PicoMite

Ein Raspberry Pi Pico mit MMBasic Interpreter

Benutzerhandbuch
MMBasic Ver 5.07.03
Rev. 6

Deutsche Ausgabe

Aktualisierungen sowie weitere Details zu MMBasic finden Sie hier:

http://geoffg.net/picomite.html

und http://mmbasic.com

Über das Projekt

Peter Mather (matherp im Back Shed Forum) hat MMBasic auf den Raspberry Pi Pico portiert, die Treiber für dessen Hardwarefeatures geschrieben und das Projekt vorangetrieben. Der MMBasic-Interpreter und dieses Handbuch wurden von Geoff Graham (http://geoffg.net) geschrieben. Mick Ames (Mixtel90 im Back Shed Forum) hat den PIO-Compiler und die dazugehörige Dokumentation geschrieben und zahlreiche Tests durchgeführt.

Support

Support-Fragen sollten im Back Shed-Forum (http://www.thebackshed.com/forum/Microcontrollers) gestellt werden, wo es viele begeisterte MMBasic-Benutzer gibt, die gerne helfen. Auch die Entwickler der PicoMite-Firmware sind Stammgäste in diesem Forum.

Copyright und Danksagungen

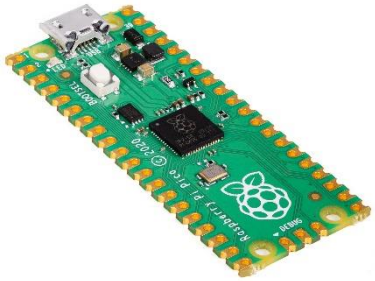
.

Dieses Handbuch

Autor dieses Handbuchs ist Geoff Graham. Das Buch basiert auf vorläufigem Material, das von Mick Ames geschrieben wurde und wird unter einer Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Australia-Lizenz (CC BY-NC-SA 3.0) vertrieben

Inhaltsverzeichnis

Einführung



Der PicoMite ist ein Raspberry Pi Pico, auf dem der MMBasic-Interpreter läuft..

MMBasic ist eine Microsoft BASIC-kompatible Implementierung der BASIC-Sprache mit Fließkomma-, Integer- und String-Variablen, Arrays, langen Variablennamen, einem eingebauten Programmeditor und vielen anderen Funktionen.

Mit MMBasic können Sie die I/O-Pins steuern und Kommunikationsprotokolle wie I2C oder SPI verwenden, um Daten von einer Vielzahl von Sensoren zu erhalten. Sie können Daten auf kostengünstigen Farb-LCD-Displays anzeigen, Spannungen messen, digitale Eingänge erkennen und Ausgangspins ansteuern, um Lichter, Relais usw. einzuschalten. Alles innerhalb dieses kostengünstigen Mikrocontrollers.

Die PicoMite-Firmware kann kostenlos heruntergeladen und verwendet werden.

Der Schwerpunkt bei MMBasic liegt auf Benutzerfreundlichkeit und Entwicklung. Der Entwicklungszyklus ist sehr schnell mit der Möglichkeit, sofort von der Bearbeitung zur Ausführung zu wechseln. Fehler werden in einfachem Englisch aufgelistet, und wenn ein Fehler auftritt, ruft ein einzelner Tastendruck den integrierten Editor auf, wobei der Cursor auf der Zeile positioniert ist die den Fehler verursacht hat.

Zusammenfassung:

- Der BASIC-Interpreter unterstützt die Verarbeitung von Fließkommazahlen, 64-Bit-Ganzzahlen und String-Variablen, langen Variablennamen, Arrays aus Fließkomma- oder Ganzzahlen oder Strings mit mehreren Dimensionen, umfangreicher String-Verarbeitung und benutzerdefinierten Subroutinen und Funktionen. Typischerweise wird ein Programm mit bis zu 100.000 Zeilen pro Sekunde ausgeführt. MMBasic erlaubt das Einbetten von kompilierten C-Programmen für schnelle Funktionen. Das laufende Programm kann durch eine PIN-Nummer vor dem Auflisten oder Modifizieren geschützt werden.
- Unterstützung für alle Raspberry Pi Pico IO-Pins. Diese können unabhängig voneinander als Digitaleingang oder -ausgang, Analogeingang, Frequenz- oder Periodenmessung und Zählung konfiguriert werden. Innerhalb von MMBasic können die I/O-Pins dynamisch als Eingänge oder Ausgänge mit oder ohne Pullup- oder Pulldown-Widerständen konfiguriert werden. MMBasic-Befehle erzeugen Impulse und können zur parallelen Datenübertragung verwendet werden. Interrupts können verwendet werden, um Nachrichten zu erzeugen, wenn ein Eingangspin seinen Zustand geändert hat. PWM-Ausgänge können verwendet werden, um verschiedene Sounds zu erzeugen, Servos zu steuern oder computergesteuerte Spannungen für den Antrieb von Geräten zu erzeugen, die einen analogen Eingang verwenden (z. B. Motorsteuerungen). Darüber hinaus kann mit MMBasic auf Pins zugegriffen werden, die auf dem Raspberry Pi Pico nicht freigelegt sind, sodass es auf anderen Modulen verwendet werden kann, die den RP2040-Prozessor verwenden.
- TFT-LCDs mit SPI-Schnittstellen werden unterstützt, sodass das BASIC-Programm Text anzeigen und Linien, Kreise, Kästchen usw. in 65.535 Farben zeichnen kann. Resistive Touch-Controller auf diesen Panels werden ebenfalls unterstützt, sodass sie als anspruchsvolle Eingabegeräte verwendet werden können. Diese Panels kosten in der Regel 7 US-Dollar und bieten eine kostengünstige grafische Hightech-Benutzeroberfläche.
- Volle Unterstützung für SD-Karten: Öffnen von Dateien zum Lesen, Schreiben oder wahlfreien Zugriff sowie Laden und Speichern von Programmen. Die Firmware funktioniert mit Karten bis zu 32 GB, die in FAT16 oder FAT32 formatiert sind, und die erstellten Dateien können auch auf PCs mit Windows, Linux oder dem Mac-Betriebssystem gelesen und geschrieben warden
- • Die Programmierung und Steuerung erfolgt über die USB-Schnittstelle. Alles, was benötigt wird, ist ein Laptop/Desktop-Computer, auf dem ein VT100-Terminalemulator läuft. Nachdem das Programm geschrieben und von Fehlern befreit wurde, kann PicoMite angewiesen werden, das Programm beim Einschalten automatisch ohne Benutzereingriff auszuführen. Für die Entwicklung von Programmen ist keine spezielle Software erforderlich..
- Ein Vollbild-Editor ist in PicoMite integriert. Dies erfordert nur einen VT100-Terminalemulator, der auf einem Laptop oder Desktop-Computer ausgeführt wird, und kann das gesamte Programm in einer Sitzung

bearbeiten. Es enthält erweiterte Funktionen wie farbcodierte Syntax, Suchen und Kopieren, Ausschneiden und Einfügen in und aus einer Zwischenablage.

- Programme können mithilfe des XModem-Protokolls oder durch Streamen des Programms über den seriellen Konsoleneingang einfach von einem Desktop- oder Laptop-Computer (Windows, Mac oder Linux) übertragen werden.
- Eine umfassende Auswahl an Kommunikationsprotokollen ist implementiert, einschließlich I2C, asynchron seriell, RS232, SPI und 1-Wire. Diese können zur Kommunikation mit vielen Sensoren (Temperatur, Feuchtigkeit, Beschleunigung usw.) sowie zum Senden von Daten an Testgeräte verwendet werden.
- Der PicoMite verfügt über integrierte Befehle zur direkten Verbindung mit Infrarot-Fernbedienungen, dem DS18B20-Temperatursensor, LCD-Anzeigemodulen, einer batteriegepufferten Uhr, numerischen Tastenfeldern und mehr.
- **Betriebsspannung: 2.0 - 5.5 Volt** , Stromverbrauch: 10 - 42 mA.

Erste Schritte

PicoMite-Firmware laden

Der Raspberry Pi Pico verfügt über einen eigenen integrierten Firmware-Loader, der einfach zu bedienen ist. Folgen Sie einfach diesen Schritten:

• Laden Sie die PicoMite-Firmware von http://geoffg.net/picomite.html herunter und entpacken Sie die Datei. Identifizieren Sie die Firmware, die etwa „PicoMiteV5.xx.xx.uf2“ heißen sollte.

• Schließen Sie den Raspberry Pi Pico mit einem USB-Kabel an Ihren Computer (Windows, Linux oder Mac) an, während Sie die weiße BOOTSEL-Taste auf dem Raspberry Pi Pico gedrückt halten.

• Der Raspberry Pi Pico sollte sich mit Ihrem Computer verbinden und ein virtuelles Laufwerk namens „RPI-RP2“ erstellen (dasselbe, als ob Sie einen USB-Speicherstick angeschlossen hätten). Dieses Laufwerk enthält zwei Dateien, die Sie ignorieren können.

• Kopieren Sie die Firmware-Datei (mit der Erweiterung .uf2) auf dieses virtuelle Laufwerk.

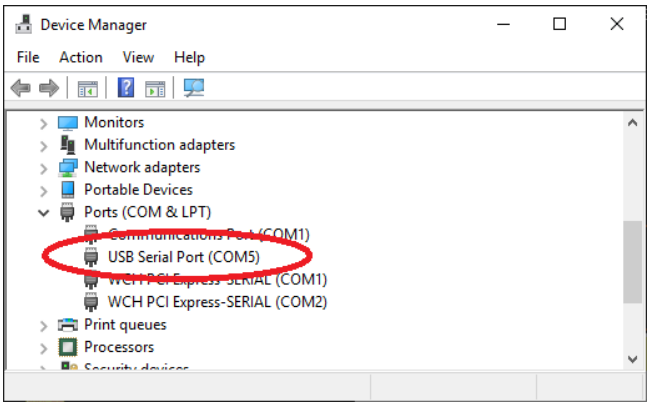
• Wenn der Kopiervorgang abgeschlossen ist, startet der Raspberry Pi Pico neu und erstellt einen virtuellen seriellen Anschluss auf Ihrem Computer. Die LED auf dem Raspberry Pi Pico blinkt langsam und zeigt damit an, dass die PicoMite-Firmware mit MMBasic jetzt ausgeführt wird.

Während das vom Raspberry Pi Pico erstellte virtuelle Laufwerk wie ein USB-Speicherstick aussieht (ist es nicht), die Firmware-Datei verschwindet nach dem Kopieren und wenn Sie versuchen, einen anderen Dateityp zu kopieren, wird sie ignoriert.

Das Laden der PicoMite-Firmware löscht den Flash-Speicher, einschließlich des aktuellen Programms, aller in Flash-Speichersteckplätzen gespeicherten Programme und aller gespeicherten Variablen. Stellen Sie daher sicher, dass Sie diese Daten sichern, bevor Sie die Firmware aktualisieren. Durch das Laden der Firmware werden auch alle Optionen auf ihre Standardwerte zurückgesetzt, sodass dies eine praktische Methode ist, um den PicoMite auf die „Werkseinstellungen“ zurückzusetzen.

.

Virtueller serieller Port



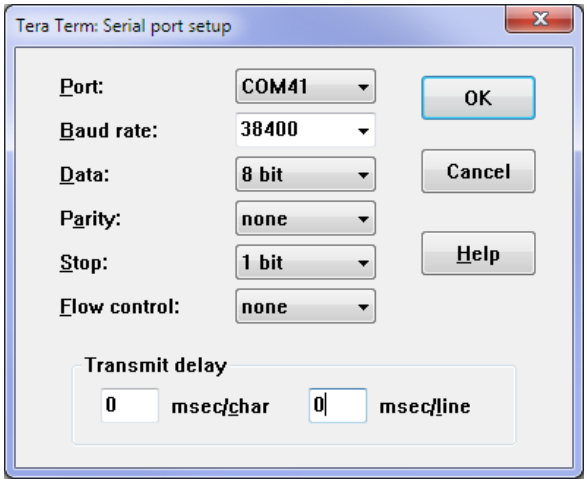
Die von der PicoMite-Firmware erstellte virtuelle serielle Schnittstelle verhält sich wie eine normale serielle Schnittstelle, funktioniert jedoch über USB. Windows 10 enthält einen Treiber für diesen virtuellen seriellen Anschluss, aber bei anderen Versionen müssen Sie möglicherweise einen Treiber laden, damit er mit dem Betriebssystem funktioniert (siehe unten). Sobald dies erledigt ist, sollten Sie die von Ihrem Computer erstellte Portnummer für die virtuelle serielle Verbindung notieren. Unter Windows können Sie dazu den Geräte-Manager starten und den Eintrag „Ports (COM & LPT)“ auf einen neuen COM-Port prüfen, wie rechts gezeigt.

.

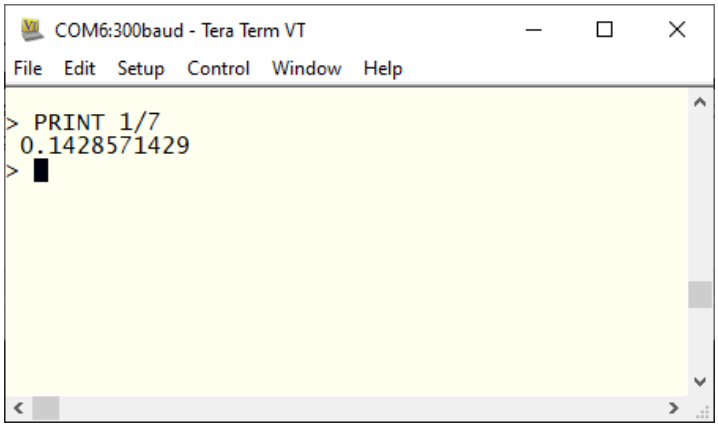
Terminalemulator

Außerdem benötigen Sie ein Terminal-Emulator-Programm auf Ihrem Desktop-Computer. Dieses Programm verhält sich wie ein altmodisches Computerterminal, wo es von einem entfernten Computer empfangenen Text anzeigt und alle Tastendrücke über die serielle Verbindung an den entfernten Computer gesendet werden.Der von Ihnen verwendete Terminal-Emulator sollte die VT100-Emulation unterstützen, da dies der in PicoMite integrierte Editor erwartet. Für Windows-Benutzer wird die Verwendung von Tera Term empfohlen, da es einen guten VT100-Emulator hat und bekanntermaßen mit dem XModem-Protokoll arbeitet, das Sie verwenden können, um Programme zu und von PicoMite zu übertragen (Tera Term kann heruntergeladen werden von: http:/ /tera-term.en.lo4d.com).

Der Screenshot rechts zeigt das Setup für Tera Term. Beachten Sie, dass die Einstellung „Port:“ davon abhängt, an welchen USB-Port Ihr Raspberry Pi Pico angeschlossen wurde. Der PicoMite ignoriert die Baudrateneinstellung, sodass er auf jede beliebige Geschwindigkeit eingestellt werden kann (außer 1200 Baud, wodurch der Pico in den Firmware-Upgrade-Modus versetzt wird). Wenn Sie Tera Term verwenden, legen Sie keine Verzögerung zwischen den Zeichen fest und wenn Sie Putty verwenden, legen Sie die Backspace-Taste fest um das BS-Zeichen zu erzeugen.



Die Konsole



Sobald Sie den virtuellen seriellen Port identifiziert und Ihren Terminalemulator damit verbunden haben, sollten Sie in der Lage sein, die Eingabetaste auf Ihrer Tastatur zu drücken und die MMBasic-Eingabeaufforderung zu sehen, die das Größer-als-Symbol ist (z. B. „>“).

Dies ist die Konsole und Sie verwenden sie, um Befehle zum Konfigurieren von PicoMite zu erteilen, das BASIC-Programm zu laden, es zu bearbeiten und auszuführen. MMBasic verwendet die Konsole auch, um Fehlermeldungen anzuzeigen.

Die Konsole ist die einzige Methode, um mit dem PicoMite zu kommunizieren und ihn zu programmieren, daher ist es wichtig, dass Sie sich damit verbinden können.

Einige Tests

Hier sind ein paar Dinge, die Sie ausprobieren können, um zu beweisen, dass Sie ein funktionierendes PicoMite haben.

Alle diese Befehle sollten an der Eingabeaufforderung (">") eingegeben werden. Was Sie eingeben, wird fett angezeigt und die Ausgabe von PicoMite wird in normalem Text angezeigt.

Versuchen Sie es mit einer einfachen Rechenaufgabe:

```
> PRINT 1/7
0.1428571429
```

Prüfen wieviel Speicher noch frei ist:

```
> MEMORY
Program:
   0K ( 0%) Program (0 lines)
  80K (100%) Free

RAM:
   0K ( 0%) 0 Variables
   0K ( 0%) General
 112K (100%) Free
```

Wie ist die aktuelle Uhrzeit? **Achtung: Die interne Uhr startet nach dem Einschalten um 00:00 Uhr.**

```
> PRINT TIME$
00:04:01
```

Einstellen der Uhrzeit:

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

Überprüfen der Uhrzeit:

```
> PRINT TIME$
10:45:09
```

Wie viele ms sind seit dem Einschalten vergangen:

```
> PRINT TIMER
 440782 .748
```

Bis 20 zählen:

```
> 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 und 8.1

Der serielle USB-Anschluss verwendet das CDC-Protokoll und die Treiber dafür sind Standard in Windows 10 und 11 und werden automatisch geladen. Die Raspberry Pi Foundation listet Windows 7 oder 8.1 als „nicht unterstützt“ auf, Sie können jedoch ein Tool wie Zadig (https://zadig.akeo.ie) verwenden, um einen generischen Treiber für ein „usbser“-Gerät zu installieren. Dieser Beitrag beschreibt den Vorgang: https://github.com/raspberrypi/pico-feedback/issues/118

Apple Macintosh

Der Apple Macintosh (OS X) ist etwas einfacher, da dort der Gerätetreiber und der Terminal-Emulator eingebaut sind. Starten Sie zuerst die Anwendung „Terminal“ und listen Sie bei der Eingabeaufforderung die angeschlossenen seriellen Geräte auf, indem Sie Folgendes eingeben:

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

Der USB-seriell-Konverter wird aufgeführt in der Art `/dev/tty.usbmodem12345`. Während Sie sich noch an der Terminal-Eingabeaufforderung befinden, können Sie den Terminal-Emulator mit 38400 Baud ausführen, indem Sie den folgenden Befehl verwenden:

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

Standardmäßig sind die Funktionstasten nicht korrekt für die Verwendung im integrierten Programmeditor von Micromite definiert, sodass Sie die Steuersequenzen verwenden müssen, wie sie im Abschnitt Vollbild-Editor dieses Handbuchs definiert sind. Um dies zu vermeiden, können Sie den Terminal-Emulator neu konfigurieren, um diese Codes zu generieren, wenn die entsprechenden Funktionstasten gedrückt werden.

Die Dokumentation für den Bildschirmbefehl finden Sie hier::
https://www.systutorials.com/docs/linux/man/1-screen/

Linux

Bitte beachten Sie diesen Link:

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

Schnellstart

Ein einfaches Programm

Um ein Programm einzugeben, können Sie den EDIT-Befehl verwenden, der später in diesem Handbuch beschrieben wird. Im Moment müssen Sie jedoch nur wissen, dass alles, was Sie eingeben, am Cursor eingefügt wird, die Pfeiltasten den Cursor bewegen und die Rücktaste das Zeichen vor dem Cursor löscht.

Um ein Gefühl zu bekommen wie PicoMite funktioniert, probieren Sie diese Sequenz aus (Ihr Terminal-Emulator muss VT100-kompatibel sein):

• Geben Sie an der Eingabeaufforderung EDIT gefolgt von der ENTER-Taste ein.

• Der Editor sollte starten und Sie können diese Zeile eingeben: PRINT "Hello World"

• Drücken Sie die F1-Taste in Ihrem Terminal-Emulator (oder STRG-Q, was dasselbe bewirkt). Dies weist den Editor an, Ihr Programm zu speichern und zur Eingabeaufforderung zurückzukehren.

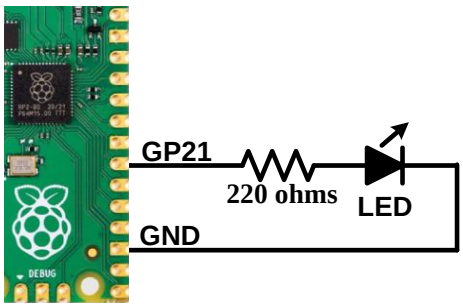
• Geben Sie an der Eingabeaufforderung AUSFÜHREN gefolgt von der EINGABETASTE ein.

• Sie sollten die Nachricht sehen: Hello World

Herzliche Glückwünsche. Sie haben gerade Ihr erstes Programm auf dem PicoMite geschrieben und ausgeführt. Wenn Sie erneut EDIT eingeben, gelangen Sie zurück in den Editor, wo Sie Ihr Programm ändern oder ergänzen können.

Eine LED zum Blinken bringen

Schließen Sie eine LED an Pin GP21 (auf der Unterseite der Platine markiert) und Masse an wie im Diagramm rechts gezeigt.
Verwenden Sie dann den EDIT-Befehl um das folgende Programm einzugeben:




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

Wenn Sie dieses Programm gespeichert und ausgeführt haben, sollten Sie von der blinkenden LED begrüßt werden. Es ist kein großartiges Programm, aber es soll lediglich der Veranschaulichung dienen. Die erste Zeile setzt Pin GP21 als Ausgang. Dann tritt das Programm in eine Endlosschleife ein, in der der Ausgang dieses Pins auf H gesetzt wird um die LED einzuschalten, gefolgt von einer kurzen Pause (300 Millisekunden). Der Ausgang wird dann auf L gesetzt, gefolgt von einer weiteren Pause. Das Programm wiederholt dann die Schleife. Wenn Sie es so belassen, bleibt der PicoMite für immer dort und die LED blinkt. Wenn Sie etwas ändern möchten (z. B. die Geschwindigkeit des Flashens) können Sie das Programm unterbrechen, indem Sie STRG-C auf der Konsole eingeben, und es dann nach Bedarf bearbeiten. Das ist der große Vorteil von MMBasic, es ist sehr einfach, ein Programm zu schreiben und zu ändern.

Wenn Sie möchten, dass dieses Programm bei jedem Einschalten automatisch gestartet wird, können Sie den folgenden Befehl verwenden:

```
OPTION AUTORUN ON
```

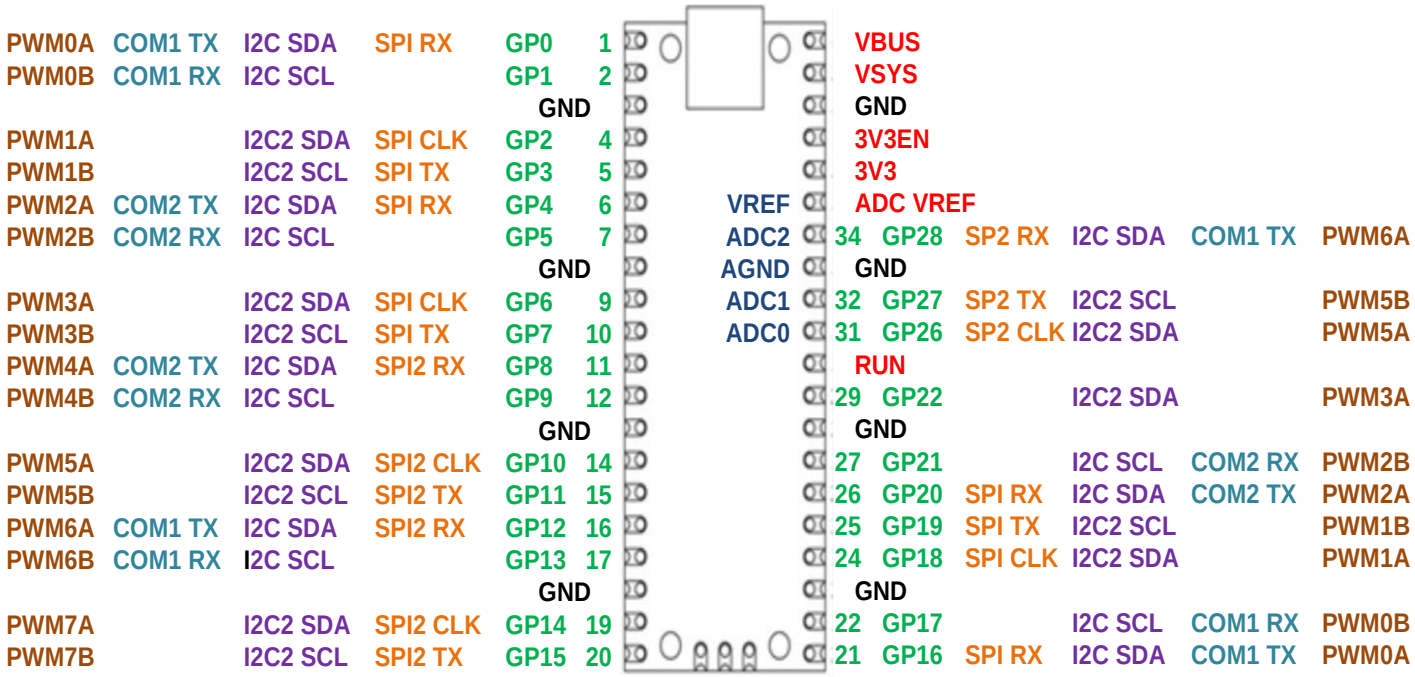
Um dies zu testen, können Sie die Stromversorgung entfernen und dann erneut anwenden. Der PicoMite sollte starten und die LED blinken lassen.

Tutorial BASIC-Programmierung

Wenn Sie neu im Bereich BASIC sind wäre jetzt ein guter Zeitpunkt, Anhang F (Programmieren in BASIC - Ein Tutorial) am Ende dieses Handbuchs aufzuschlagen. Dies ist ein umfassendes Tutorial zur Sprache, das Sie in einem leicht lesbaren Format mit vielen Beispielen durch die Grundlagen führt.

PicoMite-Hardware

Dieses Diagramm zeigt die Verwendungsmöglichkeiten der I/O-Pins auf dem RP Pico:



Notation:

GP0 bis GP28	Können als digitale Ein- oder Ausgänge verwendet werden..
COM1, COM2	Können für asynchrone serielle E/A verwendet werden.
I2C, I2C2	Können für die I²C Kommunication verwendet werden.
SPI, SPI2	Können für SPI I/O verwendet werden (siehe Anhang D).
PWMnx	Kann für die PWM-Ausgabe verwendet werden (siehe unter PWM).
GND	Masseanschluß.
VBUS	5V-Versorgung direkt vom USB-Anschluss.
VSYS	5-V-Versorgung, die vom SMPS verwendet wird, um 3,3 V bereitzustellen. Dieser kann als 5V Ausgang oder Eingang verwendet werden.
3V3EN	3.3V Spannungsregler (low = off, high = aktiv).
RUN	Reset-Pin, Low hält den Picomite im Reset-Modus.
ADCn	Analogeingänge zur Spannungsmessung.
ADC VREF	Referenzspannung für AD-Wandler.
AGND	Analoge Masse

Alle Pins können für digitale Ein- oder Ausgänge verwendet werden, sind jedoch auf eine max. Spannung von 3,6 V begrenzt. Wenn sie mit Geräten verwendet werden, die mit 5 V oder höher arbeiten ist ein Konverter zur Pegelanpassung erforderlich.

Innerhalb des MMBasic-Programms kann auf E/A-Pins unter Verwendung der physikalischen Pin-Nummer (z. B. 1 bis 40) oder der GP-Nummer (z. B. GP0 bis GP28) verwiesen werden. Zum Beispiel beziehen sich die folgenden auf denselben Pin und funktionieren identisch:

```
SETPIN 32, DOUT
```

und

```
SETPIN GP27, DOUT
```

Auf dem PicoMite sind On-Chip-Funktionen wie die SPI- und I2C-Schnittstellen nicht festen Pins zugeordnet, anders als (beispielsweise) beim Micromite. Der PicoMite nutzt den SETPIN-Befehl ausgiebig, nicht nur um I/O-Pins zu konfigurieren, sondern auch um die Pins zu konfigurieren, die für Schnittstellen wie seriell, SPI, I2C usw. verwendet werden. Pins müssen gemäß dieser Zeichnung zugeordnet werden. Beispielsweise kann der SPI TX den Pins GP3, GP7 oder GP19 zugewiesen werden, aber nicht dem Pin GP11, der nur dem SPI2-Kanal

zugewiesen werden kann. Zuweisungen müssen nicht im selben "Block" sein, sodass Sie beispielsweise SPI2 TX Pin GP11 und SPI2 RX Pin GP28 zuweisen können. Pins, die auf dem Raspberry Pi Pico nicht exponiert sind, können weiterhin mit MMBasic über eine Pseudo-Pin-Nummer oder ihre GPn-Nummer erreicht werden. Dadurch kann MMBasic auf anderen Modulen verwendet werden, die den RP2040-Prozessor verwenden. Diese versteckten Pins sind Pin 41 oder GP23, Pin 42 oder GP24, Pin 43 oder GP25 und Pin 44 oder GP29.

Auf dem Raspberry Pi Pico werden diese Pins für interne Funktionen wie folgt verwendet:

• Pin 41 oder GP23 ist ein digitaler Ausgang, der auf den Wert von OPTION POWER eingestellt ist. (EIN=PWM, AUS=PFM).

• Pin 42 oder GP24 ist ein digitaler Eingang, der hoch ist, wenn VBUS vorhanden ist.

• Pin 43 oder GP25 ist ebenfalls PWM4B. Es ist ein Ausgang, der mit der integrierten LED verbunden ist.

• Pin 44 oder GP29 ist auch ADC3, ein analoger Eingang, der ⅓ von VSYS liest.

I/O Pin Grenzwerte

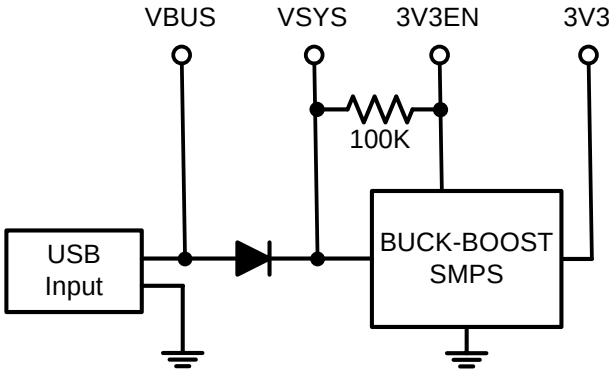
Die maximale Spannung, die an jeden I/O-Pin angelegt werden kann, beträgt 3,6 V.

Alle I/O-Pins können als Ausgang jeweils max. 12mA liefern oder ziehen. Bei max. Belastung sackt die Ausgangsspannung auf etwa 2,3 V ab. Praktikabel sind 5 mA, wobei die Ausgangsspannung typ. 3 V betragen würde. Um eine rote LED mit 5 mA anzusteuern, beträgt der empfohlene Widerstand 220 Ω. Andere Farben erfordern möglicherweise einen anderen Wert.

Die maximale Gesamt-I/O-Stromlast für den gesamten Chip beträgt 50 mA.

Stromversorgung

Der Raspberry Pi Pico verfügt über ein flexibles Stromversorgungssystem. Die Eingangsspannung von den USB- oder VBUS-Eingängen wird über eine Schottky-Diode mit dem Buck-Boost-SMPS (Switch Mode Power Supply) verbunden, das einen Ausgang von 3,3 V hat. Das SMPS ist für Eingangsspannungen von 1,8 V bis 5,5 V geeignet, sodass der PicoMite mit einer Vielzahl von Stromquellen, einschließlich Batterien, betrieben werden kann.



Externe Schaltungen können über VBUS (normalerweise 5 V) oder über den 3V3-Ausgang (3,3 V) mit Strom versorgt werden, der bis zu 300 mA liefern kann.

Für Embedded-Controller-Anwendungen ist im Allgemeinen eine externe Stromquelle (außer USB) erforderlich, die über eine Schottky-Diode mit VSYS verbunden werden kann. Dadurch kann der PicoMite mit der Stromversorgung betrieben werden, die die höchste Spannung erzeugt (USB oder VSYS). Die Dioden verhindern eine Rückkopplung in die niedrigere Spannungsversorgung.

Um das Rauschen der Stromversorgung zu minimieren, ist es möglich, 3V3EN zu erden, um das SMPS auszuschalten. Beim Herunterfahren hört der Wandler auf zu schalten, die interne Steuerschaltung wird ausgeschaltet und die Last vom Eingang getrennt. Sie können die Platine dann über einen 3,3-V-Linearregler mit Strom versorgen, der in den 3V3-Pin eingespeist wird.

Taktfrequenz

Standardmäßig beträgt die Taktfrequenz für den PicoMite 125 MHz und das empfohlene Maximum für den Raspberry Pi Pico beträgt 133 MHz.

Mit dem Befehl OPTION CPUSPEED kann die CPU jedoch auf bis zu 252 MHz übertaktet oder auf mind. 48 MHz langsamer betrieben werden. Nahezu alle getesteten Raspberry Pi Picos haben bei 250 MHz korrekt funktioniert, sodass eine Übertaktung sinnvoll sein kann.

Diese Option wird gespeichert und beim Einschalten wieder angewendet. Beim Ändern der Taktfrequenz wird der PicoMite zurückgesetzt und dann neu gestartet, sodass die USB-Verbindung getrennt wird.

Energieverbrauch

Der Stromverbrauch ist abhängig von der Taktrate. Dies sind typische Messwerte und beinhalten keinen Strom, der von den I/O-Pins oder dem 3V3-Pin bezogen oder aufgenommen wird:

250MHz	42mA
125MHz	20mA
48MHz	10mA.

Verwendung von MMBasic

Befehle und Programmeingabe

An der Eingabeaufforderung werden Befehle sofort ausgeführt. Üblicherweise starten Sie so ein Programm oder stellen eine Option ein aber Sie können auch Befehle testen. Die einfachste Methode zur Eingabe eines Programms ist die Verwendung des EDIT-Befehls. Dadurch wird der Vollbild-Programmeditor aufgerufen, der in PicoMite integriert ist und später in diesem Handbuch beschrieben wird. Es enthält erweiterte Funktionen wie Suchen und Kopieren, Ausschneiden und Einfügen in und aus einer Zwischenablage. Sie können das Programm auch auf Ihrem Desktop-Computer z.B. mit Notepad erstellen und es dann über das XModem-Protokoll (siehe XMODEM-Befehl) oder durch Streamen über die serielle Konsolenverbindung (siehe AUTOSAVE-Befehl) auf PicoMite übertragen. Eine dritte und praktische Methode zum Schreiben und Debuggen eines Programms ist die Verwendung von MMEdit. Dies ist ein Programm, das auf Ihrem Windows-Computer ausgeführt wird und Ihnen ermöglicht, Ihr Programm auf Ihrem Computer zu bearbeiten und es dann mit einem einzigen Mausklick auf PicoMite zu übertragen. MMEdit wurde von Jim Hiley geschrieben und kann kostenlos von https://www.c-com.com.au/MMedit.htm heruntergeladen werden.

Eine Sache, die Sie nicht tun können, ist die alte BASIC-Methode zur Eingabe eines Programms, bei der jeder Zeile eine Zeilennummer vorangestellt wurde. Zeilennummern sind in MMBasic optional, Sie können sie also weiterhin verwenden, wenn Sie möchten, **aber wenn Sie eine Zeile mit einer Zeilennummer am Prompt eingeben, wird MMBasic diese einfach sofort ausführen**.

.

Programmstruktur

Ein BASIC-Programm beginnt in der ersten Zeile und fährt fort, bis es am Ende des Programms abläuft oder auf einen END-Befehl trifft – an diesem Punkt zeigt MMBasic die Eingabeaufforderung (>) auf der Konsole an und wartet darauf, dass etwas eingegeben wird.

Ein Programm besteht aus einer Reihe von Anweisungen oder Befehlen, die jeweils den BASIC-Interpreter veranlassen, etwas zu tun (die Wörter Anweisung und Befehl bedeuten im Allgemeinen dasselbe und werden austauschbar verwendet). Normalerweise befindet sich jede Anweisung in einer eigenen Zeile, aber Sie können mehrere Anweisungen in einer Zeile haben, die durch den Doppelpunkt (:) getrennt sind.

Beispiel: `A = 24.6 : PRINT A`

Jede Zeile kann mit einer Zeilennummer beginnen. Zeilennummern waren in den frühen BASIC-Interpretern obligatorisch, moderne Implementierungen (wie MMBasic) benötigen sie jedoch nicht. Sie können sie immer noch verwenden, wenn Sie möchten, aber sie haben keinen Nutzen und überladen im Allgemeinen nur Ihre Programme. Dies ist ein Beispiel für ein Programm, das Zeilennummern verwendet:

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

Eine Zeile kann auch mit einem Label beginnen, das mit dem GOTO-Befehl als Ziel für einen Programmsprung verwendet werden kann. Zum Beispiel (der Labelname ist JmpBack):

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

Ein Label hat die gleichen Spezifikationen (Länge, Zeichensatz usw.) wie ein Variablenname, kann aber nicht mit einem Befehlsnamen identisch sein. Wenn es zur Beschriftung einer Zeile verwendet wird, muss die Beschriftung am Anfang einer Zeile, aber nach einer Zeilennummer (falls verwendet) erscheinen und mit einem Doppelpunkt (:) abgeschlossen werden.

Bearbeiten der Befehlszeile

Wenn Sie eine Zeile an der Eingabeaufforderung eingeben, kann die Zeile mit den Links- und Rechtspfeiltasten bearbeitet werden, um sich entlang der Zeile zu bewegen, der Entf-Taste, um ein Zeichen zu löschen, und der

Einfügetaste, um zwischen Einfügen und Überschreiben zu wechseln. An jedem Punkt wird die Eingabetaste die Zeile an MMBasic senden, das sie ausführt. Die Aufwärts- und Abwärtspfeiltasten bewegen sich durch eine Historie zuvor eingegebener Befehlszeilen, die bearbeitet und wiederverwendet werden können.

Funktionstasten

Die Funktionstasten auf der Tastatur oder der seriellen Konsole können an der Eingabeaufforderung verwendet werden, um allgemeine Befehle automatisch einzugeben (Ihr Terminal-Emulator muss VT100-kompatibel sein). Diese Funktionstasten fügen den Text gefolgt von der Eingabetaste ein, sodass der Befehl sofort ausgeführt wird:

F1	DATEIEN
F2	RUN
F3	AUFLISTEN
F4	BEARBEITUNG
F10	AUTOM. SPEICHERN
F11	XMODEM EMPFANG
F12	XMODEM SENDEN

Die Funktionstasten F5 bis F9 können mit benutzerdefiniertem Text programmiert werden. Siehe den Befehl OPTION FNKey.

Programme Speichern

Auf PicoMite (Version 5.05.03 oder höher) wird das Programm im Flash-Speicher gehalten und von dort ausgeführt. Wenn ein Programm über EDIT bearbeitet oder geladen wird, wird es dort gespeichert. Der Flash-Speicher ist nichtflüchtig, sodass das Programm nicht verloren geht, wenn die Stromversorgung unterbrochen oder der Prozessor zurückgesetzt wird. Die maximale Programmgröße beträgt 124 KB. Programme können auch auf einer SD-Karte (falls konfiguriert) oder auf einem der sieben nummerierten Speicherplätze (oder „Slots“) im Flash-Speicher gespeichert werden. Diese können verwendet werden, um frühere Versionen des Programms zu speichern (falls Sie zu einer früheren Version zurückkehren müssen) oder sie können verwendet werden, um völlig andere Programme zu speichern, die schnell in den Programmspeicher geladen und ausgeführt werden können.

Darüber hinaus ermöglicht MMBasic einem BASIC-Programm, ein anderes Programm zu laden und auszuführen, das an einem nummerierten Flash-Speicherort gespeichert ist, während alle Variablen und Einstellungen des ursprünglichen Programms beibehalten werden – dies wird als Verkettung bezeichnet und ermöglicht die Ausführung eines viel größeren Programms als die Anzahl von Programmspeicher normalerweise zulassen würde.):

FLASH SAVE *n*	Speichert das Programm im RAM an den Flash-Speicherort *n*.
FLASH LOAD *n*	Lädt ein Programm von der Flash-Position n in den RAM.
FLASH RUN *n*	Führt ein Programm von Flash-Speicherort n aus, löscht alle Variablen, löscht oder ändert jedoch nicht das Programm im Hauptprogrammspeicher.
FLASH LIST	Zeigt eine Liste der Inhalte aller 10 Flash-Speicherorte an.
FLASH ERASE *n*	Flash-Speicherort löschen *n*.
FLASH ERASE ALL	Löscht alle Flash-Speicherorte.
FLASH CHAIN *n*	Laden Sie ein Programm vom Flash-Speicherort n und führen Sie es aus, wobei alle Variablen intakt bleiben. Wie bei FLASH RUN löscht oder ändert dieser Befehl jedoch nicht das im Hauptprogrammspeicher gehaltene Programm.
FLASH OVERWRITE *n*	Löschen Sie den Flash-Speicherort n und speichern Sie dann das Programm im RAM an diesem Speicherort..

Zusätzlich kann mit dem Befehl OPTION AUTORUN ein Speicherort des Flash-Programms angegeben werden, der ausgeführt werden soll, wenn die Stromversorgung eingeschaltet oder die CPU neu gestartet wird. Diese Option kann auch ohne Angabe eines Flash-Speicherorts verwendet werden, und in diesem Fall lädt MMBasic automatisch das Programm, das sich im Programmspeicher befindet, und führt es aus.

Es wird dringend empfohlen, als erste Zeile des Programms einen Kommentar einzufügen, der das Programm beschreibt. Diese wird dann durch den Befehl FLASH LIST angezeigt und hilft bei der Identifizierung des Programms in der Auflistung.

Alle im Flash gespeicherten BASIC-Programme werden gelöscht, wenn Sie die PicoMite-Firmware upgraden (oder downgraden). **Stellen Sie also sicher, dass Sie diese zuerst sichern..**

Ein laufendes Programm unterbrechen

Ein Programm wird durch den RUN-Befehl zum Laufen gebracht. Sie können MMBasic und das laufende Programm jederzeit unterbrechen, indem Sie in der Konsoleneingabe STRG-C eingeben, und MMBasic kehrt zur Eingabeaufforderung zurück.

Optionen einstellen

Viele Optionen können mit Befehlen gesetzt werden, die mit dem Schlüsselwort OPTION beginnen. Sie sind in einem eigenen Abschnitt dieses Handbuchs aufgeführt. Sie können beispielsweise die CPU-Taktrate mit dem Befehl ändern:

