p>This command should be used in the send interrupt (ie in the ‘send_int’ subroutine when the master has requested data). Alternatively, a flag can be set in the interrupt subroutine and the command invoked from the main program loop when the flag is set.</p> <p>‘sendlen’ is the number of bytes to send.</p> <p>‘senddata’ is the data to be sent. This can be specified in various ways, see the I2C WRITE commands for details.</p>
I2C SLAVE READ rcvlen, rcvbuf, rcvd	<p>Receive data from the I²C master device. The I2C command refers to channel 1 while the command I2C2 refers to channel 2 using the same syntax.</p> <p>This command should be used in the receive interrupt (ie in the ‘recv_int’ subroutine when the master has sent some data). Alternatively a flag can be set in the receive interrupt subroutine and the command invoked from the main program loop when the flag is set.</p> <p>‘rcvlen’ is the maximum number of bytes to receive.</p> <p>‘rcvbuf’ is the variable to receive the data. This can be specified in various ways, see the I2C READ commands for details.</p> <p>‘rcvd’ is a variable that, at the completion of the command, will contain the actual number of bytes received (which might differ from ‘rcvlen’).</p>
I2C SLAVE CLOSE	Disables the slave I ² C module and returns the external I/O pins to a "not configured" state. They can then be configured using SETPIN.

I2C READ addr,
option, rcvlen, rcvbuf 从I2C从设备获取数据。I2C命令指的是通道1，而命令I2C2指的是通道²，使用相同的语法。

‘addr’是从设备的I²C地址。

‘option’可以为0表示正常操作，或1表示在命令后保持对总线的控制（在命令完成时不会发送停止条件），‘rcvlen’是要接收的字节数。

‘rcvbuf’是用于保存接收数据的变量或数组 - 这可以是：

- 一个字符串变量。字节将作为字符串中的连续字符存储。
- 一个用空括号指定的一维数字数组。接收到的字节将从第一个开始存储在数组的连续元素中。
示例：I2C READ &H6F, 0, 3, ARRAY()
- 一个普通的数字变量（在这种情况下，rcvlen 必须为 1）。

I2C 关闭

禁用主控 I²C 模块，并将 I/O 引脚返回到“未配置”状态。
如果总线仍然被占用，此命令还将发送一个停止信号。

I²C 从属命令

I2C SLAVE OPEN
addr, send_int,
recv_int

在从属模式下启用 I²C 模块。I2C 命令指的是通道 1 而命令 I2C2 指的是通道 2，使用相同的语法。

‘addr’ 是从属 I²C 地址。

‘send_int’ 是在模块检测到主控期望数据时要调用的子程序。

‘recv_int’ 是在模块从主控接收到数据时要调用的子程序。请注意，这在接收到第一个字节时触发，因此您的程序可能需要等待直到所有数据接收完毕。

I2C 从属 写入
sendlen, senddata
[,senddata ..]

将数据发送到 I²C 主控。I2C 命令指的是通道 1，而命令 I2C2 指的是通道 2，使用相同的语法。

此命令应在发送中断中使用（即在主控请求数据时的 ‘send_int’ 子程序中）。或者，可以在中断子程序中设置一个标志，并在标志被设置时从主程序循环中调用该命令。

‘sendlen’ 是要发送的字节数。

‘senddata’ 是要发送的数据。这可以通过多种方式指定，详细信息请参见 I2C 写入命令。

I2C 从属 读取
recvlen, rcvbuf, rcvd

从 I²C 主控设备接收数据。I2C 命令指的是通道 1 而命令 I2C2 指的是通道 2，使用相同的语法。

此命令应在接收中断中使用（即在主控发送数据时的 ‘recv_int’ 子程序中）。或者，可以在接收中断子程序中设置一个标志，并在标志被设置时从主程序循环中调用该命令。

‘recvlen’ 是要接收的最大字节数。

‘rcvbuf’ 是用于接收数据的变量。这可以通过多种方式指定，有关详细信息，请参见 I2C 读取命令。

‘rcvd’ 是一个变量，在命令完成时将包含实际接收到的字节数（这可能与 ‘recvlen’ 不同）。

I2C 从属关闭 禁用从属 I²C 模块，并将外部 I/O 引脚恢复到“未配置”状态。然后可以使
用 SETPIN 进行配置。

Errors

Following an I²C write or read the automatic variable MM.I2C will be set to indicate the result as follows:

- 0 = The command completed without error.
- 1 = Received a NACK response
- 2 = Command timed out

7-Bit Addressing

The standard addresses used in these commands are 7-bit addresses (without the read/write bit). MMBasic will add the read/write bit and manipulate it accordingly during transfers.

Some vendors provide 8-bit addresses which include the read/write bit. You can determine if this is the case because they will provide one address for writing to the slave device and another for reading from the slave. In these situations you should only use the top seven bits of the address. For example: If the read address is 9B (hex) and the write address is 9A (hex) then using only the top seven bits will give you an address of 4D (hex).

Another indicator that a vendor is using 8-bit addresses instead of 7-bit addresses is to check the address range. All 7-bit addresses should be in the range of 08 to 77 (hex). If your slave address is greater than this range then probably your vendor has provided an 8-bit address.

Examples

As an example of a simple communications where the PicoMite is the master, the following program will read and display the current time (hours and minutes) maintained by a PCF8563 real time clock chip connected to the second I²C channel:

```
DIM AS INTEGER RData(2)                                ' this will hold received data
SETPIN GP6, GP5, I2C2                                  ' assign the I/O pins for I2C2
I2C2 OPEN 100, 1000                                    ' open the I2C channel
I2C2 WRITE &H51, 0, 1, 3                               ' set the first register to 3
I2C2 READ &H51, 0, 2, RData()                          ' read two registers
I2C2 CLOSE                                              ' close the I2C channel
PRINT "Time is " RData(1) ":" RData(0)
```

This is an example of communications between two PicoMites where one is the master and the other is the slave.

First the master:

```
SETPIN GP2, GP3, I2C2
I2C2 OPEN 100, 1000
i = 10
DO
    i = i + 1
    a$ = STR$(i)
    I2C2 WRITE &H50, 0, LEN(a$), a$
    PAUSE 200
    I2C2 READ &H50, 0, 8, a$
    PRINT a$
    PAUSE 200
LOOP
```

Then the slave:

```
SETPIN GP2, GP3, I2C2
I2C2 SLAVE OPEN &H50, tint, rint
DO : LOOP

SUB rint
    LOCAL count, a$
    I2C2 SLAVE READ 10, a$, count
    PRINT LEFT$(a$, count)
END SUB

SUB tint
    LOCAL a$ = Time$
    I2C2 SLAVE WRITE LEN(a$), a$
END SUB
```

错误

在 I²C 写入或读取后，自动变量 MM.I2C 将被设置以指示结果，如下所示：

- 0 = 命令成功完成，没有错误。
- 1 = 收到NACK响应
- 2 = 命令超时

7位寻址

这些命令中使用的标准地址是7位地址（不包括读/写位）。MMBasic将在传输过程中添加读/写位并相应地进行操作。

一些供应商提供的8位地址包括读/写位。您可以通过检查它们提供的写入从设备的地址和读取从设备的地址来确定是否如此。在这些情况下，您应该仅使用地址的最高七位。例如：如果读取地址是9B（十六进制），写入地址是9A（十六进制），那么仅使用最高七位将给您一个地址4D（十六进制）。

另一个指示供应商使用8位地址而不是7位地址的方法是检查地址范围。

所有7位地址应在08到77（十六进制）的范围内。如果您的从属地址超出此范围，则您的供应商可能提供了一个8位地址。

示例

作为一个简单通信的例子，其中PicoMite是主控，以下程序将读取并显示由连接到第二个I²C通道的PCF8563实时时钟芯片维护的当前时间（小时和分钟）：

```
DIM AS INTEGER RData(2)                                ' 这将保存接收到的数据
SETPIN GP6, GP5, I2C2                                  ' 为I2C2分配I/O引脚
I2C2 OPEN 100, 1000                                    ' 打开I2C通道
I2C2 WRITE &H51, 0, 1, 3                                ' 将第一个寄存器设置为3
I2C2 READ &H51, 0, 2, RData()                          ' 读取两个寄存器
I2C2 CLOSE                                              ' 关闭I2C通道
PRINT "时间是 " RData(1) ":" RData(0)
```

这是两个PicoMite之间通信的例子，其中一个为主控，另一个是从属。

首先是主控：

```
SETPIN GP2, GP3, I2C2
I2C2 打开 100, 1000
i = 10
DO
    i = i + 1
    a$ = STR$(i)
    I2C2 写入 &H50, 0, LEN(a$), a$
    暂停 200
    I2C2 读取 &H50, 0, 8, a$
    打印 a$
    暂停 200
    循环
```

然后是从属：

```
SETPIN GP2, GP3, I2C2
I2C2 从属 打开 &H50, tint, rint
DO : 循环

子程序 rint
    局部 计数, a$
    I2C2 从属 读取 10, a$, 计数
    打印 LEFT$(a$, 计数)
    结束子程序

子程序 tint
    局部 a$ = Time$
    I2C2 从属 写入 LEN(a$), a$
    结束子程序
```

Appendix C

1-Wire Communications

The 1-Wire protocol was developed by Dallas Semiconductor to communicate with chips using a single signalling line. This implementation was written for MMBasic by Gerard Sexton.

There are three commands that you can use:

ONEWIRE RESET pin	Reset the 1-Wire bus
ONEWIRE WRITE pin, flag, length, data [, data...]	Send a number of bytes
ONEWIRE READ pin, flag, length, data [, data...]	Get a number of bytes

Where:

pin - The PicoMite I/O pin to use. It can be any pin capable of digital I/O.

flag - A combination of the following options:

- 1 - Send reset before command
- 2 - Send reset after command
- 4 - Only send/recv a bit instead of a byte of data
- 8 - Invoke a strong pullup after the command (the pin will be set high and open drain disabled)

length - Length of data to send or receive

data - Data to send or variable to receive.

The number of data items must agree with the length parameter.

The automatic variable MM.ONEWIRE returns true if a device was found

After the command is executed, the I/O pin will be set to the not configured state unless flag option 8 is used.

When a reset is requested the automatic variable MM.ONEWIRE will return true if a device was found. This will occur with the ONEWIRE RESET command and the ONEWIRE READ and ONEWIRE WRITE commands if a reset was requested (flag = 1 or 2).

The 1-Wire protocol is often used in communicating with the DS18B20 temperature measuring sensor and to help in that regard MMBasic includes the TEMP() function which provides a convenient method of directly reading the temperature of a DS18B20 without using these functions.

C 通信

1-Wire 通信

1-Wire协议是由达拉斯半导体公司开发的，用于通过单一信号线与芯片进行通信。此实现是由Gerard Sexton为MMBasic编写的。

您可以使用三个命令：

ONEWIRE RESET 引脚	重置1-Wire总线
ONEWIRE WRITE 引脚, 标志, 长度, 数据 [, 数据...]	发送多个字节
ONEWIRE READ 引脚, 标志, 长度, 数据 [, 数据...]	获取多个字节

其中：

引脚 - 要使用的PicoMite输入/输出引脚。它可以是任何能够进行数字输入/输出的引脚。

标志 - 以下选项的组合：

- 1 - 在命令之前发送重置
- 2 - 在命令之后发送重置
- 4 - 仅发送/接收一个位而不是一个字节的数据
- 8 - 在命令后启用强上拉（引脚将被设置为高电平，开漏禁用）

长度 - 要发送或接收的数据长度

data - 要发送的数据或要接收的变量。
数据项的数量必须与长度参数一致。

自动变量MM.ONEWIRE在找到设备时返回真。

命令执行后，输入/输出引脚将被设置为未配置状态，除非使用标志选项8。

当请求重置时，如果找到设备，自动变量MM.ONEWIRE将返回真。如果请求了重置（标志 = 1 或 2），则会在ONEWIRE RESET命令和ONEWIRE READ及ONEWIRE WRITE命令中发生此情况。

1-Wire协议通常用于与DS18B20温度测量传感器进行通信，为此，MMBasic包含TEMPR()函数，该函数提供了一种方便的方法，可以直接读取DS18B20的温度，而无需使用这些函数。

Appendix D

SPI Communications

The Serial Peripheral Interface (SPI) communications protocol is used to send and receive data between integrated circuits. The PicoMite acts as the master (i.e. it generates the clock).

I/O Pins

Before an SPI interface can be used the I/O pins for the channel must be allocated using the following commands. For the first channel (referred as SPI) it is:

```
SETPIN rx, tx, clk, SPI
Valid pins are      RX:    GP0, GP4, GP16 or GP20
                    TX:    GP3, GP7 or GP19
                    CLK:   GP2, GP6 or GP18
```

And the following command for the second channel (referred to as SPI2) is:

```
SETPIN rx, tx, clk, SPI2
Valid pins are      RX:    GP8, GP12 or GP28
                    TX:    GP11, GP15 or GP27
                    CLK:   GP10, GP14 or GP26
```

TX is data from the PicoMite and RX is data to it.

SPI Open

To use the SPI function the SPI channel must be first opened.

The syntax for opening the first SPI channel is (use SPI2 for the second channel):

```
SPI OPEN speed, mode, bits
```

Where:

- 'speed' is the speed of the clock. It is a number representing the clock speed in Hz.
- 'mode' is a single numeric digit representing the transmission mode – see Transmission Format below.
- 'bits' is the number of bits to send/receive. This can be any number in the range of 4 to 16 bits.
- It is the responsibility of the program to separately manipulate the CS (chip select) pin if required.

Transmission Format

The most significant bit is sent and received first. The format of the transmission can be specified by the 'mode' as shown below. Mode 0 is the most common format.

Mode	Description	CPOL	CPHA
0	Clock is active high, data is captured on the rising edge and output on the falling edge	0	0
1	Clock is active high, data is captured on the falling edge and output on the rising edge	0	1
2	Clock is active low, data is captured on the falling edge and output on the rising edge	1	0
3	Clock is active low, data is captured on the rising edge and output on the falling edge	1	1

For a more complete explanation see: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Standard Send/Receive

When the first SPI channel is open data can be sent and received using the SPI function (use SPI2 for the second channel). The syntax is:

```
received_data = SPI(data_to_send)
```

Note that a single SPI transaction will send data while simultaneously receiving data from the slave.

'data_to_send' is the data to send and the function will return the data received during the transaction.

'data_to_send' can be an integer or a floating point variable or a constant.

If you do not want to send any data (i.e. you wish to receive only) any number (e.g. zero) can be used for the data to send. Similarly if you do not want to use the data received it can be assigned to a variable and ignored.

D SPI通信

串行外设接口（SPI）通信协议用于在集成电路之间发送和接收数据。PicoMite作为主控（即生成时钟）。

I/O 引脚

在使用SPI接口之前，必须使用以下命令分配通道的I/O引脚。对于第一个通道（称为SPI），它是：

SETPIN rx, tx, clk, SPI
有效引脚为 RX: GP0, GP4, GP16或GP20
 TX: GP3, GP7或GP19
 CLK: GP2, GP6或GP18

第二个通道（称为SPI2）的命令为：

SETPIN rx, tx, clk, SPI2
有效引脚为 RX: GP8, GP12或GP28
 TX: GP11, GP15或GP27
 CLK: GP10, GP14或GP26

TX是来自PicoMite的数据，RX是发送给它的数据。

SPI 打开

要使用SPI功能，必须首先打开SPI通道。

打开第一个SPI通道的语法是（使用SPI2作为第二个通道）：

SPI 打开 速度，模式，位数

其中：

- ‘速度’是时钟的速度。它是一个表示时钟速度的数字，单位为赫兹。
- ‘模式’是一个单一的数字，表示传输模式 – 请参见下面的传输格式。
- ‘位数’是要发送/接收的位数。这可以是4到16位范围内的任何数字。
- 如果需要，程序有责任单独操作CS（芯片选择）引脚。

传输格式

最重要的位首先被发送和接收。传输的格式可以通过‘模式’来指定，如下所示。模式0是最常见的格式。

模式	描述	CPOL	CPHA
0	时钟为高电平有效，数据在上升沿被捕获，在下降沿输出	0	0
1	时钟为高电平有效，数据在下降沿被捕获，在上升沿输出	0	1
2	时钟为低电平有效，数据在下降沿被捕获，在上升沿输出	1	0
3	时钟为低电平有效，数据在上升沿捕获，在下降沿输出	1	1

有关更完整的解释，请参见：http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

标准发送/接收

当第一个SPI通道打开时，可以使用SPI函数发送和接收数据（使用SPI2进行第二个通道）。语法为：

```
received_data = SPI(data_to_send)
```

请注意，单个SPI事务将在发送数据的同时从从属设备接收数据。

‘data_to_send’是要发送的数据，该函数将返回在事务期间接收到的数据。

‘data_to_send’可以是整数、浮点变量或常量。

如果您不想发送任何数据（即您只希望接收），可以使用任何数字（例如零）作为要发送的数据。同样，如果您不想使用接收到的数据，可以将其分配给一个变量并忽略。

Bulk Send/Receive

Data can also be sent in bulk (use SPI2 for the second channel):

```
SPI WRITE nbr, data1, data2, data3, ... etc
```

or

```
SPI WRITE nbr, string$
```

or

```
SPI WRITE nbr, array()
```

In the first method 'nbr' is the number of data items to send and the data is the expressions in the argument list (i.e. 'data1', 'data2' etc). The data can be an integer or a floating point variable or a constant.

In the second or third method listed above the data to be sent is contained in the 'string\$' or the contents of 'array()' (which must be a single dimension array of integer or floating point numbers). The string length, or the size of the array must be the same or greater than nbr. Any data returned from the slave is discarded.

Data can also be received in bulk (use SPI2 for the second channel):

```
SPI READ nbr, array()
```

Where 'nbr' is the number of data items to be received and array() is a single dimension integer array where the received data items will be saved. This command sends zeros while reading the data from the slave.

SPI Close

If required the first SPI channel can be closed as follows (the I/O pins will be set to inactive):

```
SPI CLOSE
```

Use SPI2 for the second channel.

Examples

The following example shows how to use the SPI port for general I/O. It will send a command 80 (hex) and receive two bytes from the slave SPI device using the standard send/receive function:

```
PIN(10) = 1 : SETPIN 10, DOUT      ' pin 10 will be used as the enable signal
SETPIN GP20, GP3, GP2, SPI          ' assign the I/O pins
SPI OPEN 5000000, 3, 8              ' speed is 5MHz and the data size is 8 bits
PIN(10) = 0                         ' assert the enable line (active low)
junk = SPI(&H80)                   ' send the command and ignore the return
byte1 = SPI(0)                     ' get the first byte from the slave
byte2 = SPI(0)                     ' get the second byte from the slave
PIN(10) = 1                         ' deselect the slave
SPI CLOSE                           ' and close the channel
```

The following is similar to the example given above but this time the transfer is made using the bulk send/receive commands:

```
OPTION BASE 1
DIM data%(2)
SETPIN GP20, GP3, GP2, SPI
PIN(10) = 1 : SETPIN 10, DOUT      ' our array will start with the index 1
SPI OPEN 5000000, 3, 8              ' define the array for receiving the data
PIN(10) = 0                         ' assign the I/O pins
SPI WRITE 1, &H80                  ' pin 10 will be used as the enable signal
SPI READ 2, data%()                ' speed is 5MHz, 8 bits data
PIN(10) = 1                         ' assert the enable line (active low)
SPI CLOSE                           ' send the command
                                    ' get two bytes from the slave
                                    ' deselect the slave
                                    ' and close the channel
```

批量发送/接收

数据也可以批量发送（使用SPI2作为第二通道）：

SPI 写入 编号, 数据1, 数据2, 数据3, ... 等等

或

SPI 写入 编号, 字符串\$

或

SPI 写入 编号, 数组()

在第一种方法中，'编号'是要发送的数据项数量，数据是参数列表中的表达式
(即'数据1', '数据2'等等。数据可以是整数或浮点变量或常量。在上述第二或第三种

方法中，要发送的数据包含在'字符串或'

array()（必须是一个单维的整数或浮点数数组）。字符串长度或数组的大小必须与nbr相同或大于。
任何从从属设备返回的数据都会被丢弃。

数据也可以批量接收（使用SPI2作为第二通道）：

SPI 读取 nbr, array()

其中'nbr'是要接收的数据项数量，array()是一个单维整数数组，用于保存接收到的数据项。此命令在从从属设备读取数据时发送零。

关闭SPI

如果需要，可以按如下方式关闭第一个SPI通道（I/O引脚将被设置为非活动状态）：

关闭SPI

使用SPI2作为第二个通道。

示例

以下示例演示如何使用SPI端口进行一般I/O。它将发送命令80（十六进制）并使用标准的发送/接收函数从从属SPI设备接收两个字节：

```
PIN(10) = 1 : SETPIN 10, DOUT      | 引脚10将用作使能信号
SETPIN GP20, GP 3, GP 2, SPI        | 分配I/O引脚
SPI 打开 5000000, 3, 8             | 速度为5MHz，数据大小为8位
PIN(10) = 0                         | 使能引脚（低电平有效）
junk = SPI(&H80)                   | 发送命令并忽略返回值
byte1 = SPI(0)                     | 从从属设备获取第一个字节
byte2 = SPI(0)                     | 从从属设备获取第二个字节
PIN(10) = 1                         | 取消选择从属设备
关闭SPI                            | 并关闭通道
```

以下内容与上面的示例类似，但这次使用批量发送/接收命令进行传输：

```
OPTION BASE 1
DIM data%(2)
SETPIN GP20, GP3, GP2, SPI
PIN(10) = 1 : SETPIN 10, DOUT
SPI 打开 5000000, 3, 8
PIN(10) = 0
SPI WRITE 1, &H80
SPI 读取 2, 数据%()
PIN(10) = 1
关闭SPI
```

| 我们的数组将从索引1开始
| 定义用于接收数据的数组
| 分配I/O引脚
| 引脚10将用作使能信号
| 速度为5MHz，8位数据
| 使能引脚（低电平有效）
| 发送命令
| 从从属设备获取两个字节
| 取消选择从属设备
| 并关闭通道

Appendix E

Regex Syntax

The alternate forms of the INSTR() and LINSTR() functions can take a regular expression as the search pattern.
The alternate form of the commands are:

```
INSTR([start],text$, search$ [,size])
LINSTR(text%,search$ [,start] [,size])
```

In both cases specifying the size parameter causes the firmware to interpret the search string as a regular expression. The size parameter is a floating point variable that is used by the firmware to return the size of a matching string. If the variable doesn't exist it is created. As implemented in MMBasic you need to apply the returned start and size values to the MID\$ function to extract the matched string. e.g.

```
IF start THEN match$=MID$(text$,start,size) ELSE match$="" ENDIF
```

The library used for the regular expressions “implements POSIX draft P1003.2/D11.2, except for some of the internationalization features”. See <http://mirror.math.princeton.edu/pub/oldlinux/Linux.old/Ref-docs/POSIX/all.pdf> section 2.8 for details of constructing Regular Expressions or other online tutorials if you are not familiar with them.

The syntax of regular expressions can vary slightly with the various implementations. This document is a summary of the syntax and supported operations used in the MMBasic implementation.

Anchors

- ^ Start of string
- \$ End of string
- \b Word Boundary
- \B Not a word boundary
- \< Start of word
- \> End of word

Qualifiers

- * 0 or more (not escaped)
- \+ 1 or more
- \? 0 or 1
- \{3\} Exactly 3
- \{3,\} 3 or more
- \{3,5\} 3,4 or 5

Groups and Ranges

- (a\b) a or b
- \(...\)\) group
- [abc] Range (a or b or c)
- [^abc] Not (a or b or c)
- [a-q] lower case letters a to q
- [A-Q] upper case letters A to Q
- [0-7] Digits from 0 to 7

Escapes Required to Match Normal Characters

- \^ to match ^ (caret)
- \. to match . (dot)
- * to match * (asterix)
- \\$ to match \$ (dollar)
- \[to match [(left bracket)
- \\\ to match \ (backslash)

E 正则表达式语法

INSTR() 和 LINSTR() 函数的替代形式可以将正则表达式作为搜索模式。
命令的替代形式为：

```
INSTR([start],text$, search$ [,size])  
LINSTR(text%,search$ [,start] [,size])
```

在这两种情况下，指定大小参数会导致固件将搜索字符串解释为正则表达式。大小参数是一个浮点变量，固件使用它来返回匹配字符串的大小。如果变量不存在，则会创建它。在 MMBasic 中实现时，您需要将返回的起始和大小值应用于 MID\$ 函数以提取匹配的字符串。例如：IF start THE N match\$=MID\$(text\$,start,size) ELSE match\$="" ENDIF

用于正则表达式的库“实现了 POSIX 草案 P1003.2/D11.2，除了某些国际化特性”。请参见<http://mirror.math.princeton.edu/pub/oldlinux/Linux.old/Ref-docs/POSIX/all.pdf> 第2.8节，了解构建正则表达式的详细信息，或者如果您不熟悉它们，可以查看其他在线教程。

正则表达式的语法在不同的实现中可能会略有不同。本文档是MMBasic实现中使用的语法和支持的操作的摘要。

锚点

- ^ 字符串开始
- \$ 字符串结束
- \b 单词边界
- \B 不是单词边界
- \< 单词开始
- \> 单词结束

限定符

- * 0个或多个（未转义）
- \+ 1个或多个
- \? 0个或1个
- \{3\} 恰好3个
- \{3,\} 3个或更多
- \{3,5\} 3、4或5

组和范围

- (a\b) a或b
- \(...\)\组
- [abc] 范围 (a或b或c)
- [^abc] 不(a 或 b 或 c)
- [a-q] 小写字母 a 到 q
- [A-Q] 大写字母 A 到 Q
- [0-7] 数字从 0 到 7

匹配普通字符所需的转义

- \^ 匹配 ^ (插入符号)
- \. 匹配 . (点)
- * 匹配 * (星号)
- \\$ 匹配 \$ (美元)
- \[匹配 [(左括号)
- \\\ 匹配 \ (反斜杠)

Escapes with Special Functions

\+ See Quantifiers
\? See Quantifiers
\{ See Quantifiers
\} See Quantifiers
\| See Groups and Ranges
\(See Groups and Ranges
\) See Groups and Ranges
\w See Character Classes

Character Classes

\w digits,letters and _
[:word:] digits,letters and _
[:upper:] Upper case letters _
[:lower:] Lower case letters _
[:alpha:] All letters
[:alnum:] Digits and letters
[:digit:] Digits
[:xdigit:] Hexidecimal digits
[:punct:] Punctuation
[:blank:] Space and tab
[:space:] Blank charaters
[:cntrl:] Control charaters
[:graph:] Printed characters
[:print:] Printed chars and spaces

Example expression to match an IP Address which is contained within a word boundary.

"\<[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\>"

带特殊函数的转义

\+ 见量词
\? 见量词
\{ 见量词
\} 见量词
\| 见分组和范围
\(见分组和范围
\) 见分组和范围
\查看字符类

字符类

\w 数字、字母和_
[:word:] 数字、字母和_
[:upper:] 大写字母_
[:lower:] 小写字母_
[:alpha:] 所有字母
[:alnum:] 数字和字母
[:digit:] 数字
[:xdigit:] 十六进制数字
[:punct:] 标点符号
[:blank:] 空格和制表符
[:space:] 空白字符
[:cntrl:] 控制字符
[:graph:] 打印字符
[:print:] 打印字符和空格

匹配IP地址的示例表达式，该地址位于单词边界内。

"\<[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\>"

Appendix F

The PIO Programming Package

Introduction to the PIO

The RP2040 has many built in peripherals like PWM, UART, ADC, SPI. In addition the RP2040 chip contains two PIO blocks, rather like cut-down, highly specialised CPU cores. MMBasic refers to them as PIO0 and PIO1 in line with the Raspberry Pi documentation. They are capable of running completely independently of the main system and of each other. They can be used to create such things as very high accuracy serial data interfaces and bit streams, although they are by no means restricted to this sort of thing. They can be made to run extremely fast, with a throughput of up to 32 bits during every clock cycle.

Before a state machine can execute it's program, the program needs to be written to PIO memory, and the state machine needs to be configured.

This appendix describes the support MMBasic can give in using PIO. It does not contain an explanation how to write PIO state machine programs. For better understanding how the PIO state machines work look at following thread "PIO explained PICOMITE" on the thebackshed.com forum:

<https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=15385>

Overview of PIO

A single PIO block has four independent state machines. All four state machines share a single 32 instruction program area of flash memory. This memory has write-only access from the main system, but has four read ports, one for each state machine, so that each can access it independently at its own speed. Each state machine has its own program counter.

Each state machine also has two 32-bit "scratchpad" registers, X and Y, which can be used as temporary data stores.

I/O pins are accessed via an input/output mapping module that can access 32 pins (but limited to 30 for the RP2040). All state machines can access all the pins independently and simultaneously.

The system can write data into the input end of a 4-word 32-bit wide TX FIFO buffer. The state machine can then use pull to move the output word of the FIFO into the OSR (Output Shift Register). It can also use out to shift 1-32 bits at a time from the OSR into the output mapping module or other destinations. AUTOPULL can be used to automatically pull data until the TX FIFO is empty or reaches a preset level.

The system can read data from the output end of a 4-word 32-bit wide RX FIFO buffer. The state machine can then use in to shift 1-32 bits of data at a time from the input mapping module into the ISR (Input Shift Register). It can also use push to move the contents of the ISR into the FIFO. AUTOPUSH can be used to automatically push data until the RX FIFO is full or reaches a preset level.

The FIFO buffers can be reconfigured to form a single direction 8-word 32-bit FIFO in a single direction. The buffers allow data to be passed to and from the state machines without either the system or the state machine having to wait for the other.

Each of the four state machines in the PIO has four registers associated with it:

- CLKDIV is the clock divider, which has a 16-bit integer divider and an 8-bit fractional divider. This sets how fast the state machine runs. It divides down from the main system clock.
- EXECCTRL holds information controlling the translation and execution of the program memory
- SHIFTCTRL controls the arrangement and usage of the shift registers
- PINCTRL controls which and how the GPIO pins are used.

The four state machines of a PIO have shared access to its block of 8 interrupt flags. Any state machine can use any flag. They can set, reset or wait for them to change. In this way they can be made to run synchronously if required. The lower four flags are also accessible to and from the main system, so the PIO can be controlled or pass interrupts back.

DMA can be used to pass information to and from the PIO block via its FIFO from the RP2040's memory

A PIO has nine possible programming instructions, but there can be many variations on each one. For example, Mov can have up to 8 sources, 8 destinations, 3 process operations during the copy, with optional delay and/or side set operations!

- Jmp Jump to an absolute address in program memory if a condition is true (or instantly).
- Wait Stall operation of the state machine until a condition is true.
- In Shift a number of bits from a source into the ISR.

F PIO编程包

PIO简介

RP2040内置了许多外设，如PWM、UART、ADC、SPI。此外，RP2040芯片包含两个PIO模块，类似于简化版的高度专业化的CPU核心。MMBasic将它们称为PIO0和PIO1，符合树莓派的文档。它们能够完全独立于主系统和彼此运行。它们可以用于创建非常高精度的串行数据接口和位流，尽管它们并不局限于此类应用。它们可以以极快的速度运行，在每个时钟周期内的吞吐量可达32位。

在状态机执行其程序之前，需要将程序写入PIO内存，并配置状态机。

本附录描述了MMBasic在使用PIO时提供的支持。它不包含如何编写PIO状态机程序的说明。为了更好地理解PIO状态机的工作原理，请查看thebackshed.com论坛上的主题"PIO explained PICOMIT E"：

<https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=15385>

PIO概述

单个PIO块有四个独立的状态机。所有四个状态机共享一个32条指令的闪存程序区域。该内存只能由主系统写入，但有四个读取端口，每个状态机一个，因此每个状态机可以以自己的速度独立访问。每个状态机都有自己的程序计数器。

每个状态机还有两个32位的"临时"寄存器，X和Y，可以用作临时数据存储。

I/O引脚通过一个输入/输出映射模块访问，该模块可以访问32个引脚（但RP2040限制为30个）。所有状态机可以独立且同时访问所有引脚。

系统可以将数据写入一个4字32位宽的TX FIFO缓冲区的输入端。状态机可以然后使用拉取将FIFO的输出字移动到OSR（输出移位寄存器）。它还可以使用输出将1-32位的数据从OSR移位到输出映射模块或其他目的地。AUTOPULL可以用于自动拉取数据，直到TX FIFO为空或达到预设水平。

系统可以从4字32位宽RX FIFO缓冲区的输出端读取数据。状态机可以使用输入将1-32位的数据一次性从输入映射模块移位到ISR（输入移位寄存器）。它还可以使用推送将ISR的内容移动到FIFO。AUTOPUSH可以用于自动推送数据，直到RX FIFO满或达到预设水平。

FIFO缓冲区可以重新配置为形成一个单向的8字32位FIFO。缓冲区允许数据在状态机与系统之间传递，而无需等待对方。

PIO中的四个状态机每个都有四个相关的寄存器：

- CLKDIV是时钟分频器，具有16位整数分频器和8位分数分频器。这设置状态机运行的速度。它从主系统时钟进行分频。
- EXECCTRL保存控制程序内存翻译和执行的信息。
- SHIFTCTRL控制移位寄存器的排列和使用。
- PINCTRL控制GPIO引脚的使用方式和选择。

PIO的四个状态机共享访问其8个中断标志的块。任何状态机都可以使用任何标志。它们可以设置、重置或等待标志变化。通过这种方式，如果需要，它们可以被设置为同步运行。下四个标志也可以被主系统访问，因此PIO可以被控制或传递中断。

DMA可用于通过其FIFO在RP2040的内存与PIO块之间传递信息

一个PIO有九个可能的编程指令，但每个指令可以有许多变体。例如，Mov可以有多达8个源，8个目标，在复制过程中有3个处理操作，带有可选的延迟和/或侧设置操作！

- Jmp 如果条件为真（或立即），则跳转到程序内存中的绝对地址。
- Wait 在状态机的操作中，直到条件为真时暂停。
- 输入 将若干位从源移入ISR。

- Out Shift a number of bits out of the OSR to a destination.
 - Push Push the contents of the ISR into the RX FIFO as a single 32-bit word.
 - Pull Load a 32-bit word from the TX FIFO into the OSR.
 - Mov Copy date from a source to a destination.
 - Irq Set or clear an interrupt flag.
 - Set Immediately write data to a destination.

Instructions are all 16-bit and contain both the instruction and all data associated with it. All instructions operate in 1 clock cycle, but it is possible to introduce a delay of several idle clock cycles between an instruction and the next.

Additionally, there is a facility called "side-set" which allows a value to be written to some pre-defined output pins while an instruction is being read from memory. This is transparent to the program.

Programming PIO

PicoMite programs the PIO statemachine memory using one of the following commands. Each option will be explained with an example of the exact same program that toggles one of the GPIO lines of the PicoMite. Which GPIO line is toggled, is determined in the configuration.

PIO ASSEMBLE

This command is used to use the build in assembler to generate the program from mnemonics, then write it directly into PIO memory.

```
PIO ASSEMBLE 1,".program test"          'a program has to have a name
PIO ASSEMBLE 1,".line 0"                'start the program at line 0
PIO ASSEMBLE 1,"SET PINDIRS 1"         'SET the GPIO line to output
PIO ASSEMBLE 1,"label:"                 'define a label called "label"
PIO ASSEMBLE 1,"SET PIN 1"              'SET the GPIO pin high
PIO ASSEMBLE 1,"SET PIN 0"              'SET the GPIO pin low
PIO ASSEMBLE 1,"JMP label"             'JuMP to "label" in an endless loop
PIO ASSEMBLE 1,".end program list"     'end program, list=show result
```

PIO PROGRAM LINE

This command can be used to program 16bit values to individual lines in the PIO memory.

```
pio program line 1,0,&hE081      'SET pin output
pio program line 1,1,&hE001      'SET pin high
pio program line 1,2,&hE000      'SET pin low
pio program line 1,3,&h0001     'JMP to line 1
```

PIO PROGRAM

This command writes all 32 lines in one PIO from an array. This is useful once a PIO program is debugged. It is extremely compact.

```
Dim a%(7)=(&h0001E0000E001E081,0,0,0,0,0,0,0)  
PIO program 1,a%()
```

Configuring PIO

The PicoMite can configure each state machine individually. Configuration allows 2 state machines to run the exact same program lines (e.g. an SPI interface) but operate with different GPIO pins and at different speeds. There are several configuration fields.

FREQUENCY

PicoMite contains a default configuration for each configuration field, except for the frequency. The frequency is set by a 16 bit divider from the ARM clock. Example: when OPTION CPUSPEED 126000 is set the PIO can run at speeds between 126MHz and 1.922kHz ($126000000 / 65536$). Be aware that higher CPU speeds (overclocking) directly impact the state machine frequency.

PIN CONTROL

PicoMite defaults the GPIO pins for use by MMBasic. For the PIO to take ownership of a GPIO pin MMBasic needs to assign it to PIO as below.

SETPIN GPxx,PIOx (e.g. SETPIN gp0,pio1)

- Out 将若干位从 OSR 移出到目标。
- Push 将 ISR 的内容作为单个 32 位字推入 RX FIFO。
- Pull 从 TX FIFO 加载一个 32 位字到 OSR。
- Mov 将数据从源复制到目标。
- Irq 设置或清除中断标志。
- 设置立即将数据写入目标。

指令都是16位的，包含指令及其所有相关数据。所有指令在1个时钟周期内操作，但可以在指令与下一条指令之间引入几个空闲时钟周期的延迟。

此外，还有一个称为"side-set"的功能，它允许在从内存中读取指令时，将值写入一些预定义的输出引脚。这对程序是透明的。

编程PIO

PicoMite使用以下命令之一编程PIO状态机内存。每个选项将通过一个示例来解释，该示例是切换PicoMite的一个GPIO引脚的完全相同的程序。

切换哪个GPIO引脚由配置决定。

PIO汇编

此命令用于使用内置汇编器从助记符生成程序，然后直接写入PIO内存。

PIO ASSEMBLE 1, ".program test"	'程序必须有一个名称
PIO ASSEMBLE 1, ".行 0"	'从行 0 开始程序
PIO ASSEMBLE 1, "SET PINDIRS 1"	'将 GPIO 引脚设置为输出
PIO ASSEMBLE 1, "label:"	'定义一个名为 "label" 的标签
PIO ASSEMBLE 1, "SET PIN 1"	'将 GPIO 引脚设置为高电平
PIO ASSEMBLE 1, "SET PIN 0"	'将 GPIO 引脚设置为低电平
PIO ASSEMBLE 1, "JMP label"	'在无限循环中跳转到 "label"
PIO ASSEMBLE 1, ".结束 程序 列表"	'结束程序，列表=显示结果

PIO PROGRAM LINE

此命令可用于将 16 位值编程到 PIO 内存中的各个行。

pio program line 1,0,&hE081	'设置引脚输出
pio 程序 行 1,1,&hE001	'设置引脚高
pio 程序 行 1,2,&hE000	'设置引脚低
pio 程序 行 1,3,&h0001	'跳转到行 1

PIO 程序

此命令从数组中将所有 32 行写入一个 PIO。一旦 PIO 程序调试完成，这非常有用。它极其紧凑。

```
Dim a%(7)=(&h0001E0000E001E081,0,0,0,0,0,0,0)
PIO 程序 1,a%()
```

配置 PIO

PicoMite 可以单独配置每个状态机。配置允许 2 个状态机运行完全相同的程序行（例如 SPI 接口），但使用不同的 GPIO 引脚并以不同的速度操作。
有几个配置字段。

频率

PicoMite 为每个配置字段提供默认配置，频率除外。频率由ARM时钟的16位分频器设置。示例：当设置OPTION CPUSPEED 126000时，PIO可以在126MHz到1.922kHz之间运行（126000000 / 65536）。请注意，更高的CPU速度（超频）会直接影响状态机频率。

引脚控制

PicoMite默认将GPIO引脚用于MMBasic。为了让PIO拥有一个GPIO引脚，MMBasic需要将其分配给PIO，如下所示。

SETPIN GPxx,PIOx (例如：SETPIN gp0,pio1)

A state machine can SET the state of a pin (SET is a state machine instruction), but can also output serial data to one or more GPIO pins using the OUT instruction. Or read serial data using the IN instruction. And GPIO pins can be set as a side effect of any state machine instruction (SIDE SET). For each method of interfacing, different pins can be mapped to the state machine.

It is important to understand is that these instructions work on consecutive pins. This means that there is a range of pins that can be controlled, starting at the lowest GPx pin number (e.g. GP0), and pins next to it can be included (up to 5 pins in total). So GP0,GP1,GP2 is a valid set of IO pins. GP0,GP1,GP6 is not. Consider this when designing a PIO application.

Assigning GPIO pins to a state machine uses the PIO helper function:

PIO(PINCTRL a,b,c,d,e,f,g)

- a/ the number of SIDE SET pins (0...5), SIDE SET can write 5 pins at once
- b/ the number of SET pins (0...5), SET can write 5 pins at once
- c/ the number of OUT pins (0...31), OUT can write 32 pins at once
- d/ the lowest pin for IN pins (GP0.....GP31) IN can read up to 32 pins at once
- e/ the lowest pin for SIDE SET (GP0.....GP31)
- f/ the lowest pin for SET (GP0.....GP31)
- g/ the lowest pin for OUT (GP0.....GP31)

Ranges for the different functions can overlap, be identical, or adjacent.

EXECUTE CONTROL

The execute control register EXECCTRL configures the program flow. There is a field that connects a GPIO pin to a conditional jump (JMP instruction), and fields that hold the line address of the main program loop begin (.WRAP TARGET) and end (.WRAP).

If we want the program flow to change in response of a GPIO pin state, a JMP PIN is used. The JMP pin is assigned in the execute control configuration (there can only be 1 pin per state machine) and the JMP happens only when the pin is high).

The state machine program starts at the beginning and runs until it reaches the end. In above demo program, the program loops from the end to beginning using a (unconditional) JMP instruction. An alternative way to using the JMP instruction is defining the beginning of the loop (WRAP TARGET = line 1) and end of the loop (WRAP = line 2) and configure the state machine to only execute these instructions in between. The JMP instruction in line 3 is obsolete when WRAP/WRAP TARGET is used.

PIO(EXECCTRL a,b,c)

- a/ the GPIO pin for conditional JMP (e.g. GP0)
- b/ the WRAP TARGET line number (e.g. 1)
- c/ the WRAP line number (e.g. 2)

SHIFT CONTROL

The IN and OUT instructions shift data from the FIFO register to the GPIO pins. In between MMBasic and the PIO, 32bit words can be communicated. Since both the ARM cores and the PIO processors operate independently, the data is exchanged through FIFO's. The ARM (MMBasic) puts data in the FIFO, PIO reads it. This uses the TX FIFO. The other way around uses the RX FIFO. The FIFO's are normally 4 words deep but can be configured to a single 8 word deep RX or TX FIFO.

The PIO can "shift" data IN the RX FIFO from the MSB side, or from the LSB side. That is set with the IN SHIFTDIR bit. Similar the OUT SHIFTDIR bit for OUT data. The autopull and autopush flags in combination with the pull and push thresholds determine when FIFO is replenished.

PIO(SHIFTCTRL a,b,c,d,e,f,g,h)

- a/ push threshold (leave 0 for now)
- b/ pull threshold (leave 0 for now)
- c/ autopush (leave 0 for now)
- d/ autopull (leave 0 for now)
- e/ IN-shiftdir (1 = shift MSB, 0 = shift LSB)
- f/ OUT-shiftdir (1 = shift MSB, 0 = shift LSB)
- g/ fjoin_rx (join TX and RX fifo to 1 RX fifo)
- h/ fjoin_tx (join TX and RX fifo to 1 TX fifo)

状态机可以设置引脚的状态（SET是状态机指令），但也可以使用OUT指令向一个或多个GPIO引脚输出串行数据。或者使用IN指令读取串行数据。并且GPIO引脚可以作为任何状态机指令的副作用被设置（侧面设置）。对于每种接口方法，可以将不同的引脚映射到状态机。

重要的是要理解，这些指令适用于连续的引脚。这意味着可以控制一系列引脚，从最低的 GPx 引脚编号（例如 GP0）开始，旁边的引脚可以包括在内（最多总共 5 个引脚）。因此，GP0、GP1、GP2 是有效的 IO 引脚组合。GP0、GP1、GP6 不是。在设计 PIO 应用程序时请考虑这一点。

将 GPIO 引脚分配给状态机使用 PIO 辅助函数：

PIO(PINCTRL a,b,c,d,e,f,g)

- a/ SIDE SET 引脚的数量 (0...5)，SIDE SET 可以同时写入 5 个引脚
- b/ SET 引脚的数量 (0...5)，SET 可以同时写入 5 个引脚
- c/ OUT 引脚的数量 (0...31)，OUT 可以同时写入 32 个引脚
- d/ 输入引脚的最低引脚 (GP0.....GP31)，IN 可以同时读取最多 32 个引脚
- e/ SIDE SET 的最低引脚 (GP0.....GP31)
- f/ SET 的最低引脚 (GP0.....GP31)
- g/ 输出的最低引脚 (GP0.....GP31)

不同函数的范围可以重叠、相同或相邻。

执行控制

执行控制寄存器 EXECCTRL 配置程序流程。有一个字段将 GPIO 引脚连接到条件跳转（JMP 指令），还有字段保存主程序循环的行地址开始 (.WRAP TARGET) 和结束 (.WRAP)。

如果我们希望程序流程根据 GPIO 引脚状态变化，则使用 JMP PIN。JMP 引脚在执行控制配置中分配（每个状态机只能有一个引脚），并且只有当引脚为高电平时才会发生 JMP。

状态机程序从开始处启动，并运行直到达到结束。在上述演示程序中，程序使用（无条件）JMP 指令从结束循环到开始。使用 JMP 指令的另一种方法是定义循环的开始（WRAP TARGET = 行 1）和循环的结束（WRAP = 行 2），并配置状态机仅在这两者之间执行这些指令。在使用WRAP/WRAP TARGET时，第3行的JMP指令已过时。

PIO(EXECCTRL a,b,c)

- a/ 条件JMP的GPIO引脚（例如 GP0）
- b/ WRAP TARGET行号（例如 1）
- c/ WRAP行号（例如 2）

SHIFT CONTROL

IN和OUT指令将数据从FIFO寄存器移至GPIO引脚。在MMBasic和PIO之间，可以传输32位字。由于ARM核心和PIO处理器独立运行，数据通过FIFO进行交换。ARM（MMBasic）将数据放入FIFO，PIO读取它。这使用TX FIFO。反向操作使用RX FIFO。FIFO通常深度为4个字，但可以配置为单个8字深的RX或TX FIFO。

PIO可以从MSB侧或LSB侧“移入”RX FIFO中的数据。这是通过 IN SHIFTDIR 位设置的。类似于用于输出数据的 OUT SHIFTDIR 位。自动拉取和自动推送标志与拉取和推送阈值结合，决定了何时补充 FIFO。

PIO(SHIFTCTRL a,b,c,d,e,f,g,h)

- a/ 推送阈值 (现在留为 0)
- b/ 拉取阈值 (现在留为 0)
- c/ 自动推送 (现在留为 0)
- d/ 自动拉取 (现在留为 0)
- e/ 输入移位方向 (1 = 移位 MSB, 0 = 移位 LSB)
- f/ 输出移位方向 (1 = 移位 MSB, 0 = 移位 LSB)
- g/ fjoin_rx (将 TX 和 RX FIFO 合并为 1 个 RX FIFO)
- h/ fjoin_tx (将 TX 和 RX FIFO 合并为 1 个 TX FIFO)

WRITING THE STATE MACHINE CONFIGURATION

A state machine configuration is written using the command:

```
PIO INIT MACHINE a,b,c,d,e,f,g
    a/ the PIO          (0 or 1)
    b/ the state machine number (0...3)
    c/ frequency        (CPUSPEED/65536...CPUSPEED in Hz)
    d/ pincontrol value (PIO(PINCTRL .....))
    e/ execure control value (PIO(EXECCTRL.....))
    f/ shiftcontrol value (PIO(SHIFCTRL.....))
    g/ start address     (0....31, the line at which the state machine starts executing)
```

STARTING AND STOPPING A STATE MACHINES

Once the PIO is configured, you can start and stop the state machine using:

```
PIO START a,b
PIO STOP a,b
    a/ the PIO number      (0 or 1)
    b/ the state machine    (0...3)
```

Note that when stopping a state machine, it stops right where it is. To restart the state machine it is advisable to PIO INIT MACHINE first.