```
OPTION CPUSPEED speed
```

Gespeicherte Variablen

Da der PicoMite nicht unbedingt über ein normales Speichersystem verfügt, muss er Daten speichern, die bei Wiederherstellung der Stromversorgung wiederhergestellt werden können. Dies kann mit dem Befehl VAR SAVE erfolgen, der die in seiner Befehlszeile aufgelisteten Variablen im nichtflüchtigen Flash-Speicher speichert. Der für gespeicherte Variablen reservierte Platz beträgt 16 KB. Diese Variablen können mit dem Befehl VAR RESTORE wiederhergestellt werden, der alle gespeicherten Variablen zur Variablentabelle des laufenden Programms hinzufügt. Normalerweise wird dieser Befehl am Anfang eines Programms platziert, damit die Variablen für die Verwendung durch das Programm bereit sind.

Diese Funktion dient zum Speichern von Kalibrierungsdaten, vom Benutzer ausgewählten Optionen und anderen Elementen, die sich selten ändern. Es sollte nicht für Hochgeschwindigkeitsspeicherungen verwendet werden, da Sie den Flash-Speicher verschleißen können. Der für den Raspberry Pi Pico verwendete Flash hat eine hohe Ausdauer, die aber durch ein Programm, das immer wieder Variablen speichert, übertroffen werden kann. Wenn Sie häufig Daten speichern möchten, sollten Sie einen Echtzeituhr-Chip hinzufügen. Die RTC-Befehle können dann zum Speichern und Abrufen von Daten aus dem batteriegepufferten Speicher der RTC verwendet werden. Siehe den RTC-Befehl für weitere Details.

Watchdog-Timer

Der PicoMite wird hauptsächlich als Embedded Controller verwendet. Es kann in MMBasic programmiert werden, und wenn das Programm fehlerfrei und bereit für den Praxiseinsatz ist, kann die Konfigurationseinstellung OPTION AUTORUN eingeschaltet werden. Das Modul startet automatisch das Programm sobald die Versorgungsspannung angelegt wird, und fungiert als benutzerdefinierte Schaltung. Es besteht aber die Möglichkeit, dass ein Programmierfehler zu einer Fehlermeldung von MMBasic führt d.h. man landet in der Eingabeaufforderung, die aber nur in der Terminalemulation sichtbar wäre. Eine andere Möglichkeit ist, dass das BASIC-Programm aus irgendeinem Grund in einer Endlosschleife hängen bleibt. In beiden Fällen wäre der sichtbare Effekt derselbe ... das Programm würde aufhören zu laufen, bis die Stromversorgung aus- und wieder eingeschaltet wurde. Zur Vorbeugung kann der Watchdog-Timer verwendet werden. Dies ist ein Timer, der bis Null herunterzählt und wenn er Null erreicht, wird der Prozessor automatisch neu gestartet (dasselbe wie beim ersten Anlegen der Stromversorgung), dies geschieht auch dann, wenn MMBasic in der Eingabeaufforderung “festhängt”. Nach dem Neustart wird die automatische Variable MM.WATCHDOG auf wahr gesetzt, um anzuzeigen, dass der Neustart durch ein Watchdog-Timeout verursacht wurde. Der WATCHDOG-Befehl sollte an strategischen Stellen im Programm platziert werden, um den Timer immer wieder zurückzusetzen und damit zu verhindern, dass er auf Null herunterzählt. Wenn dann ein Fehler auftritt, wird der Timer nicht zurückgesetzt, er zählt auf Null herunter und das Programm wird neu gestartet (vorausgesetzt, die Option AUTORUN ist eingestellt).

PIN-Sicherheit

Manchmal ist es wichtig die Daten und Programme in einem Embedded Controller vertraulich zu behandeln. In PicoMite kann dies mit dem Befehl OPTION PIN erfolgen. Dieser Befehl setzt eine PIN-Nummer (die im Flash gespeichert wird) und immer wenn PicoMite zur Eingabeaufforderung zurückkehrt (aus welchen Gründen auch immer), wird der Benutzer an der Konsole aufgefordert, die PIN-Nummer einzugeben. Ohne die

richtige PIN kann der Benutzer nicht zur Eingabeaufforderung gelangen und seine einzige Möglichkeit besteht darin, die richtige PIN einzugeben oder PicoMite neu zu starten. Wenn es neu gestartet wird, benötigt der Benutzer immer noch die richtige PIN um auf die Eingabeaufforderung zuzugreifen. Da ein Eindringling die Eingabeaufforderung nicht erreichen kann, kann er kein Programm auflisten oder kopieren, er kann das Programm nicht ändern oder einen Aspekt von MMBasic oder PicoMite ändern. Eine einmal festgelegte PIN kann nur entfernt werden, indem die korrekte PIN, wie sie ursprünglich festgelegt wurde, eingegeben wird. Wenn die Nummer verloren geht, besteht die einzige Wiederherstellungsmethode darin, die PicoMite-Firmware neu zu laden (wodurch das Programm und alle Optionen gelöscht werden). Es gibt zeitaufwändige Möglichkeiten, auf die Daten zuzugreifen z.B. die Verwendung eines Programmiergeräts zur Untersuchung des Flash-Speichers. Eine absolute Sicherheit ist somit nicht gegeben, aber das Verfahren wirkt abschreckend.

.

MM.STARTUP

Es kann erforderlich sein, beim ersten Einschalten Code auszuführen, vielleicht um Hardware zu initialisieren, einige Optionen einzustellen oder ein benutzerdefiniertes Startbanner zu drucken. Dies kann erreicht werden, indem ein Unterprogramm mit dem Namen MM.STARTUP erstellt wird. Wenn der Micromite zum ersten Mal eingeschaltet oder zurückgesetzt wird, sucht er nach dieser Unterroutine und, wenn sie gefunden wird, wird sie einmal ausgeführt. Wenn der Micromite beispielsweise eine Echtzeituhr angeschlossen hat, könnte das Programm den folgenden Code enthalten:

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

Dies würde dazu führen, dass die interne Uhr in MMBasic bei jedem Einschalten oder Zurücksetzen auf die aktuelle Zeit eingestellt wird. Nachdem der Code in MM.STARTUP ausgeführt wurde, fährt MMBasic damit fort, den Rest des Programms im Programmspeicher auszuführen. Wenn kein anderer Code vorhanden ist, kehrt MMBasic zur Eingabeaufforderung zurück.

Beachten Sie, dass Sie MM.STARTUP nicht für die allgemeine Einrichtung von MMBasic (wie das Dimensionieren von Arrays, das Öffnen von Kommunikationskanälen usw.) verwenden sollten, bevor Sie ein Programm ausführen. Der Grund dafür ist, dass MMBasic bei Verwendung des RUN-Befehls den Status des Interpreters für einen Neustart löscht.

MM.PROMPT

Wenn eine Unterroutine mit diesem Namen existiert, wird sie automatisch von MMBasic ausgeführt, anstatt die Eingabeaufforderung anzuzeigen. Dies kann verwendet werden, um eine benutzerdefinierte Eingabeaufforderung anzuzeigen, Farben festzulegen, Variablen zu definieren usw., die alle an der Eingabeaufforderung aktiv sind.

Beachten Sie, dass MMBasic alle Variablen und E/A-Pin-Einstellungen löscht, wenn ein Programm ausgeführt wird, sodass alles, was in dieser Unterroutine eingestellt ist, nur für Befehle gültig ist, die an der Eingabeaufforderung (d. h. im Direktmodus) eingegeben werden.

Als Beispiel wird im Folgenden eine benutzerdefinierte Eingabeaufforderung angezeigt:

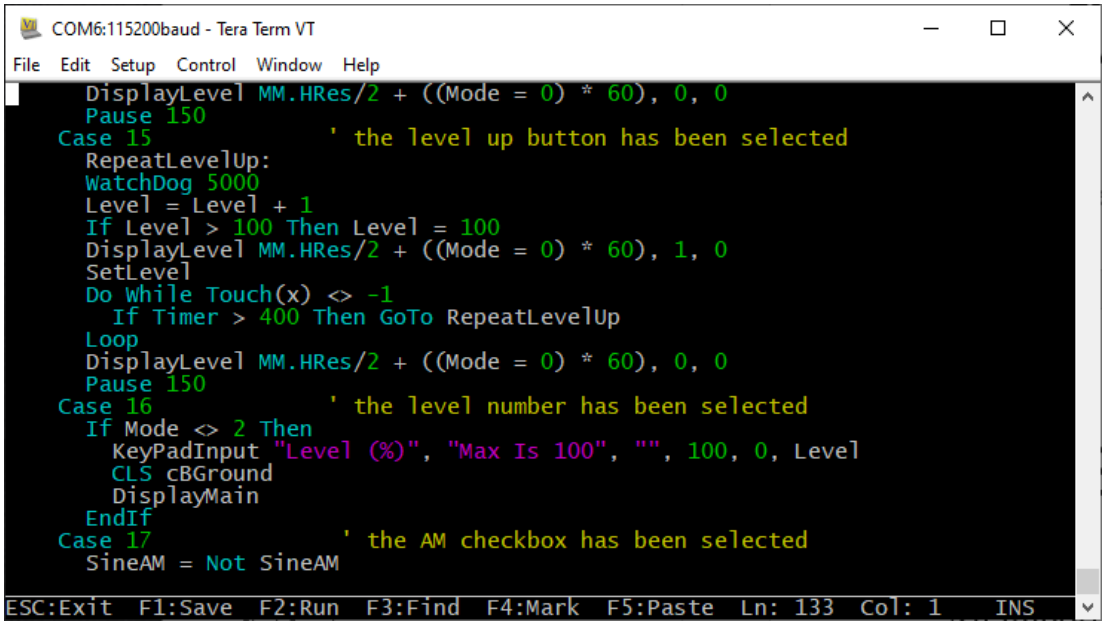
```
SUB MM.PROMPT
  PRINT TIME$ "> ";
END SUB
```

Beachten Sie, dass Konstanten zwar definiert werden können, aber nicht sichtbar sind, da eine in einer Subroutine definierte Konstante lokal für eine Subroutine ist. DIM erstellt jedoch globale Variablen, die stattdessen verwendet werden sollten.

.

Bildschirmeditor

Wichtig für die Produktivität bei der Programmierung ist der integrierte Bildschirmeditor. Hier eine Hardcopy:



Beim Start des Editors wird der Cursor automatisch an der Stelle positioniert, die Sie zuletzt bearbeitet haben Wurde Ihr Programm durch einen Fehler gestoppt wird der Cursor auf die Zeile positioniert, die den Fehler verursacht hat. Am unteren Rand des Bildschirms listet die Statuszeile Details wie die aktuelle Cursorposition und die allgemeinen Funktionen auf, die vom Editor unterstützt werden. Wenn Sie zuvor einen Editor wie Windows Notepad verwendet haben, werden Sie feststellen, dass die Bedienung dieses Editors vertraut ist. Die Pfeiltasten bewegen den Cursor im Text, Pos1 und Ende bringen Sie zum Anfang oder Ende der Zeile. Bild-auf und Bild-ab machen das, was ihre Titel suggerieren. Die Löschtaste löscht das Zeichen am Cursor und die Rücktaste löscht das Zeichen vor dem Cursor. Die Einfügetaste schaltet zwischen Einfüge- und Überschreibmodus um. Die einzige ungewöhnliche Tastenkombination ist, dass Sie durch zweimaliges Drücken der Home-Taste zum Anfang des Programms und durch zweimaliges Drücken der Ende-Taste zum Ende gelangen.

Die Befehlsreferenz des Editors:

ESC	Der Editor verwirft alle Änderungen und kehrt mit unveränderter Datei zur Eingabeaufforderung zurück. Wenn Sie den Text geändert haben, werden Sie aufgefordert, diese Aktion noch zweimal mit ESC zu bestätigen.
F1	Speichert die Datei und kehrt zurück zur Eingabeaufforderung.
F2	Speichert die Datei und führt sie sofort aus.
F3 or CTRL-F	Startet den Suchmodus. Sie werden nach dem Suchtext gefragt und während Sie diesen in den Editor eingeben, wird der Cursor automatisch auf den ersten gefundenen Text positioniert. Sie können dann die Pfeiltaste nach unten verwenden, um nach dem nächsten Vorkommen oder die Pfeiltaste nach oben nach dem vorherigen Vorkommen zu suchen. Die Eingabetaste lässt den Cursor an dieser Position und kehrt zum normalen Bearbeitungsmodus zurück. F5 oder STRG-V ersetzt den gesuchten Text mit dem, was sich in der Zwischenablage befindet (siehe unten). Escape bricht die Suche ab.
F4 or CTRL-S	Startet den Auswahlmodus. In diesem Modus können Sie mit den Pfeiltasten, HOME oder END Text auswählen und in die Zwischenablage kopieren. Es wird auf dem Bildschirm hervorgehoben, wenn Sie es auswählen. Dann kopiert F5 oder STRG-C die Auswahl in die Zwischenablage, F4 oder STRG-X kopiert und löscht die Auswahl. DELETE löscht einfach die Auswahl und ESCAPE kehrt zum normalen Bearbeitungsmodus zurück, ohne etwas zu ändern.
F5 or CTRL-V	Dies fügt den Text an der aktuellen Cursorposition ein der zuvor im Auswahlmodus (siehe oben) ausgeschnitten oder kopiert wurde.

END or CTRL-K	Bewegt den Cursor an das Ende der Zeile.
TAB	Bewegt den Cursor zur nächsten Tabulatorposition, wie durch OPTION TAB definiert.

Anstelle der oben aufgeführten Funktionstasten können Sie auch Steuertasten verwenden. Diese Steuertasten sind:

LINKS	Ctrl-S	RECHTS	Ctrl-D	AUF	Ctrl-E	AB	Ctrl-X
HOME	Ctrl-U	ENDE	Ctrl-K	PageUp	Ctrl-P	PageDn	Ctrl-L
DEL	Ctrl-]	EINFÜ	Ctrl-N	F1	Ctrl-Q	F2	Ctrl-W
F3	Ctrl-R	ShiftF3	Ctrl-G	F4	Ctrl-T	F5	Ctrl-Y

Wenn Sie Tera Term, Putty, MMEdit oder GFXterm als Terminal-Emulator verwenden können Sie den Cursor auch per Linksklick mit der PC-Maus im Fenster des Terminal-Emulators positionieren. Der beste Weg den Umgang mit dem Editor zu erlernen besteht darin ihn einfach zu starten und zu experimentieren. Der Editor ist eine sehr produktive Methode, um ein Programm zu schreiben. Mit dem Befehl EDIT können Sie Ihr Programm eingeben und durch Drücken der F2-Taste können Sie das Programm speichern und ausführen. Wenn Ihr Programm mit einem Fehler stoppt, wird durch Drücken der Funktionstaste F4 an der Eingabeaufforderung der Befehl BEARBEITEN ausgeführt und Sie kehren in den Editor zurück, wobei der Cursor auf der Zeile steht, die den Fehler verursacht hat. Dieser Edit/Run/Edit-Zyklus ist sehr schnell.

Bildschirm-Editor mit Farbcodierung

Dieser Editor kann den Programmtext mit Schlüsselwörtern, Zahlen und Kommentaren verschiedenfarbig kennzeichnen. Diese Funktion kann per Befehl ein- oder ausgeschaltet werden:

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

Diese Einstellung wird im nichtflüchtigen Speicher gespeichert und beim Start automatisch angewendet..

Variablen und Ausdrücke

In MMBasic wird bei Befehlsnamen, Funktionsnamen, Bezeichnungen, Variablennamen, Dateinamen usw. nicht zwischen Groß- und Kleinschreibung unterschieden, sodass „Run“ und „RUN“ gleichwertig sind und „dOO“ und „Doo“ auf dieselbe Variable verweisen.

Variablen

Variablen können mit einem Buchstaben oder Unterstrich beginnen und können beliebige Buchstaben oder Ziffern, den Punkt (.) und den Unterstrich (_) enthalten. Sie dürfen bis zu 32 Zeichen lang sein.Ein Variablenname oder ein Label darf nicht dasselbe sein wie eine Funktion oder eines der folgenden Schlüsselwörter: THEN, ELSE, GOTO, GOSUB, TO, STEP, FOR, WHILE, UNTIL, LOAD, MOD, NOT, AND, OR, XOR , WIE.

Z.B. step = 5 ist unzulässig, da STEP ein Schlüsselwort ist. MMBasic unterstützt drei Arten von Variablen:

1. Gleitkommazahl mit doppelter Genauigkeit.

Diese können eine Zahl mit Dezimalpunkt und Bruch (z. B. 45,386) speichern, verlieren jedoch an Genauigkeit, wenn mehr als 14 Stellen verwendet werden. Fließkommavariablen werden durch Anhängen des Suffixes '!' auf den Namen einer Variablen (z. B. i!, nbr! usw.). Sie sind auch die Standardeinstellung, wenn eine Variable ohne Suffix erstellt wird (z. B. i, nbr usw.).

2. 64-Bit-Ganzzahl mit Vorzeichen.

Diese können positive oder negative Zahlen mit bis zu 19 Dezimalstellen speichern, ohne an Genauigkeit zu verlieren, aber sie können keine Brüche speichern (d. h. den Teil nach dem Dezimalpunkt). Diese werden durch Hinzufügen des Suffix '%' an den Namen einer Variablen angegeben. Zum Beispiel i%, nbr% usw.

3. Eine Zeichenfolge.

Ein String speichert eine Folge von Zeichen (z. B. "Tom"). Jedes Zeichen in der Zeichenkette wird als Acht-Bit-Zahl gespeichert und kann daher einen Dezimalwert von 0 bis 255 haben. Zeichenketten-Variablennamen werden mit einem '$'-Symbol abgeschlossen (z. B. name$, s$ usw.). Zeichenfolgen können bis zu 255 Zeichen lang sein.

Es ist unzulässig denselben Variablennamen mit unterschiedlichen Typen zu verwenden. Z.B. mit nbr! und nbr% im selben Programm würde einen Fehler verursachen.

Die meisten Programme verwenden Fließkommavariablen für die Arithmetik, da diese mit den in typischen Situationen verwendeten Zahlen umgehen können und intuitiver sind als ganze Zahlen, wenn es um Divisionen und Brüche geht. Wenn Sie sich also nicht um die Details kümmern, verwenden Sie immer Gleitkommazahlen.

Konstanten

Numerische Konstanten können mit einer numerischen Ziffer (0-9) für eine Dezimalkonstante, &H für eine Hexadezimalkonstante, &O für eine Oktalkonstante oder &B für eine Binärkonstante beginnen. Zum Beispiel ist &B1000 dasselbe wie die Dezimalkonstante 8. Konstanten, die mit &H, &O oder &B beginnen, werden immer als vorzeichenlose 64-Bit-Ganzzahlkonstanten behandelt.

Dezimalkonstanten kann ein Minus (-) oder Plus (+) vorangestellt werden und mit einem „E“ abgeschlossen werden, gefolgt von einer Exponentenzahl, um die Exponentialschreibweise anzuzeigen. Zum Beispiel ist 1.6E+4 dasselbe wie 16000.

Wenn eine konstante Zahl verwendet wird, wird davon ausgegangen, dass es sich um eine Ganzzahl handelt, wenn kein Dezimalpunkt oder Exponent verwendet wird. Beispielsweise wird 1234 als Ganzzahl interpretiert, während 1234,0 als Fließkommazahl interpretiert wird.

Zeichenfolgenkonstanten müssen in doppelte Anführungszeichen (") eingeschlossen werden, z. B. "Hello World".

OPTION DEFAULT

Eine Variable kann ohne Suffix verwendet werden (d. h. !, % oder $) und in diesem Fall verwendet MMBasic den Standardtyp Fließkomma. Folgendes erstellt beispielsweise eine Fließkommavariable:

```
Nbr = 1234
```

Jedoch. die Standardeinstellung kann mit dem Befehl OPTION DEFAULT geändert werden. Beispielsweise gibt OPTION DEFAULT INTEGER an, dass alle Variablen ohne einen bestimmten Typ Integer sind. Folgendes erstellt also eine Integer-Variable:

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

Der Standardwert kann auf FLOAT (das ist der Standardwert, wenn ein Programm ausgeführt wird), INTEGER, STRING oder NONE gesetzt werden. In letzterem müssen alle Variablen explizit typisiert werden, sonst kommt es zu einem Fehler. Der Befehl OPTION DEFAULT kann überall im Programm platziert und jederzeit geändert werden, aber gute Praxis schreibt vor, dass er, wenn er verwendet wird, an den Anfang des Programms gestellt und unverändert gelassen werden sollte..

OPTION EXPLICIT

Standardmäßig erstellt MMBasic automatisch eine Variable, wenn sie zum ersten Mal referenziert wird. Nbr = 1234 erstellt also die Variable und setzt sie gleichzeitig auf die Nummer 1234. Dies ist praktisch für kurze und schnelle Programme, kann jedoch zu subtilen und schwer zu findenden Fehlern in großen Programmen führen. Beispielsweise wurde in der dritten Zeile dieses Fragments die Variable Nbr als Nbrs falsch geschrieben. Als Folge würde die Variable Nbrs mit einem Wert von Null angelegt und der Wert von Total wäre falsch.

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

Der Befehl OPTION EXPLICIT weist MMBasic an, Variablen nicht automatisch zu erstellen. Stattdessen müssen sie explizit mit den Befehlen DIM, LOCAL oder STATIC (siehe unten) definiert werden, bevor sie verwendet werden. Die Verwendung dieses Befehls wird empfohlen, um eine gute Programmierpraxis zu unterstützen. Wenn es verwendet wird, sollte es am Anfang des Programms platziert werden, bevor irgendwelche Variablen verwendet werden.

DIM und LOCAL

Die Befehle DIM und LOCAL können verwendet werden, um eine Variable zu definieren und ihren Typ festzulegen, und sind obligatorisch, wenn der Befehl OPTION EXPLICIT verwendet wird.

Der Befehl DIM erstellt eine globale Variable, die im gesamten Programm angezeigt und verwendet werden kann, einschließlich in Unterroutinen und Funktionen. Wenn die Definition jedoch nur innerhalb eines Unterprogramms oder einer Funktion sichtbar sein soll, sollten Sie den Befehl LOCAL am Anfang des Unterprogramms oder der Funktion verwenden. LOCAL hat genau dieselbe Syntax wie DIM.

Wenn LOCAL verwendet wird, um eine Variable mit demselben Namen wie eine globale Variable anzugeben, wird die globale Variable für die Subroutine oder Funktion verborgen, und alle Verweise auf die Variable beziehen sich nur auf die durch den LOCAL-Befehl definierte Variable. Jede von LOCAL erstellte Variable verschwindet, wenn das Programm die Subroutine verlässt.

Auf der einfachsten Ebene können DIM und LOCAL verwendet werden, um eine oder mehrere Variablen basierend auf ihrem Typsuffix oder der zu diesem Zeitpunkt gültigen OPTION DEFAULT zu definieren. Beispielsweise:

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

Es kann aber auch verwendet werden, um eine oder mehrere Variablen mit einem bestimmten Typ zu definieren, wenn das Typ-Suffix nicht verwendet wird:

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

In diesem Fall werden nbr, nbr2, nbr3 usw. alle als ganze Zahlen erstellt. Wenn Sie die Variable innerhalb eines Programms verwenden, brauchen Sie das Typsuffix nicht anzugeben. Zum Beispiel funktioniert MyStr im Folgenden perfekt als String-Variable:

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

Die Befehle DIM und LOCAL akzeptieren auch die Microsoft-Praxis, den Variablentyp nach der Variablen mit dem Schlüsselwort "AS" anzugeben. Beispielsweise:

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

In diesem Fall wird der Typ jeder Variablen einzeln festgelegt (nicht als Gruppe, wie wenn der Typ vor der Variablenliste steht). Die Variablen können auch während der Definition initialisiert werden. Beispielsweise:

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

Der zum Initialisieren der Variablen verwendete Wert kann ein Ausdruck sein, der benutzerdefinierte Funktionen enthält. Die Befehle DIM oder LOCAL werden auch zum Definieren eines Arrays verwendet, und alle oben aufgeführten Regeln gelten beim Definieren eines Arrays. Sie können zum Beispiel Folgendes verwenden:

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

Beim Initialisieren eines Arrays werden die Werte als kommagetrennte Werte aufgelistet, wobei die gesamte Liste in Klammern eingeschlossen ist. Beispielsweise:

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

oder

```
DIM days(7) AS STRING = ("","So","Mo","Die","Mi","Do","Fr","Sa")
```

STATIC

Innerhalb eines Unterprogramms oder einer Funktion ist es manchmal nützlich, eine Variable zu erstellen, die nur innerhalb des Unterprogramms oder der Funktion sichtbar ist (wie eine LOCAL-Variable), aber ihren Wert zwischen Aufrufen des Unterprogramms oder der Funktion beibehält.

Sie können dies tun, indem Sie den STATIC-Befehl verwenden. STATIC kann nur innerhalb einer Subroutine oder Funktion verwendet werden und verwendet die gleiche Syntax wie LOCAL und DIM. Der Unterschied besteht darin, dass sein Wert zwischen Aufrufen der Subroutine oder Funktion beibehalten wird (d. h. er wird beim zweiten und nachfolgenden Aufrufen nicht initialisiert).

Wenn Sie zum Beispiel die folgende Subroutine haben und sie wiederholt aufrufen, würde der erste Aufruf 5 ausgeben, der zweite 6, der dritte 7 und so weiter.

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

Beachten Sie, dass die Initialisierung der statischen Variablen auf 5 (wie im obigen Beispiel) nur beim ersten Aufruf des Unterprogramms wirksam wird. Bei weiteren Aufrufen wird die Initialisierung ignoriert, da die Variable bereits beim ersten Aufruf angelegt wurde.

Wie bei DIM und LOCAL können die mit STATIC erstellten Variablen Float, Integer oder Strings und Arrays davon mit oder ohne Initialisierung sein. Die Länge des von STATIC erstellten Variablennamens und die Länge des Subroutinen- oder Funktionsnamens zusammengenommen dürfen 32 Zeichen nicht überschreiten.

CONST

Oft ist es nützlich, einen Bezeichner zu definieren, der einen Wert darstellt ohne dass das Risiko besteht, dass der Wert versehentlich geändert wird – was passieren kann, wenn Variablen für diesen Zweck verwendet werden (diese Praxis fördert eine andere Klasse von schwer zu findenden Fehlern).

Mit dem Befehl CONST können Sie einen Bezeichner erstellen, der sich wie eine Variable verhält, aber auf einen Wert gesetzt ist, der nicht geändert werden kann. Beispielsweise:

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

Die Bezeichner können dann in einem Programm verwendet werden, wo sie für den gelegentlichen Leser sinnvoller sind als einfache Zahlen. Beispielsweise:

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

Auf einer Zeile können mehrere Konstanten erstellt werden:

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

Der zum Initialisieren der Konstante verwendete Wert wird ausgewertet, wenn die Konstante erstellt wird, und kann ein Ausdruck sein, der benutzerdefinierte Funktionen enthält.

Der Typ der Konstante ergibt sich aus dem ihr zugewiesenen Wert; so ist zum Beispiel MaxValue oben eine Fließkommakonstante, weil 2,4 eine Fließkommazahl ist. Der Typ einer Konstante kann auch explizit festgelegt werden, indem ein Typsuffix verwendet wird (z. B. !, % oder $), aber er muss mit seinem zugewiesenen Wert übereinstimmen.

Ausdrücke und Operatoren

MMBasic wertet einen mathematischen Ausdruck unter Verwendung der mathematischen Standardregeln aus. Zum Beispiel werden zuerst Multiplikation und Division durchgeführt, gefolgt von Addition und Subtraktion. Diese werden als Vorrangregeln bezeichnet und im Folgenden detailliert beschrieben. Das bedeutet, dass 2 + 3 * 6 zu 20 aufgelöst wird, ebenso 5 * 4 und auch 10 + 4 * 3 – 2.Wenn Sie den Interpreter zwingen möchten, zuerst Teile des Ausdrucks auszuwerten, können Sie diesen Teil des Ausdrucks in Klammern setzen. Beispiel: (10 + 4) * (3 – 2) ergibt 14 und nicht 20, wie es der Fall gewesen wäre, wenn die Klammern nicht verwendet worden wären. Die Verwendung von Klammern verlangsamt das Programm nicht merklich, daher sollten Sie sie großzügig verwenden, wenn die Möglichkeit besteht, dass MMBasic Ihre Absicht falsch interpretiert. Die folgenden Operatoren, in der Reihenfolge ihrer Priorität, sind in MMBasic implementiert. Operatoren auf derselben Ebene (z. B. + und -) werden mit Vorrang von links nach rechts verarbeitet, wenn sie in der Programmzeile auftreten.

.Arithmetic operators:

^	Potenzierung (z.B. `b^n` bedeutet b^n)
* / \ MOD	Multiplikation, Division, ganzz. Division und Modul (Rest)
+ -	Addition und Subtraktion

Shift operators:

x << y x >> y	<< bedeutet, dass der zurückgegebene Wert der Wert von x ist, der um y Bits nach links verschoben ist, während >> dasselbe bedeutet, nur nach rechts verschoben. Sie sind ganzzahlige Funktionen und alle weggeschobenen Bits werden verworfen. Bei einer Rechtsverschiebung werden alle eingefügten Bits auf den Wert des obersten Bits (Bit 63) gesetzt. Bei einer Linksverschiebung werden alle eingefügten Bits auf Null gesetzt

Logical operators:

NOT INV	den logischen Wert rechts invertieren (z.B. NOT a=b ist a<>b) oder bitweise Invertierung des Wertes rechts (z. B. a = INV b)
<> < > <= =< >= =>	Ungleichheit, kleiner als, größer als, kleiner oder gleich, kleiner oder gleich (alternative Version), größer als oder gleich, größer oder gleich (alternative Version)
=	Gleichheit
AND OR XOR	Konjunktion, Disjunktion, exklusives oder

Aus Gründen der Microsoft-Kompatibilität sind die Operatoren AND, OR und XOR ganzzahlige bitweise Operatoren. Beispielsweise gibt PRINT (3 AND 6) die Zahl 2 aus. Denn diese Operatoren können sowohl als logische Operatoren (z. B. IF a=5 AND b=8 THEN …) als auch als bitweise Operatoren (z. B. y% = x% fungieren. UND &B1010) wird der Interpreter verwirrt, wenn sie im selben Ausdruck gemischt werden. Werten Sie daher logische und bitweise Ausdrücke immer in separaten Ausdrücken aus.

Die anderen logischen Operationen ergeben die ganze Zahl 0 (Null) für falsch und 1 für wahr. Zum Beispiel die Aussage

PRINT 4 >= 5 druckt die Zahl Null auf der Ausgabe und der Ausdruck A = 3 > 2 speichert +1 in A.

Der NOT-Operator invertiert den logischen Wert auf seiner rechten Seite (es ist keine bitweise Umkehrung), während der INV-Operator eine bitweise Umkehrung durchführt. Beide haben die höchste Priorität, sodass sie eng an den nächsten Wert gebunden sind. Für die normale Verwendung von NOT oder INV sollte der zu

bearbeitende Ausdruck in Klammern gesetzt werden. z.B:

```
IF NOT (A = 3 OR A = 8) THEN …
```

Stringoperatoren:

+	Zwei Strings verbinden
<> < > <= =< >= =>	Ungleichheit, kleiner als, größer als, kleiner oder gleich, kleiner oder gleich (alternative Version), größer als oder gleich, größer oder gleich (alternative Version)
=	Gleichheit

Achtung: Zeichenfolgen-Vergleiche berücksichtigen die Groß-/Kleinschreibung ! Zum Beispiel ist "A" größer als "a".

Mischen von Gleitkomma- und Integerzahlen

MMBasic handhabt automatisch die Konvertierung von Zahlen zwischen Gleitkommazahlen und ganzen Zahlen. Wenn eine Operation sowohl Gleitkommazahlen als auch Ganzzahlen kombiniert (z. B. PRINT A% + B!), wird die Ganzzahl zuerst in eine Gleitkommazahl umgewandelt, dann die Operation ausgeführt und eine Gleitkommazahl zurückgegeben. Wenn beide Seiten des Operators Ganzzahlen sind wird eine Ganzzahloperation ausgeführt und eine Ganzzahl zurückgegeben.

Die einzige Ausnahme ist die normale Division ("/"), die immer beide Seiten des Ausdrucks in eine Fließkommazahl umwandelt und dann eine Fließkommazahl zurückgibt. Für die ganzzahlige Division sollten Sie den ganzzahligen Divisionsoperator "\" verwenden.

MMBasic-Funktionen geben abhängig von ihren Eigenschaften eine Float- oder Integer-Zahl zurück. Zum Beispiel gibt PIN() eine ganze Zahl zurück, wenn der Pin als digitaler Eingang konfiguriert ist, aber ein Float, wenn er als analoger Eingang konfiguriert ist.

Bei Bedarf können Sie mit der Funktion INT() einen Float in einen Integer umwandeln. Es ist nicht notwendig eine ganze Zahl speziell in eine Fließkommazahl umzuwandeln aber wenn es nötig wäre könnte der ganzzahlige Wert einer Fließkommavariable zugewiesen werden und er wird bei der Zuweisung automatisch umgewandelt..

64-bit Ganzzahlen ohne Vorzeichen

MMBasic auf PicoMite unterstützt 64-Bit-Ganzzahlen mit Vorzeichen. Dies bedeutet, dass es 63 Bits zum Halten der Zahl und ein Bit (das höchstwertige Bit) gibt, das verwendet wird, um das Vorzeichen (positiv oder negativ) anzugeben. Es ist jedoch möglich, vollständige 64-Bit-Zahlen ohne Vorzeichen zu verwenden, solange Sie keine Arithmetik mit den Zahlen durchführen.

64-Bit-Zahlen ohne Vorzeichen können mit den Präfixen &H, &O oder &B einer Zahl erstellt werden, und diese Zahlen können in einer Integer-Variablen gespeichert werden. Sie haben dann eine begrenzte Anzahl von Operationen, die Sie an diesen ausführen können. Sie sind << (nach links verschieben), >> (nach rechts verschieben), AND (bitweise und), OR (bitweises Oder), XOR (bitweises exklusives Oder), INV (bitweise Umkehrung), = (gleich) und <> (Nicht gleichzusetzen mit). Arithmetische Operatoren wie Division oder Addition können durch eine 64-Bit-Zahl ohne Vorzeichen verwechselt werden und unsinnige Ergebnisse liefern. Beachten Sie, dass die Verschiebung nach rechts eine vorzeichenbehaftete Operation ist. Das heißt, wenn das obere Bit eine Eins ist (eine negative vorzeichenbehaftete Zahl) und Sie nach rechts verschieben, wird es um Einsen verschoben, um das Vorzeichen beizubehalten.

Um 64-Bit-Zahlen ohne Vorzeichen anzuzeigen, sollten Sie die Funktionen HEX$(), OCT$() oder BIN$() verwenden.

Beispielsweise gibt die folgende 64-Bit-Operation ohne Vorzeichen die erwarteten Ergebnisse zurück:

```
X% = &HFFFF0000FFFF0044
Y% = &H800FFFFFFFFFFFF
X% = X% AND Y%
PRINT HEX$(X%, 16)
```

Zeigt "800F0000FFFF0044" an.

Unterprogramme und Funktionen

Eine programmdefinierte Subroutine oder Funktion ist einfach ein Block von Programmiercode, der in einem Modul enthalten ist und von überall innerhalb Ihres Programms aufgerufen werden kann. Es ist dasselbe, als ob Sie der Sprache Ihren eigenen Befehl oder Ihre eigene Funktion hinzugefügt hätten.

Unterprogramme

Eine Unterroutine verhält sich wie ein Befehl und kann Argumente haben (manchmal auch als Parameterliste bezeichnet). In der Definition des Unterprogramms sehen sie so aus::

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

Wenn Sie die Subroutine aufrufen, können Sie den Argumenten Werte zuweisen beispielsweise::

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

Innerhalb der Subroutine hat arg1 den Wert 23, arg2$ den Wert von „Cat“ und so weiter. Die Argumente verhalten sich wie gewöhnliche Variablen, aber sie existieren nur innerhalb der Subroutine und verschwinden, wenn die Subroutine endet. Sie können Variablen mit demselben Namen im Hauptprogramm haben und sie werden durch die für das Unterprogramm definierten Argumente ausgeblendet.

Wenn Sie ein Unterprogramm aufrufen können Sie weniger als die erforderliche Anzahl von Werten angeben, und in diesem Fall wird angenommen, dass die fehlenden Werte entweder Null oder eine leere Zeichenfolge sind. Sie können auch einen Wert in der Mitte der Liste weglassen und das Gleiche passiert, beispielsweise:

```
MYSUB  23, , 55
```

Führt dazu, dass arg2$ auf die leere Zeichenfolge "" gesetzt wird

Anstatt das Typ-Suffix zu verwenden (z. B. das $ in arg2$), können Sie das Suffix AS <Typ> in der Definition des Subroutinenarguments verwenden, und dann wird das Argument als der angegebene Typ bezeichnet, selbst wenn das Suffix nicht verwendet wird. Beispielsweise:

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

Innerhalb eines Unterprogramms können Sie eine Variable mit LOCAL definieren (das dieselbe Syntax wie DIM hat). Diese Variable existiert nur innerhalb der Subroutine und verschwindet, wenn die Subroutine beendet wird.

Funktionen

Funktionen ähneln Subroutinen. Hauptunterschied:Die Funktion wird verwendet um einen Wert in einem Ausdruck zurückzugeben. Die Regeln für die Argumentliste in einer Funktion sind ähnlich wie bei Unterroutinen. Der einzige Unterschied besteht darin dass Klammern um die Argumentliste herum erforderlich sind wenn Sie eine Funktion aufrufen selbst wenn keine Argumente vorhanden sind (sie sind optional wenn Sie eine Subroutine aufrufen).

Um einen Wert von der Funktion zurückzugeben, weisen Sie dem Namen der Funktion innerhalb der Funktion einen Wert zu. Wenn der Name der Funktion mit einem $, einem % oder einem ! die Funktion gibt diesen Typ zurück, andernfalls gibt sie zurück, was auch immer die OPTION DEFAULT eingestellt ist. Sie können den Typ der Funktion auch angeben, indem Sie am Ende der Funktionsdefinition AS <Typ> hinzufügen.

Beispielsweise:

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

Argumente per Referenz übergeben

Wenn Sie beim Aufrufen einer Subroutine oder Funktion eine gewöhnliche Variable d.h. keinen Ausdruck als Wert verwenden, zeigt das Argument innerhalb der Subroutine/ Funktion auf die beim Aufruf verwendete

Variable zurück und alle Änderungen am Argument werden ebenfalls an der gelieferten Variable vorgenommen. Dies wird als Übergabe von Argumenten per Referenz bezeichnet.

Beispielsweise könnten Sie eine Subroutine definieren, um zwei Werte wie folgt auszutauschen:

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

In Ihrem aufrufenden Programm würden Sie Variablen für beide Argumente verwenden:

```
Swap nbr1, nbr2
```

Und das Ergebnis wird sein, dass die Werte von nbr1 und nbr2 vertauscht werden.

Damit dies funktioniert, müssen der Typ der übergebenen Variablen (z. B. nbr1) und das definierte Argument (z. B. a) gleich sein (im obigen Beispiel sind beide standardmäßig Float).

Sofern Sie keinen Wert über das Argument zurückgeben müssen, sollten Sie ein Argument nicht als Allzweckvariable innerhalb einer Subroutine oder Funktion verwenden. Dies liegt daran, dass ein anderer Benutzer Ihrer Routine möglicherweise unwissentlich eine Variable in seinem Aufruf verwendet und diese Variable von Ihrer Routine "magisch" geändert werden könnte. Es ist viel sicherer, das Argument einer lokalen Variablen zuzuweisen und diese stattdessen zu manipulieren.

Übergabe von Arrays

Einzelne Elemente eines Arrays können an eine Subroutine oder Funktion übergeben werden und werden wie eine normale Variable behandelt. Dies ist beispielsweise eine gültige Methode zum Aufrufen der Swap-Subroutine (siehe oben):

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

Diese Art von Konstrukt wird häufig beim Sortieren von Arrays verwendet.

Sie können auch ein oder mehrere vollständige Arrays an eine Unterroutine oder Funktion übergeben, indem Sie das Array mit leeren Klammern anstelle der normalen Dimensionen angeben. Zum Beispiel a(). In der Unterprogramm- oder Funktionsdefinition muss der zugehörige Parameter ebenfalls mit leeren Klammern angegeben werden. Der Typ (d. h. Float, Integer oder String) des gelieferten Arguments und der Parameter in der Definition müssen identisch sein.

In der Subroutine oder Funktion erbt das Array die Dimensionen des übergebenen Arrays und diese müssen beim Indizieren in das Array berücksichtigt werden. Falls erforderlich, könnten die Dimensionen des Arrays als zusätzliche Argumente an die Subroutine oder Funktion übergeben werden, damit sie das Array korrekt manipulieren könnte. Das Array wird als Referenz übergeben, was bedeutet, dass alle Änderungen, die innerhalb der Subroutine oder Funktion am Array vorgenommen werden, auch für das bereitgestellte Array gelten.

Wenn beispielsweise Folgendes ausgeführt wird, werden die Worte „Hello World“ ausgedruckt:

```
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
```

Früherer Exit

Für jede Definition eines Unterprogramms oder einer Funktion kann es nur ein END SUB oder END FUNCTION geben. Um ein Unterprogramm vorzeitig zu verlassen (d. h. bevor der END SUB-Befehl erreicht wurde), können Sie den EXIT SUB-Befehl verwenden. Dies hat den gleichen Effekt, als ob das Programm die Anweisung END SUB erreicht hätte. Ebenso können Sie EXIT FUNCTION verwenden, um eine Funktion vorzeitig zu verlassen..

Rekursion

Bei Rekursion ruft sich eine Subroutine oder Funktion selbst auf. Sie können Rekursion in MMBasic durchführen, aber es gibt eine Reihe von Problemen (diese sind eine direkte Folge der Einschränkungen von Mikrocontrollern und der BASIC-Sprache):

- Es gibt eine feste Grenze für die Rekursionstiefe. Beim PicoMite sind das 50 Stufen.
- Wenn Sie viele Argumente für das Unterprogramm oder die Funktion und viele LOKALE Variablen (insbesondere Zeichenfolgen) haben, könnte Ihnen leicht der Speicher ausgehen, bevor Sie die Grenze von 50 Ebenen erreichen.

  Alle FOR…NEXT-Schleifen und DO…LOOPs werden beschädigt, wenn die Subroutine oder Funktion rekursiv aus diesen Schleifen heraus aufgerufen wird.

Beispiele

Es besteht oft die Notwendigkeit, einen speziellen Befehl oder eine spezielle Funktion in MMBasic zu implementieren, aber in vielen Fällen können diese mithilfe einer gewöhnlichen Subroutine oder Funktion erstellt werden die dann genauso wie ein eingebauter Befehl oder eine eingebaute Funktion funktioniert.

Beispielsweise ist manchmal eine TRIM-Funktion erforderlich, die bestimmte Zeichen vom Anfang und Ende einer Zeichenfolge entfernt. Im Folgenden finden Sie ein Beispiel dafür, wie eine solche einfache Funktion in MMBasic erstellt wird.

Das erste Argument der Funktion ist die zu kürzende Zeichenkette und das zweite ist eine Zeichenkette, die die Zeichen enthält, die aus der ersten Zeichenkette gekürzt werden sollen. RTrim$() schneidet die angegebenen Zeichen vom Ende des Strings ab, LTrim$() vom Anfang und Trim$() von beiden Enden.

```
' Alle Zeichen in c$ vom Anfang und Ende von s$ abschneiden
Function Trim$(s$, c$)
  Trim$ = RTrim$(LTrim$(s$, c$), c$)
End Function

' Schneiden Sie alle Zeichen in c$ vom Ende von s$ ab
Function RTrim$(s$, c$)
  RTrim$ = s$
  Do While Instr(c$, Right$(RTrim$, 1))
    RTrim$ = Mid$(RTrim$, 1, Len(RTrim$) - 1)
  Loop
End Function

' Alle Zeichen in c$ ab dem Anfang von s$ kürzen
Function LTrim$(s$, c$)
  LTrim$ = s$
  Do While Instr(c$, Left$(LTrim$, 1))
    LTrim$ = Mid$(LTrim$, 2)
  Loop
End Function
```

Anwendungsbeispiel:

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

ergibt "****23.56700"

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

ergibt "23.567"

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

ergibt "23.56700"

Verwendung der I/O-Pins

Der Raspberry Pi Pico verfügt über 26 Eingangs-/Ausgangspins, die vom BASIC-Programm aus gesteuert werden können, wobei 3 davon einen Hochgeschwindigkeits-ADC (Analog-Digital-Wandler) unterstützen.

Auf einen I/O-Pin wird durch seine Pin-Nummer verwiesen, und dies kann die Nummer (z. B. 2) oder seine GP-Nummer (z. B. GP1) sein.

Digitale Eingänge

Ein digitaler Eingang ist die einfachste Art der Eingangskonfiguration. Wenn die Eingangsspannung höher als 2,5 V ist, ist der Logikpegel wahr (numerischer Wert von 1) und alles unter 0,65 V ist falsch (numerischer Wert von 0). Die Eingänge verwenden einen Schmitt-Trigger-Eingang, sodass alles zwischen diesen Pegeln den vorherigen Logikpegel behält. Alle Pins sind auf eine maximale Spannung von 3,6 V begrenzt. Dies bedeutet, dass Widerstandsteiler erforderlich sind, wenn sie mit höheren Eingangsspannungen verwendet werden.

In Ihrem BASIC-Programm würden Sie den Eingang als digitalen Eingang einstellen und die PIN()-Funktion verwenden, um seinen Pegel zu erhalten. Beispiel:

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

Der SETPIN-Befehl konfiguriert Pin GP4 als digitalen Eingang und die Funktion PIN() gibt den Wert dieses Pins zurück (die Zahl 1, wenn der Pin hoch ist). Der IF-Befehl führt dann den Befehl nach der THEN-Anweisung aus, wenn der Eingang hoch war. Wenn der Eingangsstift niedrig war, würde das Programm einfach mit der nächsten Zeile im Programm fortfahren.

Der SETPIN-Befehl erkennt auch einige Optionen, die einen internen Widerstand vom Eingang entweder mit der Versorgung oder Masse verbinden. Dies wird als "Pullup"- oder "Pulldown"-Widerstand bezeichnet und ist praktisch beim Anschließen an einen Schalter, da es die Installation eines externen Widerstands erspart, um eine Spannung über die Kontakte zu legen.

Analoge Eingänge

Als ADC markierte Pins können so konfiguriert werden, dass sie die Spannung am Pin messen. Der Eingangsbereich reicht von null bis 3,3 V und die Funktion PIN() gibt die Spannung zurück. Beispielsweise:

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

Wenn Sie Spannungen über 3,3 V messen möchten, benötigen Sie einen Spannungsteiler. Bei kleinen Spannungen benötigen Sie möglicherweise einen Verstärker, um die Eingangsspannung in einen für die Messung angemessenen Bereich zu bringen. Die Messung verwendet die 3,3-V-Stromversorgung der CPU als Referenz, und es wird angenommen, dass dies genau 3,3 V sind. Dieser Wert kann mit dem OPTION-Befehl geändert werden. Die ADC-Befehle bieten eine alternative Methode zur Aufzeichnung analoger Eingänge und sind für die Hochgeschwindigkeitsaufzeichnung vieler Messwerte in einem Array vorgesehen.

Zähler-Eingänge

Alle vier Pins können als Zähleingänge verwendet werden, um Frequenz, Periode oder einfach nur Impulse am Eingang zu zählen. Die für diese Funktion verwendeten Pins können mit dem Befehl OPTION COUNT konfiguriert werden, werden aber, wenn sie nicht geändert werden, standardmäßig auf GP6, GP7, GP8 und GP9 gesetzt.

Als Beispiel wird im Folgenden die Frequenz des Signals auf Pin GP7 angezeigt:

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

In diesem Fall beträgt die Frequenz 110,374 kHz. Standardmäßig beträgt die Gatterzeit eine Sekunde, was der Zeitdauer entspricht, die MMBasic verwendet, um die Anzahl der Zyklen am Eingang zu zählen. Das bedeutet, dass der Messwert einmal pro Sekunde mit einer Auflösung von 1 Hz aktualisiert wird. Durch Angabe eines dritten Arguments zum SETPIN-Befehl kann eine alternative Torzeit zwischen 10 ms und 100000 ms angegeben werden. Kürzere Zeiten führen dazu, dass die Messwerte häufiger aktualisiert werden als der Wert zurückgegeben wird eine niedrigere Auflösung haben. Die Funktion PIN() skaliert die zurückgegebene Zahl immer als Frequenz in Hz, unabhängig von der verwendeten Torzeit.

Folgendes setzt beispielsweise die Gate-Zeit auf 10 ms mit entsprechendem Auflösungsverlust:

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

Für die genaue Messung von Signalen unter 10 Hz ist es im Allgemeinen besser, die Periode des Signals zu messen. In diesem Modus misst der PicoMite die Anzahl der Millisekunden zwischen aufeinanderfolgenden ansteigenden Flanken des Eingangssignals. Der Wert wird beim Übergang von niedrig nach hoch aktualisiert. Wenn Ihr Signal also eine Periode von (sagen wir) 100 Sekunden hat, sollten Sie darauf vorbereitet sein, diese Zeit zu warten, bevor die Funktion PIN() einen aktualisierten Wert zurückgibt. Die Zähler können auch die Anzahl der Impulse an ihrem Eingang zählen. Wenn ein Pin als Zähler konfiguriert ist (z. B. SETPIN 7, CIN), wird der Zähler auf Null zurückgesetzt und PicoMite zählt dann jeden Übergang von einer niedrigen zu einer hohen Spannung. Durch Ausführen von PIN(7) = 0 kann der Zähler wieder auf Null zurückgesetzt werden.

Digitale Ausgänge

Alle I/O-Pins können als Digitalausgang konfiguriert werden. Das bedeutet, wenn ein Ausgang auf LOW gesetzt ist, zieht er seinen Ausgang auf Null und wenn er hoch gesetzt ist, zieht er seinen Ausgang auf 3,3 V. In MMBasic geschieht dies mit dem PIN-Befehl. Zum Beispiel wird PIN(GP15) = 0 den Pin GP15 auf Low setzen, während PIN(GP15) = 1 ihn auf High setzen wird.

Die Option "OC" im SETPIN-Befehl macht den Ausgangspin zum offenen Kollektor. Dies bedeutet, dass der Ausgangstreiber den Ausgang auf Low (auf null Volt) zieht, wenn der Ausgang auf logisch Low gesetzt ist, aber in einen hochohmigen Zustand übergeht, wenn er auf logisch High gesetzt ist. Beachten Sie, dass die maximale Spannung an einem Pin 3,6 V beträgt, sodass Open-Collector-Ausgänge nicht verwendet werden können, um eine Logik mit höherer Spannung (z. B. 5 V) zu treiben.

PWM

Der PWM-Befehl (Pulsweitenmodulation) ermöglicht es dem PicoMite, Rechteckwellen mit einem programmgesteuerten Arbeitszyklus zu erzeugen. Durch Variieren des Arbeitszyklus können Sie einen programmgesteuerten Spannungsausgang zur Verwendung bei der Steuerung externer Geräte erzeugen, die einen analogen Eingang benötigen (Netzteile, Motorsteuerungen usw.). Die PWM-Ausgänge sind auch nützlich zum Ansteuern von Servos und zum Erzeugen einer Tonausgabe über einen kleinen Wandler.

Die PWM-Ausgänge bestehen aus bis zu 8 Kanälen (nummeriert von 0 bis 7), wobei jeder Kanal zwei Ausgänge (A und B) hat. Für jeden Kanal kann die Frequenz gewählt und für jeden Ausgang ein anderes Tastverhältnis eingestellt werden.

Mit dem SETPIN-Befehl können bis zu 16 Pins als PWM-Ausgänge konfiguriert werden.

Kommunikations-Schnittstellen (Seriell, SPI und I^2C)

werden im Anhang am Ende dieses Handbuchs beschrieben. Bevor diese Schnittstellen verwendbar sind müssen die Pins die verwendet werden sollen mit dem SETPIN-Befehl konfiguriert werden.

Einige Geräte wie SD-Karten, LCD-Panels, Touch usw. verwenden auch SPI- oder I2C-Schnittstellen, und die dafür verwendeten Pins müssen ebenfalls mit dem Befehl OPTION SYSTEM konfiguriert werden, bevor sie verwendet werden können

Interrupts

Unterbrechungen sind eine praktische Möglichkeit, mit einem Ereignis umzugehen, das zu einem unvorhersehbaren Zeitpunkt eintreten kann. Ein Beispiel ist, wenn der Benutzer eine Taste drückt. In Ihrem Programm könnten Sie nach jeder Anweisung Code einfügen, um zu überprüfen, ob die Taste gedrückt wurde aber ein Interrupt sorgt für ein saubereres und besser lesbares Programm.

Wenn ein Interrupt auftritt, führt MMBasic eine spezielle Unterroutine aus und kehrt nach Beendigung zum Hauptprogramm zurück. Das Hauptprogramm ist sich des Interrupts überhaupt nicht bewusst und wird normal weitermachen.

Jeder E/A-Pin, der als digitaler Eingang verwendet werden kann, kann so konfiguriert werden, dass er mit dem Befehl SETPIN einen Interrupt generiert, wobei bis zu zehn Interrupts gleichzeitig aktiv sind. Interrupts können so eingerichtet werden dass sie bei einem steigenden oder fallenden digitalen Eingangssignal (oder beidem) auftreten und eine sofortige Verzweigung zur angegebenen benutzerdefinierten Subroutine bewirken. Das Ziel kann für jeden Interrupt gleich oder unterschiedlich sein. Die Rückkehr von einem Interrupt erfolgt über die Befehle END SUB oder EXIT SUB. Beachten Sie, dass keine Parameter an die Subroutine übergeben werden können, aber innerhalb des Interrupts Aufrufe an andere Subroutinen und Funktionen erlaubt sind.

Treten zwei oder mehr Interrupts gleichzeitig auf, werden sie in der mit SETPIN definierten Reihenfolge der Interrupts abgearbeitet. Während der Bearbeitung eines Interrupts sind alle anderen Interrupts gesperrt bis das Interrupt-Unterprogramm zurückkehrt. Während eines Interrupts (und jederzeit) kann mit der Funktion PIN() auf den Wert des Interrupt-Pins zugegriffen werden.

Interrupts können jederzeit auftreten, sie werden jedoch während INPUT-Anweisungen deaktiviert. Auch Interrupts werden während einiger langer hardwarebezogener Operationen (z. B. der TEMPR()-Funktion) nicht erkannt, obwohl sie erkannt werden, wenn sie nach Abschluss der Operation noch vorhanden sind. Bei der Verwendung von Interrupts ist das Hauptprogramm von der Interrupt-Aktivität völlig unbeeinflusst, es sei denn, eine vom Hauptprogramm verwendete Variable wird während des Interrupts geändert.

Da Interrupts im Hintergrund laufen, können sie schwer zu diagnostizierende Fehler verursachen. Beachten Sie bei der Verwendung von Interrupts die folgenden Faktoren:

• Bei den meisten Programmen reagiert MMBasic auf einen Interrupt in weniger als 15 µs, jedoch können einige Befehle (wie die TEMPR()-Funktion) Interrupts für bis zu 200 ms blockieren, sodass es möglich ist, dass ein Interrupt (zB ein Tastendruck) auftritt und innerhalb dieser Zeit verschwindet dieses Fenster und wird in nicht erkannt.

• Wenn Sie sich in einem Interrupt befinden, werden alle anderen Interrupts blockiert, daher sollten Ihre Interrupts kurz sein und so schnell wie möglich beendet werden. Verwenden Sie beispielsweise niemals PAUSE innerhalb eines Interrupts. Wenn Sie eine langwierige Verarbeitung zu erledigen haben, sollten Sie einfach ein Flag setzen und den Interrupt sofort verlassen, dann kann Ihre Hauptprogrammschleife das Flag erkennen und alles tun, was erforderlich ist.

• Die Subroutine, die der Interrupt aufruft (und alle anderen Subroutinen oder Funktionen, die von ihm aufgerufen werden), sollte immer exklusiv für den Interrupt sein. Wenn Sie ein Unterprogramm aufrufen müssen, das auch von einem Interrupt verwendet wird, müssen Sie den Interrupt zuerst deaktivieren (Sie können ihn wieder aktivieren, nachdem Sie mit dem Unterprogramm fertig sind).

• Denken Sie daran, einen Interrupt zu deaktivieren, wenn Sie ihn nicht mehr benötigen – Hintergrund-Interrupts können seltsame und nicht intuitive Fehler verursachen.

Zusätzlich zu Interrupts, die durch die Zustandsänderung eines I/O-Pins erzeugt werden, kann ein Interrupt auch von anderen Abschnitten von MMBasic erzeugt werden, einschließlich Zeitgebern und Kommunikationsports und dem Obigen.

Tonausgabe

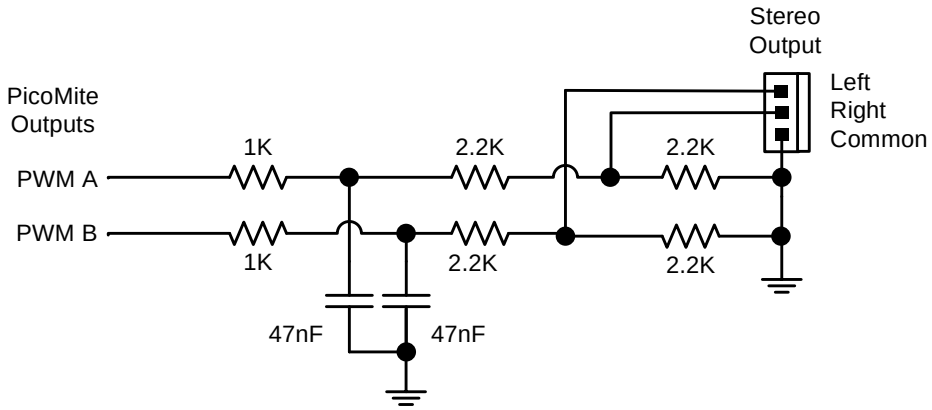
Der PicoMite kann Stereo-WAV-Dateien abspielen, die sich auf der SD-Karte befinden, oder mit dem PLAY-Befehl präzise Sinuswellen erzeugen.

Bevor diese Befehle verwendet werden können, müssen die zu verwendenden Ausgangspins als Audioausgänge zugewiesen werden. Dies geschieht mit dem Befehl OPTION AUDIO. Beispielsweise:

```
OPTION AUDIO GP0, GP1
```

konfiguriert Pin GP0 als linken Kanal und GP1 als rechten Kanal.

Das Audiosignal wird als pulsweitenmoduliertes (PWM) Signal einer Rechteckwelle überlagert. Dies bedeutet, dass ein Tiefpassfilter erforderlich ist, um das Audiosignal wie unten gezeigt wiederherzustellen. Dies ist ein einfaches Beispiel das auf einer kapazitiven Kopplung im Verstärker beruht (die meisten haben dies). Es hat einen Ausgangspegel von etwa 1 Vss d.h. 650 mV RMS.



Dies ist für den allgemeinen Gebrauch geeignet jedoch können bei Bedarf ausgefeiltere Designs implementiert werden um den Frequenzgang zu verbessern und mehr von der Trägerfrequenz zu unterdrücken. Diese Schaltung eignet sich auch zur Erzeugung eines DC-Ausgangssignals unter Verwendung der PWM-Befehle, obwohl in diesem Fall beide 47-nF-Kondensatoren auf 4,7 µF erhöht werden sollten.

Abspielen von WAV Dateien

Der PLAY-Befehl spielt eine Audiodatei, die sich auf einer SD-Karte befindet, an der Tonausgabe ab. Es kann verwendet werden, um Hintergrundmusik bereitzustellen, Soundeffekte zu Programmen hinzuzufügen und informative Durchsagen bereitzustellen.

Befehlssyntax:

```
PLAY WAV file$, interrupt
```

file$ ist der Name der abzuspielenden Audiodatei. Sie muss sich auf der SD-Karte befinden und die entsprechende Erweiterung (z. B. .WAV) wird angehängt, falls sie fehlt. Der Ton wird im Hintergrund abgespielt (dh das Programm wird ohne Pause fortgesetzt). interrupt ist optional und ist der Name einer Unterroutine, die aufgerufen wird, wenn die Wiedergabe der Datei beendet ist.

Erzeugung von Sinussignalen

Der Befehl PLAY TONE verwendet den Audioausgang, um Sinuswellen mit wählbaren Frequenzen für den linken und rechten Kanal zu erzeugen. Das ist dafür gedacht Aufmerksamkeits zu erzeugen, aber da die Frequenz sehr genau ist, kann sie für viele andere Anwendungen verwendet werden. Zum Beispiel das Signalisieren von DTMF-Tönen einer Telefonleitung oder das Testen des Frequenzgangs von Lautsprechern.

Befehlssyntax:

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

links und rechts sind die Frequenzen in Hz, die für den linken und rechten Kanal verwendet werden. Der Ton spielt im Hintergrund (das Programm läuft nach diesem Befehl weiter) und 'dur' gibt die Anzahl der Millisekunden an, die der Ton ertönen soll.

Die Dauer ist optional und wenn nicht angegeben wird der Ton fortgesetzt, bis er explizit gestoppt oder das Programm beendet wird. Interrupt (falls angegeben) wird ausgelöst, wenn die Dauer abgelaufen ist.

Die Frequenz kann zwischen 1 Hz und 20 KHz liegen und ist sehr genau (sie basiert auf einem Kristalloszillator). Die Frequenz kann jederzeit geändert werden, indem ein neuer PLAY TONE-Befehl ausgegeben wird. Beachten Sie, dass die Sinuswelle erzeugt wird durch schrittweises Durchlaufen einer

LookUp-Tabelle. Die Audioausgabe sollte daher durch einen Tiefpassfilter geleitet werden um Verzerrungen zu reduzieren..

Spezielle Audioausgabe

Der PLAY SOUND-Befehl erzeugt eine Ausgabe basierend auf einer Mischung aus Sinus-, Rechteck- usw. Wellenformen. Siehe die Details in der Befehlsliste..

Verwendung von PLAY

Es ist wichtig zu wissen, dass der PLAY-Befehl das Audio im Hintergrund erzeugt. Dadurch kann ein Programm etwa den Klang einer Glocke spielen während es mit seiner Steuerfunktion fortfährt. Ohne die Hintergrundfunktion würde das gesamte BASIC-Programm quasi einfrieren während der Ton zu hören ist.

Das Generieren des Tons im Hintergrund kann jedoch seltsame Effekte zeigen. Nehmen Sie zum Beispiel das folgende Programm:

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

Man könnte normal erwarten, dass der 500-Hz-Ton 2 Sekunden lang ertönt, aber in der Praxis wird überhaupt keinen Ton zuhören sein. Dies liegt daran, dass MMBasic den PLAY TONE-Befehl ausführt (der mit der Generierung des Tons im Hintergrund beginnt) und dann sofort den END-Befehl ausführt, der das Programm und den Hintergrundton beendet. Dies geschieht so schnell, dass nichts zu hören ist.

Auch das folgende Programm funktioniert nicht:

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

Dies liegt daran, dass der erste Befehl einen 500-Hz-Ton einstellt, aber der zweite PLAY-Befehl diesen sofort durch einen 300-Hz-Ton ersetzt, und danach das Programm am Ende abläuft und das Programm (und das Hintergrundaudio) beendet, was dazu führt dass nichts gehört wird.

Wenn Sie möchten, dass MMBasic wartet, während der PLAY-Befehl ausgeführt wird, sollten Sie geeignete PAUSE-Befehle verwenden. Beispielsweise:

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

Dies gilt für alle Versionen des PLAY-Befehls einschließlich PLAY WAV.

Nützliche Befehle

Es gibt eine Reihe von Befehlen, mit denen die Tonausgabe verwaltet werden kann:

`PLAY PAUSE`	Stoppen (pausieren) Sie vorübergehend die aktuell wiedergegebene Datei oder den Ton..
`PLAY RESUME`	Setzt die Wiedergabe einer Datei oder eines Tons fort, die bzw. der zuvor angehalten wurde..
`PLAY STOP`	Beendet die Wiedergabe der Datei oder des Tons. Die Tonausgabe wird auch automatisch beendet, wenn das Programm endet.
`PLAY VOLUME L, R`	Der Lautstärkebereich beträgt min.= 0 und max.= 100. Die Lautstärke wird auf den maximalen Pegel zurückgesetzt, wenn ein Programm ausgeführt wird. Es wird eine logarithmische Skala verwendet, sodass PLAY VOLUME 50,50 halb so laut klingen soll wie 100,100.

Unterstützung für spez. Hardware

Um die Interaktion eines Programms mit der Außenwelt zu erleichtern, enthält PicoMite Treiber für eine Reihe gängiger Peripheriegeräte:

- Infrarot-Fernbedienungsempfänger und -sender
- Der Temperatursensor DS18B20 und der Temperatur-/Feuchtigkeitssensor DHT22
- LCD-Anzeigemodule
- Numerische Tastenfelder
- Batteriegepufferte Uhr
- Ultraschall-Abstandssensor
- WS2812 RGB-LEDs
- PS2 Tastatur

Decoder für Infrarot-Fernbedienungen

Mit dem IR-Befehl können Sie ganz einfach eine Fernbedienung zu Ihrem Projekt hinzufügen. Wenn diese Funktion aktiviert ist, läuft sie im Hintergrund und unterbricht das laufende Programm, wenn eine Taste auf der IR-Fernbedienung gedrückt wird.

Es funktioniert mit allen NEC- oder Sony-kompatiblen Fernbedienungen, einschließlich solcher, die erweiterte Meldungen generieren. Die meisten billigen programmierbaren Fernbedienungen generieren beide Protokolle, und mit einem dieser Protokolle können Sie Ihrem PicoMite-basierten Projekt ein raffiniertes Flair verleihen. Das NEC-Protokoll wird auch von vielen anderen Herstellern verwendet, darunter Apple, Pioneer, Sanyo, Akai und Toshiba, sodass deren Markenfernbedienungen verwendet werden können.

Um das IR-Signal zu erkennen, benötigen Sie einen IR-Empfänger. NEC-Fernbedienungen verwenden eine 38-kHz-Modulation des IR-Signals, und geeignete Empfänger, die auf diese Frequenz abgestimmt sind, umfassen Vishay TSOP4838, Jaycar ZD1952 und Altronics Z1611A. Beachten Sie, dass die E/A-Pins am PicoMite nur 3,3 V tolerant sind und der Empfänger daher mit maximal 3,3 V versorgt werden muss.

Sony-Fernbedienungen verwenden eine 40-kHz-Modulation, aber Empfänger für diese Frequenz können schwer zu finden sein. Im Allgemeinen funktionieren 38-kHz-Empfänger, aber die maximale Empfindlichkeit wird mit einem 40-kHz-Empfänger erreicht.

Der IR-Empfänger kann an jeden Pin des PicoMite angeschlossen werden. Dieser Pin muss vom Programm mit folgendem Befehl konfiguriert warden:

```
SETPIN n, IR
```

wobei n der E/A-Pin ist, der für diese Funktion verwendet werden soll.

Um den Decoder einzurichten, verwenden Sie den Befehl:

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

Wobei dev eine Variable ist, die mit dem Gerätecode aktualisiert wird, und key die Variable ist, die mit dem Schlüsselcode aktualisiert werden soll. Interrupt ist das Interrupt-Unterprogramm, das aufgerufen wird, wenn ein neuer Tastendruck erkannt wurde. Die IR-Decodierung erfolgt im Hintergrund und das Programm läuft nach diesem Befehl ohne Unterbrechung weiter.

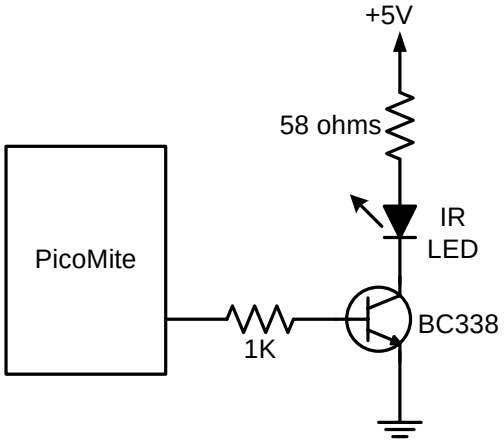
Dies ist ein Beispiel für die Verwendung des an den GP6-Pin angeschlossenen IR-Decoders:

```
SETPIN GP6, IR                  ' Pin festlegen
DIM INTEGER DevCode, KeyCode    ' Decoder-Variablen
IR DevCode, KeyCode, IRInt      ' Start IR-Decoder
DO
  ' < body of the program >
LOOP

SUB IRInt                        ' Welcher Tastendruck erkannt ?
  PRINT "Received  device = " DevCode "  key = " KeyCode
END SUB
```

IR-Fernbedienungen können viele verschiedene Geräte (Videorecorder, Fernseher usw.) ansprechen, sodass das Programm normalerweise zuerst den Gerätecode untersucht, um festzustellen, ob das Signal für das Programm bestimmt war, und, wenn dies der Fall ist, dann basierend auf der gedrückten Taste Maßnahmen ergreift. Es

gibt viele verschiedene Geräte und Schlüsselcodes, daher ist die beste Methode, um festzustellen, welche Codes Ihre Fernbedienung generiert, die Verwendung des obigen Programms, um die Codes zu ermitteln.

Sender für IR-Fernbedienungssender



Mit dem IRSEND-Befehl können Sie ein 12-Bit-Infrarot-Fernbedienungssignal von Sony übertragen. Dies ist für die Kommunikation von PicoMite zu PicoMite oder Micromite gedacht, funktioniert aber auch mit Sony-Geräten, die 12-Bit-Codes verwenden. Beachten Sie, dass alle Sony-Produkte erfordern, dass die Nachricht dreimal mit einer Verzögerung von 26 ms zwischen jeder Nachricht gesendet wird.

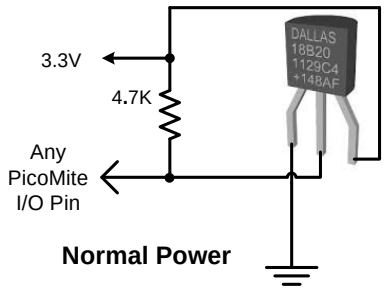
Die Schaltung auf der rechten Seite zeigt, was erforderlich ist. Der Transistor wird zur Ansteuerung der Infrarot-LED verwendet, da die Ausgangsleistung des PicoMite begrenzt ist. Diese Schaltung versorgt die LED mit etwa 50 mA.

Um ein Signal zu senden, verwenden Sie den Befehl:

```
IRSEND pin, dev, key
```

Dabei ist pin der verwendete E/A-Pin, dev der zu sendende Gerätecode und key der Tastencode. Jeder I/O-Pin am PicoMite kann verwendet werden und Sie müssen ihn nicht vorher einrichten (IRSEND erledigt dies automatisch). Die verwendete Modulationsfrequenz beträgt 38 kHz und entspricht den üblichen IR-Empfängern (auf der vorherigen Seite beschrieben) für maximale Empfindlichkeit bei der Kommunikation zwischen zwei PicoMites oder mit einem Micromite.

Temperaturmessung

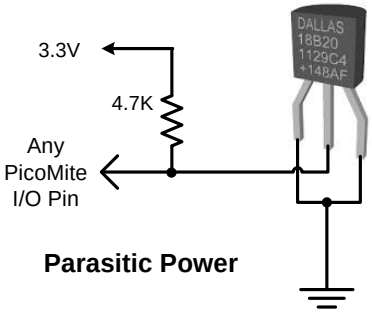


Die TEMPR()-Funktion erhält die Temperatur von einem DS18B20-Temperatursensor. Dieses Gerät kann bei eBay für etwa 5 US-Dollar in einer Vielzahl von Paketen einschließlich einer wasserdichten Sondenversion erworben werden.

Der DS18B20 kann separat mit einer 3,3-V-Versorgung versorgt werden oder wie rechts gezeigt mit parasitärer Stromversorgung vom PicoMite betrieben werden. Es können mehrere Sensoren verwendet werden, aber für jeden ist ein separater I/O-Pin und ein 4,7-K-Pullup-Widerstand erforderlich.

Um die aktuelle Temperatur zu erhalten, verwenden Sie einfach die Funktion TEMPR() in einem Ausdruck. Beispielsweise:

```
PRINT "Temperature: " TEMPR(pin)
```



Wobei "Pin" der I/O-Pin ist, an den der Sensor angeschlossen ist. Sie müssen den I/O-Pin nicht konfigurieren, der von MMBasic verwaltet wird. Der zurückgegebene Wert ist in Grad C mit einer Auflösung von 0,25 ºC und einer Genauigkeit von ±0,5 ºC. Wenn während der Messung ein Fehler auftritt, ist der zurückgegebene Wert 1000.

Die für die Gesamtmessung benötigte Zeit beträgt 200 ms und das laufende Programm wird für diesen Zeitraum angehalten, während die Messung durchgeführt wird. Dies bedeutet auch, dass Interrupts für diesen Zeitraum gesperrt werden. Wenn Sie dies nicht möchten, können Sie die Konvertierung separat mit dem Befehl TEMPR START auslösen und später die Funktion TEMPR() verwenden, um den Temperaturmesswert abzurufen. Die TEMPR()-Funktion wartet immer, wenn der Sensor noch die Messung durchführt.

Beispielsweise:

```
TEMPR START GP15
< do other tasks >
PRINT "Temperature: " TEMPR(GP15)
```

Feuchtigkeits- und Temperaturmessung

Der Befehl BITBANG HUMID liest die Feuchtigkeit und Temperatur von einem DHT22 Feuchtigkeits-/Temperatursensor. Dieses Gerät wird auch als RHT03 oder AM2302 verkauft, aber alle sind kompatibel und können bei eBay für weniger als 5 $ erworben werden. Der DHT11-Sensor wird ebenfalls unterstützt.

Das DHT22 muss mit 3,3 V versorgt werden (die maximale Spannung für die I/O-Pins des PicoMite) und es sollte einen Pullup-Widerstand auf der Datenleitung haben, wie gezeigt. Dies ist für lange Kabelwege (bis zu 20 Meter) geeignet, aber für kurze Wege kann der Widerstand weggelassen werden, da der PicoMite auch einen internen schwachen Pullup bietet.

Um die Temperatur oder Luftfeuchtigkeit zu erhalten, verwenden Sie den HUMID-Befehl mit drei Argumenten wie folgt:

```
BITBANG HUMID pin, tVar, hVar [,DHT11]
```

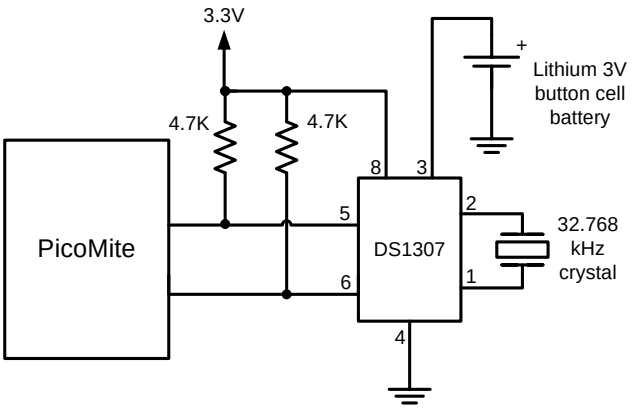
Wobei "Pin" der I/O-Pin ist, an den der Sensor angeschlossen ist. Der I/O-Pin wird automatisch von MMBasic konfiguriert. 'tVar' ist eine Fließkommavariable, in der die Temperatur zurückgegeben wird und 'hVar' ist eine zweite Variable für die Luftfeuchtigkeit. Die Temperatur wird in Grad C mit einer Auflösung von einer Dezimalstelle (z. B. 23,4) und die Luftfeuchtigkeit in Prozent relativer Luftfeuchtigkeit (z. B. 54,3) zurückgegeben. Wenn der optionale DHT11-Parameter auf 1 gesetzt ist, verwendet der Befehl Geräte-Timings, die für dieses Gerät geeignet sind. In diesem Fall werden die Ergebnisse mit einer Auflösung von 1 Grad und 1 % Luftfeuchtigkeit zurückgegeben

Beispielsweise:

```
DIM FLOAT temp, humidity
BITBANG HUMID GP15, temp, humidity
PRINT "The temperature is" temp " and the humidity is" humidity
```

Echtzeituhr

Mit dem RTC GETTIME-Befehl ist es einfach, die aktuelle Uhrzeit von einer PCF8563-, DS1307-, DS3231- oder DS3232-Echtzeituhr sowie kompatiblen Geräten wie dem M41T11 zu erhalten. Diese integrierten Schaltungen sind beliebt und billig, halten die genaue Zeit auch ohne Strom und können für 2 bis 8 US-Dollar bei eBay erworben werden. Komplette Module inklusive Akku gibt es auch bei eBay für etwas mehr zu kaufen. Der PCF8563 und der DS1307 halten die Zeit auf ein bis zwei Minuten über einen Monat, während der DS3231 und der DS3232 besonders präzise sind und über ein Jahr auf eine Minute genau bleiben.



Diese Chips sind I2C-Geräte und sollten mit den I2C-I/O-Pins des PicoMite verbunden werden. Diese Schaltung zeigt eine typische Schaltung für den DS1307. Die anderen Chips haben ähnliche Schaltungen (überprüfen Sie ihre Datenblätter). Interne Pullup-Widerstände (100 kΩ) werden an die I2C-E/A-Pins angelegt, sodass in vielen Fällen externe Widerstände (wie im Diagramm gezeigt) nicht benötigt werden.

Um die RTC zu aktivieren, müssen Sie zuerst die zu verwendenden I2C-Pins mit dem folgenden Befehl zuweisen:

```
OPTION SYSTEM I2C  SDApin, SCLpin
```

Die von der RTC verwendete Zeit muss ebenfalls eingestellt werden. Dies geschieht mit dem Befehl RTC SETTIME, der das Format RTC SETTIME Jahr, Monat, Tag, Stunde, Minute, Sekunde verwendet. Beachten Sie, dass die Stunde im 24-Stunden-Format angegeben werden muss.

Folgendes Beispiel stellt die Echtzeituhr am 10. November 2021 auf 16:00 Uhr ein:

```
RTC SETTIME 2021, 11, 10, 16, 0, 0
```

Um die Zeit zu erhalten, verwenden Sie den RTC GETTIME-Befehl, der die Zeit vom Echtzeituhr-Chip liest und die Uhr im PicoMite stellt. Normalerweise wird dieser Befehl an den Anfang des Programms oder in das Unterprogramm MM.STARTUP gestellt, damit die Zeit beim Einschalten eingestellt wird. Mit dem Befehl OPTION RTC AUTO ENABLE kann auch ein automatisches Update eingestellt werden.

Entfernungsmessung

Mit einem Ultraschallsensor HC-SR04 und der Funktion DISTANCE() können Sie die Entfernung zu einem Ziel messen.

Dieses Gerät ist bei eBay für etwa 4 US-Dollar erhältlich und misst die Entfernung zu einem Ziel von 3 cm bis 3 m. Es sendet einen Ultraschallimpuls und misst die Zeit, die es dauert, bis das Echo zurückkommt.

Kompatible Sensoren sind der SRF05, SRF06, Parallax PING und der DYP-ME007 (der wasserdicht ist und sich daher gut zur Überwachung des Füllstands eines Wassertanks eignet).

Auf dem PicoMite verwenden Sie die DISTANCE-Funktion wie folgt:

```
d = DISTANCE(trig, echo)
```

Der zurückgegebene Wert ist die Entfernung in Zentimetern zum Ziel.

Wobei trig der I/O-Pin ist, der mit dem "trig"-Eingang des Sensors verbunden ist, und echo der Pin ist, der mit dem "echo"-Ausgang des Sensors verbunden ist. Sie können auch 3-Pin-Geräte verwenden und in diesem Fall wird nur eine Pin-Nummer angegeben. Die maximale Spannung an den I/O-Pins des PicoMite beträgt 3,3 V, daher ist ein Widerstandsteiler erforderlich, um den PicoMite mit dem Echo-Pin des Sensors (der mit 5 V arbeitet) zu verbinden.

LCD Display

Der LCD-Befehl zeigt Text auf einem Standard-LCD-Modul mit minimalem Programmieraufwand an.

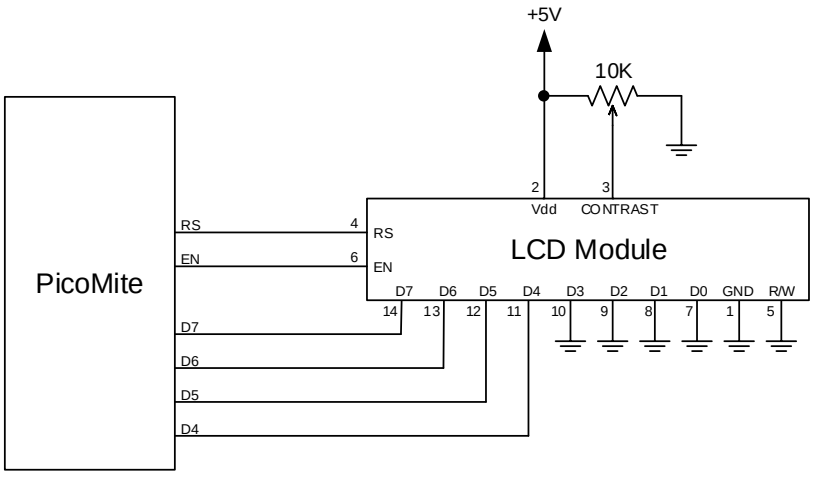
Dieser Befehl funktioniert mit LCD-Modulen, die den KS0066-, HD44780- oder SPLC780-Controller-Chip verwenden und 1, 2 oder 4 Zeilen haben. Typische Displays sind das LCD16X2 (futurlec.com), das Z7001 (altronics.com.au) und das QP5512 (jaycar.com.au). eBay ist eine weitere gute Quelle, bei der die Preise zwischen 10 und 50 US-Dollar liegen können.

Um das Display einzurichten, verwenden Sie den Befehl BITBANG LCD INIT:

```
BITBANG LCD INIT d4, d5, d6, d7, rs, en
```

d4, d5, d6 und d7 sind die Nummern der E/A-Pins, die mit den Eingängen D4, D5, D6 und D7 am LCD-Modul verbunden sind (die Eingänge D0 bis D3 und R/W am Modul sollten mit Masse verbunden sein). 'rs' ist der Pin, der mit dem Registerauswahleingang des Moduls verbunden ist (manchmal auch als CMD oder DAT bezeichnet). 'en' ist der Pin, der mit dem Enable- oder Chip-Select-Eingang des Moduls verbunden ist.

Alle I/O-Pins auf dem PicoMite können verwendet werden und Sie müssen sie nicht vorher einrichten (der LCD-Befehl erledigt das automatisch für Sie). Das Folgende zeigt einen typischen Aufbau.



Um Zeichen auf dem Modul anzuzeigen, verwenden Sie den LCD-Befehl:

```
BITBANG LCD line, pos, data$
```

Dabei ist line die Zeile auf dem Display (1 bis 4) und pos die Position auf der Zeile, an der die Daten geschrieben werden sollen (die erste Position auf der Zeile ist 1). data$ ist eine Zeichenfolge, die die Daten enthält, die auf das LCD-Display geschrieben werden sollen. Die Zeichen in data$ überschreiben alles, was sich auf diesem Teil des LCDs befand. Das Folgende zeigt eine typische Verwendung, bei der d4 bis d7 mit den Pins GP2 bis GP4 auf dem PicoMite verbunden sind, rs mit Pin GP23 und en mit Pin GP24 verbunden ist.

```
BITBANG LCD INIT GP2, GP3, GP4, GP5, GP23, GP24
BITBANG LCD 1, 2, "Temperature"
BITBANG LCD 2, 6, STR$(TEMPR(GP15))   ' DS18B20 connected to pin GP15
```

Beachten Sie, dass dieses Beispiel auch die Funktion TEMPR() verwendet, um die Temperatur abzurufen (oben beschrieben).

Anschluß eines Tastenfeldes

Eine Tastenfeld ist eine Low-Tech-Methode zur Eingabe von Daten in ein PicoMite-basiertes System. Der PicoMite unterstützt entweder eine 4x3-Tastatur oder eine 4x4-Tastatur, und die Überwachung und Dekodierung von Tastendrücken erfolgt im Hintergrund. Wenn ein Tastendruck erkannt wird, wird ein Interrupt ausgegeben, wo das Programm damit umgehen kann.

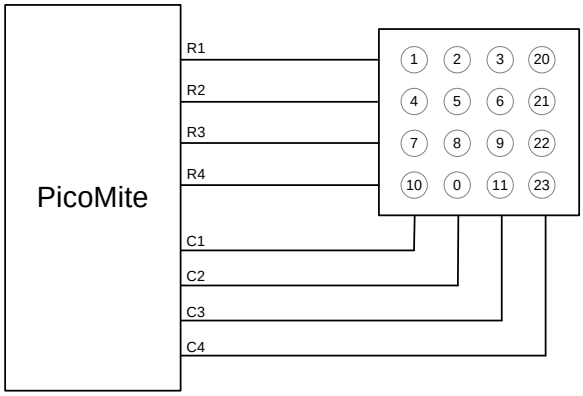
Beispiele für eine 4x3-Tastatur und eine 4x4-Tastatur sind Altronics S5381 und S5383 (siehe www.altronics.com).

Um die Tastaturfunktion zu aktivieren, verwenden Sie den Befehl:

```
KEYPAD  var, int, r1, r2, r3, r4, c1, c2, c3, c4
```

Wobei var eine Variable ist, die mit dem Tastencode aktualisiert wird, und int der Name der Interrupt-Subroutine ist, die aufgerufen wird, wenn ein neuer Tastendruck erkannt wurde. r1, r2, r3 und r4 sind die Pin-Nummern, die für die vier Reihenverbindungen zum Tastenfeld verwendet werden (siehe Diagramm unten), und c1, c2, c3 und c4 sind die Spaltenverbindungen. c4 wird nur mit 4x4-Tastaturen verwendet und sollte weggelassen werden, wenn Sie eine 4x3-Tastatur verwenden.

Alle I/O-Pins auf dem PicoMite können verwendet werden und Sie müssen sie nicht vorher einrichten, der KEYPAD-Befehl erledigt das automatisch für Sie.



Die Erkennung und Dekodierung von Tastendrücken erfolgt im Hintergrund und das Programm wird nach diesem Befehl ohne Unterbrechung fortgesetzt. Wenn ein Tastendruck erkannt wird, wird der Wert der

Variablen var auf die Zahl gesetzt, die die Taste darstellt (dies ist die Zahl innerhalb der Kreise im obigen Diagramm). Dann wird der Interrupt aufgerufen.

Beispielsweise:

```
Keypad KeyCode, KP_Int,GP2,GP3,GP4,GP5,GP21,GP22,GP23   ' 4x3 keybd
DO
  < body of the program >
LOOP

SUB KP_Int                              ' a key press has been detected
  PRINT "Key press = " KeyCode
END SUB
```

Unterstützung für LED WS 2812

Der PicoMite hat eine eingebaute Unterstützung für den mehrfarbigen LED-Chip WS2812. Dieser Chip benötigt ein sehr spezifisches Timing, um richtig zu funktionieren, und mit dem BITBANG WS8212-Befehl ist es einfach, diese Geräte mit minimalem Aufwand zu steuern.

Dieser Befehl gibt die erforderlichen Signale aus, die zum Ansteuern einer Kette von WS2812-LED-Chips erforderlich sind, die mit dem angegebenen Pin verbunden sind, und stellt die Farben jeder LED in der Kette ein. Die Syntax des Befehls lautet:

```
BITBANG WS2812 type, pin, colours%()
```

Beachten Sie, dass der Pin auf einen digitalen Ausgang gesetzt werden muss, bevor dieser Befehl verwendet wird. Das Array colors%() sollte so bemessen sein, dass es genau die gleiche Anzahl von Elementen wie die Anzahl der anzusteuernden LEDs hat. Jedes Element im Array sollte die Farbe im normalen RGB888-Format enthalten (0 - HFFFFFF).

Es werden bis zu 256 WS2812-Chips in einer Kette unterstützt.

„Typ“ ist ein einzelnes Zeichen, das den Typ des angesteuerten Chips wie folgt angibt:

O = original WS2812

B = WS2812B

S = SK6812

Beispiel:

```
DIM b%(4)=(RGB(red), Rgb(green), RGB(blue), RGB(Yellow), rgb(cyan))
SETPIN GP5, DOUT
BITBANG WS2812 O, GP5, b%()
```

gibt die angegebenen Farben an ein Array von fünf WS2812-LEDs aus, die an Pin GP5 verkettet sind.

Es ist möglich, dass ein WS2812 mit dem 3,3-V-Ausgang des PicoMite nicht zuverlässig funktioniert. In diesem Fall gibt es mehrere Lösungen:

Verwenden Sie den WS2812B, der mit einer 3,3-V-Versorgung und Eingängen funktioniert.

- Verwenden Sie einen Level-Shifter, um den WS2812 anzusteuern.
- Verwenden Sie einen einzelnen WS2812 mit 3,3 V als erste Stufe, um den Eingang der ersten "echten" LED in der Kette zu puffern. Die Mindestversorgung für den WS2812 beträgt 4 V, aber in vielen Fällen funktioniert er bei 3,3 V..

PS2-Tastatur

Am PicoMite können Sie eine PS2-Tastatur anschließen und die Ausgabe des Interpreters auf dem LCD anzeigen (siehe „LCD-Display als Konsolenausgang“ unten). Dies macht den PicoMite zu einem völlig eigenständigen Computer mit Tastatur und Display. Mit dem eingebauten farbcodierten Editor können Programme eingegeben, bearbeitet und ausgeführt werden ohne dass ein weiterer Computer erforderlich ist.

Da der PicoMite I/O max. 3,6V verträgt und die PS2-Tastaturen 5 V benötigen sollten für CLOCK- und DATA-Pins Pegelwandler eingesetzt werden. Geeignet ist z.B. der Adafruit 4-Kanal bidirektional-Pegelwandler. Der PS2 CLOCK-Pin sollte über den Konverter mit dem PicoMite Pin 11 (GP8) und den PS2 DATA Pin an PicoMite Pin 12

(GP9). Bevor die Tastatur verwendet werden kann, muss sie zunächst durch Festlegen der Tastaturlayouts aktiviert werden:

OPTION TASTATUR language [,capslock][,numlock][,repeatstart] [,repeat] Wobei „languge“ ein zweistelliger Code wie z.B. „US“ . Andere Tastaturlayouts: Vereinigtes Königreich (UK), Französisch (FR), Deutsch (GR), Belgien (BE), Italienisch (IT) oder Spanisch (ES). Beachten Sie, dass die Nicht-US-Layouts einige der Sondertasten abbilden können aber die entsprechenden Sonderzeichen nicht angezeigt werden da sie nicht Teil der Standard-PicoMite-Schriftarten sind. Stattdessen wird ein anderes Zeichen verwendet. Dieser Befehl konfiguriert die der Tastatur zugewiesenen I/O-Pins und initialisiert sie für die Verwendung. Wie bei ähnlichen Befehlen wird diese Option im Flash-Speicher gespeichert und beim Einschalten automatisch angewendet. Wird die Tastatur entfernt wird kann mit dem Befehl OPTION KEYBOARD DISABLE der Tastaturstatus zurückgesetzt werden.

Einzelheiten zu den optionalen Parametern finden Sie unter OPTION KEYBOARD.

LCD-Display als Konsolenausgang

Die Tastatur kann allein als alternative Eingabemethode verwendet werden, funktioniert aber besonders gut, wenn die als Konsolenausgang ein LCD-Anzeigefeld verwendet wird. Das LCD soll eine der SSD1963-Versionen im Querformat sein und muss zuerst mit OPTION LCDPANEL konfiguriert werden. Um die Ausgabe an das LCD-Panel zu aktivieren, sollten Sie den folgenden Befehl verwenden:

OPTION LCDPANEL-KONSOLE [font [, fc [, bc [, blight]]]

„font“ ist die Standardschriftart, „fc“ ist die Standardvordergrundfarbe, „bc“ ist die Standardhintergrundfarbe und „blight“ ist die Standardhelligkeit der Hintergrundbeleuchtung (2 bis 100). Diese Einstellungen werden im Flash gespeichert und zur Konfiguration von MMBasic verwendet beim Einschalten. Sie sind alle optional und standardmäßig auf Schriftart 2, Weiß, Schwarz und 100 % eingestellt.

Auch die Farbcodierung im Editor (siehe unten) wird mit diesem Befehl eingeschaltet (OPTION COLOURCODE OFF wird es wieder ausschalten). Um die Verwendung des LCD-Panels als Konsole zu deaktivieren lautet der Befehl OPTION LCDPANEL NOCONSOLE.

Die Option LCDPANEL CONSOLE kann mit SPI-Displays verwendet werden, die BLIT unterstützen (ILI9341, ST7789_320, ILI9841, wenn MISO verkabelt ist), aber das Scrollen von Text auf dem Bildschirm ist langsam.

In Verbindung mit einer PS2-Tastatur verwandelt diese Option den PicoMite in einen eigenständigen Computer mit eigener Tastatur und Anzeige. Eher wie eine moderne Version des Maximite (siehe http://geoffg.net/maximite.html).

SD Karten-Unterstützung

Der PicoMite bietet volle Unterstützung für SD-Karten. Dazu gehören das Öffnen von Dateien zum Lesen, Schreiben oder wahlfreien Zugriff sowie das Laden und Speichern von Programmen.

Die Firmware funktioniert mit Karten bis zu 32 GB, die in FAT16 oder FAT32 formatiert sind, und die erstellten Dateien können auch auf PCs mit Windows-, Linux- oder Mac-Betriebssystem gelesen/geschrieben werden.

Anschluß einer SD Karte

SD-Karten verwenden das SPI-Protokoll für die Kommunikation und dies muss speziell konfiguriert werden, bevor sie verwendet werden können. Zuerst muss der „System"-SPI-Port konfiguriert werden. Dies ist ein Port, der für die Systemnutzung verwendet wird (SD-Karte, LCD-Display und der Touch-Controller auf einem LCD-Panel).

Es gibt eine Reihe von Ports und Pins, die verwendet werden können (siehe Abschnitt PicoMite-Hardware), aber dieses Beispiel verwendet SPI auf den Pins GP18, GP19 und GP16 für Clock, MOSI und MISO.

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

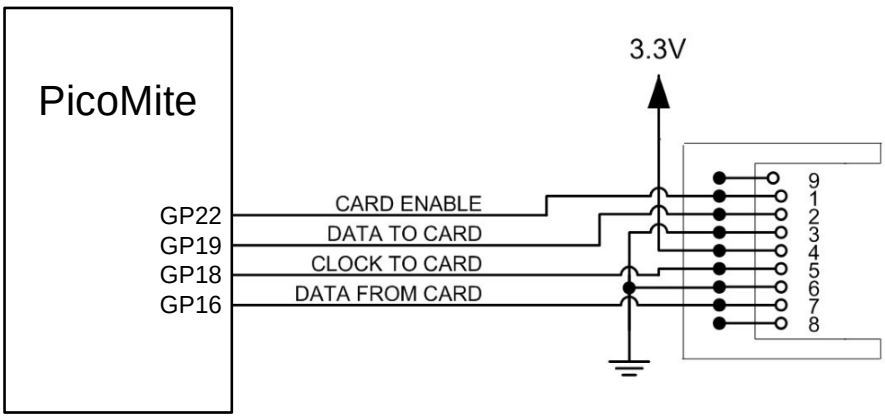
Dann muss MMBasic mitgeteilt werden, dass eine SD-Karte angeschlossen ist und welcher Pin für das Chip-Select-Signal verwendet wird:

```
OPTION SDCARD GP22
```

Diese Befehle müssen an der Eingabeaufforderung (nicht in einem Programm) eingegeben werden und bewirken, dass PicoMite neu gestartet wird. Dies hat den Nebeneffekt, dass die USB-Konsolenschnittstelle getrennt wird, die neu verbunden werden muss.

Wenn PicoMite neu gestartet wird, initialisiert MMBasic automatisch die SD-Kartenschnittstelle. Dieser SPI-Port steht dann BASIC-Programmen nicht zur Verfügung (d. h. er ist reserviert). Um die Konfiguration zu überprüfen, können Sie den Befehl OPTION LIST verwenden, um alle eingestellten Optionen einschließlich der Konfiguration der SD-Karte aufzulisten.

Das Prinzipschaltbild zum Anschluss des SD-Kartensteckers mit dieser Pinbelegung ist unten dargestellt..



Beachten Sie, dass Sie viele verschiedene Konfigurationen mit verschiedenen Pin-Zuweisungen verwenden können – dies ist nur ein Beispiel, das auf den oben aufgeführten Konfigurationsbefehlen basiert.

Vorsicht ist geboten, wenn der SPI-Port von mehreren Geräten (SD-Karte, Touch usw.) gemeinsam genutzt wird. In diesem Fall müssen alle Chip-Select-Signale in MMBasic konfiguriert oder alternativ durch eine dauerhafte Verbindung mit 3,3 V deaktiviert werden. Wenn dies nicht getan wird, werden eventuell schwebende Chip-Select-Signalleitungen dazu führen, dass der falsche Controller auf Befehle auf dem SPI-Bus antwortet.

Unterstützung durch MMBasic

MMBasic auf PicoMite unterstützt die Standard-BASIC-Befehle für die Arbeit mit Speichersystemen.

Bitte beachten Sie:

• Zusätzlich zum alten 8.3-Format werden lange Datei-/Verzeichnisnamen unterstützt.

• Die maximale Datei-/Pfadlänge beträgt 63 Zeichen.

• Groß-/Kleinbuchstaben und Leerzeichen sind erlaubt, obwohl das Dateisystem nicht zwischen Groß- und Kleinschreibung unterscheidet.

• Verzeichnispfade sind in Datei-/Verzeichnis-Strings erlaubt. (d. h. ÖFFNE „\dir1\dir2\file.txt“ FÜR …).

• In Pfaden können entweder Schrägstriche oder umgekehrte Schrägstriche verwendet werden. Z.B. \dir\file.txt ist dasselbe wie /dir/file.txt.

• Die aktuelle PicoMite-Zeit wird für die Dateierstellung und die letzten Zugriffszeiten verwendet.

• Bis zu zehn Dateien können gleichzeitig geöffnet sein.

• Außer bei INPUT, LINE INPUT und PRINT ist das # in #fnbr optional und kann weggelassen werden.

Befehle zum Lesen/ Schreiben von Daten auf der SD-Karte:

□ • OPEN fname$ FOR mode AS #fnbr

Öffnet eine Datei zum Lesen oder Schreiben. „fname$“ ist der Dateiname im 8.3-Format. 'mode' kann INPUT, OUTPUT, APPEND oder RANDOM sein. „#fnbr“ ist die Dateinummer (1 bis 10).

□ • PRINT #fnbr, Ausdruck [[,;]Ausdruck] … usw

Gibt Text in die geöffnete Datei als #fnbr aus.

□ • INPUT #fnbr, Liste der Variablen

Liest eine Liste von kommagetrennten Daten in die angegebenen Variablen aus der Datei, die zuvor als #fnbr geöffnet wurde.

□ • LINE-EINGANG #fnbr, Variable$

Liest eine komplette Zeile in die angegebene String-Variable aus der zuvor als #fnbr geöffneten Datei ein.

□ CLOSE #fnbr [,#fnbr] …
Schließen Sie die zuvor geöffnete(n) Datei(en) mit der Dateinummer „#fnbr“.

Programme können mit zwei Befehlen von der SD-Karte geladen oder auf der SD-Karte gespeichert werden.

□ LOAD fname$ [, R]
Laden Sie ein BASIC-Programm von der SD-Karte. Das optionale Suffix „„R“ bewirkt, dass das Programm ausgeführt wird, nachdem es geladen wurde (in diesem Fall muss fname$ eine String-Konstante sein).

□ RUN fname$
Laden Sie ein BASIC-Programm von der SD-Karte und führen Sie es aus. fname$ muss eine String-Konstante sein.

□ SAVE fname$
Speichern Sie das aktuelle Programm auf der SD-Karte.

Bilder können mit zwei Befehlen von der SD-Karte geladen oder auf der SD-Karte gespeichert werden.

□ LOAD IMAGE fname$
Laden Sie eine BMP-Datei und zeigen Sie sie auf dem LCD-Bildschirm an.

□ SAVE IMAGE fname$
Speichern Sie das aktuelle LCD-Bildschirmbild als BMP-Datei (das LCD-Panel muss transparenten Text unterstützen).

Grundlegende Datei- und Verzeichnismanipulationen können innerhalb eines BASIC-Programms durchgeführt werden.

□ FILES [wildcard]
Durchsuchen Sie das aktuelle Verzeichnis und listen Sie die gefundenen Dateien/Verzeichnisse auf.

- KILL fname$
  Löscht eine Datei im aktuellen Verzeichnis.
- MKDIR dname$
  Erstellen Sie im aktuellen Verzeichnis ein Unterverzeichnis.
- CHDIR dname$
  Wechseln Sie in das Verzeichnis $dname. $dname kann auch ".." (Punkt Punkt) für ein Verzeichnis oder "\" für das Stammverzeichnis sein.
- RMDIR dir$
  Entfernen oder löschen Sie das Verzeichnis „dir$“ auf der SD-Karte
- SEEK #fnbr, pos
  Positioniert den Lese-/Schreibzeiger in einer Datei, die für den RANDOM-Zugriff auf das 'pos'-Byte geöffnet wurde.
- RENAME fromname$ AS toname$
  Benennt die Datei fromname$ in den Namen toname$ um. Außerdem gibt es eine Reihe von Funktionen, die die obigen Befehle unterstützen.
- INPUT$(nbr, #fnbr)
  Gibt eine Zeichenfolge zurück, die aus „nbr“-Zeichen besteht, die aus einer zuvor für INPUT geöffneten Datei mit der Dateinummer „#fnbr“ gelesen wurden. Wenn weniger als „nbr“ Zeichen verfügbar sind, gibt die Funktion das zurück, was sie hat (einschließlich einer leeren Zeichenfolge, wenn keine Zeichen verfügbar sind).
- DIR$(fspec, type)
  Durchsucht eine SD-Karte nach Dateien und gibt die Namen der gefundenen Einträge zurück.
- CWD$
  Gibt das aktuelle Arbeitsverzeichnis zurück.
- EOF(#fnbr)
  Gibt „true“ zurück, wenn die zuvor für INPUT geöffnete Datei mit der Dateinummer „#fnbr“ am Ende der Datei positioniert ist.
- LOC(#fnbr)
  Für eine Datei, die als RANDOM geöffnet wurde, gibt dies die aktuelle Position des Lese-/Schreibzeigers in der Datei zurück.
- LOF(#fnbr)
  Gibt die aktuelle Länge der Datei in Bytes zurück.

Dateiinhalt

Der Inhalt einer Datei auf der SD-Karte wird mit LIST file$ auf der Konsole angezeigt.

Übertragung mit XModem-Protokoll

Neben der Standardmethode der XModem-Übertragung, bei der in den oder aus dem Programmspeicher kopiert wird, kann PicoMite auch in eine und aus einer Datei auf der SD-Karte kopieren. Die Syntax lautet::

```
   XMODEM SEND filename$
```

oder

```
   XMODEM RECEIVE filename$
```

Wobei „filename$“ die Datei ist, die gespeichert oder gesendet werden soll. Wie in MMBasic üblich, kann „filename$“ ein Zeichenfolgenausdruck, eine Variable oder eine Konstante sein. Wenn es sich um eine Konstante handelt, muss der String in Anführungszeichen gesetzt werden (z. B. XMODEM SEND „PRBAS“). Beim Empfang einer Datei wird jede gleichnamige Datei auf der SD-Karte überschrieben.

Laden und Speichern von Bildern

Mit dem Befehl LOAD IMAGE kann ein Bitmap von der SD-Karte geladen und auf einem angeschlossenen LCD angezeigt werden. Dies kann verwendet werden, um ein Logo zu zeichnen oder einen Hintergrund auf dem Display hinzuzufügen. Die Syntax des Befehls lautet:

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

Wobei „filename$“ das zu ladende Bild ist und „StartX“/„StartY“ die Koordinaten der oberen linken Ecke des Bildes sind (diese sind optional und werden standardmäßig in der oberen linken Ecke der Anzeige angezeigt, wenn sie nicht angegeben sind). Das Bild muss im BMP-Format vorliegen und MMBasic fügt dem Dateinamen „.BMP“ hinzu, wenn keine Erweiterung angegeben wird. Alle Typen des BMP-Formats werden unterstützt, einschließlich Schwarzweiß- und Echtfarben-24-Bit-Bildern. Das aktuelle Bild auf einem ILI9341-basierten LCD-Bildschirm kann mit dem folgenden Befehl in einer Datei gespeichert werden:

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

Dadurch wird das Bild oder ein Teil des Bilds als 24-Bit-True-Color-BMP-Datei (mit der Erweiterung .BMP) gespeichert, die hinzugefügt wird, wenn keine Erweiterung angegeben wird.

Beispiel für sequentielle I/O

Im Beispiel unten wird eine Datei erstellt und zwei Zeilen werden in die Datei geschrieben (mithilfe des PRINT-Befehls). Die Datei wird dann geschlossen.

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

Sie können den Inhalt der Datei mit dem Befehl LINE INPUT lesen. Beispielsweise:

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

LINE INPUT liest jeweils eine Zeile, also enthält die Variable a$ den Text "Der schnelle braune Fuchs" und b$ enthält "springt über den faulen Hund".

Eine andere Möglichkeit, aus einer Datei zu lesen, ist die Verwendung der Funktion INPUT$(). Dadurch wird eine bestimmte Anzahl von Zeichen gelesen. Beispielsweise:

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

Das erste INPUT$() liest 12 Zeichen und das zweite drei Zeichen. Die Variable ta$ wird also "The quick br" enthalten und die Variable tb$ wird "own" enthalten.

Dateien enthalten normalerweise nur Text und der Druckbefehl wandelt Zahlen in Text um. Im folgenden Beispiel enthält die erste Zeile also die Zeile „123“ und die zweite „56789“.

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

Auch hier können Sie den Inhalt der Datei mit dem Befehl LINE INPUT lesen, aber dann müssten Sie den Text mit VAL() in eine Zahl umwandeln. Beispielsweise:

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

Danach hätte die Variable x den Wert 123 und y den Wert 56789.

Datei mit wahlfreiem I/O

Für den wahlfreien Zugriff sollte die Datei mit dem Schlüsselwort RANDOM geöffnet werden. Beispielsweise:

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

Um nach einem Datensatz in der Datei zu suchen, verwenden Sie den SEEK-Befehl, der den Lese-/Schreibzeiger auf ein bestimmtes Byte positioniert. Das erste Byte in einer Datei ist mit Eins nummeriert, sodass beispielsweise der fünfte Datensatz in einer Datei, die 64-Byte-Datensätze verwendet, bei Byte 257 beginnen würde. In diesem Fall würden Sie Folgendes verwenden, um darauf zu verweisen:

```
   SEEK #1, 257
```

Beim Lesen aus einer Datei mit wahlfreiem Zugriff sollte die Funktion INPUT$() verwendet werden, da diese eine feste Anzahl von Bytes (d. h. einen vollständigen Datensatz) aus der Datei liest. Um beispielsweise einen Datensatz von 64 Byte zu lesen, würden Sie Folgendes verwenden:

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

Beim Schreiben in die Datei sollte eine feste Datensatzgröße verwendet werden was leicht durch Hinzufügen ausreichender Füllzeichen (normalerweise Leerzeichen) zu den zu schreibenden Daten erreicht werden kann. Beispielsweise:

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

Die Funktion SPACE$() wird verwendet um genügend Leerzeichen hinzuzufügen um sicherzustellen dass die geschriebenen Daten eine exakte Länge haben (in diesem Beispiel 64 Bytes). Das Semikolon am Ende des Druckbefehls unterdrückt das Hinzufügen von Wagenrücklauf- und Zeilenvorschubzeichen, die den Datensatz länger als beabsichtigt machen würden. Zwei weitere Funktionen können beim wahlfreien Dateizugriff helfen. Die Funktion LOC() gibt die aktuelle Byteposition des Lese-/Schreibzeigers zurück und die Funktion LOF() gibt die Gesamtlänge der Datei in Bytes zurück. Das folgende Programm demonstriert den wahlfreien Dateizugriff. Mit ihm können Sie an die Datei anhängen (um zunächst einige Daten hinzuzufügen) und dann Datensätze mit zufälligen Datensatznummern lesen / schreiben. Der erste Datensatz in der Datei ist Datensatznummer 1, der zweite ist 2 usw.

```
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
      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
```

Der wahlfreie Zugriff kann auch auf eine normale Textdatei angewendet werden. Zum Beispiel wird dies eine Datei rückwärts ausdrucken:

```
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
```

Displays

PicoMite bietet Unterstützung für viele LCD-Anzeigefelder mit SPI- oder I2C-Schnittstelle. Die Befehle müssen an der Eingabeaufforderung (nicht in einem Programm) eingegeben werden und bewirken, dass PicoMite neu gestartet wird. Dies hat den Nebeneffekt, dass die USB-Konsolenschnittstelle getrennt wird und neu verbunden werden muss. Beachten Sie dass die maximale Spannung an allen PicoMite I/O-Pins 3,3 V beträgt. Eine Pegelverschiebung ist erforderlich, wenn das Display 5-V-Pegel zur Signalisierung verwendet.

Anzeigefelder mit SPI

Die SPI-basierten Display-Controller teilen sich die SYSTEM-SPI-Kanalschnittstelle auf dem PicoMite mit dem Touch-Controller (falls vorhanden). Eine SD-Karte kann auch so konfiguriert werden dass sie dieselben Pins verwendet. Wenn dies geschehen ist, stehen die dem SYSTEM-SPI zugewiesenen Pins nicht für andere MMBasic-Befehle zur Verfügung. Die Geschwindigkeit des Zeichnens auf SPI-basierten Displays wird weitgehend unbeeinflusst von der CPU-Geschwindigkeit sein.

Die Panels werden mit diesen Befehlen konfiguriert. In allen Befehlen sind die Parameter:

• ODER = Dies ist die Ausrichtung des Displays und kann LANDSCAPE, PORTRAIT, RLANDSCAPE oder RPORTRAIT sein. Diese können mit L, P, RL oder RP abgekürzt werden. Das R-Präfix zeigt die umgekehrte oder "umgedrehte" Ausrichtung an.

• DC = Anzeigedaten/ Befehlssteuerungsstift.

• RESET = Anzeige-Reset-Stift (Aktiv-LOW).

• CS = Anzeige-Chip-Select-Pin.

Es können beliebige freie Pins verwendet werden.