EXAMPLE PROGRAM 1

A complete PIO implementation that toggles a GPIO pins can be implemented in MMBasic as shown below. Connect a buzzer to GP0, and hear the audio tone generated by the PIO.

```
'disconnect ARM from GP0
setpin gp0,pi01                                'use GP0 as output pin for PIO 1

'pio program used
'0 E081    'SET pin output
'1 E001    'SET pin high
'2 E000    'SET pin low
'3 0001    'jmp 1

'program pio 1 using an array to write the program in PIO memory, and start
Dim a%(7)=(&h0001E000E001E081,0,0,0,0,0,0,0)
PIO program 1,a%()

'configure pio 1 statemachine 0
p=Pio(pinctrl 0,1,,,gp0,)                      'define SET uses 1 pin, and that is GP0
f=2029                                         '2029 Hz is lowest frequency for CPUSPEED
133000
PIO init machine 1,0,f,p                         'use default for execctrl, shiftctrl, start
address(=0)

'start the PIO 1 state machine 0
PIO start 1,0
```

Note that the MMBasic program ends, but the sound on the buzzer continues. PIO is independent of the ARM processor, and continues until it is stopped. Entering the MMBasic editor stops the PIO.

FIFO's

MMBasic and the PIO exchange information using FIFO's. The PIO's PUSH data into the RX FIFO (MMBasic is the receiver), or PULL data from the TX FIFO (MMBasic is the transmitter).

When PIO is fetching data from the FIFO the data is transferred to the OSR (Output Shift Register), from there is can be processed. The PIO can push the data from the ISR (Input shift register) into the FIFO. Additionally, the PIO has 2 registers X and Y that can be used for storage, or counting. PIO cannot add or subtract or compare.

Data flow:

```
MMBasic -> FIFO -> OSR -> PIO (or pins)
PIO (or pins) -> ISR -> FIFO -> MMBasic
```

MMBasic can write data into the TX FIFO and read data from the RX FIFO using:

编写状态机配置

状态机配置是使用以下命令编写的：

PIO 初始化机器 a,b,c,d,e,f,g	
a/ PIO	(0 或 1)
b/ 状态机编号 (0...3)	
c/ 频率	(CPU速度/65536...CPU速度以赫兹为单位)
d/ 引脚控制值	(PIO(PINCTRL.....))
e/ 执行控制值	(PIO(EXECCTRL.....))
f/ 移位控制值	(PIO(SHIFCTRL.....))
g/ 起始地址	(0....31, 状态机开始执行的行)

启动和停止状态机

一旦PIO配置完成，您可以使用以下命令启动和停止状态机：

PIO 启动 a,b	
PIO 停止 a,b	
a/ PIO 编号	(0或1)
b/ 状态机	(0...3)

请注意，当停止状态机时，它会在当前位置停止。要重新启动状态机，建议先执行PIO 初始化机器。

示例程序 1

一个完整的PIO实现，可以在MMBasic中实现GPIO引脚的切换，如下所示。

将蜂鸣器连接到GP0，并听到由PIO生成的音频音调。

```
' 将ARM从GP0断开
设置引脚 gp0,pio1          ' 将 GP0 用作 PIO 1 的输出引脚

' pio 程序使用
'0 E081      ' 设置引脚输出
'1 E001      ' 设置引脚高
'2 E000      ' 设置引脚低
'3 0001      ' jmp 1

' 使用数组在 PIO 内存中写入程序并启动 pio 1
Dim a%(7)=(&h0001E000E001E081,0,0,0,0,0,0,0)
PIO 程序 1,a%()

' 配置 pio 1 状态机 0
p=Pio(pinctrl 0,1,,,gp0,)          ' 定义 SET 使用 1 个引脚，即 GP0
f=2029                                ' 2029 赫兹是 CPUSPEED 的最低频率
133000
PIO 初始化机器 1,0,f,p                ' 使用 execctrl、shiftctrl、start 的默认值
地址(=0)

' 启动 PIO 1 状态机 0
PIO 启动 1,0
```

请注意，MMBasic 程序结束，但蜂鸣器的声音仍在继续。PIO独立于ARM处理器，并持续运行直到被停止。进入MMBasic编辑器会停止PIO。

FIFO's

MMBasic和PIO通过FIFO交换信息。PIO将数据推送到RX FIFO（MMBasic是接收器），或从TX FIFO中拉取数据（MMBasic是发射器）。

当PIO从FIFO中获取数据时，数据会被传输到OSR（输出移位寄存器），然后可以进行处理。PIO可以将数据从ISR（输入移位寄存器）推送到FIFO。此外，PIO还有两个寄存器X和Y，可以用于存储或计数。PIO不能进行加法、减法或比较。

数据流：

MMBasic-> FIFO-> OSR-> PIO (或引脚)
PIO (或引脚) -> ISR-> FIFO-> MMBasic

MMBasic可以使用以下方式将数据写入TX FIFO并从RX FIFO读取数据：

```

PIO READ a,b,c,d
PIO WRITE a,b,c,d
    a/ PIO number          (0 or 1)
    b/ state machine number (0...3)
    c/ number of 32 bit words (1...4)
    d/ integer variable name(i.e. variable% or array%())

```

PIO CLEAR clears all the PIO FIFO's, as does PIO START and PIO INIT MACHINE.

The MMBasic program doesn't need to wait for data in the FIFO to appear since the RX FIFO can be assigned an interrupt. The MMBasic interrupt routine can fetch the data from the FIFO.

Similar for TX interrupt in which case MMBasic gets an interrupt when data is needed for the TX FIFO.

```

PIO INTERRUPT a,b,c,d
    a/ PIO                  (0 or 1)
    b/ state machine        (0...3)
    c/ Name of RX interrupt handler (i.e. "myRX_Interrupt" or 0 to disable)
    d/ Name of TX interrupt handler (i.e. "myTX_Interrupt" or 0 to disable)

```

EXAMPLE PROGRAM 2

Below program explains many of the above presented MMBasic functions and commands. The program reads a NES controller (SPI) connected to the PicoMite. The NES controller consists of a HEF4021 shift register connected to 8 push button switches.

Program uses: wrap and wrap target, IN, side set and delay, PUSH, PIO READ. GP0 and GP1 are in SET for pin direction, and in side set for compact code.

The wiring is as defined in the code:

```

'disconnect ARM from GP0/1/2
    setpin gp0,piol           'clock out
    setpin gp1,piol           'load out
    setpin gp2,piol           'data in

'PIO program
    PIO assemble 1,".program NES"
    PIO assemble 1,".side_set 2"
    PIO assemble 1,".line 0"
    PIO assemble 1,"SET pindirs,&b11"
    PIO assemble 1,".wrap target"
    PIO assemble 1,"IN null,32 side 2"
    PIO assemble 1,"SET X,7 side 0"
    PIO assemble 1,"loop:"
    PIO assemble 1,"IN pins,1 side 0"
    PIO assemble 1,"JMP X-- loop side 1"
    PIO assemble 1,"PUSH side 0 [7]"
    PIO assemble 1,".wrap"
    PIO assemble 1,".end program list"

'configure piol
    p=pio(pinctrl 2,2,,gp2,gp0,gp0,)           'GP0,GP1 out (SET and SIDE SET), GP2
    IN
    f=1e5                                         '100kHz
    s=PIO(shiftctrl 0,0,0,0,0,0)                 'shift in from LSB for IN (and OUT)
    e=PIO(execctrl gp0,PIO(.wrap target),PIO(.wrap)) 'wrap and wrap target

'write the configuration
    PIO init machine 1,0,f,p,e,s,0

'start the piol code
    PIO start 1,0

'Check the the read data in MMBasic and print
    dim d%
    do
        pio read 1,0,1,d%
        print bin$(d%)

```

PIO 读取 a,b,c,d

PIO 写入 a,b,c,d

a/ PIO 数字	(0或1)
b/ 状态机数字	(0...3)
c/ 32 位字的数量	(1...4)
d/ 整数变量名称 (即变量%或数组%())	

PIO CLEAR 清除所有 PIO FIFO， PIO START 和 PIO INIT MACHINE 也会这样做。

MMBasic 程序不需要等待 FIFO 中的数据出现，因为 RX FIFO 可以分配一个中断。MMBasic 中断例程可以从 FIFO 中获取数据。

TX 中断也是类似的，在这种情况下，当需要数据到 TX FIFO 时，MMBasic 会收到一个中断。

PIO 中断 a,b,c,d

a/ PIO	(0或1)
b/ 状态机	(0...3)
c/ RX 中断处理程序的名称 (即 "myRX_Interrupt" 或 0 以禁用)	
d/ TX 中断处理程序的名称 (即 "myTX_Interrupt" 或 0 以禁用)	

示例程序 2

下面的程序解释了许多上述介绍的MMBasic函数和命令。该程序读取连接到PicoMite的NES控制器（S PI）。NES控制器由一个HEF4021移位寄存器和8个按钮开关组成。

程序使用：换行和换行目标，输入，侧面设置和延迟，推送，PIO读取。GP0和GP1在设置中用于引脚方向，并在侧面设置中用于紧凑代码。

接线如代码中定义：

```
' 断开ARM与GP0/1/2的连接
设置引脚 gp0,pio1                                ' 时钟输出
设置引脚 gp1,pio1                                ' 加载输出
设置引脚 gp2,pio1                                ' 数据输入

' PIO程序
PIO汇编 1, ".程序 NES"
PIO汇编 1, ".侧面设置 2"
PIO汇编 1, ".行 0"
PIO汇编 1, "设置引脚方向 ,&b11"
PIO 汇编 1, ".换行目标"
PIO 汇编 1, "IN null,32 side 2"
PIO 汇编 1, "SET X,7 side 0"
PIO 汇编 1, "循环:"
PIO 汇编 1, "IN pins,1 side 0"
PIO 汇编 1, "JMP X-- 循环 side 1" '跳转到循环,
PIO 汇编 1, "PUSH side 0 [7]"
PIO 汇编 1, ".换行"
PIO 汇编 1, ".结束程序列表"

' 配置 pio1
p=PIO(pinctrl 2,2,,gp2,gp0,gp0,)           ' GP0,GP1 输出 (设置和侧面设置) , GP2
输入
f=1e5                                              ' 100千赫
s=PIO(shiftctrl 0,0,0,0,0,0)                   ' 从 LSB 进行输入 (和输出)
e=PIO(execctrl gp0,PIO(.wrap target),PIO(.wrap)) ' 换行和换行目标

' 写入配置
PIO 初始化机器 1,0,f,p,e,s,0

' 启动 pio1 代码
PIO 启动 1,0

' 检查 MMBasic 中读取的数据并打印 dim d% do
    pio 读取 1,0,1,d%
    打印 bin$(d%)
```

```

    pause 200
loop
END

```

DMA To and From the FIFOs

The way that DMA works is as follows:

When reading from the FIFO the DMA controller waits on data being in the FIFO and when it appears transfers that data into processor memory. Each time it reads it increments the pointer into the processor memory so that it can, for example, incrementally fill an array as each and every data item is made available.

When writing to the FIFO the DMA controller writes data from processor memory to the FIFO automatically waiting whenever the FIFO is full. Thus, data can be prepared in an array and the DMA controller will stream that data to the PIO FIFO as fast as the PIO program requires it.

DMA can transfer a 32-bit word, a 16-bit short, or an 8-bit byte and when setting up DMA you need to tell it the size of the transfer and how many transfers to make. Because each transfer will increment the memory pointer by 1,2, or 4 bytes MMBasic must deal with the data packed into memory rather than the 64-bits used for MMBasic integers and floats. Luckily MMBasic implements two commands MEMORY PACK and MEMORY UNPACK to do this very efficiently but it could equally be done using standard BASIC arithmetic.

The DMA can be configured to repeatedly loop data into or out of a section of memory (a ring buffer)

The commands are:

PIO DMA_IN a,b,c,d,e,f,g	
PIO DMA_OUT a,b,c,d,e,f,g	
a/ pio	(0 or 1)
b/ state machine	(0...3)
c/ nbr	(number of words to be transferred)
d/ data%()	(integer array name)
e/ completioninterrupt	(where to go when done, optional)
f/ transfersize	(8/16/32, optional)
g/ loopbackcount	(used data%() as a ring buffer, optional, loopbackcount = 2^n)

The DMA will start the state machine automatically and there is no need for a PIO START command. But, before starting the transfer make sure a fresh PIO INIT MACHINE is done, so the state machine starts at the required start address.

When a ring buffer is used, it requires special preparation:

PIO MAKE RING BUFFER a,b	
a/ name of integer buffer	
b/ size of the array in bytes	

Example :

```

DIM packed%
PIO MAKE RING BUFFER packed%,4096

```

packed% will then be an integer array holding 4096/8=512 integers

This can then be used by the DMA for a loopbackcounter with DMA of 1024 32-bit words, 2048 16-bit shorts or 4096 8-bit bytes

EXAMPLE PROGRAM 3

This program brings everything together and uses DMA to read 128 samples from the PIO RX FIFO. For the demonstration, GP0 to GP5 are outputs of 3 PWMS, and are ,at the same time, sampled by the PIO as a 6 channel logic analyser or oscilloscope. The 128 samples are sent to the serial port as waveforms.

This program also demonstrates PIO DMA RX, MEMORY UNPACK, the use of buffers.

```

'generate a 50Hz 3 phase test signal to demonstrate the DMA on 6 GPIO pins.
SetPin gp0,pwm  'CH 0a
SetPin gp1,pwm  'CH 0b
SetPin gp2,pwm  'CH 1a
SetPin gp3,pwm  'CH 1b
SetPin gp4,pwm  'CH 2a
SetPin gp5,pwm  'CH 2b

Fpwm = 50: PW = 100 / 3
PWM 0, Fpwm, PW, PW - 100, 1, 1
PWM 1, Fpwm, PW, PW - 100, 1, 1

```

暂停 200
循环
结束

DMA 从 FIFO 到 FIFO

DMA 的工作方式如下：

在从 FIFO 读取时，DMA 控制器会等待 FIFO 中有数据，当数据出现时，将其传输到处理器内存中。每次读取时，它会将指针递增到处理器内存中，以便例如在每个数据项可用时逐步填充数组。

在向 FIFO 写入时，DMA 控制器会自动将数据从处理器内存写入 FIFO，并在 FIFO 满时等待。因此，可以在数组中准备数据，DMA 控制器将根据 PIO 程序的需求以最快的速度将数据流式传输到 PIO FIFO。

DMA 可以传输 32 位字、16 位短整型或 8 位字节，在设置 DMA 时，需要告诉它传输的大小以及要进行多少次传输。由于每次传输会将内存指针递增 1、2 或 4 个字节，MMBasic 必须处理打包到内存中的数据，而不是用于 MMBasic 整数和浮点数的 64 位。幸运的是，MMBasic 实现了两个命令 MEMORY PACK 和 MEMORY UNPACK，可以非常高效地完成此操作，但也可以使用标准 BASIC 算术来实现。

DMA 可以配置为重复循环数据进出内存的某个区域（环形缓冲区）

命令如下：

PIO DMA_IN a,b,c,d,e,f,g	
PIO DMA_OUT a,b,c,d,e,f,g	
a/ pio	(0 或 1)
b/ 状态机	(0...3)
c/ 编号	(要传输的字数)
d/ 数据%()	(整数数组名称)
e/ 完成中断	(完成后要去的地方，选项)
f/ 传输大小	(8/16/32，选项)
g/ 回环计数	(使用数据%() 作为环形缓冲区，选项，回环计数 = 2^n)

DMA 将自动启动状态机，无需 PIO START 命令。但是，在开始传输之前，请确保已完成新的 PIO INIT MACHINE，以便状态机从所需的起始地址开始。

使用环形缓冲区时，需要特别准备：

PIO MAKE RING BUFFER a,b	
a/ 整数缓冲区的名称	
b/ 数组的大小（以字节为单位）	

示例：

```
DIM packed%
PIO MAKE RING BUFFER packed%, 4096
```

然后，packed% 将是一个整数数组，包含 $4096/8=512$ 个整数

这可以被 DMA 用于一个回环计数器，支持 1024 个 32 位字、2048 个 16 位短整型或 4096 个 8 位字节

示例程序 3

该程序将所有内容结合在一起，并使用 DMA 从 PIO RX FIFO 读取 128 个样本。为了演示，GP0 到 GP5 是 3 个 PWM 的输出，同时被 PIO 作为 6 通道逻辑分析仪或示波器进行采样。这 128 个样本作为波形发送到串行端口。

该程序还演示了 PIO DMA RX、内存解包和缓冲区的使用。

' 生成一个 50 赫兹的三相测试信号，以演示在 6 个 GPIO 引脚上的 DMA。

```
SetPin gp0, pwm      ' CH 0a
SetPin gp1, pwm      ' CH 0b
SetPin gp2, pwm      ' CH 1a
SetPin gp3, pwm      ' CH 1b
SetPin gp4, pwm      ' CH 2a
SetPin gp5, pwm      ' CH 2b
```

```
Fpwm = 50: PW = 100 / 3
PWM 0, Fpwm, PW, PW - 100, 1, 1
PWM 1, Fpwm, PW, PW - 100, 1, 1
```

```

PWM 2, Fpwm, PW, PW - 100, 1, 1
PWM sync 0, 100/3, 200/3

'----- LA code PIO -----
'PIO code to sample GP0..GP6 as elementary logic analyser
PIO clear 1

'in this program the PIO reads GP0..GP5 brute force
'and pushes data into FIFO. The clock speed determines the
'sampling rate. There are 2 instructions per cycle
'taking 10000/2 / 50 = 100 samples per 50Hz cycle.

PIO assemble 1,.program push"
PIO assemble 1,.line 0"
PIO assemble 1,.wrap target"

PIO assemble 1,"IN pins,6"           "'get 6 bits from GPIO pins (GP0..GP5)
PIO assemble 1,"PUSH block"         "'only push data when FIFO has room

PIO assemble 1,.wrap"
PIO assemble 1,.end program list"

'configuration
f=1e4                                'PIO run at 10kHz
p=Pio(pinctrl 0,0,0,gp0,,,)            'IN base = GP0
e=Pio(execctrl gp0,PIO(.wrap target),PIO(.wrap)) 'wrap 1 to 0, gp0 is default
s=Pio(shiftctrl 0,0,0,0,0,0)           'shift in through LSB, out is not used

'write the configuration, running 10kHz (data in FIFO 10us after edge GP0)
PIO init machine 1,0,f,p,e,s,0        'start address = 0

'----- LA code MMBasic -----
'define memory buffers
Dim a$(1)=("_","-")                  'characters for the printout
length%=64                            'size of the packed array
Dim data%(2*length%-1)                'array to put the 32 bit samples FIFO
format
Dim packed%(length%-1)                'DMA array to pack 32 bit samples in 64
bit integers

'let the DMA machine run, and repeat at will
Do
    PIO DMA RX 1,0,2*length%,packed%,ReadyInt
    print "press any key to restart sampling"
    do:loop while inkey$=""
Loop
End

'-----SUBS MMBasic -----
Sub ReadyInt
    'stop the PIO and re-init for next run
    PIO stop 1,0
    PIO init machine 1,0,f,p,e,s,0      'start address = 0

    'get the data from the packed DMA buffer and unpack to original 32 bit format
    Memory unpack packed%,data%,2*length%,32

    'Serial output as if logic analyzer traces
    For j=0 To 5
        mask%=2^j
        For i=0 To 2*length%-1
            If i<106 Then Print a$(((data%(i) And mask%)=mask%));
        Next i
        Print : Print
    Next j
End Sub

```

```

PWM 2, Fpwm, PW, PW - 100, 1, 1
PWM sync 0, 100/3, 200/3

'-----LA代码 PIO -----
'PIO代码用于采样GP0..GP6作为基本逻辑分析仪
PIO clear 1

'在这个程序中，PIO从GP0..GP5进行强制读取，并将数据推送到
FIFO中。时钟速度决定了采样率。每个周期有2条指令，'计算为10000/2 /
50 = 每50Hz周期100个样本。

PIO assemble 1, ".program push"
PIO assemble 1, ".line 0"
PIO assemble 1, ".wrap target"

PIO assemble 1, "IN pins,6"           ''从GPIO引脚(GP0..GP5)获取6位数据
PIO assemble 1, "PUSH block"         '仅在FIFO有空间时推送数据

PIO assemble 1, ".wrap"
PIO assemble 1, ".end program list"

'配置
f=1e4                                'PIO以10kHz运行
p=Pio(pinctrl 0,0,0,gp0,,,)          '输入基准=GP0
e=Pio(execctrl gp0,PIO(.wrap target),PIO(.wrap)) '将1换行到0, gp0是默认值
s=Pio(shiftctrl 0,0,0,0,0)            '通过LSB进行移位,输出未使用

'写入配置,运行10kHz(数据在边缘GP0后10微妙进入FIFO)
PIO 初始化机器 1,0,f,p,e,s,0        '起始地址=0

'-----LA代码 MMBasic -----
'定义内存缓冲区
Dim a$(1)=(" ", "-")                '打印输出的字符
length%=64                            '打包数组的大小
Dim data%(2*length%-1)                '用于放置32位样本的FIFO数组
格式
Dim packed%(length%-1)                'DMA数组用于打包32位样本到64
位整数

'让DMA机器运行,并随意重复
执行
    PIO DMA RX 1,0,2*length%,packed%,ReadyInt
    打印 "按任意键重新开始采样"
    执行:循环 直到 inkey$=""
循环
结束

'-----子程序 MMBasic -----
子程序 ReadyInt
'停止PIO并重新初始化以便下次运行
PIO 停止 1,0
PIO 初始化机器 1,0,f,p,e,s,0        '起始地址=0

'从打包的DMA缓冲区获取数据并解包为原始32位格式
内存解包 packed%,data%,2*长度%,32

'串行输出就像逻辑分析仪的跟踪
对于 j=0 到 5
    mask%=2^j
    对于 i=0 到 2*长度%-1
        如果 i<106 则 打印 a$(((data%(i) 和 mask%)=mask%));
        下一个 i
        打印 : 打印
    下一个 j
结束子程序

```

Appendix G

Programming in BASIC - A Tutorial

The PicoMite is programmed using the BASIC programming language. The PicoMite version of BASIC is called MMBasic which loosely emulates the Microsoft BASIC interpreter that was popular years ago.

The BASIC language was introduced in 1964 by Dartmouth College in the USA as a computer language for teaching programming and accordingly it is easy to use and learn. At the same time, it has proved to be a competent and powerful programming language and as a result it became very popular in the late 70s and early 80s. Even today some large commercial data systems are still written in the BASIC language (primarily Pick Basic).

For the PicoMite the greatest advantage of BASIC is its ease of use. Some more modern languages such as C and C++ can be truly mind bending but with BASIC you can start with a one line program and get something sensible out of it. MMBasic is also powerful in that you can draw sophisticated graphics, manipulate the external I/O pins to control other devices and communicate with other devices using a range of built-in communications protocols.

Command Prompt

Interaction with MMBasic is done via the console at the command prompt (ie, the greater than symbol (>) on the console). On startup MMBasic will issue the command prompt and wait for some command to be entered. It will also return to the command prompt if your program ends or if it generated an error message.

When the command prompt is displayed you have a wide range of commands that you can enter and execute. Typically they would list the program held in memory (LIST) or edit it (EDIT) or perhaps set some options (the OPTION command). Most times the command is just RUN which instructs MMBasic to run the program held in program memory.

Almost any command can be entered at the command prompt and this is often used to test a command to see how it works. A simple example is the PRINT command (more on this later), which you can test by entering the following at the command prompt:

```
PRINT 2 + 2
```

and not surprisingly MMBasic will print out the number 4 before returning to the command prompt.

This ability to test a command at the command prompt is useful when you are learning to program in BASIC, so it would be worthwhile having the PicoMite handy for the occasional test while you are working through this tutorial.

Structure of a BASIC Program

A BASIC program starts at the first line and continues until it runs off the end or hits an END command - at which point MMBasic will display the command prompt (>) on the console and wait for something to be entered.

A program consists of a number of statements or commands, each of which will cause the BASIC interpreter to do something (the words *statement* and *command* generally mean the same and are used interchangeable in this tutorial).

Normally each statement is on its own line but you can have multiple statements in the one line separated by the colon character (:).

For example;

```
A = 24.6 : PRINT A
```

G

BASIC编程 - 教程

PicoMite 使用BASIC编程语言进行编程。PicoMite 版本的BASIC称为MMBasic，它大致模拟了多年前流行的微软BASIC解释器。

BASIC语言于1964年由美国达特茅斯学院引入，作为一种用于教授编程的计算机语言，因此它易于使用和学习。与此同时，它已被证明是一种称职且强大的编程语言，因此在70年代末和80年代初变得非常流行。即使在今天，一些大型商业数据系统仍然是用BASIC语言编写的（主要是Pick Basic）。

对于PicoMite来说，BASIC最大的优势在于其易用性。一些更现代的语言，如C和C++，可能会让人感到困惑，但使用BASIC，你可以从一行程序开始，并得到一些合理的结果。MM Basic也很强大，你可以绘制复杂的图形，操控外部I/O引脚以控制其他设备，并使用一系列内置通信协议与其他设备进行通信。

命令提示符

与MMBasic的交互是通过控制台上的命令提示符（即控制台上的大于号(>)）进行的。在启动时，MMBasic会发出命令提示符并等待输入命令。如果您的程序结束或生成了错误消息，它也会返回到命令提示符。

当命令提示符显示时，您可以输入并执行各种命令。通常，它们会列出保存在内存中的程序（LIST）或编辑它（EDIT），或者设置一些选项（OPTION命令）。大多数情况下，命令只是RUN，它指示MMBasic运行保存在程序内存中的程序。

几乎可以在命令提示符下输入任何命令，这通常用于测试命令以查看其工作原理。一个简单的例子是PRINT命令（稍后会详细介绍），您可以通过在命令提示符下输入以下内容来测试：

```
PRINT 2 + 2
```

毫不奇怪，MMBasic会打印出数字4，然后返回到命令提示符。

在命令提示符下测试命令的能力在您学习BASIC编程时非常有用，因此在您完成本教程时，随时准备好使用PicoMite进行偶尔的测试是值得的。

BASIC程序的结构

BASIC程序从第一行开始，直到运行到结束或遇到END命令为止——此时MMBasic将在控制台上显示命令提示符(>)并等待输入。

一个程序由多个语句或命令组成，每个语句或命令都会使BASIC解释器执行某个操作（在本教程中，词语语句和命令通常是相同的，可以互换使用）。

通常每个语句占据一行，但您可以在同一行中使用冒号字符(:)分隔多个语句。

例如；

```
A = 24.6 : 打印 A
```

Each line can start with a line number. Line numbers were mandatory in the early BASIC interpreters however modern implementations (such as MMBasic) do not need them. You can still use them if you wish but they have no benefit and generally just clutter up your programs.

This is an example of a program that uses line numbers:

```
50 A = 24.6
60 PRINT A
```

A line can also start with a label which can be used as the target for a program jump using the GOTO command. This will be explained in more detail when we cover the GOTO command but this is an example (the label name is JmpBack):

```
JmpBack: A = A + 1
PRINT A
GOTO JmpBack
```

Comments

A comment is any text that follows the single quote character ('). A comment can be placed anywhere and extends to the end of the line. If MMBasic runs into a comment it will just skip to the end of it (ie, it does not take any action regarding a comment).

Comments should be used to explain non obvious parts of the program and generally inform someone who is not familiar with the program how it works and what it is trying to do. Remember that after only a few months a program that you have written will have faded from your mind and will look strange when you pick it up again. For this reason you will thank yourself later if you use plenty of comments.

The following are some examples of comments:

```
' calculate the hypotenuse
PRINT SQR(a * a + b * b)
```

or

```
INPUT var      ' get the temperature
```

Older BASIC programs used the command REM to start a comment and you can also use that if you wish but the single quote character is easier to use and more convenient.

The PRINT Command

There are a number of common commands that are fundamental and we will cover them in this tutorial but arguably the most useful is the PRINT command. Its job is simple; to print something on the console. This is mostly used to output data for you to see (like the result of calculations) or provide informative messages.

PRINT is also useful when you are tracing a fault in your program; you can use it to print out the values of variables and display messages at key stages in the execution of the program.

In its simplest form the command will just print whatever is on its command line. So, for example:

```
PRINT 54
```

will display on the console the number 54 followed by a new line.

The data to be printed can be something simple like this or an expression, which means something to be calculated. We will cover expressions in more detail later but as an example the following:

```
> PRINT 3/21
0.1428571429
>
```

would calculate the result of three divided by twenty one and display it. Note that the greater than symbol (>) is the command prompt produced by MMBasic – you do not type that in.

每行可以以行号开头。在早期的BASIC解释器中，行号是必需的，但现代实现（如MMBasic）不需要它们。如果您愿意，仍然可以使用行号，但它们没有任何好处，通常只会使您的程序变得杂乱。

这是一个使用行号的程序示例：

```
50 A = 24.6  
60 打印 A
```

一行也可以以标签开头，该标签可以用作使用GOTO命令进行程序跳转的目标。这将在我们讲解GOTO命令时详细说明，但这是一个示例（标签名称是 JmpBack）：

```
JmpBack: A = A + 1  
打印 A  
转到 JmpBack
```

注释

注释是任何跟在单引号字符（'）后面的文本。注释可以放在任何地方并延伸到行的末尾。如果MMBasic遇到注释，它将跳过到注释的末尾（即，它不会对注释采取任何行动）。

注释应当用于解释程序中不明显的部分，并一般性地告知不熟悉该程序的人它是如何工作的以及它试图做什么。请记住，几个月后，您编写的程序可能会淡出您的记忆，当您再次查看时会显得陌生。因此，如果您使用大量注释，您会在以后感谢自己。

以下是一些注释的示例：

```
' 计算斜边  
PRINT SQR(a * a + b * b)
```

或

```
输入 var      ' 获取温度
```

较早的BASIC程序使用命令REM来开始注释，如果您愿意也可以使用它，但单引号字符更容易使用且更方便。

PRINT命令

有许多常见的命令是基础的，我们将在本教程中介绍它们，但可以说最有用的是PRINT命令。它的工作很简单；在控制台上打印某些内容。这主要用于输出数据以供您查看（例如计算结果）或提供信息性消息。

当您在程序中追踪故障时，PRINT也很有用；您可以使用它打印变量的值，并在程序执行的关键阶段显示消息。

在最简单的形式中，该命令将仅打印其命令行上的内容。因此，例如：

```
PRINT 54
```

将在控制台上显示数字54，后面跟着一个换行符。

要打印的数据可以是像这样的简单内容，也可以是一个表达式，这意味着要计算的内容。我们稍后将更详细地介绍表达式，但作为示例，以下内容：

```
> PRINT 3/21  
0.1428571429  
>
```

将计算三除以二十一的结果并显示出来。请注意，大于符号(>)是由MMBasic生成的命令提示符——您不需要输入它。

Other examples of the PRINT command include:

```
> PRINT "Wonderful World"
Wonderful World
> PRINT (999 + 1) / 5
200
>
```

You can try these out at the command prompt.

The PRINT command will also work with multiple values at the same time, for example:

```
> PRINT "The amount is" 345 " and the second amount is" 456
The amount is 345 and the second amount is 456
>
```

Normally each value is separated by a space character as shown in the previous example but you can also separate values with a comma (,). The comma will cause a tab to be inserted between the two values. In MMBasic tabs in the PRINT command are eight characters apart.

To illustrate tabbing, the following command prints a tabbed list of numbers:

```
> PRINT 12, 34, 9.4, 1000
12          34          9.4          1000
>
```

Note that there is a space printed before each number. This space is a place holder for the minus symbol (-) in case the value is negative. You can see the difference with the number 12 in this example:

```
> PRINT -12, 34, -9.4, 1000
-12          34          -9.4          1000
>
```

The print statement can be terminated with a semicolon (;). This will prevent the PRINT command from moving to a new line when it has printed all the text. For example:

```
PRINT "This will be";
PRINT " printed on a single line."
```

Will result in this output:

```
This will be printed on a single line.
```

The message would be look like this without the semicolon at the end of the first line:

```
This will be
printed on a single line.
```

Variables

Before we go much further we need to define what a "variable" is as they are fundamental to the operation of the BASIC language (in fact, most programming languages). A variable is simply a place to store an item of data (ie, its "value"). This value can be changed as the program runs which why it is called a "variable".

Variables in MMBasic can be one of three types. The most common is floating point and this is automatically assumed if the type of the variable is not specified. The other two types are integer and string and we will cover them later. A floating point number is an ordinary number which can contain a decimal point. For example 3.45 or -0.023 or 100.00 are all floating point numbers.

A variable can be used to store a number and it can then be used in the same manner as the number itself, in which case it will represent the value of the last number assigned to it.

PRINT命令的其他示例包括：

```
> PRINT "美好的世界"  
美好的世界  
> PRINT (999 + 1) / 5  
200  
>
```

您可以在命令提示符下尝试这些。

PRINT命令也可以同时处理多个值，例如：

```
> PRINT "金额是" 345 ", 第二个金额是" 456  
金额是 345, 第二个金额是 456  
>
```

通常，每个值之间用空格字符分隔，如前面的示例所示，但您也可以用逗号 (,) 分隔值。逗号会在两个值之间插入一个制表符。在MMBasic中，PRINT命令中的制表符相隔八个字符。

为了说明制表，以下命令打印一个带制表符的数字列表：

```
> 打印 12, 34, 9.4, 1000  
12          34          9.4          1000  
>
```

请注意，每个数字前面都有一个空格。这个空格是负号 (-) 的占位符，以防值为负。在这个例子中，你可以看到数字 12 的区别：

```
> 打印 -12, 34, -9.4, 1000  
-12          34          -9.4          1000  
>
```

打印语句可以用分号 (;) 结束。这将防止 PRINT 命令在打印完所有文本后换行。例如：

```
打印 "这将是";  
打印 "在一行中打印。"
```

将产生以下输出：

```
这将是在一行中打印。
```

如果第一行末尾没有分号，消息将如下所示：

```
这将是  
在一行中打印。
```

变量

在我们进一步之前，我们需要定义什么是“变量”，因为它们是 BASIC 语言（实际上，大多数编程语言）操作的基础。变量只是存储数据项（即其“值”）的地方。这个值可以在程序运行时改变，这就是为什么它被称为“变量”。

MMBasic 中的变量可以是三种类型之一。最常见的是浮点数，如果未指定变量的类型，则自动假定为浮点数。另外两种类型是整数和字符串，我们稍后会介绍它们。浮点数是一个普通数字，可以包含小数点。例如 3.45 或 -0.023 或 100.00 都是浮点数。

变量可以用来存储一个数字，然后可以像使用数字本身一样使用它，在这种情况下，它将表示最后分配给它的数字的值。

As a simple example:

```
A = 3  
B = 4  
PRINT A + B
```

will display the number 7. In this case both A and B are variables and MMBasic used their current values in the PRINT statement. MMBasic will automatically create a variable when it first encounters it so the statement A = 3 both created a floating point variable (the default type) with the name of A and then it assigned the value of 3 to it.

The name of a variable must start with a letter while the remainder of the name can use letters, numbers, the underscore or the full stop (or period) characters. The name can be up to 31 characters long and the case (ie, capitals or not) is not important. Here are some examples:

```
Total_Count  
ForeColour  
temp3  
count  
x  
ThisIsALongVariableName  
increment.value
```

You can change the value of a variable anywhere in your program by using the assignment command, ie:

```
variable = expression
```

For example:

```
temp3 = 24.6  
count = 5  
CTemp = (FTemp - 32) * 0.5556
```

In the last example both CTemp and FTemp are variables and this line converts the value of FTemp (in degrees Fahrenheit) to degrees Celsius and stores the result in the variable CTemp.

Expressions

We have met the term ‘expression’ before in this tutorial and in programming it has a specific meaning. It is a formula which can be resolved by the BASIC interpreter to a single number or value.

MMBasic will evaluate numeric expressions using the same rules that we learnt at school. For example, multiplication and division are performed first followed by addition and subtraction. These are called the rules of precedence and are fully spelt out in this manual.

This means that MMBasic will resolve $2 + 3 * 6$ by first multiplying 3 by 6 giving 18 then adding 2 resulting in a final value of 20. Similarly, both $5 * 4$ and $10 + 4 * 3 - 2$ will also resolve to 20.

If you want to force the interpreter to evaluate parts of the expression first you can surround that part of the expression with brackets. For example, $(10 + 4) * (3 - 2)$ will resolve to 14 not 20 as would have been the case if the brackets were not used. Using brackets does not appreciably slow down the program so you should use them liberally if there is a chance that MMBasic will misinterpret your intention.

As pointed out earlier, you can use variables in an expression exactly the same as straight numbers. For example, this will increment the value of the variable temp by one:

```
temp = temp + 1
```

You can also use functions in expressions. These are special operations provided by MMBasic, for example to calculate trigonometric values.

作为一个简单的例子：

```
A = 3  
B = 4  
PRINT A + B
```

将显示数字7。在这种情况下，A和B都是变量，MMBasic在PRINT语句中使用了它们的当前值。MMBasic会在首次遇到变量时自动创建它，因此语句 A = 3 创建了一个浮点变量（默认类型），名称为 A，并将值3赋给它。

变量的名称必须以字母开头，其余部分可以使用字母、数字、下划线或句点（或周期）字符。名称最长可以为31个字符并且大小写（即大写或小写）并不重要。以下是一些示例：

```
Total_Count  
ForeColour  
temp3  
count  
x  
ThisIsALongVariableName  
increment.value
```

您可以通过使用赋值命令在程序的任何地方更改变量的值，即：

变量 = 表达式

例如：

```
temp3 = 24.6  
count = 5  
CTemp = (FTemp - 32) * 0.5556
```

在上一个例子中，CTemp 和 FTemp 都是变量，这一行将 FTemp（以华氏温度为单位）的值转换为摄氏度，并将结果存储在变量 CTemp 中。

表达式

我们在本教程中之前遇到过‘表达式’这个术语，在编程中它有特定的含义。它是一个可以被 BASIC 解释器解析为单个数字或值的公式。

MMBasic 将使用我们在学校学到的相同规则来求值数值表达式。例如，乘法和除法优先执行，其次是加法和减法。这些被称为优先级规则，并在本手册中有详细说明。

这意味着 MMBasic 将首先计算 $3 * 6$ 得到 18，然后再加上 2，最终结果为 20。同样， $5 * 4$ 和 $10 + 4 * 3 - 2$ 也将解析为 20。

如果你想强制解释器先求值表达式的某些部分，可以用括号将该部分包围起来。例如， $(10 + 4) * (3 - 2)$ 将计算为 14，而不是 20，如果没有使用括号的话。使用括号不会显著减慢程序的运行速度，因此如果有可能导致 MMBasic 错误解读你的意图，应该自由使用它们。

如前所述，你可以在表达式中使用变量，方式与使用数字完全相同。

例如，这将使变量 temp 的值增加一：

```
temp = temp + 1
```

你也可以在表达式中使用函数。这些是 MMBasic 提供的特殊操作，例如用于计算三角函数值。

As an example, the following will print the length of the hypotenuse of a right angled triangle using the SQR() function which returns the square root of a number (a and b are variables holding the lengths of the other sides):

```
PRINT SQR(a * a + b * b)
```

MMBasic will first evaluate this expression by multiplying a by a, then multiplying b by b, then adding the results together. The resulting number is then passed to the SQR() function which will calculate the square root of that number (ie, the hypotenuse) and return it for the PRINT command to display.

Some other mathematical functions provided by MMBasic include:

SIN(r) – the sine of r

COS(r) – the cosine of r

TAN(r) – the tangent of r

There are many more functions available to you and they are all listed earlier in this manual.

Note that in the above trigonometric functions the value passed to the function (ie, 'r') is the angle in radians. In MMBasic you can use the function RAD(d) to convert an angle from degrees to radians ('d' is the angle in degrees).

Another feature of most programming languages including BASIC is that you can nest function calls within each other. For example, given the angle in degrees (ie, 'd') the sine of that angle can be found with this expression:

```
PRINT SIN(RAD(d))
```

In this case MMBasic will first take the value of d and convert it to radians using the RAD() function. The output of this function then becomes the input to the SIN() function.

The IF Statement

Making decisions is at the core of most computer programs and in BASIC this is usually done with the IF statement. This is written almost like an English sentence:

IF *condition* THEN *action*

The *condition* is usually a comparison such as equals, less than, more than, etc.

For example:

```
IF Temp < 25 THEN PRINT "Cold"
```

Temp would be a variable holding the current temperature (in °C) and PRINT "Cold" the action to be done.

There are a range of tests that you can make:

=	equals	<>	not equal
<	less than	<=	less than or equals
>	greater than	>=	greater than or equals

You can also add an ELSE clause which will be executed if the initial condition tested false:

IF *condition* THEN *true-action* ELSE *false-action*

For example, this will execute different actions when the temperature is under 25 or 25 or more:

```
IF Temp < 25 THEN PRINT "Cold" ELSE PRINT "Hot"
```

The previous examples all used single line IF statements but you can also have multiline IF statements.

作为一个例子，以下代码将使用 SQR() 函数打印直角三角形的斜边长度，该函数返回一个数字的平方根（*a*和 *b*是存储其他边长度的变量）：

打印 `SQR(a * a + b * b)`

MMBasic 将首先通过将 *a*与 *a*相乘，然后将 *b*与 *b*相乘，最后将结果相加来求值此表达式。得到的数字将传递给 `SQR()` 函数，该函数将计算该数字的平方根（即斜边），并将其返回以供 `PRINT` 命令显示。

MMBasic 提供的一些其他数学函数包括：

`SIN(r)` – *r* 的正弦

`COS(r)` – *r* 的余弦

`TAN(r)` – *r* 的正切

还有许多其他函数可供使用，它们都在本手册的前面列出。请注意，在上述三角函数中，传递给函数的值（即 '*r*'）是以弧度表示的角度。在 MMBasic 中，您可以使用函数 `RAD(d)` 将角度从度转换为弧度（'*d*' 是以度为单位的角度）。

大多数编程语言，包括BASIC的另一个特性是你可以将函数调用嵌套在彼此之中。例如，给定角度（即'*d*'）的度数，可以使用以下表达式找到该角度的正弦值：

```
PRINT SIN(RAD(d))
```

在这种情况下，MMBasic将首先获取*d*的值，并使用RAD()函数将其转换为弧度。该函数的输出然后成为SIN()函数的输入。

IF语句

做出决策是大多数计算机程序的核心，而在BASIC中，这通常是通过IF语句来完成的。这几乎像一句英语句子一样书写：

IF 条件 THEN 动作

条件通常是一个比较，例如等于、小于、大于等。

例如：

```
IF Temp < 25 THEN PRINT "冷"
```

*Temp*将是一个变量，保存当前温度（以°C为单位），而 `PRINT "冷"` 是要执行的动作。

你可以进行一系列的测试：

=	等于	<>	不等于
<	小于	<=	小于或等于
>	大于	>=	大于或等于

你还可以添加一个否则子句，当初始条件测试为假时将执行该子句：

如果条件那么真动作否则假动作

例如，当温度低于25或25及以上时，这将执行不同的动作：

如果 `Temp < 25` 那么 打印 "冷" 否则 打印 "热"

前面的示例都使用了单行IF语句，但你也可以有多行IF语句。

They look like this:

```
IF condition THEN  
    true-action  
    true-action  
ENDIF
```

or

```
IF condition THEN  
    true-action  
    true-action  
ELSE  
    false-action  
    false-action  
ENDIF
```

Unlike the single line IF statement you can have many true actions with each on their own line and similarly many false actions. Generally the single line IF statement is handy if you have a simple action that needs to be taken while the multiline version is much easier to understand if the actions are numerous and more complicated.