```
OPTION LCDPANEL ILI9341, OR, DC, RESET, CS
```

Initialisiert ein TFT-Display mit dem Controller ILI9341. Dies unterstützt eine Auflösung von 320 * 240. Displays, die diesen Controller verwenden, können transparenten Text anzeigen und funktionieren mit den Befehlen BLIT und BLIT READ.

```
OPTION LCDPANEL ILI9163, OR, DC, RESET, CS
```

Initialisiert ein TFT-Display mit dem Controller ILI9163. Dies unterstützt eine Auflösung von 128 * 128.

```
OPTION LCDPANEL ILI9481, OR, DC, RESET, CS
```

Initialisiert ein TFT-Display mit dem Controller ILI9481. Dies unterstützt eine Auflösung von 480 * 320.

```
OPTION LCDPANEL ILI9488, OR, DC, RESET, CS
```

Initialisiert ein TFT-Display mit dem Controller ILI9488. Dies unterstützt eine Auflösung von 480 * 320. Beachten Sie, dass dieser Controller ein Problem mit dem MISO-Pin hat, der den Touch-Controller stört.

Damit dieses Display funktioniert, darf der MISO-Pin nicht angeschlossen sein.

```
OPTION LCDPANEL ILI9488W, OR, DC, RESET, CS
```

Initialisiert ein TFT-Display mit dem Controller ILI9488. Dies unterstützt das Waveshare 3,5-Zoll-Display wie es auf ihrem Pico Eval-Board verwendet wird und den normalen 3,5-Zoll-Display-Adapter.

```
OPTION LCDPANEL N5110, OR, DC, RESET, CS [,contrast]
```

Initialisiert ein LCD-Display mit dem Nokia 5110 Controller. Dies unterstützt eine Auflösung von 84 * 48. Ein zusätzlicher Parameter LCDVOP kann angegeben werden, um den Kontrast der Anzeige zu steuern. Probieren Sie Kontrastwerte zwischen &HA8 und &HD0 aus, um sie an Ihr Display anzupassen, Standard, wenn weggelassen, ist &HB1

```
OPTION LCDPANEL SSD1306SPI, OR, DC, RESET, CS [,offset]
```

Initialisiert ein OLED-Display unter Verwendung des SSD1306-Controllers mit einer SPI-Schnittstelle. Dies unterstützt eine Auflösung von 128 * 64. Ein zusätzlicher Parameter-Offset kann angegeben werden, um die

Position der Anzeige zu steuern. 0,96-Zoll-Displays benötigen normalerweise einen Wert von 0. 1,3-Zoll-Displays benötigen normalerweise einen Wert von 2. Der Standardwert ist 0, wenn es weggelassen wird.

```
OPTION LCDPANEL SSD1331, OR, DC, RESET, CS
```

Initialisiert ein Farb-OLED-Display mit dem SSD1331-Controller. Dies unterstützt eine Auflösung von 96 * 64.

```
OPTION LCDPANEL ST7735, OR, DC, RESET, CS
```

Initialisiert ein TFT-Display mit dem ST7735-Controller. Dies unterstützt eine Auflösung von 160 * 128.

```
OPTION LCDPANEL ST7735S, OR, DC, RESET, CS
```

Initialisiert ein IPS-Display mit dem ST7735S-Controller. Dies unterstützt eine Auflösung von 160 * 80.

```
OPTION LCDPANEL ST7789, OR, DC, RESET, CS
```

Initialisiert ein IPS-Display mit dem 7789-Controller. Dies unterstützt eine Auflösung von 240 * 240.

HINWEIS: Anzeigetafeln ohne CS-Pin werden derzeit nicht vom PicoMite unterstützt, außer nach Modifikation.

```
OPTION LCDPANEL ST7789_135, OR, DC, RESET, CS
```

Initialisiert ein IPS-Display mit dem 7789-Controller. Dies unterstützt eine Auflösung von 240 * 135.

HINWEIS: Anzeigetafeln ohne CS-Pin werden derzeit nicht vom PicoMite unterstützt, es sei denn, sie werden geändert.

```
OPTION LCDPANEL ST7789_320, OR, DC, RESET, CS
```

Initialisiert ein IPS-Display mit dem 7789-Controller. Dieser Typ unterstützt das Display mit einer Auflösung von 320 * 240 von Waveshare (https://www.waveshare.com/wiki/Pico-ResTouch-LCD-2.8). Diese können transparenten Text verarbeiten und funktionieren mit den Befehlen BLIT und BLIT READ.

HINWEIS: Anzeigetafeln ohne CS-Pin werden derzeit nicht vom PicoMite unterstützt, außer nach vorh. Modifikation.

```
OPTION LCDPANEL ST7920, OR, DC, RESET
```

Initialisiert ein LCD-Display mit dem ST7920-Controller. Dies unterstützt eine Auflösung von 128 * 64. Beachten Sie, dass dieses Display keine Chipauswahl unterstützt, sodass der SPI-Bus nicht gemeinsam genutzt werden kann, wenn dieses Display verwendet wird.

```
OPTION LCDPANEL GDEH029A1, OR, DC, RESET, CS, BUSYpin [,refreshcount]
```

Initialisiert ein e-Ink-Display mit dem GDEH029A1-Controller. Dies unterstützt eine Auflösung von 128 * 296. . 'BUSYpin' ist eine Eingabe, die vom Treiber verwendet wird, um den Befehlsabschluss auf dem langsamen e-Ink-Display herzustellen. Ein zusätzlicher Parameter „Refreshcount“ kann angegeben werden, um zu steuern, wann der Treiber eine vollständige Aktualisierung der Anzeige durchführt (Schwarz/Weiß-Blinken). Der Standardwert ist 1, d. h. die Anzeige führt bei jedem Schreibvorgang eine vollständige Aktualisierung durch. Das Festlegen eines höheren Werts kann Aktualisierungen schneller und weniger offensichtlich machen birgt jedoch das Risiko Geisterbilder auf dem Display zu erzeugen.

LCD-Anzeigefelder mit I2C

Die I2C-basierten Display-Controller verwenden die SYSTEM-I2C-Pins gemäß der Pinbelegung für das spezifische Gerät. Andere I2C-Geräte können den Bus teilen sofern ihre Adressen eindeutig sind.

Wenn ein I2C-Display konfiguriert ist ist es nicht erforderlich den I2C-Port für ein zusätzliches Gerät zu "öffnen" (I2C OPEN), I2C CLOSE wird blockiert, und alle I2C-Geräte müssen für den 400-kHz-Betrieb geeignet sein. Die I2C-Busgeschwindigkeit wird durch Änderungen der CPU-Taktgeschwindigkeit nicht beeinflusst

Diese Panels werden mit den folgenden Befehlen konfiguriert. Bei allen Befehlen ist der Parameter OR die Ausrichtung der Anzeige und kann LANDSCAPE, PORTRAIT, RLANDSCAPE oder RPORTRAIT sein. Diese können mit L, P, RL oder RP abgekürzt werden. Das R-Präfix zeigt die umgekehrte oder "umgedrehte" Ausrichtung an..

```
OPTION LCDPANEL SSD1306I2C, OR [,offset]
```

Initialisiert ein OLED-Display unter Verwendung des SSD1306-Controllers mit einer I2C-Schnittstelle. Dies unterstützt eine Auflösung von 128 * 64. Ein zusätzlicher Parameter-Offset kann angegeben werden, um die Position der Anzeige zu steuern. 0,96-Zoll-Displays benötigen normalerweise einen Wert von 0. 1,3-Zoll-Displays benötigen normalerweise einen Wert von 2. Der Standardwert ist 0, wenn es weggelassen wird.

Hinweis: Viele billige I2C-Versionen von SSD1306-Displays implementieren I2C aufgrund eines Verdrahtungsfehlers nicht richtig. Dies scheint insbesondere bei 1,3"-Varianten der Fall zu sein

```
OPTION LCDPANEL SSD1306I2C32, OR
```

Initialisiert ein OLED-Display unter Verwendung des SSD1306-Controllers mit einer I2C-Schnittstelle. Dies unterstützt eine Auflösung von 128 * 32.

Hintergrundbeleuchtung

Für die Displays ILI9163, ILI9341, ST7735, ST7735S, SSD1331, ST7789, ILI9481, ILI9488, ILI9488W, ST7789_135 und ST7789_320 kann am Ende der Konfigurationsparameter ein optionaler Parameter „Backlight“ hinzugefügt werden, der einen Pin angibt, der die Helligkeit der Hintergrundbeleuchtung einstellt. Dadurch wird an diesem Pin ein PWM-Ausgang mit einer Frequenz von 1 kHz und einem anfänglichen Tastverhältnis von 99 % aktiviert. Sie können den Befehl BACKLIGHT verwenden um die Helligkeit zwischen 0 und 100 % zu ändern. Der PWM-Kanal ist für die normale PWM-Nutzung gesperrt und darf nicht mit dem eventuell für Audio eingerichteten PWM-Kanal in Konflikt geraten.

Beispielsweise:

```
OPTION LCDPANEL ILI9341, OR, DC, RESET, CS, GP11
```

Die Hintergrundbeleuchtung kann dann mit diesem Befehl auf 40% eingestellt werden:

```
BACKLIGHT 40
```

Anzeigefeld- Beispiel

Dieses Beispiel basiert auf einem LCD-Panel mit dem ILI9341-Controller. Dieses Panel verwendet das SPI-Protokoll für die Kommunikation. Dies muss speziell konfiguriert werden bevor das Panel konfiguriert werden kann. Dies ist der „System“-SPI-Port, der für die Systemnutzung verwendet wird (SD-Karte, LCD-Display und der Touch-Controller auf einem LCD-Panel). Dieser SPI-Port steht dann BASIC-Programmen nicht zur Verfügung (d.h. er ist reserviert). Es gibt eine Reihe von Ports und Pins, die verwendet werden können (siehe Abschnitt PicoMite-Hardware), aber dieses Beispiel verwendet SPI auf den Pins 21, 24 und 25 für Uhr, MOSI und MISO.

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

Dies gilt auch für die zuvor in diesem Handbuch für die beispielhafte SD-Kartenschnittstelle verwendete Konfiguration und muss nicht wiederholt werden, wenn sie bereits für die SD-Karte konfiguriert wurde.

Dann muss MMBasic für das Display konfiguriert werden und die Ausrichtung des Displays mitgeteilt werden und welche Pins für die Signale Data/Command (GP15), Reset (GP14) und Chip Select (GP13) verwendet werden:

```
OPTION LCDPANEL ILI9341, L, GP15, GP14, GP13
```

Diese Befehle müssen an der Eingabeaufforderung (nicht in einem Programm) eingegeben werden und bewirken, dass PicoMite neu gestartet wird. Dies hat den Nebeneffekt, dass die USB-Konsolenschnittstelle getrennt wird, die neu verbunden werden muss.

Wenn PicoMite neu gestartet wird initialisiert MMBasic automatisch das LCD-Display (der Bildschirm wird dunkel). Um die Konfiguration zu überprüfen, können Sie den Befehl OPTION LIST verwenden um alle eingestellten Optionen einschließlich der Konfiguration der SD-Karte aufzulisten. Sie können auch den Befehl GUI TEST LCDPANEL verwenden, der mehrere bunte, sich überlappende Kreise auf dem LCD-Bildschirm zeichnet.

Beachten Sie, dass Sie viele verschiedene Konfigurationen mit verschiedenen Pin-Zuweisungen verwenden können – dies ist nur ein Beispiel, das auf den oben aufgeführten Konfigurationsbefehlen basiert.

Vorsicht ist geboten, wenn der SPI-Port von mehreren Geräten (SD-Karte, Touch usw.) gemeinsam genutzt wird. In diesem Fall müssen alle Chip-Select-Signale in MMBasic konfiguriert oder alternativ durch eine dauerhafte Verbindung mit 3,3 V deaktiviert werden. Wenn dies nicht getan wird, werden eventuell schwebende Chip-Select-Signalleitungen dazu führen, dass der falsche Controller auf Befehle auf dem SPI-Bus antwortet..

Touchpanels

Viele LCD-Panels werden mit einem resistiven berührungsempfindlichen Panel und einem zugehörigen Controller-Chip geliefert. MMBasic unterstützt diese Schnittstelle vollständig, wodurch viele der physischen Knöpfe und Schalter, die in einem Projekt verwendet werden, als Bildschirmsteuerungen implementiert werden können, die durch Berührung aktiviert werden.

Beachten Sie, dass die maximale Spannung an allen PicoMite I/O-Pins 3,3 V beträgt. Eine Pegelverschiebung ist erforderlich, wenn ein Display 5-V-Pegel zur Signalisierung verwendet..

Konfiguration

Der Touch-Controller auf einem LCD-Panel verwendet das SPI-Protokoll für die Kommunikation und dies muss speziell konfiguriert werden, bevor das Panel konfiguriert werden kann. Dies ist der „System"-SPI-Port, der für die Systemnutzung verwendet wird (SD-Karte, LCD-Display und der Touch-Controller auf einem LCD-Panel). Dieser SPI-Port steht dann BASIC-Programmen nicht zur Verfügung (d.h. er ist reserviert)

Es gibt eine Reihe von Ports und Pins, die verwendet werden können, aber diese sind die gleichen wie die Konfiguration, die für die beispielhafte LCD-Panel-Schnittstelle weiter oben in diesem Handbuch verwendet wurde. Dieser Befehl muss nicht wiederholt werden, wenn das System-SPI bereits konfiguriert wurde:

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

Um die Touch-Funktion zu verwenden, muss MMBasic mitgeteilt werden, dass sie mit dem OPTION TOUCH-Befehl verfügbar ist. Dies sollte nach der Konfiguration des LCD-Displays erfolgen. Dieser Befehl teilt MMBasic mit, welche Pins für die Chip-Select- und Interrupt-Signale verwendet werden. Dies setzt beispielsweise Chip Select auf den GP12-Pin und Interrupt auf GP11:

```
OPTION TOUCH GP12, GP11
```

Diese Befehle müssen an der Eingabeaufforderung eingegeben werden und führen zum Neustart von PicoMite. Dies hat den Nebeneffekt, dass die USB-Konsolenschnittstelle getrennt wird, die neu verbunden werden muss.

Wenn PicoMite neu gestartet wird, initialisiert MMBasic automatisch den Touch-Controller. Um die Konfiguration zu überprüfen, können Sie den Befehl OPTION LIST verwenden, um alle eingestellten Optionen aufzulisten, einschließlich der Konfiguration des Anzeigefelds und der Berührung.

Beachten Sie, dass Sie viele verschiedene Konfigurationen mit verschiedenen Pin-Zuweisungen verwenden können – dies ist nur ein Beispiel, das auf den oben aufgeführten Konfigurationsbefehlen basiert.

Vorsicht ist geboten, wenn der SPI-Port von mehreren Geräten (SD-Karte, Touch usw.) gemeinsam genutzt wird. In diesem Fall müssen alle Chip-Select-Signale in MMBasic konfiguriert oder alternativ deaktiviert werden.

Touchscreen kalibrieren

Bevor die Berührungsfunktion verwendet werden kann, muss sie mit dem GUI-Befehl CALIBRATE kalibriert werden.

Dieser Befehl zeigt ein Ziel in der oberen linken Ecke des Bildschirms an. Drücken Sie mit einem Gegenstand mit abgerundeter Spitze z. B. einem Zahnstocher genau auf die Mitte des Ziels und halten Sie es mindestens eine Sekunde lang gedrückt. MMBasic zeichnet diese Position auf und fährt dann mit der Kalibrierung fort, indem das Ziel nacheinander in den anderen drei Ecken des Bildschirms zum Berühren und Kalibrieren angezeigt wird. Die Kalibrierungsroutine kann warnen, dass die Kalibrierung nicht genau war. Dies ist nur eine Warnung und Sie können die Touch-Funktion immer noch verwenden, wenn Sie möchten, aber es wäre besser, die Kalibrierung mit mehr Sorgfalt zu wiederholen. Nach der Kalibrierung können Sie die Touch-Funktion mit dem GUI-Befehl TEST TOUCH testen. Dieser Befehl blendet den Bildschirm aus und wartet auf eine Berührung. Wenn der Bildschirm berührt wird, wird ein weißer Punkt auf dem Display platziert, der die Position auf dem Bildschirm markiert. Wenn die Kalibrierung erfolgreich durchgeführt wurde, sollte der Punkt genau unter der Position des Stifts auf dem Bildschirm angezeigt werden.

Um die Testroutine zu verlassen, können Sie die Leertaste auf der Tastatur der Konsole drücken.

Touchfunktionen

Um zu erkennen, ob und wo der Bildschirm berührt wird, können Sie die folgenden Funktionen in einem BASIC-Programm verwenden:

- TOUCH(X)
  Gibt die X-Koordinate der aktuell berührten Stelle oder -1 zurück, wenn der Bildschirm nicht berührt wird.

- TOUCH(Y)
  Gibt die Y-Koordinate der aktuell berührten Stelle oder -1 zurück, wenn der Bildschirm nicht berührt wird..
- TOUCH(DOWN)
  Gibt true zurück, wenn der Bildschirm gerade berührt wird (dies ist viel schneller als TOUCH(X oder Y)).
- TOUCH(UP)
  Gibt true zurück, wenn der Bildschirm gerade NICHT berührt wird (auch schneller als TOUCH(X oder Y))
- TOUCH(LASTX)
  Gibt die X-Koordinate der zuletzt berührten Position zurück.
- TOUCH(LASTY)
  Gibt die Y-Koordinate der zuletzt berührten Position zurück.
- TOUCH(REF)
  Gibt die Referenznummer des Steuerelements zurück, das gerade berührt wird, oder Null, wenn kein Steuerelement berührt wird. Weitere Einzelheiten finden Sie im Abschnitt Erweiterte Grafiken.
- TOUCH(LASTREF)
  Gibt die Referenznummer des zuletzt berührten Steuerelements zurück

Der GUI BEEP Befehl

Der im OPTION TOUCH-Befehl angegebene Piezo-Summer kann auch von einem BASIC-Programm mit dem folgenden Befehl angesteuert werden:

```
GUI BEEP msec
```

Wobei „msec“ die Anzahl der Millisekunden ist, die der Beeper angesteuert werden soll. Eine Zeit von 3 ms erzeugt einen Klick, während 100 ms einen kurzen Piepton erzeugt.

Touch-Interrupts ohne erweiterte GUI-Steuerelemente

Auf die IRQ-Pin-Nummer, die bei der Konfiguration der Touch-Funktion angegeben wurde, kann ein Interrupt gesetzt werden. Um die Berührung zu erkennen, sollte der Interrupt als INT L (fallend) konfiguriert werden.

Wenn der Befehl OPTION TOUCH 7, 15 verwendet wurde um die Berührung zu konfigurieren gibt das folgende Programm die X- und Y-Koordinaten jeder Berührung auf dem Bildschirm aus:

```
SETPIN 15, INTL, MyInt
DO : LOOP

SUB MyInt
  PRINT TOUCH(X) TOUCH(Y)
END SUB
```

Der Interrupt kann mit dem Befehl SETPIN pin, OFF aufgehoben werden..

Touch-Interrupts mit erweiterten GUI-Steuerelementen

Wenn die erweiterten GUI-Steuerungen aktiviert sind (indem die Anzahl der GUI-Steuerungen unter Verwendung von OPTION GUI CONTROLS auf eine Zahl ungleich Null gesetzt wird), wird stattdessen der GUI-INTERRUPT-Befehl verwendet, um eine Berührungsunterbrechung einzurichten. Die Syntax lautet:

```
GUI INTERRUPT down [, up]
```

Wobei „down“ das Unterprogramm ist, das aufgerufen wird, wenn ein Aufsetzen erkannt wurde. Und optional ist 'up' die Subroutine, die aufgerufen wird, wenn die Berührung vom Bildschirm aufgehoben wurde ('up' und 'down' können bei Bedarf auf dieselbe Subroutine zeigen).

Als Beispiel druckt das folgende Programm die X- und Y-Koordinaten jeder Berührung auf dem Bildschirm aus:

```
GUI INTERRUPT MyInt
DO : LOOP

SUB MyInt
  PRINT TOUCH(X) TOUCH(Y)
END SUB
```

Die Angabe der Zahl Null (eine Ziffer) als Argument löscht sowohl Aufwärts- als auch Abwärts-Interrupts d.h:

```
GUI INTERRUPT 0
```

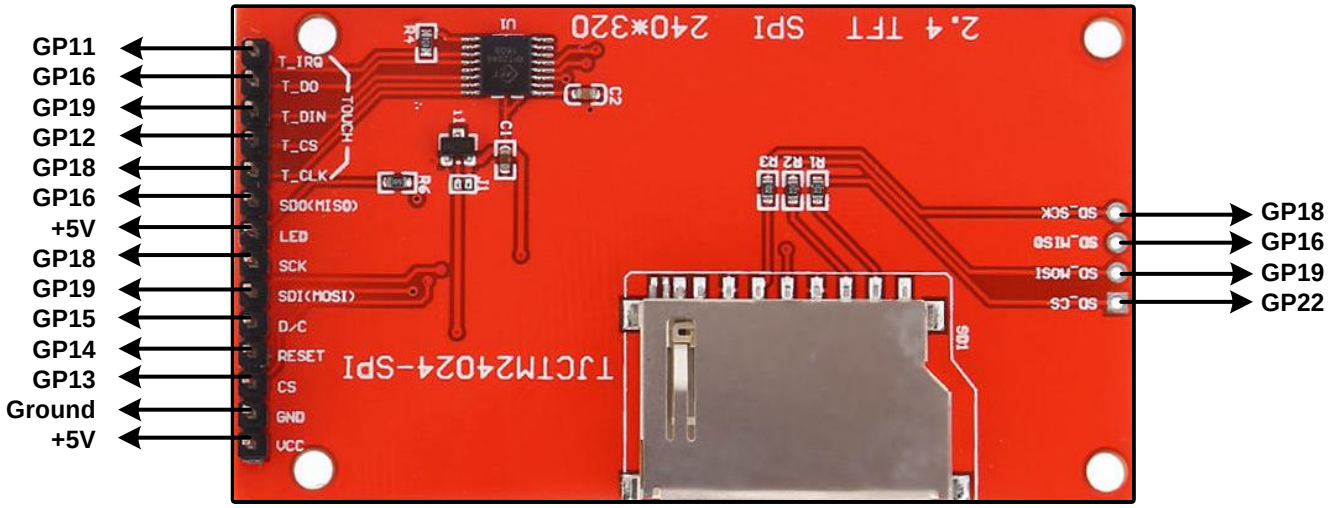
Verwendung eines LCD

Beispiel

Das Folgende ist eine Zusammenfassung, wie ein typisches LCD-Panel mit einem ILI9341-Controller angeschlossen werden kann. Dieses Beispiel unterstützt den SD-Kartensteckplatz, das LCD-Display und die Touch-Oberfläche.

Typische Panels finden Sie auf ebay.com und ähnlichen Seiten, indem Sie nach dem Schlüsselwort „ILI9341“ suchen. Stellen Sie sicher, dass die Anschlüsse auf der Rückseite des Panels wie unten gezeigt aussehen:

Das Panel sollte wie abgebildet mit dem PicoMite verbunden werden:



Um die obigen Verbindungen anzupassen, sollten die folgenden Konfigurationsbefehle nacheinander an der Eingabeaufforderung eingegeben werden:

```
OPTION SYSTEM SPI GP18, GP19, GP16
OPTION SDCARD GP22
OPTION LCDPANEL ILI9341, L, GP15, GP14, GP13
OPTION TOUCH GP12, GP11
```

Diese Befehle werden gespeichert und beim Einschalten automatisch angewendet. Beachten Sie, dass PicoMite nach Eingabe jedes Befehls neu startet und die USB-Verbindung unterbrochen wird und neu verbunden werden muss. Als nächstes sollte der Touchscreen kalibriert werden:

```
GUI CALIBRATE
```

Anschließend können Sie die verschiedenen Komponenten testen. Im Folgenden werden die Dateien auf der SD-Karte aufgelistet. Wenn sie fehlerfrei ausgeführt wird, können Sie sicher sein, dass die SD-Kartenschnittstelle in Ordnung ist.

```
FILES
```

Im Folgenden werden mehrere bunte, sich überlappende Kreise auf dem LCD-Bildschirm gezeichnet, die bestätigen, dass das LCD richtig angeschlossen ist:

```
GUI TEST LCDPANEL
```

Abschließend wird im Folgenden die Touch-Oberfläche getestet. Wenn Sie den LCD-Bildschirm berühren, sollte genau an der Berührungsstelle ein Punkt auf dem Bildschirm erscheinen.

```
GUI TEST TOUCH
```

Wenn dies nicht korrekt ist, müssen Sie den GUI CALIBRATE-Befehl möglicherweise ein zweites Mal ausführen und dabei größere Sorgfalt walten lassen.

Wenn Sie Probleme haben, das Display zum Laufen zu bringen, lohnt es sich, alles zu trennen und die Optionen mit dem Befehl OPTION CLEAR zu löschen, damit Sie mit einer sauberen Weste beginnen können. Schließen Sie es dann Stufe für Stufe wieder an und konfigurieren und testen Sie jede neue Stufe, während Sie fortfahren. Zuerst die SD-Kartenschnittstelle, dann das LCD-Display und schließlich die Touch-Schnittstelle.

Beachten Sie auch, dass der ILI9341-Controller empfindlich auf statische Entladungen reagiert. Wenn das Panel nicht reagiert, könnte es beschädigt werden und es wäre einen Test mit einem anderen Panel wert..

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.

Colours

Farbe wird als 24-Bit-True-Color-Zahl angegeben, wobei die oberen acht Bits die Intensität der roten Farbe darstellen, die mittleren acht Bits die grüne Intensität und die unteren acht Bits die blaue. Der einfachste Weg, diese Zahl zu generieren, ist mit der Funktion RGB(), die die Form hat:

```
RGB(red, green, blue)
```

Die RGB()-Funktion unterstützt auch eine Verknüpfung, mit der Sie gemeinsame Farben angeben können, indem Sie sie benennen. Zum Beispiel RGB(Rot) oder RGB(Cyan). Die Farben, die mit dem Shortcut-Formular benannt werden können, sind Weiß, Schwarz, Blau, Grün, Cyan, Rot, Magenta, Gelb, Braun, Weiß, Orange, Pink, Gold, Lachs, Beige, Hellgrau und Grau (oder USA-Schreibweise Grau / hellgrau). MMBasic übersetzt automatisch alle Farben in das vom jeweiligen Display-Controller benötigte Format. Im Fall des ILI9341-Controllers sind dies beispielsweise 64K-Farben im 565-Format. Die Standardeinstellung für Befehle, die einen Farbparameter erfordern, kann mit dem COLOR-Befehl (kann auch COLOR geschrieben werden) festgelegt werden. Dies ist praktisch, wenn Ihr Programm ein konsistentes Farbschema verwendet. Sie können dann die Standardeinstellungen festlegen und die Kurzversion der Zeichenbefehle im gesamten Programm verwenden. Der COLOR-Befehl hat folgendes Format:

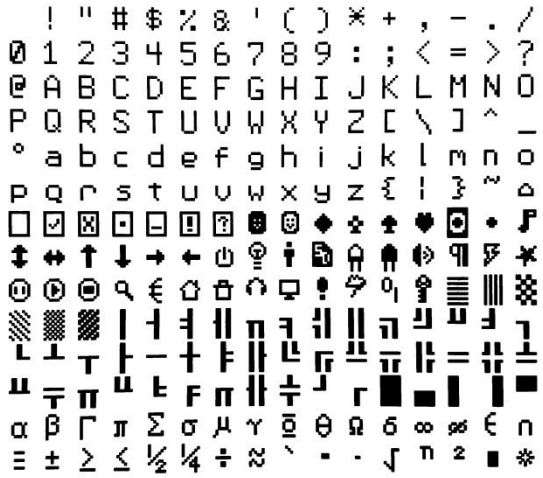
```
COLOUR foreground-colour, background-colour
```

Schriftarten

Es gibt sieben eingebaute Schriftarten :

Nummer	Breite x Höhe	Zeichensatz	Beschreibung
1	8 x 12	Alle 95ASCII Zeichen + 7F to FF (hex)	Standard bei Start Up
2	12 x 20	Alle 95ASCII Zeichen	Mittl. Größe
3	16 x 24	Alle 95ASCII Zeichen	Größerer Zeichensatz
4	10x16	Alle 95ASCII Zeichen + 7F to FF (hex)	Zeichensatz mit erweit. Grafik, für HR-Displays
5	24 x 32	Alle 95ASCII Zeichen	Zeichens. XL, sehr deutlich
6	32 x 50	0 - 9 + Einzelsymb.	0-9 + ".", "+","-", "=", "°", Colonsymbole. Sehr deutlich.
7	6 x 8	Alle 95ASCII Zeichen	Klein. Zeichensatz, nützlich bei niedr. Aufl.
8	6 x 4	Alle 95ASCII Zeichen	Noch kleinerer Zeichensatz

In allen Schriftarten (einschließlich Schriftart Nr. 6) wurde das Backquote-Zeichen (60 Hex oder 96 Dezimal) durch das Gradsymbol (º) ersetzt. Schriftart Nr. 1 (Standardschriftart) und Schriftart Nr. 4 haben einen erweiterten Zeichensatz, der alle Zeichen von CHR$(32) bis CHR$(255) oder 20 bis FF (Hex) abdeckt, wie rechts dargestellt. Bei Bedarf können zusätzliche Schriftarten in ein BASIC-Programm eingebettet werden. Diese Schriftarten funktionieren genauso wie die eingebaute Schriftart (d. h. mit dem FONT-Befehl ausgewählt oder im TEXT-Befehl angegeben).



Das Format einer eingebetteten Schriftart ist:

```
DefineFont #Nbr
    hex [[ hex[…]
    hex [[ hex[…]
```

```
END DefineFont
```

Er muss mit dem Schlüsselwort „DefineFont“ beginnen, gefolgt von der Schriftnummer (der optional ein #-Zeichen vorangestellt werden kann). Jede Schriftartnummer im Bereich von 2 bis 5 und 8 bis 16 kann angegeben werden, und wenn sie mit einer integrierten Schriftart identisch ist, wird diese Schriftart ersetzt. Der Hauptteil der Schriftart ist eine Folge von 8-stelligen Hex-Wörtern, wobei jedes Wort durch ein oder mehrere Leerzeichen oder eine neue Zeile getrennt ist. Die Fontdefinition wird durch ein "End DefineFont" beendet.

Stichwort. Diese können überall in einem Programm platziert werden und MMBasic überspringt sie. Dieses Format ist dasselbe wie das von Micromite verwendete. Zusätzliche Schriftarten und Informationen finden Sie im Ordner „Embedded Fonts“ im PicoMite-Firmware-Download. Diese Schriftarten decken eine breite Palette von Zeichensätzen ab, einschließlich einer Symbolschrift (Dingbats), die zum Erstellen von Bildschirmsymbolen usw. praktisch ist. Zusätzlich zur Verwendung eingebetteter Schriftarten kann ein Programm mit dem Befehl LOAD FONT dynamisch eine Schriftart von der SD-Karte laden. Ein Programm kann mit dieser Methode während seiner Ausführung viele Schriftarten laden, aber jede neue Schriftart überschreibt die zuvor geladene Schriftart. Das Format der mit LOAD FONT geladenen Schriftarten hat ein ähnliches Format wie die oben beschriebenen eingebetteten Schriftarten, außer dass keine Kommentare oder Leerzeilen erlaubt sind, die Schriftartnummer immer #8 sein muss, das erste Wort in einer eigenen Zeile stehen muss und die Die folgenden Zeilen (außer der letzten) müssen genau acht Wörter pro Zeile haben.

Als Beispiel ist im Folgenden eine winzige Schriftart (6 x 4 Pixel) dargestellt, die bei einem Bildschirm mit niedriger Auflösung nützlich ist. Diese kann entweder mit LOAD FONT geladen oder in das BASIC-Programm eingebettet werden:

```
DefineFont #8
  60200604
  44000000 00A04040 A0AEAE00 82406C6C EACC2048 00004460 84204424 E4A48044
  00E404A0 00800400 040000E0 00480240 4CE0AAEA 48C24044 C062C2E0 E820E2AA
  EA68E0E2 8048E2E0 EAE0EAEA 0404C0E2 80040400 0E208424 2484000E 4040E280
  4A60E84A CACAA0EA 608868C0 E8C0AACA E8E8E0E8 60EA6880 E4A0EAAA 2A22E044
  A0CAAA40 AEE08888 EEAEA0EA 40AA4AA0 4A80C8CA ECCA60AE C04268A0 AA4044E4
  A4AA60AA A0EEAA40 AAA04AAA 48E24044 E088E8E0 E2004208 004AE022 F0000000
  0C000084 AA8CE06A 608806C0 0660AA26 E42460AC 24AE0640 40A0CA88 22204044
  A0CC8AA4 0EE044C4 AA0CA0EE 40AA04A0 06C8AA0C 880662AA C0C60680 0A60444E
  AE0A60AA E0AE0A40 0AA0440A 6C0E24A6 608464E0 C4400444 006CC024 E0EEEE00
End DefineFont
```

Read Only-Variablen

Alle Koordinaten und Messungen auf dem Bildschirm erfolgen in Pixeln, wobei die X-Koordinate die horizontale Position und die Y-Koordinate die vertikale Position ist. Die linke obere Ecke des Bildschirms hat die Koordinaten X = 0 und Y = 0, die Werte warden größer wenn Sie sich auf dem Bildschirm nach unten und nach rechts bewegen.

Es gibt vier Nur-Lese-Variablen, die nützliche Informationen über das aktuell verbundene Display liefern:

- **MM. HRES**

  Gibt die Breite der Anzeige (die X-Achse) in Pixel zurück.
- **MM. VRES**

  Gibt die Höhe der Anzeige (die Y-Achse) in Pixel zurück.
- **MM.FONTHEIGHT**

  Gibt die Höhe der aktuellen Standardschriftart (in Pixel) zurück. Alle Zeichen in einer Schriftart haben die gleiche Höhe.
- **MM.SCHRIFTBREITE**

  Gibt die Breite eines Zeichens in der aktuellen Schriftart (in Pixel) zurück. Alle Zeichen haben die gleiche Breite..

Zeichenbefehle

Es gibt neun grundlegende Zeichenbefehle, die Sie innerhalb von MMBasic-Programmen auf dem PicoMite verwenden können, um mit einem angeschlossenen LCD-Display zu interagieren. Es gibt auch eine Reihe leistungsfähigerer GUI-Befehle zum Zeichnen von Schaltern, Optionsfeldern usw. Weitere Einzelheiten finden Sie im nächsten Abschnitt Erweiterte Grafiken.

Die meisten grundlegenden Zeichenbefehle haben optionale Parameter. Sie können diese am Ende eines Befehls vollständig weglassen oder zwei Kommas hintereinander verwenden, um auf einen fehlenden Parameter hinzuweisen. Beispielsweise ist der fünfte Parameter des LINE-Befehls optional, sodass Sie dieses Format verwenden können:

```
LINE 0, 0, 100, 100, , rgb(red)
```

Optionale Parameter sind unten kursiv gekennzeichnet, zum Beispiel: Schriftart.

In den folgenden Befehlen ist C die Zeichenfarbe und standardmäßig die aktuelle Vordergrundfarbe. FILL ist die Füllfarbe, die standardmäßig -1 ist, was anzeigt, dass keine Füllung verwendet werden soll.

Die Zeichenbefehle sind:

- `CLS` *`C`*
  Löscht den Bildschirm auf die Farbe C. Wenn C nicht angegeben ist, wird die aktuelle Standardhintergrundfarbe verwendet.

- `PIXEL X, Y,` *`C`*
  Beleuchtet ein Pixel. Wenn C nicht angegeben ist, wird die aktuelle Standard-Vordergrundfarbe verwendet.

- `LINE X1, Y1, X2, Y2,` *`LW, C`*
  Zeichnet eine Linie, die bei X1 und Y1 beginnt und bei X2 und Y2 endet. LW ist die Breite der Linie und gilt nur für horizontale oder vertikale Linien. Der Standardwert ist 1 soweit nicht anders angegeben oder wenn es sich um eine Diagonale handelt.

- `BOX X, Y, W, H,` *`LW, C, FILL`*
  Zeichnet ein Kästchen beginnend bei X und Y, das W Pixel breit und H Pixel hoch ist. LW ist die Breite der Seiten der Box und kann Null sein. Es ist standardmäßig auf 1.

- `RBOX X, Y, W, H,` *`R, C, FILL`*
  Zeichnet eine Box mit abgerundeten Ecken beginnend bei X und Y die W Pixel breit und H Pixel hoch ist. R ist der Radius der Ecken der Box. Der Standardwert ist 10.

- `CIRCLE X, Y, R,` *`LW, A, C, FILL`*
  Zeichnet einen Kreis mit X und Y als Mittelpunkt und einem Radius R. LW ist die Breite der Linie, die für den Umfang verwendet wird, und kann Null sein (Standardwert 1). A ist das Seitenverhältnis, das eine Fließkommazahl ist und standardmäßig 1 ist. Beispielsweise zeichnet ein Seitenverhältnis von 0,5 ein Oval, dessen Breite halb so hoch ist.

- `TEXT X, Y, STRING,` *`ALIGNMENT, FONT, SCALE, C, BC`*
  Zeigt eine Zeichenfolge an, die bei X und Y beginnt. AUSRICHTUNG besteht aus 0, 1 oder 2 Zeichen (ein Zeichenfolgenausdruck oder eine Variable ist ebenfalls zulässig), wobei der erste Buchstabe die horizontale Ausrichtung um X herum darstellt und L, C oder R für LEFT, CENTER oder sein kann RECHTS ausgerichteter Text und der zweite Buchstabe ist die vertikale Ausrichtung um Y herum und kann T, M oder B für OBEN, MITTEL oder UNTEN ausgerichteten Text sein. Die Standardausrichtung ist links/oben. Ein zusätzlicher Kennbuchstabe kann verwendet werden, um den Text zu drehen (Details siehe unten). FONT und SCALE sind optional und werden standardmäßig durch den FONT-Befehl festgelegt. C ist die Zeichenfarbe und BC ist die Hintergrundfarbe. Sie sind optional und werden standardmäßig durch den COLOR-Befehl gesetzt.

- `GUI BITMAP X, Y, BITS,` *`WIDTH, HEIGHT, SCALE, C, BC`*
  Zeigt die Bits in einer Bitmap beginnend bei X und Y an. HÖHE und BREITE sind die Abmessungen der Bitmap, wie sie auf dem LCD-Feld angezeigt werden, und sind standardmäßig 8x8. SCALE, C und BC sind die gleichen wie beim TEXT-Befehl. Die Bitmap kann eine Ganzzahl oder eine String-Variable oder -Konstante sein und wird mit dem ersten Byte als den ersten Bits der obersten Zeile gezeichnet (Bit 7 zuerst, dann Bit 6 usw.), gefolgt vom nächsten Byte usw. Wenn die oberste Zeile gefüllt wurde, beginnt die nächste Zeile der angezeigten Bitmap mit dem nächsten Bit in der Ganzzahl oder Zeichenfolge.

- `POLYGON n, xarray%(), yarray%() [, bordercolour] [, fillcolour]`
  Zeichnet ein gefülltes oder umrissenes Polygon mit n xy-Koordinatenpaaren in xarray%() und yarray%(). Wenn „Füllfarbe“ weggelassen wird, wird nur der Polygonumriss gezeichnet. Wenn „Randfarbe“ weggelassen wird, wird standardmäßig die aktuelle Standard-Vordergrundfarbe verwendet.

Gedrehter Text

Wie oben beschrieben, kann die Ausrichtung des Textes im TEXT-Befehl angegeben werden, indem ein oder zwei Zeichen in einem Zeichenfolgenausdruck für den dritten Parameter des Befehls verwendet werden. In dieser Zeichenfolge können Sie auch ein drittes Zeichen angeben, um die Drehung des Textes anzuzeigen. Dieses Zeichen kann eines sein von:

N für normale Ausrichtung

V für vertikalen Text, wobei jedes Zeichen unter dem vorherigen von oben nach unten verläuft.

I der Text wird invertiert (d.h. auf dem Kopf)

U Der Text wird um 90º gegen den Uhrzeigersinn gedreht

D Der Text wird um 90º im Uhrzeigersinn gedreht

Diese zusätzliche Funktion gilt für die Befehle TEXT und GUI CAPTION.

Als Beispiel wird im Folgenden der Text „LCD Display“ vertikal am linken Rand des Anzeigefelds und vertikal zentriert angezeig:

```
TEXT 0, 250, "LCD Display", "LMV", 5
```

Die Positionierung erfolgt relativ zur oberen linken Ecke des Zeichens bei normaler Betrachtung, sodass bei umgekehrten 100.100 das obere linke Pixel des ersten Zeichens bei 100.100 liegt und der Text dann über y=101 und links von x=101 liegt. In ähnlicher Weise wird „R“ in der Ausrichtungszeichenfolge aus der Perspektive des Zeichens betrachtet, unabhängig davon, in welcher Ausrichtung es sich befindet (nicht vom Bildschirm).

Transparenter Text

Wenn das Display transparenten Text darstellen kann, erlaubt der TEXT-Befehl die Verwendung von -1 für die Hintergrundfarbe. Das bedeutet, dass der Text über den Hintergrund gezeichnet wird, wobei das Hintergrundbild durch die Lücken in den Buchstaben hindurchscheint. Kompatible Displays verwenden die Controller ILI9341 und ST7789_320.

BLIT-Befehl

Wenn die Anzeige transparenten Text darstellen kann, ermöglicht der BLIT-Befehl, dass ein Teil des aktuell auf der Anzeige angezeigten Bildes in einen Speicherpuffer und später zurück auf die Anzeige kopiert wird. Dies ist nützlich, wenn etwas über den Hintergrund gezeichnet und später entfernt werden muss, ohne das Bild im Hintergrund zu beschädigen. Beispiele hierfür sind ein Spiel, bei dem sich eine Figur vor einer Landschaft bewegt, oder die sich bewegende Nadel einer fotorealistischen Anzeige.

Die verfügbaren Befehle sind:

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 ist die Puffernummer im Bereich von 1 bis 8. x und y sind die Koordinaten der oberen linken Ecke und w und h sind die Breite und Höhe des Bildes. READ kopiert das Anzeigebild in den Puffer, WRITE kopiert den Puffer in die Anzeige und CLOSE gibt den Puffer frei und fordert den verwendeten Speicher zurück. LOAD lädt eine Bilddatei in den Puffer.

BLIT LOAD und BLIT WRITE funktionieren auf jedem Display, während BLIT und BLIT READ nur auf Displays funktionieren, die transparenten Text verarbeiten können (d. h. mit den Controllern ILI9341 oder ST7789_320).

Diese Befehle können verwendet werden, um einen Teil der Anzeige an einen anderen Ort zu kopieren (indem in einen Puffer kopiert und dann woanders geschrieben wird), aber eine einfachere Methode besteht darin, eine alternative Version des BLIT-Befehls wie folgt zu verwenden:

BLIT x1, y1, x2, y2, w, h

Dadurch wird ein Teil des Bildes bei x1/y1 an die Position x2/y2 kopiert. w und h geben die Breite und Höhe des zu kopierenden Bildes an. Die Quell- und Zielbereiche können sich überlappen und der BLIT-Befehl wird die Kopie korrekt durchführen.

Diese Form des BLIT-Befehls ist besonders nützlich zum Erstellen von Diagrammen, die horizontal oder vertikal scrollen können, wenn neue Daten hinzugefügt werden.

Bild Laden

Wie zuvor im Abschnitt "Unterstützung von SD-Karten" beschrieben, kann der Befehl LOAD IMAGE verwendet werden, um ein Bitmap-Bild von der SD-Karte zu laden und es auf dem LCD-Display anzuzeigen. Dies kann verwendet werden, um ein Logo zu zeichnen oder den auf dem Display gezeichneten Grafiken einen kunstvollen Hintergrund hinzuzufügen.

Beispiel

Als Beispiel zeichnet das folgende Programm eine einfache Digitaluhr auf einem ILI9341-basierten LCD-Display. Das Programm wird beendet und kehrt zur Eingabeaufforderung zurück, wenn der Anzeigebildschirm berührt wird. Zunächst müssen die Anzeige- und Touch-Optionen konfiguriert werden, indem die am Anfang dieses Kapitels aufgeführten Befehle eingegeben werden. Das genaue Format davon hängt davon ab, wie Sie das Anzeigefeld angeschlossen haben. Geben Sie dann das Programm ein und führen Sie es aus:

```
CONST DBlue = 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), DBlue

DO
  TEXT MM.HRes/2, MM.VRes/4, TIME$, "CM", 1, 4, RGB(CYAN), DBlue
  TEXT MM.HRes/2, MM.VRes*3/4, DATE$, "CM"
  IF TOUCH(X) <> -1 THEN END
LOOP
```

Dieses Programm definiert zunächst eine Konstante mit einem Wert, der einer dunkelblauen Farbe entspricht, und legt dann die Standardeinstellungen für die Farben und die Schriftart fest. Dann zeichnet es eine Kiste mit roten Wänden und einem dunkelblauen Innenraum. Danach tritt das Programm in eine Endlosschleife ein, in der es drei Funktionen ausführt:

1. Zeigt die aktuelle Uhrzeit im zuvor gezeichneten Feld an. Die Schnur wird sowohl horizontal als auch vertikal in der Mitte der Box zentriert gezeichnet. Beachten Sie, dass der TEXT-Befehl sowohl die Standardschriftart als auch die Farben überschreibt, um seine eigenen Parameter festzulegen.

2. Zeichnet das Datum zentriert in die untere Hälfte des Bildschirms. In diesem Fall verwendet der TEXT-Befehl die zuvor eingestellte Standardschriftart und -farben.

3. Sucht nach einer Berührung auf dem Bildschirm. Dies wird angezeigt, wenn die Funktion TOUCH(X) etwas anderes als -1 zurückgibt. In diesem Fall wird das Programm beendet. Die Bildschirmanzeige sollte wie folgt aussehen (die in dieser Abbildung verwendete Schriftart ist anders):

Erweiterte Grafik

PicoMite enthält eine Reihe fortschrittlicher grafischer Steuerelemente, die auf Berührungen reagieren, darunter Bildschirmschalter, Schaltflächen, Anzeigeleuchten, Tastatur usw. MMBasic zeichnet das Steuerelement und animiert es (d. h. ein Schalter scheint sich zu drücken, wenn er berührt wird). Alles, was das BASIC-Programm tun muss, ist einen einzelnen Befehl aufzurufen, um die grundlegenden Details der Steuerung zu spezifizieren.

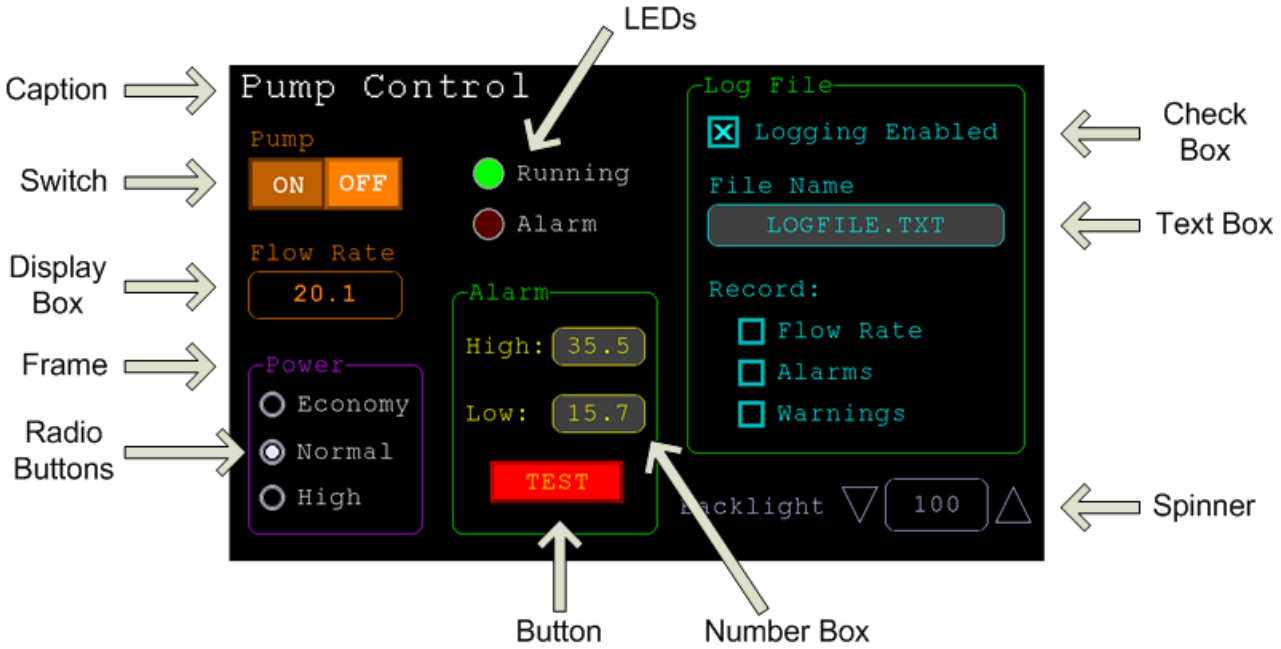
Um die GUI-Steuerelemente in PicoMite verwenden zu können, muss zuerst der für die GUI-Steuerelemente benötigte Speicher mit dem Befehl OPTION GUI CONTROLS zugewiesen werden. Normalerweise würden Sie den Befehl wie folgt verwenden:

```
OPTION GUI CONTROLS 75
```

Dadurch wird die maximale Anzahl von Steuerelementen, die Sie definieren können, auf 75 festgelegt. Diese Option ist dauerhaft (d. h. sie wird beim Ausschalten gespeichert). Standardmäßig ist die maximale Anzahl von Steuerelementen auf null gesetzt und in diesem Fall sind die GUI-Funktionen nicht verfügbar und es wird kein Speicher verwendet.

Steuerelemente definieren

Dies sind einige der erweiterten GUI-Steuerelemente, die Sie verwenden können:



Jedes Steuerelement hat eine Referenznummer namens „#ref“ in der Beschreibung des Steuerelements. Dies kann eine beliebige Zahl zwischen 1 und der durch den Befehl OPTION CONTROL festgelegten Obergrenze sein. Diese Referenznummer wird verwendet, um eine Kontrolle zu identifizieren. Beispielsweise kann ein Kontrollkästchen mit der Referenznummer #10 erstellt werden:

```
GUI CHECKBOX #10, "Test", 100, 100, 50, rgb(BLUE)
```

Nach der Erstellung kann der Benutzer das Kästchen mit der Touch-Funktion des LCD-Panels aktivieren und deaktivieren, ohne dass das laufende BASIC-Programm beteiligt ist. Bei Bedarf kann das Programm den Wert des Kontrollkästchens ermitteln, indem es seine Referenznummer in der Funktion CtrlVal() verwendet:

```
IF CtrlVal(#10) THEN ...
```

Das #-Zeichen ist optional, dient aber dazu, den Programmierer daran zu erinnern, dass dies keine gewöhnliche Zahl ist.

In den folgenden Befehlen sind alle Argumente in Kursivschrift (z. B. Breite, Höhe) optional und nehmen, wenn sie nicht angegeben werden, den Wert des vorherigen Befehls an, der sie angegeben hat. So können beispielsweise mehrere Radiobuttons gleicher Größe und Farbe angegeben werden, wobei nur der erste Button alle Details auflisten muss. Beachten Sie, dass sich dies bei der Farbspezifikation von den grundlegenden Zeichenbefehlen unterscheidet, die standardmäßig auf den letzten COLOR-Befehl eingestellt sind.

Alle Zeichenfolgen, die in GUI-Steuerelementen und der MsgBox verwendet werden, können mehrere Zeilen anzeigen, indem Sie das Tilde-Zeichen (~) verwenden, um jede Zeile in der Zeichenfolge zu trennen. Die Beschriftung einer Drucktaste kann beispielsweise "ALARM~TEST" lauten und wird in zwei Zeilen angezeigt.

Für alle Steuerelemente wird die Schriftart für die Beschriftung verwendet, die mit dem FONT-Befehl eingestellt wurde, und die Farben werden mit dem letzten COLOR-Befehl eingestellt.

Wenn die Anzeige transparenten Text darstellen kann, erlauben diese Befehle die Verwendung von -1 für die Hintergrundfarbe. Das bedeutet, dass der Text über den Hintergrund gezeichnet wird, wobei das Hintergrundbild durch die Lücken in den Buchstaben hindurchscheint:

Rahmen

`GUI FRAME #ref, caption$, StartX, StartY, Width, Height, Colour`

Dadurch wird ein Rahmen gezeichnet, der ein Kästchen mit runden Ecken und einer Beschriftung ist. Ein Rahmen reagiert nicht auf Berührungen, ist aber nützlich, wenn eine Gruppe von Steuerelementen visuell zusammengeführt werden muss. Es kann auch verwendet werden, um eine Gruppe von Optionsfeldern einzuschließen, und MMBasic sorgt dafür, dass die vom Rahmen umgebenen Optionsfelder exklusiv sind – das heißt, wenn ein Optionsfeld ausgewählt wird, werden alle anderen Schaltflächen, die ausgewählt waren und sich innerhalb des Rahmens befinden, deselektiert.

LED

`GUI LED #ref, caption$, CenterX, CenterY, Diameter, Colour`

Dadurch wird eine Anzeigeleuchte gezeichnet (sie sieht aus wie eine auf einer Platte montierte LED). Wenn sein Wert auf eins gesetzt ist, wird er beleuchtet und wenn er auf null gesetzt wird, ist er aus (eine stumpfe Version seines Farbattributs). Die LED kann zum Blinken gebracht werden, indem ihr Wert auf die Anzahl von Millisekunden eingestellt wird, die sie vor dem Ausschalten eingeschaltet bleiben soll.

Die Beschriftung wird rechts von der LED gezeichnet und verwendet die Farben, die durch den COLOR-Befehl festgelegt wurden. Die LED-Steuerung wird bei Berührung nicht animiert, aber ihre Referenznummer kann mit TOUCH(REF) und TOUCH(LASTREF) in den Berührungsunterbrechungen gefunden werden, und jede erforderliche Animation kann in MMBasic durchgeführt werden.

Kontrollkästchen

`GUI CHECKBOX #ref, caption$, StartX, StartY, Size, Colour`

Dadurch wird ein Kontrollkästchen gezeichnet, das ein kleines Kästchen mit einer Beschriftung ist. Sowohl die Höhe als auch die Breite werden mit dem Parameter „Größe“ angegeben. Bei Berührung wird ein X in das Kästchen gezeichnet, um anzuzeigen, dass diese Option ausgewählt wurde, und der Wert des Steuerelements wird auf 1 gesetzt. Bei einer zweiten Berührung wird das Häkchen entfernt und der Wert des Steuerelements ist Null. Die Beschriftung wird rechts vom Kontrollkästchen gezeichnet und verwendet die Farben, die durch den COLOR-Befehl festgelegt wurden.

Druckknopf

`GUI BUTTON #ref, caption$, StartX, StartY, Width, Height, FColour, BColour`

Dadurch wird eine momentane Schaltfläche gezeichnet, bei der es sich um einen quadratischen Schalter mit der Beschriftung auf der Vorderseite handelt. Bei Berührung erscheint das sichtbare Bild der Schaltfläche gedrückt und der Wert des Steuerelements ist 1. Wenn die Berührung entfernt wird, wird der Wert auf Null zurückgesetzt. Die Beschriftung kann eine einzelne Zeichenfolge mit zwei Beschriftungen sein, die durch einen vertikalen Strich (|) getrennt sind (z. B. "UP|DOWN"). Wenn die Taste oben ist, wird die erste Saite verwendet und wenn sie gedrückt wird, wird die zweite verwendet.

Schalter

`GUI SWITCH #ref, caption$, StartX, StartY, Width, Height, FColour, BColour`

Dadurch wird ein Verriegelungsschalter mit der Beschriftung auf seiner Vorderseite gezeichnet. Bei Berührung erscheint das sichtbare Bild der Schaltfläche gedrückt und der Wert des Steuerelements ist 1. Bei einer zweiten Berührung wird der Schalter losgelassen und der Wert wird auf Null zurückgesetzt. Beschriftung kann eine einzelne Zeichenfolge mit zwei Beschriftungen sein, die durch ein | getrennt sind Zeichen (z. B. "ON|OFF"). Wenn dies verwendet wird, erscheint der Schalter als Kippschalter, wobei jede Hälfte der Beschriftung verwendet wird, um jede Hälfte des Kippschalters zu beschriften.

Radiobutton

`GUI RADIO #ref, caption$, CenterX, CenterY, Radius, Colour`

Dadurch wird ein Optionsfeld mit einer Beschriftung gezeichnet. Bei Berührung leuchtet die Mitte der Schaltfläche auf, um anzuzeigen, dass diese Option ausgewählt wurde, und der Wert des Steuerelements ist 1. Wenn eine andere Optionsschaltfläche ausgewählt wird, wird die Markierung auf dieser Schaltfläche entfernt

und ihr Wert ist Null. Optionsfelder werden gruppiert, wenn sie von einem Rahmen umgeben sind, und wenn ein Feld in der Gruppe ausgewählt wird, werden alle anderen in der Gruppe deselektiert. Wenn kein Rahmen verwendet wird, werden alle Schaltflächen auf dem Bildschirm gruppiert.

Die Beschriftung wird rechts von der Schaltfläche gezeichnet und verwendet die Farben, die durch den COLOR-Befehl festgelegt wurden.

Displaybox

`GUI DISPLAYBOX #ref, StartX, StartY, Width, Height, FColour, BColour`

Dadurch wird eine Box mit abgerundeten Ecken gezeichnet. Mit dem Befehl CtrlVal(r) = kann jede Zeichenfolge im Feld angezeigt werden. Dies ist nützlich, um Text, Zahlen und Nachrichten anzuzeigen. Dieses Steuerelement wird nicht animiert, wenn es berührt wird, aber seine Referenznummer kann unter Verwendung von TOUCH(REF) und TOUCH(LASTREF) in den Berührungsunterbrechungen gefunden werden, und jede erforderliche Animation kann in MMBasic durchgeführt werden.

Textbox

`GUI TEXTBOX #ref, StartX, StartY, Width, Height, FColour, BColour`

Dadurch wird eine Box mit abgerundeten Ecken gezeichnet. Wenn das Feld berührt wird, erscheint eine QWERTZ-Tastatur auf dem Bildschirm, wie rechts gezeigt. Mit dieser virtuellen Tastatur kann ein beliebiger Text in das Feld eingegeben werden, einschließlich Groß-/Kleinbuchstaben, Zahlen und alle anderen Zeichen des ASCII-Zeichensatzes. Der neue Text ersetzt jeden Text, der sich zuvor im Feld befand.

Ent ist die Eingabetaste, Can ist die Abbruchtaste und schließt das Textfeld und bringt es in seinen ursprünglichen Zustand zurück, das Dreieck ist die Umschalttaste, die Taste [ ] fügt ein Leerzeichen ein und die Taste &12 wählt eine alternative Tastenauswahl aus mit Zahlen und Sonderzeichen (es gibt zwei Gruppen von Sonderzeichen und die Umschalttaste schaltet zwischen ihnen um).

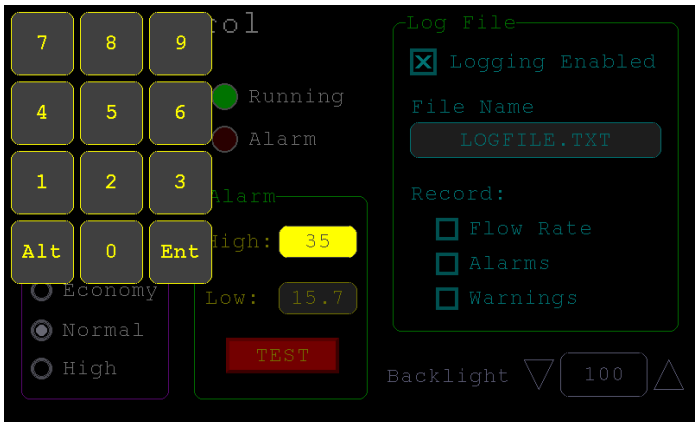
Der angezeigte String kann eingestellt werden, indem dem Feld mit dem Befehl CtrlVal(r) = ein String zugewiesen wird. Der Wert des Steuerelements kann auch auf eine Zeichenfolge gesetzt werden, die mit zwei Rautenzeichen (##) beginnt, und in diesem Fall wird die Zeichenfolge (ohne die führenden zwei Rautenzeichen) in der Box mit reduzierter Helligkeit angezeigt. Dies kann verwendet werden, um dem Benutzer einen Hinweis darauf zu geben, was eingegeben werden sollte (sogenannter "Ghost-Text"). Das Lesen des Werts des Steuerelements, das Geistertext anzeigt, gibt eine leere Zeichenfolge zurück. Wenn eine Taste gedrückt wird, verschwindet der Geistertext und wird durch den eingegebenen Text ersetzt.

MMBasic versucht, die virtuelle Tastatur auf dem Bildschirm so zu positionieren, dass das Textfeld, das sie angezeigt hat, nicht verdeckt wird. Eine Stift-unten-Unterbrechung wird generiert, kurz bevor die Tastatur aktiviert wird, und eine Taste-oben-Unterbrechung wird generiert, wenn die Eingabe- oder Abbrechen-Tasten berührt werden und die Tastatur ausgeblendet wird.

Bei Bedarf kann die virtuelle Tastatur mit dem Befehl GUI TEXTBOX ACTIVATE erzwungen werden, ohne dass das Steuerelement berührt wird, und vom Programm (wie beim Berühren der Abbruchtaste) mit dem Befehl: GUI TEXTBOX CANCEL ausgeblendet werden.

Zahlenfeld

GUI NUMBERBOX #ref, StartX, StartY, *Width, Height, FColour, BColour*



Dadurch wird eine Box mit abgerundeten Ecken gezeichnet. Wenn das Kästchen berührt wird, erscheint eine numerische Tastatur auf dem Bildschirm, wie rechts gezeigt. Mit dieser virtuellen Tastatur kann eine beliebige Zahl in das Feld eingegeben werden, einschließlich einer Fließkommazahl im Exponentialformat. Die neue Nummer ersetzt die Nummer, die zuvor im Feld stand.

Die Alt-Taste wählt eine alternative Tastenauswahl und die anderen Sondertasten sind die gleichen wie beim Textfeld.

Die angezeigte Zahl kann auch eingestellt werden, indem dem Feld mit dem Befehl CtrlVal(r) = eine Zahl (Float oder Integer) zugewiesen wird. Ähnlich wie beim Textfeld kann der Wert des Steuerelements auf einen Literal-String mit zwei führenden Hash-Zeichen (z. B. "##Hint") gesetzt werden, und in diesem Fall wird der String (ohne die beiden führenden Zeichen) in dem Feld mit angezeigt reduzierte Helligkeit. Wenn Sie dies lesen, wird Null zurückgegeben, und wenn eine Taste gedrückt wird, verschwindet der Geistertext.

MMBasic wird versuchen, die virtuelle Tastatur auf dem Bildschirm so zu positionieren, dass das Zahlenfeld, das sie angezeigt hat, nicht verdeckt wird. Eine Stift-unten-Unterbrechung wird generiert, kurz bevor das Tastenfeld aktiviert wird, und eine Taste-oben-Unterbrechung wird generiert, wenn die Eingabetaste berührt wird und das Tastenfeld ausgeblendet wird. Auch wenn die Eingabetaste berührt wird, wird der eingegebene Text als Zahl ausgewertet und das NUMBERBOX-Steuerelement neu gezeichnet, um diese Zahl anzuzeigen.

Bei Bedarf kann das virtuelle Tastenfeld mit dem Befehl GUI NUMBERBOX ACTIVATE ohne Berührung des Controls eingeblendet und vom Programm (wie beim Berühren der Abbrechen-Schaltfläche) mit dem Befehl: GUI NUMBERBOX CANCEL ausgeblendet.

.

Formatiertes Zahlenfeld

GUI FORMATBOX #ref, Format, StartX, StartY, *Width, Height, FColour, BColour*

Dadurch wird eine Box mit abgerundeten Ecken gezeichnet. Wenn das Feld berührt wird, erscheint ein numerisches Tastenfeld ähnlich einem Nummernfeld. Der Unterschied besteht darin, dass das formatierte Nummernfeld den Benutzer auffordert, Zahlen gemäß einem bestimmten Format für Datum, Uhrzeit usw. einzugeben. Ungültige Tasten auf der Tastatur werden deaktiviert und der Benutzer wird durch seine Eingabe mit Führungstext geführt. Dadurch kann der Programmierer sicher sein, dass die Eingabe des Benutzers immer in einem festen Format erfolgt.

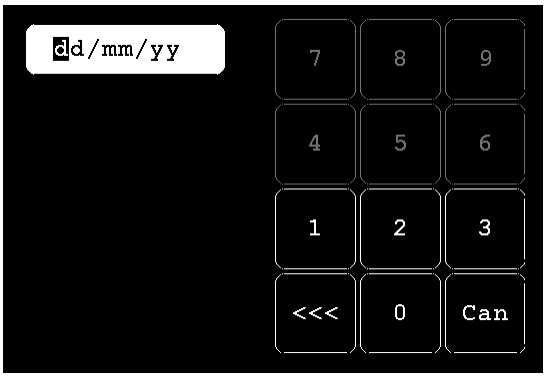
Die Art des Eintrags wird durch das Argument „Format“ wie folgt gesteuert:

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')

Beispiel:

```
GUI FORMATBOX #1, DATE1, 300, 150, 200, 50
```

würde ein Dateneingabefeld erstellen und wenn es berührt wird, erscheint eine Tastatur, wie rechts gezeigt. Beachten Sie, dass:



• Das Anzeigefeld ist mit einer Führungszeichenfolge gefüllt, um den Benutzer zu den erforderlichen Daten aufzufordern.

• Da der Tag des Monats nur mit einer Ziffer von 0 bis 3 beginnen kann, sind alle anderen Tasten gesperrt. Dies geschieht auch mit anderen Nummern, die eine begrenzte Reichweite haben.

• Der über CtrlVal(#1) abgerufene Wert des Steuerelements ist eine Zeichenfolge. Wenn der Benutzer beispielsweise das Datum für den 8. Mai 2020 eingegeben hat, wäre die zurückgegebene Zeichenfolge „08/05/20“ (d. h. das UK/Aust/NZ-Format, wie durch DATE1 angegeben).

Der Wert des Steuerelements kann mithilfe der Zeichenfolgenfunktionen auseinander gezogen werden, oder in einigen Fällen kann die Zeichenfolge direkt verwendet werden. Wenn Sie beispielsweise das obige Formatfeld verwenden, um ein Datum vom Benutzer zu erhalten, könnte die interne Uhr von PicoMite direkt wie folgt eingestellt warden:

```
DATE$ = CtrlVal(#1)
```

Der Befehl RTC SETTIME akzeptiert ein einzelnes Zeichenfolgenargument im Format tt/mm/jj hh:mm, sodass die RTC-Zeit ähnlich wie folgt eingestellt werden könnte, wenn das formatierte Feld DATETIME2 für „Format“ verwendet:

```
RTC SETTIME CtrlVal(#1)
```

Sie können den US-Stil DATETIME4 verwenden, um das Datum/die Uhrzeit zu erhalten. In diesem Fall würden Sie dies verwenden, um die RTC einzustellen:

```
RTC SETTIME MID$(CtrlVal(#1),4,3) + LEFT$(CtrlVal(#1),2) + RIGHT$((CtrlVal(#1),9)
```

MMBasic versucht, das virtuelle Tastenfeld auf dem Bildschirm so zu positionieren, dass das Formatfeld, das es angezeigt hat, nicht verdeckt wird. Eine Stift-unten-Unterbrechung wird erzeugt, wenn das Tastenfeld eingesetzt wird, und eine Taste-oben-Unterbrechung wird erzeugt, wenn alle erforderlichen Daten eingegeben wurden und das Tastenfeld ausgeblendet ist.

Bei Bedarf kann die virtuelle Tastatur mit dem Befehl GUI FORMATBOX ACTIVATE erzwungen werden, ohne dass das Steuerelement berührt wird, und sie kann vom Programm (wie beim Berühren der Abbrechen-Schaltfläche) mit dem Befehl: GUI FORMATBOX CANCEL ausgeblendet werden.

Spinbox

```
GUI SPINBOX #ref, StartX, StartY, Width, Height, FColour, BColour, Step,
        Minimum, Maximum
```

Dadurch wird ein Kästchen mit Auf-/Ab-Symbolen an beiden Enden gezeichnet. Wenn diese Symbole berührt werden, wird die Zahl im Feld um den 'StepValue' erhöht oder verringert, das Gedrückthalten der Berührung wird schnell wiederholt. „Minimum“ und „Maximum“ begrenzen den eingebbaren Wert.

'StepValue', 'Minimum' und 'Maximum' sind optional und wenn nicht angegeben, ist 'StepValue' 1 und es gibt keine Begrenzung für die eingegebene Zahl. Ein Pen-Down-Interrupt wird jedes Mal erzeugt, wenn Oben/Unten berührt wird oder wenn eine automatische Wiederholung auftritt.

Caption

```
GUI CAPTION #ref, text$, StartX, StartY, Alignment, FColour, BColour
```

Dadurch wird eine Textzeichenfolge auf dem Bildschirm gezeichnet. Es ähnelt dem grundlegenden Zeichenbefehl TEXT, mit dem Unterschied, dass MMBasic dieses Steuerelement automatisch dimmt, wenn eine Tastatur oder ein Nummernblock angezeigt wird.

„Ausrichtung“ besteht aus null bis drei Zeichen (ein Zeichenfolgenausdruck oder eine Variable ist ebenfalls zulässig), wobei der erste Buchstabe die horizontale Ausrichtung um X herum ist und L, C oder R für LINKS, MITTE, RECHTS sein kann und der zweite Buchstabe die vertikale Ausrichtung ist um Y und kann T, M oder B für TOP, MIDDLE, BOTTOM sein.

Ein drittes Zeichen kann verwendet werden, um die Rotation des Textes anzuzeigen. Dies kann 'N' für normale Ausrichtung sein, 'V' für vertikalen Text, wobei jedes Zeichen unter dem vorherigen von oben nach unten verläuft, 'I' der Text wird invertiert (dh auf dem Kopf stehend), 'U' der Text wird gedreht gegen den

Uhrzeigersinn um 90º und 'D' wird der Text um 90º im Uhrzeigersinn gedreht. Die Standardausrichtung ist links/oben ohne Drehung.

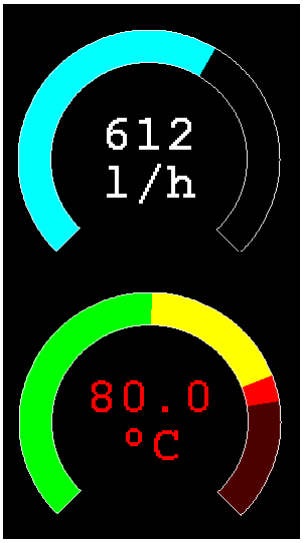
Wenn die Farben nicht angegeben sind, verwendet dieses Steuerelement die Farben, die durch den COLOR-Befehl festgelegt wurden.

Analoges Meßgerät

`GUI GAUGE #ref, StartX, StartY, Radius, FColour, BColour, min, max, nbrdec, units$, c1, ta, c2, tb, c3, tc, c4`

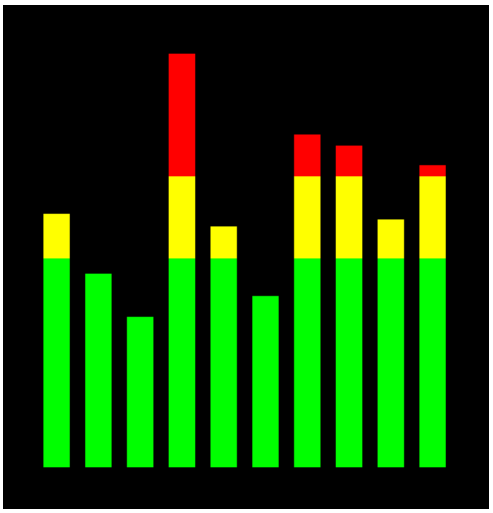
Dadurch wird ein grafisches kreisförmiges analoges Messgerät mit einer digitalen Anzeige in der Mitte definiert, die den Wert und die Einheiten anzeigt. Falls angegeben, wird das Messgerät farbig dargestellt, um den Signalpegel grafisch anzuzeigen (z. B. grün für OK, gelb für Warnung usw.).
„StartX“ und „StartY“ sind die Koordinaten des Mittelpunkts der Lehre, während „Radius“ der Abstand vom Mittelpunkt zum äußeren Rand ist.
„min“ ist der Wert, der dem Mindestwert des Messgeräts zugeordnet ist, und „max“ ist der Höchstwert. Wenn CtrlVal() verwendet wird, um dem Messgerät einen Wert (Gleitkomma oder Ganzzahl) zuzuweisen, wird der analoge Teil des Messgeräts auf eine Länge gezogen, die proportional zum Bereich zwischen „min“ und „max“ ist.
Gleichzeitig wird der digitale Wert in der Mitte des Messgeräts mit den aktuellen Schriftarteinstellungen (eingestellt mit dem FONT-Befehl) gezeichnet. 'nbrdec' gibt die Anzahl der Dezimalstellen an, die in dieser Anzeige verwendet werden sollen. Unter dem digitalen Wert werden die 'units$' angezeigt (dies kann übersprungen oder eine Zeichenfolge der Länge Null verwendet werden, wenn dies nicht erforderlich ist).
Normalerweise wird die analoge Grafik mit der in „Fcolour“ angegebenen Farbe gezeichnet, jedoch kann ein mehrfarbiges Messgerät erstellt werden, indem „c1“ bis „c4“ für die Farben und „ta“ bis „tc“ für die Schwellenwerte verwendet werden, die zur Bestimmung der Farbe verwendet werden wird sich verändern. Insbesondere ist „c1“ die Farbe, die für Werte bis zu „ta“ verwendet werden soll. 'c2' ist die Farbe, die für Werte zwischen 'ta' und 'tb' verwendet wird, 'c3' wird für Werte zwischen 'tb' und 'tc' verwendet und c4 wird für Werte über 'tc' verwendet. Nicht benötigte Farben und Schwellen können in der Liste weggelassen werden. Beispielsweise müssen für ein zweifarbiges Messgerät nur „c1“, „ta“ und „c2“ angegeben werden.
Wenn Farben und Schwellenwerte angegeben sind, wird der Hintergrund des Messgeräts mit einer matten Version der Messgerätfarbe auf dieser Ebene gezeichnet ("Geisterfarbe"), damit der Benutzer erkennen kann, wie nahe der tatsächliche Wert an den verschiedenen Schwellenwerten liegt. Auch der in der Mitte angezeigte Digitalwert ändert sich in die Farbe, die durch den aktuellen Wert festgelegt wird.

Wenn nur eine Farbe für den gesamten analogen Graphen benötigt wird, kann dies angegeben werden, indem einfach 'c1' verwendet wird und alle folgenden Parameter ausgeschaltet bleiben.



Balkenanzeige

`GUI BARGAUGE #ref, StartX, StartY, width, height, FColour, BColour, min, max, c1, ta, c2, tb, c3, tc, c4`



Dadurch wird entweder eine horizontale oder eine vertikale Balkenanzeige definiert. Das Messgerät kann eingefärbt werden, um den Signalpegel grafisch anzuzeigen (z. B. grün für OK, gelb für Warnung usw.), und viele Balkendiagramme können dicht beieinander gepackt werden, sodass mehrere Werte gleichzeitig mit einer kleinen Menge angezeigt werden können Bildschirmbereich (wie im Bild gezeigt, das aus zehn Balkenanzeigen besteht).

Wenn die Breite kleiner als die Höhe ist, wird die Balkenanzeige vertikal gezeichnet, wobei der analoge Graph von unten nach oben wächst. Andernfalls, wenn die Breite größer als die Höhe ist, wird es horizontal gezeichnet, wobei der analoge Graph von links nach rechts wächst. In beiden Fällen beziehen sich „StartX“ und „StartY“ auf die obere linke Koordinate des Balkendiagramms, während „Breite“ die horizontale Breite und „Höhe“ die vertikale Höhe ist.

Das Balkendiagramm hat keine digitale Anzeige seines Wertes, aber ansonsten sind die Parameter die gleichen wie für das kreisförmige Messgerät (oben beschrieben).

'min' und 'max' geben den Wertebereich für den Balken an und, falls angegeben, 'c1' bis 'c4' und 'ta' bis 'tc' geben die Farben und Schwellwerte für das analoge Balkenbild an. Beachten Sie, dass im Gegensatz zur Rundbalkenanzeige kein „Geisterbild“ der Farben im Hintergrund angezeigt wird.

Wenn nur eine Farbe für die gesamte Anzeige erforderlich ist, kann dies wie bei der kreisförmigen Anzeige angegeben werden, indem Sie einfach 'c1' verwenden und alle folgenden Parameter auslassens).

Area

`GUI AREA #ref, StartX, StartY, Width, Height`

Dies definiert einen unsichtbaren Bereich des Bildschirms, der berührungsempfindlich ist, und setzt TOUCH(REF) und TOUCH(LASTREF) entsprechend, wenn sie berührt oder losgelassen werden. Es kann als Grundlage zum Erstellen eines benutzerdefinierten Steuerelements verwendet werden, das vom BASIC-Programm definiert und verwaltet wird.

Interaktion mit Steuerelementen

Mit den folgenden Befehlen und Funktionen können die Eigenschaften der Bildschirmsteuerelemente geändert und ihre Werte abgerufen werden.

- = CTRLVAL(#ref)
  Dies ist eine Funktion, die den aktuellen Wert eines Steuerelements zurückgibt. Bei Steuerelementen wie Kontrollkästchen oder Schaltern ist dies die Zahl Eins (wahr), die angibt, dass das Steuerelement vom Benutzer ausgewählt wurde, oder Null (falsch), wenn dies nicht der Fall ist. Bei Steuerelementen, die eine Zahl enthalten (z. B. eine SPINBOX), ist der Wert die Zahl (normalerweise eine Fließkommazahl). Bei Steuerelementen, die eine Zeichenfolge enthalten (z. B. TEXTBOX), ist der Wert eine Zeichenfolge. Beispielsweise:
  ```
  PRINT "The number in the spin box is: " CTRLVAL(#10)
  ```

- CTRLVAL(#ref) =
  Dieser Befehl setzt den Wert eines Steuerelements. Für Ein-/Aus-Steuerelemente wie Kontrollkästchen überschreibt es jede Berührungseingabe und kann zum Drücken/Loslassen von Schaltern, Aktivieren/Deaktivieren von Kontrollkästchen usw. verwendet werden. Ein Wert von Null ist deaktiviert oder deaktiviert, und ein Wert ungleich Null schaltet das Steuerelement ein. Bei einer LED bewirkt dies, dass die LED leuchtet oder ausgeschaltet wird. Es kann auch verwendet werden, um den Anfangswert von Drehfeldern, Textfeldern usw. festzulegen. Zum Beispiel:
  ```
  CTRLVAL(#10) = 12.4
  ```

- GUI FCOLOUR colour, #ref1 [, #ref2, #ref3, etc]
  Dadurch wird die Vordergrundfarbe der angegebenen Steuerelemente in „Farbe“ geändert. Dies ist besonders praktisch für eine LED, die ihre Farbe ändern kann.

- GUI BCOLOUR colour, #ref1 [, #ref2, #ref3, etc]
  Dadurch wird die Hintergrundfarbe der angegebenen Steuerelemente in „Farbe“ geändert.

- = TOUCH(REF)
  Dies ist eine Funktion, die die Referenznummer des aktuell berührten Steuerelements zurückgibt. Ohne Berührung wird Null ausgegeben.

- = TOUCH(LASTREF)
  Dies ist eine Funktion, die die Referenznummer des zuletzt berührten Steuerelements zurückgibt.

- GUI DISABLE #ref1 [, #ref2, #ref3, etc]
  Dadurch werden die Steuerelemente in der Liste deaktiviert. Deaktivierte Steuerelemente reagieren nicht auf Berührungen und werden abgeblendet angezeigt. Das Schlüsselwort ALL kann als Argument verwendet werden und deaktiviert alle Steuerelemente auf der aktuell angezeigten Seite. Beispielsweise:
  `GUI DISABLE ALL`

- GUI ENABLE #ref1 [, #ref2, #ref3, etc]
  Dadurch werden die Auswirkungen von GUI DISABLE rückgängig gemacht und die Steuerelemente in der Liste wieder in den normalen Betrieb versetzt. Das Schlüsselwort ALL kann als Argument für alle Steuerelemente auf der aktuell angezeigten Seite verwendet werden.

- GUI HIDE #ref1 [, #ref2, #ref3, etc]
  Dadurch werden die Steuerelemente in der Liste ausgeblendet. Ausgeblendete Bedienelemente reagieren nicht auf Berührungen und werden auf dem Bildschirm durch die aktuelle Hintergrundfarbe ersetzt. Als Argument kann das Schlüsselwort ALL verwendet werden.

- GUI SHOW #ref1 [, #ref2, #ref3, etc]
  Dadurch werden die Auswirkungen von GUI HIDE rückgängig gemacht und die Steuerelemente in der Liste wieder sichtbar und normal bedienbar. Das Schlüsselwort ALL kann als Argument für alle Steuerelemente verwendet werden.

- GUI DELETE #ref1 [, #ref2, #ref3, etc]
  Dadurch werden die Steuerelemente in der Liste gelöscht. Dazu gehört das Entfernen des Bildes des Steuerelements vom Bildschirm unter Verwendung der aktuellen Hintergrundfarbe und das Freigeben des vom Steuerelement verwendeten Speichers. Das Schlüsselwort ALL kann als Argument verwendet werden und bewirkt, dass alle Steuerelemente gelöscht werden.

MsgBox()

Die Funktion MsgBox() zeigt ein Meldungsfeld auf dem Bildschirm an und wartet auf Benutzereingaben. Während das Meldungsfeld angezeigt wird, werden alle Steuerelemente deaktiviert, sodass das Meldungsfeld den vollständigen Fokus hat.

Syntax:

```
r = MsgBox(message$, button1$ [, button2$ [, button3$ [, button4$]]])
```

Alle Argumente sind Zeichenfolgen. 'message$' ist die anzuzeigende Nachricht. Dieser kann ein oder mehrere Tilde-Zeichen (~) enthalten, die einen Zeilenumbruch anzeigen. Innerhalb der Box können bis zu 10 Zeilen angezeigt werden. 'button1$' ist die Beschriftung für die erste Schaltfläche, 'button2$' ist die Beschriftung für die zweite Schaltfläche usw. Mindestens eine Schaltfläche muss angegeben werden und vier sind das Maximum. Alle Schaltflächen, die nicht in der Argumentliste enthalten sind, werden nicht angezeigt.

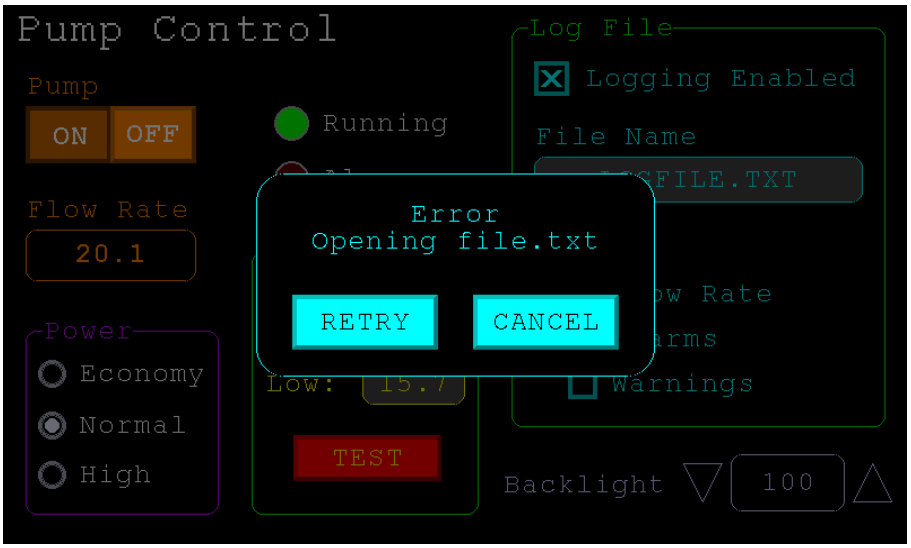
Die verwendete Schriftart ist die Standardschriftart, die mit dem Befehl FONT festgelegt wurde, und die verwendeten Farben sind die Standardeinstellungen, die mit dem Befehl COLOR festgelegt wurden. Die Größe des Felds wird automatisch angepasst, wobei die Abmessungen der Standardschriftart, die Anzahl der anzuzeigenden Zeilen und die Anzahl der angegebenen Schaltflächen berücksichtigt werden.

Wenn der Benutzer eine Schaltfläche berührt, löscht sich das Meldungsfeld selbst, stellt die Anzeige wieder her (z. B. aktiviert alle Steuerelemente erneut) und gibt die Nummer der Schaltfläche zurück, die berührt wurde (die erste Schaltfläche gibt 1 zurück, die zweite 2 usw.). Beachten Sie, dass im Gegensatz zu allen anderen GUI-Steuerelementen das BASIC-Programm aufhört zu laufen, während das Meldungsfeld angezeigt wird, Unterbrechungen jedoch berücksichtigt und bearbeitet werden.

Um die Verwendung einer Meldungsbox zu veranschaulichen, wird das folgende Programmfragment versuchen, eine Datei zu öffnen, und wenn ein Fehler auftritt, zeigt das Programm eine Fehlermeldung unter Verwendung der Funktion MsgBox() an. Die Nachricht hat zwei Zeilen und das Feld hat zwei Schaltflächen für Wiederholen und Abbrechen.

```
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
```

Wenn die Datei "file.txt" nicht existiert dann erscheint:



Fortgeschrittene Grafik-Programmierung

Beim Programmieren mit den erweiterten GUI-Befehlen sind eine Reihe von Hinweisen und Techniken zu beachten, die die Entwicklung und Wartung Ihres Programms erleichtern.

Der Benutzer sollte die Kontrolle haben

Herkömmliche zeichenbasierte Programme steuern normalerweise die Interaktion mit dem Benutzer. Beispielsweise kann das Programm ein Menü anzeigen und den Benutzer auffordern eine Aktion auszuwählen. Wenn der Benutzer eine ungültige Option auswählt, würde das Programm eine Fehlermeldung anzeigen und die Menüoptionen erneut anzeigen.

Grafikbasierte Programme, wie sie unter Verwendung der erweiterten GUI-Befehle erstellt wurden, sind jedoch anders. Normalerweise beginnt das Programm einfach zu laufen und tut, was es normalerweise tut (z. B. Temperatur, Geschwindigkeit usw. steuern), und es ist die Aufgabe des Benutzers, Parameter auszuwählen und zu ändern, ohne dazu aufgefordert zu werden. Dies ist eine andere Art des Programmierens und es ist für den traditionellen Programmierer oft schwierig, sich an diese andere Technik zu gewöhnen.

Als Beispiel sei ein Programm betrachtet, das eine Schneidevorrichtung steuern soll. Das traditionelle Programm würde den Benutzer nach der Geschwindigkeit und Schneidzeit fragen. Wenn beide eingegeben wurden, fordert das Programm zum Starten des Schneidzyklus auf. Ein grafikbasiertes Programm würde jedoch zwei Zahlenfelder anzeigen, in die der Benutzer die Geschwindigkeit und die Zeit zusammen mit einer Starttaste eingeben könnte. Die Nummernfelder könnten mit Standardwerten gefüllt werden und die Run-Schaltfläche würde deaktiviert, wenn der Benutzer eine ungültige Geschwindigkeit oder Zeit eingibt. Wenn die Run-Taste berührt wird, beginnt der Schneidezyklus.

Ein gutes Beispiel für diese Art von grafischer Benutzeroberfläche ist das Dialogfeld, das auf einem Windows/IOS/Android-Computer zum Einstellen von Uhrzeit und Datum verwendet wird. Es zeigt eine Reihe von Feldern an, in denen der Benutzer das Datum/die Uhrzeit eingeben kann, zusammen mit einer OK-Schaltfläche, die das Programm anweist, die eingegebenen Daten zu akzeptieren. Der Benutzer wird zu keinem Zeitpunkt gezwungen, eine Auswahl aus einem Menü zu treffen. Außerdem wird die aktuelle Uhrzeit/das aktuelle Datum bereits in den Eingabefeldern angezeigt, sodass der Benutzer sie als Standard akzeptieren kann, wenn er dies möchte.

Wenn Sie etwas Inspiration brauchen, wie Ihr grafisches Programm aussehen und sich anfühlen sollte, sehen Sie sich das nächstgelegene GUI-basierte Betriebssystem an, um zu sehen, wie es funktioniert.

Programmstruktur

Typischerweise würde ein Programm mit der Definition der Steuerelemente beginnen (die MMBasic auf dem Bildschirm zeichnen wird), dann würde es die Standardeinstellungen festlegen und schließlich in eine Endlosschleife fallen, in der es die Aufgabe erledigen würde, für die es entworfen wurde. Nehmen wir zum Beispiel den Fall einer einfachen Steuerung für einen Motor, bei der der Benutzer die Geschwindigkeit auswählen und den Motor zum Laufen bringen könnte, indem er eine Schaltfläche auf dem Bildschirm drückt.

Um diese Funktion zu implementieren, würde das Programm in etwa so aussehen:

```
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                                           ' this runs in a loop forever
  IF CtrlVal(#3)<10 OR CtrlVal(#3)>200 THEN  ' check the speed setting
    GUI DISABLE #3                           ' disable RUN if it is invalid
  ELSE                                       ' otherwise
    GUI ENABLE #3                            ' enable the RUN button
  ENDIF

  IF CtrlVal(#3) = 1 THEN                    ' if the button is pressed
    SetMotorSpeed CtrlVal(#2)                ' make the motor run
  ELSE                                       ' otherwise the button is up
    SetMotorSpeed 0                          ' therefore set motor speed to zero
  ENDIF
LOOP
```

Beachten Sie, dass der Benutzer zu nichts aufgefordert wird; Das Programm befindet sich nur in einer Schleife und reagiert auf die Änderungen, die der Benutzer an den Steuerelementen vorgenommen hat (dh der Benutzer hat die Kontrolle).

Ungültige Steuerelemente deaktivieren

Wie im obigen Beispiel verhindert das Deaktivieren eines Steuerelements, dass ein Benutzer es verwendet und MMBasic zeichnet es in einer matten Farbe neu um anzuzeigen dass es nicht verfügbar ist. Dies entspricht einer Fehlermeldung in einem herkömmlichen textbasierten Programm und ist benutzerfreundlicher als das Aufklappen einer Meldungsbox, die geschlossen werden muss bevor irgendetwas anderes getan werden kann.

Es gibt viele Fälle, in denen ein Steuerelement ungültig sein kann, z. B. wenn eine Eingabe nicht bereit ist oder einfach wenn eine Option oder Aktion nicht zutrifft. Später, wenn das Steuerelement gültig wird, können Sie es mit dem Befehl GUI ENABLE wieder verwenden. Ein weiteres Beispiel ist, wenn eine GUI-NUMBERBOX-Tastatur angezeigt wird deaktiviert MMBasic automatisch alle anderen Steuerelemente auf dem Bildschirm, so dass es für den Benutzer offensichtlich ist, wo seine Eingabe erforderlich ist.

Durch das Deaktivieren eines Steuerelements bleibt es weiterhin auf dem Bildschirm, sodass der Benutzer weiß, dass es vorhanden ist, aber es wird gedimmt und reagiert nicht auf Berührungen. Nicht auf Berührung zu reagieren bedeutet auch, dass der Benutzer es nicht ändern kann und kein Interrupt erzeugt wird, wenn es berührt wird. Dies ist praktisch für Sie als Programmierer, da Sie nicht prüfen müssen, ob das Steuerelement gültig ist, bevor Sie darauf reagieren.

Konstanten verwenden für Kontroll-Referenznummern

Die erweiterten Steuerelemente verwenden eine Referenznummer, um das Steuerelement zu identifizieren. Um das Lesen und Warten Ihres Programms zu erleichtern sollten Sie diese Nummern als Konstanten mit leicht erkennbaren Namen definieren.

Beispielsweise ist im folgenden Programmfragment MAIN_SWITCH als Konstante definiert, und diese Konstante wird überall dort verwendet, wo die Referenznummer für diese Steuerung erforderlich ist:

```
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
```

Es ist viel einfacher, sich zu merken was MAIN_SWITCH tut als sich daran zu erinnern, auf welche Steuerung sich die Nummer 5 bezieht. Außerdem ist es bei vielen Steuerelementen viel einfacher, die Steuerelemente neu zu nummerieren wenn alle ihre Nummern an einer Stelle am Anfang des Programms definiert werden.

Die Referenznummer muss eine Zahl zwischen 1 und dem mit dem Befehl OPTION CONTROL eingestellten Wert sein. Wenn Sie diese Zahl erhöhen, wird mehr RAM verbraucht, und wenn Sie sie verringern, wird etwas RAM wiederhergestellt.

Das Hauptprogramm läuft noch

Es ist wichtig zu wissen, dass Ihr BASIC-Hauptprogramm noch läuft, während der Benutzer mit den GUI-Steuerelementen interagiert. Beispielsweise wird es weiter ausgeführt, selbst wenn ein Benutzer einen Schalter auf dem Bildschirm gedrückt hält, und es wird weiter ausgeführt, während die virtuelle Tastatur als Ergebnis der Berührung eines TEXTBOX-Steuerelements angezeigt wird.

Aus diesem Grund sollte Ihr Hauptprogramm berührungsempfindliche Bildschirmsteuerungen nicht willkürlich aktualisieren, da sie das Bildschirmbild ändern könnten, während der Benutzer sie verwendet (mit undefinierten Ergebnissen). Wenn ein BASIC-Programm mit GUI-Steuerelementen startet, initialisiert es normalerweise Steuerelemente wie SPINBOX, NUMBERBOX und TEXTBOX auf einen Anfangswert, aber von da an sollte das Hauptprogramm nur den Wert dieser Steuerelemente lesen – es liegt in der Verantwortung des Benutzers, Änderungen vorzunehmen diese, nicht Ihr Programm.

Wenn Sie jedoch den Wert eines solchen Bildschirmsteuerelements ändern möchten, benötigen Sie einen Mechanismus, um zu verhindern, dass sowohl das Programm als auch der Benutzer gleichzeitig eine Änderung vornehmen. Ein Verfahren besteht darin, innerhalb des Key-down-Interrupts ein Flag zu setzen, um anzuzeigen, dass die Steuerung während dieser Zeit nicht aktualisiert werden sollte. Dieses Flag kann dann in der Taste-Auf-Unterbrechung gelöscht werden, damit das Hauptprogramm die Aktualisierung der Steuerung wieder aufnehmen kann.

Beachten Sie, dass diese Diskussion nur für Steuerelemente gilt, die auf Berührung reagieren. Bedienelemente wie CAPTION und LED können jederzeit vom Hauptprogramm geändert werden und werden es auch oft.

Verwenden Sie Interrupts und SELECT CASE-Anweisungen

Alles, was auf einem Bildschirm mit den erweiterten Steuerelementen passiert, wird durch einen Interrupt signalisiert, entweder Touchdown oder Touchup. Wenn Sie also sofort etwas tun möchten, wenn ein Steuerelement geändert wird, sollten Sie dies in einem Interrupt tun. Meistens interessiert Sie, wann die Berührung (oder der Stift) unten ist, aber in einigen Fällen möchten Sie vielleicht auch wissen, wann sie losgelassen wird.

Da der Interrupt ausgelöst wird, wenn der Stift ein Steuerelement oder einen Teil des Bildschirms berührt, müssen Sie herausfinden, welches Steuerelement berührt wurde. Dies wird am besten mit der Funktion TOUCH(REF) und der Anweisung SELECT CASE durchgeführt. Im folgenden Fragment wird beispielsweise die Subroutine PenDown aufgerufen, wenn eine Berührung erfolgt, und die Funktion TOUCH(REF) gibt die Referenznummer des berührten Steuerelements zurück. Mit dem SELECT CASE wird die Alarm-LED ein- oder ausgeschaltet, je nachdem, welche Taste berührt wird. Die Aktion kann eine beliebige Anzahl von Dingen sein, wie das Anheben eines I/O-Pins, um eine physische Sirene einzuschalten, oder das Drucken einer Nachricht auf der Konsole.

```
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
```

Das SELECT CASE kann auch auf andere Steuerelemente testen und alle Aktionen ausführen, die für sie in ihrem eigenen Abschnitt der CASE-Anweisung erforderlich sind.

Der wichtige Punkt ist, dass die Wartung der Steuerung (z. B. auf die Tasten reagieren und die Alarm-LED aus- oder einschalten) automatisch erfolgt, ohne dass das Hauptprogramm beteiligt ist – es kann weiterhin etwas Nützliches tun, wie z. B. die Berechnung einer Steuerungsantwort usw.

Touch Up Interrupt

In den meisten Fällen können Sie alle Benutzereingaben im Touchdown-Interrupt verarbeiten. Aber es gibt Ausnahmen und ein typisches Beispiel ist, wenn Sie die Eigenschaften des Steuerelements ändern müssen, das berührt wird. Zum Beispiel, wenn Sie die Vordergrundfarbe einer Schaltfläche von Weiß auf Rot ändern möchten, wenn sie unten ist. Wenn es in den oberen Zustand zurückkehrt, sollte die Farbe zu Weiß zurückkehren.

Das Einstellen der Farbe beim Aufsetzen ist einfach. Definieren Sie einfach einen Touchdown-Interrupt und ändern Sie die Farbe im Interrupt, wenn dieses Steuerelement berührt wird. Um die Farbe jedoch wieder auf Weiß zurückzusetzen, müssen Sie erkennen, wann die Berührung von der Steuerung entfernt wurde (d. h. nachbessern). Dies kann mit einem Touchup-Interrupt erfolgen.

Um einen Touchup-Interrupt anzugeben, fügen Sie den Namen der Subroutine für diesen Interrupt am Ende des GUI-Befehls INTERRUPT hinzu. Beispielsweise:

```
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
```

Halten Sie die Interrupts sehr kurz

Da ein Berührungs-Interrupt eine Anfrage des Benutzers anzeigt, ist es verlockend, einige umfangreiche Programmierungen innerhalb eines Interrupts vorzunehmen. Wenn zum Beispiel die Berührung anzeigt, dass der Benutzer eine Nachricht an einen anderen Controller senden möchte, klingt es logisch, diesen ganzen Code in den Interrupt zu stecken. Dies ist jedoch keine gute Idee, da MMBasic nichts anderes tun kann während Ihr Programm den Interrupt verarbeitet und das Senden einer Nachricht viele Millisekunden dauern kann.

Stattdessen sollte Ihr Programm eine globale Variable aktualisieren, um anzuzeigen, was angefordert wird, und die eigentliche Ausführung dem Hauptprogramm überlassen. Wenn der Benutzer beispielsweise die Schaltfläche "Nachricht senden" berührt hat könnte Ihr Programm einfach eine globale Variable auf wahr setzen. Dann kann das Hauptprogramm diese Variable überwachen und wenn sie sich ändert die Logik und die Kommunikation ausführen die erforderlich sind um die Anforderung zu erfüllen.

Denken Sie an das Gebot „Du sollst nicht in einer Unterbrechung herumhängen”.

Mehrere Bildschirme

Ihr Programm benötigt möglicherweise eine Reihe von Bildschirmen mit unterschiedlichen Steuerelementen auf jedem Bildschirm. Dies könnte dadurch implementiert werden, dass die alten Steuerelemente gelöscht und neue erstellt werden, wenn der Bildschirm umgeschaltet wird. Aber eine andere Möglichkeit dies zu tun ist die Verwendung der GUI-Befehle SETUP und PAGE. Mit diesen können Sie die Steuerelemente auf Seiten organisieren und mit einem einfachen Befehl zwischen den Seiten wechseln. Alle Steuerelemente auf der alten Seite werden automatisch ausgeblendet und Steuerelemente auf der neuen Seite werden automatisch angezeigt.

Um einer Seite Steuerelemente zuzuweisen, verwenden Sie den GUI SETUP nn-Befehl, wobei nn sich auf die Seite im Bereich von 1 bis 32 bezieht. Wenn Sie diesen Befehl verwendet haben, werden alle neu erstellten Steuerelemente dieser Seite zugewiesen. Sie können GUI SETUP so oft verwenden, wie Sie möchten. Beispielsweise werden im Programmfragment unten die ersten beiden Bedienelemente Seite 1 zugewiesen, die zweite Seite 2 usw.

```
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
```

Standardmäßig werden nur die auf Seite 1 eingerichteten Steuerelemente angezeigt und die anderen ausgeblendet.

Um den Bildschirm auf Seite 3 umzuschalten, brauchen Sie nur den Befehl PAGE 3 zu verwenden. Dadurch werden die Bedienelemente Nr. 1 und Nr. 2 automatisch ausgeblendet und die Bedienelemente Nr. 5 und Nr. 6 angezeigt. In ähnlicher Weise wird SEITE 2 alle außer #3 und #4 ausblenden, die angezeigt werden.

Sie können mehrere Seiten angeben, die gleichzeitig angezeigt werden sollen, z. B. zeigt SEITE 1,3 sowohl die Seiten 1 als auch 3 an, während Seite 2 ausgeblendet wird. Dies kann nützlich sein, wenn Sie eine Reihe von Steuerelementen haben, die ständig sichtbar sein müssen. Zum Beispiel lassen PAGE 1,2 und PAGE 1,3 die Bedienelemente auf Seite 1 sichtbar, während die anderen ein- und ausgeschaltet werden.

Es ist legal, dass ein Programm Steuerelemente auf anderen Seiten ändert, auch wenn sie zu diesem Zeitpunkt nicht angezeigt werden. Dazu gehört das Ändern des Werts und der Farben sowie das Deaktivieren oder Ausblenden dieser. Wenn die Anzeige auf ihre Seite umgeschaltet wird, werden die Steuerelemente mit ihren neuen Attributen angezeigt.

Es ist möglich, die PAGE-Befehle in den Touchdown-Interrupt zu platzieren, so dass das Drücken einer bestimmten Steuerung oder eines Teils des Bildschirms auf eine andere Seite umschaltet.

Beachten Sie, dass wenn ALL für die Liste der Steuerelemente in Befehlen wie GUI ENABLE ALL verwendet wird, sich dies nur auf die Steuerelemente auf den Seiten bezieht, die derzeit zur Anzeige ausgewählt sind. Steuerelemente auf anderen Seiten sind davon nicht betroffen.

Alle Programme starten mit dem Äquivalent der gültigen Befehle GUI SETUP 1 und PAGE 1. Das heißt, wenn die GUI-Befehle SETUP und PAGE nicht verwendet werden, läuft das Programm erwartungsgemäß mit allen angezeigten Steuerelementen.

Eine typische Verwendung des PAGE-Befehls ist unten gezeigt. Zwei Schaltflächen (die immer angezeigt werden) ermöglichen dem Benutzer, zwischen der ersten Seite und der zweiten Seite zu wählen. Die Umschaltung erfolgt im Touchdown-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 Page 1, 2
  If Touch(REF) = 11 Then Page 1, 3
End Sub
```

Mehrere Interrupts

Bei vielen Bildschirmseiten konnte das Interrupt-Unterprogramm lang und kompliziert werden. Um dies zu umgehen ist es möglich mehrere Interrupt-Subroutinen zu haben und zwischen ihnen nach Belieben dynamisch umzuschalten (normalerweise nach dem Seitenwechsel). Dies erfolgt durch Neudefinition der aktuellen Interrupt-Routinen unter Verwendung des GUI-Befehls INTERRUPT.

Beispielsweise verwendet dieses Programmfragment unterschiedliche Interrupt-Routinen für die Seiten 4 und 5 und sie werden unmittelbar nach dem Umschalten der Seiten angegeben.

```
PAGE 4
GUI INTERRUPT P4keydown, P4keyup
...
PAGE 5
GUI INTERRUPT P5keydown, P5keyup
...
```

Grundlegende Zeichenbefehle

Es gibt zwei Arten von Objekten, die sich auf dem Bildschirm befinden können. Dies sind die GUI-Steuerelemente und die grundlegenden Zeichenobjekte (PIXEL, LINE, TEXT usw.). Das Mischen der beiden auf dem Bildschirm ist keine gute Idee, da MMBasic die Position der grundlegenden Zeichenobjekte nicht verfolgt und sie mit den GUI-Steuerelementen kollidieren können.

Daher sollten Sie entweder die GUI-Steuerelemente oder die grundlegenden Zeichenobjekte verwenden, es sei denn, Sie sind bereit, zusätzliche Programmierungen vorzunehmen – aber Sie sollten nicht beide verwenden. Verwenden Sie also beispielsweise nicht TEXT, sondern GUI CAPTION. Wenn Sie nur GUI-Steuerelemente verwenden, verwaltet MMBasic den Bildschirm für Sie, einschließlich des Löschens und Neuzeichnens nach Bedarf, z. B. wenn eine virtuelle Tastatur angezeigt wird.

Beachten Sie, dass der CLS-Befehl (der zum Löschen des Bildschirms verwendet wird) automatisch alle GUI-Steuerelemente auf dem Bildschirm auf versteckt setzt (d. h. es führt ein GUI HIDE ALL aus, bevor der Bildschirm gelöscht wird).

Das Hauptproblem beim Mischen grundlegender Grafik- und GUI-Steuerelemente tritt bei den Steuerelementen Textfeld, Formatiertes Feld und Zahlenfeld auf, die eine virtuelle Tastatur anzeigen. Dadurch können grundlegende Grafiken gelöscht werden, und MMBasic weiß nicht, wie sie wiederhergestellt werden können, wenn die Tastatur entfernt wird. Wenn Sie grundlegende Grafiken mit GUI-Steuerelementen mischen möchten, sollten Sie:

• Unterbrechen Sie den Touchdown-Interrupt für die Steuerelemente Textfeld, Formatiertes Feld und Zahlenfeld, da dies anzeigt, dass eine virtuelle Tastatur angezeigt wird, und Ihnen die Möglichkeit gibt, Ihre Nicht-GUI-Grundgrafiken in Erwartung dieses Ereignisses neu zu zeichnen (z Zeichnen Sie sie beispielsweise in einem abgeblendeten Zustand, damit sie so erscheinen, als wären sie deaktiviert).

• Fangen Sie den Touchup-Interrupt für die gleichen Steuerelemente ab, der anzeigt, dass die virtuelle Tastatur entfernt wurde, und Sie könnten dann alle Nicht-GUI-Grafiken in ihrem ursprünglichen Zustand neu zeichnen.

Überlappende Steuerelemente

Bedienelemente können so definiert werden, dass sie sich auf dem Display überlappen, dies tritt meistens bei GUI AREA auf, wo Sie beispielsweise eine Berührung erfassen möchten, die (sagen wir) für einen GUI BUTTON gedacht war. Dadurch können Sie anstelle der von MMBasic bereitgestellten eine eigene Animation für die Schaltfläche erstellen. In diesem Fall sollte das Bedienelement, das auf die Berührung reagieren soll (d. h. der GUI-BEREICH), eine niedrigere Referenznummer (d. h. #ref) haben als das Bedienelement, das es abdeckt (d. h. der GUI-BUTTON). Dies liegt daran, dass MMBasic bei Berührung des Bildschirms die aktuelle Liste der aktiven Kontrollen prüft, beginnend mit Kontrollnummer 1 und aufwärts arbeitend. Wenn eine Übereinstimmung gefunden wird, ergreift MMBasic die entsprechende Aktion und beendet die Suche. Dies führt dazu, dass das Steuerelement mit niedrigerer Nummer effektiv ein Steuerelement mit höherer Nummer ausblendet, das denselben Bildschirmbereich abdeckt.

MMBasic Eigenschaften

Namensgebung

Befehlsnamen, Funktionsnamen, Beschriftungen, Variablennamen usw. unterscheiden nicht zwischen Groß- und Kleinschreibung, sodass „Run“ und „RUN“ gleichwertig sind und „dOO“ und „Doo“ auf dieselbe Variable verweisen.

Der Typ einer Variablen kann im DIM-Befehl oder durch Hinzufügen eines Suffixes am Ende des Variablennamens angegeben werden. Zum Beispiel ist das Suffix für eine ganze Zahl '%', wenn also eine Variable namens nbr% automatisch erstellt wird, ist es eine ganze Zahl. Es gibt drei Arten von Variablen:

1. Fließkomma. Diese können eine Zahl mit Dezimalpunkt und Bruch (z. B. 45,386) und auch sehr große Zahlen speichern. Sie verlieren jedoch an Genauigkeit, wenn mehr als 14 signifikante Ziffern gespeichert oder manipuliert werden. Das Suffix ist '!' und Fließkomma ist der Standardwert, wenn eine Variable ohne Suffix erstellt wird

2. 64-Bit-Ganzzahl. Diese können Zahlen mit bis zu 19 Dezimalstellen speichern ohne an Genauigkeit zu verlieren, aber sie können keine Brüche speichern (d. h. den Teil nach dem Dezimalpunkt). Das Suffix für eine ganze Zahl ist '%'

3. Saiten. Diese speichern eine Zeichenfolge (z. B. "Tom"). Das Suffix für eine Zeichenfolge ist das Symbol „$“ (z. B. name$, s$ usw.). Zeichenfolgen können bis zu 255 Zeichen lang sein.

Variablennamen und -bezeichnungen können mit einem Buchstaben oder Unterstrich beginnen und beliebige Buchstaben oder Ziffern, den Punkt (.) und den Unterstrich (_) enthalten. Sie dürfen bis zu 32 Zeichen lang sein. Ein Variablenname oder ein Label darf nicht dasselbe sein wie ein Befehl oder eine Funktion oder eines der folgenden Schlüsselwörter: THEN, ELSE, TO, STEP, FOR, WHILE, UNTIL, MOD, NOT, AND, OR, XOR, AS. Z.B. step = 5 ist unzulässig.

Konstanten

Numerische Konstanten können mit einer numerischen Ziffer (0-9) für eine Dezimalkonstante, &H für eine Hexadezimalkonstante, &O für eine Oktalkonstante oder &B für eine Binärkonstante beginnen. Zum Beispiel ist &B1000 dasselbe wie die Dezimalkonstante 8. Konstanten, die mit &H, &O oder &B beginnen, werden immer als 64-Bit-Ganzzahlkonstanten behandelt.

Dezimalkonstanten kann ein Minus (-) oder Plus (+) vorangestellt werden und mit einem „E“ abgeschlossen werden, gefolgt von einer Exponentenzahl, um die Exponentialschreibweise zu bezeichnen. Zum Beispiel ist 1.6E+4 dasselbe wie 16000.

Wenn die Dezimalkonstante einen Dezimalpunkt oder einen Exponenten enthält, wird sie als Fließkommakonstante behandelt; andernfalls wird sie als 64-Bit-Ganzzahlkonstante behandelt.

Zeichenfolgenkonstanten werden von doppelten Anführungszeichen (") umgeben, z. B. "Hello World".

Implementierungsmerkmale

Die maximale Programmgröße (als Klartext) beträgt 80 KB. Beachten Sie, dass MMBasic das Programm tokenisiert, wenn es im Flash gespeichert wird, sodass die endgültige Größe im Flash von der reinen Textgröße abweichen kann.

Die maximale Länge einer Befehlszeile beträgt 255 Zeichen.

Die maximale Länge eines Variablennamens oder eines Labels beträgt 32 Zeichen.

Die maximale Anzahl von Dimensionen für ein Array beträgt 5.

Die maximale Anzahl von Argumenten für Befehle, die eine variable Anzahl von Argumenten akzeptieren, beträgt 50.

Die maximale Anzahl verschachtelter FOR…NEXT-Schleifen beträgt 20.

Die maximale Anzahl verschachtelter DO…LOOP-Befehle beträgt 20.

Die maximale Anzahl verschachtelter GOSUBs, Subroutinen und Funktionen (kombiniert) beträgt 320.

Die maximale Anzahl verschachtelter mehrzeiliger IF…ELSE…ENDIF-Befehle beträgt 20.

Maximale Anzahl benutzerdefinierter Unterprogramme und Funktionen (kombiniert): 256

Maximale Anzahl konfigurierbarer Interrupt-Pins: 10

Zahlen werden als Gleitkommazahlen mit doppelter Genauigkeit oder 64-Bit-Ganzzahlen mit Vorzeichen gespeichert und verarbeitet. Der Bereich der Gleitkommazahlen ist 1.797693134862316e+308 bis 2.225073858507201e-308.

Der Bereich der manipulierbaren 64-Bit-Ganzzahlen (Ganzzahlen) beträgt ± 9223372036854775807.

Die maximale Zeichenfolgenlänge beträgt 255 Zeichen.

Die maximale Zeilennummer beträgt 65000.

Die maximale Anzahl von Hintergrundimpulsen, die durch den PULSE-Befehl gestartet werden, beträgt 5.

Die maximale Anzahl globaler Variablen und Konstanten beträgt 256

Die maximale Anzahl lokaler Variablen beträgt 256

Kompatibilität

MMBasic implementiert eine große Teilmenge von Microsofts GW-BASIC. Es gibt zahlreiche Unterschiede aufgrund physikalischer und praktischer Überlegungen, aber die meisten Standard-BASIC-Befehle und -Funktionen sind im Wesentlichen gleich. Ein Online-Handbuch für GW BASIC ist unter http://www.antonis.de/qbebooks/gwbasman/index.html verfügbar und enthält eine detailliertere Beschreibung der Befehle und Funktionen.

MMBasic implementiert auch eine Reihe moderner Programmierstrukturen, die im ANSI-Standard für Full BASIC (X3.113-1987) oder ISO/IEC 10279:1991 dokumentiert sind. Dazu gehören SUB/END SUB, die DO WHILE … LOOP, die SELECT…CASE-Anweisungen und strukturierte IF .. THEN … ELSE … ENDIF-Anweisungen.

Reservierte Variablen, Read-Only

Diese Variablen werden von MMBasic gesetzt und können vom laufenden Programm nicht geändert werden.