An example of a multiline IF statement with more than one action is:

```
IF Amount < 100 THEN  
    PRINT "Too low"  
    PRINT "Minimum value is 100"  
ELSE  
    PRINT "Input accepted"  
    SaveToSDCard  
    PRINT "Enter second amount"  
ENDIF
```

Note that in the above example each action is indented to show what part of the IF structure it belongs to. Indenting is not mandatory but it makes a program much easier to understand for someone who is not familiar with it and therefore it is highly recommended.

In a multiline IF statement you can make additional tests using the ELSE IF command. This is best explained by using an example (the temperatures are all in °C):

```
IF Temp < 0 THEN  
    PRINT "Freezing"  
ELSE IF Temp < 20 THEN  
    PRINT "Cold"  
ELSE IF Temp < 35 THEN  
    PRINT "Warm"  
ELSE  
    PRINT "Hot"  
ENDIF
```

The ELSE IF can use the same tests as an ordinary IF (ie, <, <=, etc) but that test will only be made if the preceding test was false. So, for example, you will only get the message *Warm* if *Temp < 0* failed, and *Temp < 20* failed but *Temp < 35* was true. The final ELSE will catch the case where all the tests were false.

An expression like *Temp < 20* is evaluated by MMBasic as either true or false with true having a value of one and false zero. You can see this if you entered the following at the console:

```
PRINT 30 > 20
```

MMBasic will print 1 meaning that the value of the expression is true.

它们看起来像这样：

```
如果条件那么  
    真动作  
    真动作  
结束如果
```

或

```
如果条件那么  
    真动作  
    真动作  
否则  
    假动作  
    假动作  
结束如果
```

与单行IF语句不同，你可以有多个真动作，每个动作在自己的行上，并且同样可以有多个假动作。通常，单行IF语句在你有一个简单的动作需要执行时很方便，而多行版本在动作众多且更复杂时更容易理解。

一个包含多个操作的多行IF语句示例如下：

```
如果 Amount < 100 那么  
    打印 "太低了"  
    打印 "最小值是100"  
否则  
    打印 "输入已接受"  
    保存到SD卡  
    打印 "输入第二个金额"  
结束
```

请注意，在上述示例中，每个操作都有缩进，以显示它属于IF结构的哪个部分。缩进不是强制性的，但对于不熟悉程序的人来说，它使程序更易于理解，因此强烈推荐使用。

在多行IF语句中，您可以使用ELSE IF命令进行额外的测试。这最好通过一个示例来解释（温度均以°C为单位）：

```
如果 Temp < 0 那么  
    打印 "冰冻"  
否则如果 Temp < 20 那么  
    打印 "寒冷"  
否则如果 Temp < 35 那么  
    打印 "温暖"  
否则  
    打印 "热"  
结束
```

ELSE IF可以使用与普通IF相同的测试（即，<，<=等），但该测试仅在前面的测试为假时才会进行。例如，只有在Temp < 0失败，且Temp < 20失败，但Temp < 35为真时，你才会收到消息温暖。最终的否则将捕捉到所有测试都为假的情况。

像 Temp < 20这样的表达式在MMBasic中被求值为真或假，真对应的值为1，假对应的值为0。如果你在控制台输入以下内容，你可以看到这一点：

```
PRINT 30 > 20
```

MMBasic将打印1，意味着该表达式的值为真。

Similarly the following will print 0 meaning that the expression evaluated to false.

```
PRINT 30 < 20
```

The IF statement does not really care about what the condition actually is, it just evaluates the condition and if the result is zero it will take that as false and if non zero it will take it as true. This allows for some handy shortcuts. For example, if BalanceCorrect is a variable that is true (non zero) when some feature of the program is correct then the following can be used to make a decision based on that value:

```
IF BalanceCorrect THEN ...do something...
```

FOR Loops

Another common requirement in programming is repeating a set of actions. For instance, you might want to step through all seven days in the week and perform the same function for each day. BASIC provides the FOR loop construct for this type of job and it works like this:

```
FOR day = 1 TO 7  
    Do something based on the value of 'day'  
NEXT day
```

This starts by creating the variable day and assigning the value of 1 to it. The program will then execute the following statements until it comes to the NEXT statement. This tells the BASIC interpreter to increment the value of day, go back to the previous FOR statement and re-execute the following statements a second time. This will continue looping around until the value of day exceeds 7 and the program will then exit the loop and continue with the statements following the NEXT statement.

As a simple example, you can print the numbers from one to ten like this:

```
FOR nbr = 1 TO 10  
    PRINT nbr,;  
NEXT nbr
```

The comma at the end of the PRINT statement tells the interpreter to tab to the next tab column after printing the number and the semicolon will leave the cursor on this line rather than automatically moving to the next line. As a result, the numbers will be printed in neat columns across the page.

This is what you would see:

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

The FOR loop also has a couple of extra tricks up its sleeves. You can change the amount that the variable is incremented by using the STEP keyword. So, for example, the following will print just the odd numbers:

```
FOR nbr = 1 TO 10 STEP 2  
    PRINT nbr,;  
NEXT nbr
```

The value of the step (or increment value) defaults to one if the STEP keyword is not used but you can set it to whatever number you want.

When MMBasic is incrementing the variable it will check to see if the variable has exceeded the TO value and, if it has, it will exit from the loop. So, in the above example, the value of nbr will reach nine and it will be printed but on the next loop nbr will be eleven and at that point execution will leave the loop. This test is also applied at the start of the loop (ie, if in the beginning the value of the variable exceeds the TO value (and STEP is positive) the loop will never be executed, not even once).

By setting the STEP value to a negative number you can use the FOR loop to step down from a high number to low. In that case the starting number must be greater than the TO number.

类似地，以下内容将打印0，意味着该表达式求值为假。

```
PRINT 30 < 20
```

IF语句实际上并不关心条件是什么，它只是评估条件，如果结果为零，它将视为假；如果非零，它将视为真。这允许一些方便的快捷方式。例如，如果BalanceCorrect是一个在程序某个特性正确时为真（非零）的变量，那么可以使用以下语句根据该值做出决策：

如果 BalanceCorrect 那么 ...做某事 ...

FOR循环

编程中的另一个常见需求是重复一组操作。例如，您可能想要遍历一周的七天，并对每一天执行相同的功能。BASIC 提供了 FOR 循环结构来完成这种工作，其工作方式如下：

```
FOR day = 1 TO 7  
    根据'day'的值做某事  
NEXT day
```

这首先创建了变量 day 并将值 1 赋给它。程序将执行以下语句，直到遇到 NEXT 语句。这告诉 BASIC 解释器增加 day 的值，返回到前一个 FOR 语句，并第二次重新执行以下语句。这将持续循环，直到 day 的值超过 7，然后程序将退出循环并继续执行 NEXT 语句后面的语句。

作为一个简单的例子，你可以像这样打印从1到10的数字：

```
FOR nbr = 1 TO 10  
    PRINT nbr,;  
NEXT nbr
```

PRINT语句末尾的逗号告诉解释器在打印数字后跳到下一个制表符列，而分号将光标留在这一行，而不是自动移动到下一行。因此，数字将整齐地打印在页面的列中。

这就是你会看到的：

1 2 3 4 5 6 7 8 9 10

FOR循环还有一些额外的技巧。你可以使用STEP关键字来改变变量的增量。例如，以下代码将只打印奇数：

```
FOR nbr = 1 TO 10 STEP 2  
    PRINT nbr,;  
NEXT nbr
```

如果未使用STEP关键字，则步长（或增量值）的默认值为1，但您可以将其设置为您想要的任何数字。

当MMBasic增加变量时，它将检查该变量是否超过TO值，如果超过，则将退出循环。因此，在上述示例中，nbr的值将达到九，并将被打印，但在下一个循环中，nbr将是十一，此时执行将离开循环。此测试也在循环开始时应用（即，如果在开始时变量的值超过TO值（且STEP为正），则循环将永远不会执行，甚至一次也不会）。通过将STEP值设置为负数，您可以使用FOR循环从高数字递减到低数字。在这种情况下，起始数字必须大于TO数字。

。

For example, the following will print the numbers from 1 to 10 in reverse:

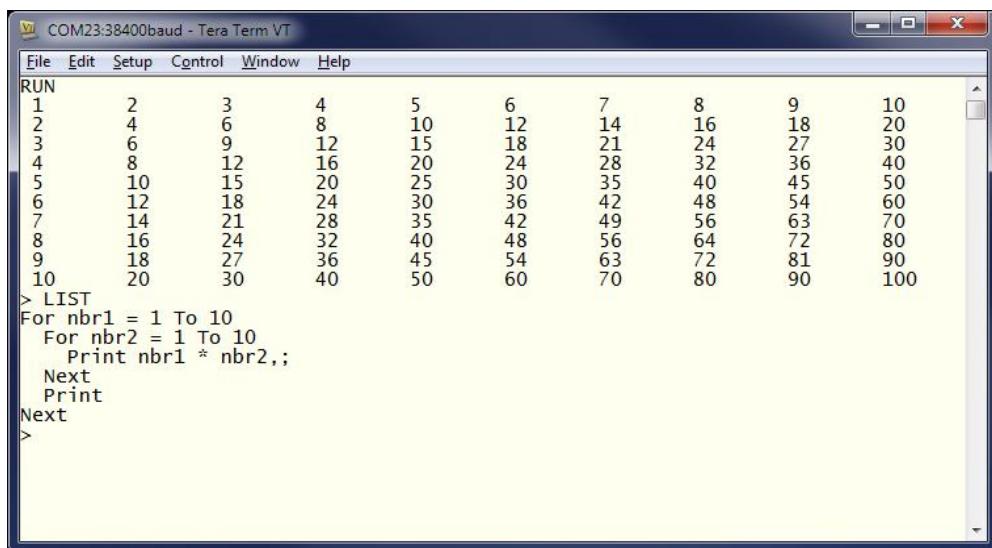
```
FOR nbr = 10 TO 1 STEP -1
    PRINT nbr, ;
NEXT nbr
```

Multiplication Table

To further illustrate how loops work and how useful they can be, the following short program will use two FOR loops to print out the multiplication table that we all learnt at school. The program for this is not complicated:

```
FOR nbr1 = 1 to 10
    FOR nbr2 = 1 to 10
        PRINT nbr1 * nbr2, ;
    NEXT nbr2
    PRINT
NEXT nbr1
```

The output is shown in the following screen grab, which also shows a listing of the program.



The screenshot shows a terminal window titled "COM23:38400baud - Tera Term VT". The window has a menu bar with File, Edit, Setup, Control, Window, Help, and a RUN button. The main area displays the multiplication table from 1 to 10. Below the table, the source code is listed:

```
RUN
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
4   8   12  16  20  24  28  32  36  40
5   10  15  20  25  30  35  40  45  50
6   12  18  24  30  36  42  48  54  60
7   14  21  28  35  42  49  56  63  70
8   16  24  32  40  48  56  64  72  80
9   18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90  100
> LIST
For nbr1 = 1 To 10
    For nbr2 = 1 To 10
        Print nbr1 * nbr2, ;
    Next
    Print
Next
>
```

You need to work through the logic of this example line by line to understand what it is doing. Essentially it consists of one loop inside another. The inner loop, which increments the variable nbr2 prints one horizontal line of the table. When this loop has finished it will execute the following PRINT command which has nothing to print - so it will simply output a new line (ie, terminate the line printed by the inner loop).

The program will then execute another iteration of the outer loop by incrementing nbr1 and re-executing the inner loop again. Finally, when the outer loop is exhausted (when nbr1 exceeds 10) the program will reach the end and terminate.

One last point, you can omit the variable name from the NEXT statement and MMBasic will guess which variable you are referring to. However, it is good practice to include the name to make it easier for someone else who is reading the program to understand it. You can also terminate multiple loops using a comma separated list of variables in the NEXT statement. For example:

```
FOR var1 = 1 TO 5
    FOR var2 = 10 to 13
        PRINT var1 * var2
    NEXT var1, var2
```

例如，以下代码将以逆序打印从1到10的数字：

```
FOR nbr = 10 TO 1 STEP -1
    PRINT nbr, ;
NEXT nbr
```

乘法表

为了进一步说明循环是如何工作的以及它们的实用性，以下短程序将使用两个FOR循环打印出我们在学校学习过的乘法表。这个程序并不复杂：

```
FOR nbr1 = 1 to 10
    FOR nbr2 = 1 to 10
        PRINT nbr1 * nbr2, ;
    NEXT nbr2
    PRINT
NEXT nbr1
```

输出如以下屏幕截图所示，同时也显示了程序的列表。

The screenshot shows a terminal window titled "COM23:38400baud - Tera Term VT". The window has a menu bar with File, Edit, Setup, Control, Window, Help, and a RUN button. The main area displays a multiplication table from 1 to 10. Below the table, the source code is listed:

```
> LIST
For nbr1 = 1 To 10
    For nbr2 = 1 To 10
        Print nbr1 * nbr2, ;
    Next
    Print
Next
```

您需要逐行分析这个示例的逻辑，以理解它的功能。

本质上，它由一个循环嵌套在另一个循环中。内层循环，增量变量nbr2打印出表格的一行。当这个循环结束时，它将执行以下PRINT命令，该命令没有内容可打印，因此它将简单地输出一个新行（即，终止内层循环打印的行）。

程序将通过增量 nbr1 并重新执行内部循环来执行外部循环的另一个迭代。最后，当外部循环结束（当 nbr1 超过 10 时），程序将到达结束并终止。

最后一点，你可以在 NEXT 语句中省略变量名称，MMBasic 将猜测你所指的变量。然而，包含名称是良好的实践，这样可以让其他阅读程序的人更容易理解。你还可以在 NEXT 语句中使用逗号分隔的变量列表来终止多个循环。例如：

```
FOR var1 = 1 TO 5
    FOR var2 = 10 TO 13
        PRINT var1 * var2
    NEXT var1, var2
```

DO Loops

Another method of looping is the DO...LOOP structure which looks like this:

```
DO WHILE condition  
    statement  
    statement  
LOOP
```

This will start by testing the condition and if it is true the statements will be executed until the LOOP command is reached, at which point the condition will be tested again and if it is still true the loop will execute again. The 'condition' is the same as in the IF command (ie, X < Y).

For example, the following will keep printing the word "Hello" on the console for 4 seconds then stop:

```
Timer = 0  
DO WHILE Timer < 4000  
    PRINT "Hello"  
LOOP
```

Note that Timer is a function within MMBasic which will return the time in milliseconds since the timer was reset. A reset is done by assigning zero to Timer (as done above) or when powering up the PicoMite .

A variation on the DO-LOOP structure is the following:

```
DO  
    statement  
    statement  
LOOP UNTIL condition
```

In this arrangement the loop is first executed once, the condition is then tested and if the condition is false, the loop will be repeatedly executed until the condition becomes true. Note that the test in LOOP UNTIL is the inverse of DO WHILE.

For example, similar to the previous example, the following will also print "Hello" for four seconds:

```
Timer = 0  
DO  
    PRINT "Hello"  
LOOP UNTIL Timer >= 4000
```

Both forms of the DO-LOOP essentially do the same thing, so you can use whatever structure fits with the logic that you wish to implement.

Finally, it is possible to have a DO Loop that has no conditions at all - ie,

```
DO  
    statement  
    statement  
LOOP
```

This construct will continue looping forever and you, as the programmer, will need to provide a way to explicitly exit the loop (the EXIT DO command will do this). For example:

```
Timer = 0  
DO  
    PRINT "Hello"  
    IF Timer >= 4000 THEN EXIT DO  
LOOP
```

DO 循环

另一种循环的方法是 DO...LOOP 结构，如下所示：

```
DO WHILE condition  
    statement  
    statement  
LOOP
```

这将首先测试条件，如果条件为真，则语句将被执行，直到达到 LOOP 命令，此时将再次测试条件，如果仍然为真，则循环将再次执行。‘条件’与 IF 命令中的相同（即， $x < y$ ）。

例如，以下代码将在控制台上持续打印单词“你好”，持续4秒钟，然后停止：

```
Timer = 0  
DO WHILE Timer < 4000  
    PRINT "你好"  
LOOP
```

请注意，Timer 是 MMBasic 中的一个函数，它将返回自定时器重置以来的毫秒数。重置可以通过将零赋值给 Timer（如上所示）或在启动 PicoMite 时完成。

DO-LOOP 结构的一个变体如下：

```
DO  
    语句  
    语句  
LOOP UNTIL 条件
```

在这种安排中，循环首先执行一次，然后测试条件，如果条件为假，则循环将重复执行，直到条件变为真。请注意，LOOP UNTIL 中的测试是 DO WHILE 的反向。

例如，类似于前面的例子，以下代码也将在四秒内打印“你好”：

```
定时器 = 0  
DO  
    打印 "你好"  
    循环直到 定时器 >= 4000
```

这两种形式的 DO 循环本质上做的是相同的，因此您可以使用适合您希望实现的逻辑的任何结构。

最后，可以有一个完全没有条件的 DO 循环，即，

```
DO  
    语句  
    语句  
循环
```

这个结构将会无限循环，作为程序员的您需要提供一种明确退出循环的方法（EXIT DO 命令可以做到这一点）。例如：

```
定时器 = 0  
DO  
    打印 "你好"  
    如果 定时器 >= 4000 那么 退出 DO  
循环
```

Console Input

As well as printing data for the user to see your programs will also want to get input from the user. For that to work you need to capture keystrokes from the console and this can be done with the INPUT command. In its simplest form the command is:

```
INPUT var
```

This command will print a question mark on the console's screen and wait for a number to be entered followed by the Enter key. That number will then be assigned to the variable var.

For example, the following program extends the expression for finding the hypotenuse of a triangle by allowing the user to enter the lengths of the other sides from the console.

```
PRINT "Length of side 1"  
INPUT a  
PRINT "Length of side 2"  
INPUT b  
PRINT "Length of the hypotenuse is" SQR(a * a + b * b)
```

This is a screen capture of a typical session:

```
COM3:115200baud - Tera Term VT  
File Edit Setup Control Window Help  
> RUN  
Length of side 1  
? 12  
Length of side 2  
? 15  
Length of the hypotenuse is 19.20937271  
>
```

The INPUT command can also print your prompt for you, so that you do not need a separate PRINT command. For example, this will work the same as the above program:

```
INPUT "Length of side 1"; a  
INPUT "Length of side 2"; b  
PRINT "Length of the hypotenuse is" SQR(a * a + b * b)
```

Finally, the INPUT command will allow you to input a series of numbers separated by commas with each number being saved in different variables.

For example:

```
INPUT "Enter the length of the two sides: ", a, b  
PRINT "Length of the hypotenuse is" SQR(a * a + b * b)
```

If the user entered 12,15 the number 12 would be saved in the variable a and 15 in b.

Another method of getting input from the console is the LINE INPUT command. This will get the whole line as typed by the user and allocate it to a string variable. Like the INPUT command you can also specify a prompt. This is a simple example:

```
LINE INPUT "What is your name? ", s$  
PRINT "Hello " s$
```

We will cover string variables later in this tutorial but for the moment you can think of them as a variable that holds a sequence of characters. If you ran the above program and typed in John when prompted the program would respond with Hello John.

控制台输入

除了打印数据供用户查看，您的程序还希望从用户那里获取输入。

为了使其正常工作，您需要从控制台捕获按键，这可以通过INPUT命令来完成。在其最简单的形式中，该命令是：

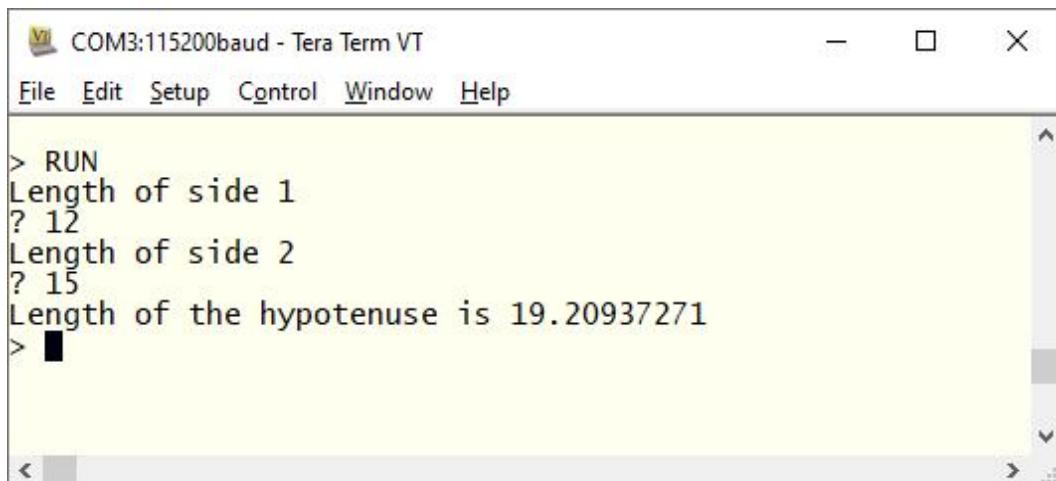
```
输入 var
```

此命令将在控制台屏幕上打印一个问号，并等待输入一个数字，随后按下回车键。该数字将被赋值给变量 var。

例如，以下程序扩展了计算三角形斜边的表达式，允许用户从控制台输入其他边的长度。

```
PRINT "边长 1"  
INPUT a  
PRINT "边长 2"  
INPUT b  
PRINT "斜边的长度是" SQR(a * a + b * b)
```

这是一个典型会话的屏幕截图：



INPUT 命令也可以为您打印提示，因此您不需要单独的 PRINT 命令。例如，这将与上述程序的效果相同：

```
INPUT "边长 1"; a  
INPUT "边长 2"; b  
PRINT "斜边的长度是" SQR(a * a + b * b)
```

最后，INPUT 命令将允许您输入一系列用逗号分隔的数字，每个数字将被保存到不同的变量中。

例如：

输入 "输入两个边的长度：", a, b 打印 "斜边的长度是" SQR(a * a + b * b) 如果用户输入了 12,15 数字 12 将被保存到变量 a 中，而 15 将被保存到 b 中。从控制台获取输入的另一种方法是使用 LINE INPUT 命令。这将获取用户输入的整行并将其分配给一个字符串变量。与 INPUT 命令一样，您也可以指定一个提示。这是一个简单的例子：

```
行输入 "你的名字是什么？", s$  
打印 "你好" s$
```

我们将在本教程后面讨论字符串变量，但目前您可以将它们视为一个保存字符序列的变量。如果您运行上述程序并在提示时输入约翰，程序将响应你好约翰。

Sometimes you do not want to wait for the user to hit the enter key, you want to get each character as it is typed in. This can be done with the INKEY\$ function which will return the value of the character as a string consisting of just one character or an empty string (ie, contains no characters) if nothing has been entered.

GOTO and Labels

One method of controlling the flow of the program is the GOTO command. This essentially tells MMBasic to jump to another part of the program and continue executing from there. The target of the GOTO is a label and this needs to be explained first.

A label is an identifier that marks part of the program. It must be the first thing on the line and it must be terminated with the colon (:) character. The name that you use can be up to 31 characters long and must follow the same rules as for a variable's name. For example, in the following program line LoopBack is a label:

```
LoopBack: a = a + 1
```

When you use the GOTO command to jump to that particular part of the program you would use the command like this:

```
GOTO LoopBack
```

To put all this into context the following program will print out all the numbers from 1 to 10:

```
z = 0
LoopBack: z = z + 1
PRINT z
IF z < 10 THEN GOTO LoopBack
```

The program starts by setting the variable z to zero then incrementing it to 1 in the next line. The value of z is printed and then tested to see if it is less than 10. If it is less than 10 the program execution will jump back to the label LoopBack where the process will repeat. Eventually the value of z will be more than 10 and the program will run off the end and terminate.

Note that a FOR loop can do the same thing (and is simpler) so this example is purely designed to illustrate what the GOTO command can do.

In the past the GOTO command gained a bad reputation. This is because using GOTOS it is possible to create a program that continuously jumps from one point to another (often referred to as "spaghetti code") and that type of program is almost impossible for another programmer to understand. With constructs like the multiline IF statements the need for the GOTO statement has been reduced and it should be used only when there is no other way of changing the program's flow.

Testing for Prime Numbers

The following is a simple program which brings together many of the programming features previously discussed.

```
DO
    InpErr:
    PRINT
    INPUT "Enter a number: "; a
    IF a < 2 THEN
        PRINT "Number must be equal or greater than 2"
        GOTO InpErr
    ENDIF

    Divs = 0
    FOR x = 2 TO SQR(a)
        r = a/x
```

有时您不想等待用户按下回车键，而是希望在每个字符输入时立即获取。这可以通过INKEY\$函数来实现，该函数将返回一个字符串，包含一个字符的值，或者如果没有输入，则返回一个空字符串（即，不包含任何字符）。

转到和标签

控制程序流程的一种方法是使用转到命令。这基本上告诉MMBasic 跳转到程序的另一部分并从那里继续执行。转到的目标是一个标签，这需要先进行解释。

标签是标记程序某部分的标识符。它必须是行中的第一个内容，并且必须以冒号 (:) 字符结束。您使用的名称可以长达 31 个字符，并且必须遵循与变量名称相同的规则。例如，在以下程序行中LoopBack 是一个标签：

```
LoopBack: a = a + 1
```

当您使用转到命令跳转到程序的特定部分时，您可以这样使用命令：

```
转到 LoopBack
```

为了将所有这些放入上下文，以下程序将打印出从 1 到 10 的所有数字：

```
z = 0
LoopBack: z = z + 1
打印 z
如果 z < 10 那么 转到 LoopBack
```

程序开始时将变量 z 设置为零，然后在下一行将其增量设置为1。变量 z 的值被打印出来，然后测试其是否小于10。如果小于10，程序执行将跳回标签 LoopBack，过程将重复。最终，变量 z 的值将超过10，程序将运行到结束并终止。

请注意，FOR循环可以做同样的事情（并且更简单），因此这个例子纯粹是为了说明GOTO命令可以做什么。

在过去，GOTO命令获得了不好的声誉。这是因为使用GOTO可以创建一个程序，该程序不断从一个点跳到另一个点（通常称为“意大利面条代码”），这种类型的程序几乎不可能让其他程序员理解。通过像多行 IF 语句这样的结构，GOTO 语句的需求减少了，应该仅在没有其他方法改变程序流程时使用。

测试质数

以下是一个简单的程序，它汇集了之前讨论的许多编程特性。

```
DO
InpErr:
打印
输入 "请输入一个数字: "; a
如果 a < 2 则
    打印 "数字必须大于或等于 2"
    转到 InpErr
结束如果

Divs = 0
对于 x = 2 到 SQR(a)
    r = a/x
```

```

    IF r = FIX(r) THEN Divs = Divs + 1
NEXT x

PRINT a " is ";
IF Divs > 0 THEN PRINT "not ";
PRINT "a prime number."
LOOP

```

This will first prompt (on the console) for a number and, when it has been entered, it will test if that number is a prime number or not and display a suitable message.

It starts with a DO Loop that does not have a condition – so it will continue looping forever. This is what we want. It means that when the user has entered a number, it will report if it is a prime number or not and then loop around and ask for another number. The way that the user can exit the program (if they wanted to) is by typing the break character (normally CTRL-C).

The program then prints a prompt for the user which is terminated with a semicolon character. This means that the cursor is left at the end of the prompt for the INPUT command which will get the number and store it in the variable a.

Following this the number is tested. If it is less than 2 an error message will be printed and the program will jump backwards and ask for the number again.

We are now ready to test if the number is a prime number. The program uses a FOR loop to step through the possible divisors testing if each one can divide evenly into the entered number. Each time it does the program will increment the variable Divs.

Note that the test is done with the function FIX(r) which simply strips off any digits after the decimal point. So, the condition `r = FIX(r)` will be true if `r` is an integer (ie, has no digits after the decimal point).

Finally, the program will construct the message for the user. The key part is that if the variable Divs is greater than zero it means that one or more numbers were found that could divide evenly into the test number. In that case the IF statement inserts the word "not" into the output message.

For example, if the entered number was 21 the user will see this response:

```
21 is not a prime number.
```

This is the result of running the program and some of the output:

```

> RUN
Enter a number: ? 64
64 is not a prime number.

Enter a number: ? 67
67 is a prime number.

Enter a number: ? 0
Number must be equal or greater than 2

Enter a number: ? █

```

You can test this program by using the editor (the EDIT command) to enter it.

```
如果 r = FIX(r) 则 Divs = Divs + 1  
下一个 x  
  
打印 a " 是 " ;  
如果 Divs > 0 则 打印 " 不是 " ;  
打印 "一个质数。"  
循环
```

这将首先在控制台上提示输入一个数字，当输入完成后，它将测试该数字是否为质数，并显示适当的消息。

它以一个没有条件的DO循环开始——因此它将永远循环下去。这正是我们想要的。这意味着当用户输入一个数字时，它将报告该数字是否为质数，然后循环并询问另一个数字。用户可以通过输入中断字符（通常是CTRL-C）来退出程序（如果他们想的话）。

程序随后打印一个提示，提示以分号字符结束。这意味着光标停留在INPUT命令的提示末尾，该命令将获取数字并将其存储在变量a中。

接下来对数字进行测试。如果它小于2，将打印错误消息，程序将回退并再次询问数字。

我们现在准备测试该数字是否为质数。程序使用FOR循环逐步检查可能的除数，测试每个除数是否能整除输入的数字。每次程序都会增加变量 Divs 的值。

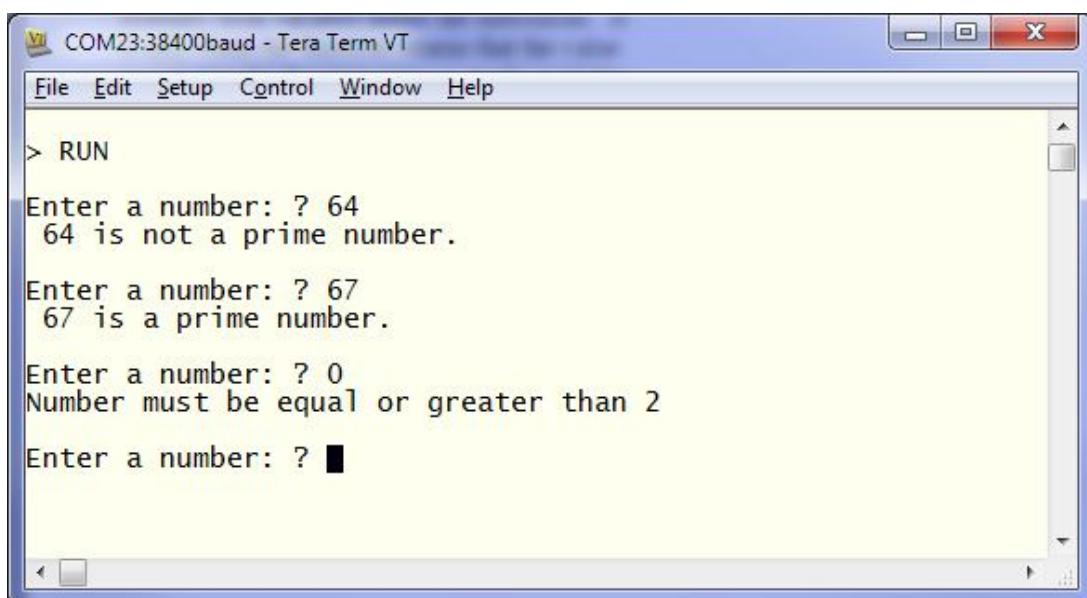
请注意，测试是通过函数 FIX(r) 完成的，该函数简单地去掉小数点后的任何数字。因此，条件 $r = \text{FIX}(r)$ 在 r 是整数（即小数点后没有数字）时为真。

最后，程序将为用户构建消息。关键部分是，如果变量 Divs 大于零，则意味着找到了一个或多个可以整除测试数字的数字。在这种情况下，IF 语句会将单词 "不" 插入输出消息中。

例如，如果输入的数字是 21，用户将看到以下响应：

21 不是质数。

这是运行程序的结果以及一些输出：



The screenshot shows a window titled 'COM23:38400baud - Tera Term VT'. The menu bar includes File, Edit, Setup, Control, Window, and Help. The main window displays the following text:

```
> RUN  
Enter a number: ? 64  
64 is not a prime number.  
Enter a number: ? 67  
67 is a prime number.  
Enter a number: ? 0  
Number must be equal or greater than 2  
Enter a number: ? ■
```

您可以使用编辑器（EDIT 命令）输入该程序来进行测试。

Using your newly learnt skills you could then have a shot at making it more efficient. For example, because the program counts how many times a number can be divided into the test number it takes a lot longer than it should to detect a non prime number. The program would run much more efficiently if it jumped out of the FOR loop at the first number that divided evenly. You could use the GOTO command to do this or you could use the command EXIT FOR – that would cause the FOR loop to terminate immediately.

Other efficiencies include only testing the division with odd numbers (by using an initial test for an even number then starting the FOR loop at 3 and using STEP 2) or by only using prime numbers for the test (that would be much more complicated).

Arrays

Arrays are something which you will probably not think of as useful at first glance but when you do need to use them you will find them very handy indeed.

An array is best thought of as a row of letterboxes for a block of units or condos as shown on the right. The letterboxes are all located at the same address and each box represents a unit or condo at that address. You can place a letter in the box for unit one, or unit two, etc.

Similarly an array in BASIC is a single variable with multiple sub units (called *elements* in BASIC) which are numbered. You can place data in element one, or element two, etc. In BASIC an array is created by the DIM command, for example:

```
DIM numarr(300)
```

This creates an array with the name of numarr containing 301 elements (think of them as letterboxes) ranging from 0 to 300. By default an array will start from zero so this is why there is an extra element making the total 301. To specify a specific element in the array (ie, a specific letterbox) you use an index which is simply the number of the array element that you wish to access. For example, if you want to set element number 100 in this array to (say) the number 876, you would do it this way:

```
numarr(100) = 876
```

Normally the index to an array is not a constant number as in this example (ie, 100) but a variable which can be changed to access different array elements.

As an example of how you might use an array, consider the case where you would like to record the maximum temperature for each day of the year and, at the end of the year, calculate the overall average. You could use ordinary variables to record the temperature for each day but you would need 365 of them and that would make your program unwieldy indeed. Instead, you could define an array to hold the values like this:

```
DIM days(365)
```

Every day you would need to save the temperature in the correct location in the array. If the number of the day in the year was held in the variable doy and the maximum temperature was held in the variable maxtemp you would save the reading like this:

```
days(doy) = maxtemp
```

At the end of the year it would be simple to calculate the average for the year:

```
total = 0
FOR i = 1 to 365
    total = total + days(i)
NEXT i
PRINT "Average is:" total / 365
```



利用您新学到的技能，您可以尝试使其更高效。例如，由于程序计算一个数字可以被测试数字整除的次数，因此检测非质数所需的时间比应该的要长得多。如果程序在第一个能整除的数字处跳出FOR循环，它的运行效率会高得多。你可以使用GOTO命令来实现这一点，或者你可以使用命令EXIT FOR——这将导致FOR循环立即终止。

其他的效率包括仅使用奇数进行除法测试（通过对偶数进行初步测试，然后从3开始FOR循环并使用STEP 2）或仅使用质数进行测试（这会复杂得多）。

数组

数组在初看时可能不会被认为是有用的，但当你确实需要使用它们时，你会发现它们非常方便。

数组最好被视为一排信箱，用于存放一组单位或公寓，如右侧所示。所有的信箱都位于同一个地址，每个信箱代表该地址的一个单元或公寓。您可以将信件放入单元一的信箱，或单元二的信箱，等等。

类似地，在BASIC中，数组是一个包含多个子单元（在BASIC中称为元素）的单一变量，这些子单元是有编号的。您可以将数据放入元素一，或元素二，等等。在BASIC中，数组是通过DIM命令创建的，例如：

```
DIM numarr(300)
```

这创建了一个名为 numarr 的数组，包含 301 个元素（可以将它们视为信箱），范围从 0 到 300。默认情况下，数组从零开始，因此会有一个额外的元素，使总计为 301。要指定数组中的特定元素（即特定的信箱），您使用一个索引，该索引就是您希望访问的数组元素的编号。例如，如果您想将这个数组中的元素编号 100 设置为（比如）数字 876，您可以这样做：

```
numarr(100) = 876
```

通常，数组的索引不是一个常量数字，如这个例子中的 100，而是一个可以更改以访问不同数组元素的变量。

作为您可能如何使用数组的一个例子，考虑一下您想记录每一天的最高温度，并在年末计算整体平均值的情况。您可以使用普通变量来记录每一天的温度，但您需要 365 个变量，这确实会使您的程序变得笨重。相反，您可以定义一个数组来保存这些值，如下所示：

```
DIM days(365)
```

每天您都需要在数组的正确位置保存温度。如果年份中的天数保存在变量 doy 中，而最高温度保存在变量 maxtemp 中，您可以这样保存读数：

```
days(doy) = maxtemp
```

在年末，计算一年的平均值将会很简单：

```
total = 0
FOR i = 1 to 365
    total = total + days(i)
NEXT i
打印 "平均值是:" total / 365
```



This is much easier than adding up and averaging 365 individual variables.

The above array was single dimensioned but you can have multiple dimensions. Reverting to our analogy of letterboxes, an array with two dimensions could be thought of as a block of flats with multiple floors. A block could have a row of four letter boxes for level one, another row of four boxes for level two, and so on. To place a letter in a letterbox you need to specify the floor number and the unit number on that floor.

In BASIC such an array is specified using two indices separated by a comma. For example:

```
LetterBox(floor, unit)
```



As a practical example, assume that you needed to record the maximum temperature for each day over five years. To do this you could dimension the array as follows:

```
DIM days(365, 5)
```

The first index is the day in the year and the second is a number representing the year. If you wanted to set day 100 in year 3 to 24 degrees you would do it like this:

```
days(100, 3) = 24
```

In MMBasic for the PicoMite you can have up to five dimensions (this is different from some other versions of MMBasic which support eight dimensions). The maximum size of an array is only limited by the amount of free RAM that is available.

Integers

So far all the numbers and variables that we have been using have been floating point. As explained before, floating point is handy because it will track digits after the decimal point and when you use division it will return a sensible result. So, if you just want to get things done and are not concerned with the details you should stick to floating point.

However, the limitation of floating point is that it stores numbers as an approximation with an accuracy of 14 digits on the PicoMite. Most times this characteristic of floating point numbers is not a problem but there are some cases where you need to accurately store larger numbers.

As an example, let us say that you want to manipulate time accurately down to the microsecond so that you can compare two different date/times to work out which one is earlier. The easy way to do this is to convert the date/time to the number of microseconds since some date (say 1st Jan in year zero) - then finding the earliest of the two is just a matter of using an arithmetic compare in an IF statement.

The problem is that the number of microseconds since that date will exceed the accuracy range of floating point variables and this is where integer variables come in. An integer variable can accurately hold very large numbers up to nine million million million (or ± 9223372036854775807 to be precise).

The downside of using an integer is that it cannot store fractions (ie, numbers after the decimal point). Any calculation that produces a fractional result will be rounded up or down to the nearest whole number when assigned to an integer. However this characteristic can be handy when dealing with money – for example, you don't want to send someone a bill for \$100.133333333.

It is easy to create an integer variable, just add the percent symbol (%) as a suffix to a variable name. For example, sec% is an integer variable. Within a program you can mix integers and floating point and MMBasic will make the necessary conversions but if you want to maintain the full accuracy of integers you should avoid mixing the two.

Just like floating point you can have arrays of integers with up to five dimensions, all you need to do is add the percent character as a suffix to the array name. For example: days%(365, 5).

这比将365个单独变量相加并求平均要简单得多。

上述数组是单维的，但您可以拥有多维数组。回到我们关于信箱的类比，一个二维数组可以被视为一个有多层的公寓楼。一个楼层可以有四个信箱的行，另一个楼层可以有四个信箱的行，依此类推。要将信件放入信箱，您需要指定楼层号和该楼层上的单元号。



在BASIC中，这样的数组使用两个用逗号分隔的索引来指定。例如：

```
LetterBox(floor, unit)
```

作为一个实际的例子，假设你需要记录五年内每一天的最高温度。为此，你可以按如下方式定义数组：

```
DIM days(365, 5)
```

第一个索引是年份中的天数，第二个是表示年份的数字。如果你想将第三年中的第100天设置为24度，你可以这样做：

```
days(100, 3) = 24
```

在PicoMite的MMBasic中，你最多可以使用五个维度（这与某些支持八个维度的MMBasic版本不同）。数组的最大大小仅受可用自由RAM的限制。

整数

到目前为止，我们使用的所有数字和变量都是浮点数。如前所述，浮点数很方便，因为它会跟踪小数点后的数字，当你进行除法时，它会返回一个合理的结果。所以，如果你只是想完成事情而不关心细节，你应该坚持使用浮点数。

然而，浮点数的限制在于它将数字存储为近似值，精度为14位数字，在PicoMite上。大多数情况下，浮点数的这一特性并不是问题，但在某些情况下，你需要准确存储较大的数字。

举个例子，假设你想准确地处理时间到微秒，以便比较两个不同的日期/时间，找出哪个更早。简单的方法是将日期/时间转换为自某个日期（例如公元零年1月1日）以来的微秒数——然后找出两个中的最早者只需在IF语句中使用算术比较。

问题是，自该日期以来的微秒数将超过浮点变量的精度范围，这就是整数变量派上用场的地方。整数变量可以准确地存储非常大的数字，最高可达九百万亿亿（或 ± 9223372036854775807 ，确切地说）。

使用整数的缺点是它无法存储小数（即小数点后的数字）。

任何产生分数结果的计算在赋值给整数时都会四舍五入到最接近的整数。然而，这一特性在处理金钱时可能会很方便——例如，你不想给某人发送一张100.1美元的账单。333333333创建一个整数变量很简单，只需在变量名称后添加百分号（%）作为后缀。

例如，sec%是一个整数变量。在程序中，你可以混合使用整数和浮点数，MMBasic会进行必要的转换，但如果你想保持整数的完整精度，应该避免混合这两者。

就像浮点数一样，你可以拥有最多五个维度的整数数组，你所需要做的就是在数组名称后添加百分号作为后缀。例如：days%(365, 5)。

Beginners often get confused as to when they should use floating point or integers and the answer is simple... always use floating point unless you need an extremely high level of accuracy in the resulting number. This does not happen often but when you need them you will find that integers are a lifesaver.

Strings

Strings are another variable type (like floating point and integers). Strings are used to hold a sequence of characters. For example, in the command:

```
PRINT "Hello"
```

The string "Hello" is a string constant. Note that a constant is something that does not change (as against a variable, which can) and that string constants are always surrounded by double quotes.

String variables names use the dollar symbol (\$) as a suffix to identify them as a string instead of a normal floating point variable and you can use ordinary assignment to set their value. The following are examples (note that the second example uses an array of strings):

```
Car$ = "Holden"  
Country$(12) = "India"  
Name$ = "Fred"
```

You can also join strings using the plus operator:

```
Word1$ = "Hello"  
Word2$ = "World"  
Greeting$ = Word1$ + " " + Word2$
```

In which case the value of Greeting\$ will be "Hello World".

Strings can also be compared using operators such as = (equals), <> (not equals), < (less than), etc. For example:

```
IF Car$ = "Holden" THEN PRINT "Was an Aussie made car"
```

The comparison is made using the full ASCII character set so a space will come before a printable character. Also the comparison is case sensitive so 'holden' will not equal "Holden". Using the function UCASE() to convert the string to upper case you can then have a case insensitive comparison. For example:

```
IF UCASE$(Car$) = "HOLDEN" THEN PRINT "Was an Aussie made car"
```

You can have arrays of strings but you need to be careful when you declare them as you can rapidly run out of RAM (general memory used for storing variables, etc). This is because MMBasic will by default allocate 255 bytes of RAM for each element of the array. For example, a string array with 100 elements will by default use 25K of RAM. To alleviate this you can use the LENGTH qualifier to limit the maximum size of each element. For instance, if you know that the maximum length of any string that will be stored in the array will be less than 20 characters you can use the following declaration to allocate just 20 bytes for each element:

```
DIM MyArray$(100) LENGTH 20
```

The resultant array will only use 2K of RAM.

Manipulating Strings

String handling is one of MMBasic's strengths and using a few simple functions you can pull apart and generally manipulate strings. The basic string functions are:

LEFT\$(string\$, nbr) Returns a substring of *string\$* with *nbr* of characters from the left (beginning) of the string.

RIGHT\$(string\$, nbr) Same as the above but return *nbr* of characters from the right (end) of the string.

初学者常常会困惑于何时使用浮点数或整数，答案很简单……除非你需要极高的结果精度，否则总是使用浮点数。这种情况并不常见，但当你需要它们时，你会发现整数是救命稻草。

字符串

字符串是另一种变量类型（如浮点数和整数）。字符串用于保存一系列字符。例如，在命令中：

```
PRINT "Hello"
```

字符串 "Hello" 是一个字符串常量。请注意，常量是指不会改变的东西（与可以改变的变量相对），并且字符串常量总是被双引号包围。字符串变量名称使用美元符号 (\$) 作为后缀，以将其标识为字符串，而不是普通的浮点变量，你可以使用普通的赋值来设置它们的值。以下是示例（请注意第二个示例使用了字符串数组）：

```
Car$ = "Holden"  
Country$(12) = "印度"  
Name$ = "弗雷德"
```

您还可以使用加号运算符连接字符串：

```
Word1$ = "你好"  
Word2$ = "世界"  
Greeting$ = Word1$ + " " + Word2$
```

在这种情况下，Greeting\$的值将是 "你好 世界"。

字符串还可以使用运算符进行比较，例如 = (等于)、<> (不等于)、< (小于) 等。例如：

如果 Car\$ = "Holden" 那么 打印 "是一辆澳大利亚制造的汽车"

比较是使用完整的 ASCII 字符集进行的，因此空格会在可打印字符之前。此外，比较是区分大小写的，因此 'holden' 不等于 "Holden"。使用函数 UCASE() 将字符串转换为大写，您可以进行不区分大小写的比较。例如：

如果 UCASE\$(Car\$) = "HOLDEN" 那么 打印 "是一辆澳大利亚制造的汽车"

你可以使用字符串数组，但在声明时需要小心，因为你可能会迅速耗尽RAM（用于存储变量等的一般内存）。这是因为MMBasic默认为数组的每个元素分配255字节的RAM。例如，一个包含100个元素的字符串数组默认将使用25K的RAM。为了缓解这个问题，你可以使用 LENGTH限定符来限制每个元素的最大大小。例如，如果你知道存储在数组中的任何字符串的最大长度将少于20个字符，你可以使用以下声明为每个元素分配仅20字节：

```
DIM MyArray$(100) LENGTH 20
```

结果数组将仅使用2K的RAM。

操作字符串

字符串处理是MMBasic的一个强项，通过使用一些简单的函数，你可以拆分和一般性地操作字符串。基本字符串函数包括：

LEFT\$(string\$, nbr) 返回一个子字符串 string\$，包含从字符串左侧（开头）开始的nbr个字符。

RIGHT\$(string\$, nbr) 与上述相同，但返回从字符串右侧（末尾）开始的nbr个字符。

MID\$(string\$, pos, nbr) Returns a substring of *string\$* with *nbr* of characters starting from the character *pos* in the string (ie, the middle of the string).

For example if *S\$* = "This is a string"

then: *R\$* = LEFT\$(*S\$*, 7) would result in the value of *R\$* being set to: "This is"
and: *R\$* = RIGHT\$(*S\$*, 8) would result in the value of *R\$* being set to: "a string"
finally: *R\$* = MID\$(*S\$*, 6, 2) would result in the value of *R\$* being set to: "is"

Note that in MID\$() the first character position in a string is number 1, the second is number 2 and so on. So, counting the first character as one, the sixth position is the start of the word "is".

Another useful function is:

INSTR(string\$, pattern\$) Returns a number representing the position at which *pattern\$* occurs in *string\$*.

This can be used to search for a string inside another string. The number returned is the position of the substring inside the main string. Like with MID\$() the start of the string is position 1.

For example if *S\$* = "This is a string"

Then: *pos* = INSTR(*S\$*, " ")
would result in *pos* being set to the position of the first space in *S\$* (ie, 5).

INSTR() can be combined with other functions so this would return the first **word** in *S\$*:

R\$ = LEFT\$(*S\$*, INSTR(*S\$*, " ") - 1)

There is also an extended version of INSTR():

INSTR(pos, string\$, pattern\$) Returns a number representing the position at which *pattern\$* occurs in *string\$* when starting the search at the character position *pos*.

So we can find the second word in *S\$* using the following:

```
pos = INSTR(S$, " ")
R$ = LEFT$(S$, INSTR(pos + 1, S$, " ") - 1)
```

This last example is rather complicated so it might be worth working through it in detail so that you can understand how it works.

Note that INSTR() will return the number zero if the sub string is not found and that any string function will throw an error (and halt the program) if that is used as a character position. So, in a practical program you would first check for zero being returned by INSTR() before using that value.

For example:

```
pos = INSTR(S$, " ")
if pos > 0 THEN R$ = LEFT$(S$, INSTR(pos + 1, S$, " ") - 1)
```

Scientific Notation

Before we finish discussing data types we need to cover off the subject of floating point numbers and scientific notation.

Most numbers can be written normally, for example 11 or 24.5, but very large or small numbers are more difficult. For example, it has been estimated that the number of grains of sand on planet Earth is 750000000000000000000000. The problem with this number is that you can easily lose track of how many zeros there are in the number and consequently it is difficult to compare this with a similar sized number.

A scientist would write this number as 7.5×10^{18} which is called scientific notation and is much easier to comprehend.

MMBasic will automatically shift to scientific notation when dealing with very large or small floating point numbers. For example, if the above number was stored in a floating point variable the PRINT

MID\$(string\$, pos, nbr) 返回一个子字符串 *string\$*, 包含从字符串中字符位置pos开始的nbr个字符 (即字符串的中间部分)。

例如，如果 `$$ = "这是一个字符串 "`

那么： $R\$ = LEFT\$(S\$, 7)$ 将导致 $R\$$ 的值被设置为："这是"， 并且： $R\$ = RIGHT\$(S\$, 8)$ 将导致 $R\$$ 的值被设置为："一个字符串"， 最后： $R\$ = MID\$(S\$, 6, 2)$ 将导致 $R\$$ 的值被设置为："是"。请注意，在 $MID\$(\cdot)$ 中，字符串中的第一个字符

位置是编号1，第二个是编号2，依此类推。因此，从第一个字符开始计数，第六个位置是单词"是"的开始。

另一个有用的函数是：

INSTR(string\$, pattern\$) 返回一个数字，表示 *pattern\$* 在 *string\$* 中出现的位置。

这可以用来在一个字符串中搜索另一个字符串。返回的数字是子字符串在主字符串中的位置。与MID\$()一样，字符串的起始位置是1。

例如，如果 `S$ = "这是一串字符串"`

那么： pos = INSTR(S\$, " ")
将导致 pos 被设置为 S\$ 中第一个空格的位置（即 5）。

`INSTR()`可以与其他函数结合使用，因此这将返回`SS`中的第一个字：

R\$ = LEFT\$(SS, INSTR(SS, " ") - 1)

`INSTR()`还有一个扩展版本：

INSTR(pos, string\$, pattern\$) 返回一个数字，表示在字符位置pos开始搜索时， pattern\$在string\$中出现的位置。

因此我们可以使用以下方法找到 `SS` 中的第二个字。

```
pos = INSTR(S$, " ")
R$ = LEFT$(S$, INSTR(pos + 1, S$, " ") - 1)
```

这个最后的例子相当复杂，因此值得详细研究，以便你能理解它是如何工作的。

请注意，如果未找到子字符串，`INSTR()` 将返回零，并且如果将其用作字符位置，任何字符串函数都会抛出错误（并停止程序）。因此，在实际程序中，你应该首先检查 `INSTR()` 是否返回零，然后再使用该值。

例如：

```
pos = INSTR(S$, " ")  
如果 pos > 0 那么 R$ = LEFT$(S$, INSTR(pos + 1, S$, " ") - 1)
```

科学记数法

在我们结束讨论数据类型之前，我们需要涵盖浮点数和科学记数法的主题。

大多数数字可以正常书写，例如 11 或 24.5，但非常大或非常小的数字则更难处理。例如，估计地球上的沙粒数量为 750000000000000000000000. 这个数字的问题在于，你很容易失去对数字中有多少个零的追踪，因此很难将其与类似大小的数字进行比较。

科学家会将这个数字写作 7.5×10^{18} ，这被称为科学记数法，理解起来要容易得多。

当处理非常大或非常小的浮点数时，MMBasic会自动转换为科学记数法。例如，如果上述数字存储在一个浮点变量中，PRINT

command would display it as 7.5E+18 (this is BASIC's way of representing 7.5×10^{18}). As another example, the number 0.000000456 would display as 4.56E-8 which is the same as 4.56×10^{-8} .

You can also use scientific notation when entering constant numbers in MMBasic. For example:

```
SandGrains = 7.5E+18
```

MMBasic only uses scientific notation for displaying floating point numbers (not integers). For instance, if you assigned the number of grains of sand to an integer variable it would print out as a normal number (with lots of zeros).

DIM Command

We have used the DIM command before for defining arrays but it can also be used to create ordinary variables. For example, you can simultaneously create four string variables like this:

```
DIM STRING Car, Name, Street, City
```

Note that because these variables have been defined as strings using the DIM command we do not need the \$ suffix, the definition alone is enough for MMBasic to identify their type. Similarly, when you use these variables in an expression you do not need the type suffix: For example:

```
City = "Sydney"
```

You can also use the keyword INTEGER to define a number of integer variables and FLOAT to do the same for floating point variables. This type of notation can similarly be used to define arrays.

For example:

```
DIM INTEGER seconds(200)
```

Another method of defining the variables type is to use the keyword AS. For example:

```
DIM Car AS STRING, Name AS STRING, Street AS STRING
```

This is the method used by Microsoft (MMBasic tries to maintain Microsoft compatibility) and it is useful if the variables have different types. For example:

```
DIM Car AS STRING, Age AS INTEGER, Value AS FLOAT
```

You can use any of these methods of defining a variable's type, they all act the same.

The advantage of defining variables using the DIM command is that they are clearly defined (preferably at the start of the program) and their type (float, integer or string) is not subject to misinterpretation.

You can strengthen this by using the following commands at the very top of your program:

```
OPTION EXPLICIT  
OPTION DEFAULT NONE
```

The first specifies to MMBasic that all variables must be explicitly defined using DIM before they can be used. The second specifies that the type of all variables must be specified when they are created.

Why are these two commands important?

The first can help avoid a common programming error which is where you accidentally misspell a variable's name. For example, your program might have the current temperature saved in a variable called Temp but at one point you accidentally misspell it as Tmp. This will cause MMBasic to automatically create a variable called Tmp and set its value to zero.

This is obviously not what you want and it will introduce a subtle error which could be hard to find, even if you were aware that something was not right. On the other hand, if you used the OPTION EXPLICIT command at the start of your program MMBasic would refuse to automatically create the variable and instead would display an error thereby saving you from a probable headache.

The command OPTION DEFAULT NONE further helps because it tells MMBasic that the programmer must specifically specify the type of every variable when they are declared. It is easy to

命令将显示为 7.5E+18（这是BASIC表示 7.5×10^{18} 的方式）。作为另一个例子，数字0.000000456将显示为 4.56E-8，这与 4.56×10^{-8} 相同。

在MMBasic中输入常量数字时，您还可以使用科学记数法。例如：

```
SandGrains = 7.5E+18
```

MMBasic仅在显示浮点数时使用科学记数法（而不是整数）。例如，如果您将沙粒的数量分配给一个整数变量，它将以正常数字的形式打印出来（有很多零）。

DIM命令

我们之前使用过DIM命令来定义数组，但它也可以用于创建普通变量。例如，您可以同时创建四个字符串变量，如下所示：

```
DIM STRING Car, Name, Street, City
```

请注意，由于这些变量是通过DIM命令定义为字符串的，因此我们不需要\$后缀，单独的定义就足以让MMBasic识别它们的类型。同样，当您在表达式中使用这些变量时，您不需要类型后缀：例如：

```
City = "Sydney"
```

您还可以使用关键字INTEGER来定义多个整数变量，使用FLOAT来定义多个浮点变量。这种类型的表示法也可以用来定义数组。

例如：

```
DIM INTEGER seconds(200)
```

定义变量类型的另一种方法是使用关键字 AS。例如：

```
DIM Car AS STRING, Name AS STRING, Street AS STRING
```

这是微软使用的方法（MMBasic 尝试保持与微软的兼容性），如果变量具有不同的类型，这种方法非常有用。例如：

```
DIM Car AS STRING, Age AS INTEGER, Value AS FLOAT
```

您可以使用这些定义变量类型的方法，它们的作用是相同的。

使用 DIM 命令定义变量的优点在于它们被清晰地定义（最好在程序的开始部分），并且它们的类型（浮点数、整数或字符串）不易被误解。

您可以通过在程序的最顶部使用以下命令来加强这一点：

```
OPTION EXPLICIT  
OPTION DEFAULT NONE
```

第一个命令指定 MMBasic 所有变量必须在使用之前通过 DIM 显式定义。第二个选项指定在创建所有变量时必须指定它们的类型。

这两个命令为什么重要？

第一个可以帮助避免一个常见的编程错误，即你不小心拼错了变量的名称。例如，你的程序可能将当前温度保存在一个名为 Temp 的变量中，但在某个时刻你不小心将其拼错为 Tmp。这将导致MMBasic自动创建一个名为 Tmp 的变量，并将其值设置为零。

显然，这不是你想要的，它会引入一个微妙的错误，即使你意识到有些地方不对，也可能很难找到。另一方面，如果你在程序开始时使用了OPTION EXPLICIT命令，MMBasic将拒绝自动创建该变量，而是会显示一个错误，从而避免你可能的麻烦。

命令OPTION DEFAULT NONE进一步帮助，因为它告诉MMBasic程序员在声明每个变量时必须明确指定其类型。

forget to specify the type and allowing MMBasic to automatically assume the type can lead to unexpected consequences.

For small, quick and dirty programs, it is fine to allow MMBasic to automatically create variables but in larger programs you should always disable this feature with OPTION EXPLICIT and strengthen it with OPTION DEFAULT NONE.

When a variable is created it is set to zero for float and integers and an empty string (ie, contains no characters) for a string variable. You can set its initial value to something else when it is created using DIM. For example:

```
DIM FLOAT nbr = 12.56
DIM STRING Car = "Ford", City = "Perth"
```

You can also initialise arrays by placing the initialising values inside brackets like this:

```
DIM s$(2) = ("zero", "one", "two")
```

Note that because arrays start from zero by default this array actually has three elements with the index numbers of 0, 1 and 2. This is why we needed three string constants to initialise it.

Constants

A common requirement in programming is to define an identifier that represents a value without the risk of the value being accidentally changed - which can happen if variables were used for this purpose. These are called constants and they can represent I/O pin numbers, signal limits, mathematical constants and so on.

You can create a constant using the CONST command. This defines an identifier that acts like a variable but is set to a value that cannot be changed.

For example, if you wanted to check the voltage of a battery connected to pin 31 you could define the relevant values thus:

```
CONST BatteryVoltagePin = 31
CONST BatteryMinimum = 1.5
```

These constants can then be used in the program where they make more sense to the casual reader than simple numbers.

For example:

```
SETPIN BatteryVoltagePin, AIN
IF PIN(BatteryVoltagePin) < BatteryMinimum THEN SoundAlarm
```

It is good programming practice to use constants for any fixed number that represents an important value. Normally they are defined at the start of a program where they are easy to see and conveniently located for another programmer to adjust (if necessary).

Subroutines

A subroutine is a block of programming code which is self contained (like a module) and can be called from anywhere within your program. To your program it looks like a built in MMBasic command and can be used the same. For example, assume that you need a command that would signal an error by printing a message on the console. You could define the subroutine like this:

```
SUB ErrMsg
    PRINT "Error detected"
END SUB
```

With this subroutine embedded in your program all you have to do is use the command ErrMsg whenever you want to display the message. For example:

```
IF A < B THEN ErrMsg
```

忘记指定类型并允许MMBasic自动假设类型可能会导致意想不到的后果。

对于小型、快速且简单的程序，允许MMBasic自动创建变量是可以的，但在较大的程序中，您应该始终使用OPTION EXPLICIT禁用此功能，并通过OPTION DEFAULT NONE加强这一点。

当变量被创建时，浮点数和整数的初始值为零，而字符串变量的初始值为一个空字符串（即，不包含任何字符）。您可以在创建时使用DIM将其初始值设置为其他值。例如：

```
DIM FLOAT nbr = 12.56  
DIM STRING Car = "福特", City = "珀斯"
```

您还可以通过将初始化值放在括号内来初始化数组，如下所示：

```
DIM s$(2) = ("零", "一", "二")
```

请注意，由于数组默认从零开始，因此该数组实际上有三个元素，索引号为0、1和2。这就是我们需要三个字符串常量来初始化它的原因。

常量

编程中的一个常见需求是定义一个标识符，表示一个值，而不必担心该值被意外更改——如果使用变量来实现这一目的，就可能发生这种情况。

这些被称为常量，它们可以表示输入/输出引脚编号、信号限制、数学常量等。

您可以使用CONST命令创建一个常量。这定义了一个像变量一样的标识符，但其值是不可更改的。

例如，如果您想检查连接到引脚31的电池电压，可以这样定义相关值：

```
CONST BatteryVoltagePin = 31  
CONST BatteryMinimum = 1.5
```

这些常量可以在程序中使用，它们对普通读者来说比简单的数字更有意义。

例如：

```
SETPIN BatteryVoltagePin, AIN  
如果 PIN(BatteryVoltagePin) < BatteryMinimum 则 SoundAlarm
```

使用常量表示任何重要值的固定数字是良好的编程实践。通常它们在程序的开始部分定义，这样便于查看，并且方便其他程序员进行调整（如果需要的话）。

子例程

子程序是一个自包含的编程代码块（类似于模块），可以在程序的任何地方被调用。对于你的程序来说，它看起来像是一个内置的MMBasic命令，并且可以以相同的方式使用。例如，假设你需要一个命令，通过在控制台上打印消息来发出错误信号。你可以这样定义子程序：

```
SUB ErrMsg  
    PRINT "检测到错误"  
END SUB
```

通过将此子程序嵌入到您的程序中，您只需在想要显示消息时使用命令ErrMsg。例如：

```
IF A < B THEN ErrMsg
```

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

The above example is fine enough but it would be better if a more useful message could be displayed, one that could be customised every time the subroutine was called. This can be done by passing a string to the subroutine as an argument (sometimes called a parameter).

In this case the definition of the subroutine would look like this:

```
SUB ErrMsg  Msg$  
    PRINT "Error: " + Msg$  
END SUB
```

Then, when you call the subroutine, you can supply the string to be printed on the command line of the subroutine.

For example:

```
IF A < B THEN ErrMsg "Number too small"
```

When the subroutine is called like this the message "Error: Number too small" will be printed on the console. Inside the subroutine `Msg$` will have the value of "Number too small" when called like this and it will be concatenated in the `PRINT` statement to make the full error message.

A subroutine can have any number of arguments which can be float, integer or string with each argument separated by a comma.

Within the subroutine 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 hidden within the subroutine and be different from arguments defined for the subroutine.

The type of the argument to be supplied can be specified with a type suffix (ie, `$`, `%` or `!` for string, integer and float). For example, in the following the first argument must be a string and the second an integer:

```
SUB MySub Msg$, Nbr%  
    ...  
END SUB
```

MMBasic will convert the supplied values if it can, so if your program supplied a floating point value as the second argument MMBasic will convert it to an integer. If MMBasic cannot convert the value it will display an error and return to the command prompt. For example, if you supplied a string for the second argument your program will stop with an error.

You do not have to use the type suffixes, you can instead define the type of the arguments using the `AS` keyword similar to the way it is used in the `DIM` command.

For example, the following is identical to the above example:

```
SUB MySub Msg AS STRING, Nbr AS INTEGER  
    ...  
END SUB
```

Of course, if you used only one variable type throughout the program and used `OPTION DEFAULT` to set that type you could ignore the question of variable types completely.

When a subroutine is called with an argument that is a variable (ie, not a constant or expression) MMBasic will create a corresponding variable within the subroutine *that points back to this variable*. Any changes to the variable representing the argument inside the subroutine will also change the variable used in the call. This is called passing arguments by reference.

子程序的定义可以在程序的任何地方，但通常位于末尾。如果MMBasic在运行你的程序时遇到定义，它将简单地跳过它。上述示例很好，但如果能够显示更有用的消息，那就更好了，这样每次调用子程序时都可以自定义消息。这可以通过将字符串作为参数（有时称为参数）传递给子程序来完成。

在这种情况下，子程序的定义看起来像这样：

```
SUB ErrMsg    Msg$  
    PRINT "错误：" + Msg$  
END SUB
```

然后，当你调用子程序时，可以在子程序的命令行中提供要打印的字符串。

例如：

如果 A < B 那么 ErrMsg "数字太小"

当子程序像这样被调用时，控制台上将打印消息 "错误：数字太小"。在子程序内部，Msg\$在像这样被调用时将具有 "数字太小" 的值，并将在 PRINT 语句中连接以形成完整的错误信息。

子程序可以有任意数量的参数，这些参数可以是浮点数、整数或字符串，每个参数之间用逗号分隔。

在子程序内部，参数像普通变量一样起作用，但它们仅存在于子程序内，并将在子程序结束时消失。您可以在主程序中使用相同名称的变量，它们将在子程序中被隐藏，并且与为子程序定义的参数不同。

可以使用类型后缀（即，\$、%或!分别表示字符串、整数和浮点数）来指定要提供的参数类型。例如，在以下示例中，第一个参数必须是字符串，第二个参数必须是整数：

```
SUB MySub  Msg$, Nbr%  
    ...  
    结束子程序
```

如果可以，MMBasic将转换提供的值，因此如果您的程序将浮点值作为第二个参数提供，MMBasic将其转换为整数。如果MMBasic无法转换该值，它将显示错误并返回到命令提示符。例如，如果您为第二个参数提供了一个字符串，您的程序将因错误而停止。

您不必使用类型后缀，您可以使用AS关键字定义参数的类型，类似于在DIM命令中使用的方式。

例如，以下内容与上述示例相同：

```
SUB MySub  Msg AS STRING, Nbr AS INTEGER  
    ...  
    结束子程序
```

当然，如果在整个程序中只使用一种变量类型，并使用选项默认来设置该类型，则可以完全忽略变量类型的问题。

当子程序被调用时，如果参数是一个变量（即，不是常量或表达式），MMBasic将在子程序中创建一个相应的变量指向这个变量。

在子程序内部对表示参数的变量的任何更改也会更改调用中使用的变量。这称为通过引用传递参数。

This is best explained by example:

```
DIM MyNumber = 5          ' set the variable to 5
CalcSquare MyNumber      ' the subroutine will square its value
PRINT MyNumber           ' this will print the number 25
END

SUB CalcSquare nbr
    nbr = nbr * nbr      ' square the argument and pass it back
END SUB
```

The subroutine CalcSquare will take its argument, square it and write it back to the variable representing the argument (nbr). Because the subroutine was called with a variable (MyNumber) the variable nbr will point back to MyNumber and any change to nbr will also change MyNumber accordingly. As a result the PRINT statement will output 25.

Passing arguments by reference is handy because it allows a subroutine to pass values back to the code that called it. However it could lead to trouble if a subroutine used the variable representing an argument as a general purpose variable and changed its value. Then, if it were called with a variable as an argument, that variable would be inadvertently changed. For this reason **you should avoid manipulating variables representing arguments inside a subroutine**, instead assign the value to a local variable (see below) and manipulate that instead.

When you call a subroutine you can omit some (or all) of the parameters and they will take the value of zero (for floats or integers) or an empty string. This is handy as your subroutine can tell if a parameter is missing and act accordingly.

For example, here is our subroutine to generate an error message but this version can be used without specifying an error message as a parameter:

```
SUB ErrMsg  Msg$
    IF Msg$ = "" THEN
        PRINT "Error detected"
    ELSE
        PRINT "Error: " + Msg$
    ENDIF
END SUB
```

Within a subroutine you can use most features of MMBasic including calling other subroutines, IF...THEN commands, FOR...NEXT loops and so on. However, one thing that you cannot do is jump out of a subroutine using GOTO (if you do the result will be undefined and may cause your hair to turn grey).

Normally the subroutine will exit when the END SUB command is reached but you can also terminate the subroutine early by using the EXIT SUB command.

Functions

Functions are similar to subroutines with the main difference being that a function is used to return a value in an expression. For example, if you wanted a function to convert a temperature from degrees Celsius to Fahrenheit you could define:

```
FUNCTION Fahrenheit(C)
    Fahrenheit = C * 1.8 + 32
END FUNCTION
```

Then you could use it in an expression:

```
Input "Enter a temperature in Celsius: ", t
PRINT "That is the same as" Fahrenheit(t) "F"
```

这最好通过示例来解释：

```
DIM MyNumber = 5      ' 将变量设置为 5
CalcSquare MyNumber   ' 子程序将平方其值
打印 MyNumber         ' 这将打印数字 25
结束
```

```
SUB CalcSquare nbr
    nbr = nbr * nbr      ' 平方参数并将其传回
结束子程序
```

子程序 CalcSquare 将接受其参数，平方它并将其写回表示参数的变量 (nbr)。因为子程序是通过变量 (MyNumber) 调用的，所以变量 nbr 将指向 MyNumber，任何对 nbr 的更改也会相应地更改 MyNumber。因此，PRINT语句将输出25。

通过引用传递参数很方便，因为它允许子程序将值传回调用它的代码。然而，如果子程序将表示参数的变量作为通用变量并更改其值，可能会导致问题。然后，如果它被一个变量作为参数调用，该变量将被意外更改。因此，**你应该避免在子程序内部操作表示参数的变量**，而是将值分配给局部变量（见下文）并操作该变量。

当你调用子程序时，可以省略一些（或全部）参数，它们将取值为零（对于浮点数或整数）或空字符串。这很方便，因为你的子程序可以判断参数是否缺失并相应地采取行动。

例如，这是我们的子程序，用于生成错误信息，但这个版本可以在不指定错误信息作为参数的情况下使用：

```
SUB ErrMsg    Msg$
    IF Msg$ = "" THEN
        PRINT "检测到错误"
    ELSE
        PRINT "错误：" + Msg$
    ENDIF
END SUB
```

在子程序中，你可以使用MMBasic的大多数功能，包括调用其他子程序、IF...THEN命令、FOR...NEXT循环等。然而，有一件事你不能做，那就是使用GOTO跳出子程序（如果这样做，结果将是未定义的，可能会让你的头发变灰）。

通常，当达到END SUB命令时，子程序将退出，但你也可以通过使用EXIT SUB命令提前终止子程序。

函数

函数与子程序类似，主要区别在于函数用于在表达式中返回一个值。例如，如果你想要一个将摄氏度转换为华氏度的函数，你可以定义：

```
FUNCTION Fahrenheit(C)
    Fahrenheit = C * 1.8 + 32
END FUNCTION
```

然后你可以在一个表达式中使用它：

```
输入 "请输入摄氏温度：" , t
打印 "这相当于" Fahrenheit(t) "华氏度"
```

Or as another example:

```
IF Fahrenheit(temp) <= 32 THEN PRINT "Freezing"
```

You could also define the reverse:

```
FUNCTION Celsius(F)
    Celsius = (F - 32) * 0.5556
END FUNCTION
```

As you can see, the function name is used as an ordinary local variable inside the subroutine. It is only when the function returns that the value is made available to the expression that called it.

The rules for the argument list in a function are similar to subroutines. The only difference is that parentheses are always required around the argument list when you are calling a function, even if there are no arguments (parentheses are optional when calling a subroutine).

To return a value from the function you assign a value to the function's name within the function. If the function's name is terminated with a type suffix (ie, \$, a % or a !) the function will return that type (string, integer or float), otherwise it will return whatever the OPTION DEFAULT is set to. For example, the following function will return a string:

```
FUNCTION LVal$(nbr)
    IF nbr = 0 THEN LVal$ = "False" ELSE LVal$ = "True"
END FUNCTION
```

You can explicitly specify the type of the function by using the AS keyword and then you do not need to use a type suffix (similar to defining a variable using DIM).

This is the above example rewritten to take advantage of this feature:

```
FUNCTION LVal(nbr) AS STRING
    IF nbr = 0 THEN LVal = "False" ELSE LVal = "True"
END FUNCTION
```

In this case the type returned by the function LVal will be a string.

As for subroutines you can use most features of MMBasic within functions. This includes FOR...NEXT loops, calling other functions and subroutines, etc. Also, the function will return to the expression that called it when the END FUNCTION command is reached but you can also return early by using the EXIT FUNCTION command.

Local Variables

Variables that are created using DIM or that are just automatically created are called *global* variables. This means that they can be seen and used anywhere in the program including within subroutines and functions. However, inside a subroutine or function you will often need to use variables for various tasks that are internal to the subroutine/function. In portable code you do not want the name you chose for such a variable to clash with a global variable of the same name. To this end you can define a variable using the LOCAL command within the subroutine/function.

The syntax for LOCAL is identical to the DIM command, this means that the variable can be an array, you can set the type of the variable and you can initialise it to some value.

For example, this is our ErrMsg subroutine but this time it has been extended to use a local variable for joining the error message strings.

```
SUB ErrMsg  Msg$
    LOCAL STRING tstr
    tstr = "Error: " + Msg$
    PRINT tstr
END SUB
```

The variable tstr is declared as LOCAL within the subroutine, which means that (like the argument list) it will only exist within the subroutine and will vanish when the subroutine exits. You can have a

或者作为另一个例子：

```
如果 Fahrenheit(temp) <= 32 那么 打印 "冰冻"
```

你也可以定义反向转换：

```
FUNCTION Celsius(F)
    Celsius = (F - 32) * 0.5556
END FUNCTION
```

如你所见，函数名称在子程序内部被用作普通的局部变量。只有当函数返回时，值才会提供给调用它的表达式。函数的参数列表规则与子例程类似。唯一的区别是，当你调用一个函数时，参数列表总是需要用括号括起来，即使没有参数（调用子例程时括号是可选的）。

要从函数返回一个值，你需要在函数内部将一个值赋给函数的名称。如果函数的名称以类型后缀（即，\$、%或!）结束，则该函数将返回该类型（字符串、整数或浮点数），否则它将返回OPTION DEFAULT设置的值。例如，以下函数将返回一个字符串：

```
FUNCTION LVal$(nbr)
    IF nbr = 0 THEN LVal$ = "假" ELSE LVal$ = "真"
END FUNCTION
```

您可以通过使用AS关键字显式指定函数的类型，这样您就不需要使用类型后缀（类似于使用DIM定义变量）。这是上述示例重写以利用此功能：

```
FUNCTION LVal(nbr) AS STRING
    IF nbr = 0 THEN LVal = "假" ELSE LVal = "真"
END FUNCTION
```

在这种情况下，函数 LVal 返回的类型将是字符串。

至于子例程，您可以在函数中使用MMBasic的大多数功能。这包括FOR...NEXT循环、调用其他函数和子例程等。此外，当达到END FUNCTION命令时，函数将返回到调用它的表达式，但您也可以通过使用EXIT FUNCTION命令提前返回。

局部变量

使用DIM创建的变量或自动创建的变量称为全局变量。

这意味着它们可以在程序的任何地方被看到和使用，包括在子例程和函数内。然而，在子例程或函数内部，您通常需要使用变量来执行子例程/函数内部的各种任务。在可移植代码中，您不希望为这样的变量选择的名称与同名的全局变量冲突。为此，您可以在子例程/函数内使用LOCAL命令定义一个变量。

LOCAL的语法与DIM命令相同，这意味着变量可以是数组，您可以设置变量的类型，并可以将其初始化为某个值。

例如，这是我们的ErrMsg子程序，但这次它已扩展为使用局部变量来连接错误消息字符串。

```
SUB ErrMsg    Msg$
    LOCAL STRING tstr
    tstr = "错误：" + Msg$
    PRINT tstr
END SUB
```

变量tstr在子程序中被声明为局部，这意味着（像参数列表一样）它只会在子程序内存在，并在子程序退出时消失。你可以有一个

global variable called `tstr` in your main program and it will be different from the variable `tstr` in the subroutine (in this case the global `tstr` will be hidden within the subroutine).

You should always use local variables for operations within your subroutine or function because they help make the module much more self contained and portable.

Static Variables

LOCAL variables are reset to their initial values (normally zero or an empty string) every time the subroutine or function starts, however there are times when you would like the variable to retain its value between calls. This type of variable is defined with the STATIC command.

We can demonstrate how STATIC variables are useful by extending the ErrMsg subroutine to prevent duplicated calls to the subroutine repeatedly displaying the same message. For example, our program might call this subroutine from multiple places but if the message is the same in a number of subsequent calls we would like to see the message just once. This is our new subroutine:

```
SUB ErrMsg  Msg$  
    STATIC STRING lastmsg  
    LOCAL STRING tstr  
    IF Msg$ <> lastmsg THEN  
        tstr = "Error: " + Msg$  
        PRINT tstr  
        lastmsg = Msg$  
    ENDIF  
END SUB
```

To keep track of the last message displayed we use a static variable called `lastmsg`. This will hold the text of the last message and we can compare it to the current message text to determine if it is different and therefore should be printed. This would give just one message every time a call is made with a duplicate message text.

The STATIC command uses exactly the same syntax as DIM. This means that you can define different types of static variables including arrays and you can also initialise them to some value.

The static variable is created on the first time the STATIC command is encountered and it is automatically set to zero (if a float or integer) or an empty string. On subsequent calls to the subroutine or function MMBasic will recognise that the variable has already been created and it will leave its value untouched (ie, whatever it was in the previous call). As with DIM you can also initialise a static variable to some value. For example:

```
STATIC INTEGER var = 123
```

On the first call (when the variable is created) it will be initialised to 123 but on subsequent calls it will keep whatever its value was previously set to.

Mostly static variables are used to keep track of the *state* of something while inside a subroutine or function. A *state* is a record of something that has happened previously.

Examples include:

- Has the COM port already been opened?
- What steps in a sequence have we completed?
- What text has already been displayed?

Normally you will use global variables (created using DIM) to track a *state* but sometimes you want this to be contained within a module and this is where static variables are valuable. Just like LOCAL the use of STATIC helps to make your subroutines and functions more self contained and portable.

在你的主程序中定义一个名为 `tstr` 的全局变量，它将与子程序中的变量 `tstr` 不同（在这种情况下，全局 `tstr` 将在子程序中被隐藏）。

你应该始终在子程序或函数中使用局部变量，因为它们有助于使模块更加自包含和可移植。

静态变量

局部变量在每次子程序或函数开始时会重置为其初始值（通常为零或空字符串），然而有时您希望变量在调用之间保留其值。这种类型的变量是通过 STATIC 命令定义的。

我们可以通过扩展 ErrMsg 子程序来演示 STATIC 变量的用处，以防止重复调用子程序而不断显示相同的消息。例如，我们的程序可能会从多个地方调用这个子程序，但如果在多次后续调用中消息相同，我们希望只看到一次该消息。这是我们的新子程序：

```
SUB ErrMsg    Msg$  
    STATIC STRING lastmsg  
    LOCAL STRING tstr  
    IF Msg$ <> lastmsg THEN  
        tstr = "错误: " + Msg$  
        PRINT tstr  
        lastmsg = Msg$  
    ENDIF  
END SUB
```

为了跟踪最后显示的消息，我们使用一个名为 `lastmsg` 的静态变量。这将保存最后消息的文本，我们可以将其与当前消息文本进行比较，以确定它是否不同，因此应该被打印。这样每次调用重复消息文本时只会显示一条消息。

STATIC 命令的语法与 DIM 完全相同。这意味着您可以定义不同类型的静态变量，包括数组，并且您还可以将它们初始化为某个值。

静态变量在第一次遇到 STATIC 命令时创建，并且会自动设置为零（如果是浮点数或整数）或空字符串。在后续对子程序或函数的调用中，MMBasic 将识别该变量已经被创建，并且将保持其值不变（即，无论在上一次调用中是什么值）。与 DIM 一样，您还可以将静态变量初始化为某个值。例如：

```
STATIC INTEGER var = 123
```

在第一次调用时（当变量被创建时），它将被初始化为 123，但在后续调用中，它将保持之前设置的值。

静态变量主要用于在子程序或函数内部跟踪某个事物的状态。状态是对之前发生过的事情的记录。

示例包括：

- COM 端口已经打开了吗？
- 我们完成了序列中的哪些步骤？
- 已经显示了哪些文本？

通常，您会使用全局变量（使用 DIM 创建）来跟踪状态，但有时您希望将其包含在模块中，这时静态变量就显得非常有价值。就像 LOCAL 一样，使用 STATIC 有助于使您的子例程和函数更加自包含和可移植。

Calculate Days

We have covered a lot of programming commands and techniques so far in this tutorial and before we finish it would be worth giving an example of how they work together. The following is an example that uses many features of the BASIC language to calculate the number of days between two dates:

```
' Example program to calculate the number of days between two dates

OPTION EXPLICIT
OPTION DEFAULT NONE

DIM STRING s
DIM FLOAT d1, d2

DO
    ' main program loop
    PRINT : PRINT "Enter the date as dd mmmm yyyy"
    PRINT " First date";
    INPUT s
    d1 = GetDays(s)
    IF d1 = 0 THEN PRINT "Invalid date!" : CONTINUE DO
    PRINT "Second date";
    INPUT s
    d2 = GetDays(s)
    IF d2 = 0 THEN PRINT "Invalid date!" : CONTINUE DO
    PRINT "Difference is" ABS(d2 - d1) " days"
LOOP

' Calculate the number of days since 1/1/1900
FUNCTION GetDays(d$) AS FLOAT
    LOCAL STRING Month(11) =
("jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec")
    LOCAL FLOAT Days(11) = (0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334)
    LOCAL FLOAT day, mth, yr, s1, s2

    ' Find the separating space character within a date
    s1 = INSTR(d$, " ")
    IF s1 = 0 THEN EXIT FUNCTION
    s2 = INSTR(s1 + 1, d$, " ")
    IF s2 = 0 THEN EXIT FUNCTION

    ' Get the day, month and year as numbers
    day = VAL(MID$(d$, 1, s2 - 1)) - 1
    IF day < 0 OR day > 30 THEN EXIT FUNCTION
    FOR mth = 0 TO 11
        IF LCASE$(MID$(d$, s1 + 1, 3)) = Month(mth) THEN EXIT FOR
    NEXT mth
    IF mth > 11 THEN EXIT FUNCTION
    yr = VAL(MID$(d$, s2 + 1)) - 1900
    IF yr < 1 OR yr >= 200 THEN EXIT FUNCTION

    ' Calculate the number of days including adjustment for leap years
    GetDays = (yr * 365) + FIX((yr - 1) / 4)
    IF yr MOD 4 = 0 AND mth >= 2 THEN GetDays = GetDays + 1
    GetDays = GetDays + Days(mth) + day
END FUNCTION
```

Note that the line starting LOCAL STRING Month(11) has been wrapped around because of the limited page width – it is one line as follows:

```
LOCAL STRING Month(11) = ("jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec")
```

This program works by getting two dates from the user at the console and then converting them to integers representing the number of days since 1900. With these two numbers a simple subtraction will give the number of days between them.

计算天数

到目前为止，我们在本教程中涵盖了许多编程命令和技术，在结束之前，给出一个它们如何协同工作的示例是值得的。以下是一个示例使用BASIC语言的许多特性来计算两个日期之间的天数：

· 计算两个日期之间天数的示例程序

```
OPTION EXPLICIT
OPTION DEFAULT NONE

DIM STRING s
DIM FLOAT d1, d2

DO
    ' 主程序循环
    PRINT : PRINT "请输入日期，格式为 dd mmm yyyy"
    PRINT " 第一个日期";
    INPUT s
    d1 = GetDays(s)
    如果 d1 = 0 那么 打印 "无效的日期！" : 继续循环
    打印 "第二个日期";
    输入 s
    d2 = 获取天数(s)
    如果 d2 = 0 那么 打印 "无效的日期！" : 继续循环
    打印 "差异是" ABS(d2 - d1) " 天数"
循环

· 计算自1900年1月1日以来的天数
函数 获取天数(d$) 作为 浮点数
局部 字符串 月(11) =
("一月", "二月", "三月", "四月", "五月", "六月", "七月", "八月", "九月", "十月", "十一月", "十二月")
局部 浮点数 天数(11) = (0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334)
局部 浮点数 day, mth, yr, s1, s2

· 在日期中查找分隔空格字符
s1 = INSTR(d$, " ")
如果 s1 = 0 那么 退出函数
s2 = INSTR(s1 + 1, d$, " ")
如果 s2 = 0 那么 退出函数

· 获得天、月和年作为数字
day = VAL(MID$(d$, 1, s2 - 1)) - 1
如果 day < 0 或 day > 30 则 退出函数
对于 mth = 0 到 11
    如果 LCASE$(MID$(d$, s1 + 1, 3)) = Month(mth) 则 退出循环
下一个 mth
如果 mth > 11 则 退出函数
yr = VAL(MID$(d$, s2 + 1)) - 1900
如果 yr < 1 或 yr >= 200 则 退出函数

· 计算天数，包括闰年的调整
GetDays = (yr * 365) + FIX((yr - 1) / 4)
如果 yr MOD 4 = 0 且 mth >= 2 则 GetDays = GetDays + 1
    GetDays = GetDays + Days(mth) + day
结束函数
```

请注意，由于页面宽度有限，开始于LOCAL STRING Month(11)的行已被换行- 它作为一行如下所示：

```
LOCAL STRING Month(11) = ("jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec")
```

该程序通过从控制台获取两个日期，然后将它们转换为表示自1900年以来的天数的整数来工作。通过这两个数字，简单的减法将给出它们之间的天数。

When this program is run it will ask for the two dates to be entered and you need to use the form of:
dd mmm yyyy.

This screen capture shows what the running program will look like.

The main feature of the program is the defined function `GetDays()` which takes a string entered at the console, splits it into its day, month and year components then calculates the number of days since 1st January 1900.

This function is called twice, once for the first date and then again for the second date. It is then just a matter of subtracting one date (in days) from the other to get the difference in days.

```
File Edit Setup Control Window Help
> RUN
Enter the date as dd mmm yyyy
First date? 20 January 1980
Second date? 1 March 2016
Difference is 13190 days

Enter the date as dd mmm yyyy
First date? 1 MAR 2016
Second date? 31 OCT 2018
Difference is 974 days

Enter the date as dd mmm yyyy
First date? █
```

We will not go into the detail of how the calculations are made (ie, handling leap years) as that can be left as an exercise for the reader. However, it is appropriate to point out some features of MMBasic that are used by the program.

It demonstrates how local variables can be used and how they can be initialised. In the function `GetDays()` two arrays are declared and initialised at the same time. These are just a convenient method of looking up the names of the months and the cumulative number of days for each month. Later in the function (the FOR loop) you can see how they make dealing with twelve different months quite efficient.

Another feature highlighted by this program is the string handling features of MMBasic. The `INSTR()` function is used to locate the two space characters in the date string and then later the `MID$()` function uses these to extract the day, month and year components of the date. The `VAL()` function is used to turn a string of digits (like the year) into a number that can be stored in a numeric variable.

Note that the value of a function is initialised to zero every time the function is run and unless it is set to some value it will return a zero value. This makes error handling easy because we can just exit the function if an error is discovered. It is then the responsibility of the calling program code to check for a return value of zero which signifies an error.

This program illustrates one of the benefits of using subroutines and functions which is that when written and fully tested they can be treated as a trusted "black box" that does not have to be opened. For this reason functions like this should be the properly tested and then, if possible, left untouched (in case you add some error).

There are a few features of this program that we have not covered before. The first is the `MOD` operator which will calculate the remainder of dividing one number into another. For example, if you divided 4 into 15 you would have a remainder of 3 which is exactly what the expression `15 MOD 4` will return. The `ABS()` function is also new and will return its argument as a positive number (eg, `ABS(-15)` will return `+15` as will `ABS(15)`).

The `EXIT FOR` command will exit a `FOR` loop even though it has not reached the end of its looping, `EXIT FUNCTION` will immediately exit a function even though execution has not reached the end of the function and `CONTINUE DO` will immediately cause the program to jump to the end of a `DO` loop and execute it again.

Why would this program be useful? Well some people like to count their age in days, that way every day is a birthday! You can calculate your age in days, just enter the date that you were born and today's date. That is not particularly useful but the program itself is valuable as it demonstrates many of the characteristics of programming in MMBasic. So, work your way through the program and review each section until you understand it – it should be a rewarding experience.

当运行此程序时，它将要求输入两个日期，您需要使用以下格式：

dd mmm yyyy .

此屏幕截图显示了运行程序的外观。

该程序的主要功能是定义的函数 GetDays()，它接受在控制台输入的字符串，将其拆分为日期、月份和年份组件，然后计算自1900年1月1日以来的天数。

这个函数被调用两次，第一次用于第一个日期，然后再次用于第二个日期。然后只需从一个日期（以天数为单位）中减去另一个日期即可得到天数差。

我们不会详细讨论计算是如何进行的（即处理闰年），因为这可以留给读者作为练习。然而，指出程序中使用的一些MMBasic特性是合适的。

The screenshot shows a terminal window titled 'COM23:38400baud - Tera Term VT'. The window has a menu bar with File, Edit, Setup, Control, Window, and Help. A status bar at the bottom says 'RUN'. The main text area contains three entries:

```
> RUN
Enter the date as dd mmm yyyy
First date? 20 January 1980
Second date? 1 March 2016
Difference is 13190 days

Enter the date as dd mmm yyyy
First date? 1 MAR 2016
Second date? 31 OCT 2018
Difference is 974 days

Enter the date as dd mmm yyyy
First date? ■
```

它演示了如何使用局部变量以及如何初始化它们。在函数GetDays()中，两个数组被声明并同时初始化。这些只是查找月份名称和每个月累计天数的方便方法。

在函数的后面（FOR循环）中，您可以看到它们如何使处理十二个月变得相当高效。

这个程序强调的另一个特性是MMBasic的字符串处理功能。INSTR()函数用于定位日期字符串中的两个空格字符，然后稍后MID\$()函数使用这些字符提取日期的日、月和年组成部分。VAL()函数用于将一串数字（如年份）转换为可以存储在数值变量中的数字。

请注意，每次运行函数时，函数的值都会初始化为零，除非将其设置为某个值，否则它将返回零值。这使得错误处理变得简单，因为如果发现错误，我们可以直接退出函数。然后，调用程序代码有责任检查返回值是否为零，这表示发生了错误。

这个程序说明了使用子例程和函数的一个好处，即在编写并充分测试后，它们可以被视为一个可信的“黑箱”，无需打开。

因此，像这样的函数应该经过适当的测试，然后如果可能的话，保持不变（以防你添加了一些错误）。

这个程序有一些我们之前没有涵盖的特性。第一个是 MOD 运算符，它将计算一个数字除以另一个数字的余数。例如，如果你将4除以15，你会得到余数3，这正是表达式15MOD 4的返回值。ABS()函数也是新的，它将返回其参数的正数（例如，ABS(-15)将返回+15，ABS(15)也将返回+15）。

EXIT FOR命令将退出FOR循环，即使它尚未达到循环的结束，EXIT FUNCTION将立即退出一个函数，即使执行尚未达到函数的结束，而CONTINUE DO将立即使程序跳转到DO循环的末尾并再次执行。

这个程序有什么用处？有些人喜欢用天数来计算他们的年龄，这样每天都是生日！你可以计算你的年龄（以天为单位），只需输入你的出生日期和今天的日期。这并不是特别有用，但程序本身是有价值的，因为它展示了MMBasic编程的许多特性。因此，逐步浏览程序并审查每个部分，直到你理解它——这应该是一个有益的体验。