MM.VER	Die Versionsnummer der Firmware als Fließkommazahl im Format aa.bbcc, wobei aa die Hauptversionsnummer, bb die Nebenversionsnummer und cc die Revisionsnummer ist. Beispielsweise gibt Version 5.03.00 5.03 zurück und Version 5.03.01 gibt 5.0301 zurück.
MM.DEVICE$	Eine Zeichenfolge, die das Gerät oder die Plattform darstellt, auf der MMBasic läuft. Derzeit enthält diese Variable eines der folgenden Elemente: "Maximite" auf dem Standard-Maximite und Kompatiblen. "Colour Maximite" auf dem Color Maximite und UBW32. "Colour Maximite 2" auf der Color Maximite 2. "DuinoMite" beim Laufen auf einem der DuinoMite-Familie. "DOS", wenn es unter Windows in einer DOS-Box ausgeführt wird. "Generic PIC32" für die generische Version von MMBasic auf einem PIC32. "Micromite" auf dem PIC32MX150/250 "Micromite MkII" auf dem PIC32MX170/270 „Micromite Plus“ auf dem PIC32MX470 „Micromite Extreme“ auf der PIC32MZ-Serie „PicoMite“ auf dem Raspberry Pi Pico
MM.ERRNO MM.ERRMSG$	Wenn eine Anweisung einen Fehler verursacht hat, der ignoriert wurde, werden diese Variablen entsprechend gesetzt. MM.ERRNO ist eine Zahl, bei der ungleich Null bedeutet, dass ein Fehler aufgetreten ist, und MM.ERRMSG$ eine Zeichenfolge ist, die die Fehlermeldung darstellt, die normalerweise auf der Konsole angezeigt worden wäre. Sie werden durch RUN, ON ERROR IGNORE oder ON ERROR SKIP auf Null und einen leeren String zurückgesetzt.
MM.INFO() MM.INFO$()	Diese beiden Versionen können austauschbar verwendet werden, aber eine gute Programmierpraxis würde erfordern, dass Sie diejenige verwenden, die dem zurückgegebenen Datentyp entspricht..
MM.INFO$(AUTORUN)	Gibt die Einstellung des Befehls OPTION AUTORUN zurück
MM.INFO$(CPUSPEED)	Gibt die CPU-Geschwindigkeit als String zurück
MM.INFO$(LCDPANEL)	Gibt den Namen des konfigurierten LCD-Panels oder eine leere Zeichenfolge zurück
MM.INFO$(SDCARD)	Gibt den Status der SD-Karte zurück. Gültige Rücksendungen sind: DEAKTIVIERT, NICHT VORHANDEN, BEREIT und NICHT VERWENDET
MM.INFO(DISK SIZE)	Gibt die Kapazität der SD-Karte in Bytes zurück
MM.INFO$(FREE SPACE)	Gibt den freien Speicherplatz auf der SD-Karte zurück.

MM.INFO$(FILESIZE file$)	Gibt die Größe von file$ in Bytes oder -1 zurück, falls nicht gefunden, -2 falls ein Verzeichnis..
MM.INFO(ID)	Gibt die eindeutige ID der Pico-Leiterplatte zurück
MM.INFO$(MODIFIED file$)	Gibt Datum/Uhrzeit der Änderung von Datei$ zurück, leere Zeichenfolge, falls nicht gefunden
MM.INFO(FCOLOUR)	Gibt die aktuelle Vordergrundfarbe zurück
MM.INFO(BCOLOUR)	Gibt die aktuelle Hintergrundfarbe zurück
MM.INFO(FONT)	Gibt die Nummer der aktuell aktiven Schriftart zurück
MM.INFO(FONT ADDRESS n)	Gibt die Adresse des Speicherplatzes mit der Adresse von FONT n zurück
MM.INFO(FONT POINTER n)	Gibt einen POINTER auf den Anfang von FONT n im Speicher zurück
MM.INFO(FONTHEIGHT) MM.INFO(FONTWIDTH)	Ganzzahlen, die die Höhe und Breite der aktuellen Schriftart (in Pixel) darstellen.
MM.INFO$(FLASH)	Gibt ggf. an, aus welchem Flash-Slot das Programm geladen wurde
MM.INFO(HPOS) MM.INFO(VPOS)	Die aktuelle horizontale und vertikale Position (in Pixel) nach dem letzten Grafik- oder Druckbefehl.
MM.INFO(OPTION option)	Gibt den aktuellen Wert einer Reihe von Optionen zurück, die sich auf die Funktionsweise eines Programms auswirken. Option: AUTORUN, BASE, BREAK, DEFAULT oder EXPLICIT
MM.INFO$(PIN pinno)	Gibt den Status des I/O-Pins „pinno“ zurück. Gültige Rücksendungen sind: INVALID, RESERVED, IN USE, und UNUSED
MM.INFO$(PINNO GPnn)	Gibt die physische PIN-Nummer für eine gegebene GP-Nummer zurück.
MM.INFO$(RESET)	Gibt die Ursache eines Firmware-Neustarts zurück. Der zurückgegebene Wert ist einer von „Switch“ dh Drücken des Reset-Schalters, „Power-On“, „Software“ und „Watchdog“ NB, letzteres ist der H/W-Watchdog und hat nichts mit der MMbasic-Version zu tun, die verursacht ein Software-Reset.
MM.INFO$(TOUCH)	Gibt den Status des Touch-Controllers zurück. Gültige Rücksendungen sind: “Disabled”, “Not calibrated”, and “Ready”.
MM.HRES MM.VRES	Ganzzahlen, die die horizontale und vertikale Auflösung des LCD-Anzeigefelds (falls konfiguriert) in Pixel darstellen.
MM.FONTHEIGHT MM.FONTWIDTH	Ganzzahlen, die die Höhe und Breite der aktuellen Schriftart (in Pixeln) darstellen und aus Kompatibilitätsgründen beibehalten werden. Hinweis: Diese werden von MMBasic automatisch in MM.INFO(FONTHEIGHT) und MM.INFO(FONTWIDTH) konvertiert
MM.ONEWIRE	Nach einer 1-Wire-Reset-Funktion wird diese Integer-Variable gesetzt, um das Ergebnis der Operation anzuzeigen: 0 = Gerät nicht gefunden, 1 = Gerät gefunden

MM.I2C	Nach einem I2C-Schreib- oder Lesebefehl wird diese Integer-Variable gesetzt, um das Ergebnis der Operation wie folgt anzuzeigen: 0 = Der Befehl wurde ohne Fehler ausgeführt. 1 = Eine NACK-Antwort empfangen 2 = Zeitüberschreitung des Befehls
MM.WATCHDOG	Eine Ganzzahl, die wahr ist, wenn MMBasic als Ergebnis eines Watchdog-Timeouts neu gestartet wurde (siehe Befehl WATCHDOG). False, wenn MMBasic normal gestartet wurde.

Optionen

Diese Tabelle listet die verschiedenen Optionsbefehle auf, die verwendet werden können, um MMBasic zu konfigurieren und seine Arbeitsweise zu ändern. Als dauerhaft markierte Optionen werden im nichtflüchtigen Speicher gespeichert und automatisch wiederhergestellt, wenn PicoMite neu gestartet wird. Optionen, die nicht dauerhaft sind, werden beim Start zurückgesetzt.

Viele OPTION-Befehle erzwingen einen Neustart des PicoMite, wodurch die USB-Konsolenschnittstelle zurückgesetzt wird. Das gespeicherte Programm geht nicht verloren, da die Firmware beim Neustart automatisch eine Sicherungskopie wiederherstellt.

	Dauerhaft?	
OPTION AUDIO PWMnApin, PWMnBpin	✓	Konfiguriert einen der PWM-Kanäle als Audioausgang. Beispiel: OPTION AUDIO PWM GP18, GP19 würde PWM1A und PWM1B an den Pins 24 bzw. 25 verwenden. Diese Option verhindert die Verwendung dieser Pins im BASIC-Programm. Die Audioausgabe wird mit PWM erzeugt, daher ist am Ausgang ein Tiefpassfilter erforderlich. Die Audioausgabe des PicoMite ist sehr laut. Die Verwendung von OPTION POWER und/oder die Stromversorgung über einen separaten 3,3-V-Linearregler kann dies reduzieren.
OPTION AUTOREFRESH OFF | ON		Schwarz-Weiß-Anzeigen können jeweils nur im Vollbildmodus aktualisiert werden. Mit OPTION AUTOREFRESH OFF/ON können Sie steuern, ob ein Schreibbefehl die Anzeige sofort aktualisiert oder nicht. Wenn AUTOREFRESH AUS ist, kann der REFRESH-Befehl verwendet werden, um den Schreibvorgang auszulösen. Dies gilt für folgende Displays: N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI, ST7920, GDEH029A1
OPTION AUTORUN ON oder OPTION AUTORUN n oder OPTION AUTORUN OFF	✓	Weist MMBasic an, beim Einschalten oder Neustart automatisch ein Programm auszuführen. ON bewirkt, dass die automatische Sicherung des aktuellen Programms im RAM abgerufen und ausgeführt wird. Die Angabe von „n“ bewirkt, dass dieser Speicherort im Flash-Speicher geladen und ausgeführt wird. „n“ muss im Bereich von 1 bis 10 liegen. OFF deaktiviert die Autorun-Option und ist die Standardeinstellung für ein neues Programm. Die Eingabe der Unterbrechungstaste (standardmäßig STRG-C) an der Konsole unterbricht das laufende Programm und kehrt zur Eingabeaufforderung zurück.
OPTION BASE 0 | 1		Legen Sie den niedrigsten Wert für Array-Indizes entweder auf 0 oder 1 fest. Dies muss verwendet werden, bevor Arrays deklariert werden, und wird beim Einschalten auf den Standardwert 0 zurückgesetzt.
OPTION BAUDRATE nn		Legt die Baudrate der seriellen Konsole fest (falls konfiguriert).

OPTION BREAK nn		Setzt den Wert des Unterbrechungsschlüssels auf den ASCII-Wert 'nn'. Mit dieser Taste wird ein laufendes Programm unterbrochen. Der Wert der Unterbrechungstaste wird beim Einschalten auf die Taste STRG-C gesetzt, kann aber mit diesem Befehl auf eine beliebige Tastaturtaste geändert werden (z. B. setzt OPTION BREAK 4 die Unterbrechungstaste auf die Taste STRG-D). Wenn Sie diese Option auf Null setzen, wird die Unterbrechungsfunktion vollständig deaktiviert.
OPTION CASE LOWER | UPPER | TITLE	✓	Ändern Sie die Groß-/Kleinschreibung, die zum Auflisten von Befehls- und Funktionsnamen verwendet wird, wenn Sie den LIST-Befehl verwenden. Der Standardwert ist TITLE, aber der alte Standard von MMBasic kann mit OPTION CASE UPPER wiederhergestellt werden.
OPTION COLOURCODE ON oder OPTION COLOURCODE OFF	✓	Farbcodierung für die Ausgabe des Editors ein- oder ausschalten. Schlüsselwörter werden in Cyan, Kommentare in Gelb usw. angezeigt. Die Standardeinstellung ist AUS. Es kann auch das Schlüsselwort COLORCODE (USA-Schreibweise) verwendet werden. Dazu ist ein Terminal-Emulator erforderlich, der die entsprechenden Escape-Codes interpretieren kann (z. B. Tera Term).
OPTION CPUSPEED speed	✓	CPU-Takt ändern. „Speed“ ist der CPU-Takt in KHz im Bereich von 48000 bis 252000. Geschwindigkeiten über 133 MHz gelten als Übertaktung, da dies die angegebene maximale Geschwindigkeit des Standard-Raspberry Pi Pico ist. Wenn keine Option eingestellt ist, wird die CPU-Geschwindigkeit standardmäßig auf 125000 eingestellt.
OPTION COUNT pin1, pin2, pin3, pin4	✓	Gibt an welche Pins als Zähleingänge verwendet werden sollen. Standardmäßig sind dies GP6, GP7, GP8 und GP9. Der Befehl SETPIN definiert den Zählermodus.
OPTION DEFAULT FLOAT | INTEGER | STRING | NONE		Wird verwendet um den Standardtyp für eine nicht explizit definierte Variable festzulegen. Wenn OPTION DEFAULT NONE verwendet wird muss der Typ aller Variablen explizit definiert werden. Wenn ein Programm ausgeführt wird ist der Standardwert auf FLOAT gesetzt, um die Kompatibilität mit Microsoft BASIC und früheren Versionen von MMBasic zu gewährleisten..
OPTION DISPLAY lines [,chars]	✓	Legen Sie die Eigenschaften des Anzeigeterminals fest, das für die Konsole verwendet wird. Sowohl der LIST- als auch der EDIT-Befehl müssen diese Informationen kennen, um den Text für die Anzeige korrekt zu formatieren. 'lines' ist die Anzahl der Zeilen auf dem Display und 'chars' ist die Breite des Displays in Zeichen. Der Standardwert ist 24 Zeilen x 80 Zeichen und wenn diese Option geändert wird, wird sie gespeichert, selbst wenn die Stromversorgung unterbrochen wird.

OPTION EXPLICIT		Wenn Sie diesen Befehl am Anfang eines Programms platzieren, müssen Sie jede Variable explizit mit den Befehlen DIM, LOCAL oder STATIC deklarieren, bevor sie im Programm verwendet werden können. Diese Option ist standardmäßig deaktiviert, wenn ein Programm ausgeführt wird. Wenn es verwendet wird, muss es angegeben werden, bevor Variablen verwendet werden.
OPTION FNKey string$	✓	Definieren Sie die Zeichenfolge, die generiert wird, wenn eine Funktionstaste an der Eingabeaufforderung gedrückt wird. „FNKey“ kann F5 bis F9 sein. Beispiel: OPTION F8 „RUN „+chr$(34)+“myprog“ +chr$(34)+chr$(13)+chr$(10).
OPTION GUI CONTROLS NbrOfGUIControls	✓	Gibt die maximale Anzahl von GUI-Steuerelementen an, die definiert werden können. Jedes Steuerelement verwendet 52 Byte und der verwendete Gesamtspeicher muss auf das nächste Vielfache von 2048 Byte aufgerundet werden. Wenn Sie beispielsweise 70 Steuerelemente angeben werden 4 KB RAM verwendet. Standardmäßig ist die Anzahl der GUI-Steuerelemente auf Null gesetzt sodass diese Option verwendet werden muss bevor GUI-Steuerelemente definiert werden.
OPTION KEYBOARD nn	✓	Konfigurieren Sie eine PS2-Tastatur. Dies kann für die Konsoleneingabe verwendet werden, und alle eingegebenen Zeichen sind über alle Befehle verfügbar, die von der Konsole gelesen werden (seriell über USB). ‘nn ist ein zweistelliger Code, der das Tastaturlayout definiert. Zur Auswahl stehen US für das Standard-Tastaturlayout in den USA, Australien und Neuseeland und UK für das Vereinigte Königreich, DE für Deutschland, FR für Frankreich und ES für Spanien. Die Tastatur muss wie folgt angeschlossen werden: • Schließen Sie GP8 an den CLOCK-Pin der PS2-Buchse an. • Verbinden Sie GP9 mit dem DATA-Pin der PS2-Buchse. • VBUS (5V) oder 3V3 (3,3V) an PS2 Buchse +5V anschließen. • Verbinden Sie GND mit GND der PS2-Buchse. Einige Tastaturen werden mit 3,3 V betrieben und in diesem Fall können die Takt- und Datenstifte direkt angeschlossen werden. **Wenn die Tastatur jedoch mit 5 V versorgt wird, muss für diese Verbindungen eine Pegelanpassung verwendet werden, damit die PicoMite-I/O-Pins nicht mehr als 3,6 V ausgesetzt werden** Der Befehl kann nur über die Befehlszeile ausgeführt werden und führt zu einem Neustart. Diese Einstellung kann mit dem Befehl OPTION KEYBOARD NO_KEYBOARD zurückgesetzt werden. Die optionalen Parameter capslock und numlock setzen den Anfangszustand der Tastatur (Standard 0, 1). Der Wiederholungsstart definiert wie lange bevor sich ein Zeichen wiederholt (gültig 0-3 = 250mSec, 500 ms, 750 ms, 1 s: Standard 1 = 500 ms). Die Wiederholungsrate definiert wie schnell sich ein Zeichen nach der ersten Wiederholung wiederholt (gültig 0-31 = 33mSec bis 500 ms: Standard 12 = 100 ms)

OPTION LCDPANEL *options* oder OPTION LCDPANEL DISABLE	✓	Konfiguriert den PicoMite so, dass er mit einem angeschlossenen LCD-Panel funktioniert. Einzelheiten finden Sie im Abschnitt LCD-Anzeigen.
OPTION LCDPANEL CONSOLE [font [, fc [,bc blight]]] oder OPTION LCDPANEL NOCONSOLE	✓	Konfiguriert das LCD-Anzeigefeld zur Verwendung als Konsolenausgang. Das LCD muss transparenten Text unterstützen (z. B. die Controller ILI9341 oder ST7789_320). „font“ ist die Standardschriftart, „fc“ ist die Standardvordergrundfarbe, „bc“ ist die Standardhintergrundfarbe und „blight“ ist die Standardhelligkeit der Hintergrundbeleuchtung (2 bis 100). Diese Parameter sind optional und standardmäßig auf Schriftart 1, Weiß, Schwarz und 100 % eingestellt. Diese Einstellungen werden beim Einschalten angewendet. Beachten Sie, dass das Scrollen für jede Konsolenausgabe sehr langsam ist, daher wird diese Funktion nicht für den allgemeinen Gebrauch empfohlen. Diese Einstellung wird im Flash gespeichert und beim Start automatisch angewendet. Verwenden Sie zum Deaktivieren den Befehl OPTION LCDPANEL NOCONSOLE.
OPTION LEGACY ON oder OPTION LEGACY OFF		Hiermit wird der Kompatibilitätsmodus mit den im ursprünglichen Color Maximite verwendeten Grafikbefehlen ein- oder ausgeschaltet. Die Befehle COLOUR, LINE, CIRCLE und PIXEL verwenden die Legacy-Syntax und alle Zeichenbefehle akzeptieren Farben im Bereich von 0 bis 7. Hinweise: • Schlüsselwörter wie ROT, BLAU usw. sind nicht implementiert, daher sollten sie bei Bedarf als Konstanten definiert werden. • Informationen zur Syntax der Legacy-Befehle finden Sie im Color Maximite MMBasic Language Manual. Diese kann unter https://geoffg.net/OriginalColourMaximite.html heruntergeladen werden .
OPTION LIST OPTION RESET		Dies listet die Einstellungen aller Optionen auf, die von ihrer Standardeinstellung geändert wurden und vom permanenten Typ sind. Löscht alle Optionen
OPTION PIN nbr		Legen Sie „nbr“ als PIN (Personal Identification Number) für den Zugriff auf die Eingabeaufforderung der Konsole fest. 'nbr' kann eine beliebige Zahl ungleich Null mit bis zu acht Ziffern sein. Immer wenn ein laufendes Programm aus irgendeinem Grund versucht, zur Eingabeaufforderung zurückzukehren, fordert MMBasic diese Nummer an, bevor die Eingabeaufforderung angezeigt wird. Dies ist eine Sicherheitsfunktion, da ein Eindringling ohne Zugriff auf die Eingabeaufforderung das Programm im Speicher nicht auflisten oder ändern oder den Betrieb von MMBasic in irgendeiner Weise ändern kann. Um diese Funktion zu deaktivieren, geben Sie Null als PIN-Nummer ein (z. B. OPTION PIN 0). Eine dauerhafte Sperre kann durch die Verwendung von 99999999 als PIN-Nummer angewendet werden. Wenn eine dauerhafte Sperre angewendet wird oder die PIN-Nummer verloren geht, besteht die einzige Möglichkeit zur Wiederherstellung darin, die PicoMite-Firmware neu zu laden.

OPTION POWER PFM | PWM	✓	Ändert den Betrieb des 3,3-V-Schaltnetzteils. Standardmäßig läuft dies im PFM-Modus. PWM bietet ein besseres Rauschverhalten, ist aber weniger energieeffizient. Beachten Sie, dass das System unter hoher Last unabhängig von dieser Einstellung im PWM-Modus läuft..
OPTION RESET	✓	Setzt alle gespeicherten Optionen auf ihre Standardwerte zurück.
OPTION RTC AUTO ENABLE | DISABLE	✓	Automatisches Laden von time$ und date$ von RTC beim Booten und stündlich aktivieren.
OPTION SDCARD CSpin [,CLKpin, MOSIpin, MISOpin]	✓	Geben Sie die E/A-Pins an, die für die SD-Kartenschnittstelle verwendet werden sollen. Wenn die optionalen Pins nicht angegeben sind, verwendet die SD-Karte die von OPTION SYSTEM SPI angegebenen Pins. Hinweis: Die von OPTION SYSTEM SPI angegebenen Pins müssen ein gültiger Satz von Hardware-SPI-Pins (SPI oder SPI2) sein, die von OPTION SDCARD angegebenen Pins können jedoch beliebige Pins sein. Die von OPTION SYSTEM SPI und OPTION SDCARD angegebenen Pins dürfen nicht identisch sein.
OPTION SYSTEM SPI CLKpin, MOSIpin, MISOpin	✓	Geben Sie den SPI-Port und die Pins für die Verwendung durch Systemgeräte (SD-Karte, LCD-Panel usw.) an. Der PicoMite verwendet einen bestimmten Hardware-SPI-Port für Systemgeräte und überlässt den anderen dem Benutzer. Dieser Befehl gibt an, welche Pins verwendet werden sollen und somit welcher der SPI-Ports verwendet werden soll. Die dem SYSTEM SPI zugewiesenen Pins stehen anderen MMBasic-Befehlen nicht zur Verfügung..
OPTION SERIAL CONSOLE uartapin, uartbpin OPTION SERIAL CONSOLE DISABLE	✓	Geben Sie an, dass auf die Konsole über einen seriellen Hardware-Port zugegriffen werden soll (anstatt virtuell seriell über USB). „uartapin“ und „uartbpin“ können ein beliebiges gültiges Paar von rx- und tx-Pins für entweder COM1 oder COM2 sein. Die Reihenfolge, in der sie angegeben werden, ist nicht wichtig. Die Geschwindigkeit ist standardmäßig auf 115200 Baud eingestellt, kann aber mit OPTION BAUDRATE geändert werden. Kehren Sie zur normalen USB-Konsole zurück.
OPTION SYSTEM I2C sdapin, sclpin	✓	Geben Sie den I2C-Port und die Pins für die Verwendung durch Systemgeräte (LCD-Anzeige und RTC) an. Der PicoMite verwendet einen bestimmten I2C-Port für Systemgeräte und überlässt den anderen dem Programmierer. Dieser Befehl gibt an, welche Pins verwendet werden sollen und somit welcher der I2C-Ports verwendet werden soll. Die dem SYSTEM I2C zugewiesenen Pins stehen nicht für andere MMBasic SETPIN-Einstellungen zur Verfügung, können aber mit dem Standard-I2C-Befehl für zusätzliche I2C-Geräte verwendet werden. Hinweis: I2C(2) OPEN und I2C(2) CLOSE sind in diesem Fall nicht verfügbar
OPTION TAB 2 | 3 | 4 | 8	✓	Legen Sie den Abstand für die Tabulatortaste fest. Standard ist 2.

OPTION TOUCH T_CS pin, T_IRQ pin [, Beep]	✓	Konfiguriert MMBasic für die berührungsempfindliche Funktion eines angeschlossenen LCD-Panels. 'T_CS-Pin' und 'T_IRQ-Pin' sind die PicoMite-E/A-Pins, die für die Chipauswahl bzw. Berührungsunterbrechung verwendet werden (jede freie Pins können verwendet werden). Die verbleibenden Pins werden mit den mit dem OPTION SYSTEM SPI-Befehl angegebenen Pins verbunden. „Beep“ ist ein optionaler Pin, der mit einem kleinen Summer/Beeper verbunden werden kann, um einen „Klick“- oder Piepton zu erzeugen, wenn ein erweitertes Bedienelement berührt wird.
OPTION VCC voltage		Gibt die Spannung (Vcc) an, mit der PicoMite versorgt wird. Bei Verwendung der ADC-Pins zur Spannungsmessung verwendet der PicoMite die Spannung an dem mit VREF gekennzeichneten Pin als Referenz. Diese Spannung kann mit einem DMM genau gemessen und mit diesem Befehl für eine genauere Messung eingestellt werden. Der Parameter wird nicht gespeichert und sollte entweder auf der Kommandozeile oder in einem Programm initialisiert werden. Der Standardwert, wenn nicht festgelegt, ist 3.3.

Befehlsreferenz

Eckige Klammern zeigen an, dass der Parameter oder die Zeichen optional sind.

‘ (einfaches Anführungszeichen)	Beginnt einen Kommentar und jeglicher nachfolgender Text wird ignoriert. Kommentare können überall in einer Zeile platziert werden.
? (Fragezeichen)	Abkürzung für den PRINT-Befehl.
ADC OPEN freq, n_channels [,interrupt]	Dies weist bis zu 4 ADC-Kanälen GP26, GP27, GP28 und GP29 zu und setzt sie zur Konvertierung bei der angegebenen Frequenz. Die maximale Gesamtfrequenz beträgt 500 kHz (z. B. 125 kHz, wenn alle vier Kanäle abgetastet werden sollen). Wenn die Anzahl der Kanäle eins ist, wird immer GP26 verwendet, wenn zwei, dann GP26 und GP27 usw.. Die Abtastung mehrerer Kanäle erfolgt sequentiell (es gibt nur einen ADC). Die angegebenen Pins sind für die Funktion gesperrt, wenn ADC OPEN aktiv ist Der optionale Interrupt-Parameter gibt einen Interrupt an, der aufgerufen werden soll, wenn die Konvertierung abgeschlossen ist. Wenn nicht angegeben, blockiert die Konvertierung
ADC FREQUENCY freq	Dies ändert die Abtastfrequenz der ADC-Wandlung, ohne dass sie geschlossen und wieder geöffnet werden muss
ADC CLOSE	Gibt die Stifte für den normalen Gebrauch frei
ADC START array1!() [,array2!()] [,array3!()] [,array4!()]	Dadurch wird die Konvertierung in die angegebenen Arrays gestartet. Die Arrays müssen Gleitkommazahlen und dieselbe Größe haben. Die Größe der Arrays definiert die Anzahl der Konvertierungen. Die Ergebnisse werden als Spannung zwischen 0 und OPTION VCC (standardmäßig 3,3 V) zurückgegeben. Start kann wiederholt aufgerufen werden, sobald der ADC OFFEN ist
ARC x, y, r1, [r2], a1, a2 [, c]	Zeichnet einen Kreisbogen mit gegebener Farbe und Breite zwischen zwei Radialen (definiert in Grad). Parameter für den ARC-Befehl sind: x: X-Koordinate des Bogenmittelpunkts y: Y-Koordinate des Bogenmittelpunkts r1: Innenradius des Bogens r2: Außenradius des Bogens - kann weggelassen werden, wenn er 1 Pixel breit ist a1: Startwinkel des Bogens in Grad a2: Bogenendwinkel in Grad c: Farbe des Bogens (wenn weggelassen, wird standardmäßig die Vordergrundfarbe verwendet) Null Grad befindet sich auf der 9-Uhr-Position.
AUTOSAVE oder AUTOSAVE CRUNCH	Wechseln Sie in den automatischen Programmeingabemodus. Dieser Befehl nimmt Textzeilen aus der seriellen Eingabe der Konsole und speichert sie im Programmspeicher. Dieser Modus wird durch Eingabe von Strg-Z beendet, was dann bewirkt, dass die empfangenen Daten in den Programmspeicher übertragen werden und das vorherige Programm überschreiben. Die Option CRUNCH weist MMBasic an, vor dem Speichern alle Kommentare, Leerzeilen und unnötigen Leerzeichen aus dem Programm zu entfernen. Dies kann bei großen Programmen verwendet werden, damit sie in den begrenzten Speicher passen. CRUNCH kann mit dem einzelnen Buchstaben C abgekürzt werden. Dieser Befehl kann jederzeit mit Strg-C abgebrochen werden, wodurch der Programmspeicher unberührt bleibt.

	Dies ist eine Möglichkeit, ein BASIC-Programm in den PicoMite zu übertragen. Das zu übertragende Programm kann in einen Terminal-Emulator eingefügt werden, und dieser Befehl erfasst den Textstrom und speichert ihn im Programmspeicher. Es kann auch verwendet werden, um ein kleines Programm direkt am Konsoleneingang einzugeben.
BITBANG BITSTREAM pinno, n_transitions, array%()	Dieser Befehl wird verwendet, um eine äußerst genaue Bitfolge auf dem angegebenen Pin zu erzeugen. Der Pin muss zuvor als Ausgang eingerichtet und auf den gewünschten Ausgangspegel gesetzt worden sein. Anmerkungen: • Das Array enthält die Länge jeder Ebene im Bitstrom in Mikrosekunden. Die maximal zulässige Dauer beträgt 65,5 ms • Der erste Übergang erfolgt unmittelbar nach Ausführung des Befehls. • Der letzte Punkt im Array wird außer der Definition der Zeit ignoriert, bevor die Steuerung an das Programm oder die Befehlszeile zurückgegeben wird. • Der Pin wird im Anfangszustand belassen, wenn die Anzahl der Übergänge gerade ist, und im entgegengesetzten Zustand, wenn die Anzahl der Übergänge ungerade ist..
BITBANG HUMID pin, tvar, hvar [,DHT11]	Gibt die Temperatur und Luftfeuchtigkeit mit dem DHT22-Sensor zurück. Alternative Versionen des DHT22 sind das AM2303 oder das RHT03 (alle sind kompatibel). 'Pin' ist der I/O-Pin, der mit dem Sensor verbunden ist. Jeder I/O-Pin kann verwendet werden. „tvar“ ist die Variable, die die gemessene Temperatur enthält, und „hvar“ ist die gleiche für die Luftfeuchtigkeit. Beide müssen vorhanden sein und beide müssen Fließkommavariablen sein. Zum Beispiel: HUMID 3, TEMP!, FEUCHTIGKEIT! Die Temperatur wird in ºC gemessen und die Luftfeuchtigkeit in Prozent relative Luftfeuchtigkeit. Beide werden mit einer Auflösung von 0,1 gemessen. Wenn ein Fehler auftritt (Sensor nicht angeschlossen oder beschädigtes Signal), sind beide Werte 1000,0. Normalerweise sollte der DHT22 mit 3,3 V betrieben werden, um seinen Ausgang für den PicoMite unter 3,6 V zu halten, und der Signalstift von sollte durch einen 1K bis 10K Widerstand (4,7K empfohlen) auf 3,3V hochgezogen werden. Der optionale DHT11-Parameter modifiziert die Timings, um mit dem DHT11 zu arbeiten. Für DHT11 auf 1 und für DHT22 auf 0 setzen oder weglassen.
BITBANG WS2812 type, pin, nbr, value%[()]	Dieser Befehl gibt die erforderlichen Signale aus, um einen oder mehrere WS2812-LED-Chips anzusteuern, die mit 'pin' verbunden sind. Beachten Sie, dass der Pin auf einen digitalen Ausgang gesetzt werden muss, bevor dieser Befehl verwendet wird. 'Typ' ist ein einzelnes Zeichen, das den Typ des angesteuerten Chips angibt: O = original WS2812 B = WS2812B S = SK6812 „nbr“ ist die Anzahl der LEDs in der Kette (1 bis 256). Das 'value%()'-Array sollte ein ganzzahliges Array sein, das so bemessen ist, dass es genau die gleiche Anzahl von Elementen wie die Anzahl der anzusteuernden LEDs hat. Jedes Element im Array sollte die Farbe im normalen RGB888-Format enthalten (also 0 bis &HFFFFFF). Wenn nur eine LED angeschlossen ist, sollte für value% eine einzelne Ganzzahl verwendet werden dh kein Array.
BITBANG LCD INIT d4, d5, d6, d7, rs, en	Zeigt Text auf einem LCD-Zeichenanzeigemodul an. Dieser Befehl funktioniert mit den meisten 1-zeiligen, 2-zeiligen oder 4-zeiligen LCD-

oder BITBANG LCD line, pos, text$ oder BITBANG LCD CLEAR oder BITBANG LCD CLOSE	Modulen, die den KS0066-, HD44780- oder SPLC780-Controller verwenden (dies kann jedoch nicht garantiert werden). Der Befehl LCD INIT wird verwendet, um das LCD-Modul für die Verwendung zu initialisieren. 'd4' bis 'd7' sind die E/A-Pins, die mit den Eingängen D4 bis D7 auf dem LCD-Modul verbunden sind (Eingänge D0 bis D3 sollten mit Masse verbunden sein). 'rs' ist der Pin, der mit dem Registerauswahleingang auf dem Modul verbunden wird (manchmal auch als CMD bezeichnet). 'en' ist der Pin, der mit dem Enable- oder Chip-Select-Eingang des Moduls verbunden ist. Der R/W-Eingang am Modul sollte immer geerdet sein. Die obigen I/O-Pins werden durch diesen Befehl automatisch auf Ausgänge gesetzt. Wenn das Modul initialisiert wurde, können Daten mit dem LCD-Befehl darauf geschrieben werden. 'line' ist die Zeile auf dem Display (1 bis 4) und 'pos' ist die Zeichenposition auf der Zeile (die erste Position ist 1). 'text$' ist eine Zeichenfolge, die den Text enthält, der auf das LCD-Display geschrieben werden soll. 'pos' kann auch C8, C16, C20 oder C40 sein, in diesem Fall wird die Zeile gelöscht und der Text auf einer Anzeige mit 8 oder 16, 20 oder 40 Zeilen zentriert. Beispielsweise: LCD 1, C16, "Hallo" LCD CLEAR löscht alle auf dem LCD angezeigten Daten und LCD CLOSE beendet die LCD-Funktion und bringt alle I/O-Pins in den nicht konfigurierten Zustand zurück. Weitere Einzelheiten finden Sie im Kapitel „Spezielle Hardwaregeräte“.
BITBANG LCD CMD d1 [, d2 [, etc]] oder BITBANG LCD DATA d1 [, d2 [, etc]]	Diese Befehle senden ein oder mehrere Bytes entweder als Befehl (LCD CMD) oder als Daten (LCD DATA) an ein LCD-Display. Jedes Byte ist eine Zahl zwischen 0 und 255 und muss durch Kommas getrennt werden. Das LCD muss zuvor mit dem Befehl LCD INIT (siehe oben) initialisiert worden sein. Diese Befehle können verwendet werden, um ein nicht standardmäßiges LCD im "Raw-Modus" zu steuern, oder sie können verwendet werden, um spezielle Funktionen wie Scrollen, Cursor und benutzerdefinierte Zeichensätze zu aktivieren. Sie müssen sich auf das Datenblatt Ihres LCDs beziehen, um die erforderlichen Befehls- und Datenwerte zu finden..

BLIT READ [#]b, x, y, w, h BLIT WRITE [#]b, x, y[, w] [, h] BLIT LOAD [#]b, fname$ [,x] [,y] [,w] [,h] BLIT CLOSE [#]b	Kopieren Sie einen Abschnitt des Anzeigebildschirms auf einem LCD-Panel mit den Controllern ILI9341 und ST7789_320 in oder aus einem Pufferspeicher. BLIT READ kopiert einen Teil der Anzeige in den Speicherpuffer '#b'. Die Quellkoordinate ist 'x' und 'y' und die Breite des zu kopierenden Anzeigebereichs ist 'w' und die Höhe ist 'h'. Wenn dieser Befehl verwendet wird, wird der Speicherpuffer automatisch erstellt und ausreichend Speicher zugewiesen. Mit dem Befehl BLIT CLOSE kann dieser Puffer freigegeben und der Speicher wiederhergestellt werden. BLIT WRITE kopiert den Speicherpuffer '#b' auf die Anzeige. Die Zielkoordinate ist 'x' und 'y' und die Breite/Höhe des zu kopierenden Puffers ist 'w' und 'h'. Wenn w,h weggelassen wird, nimmt es standardmäßig die Breite und Höhe des Blit-Puffers an. BLIT LOAD lädt einen Blit-Puffer aus einer 24-Bit-bmp-Bilddatei. x,y definieren die Startposition im Bild, um mit dem Laden zu beginnen, und w,h geben die Breite und Höhe des zu ladenden Bereichs an. Dieser Befehl funktioniert auf den meisten Anzeigetafeln (nicht nur auf Tafeln, die den ILI9341-Controller verwenden). Beispiel: `BLIT LOAD #1,"image1", 50,50,100,100` lädt einen Bereich von 100 Pixeln im Quadrat mit der oberen linken Ecke bei 50,50 aus dem Bild image1.bmp BLIT CLOSE schließt den Speicherpuffer '#b', damit er für eine weitere BLIT READ-Operation verwendet werden kann, und stellt den verwendeten Speicher wieder her. Anmerkungen: • Es sind acht Puffer von Nr. 1 bis Nr. 8 verfügbar. • Bei der Angabe der Puffernummer ist das #-Zeichen optional. • Alle anderen Argumente sind in Pixel angegeben. • Das Display muss die Controller ILI9341 und ST7789_320 verwenden.
BLIT x1, y1, x2, y2, w, h	Kopiert einen Abschnitt des Anzeigebildschirms in einen anderen Teil der Anzeige. Die Quellkoordinate ist 'x1' und 'y1'. Die Zielkoordinate ist 'x2' und 'y2'. Die Breite des zu kopierenden Bildschirmbereichs ist 'w' und die Höhe 'h'. Alle Argumente sind in Pixel angegeben. Das Display muss den ILI9341-Controller verwenden.
BOX x, y, w, h [, lw] [,c] [,fill]	Zeichnet ein Kästchen auf dem angeschlossenen LCD-Panel mit der oberen linken Ecke bei 'x' und 'y' mit einer Breite von 'w' Pixeln und einer Höhe von 'h' Pixeln. 'lw' ist die Breite der Seiten der Box und kann Null sein. Es ist standardmäßig auf 1. 'c' gibt die Farbe an und verwendet standardmäßig die Standard-Vordergrundfarbe, wenn sie nicht angegeben ist. 'fill' ist die Füllfarbe. Es kann weggelassen oder auf -1 gesetzt werden, in diesem Fall wird das Feld nicht ausgefüllt. Alle Parameter können als Arrays ausgedrückt werden, und die Software zeichnet die Anzahl der Boxen auf, die durch die Abmessungen des kleinsten Arrays bestimmt wird. 'x', 'y', 'w' und 'h' müssen alle Arrays oder alle einzelne Variablen/Konstanten sein, sonst wird ein Fehler generiert. 'lw', 'c' und fill können entweder Arrays oder einzelne Variablen/Konstanten sein. Eine Definition der Farben und Grafikkoordinaten finden Sie im Kapitel "Grundlegende Zeichenbefehle"..

CALL usersubname$ [,usersubparameters,....]	Dies ist eine effiziente Möglichkeit, benutzerdefinierte Subroutinen programmgesteuert aufzurufen (siehe auch die Funktion CALL()). In vielen Fällen kann es Ihnen ermöglichen, komplexe SELECT- und IF THEN ELSEIF ENDIF-Klauseln loszuwerden, und es wird viel effizienter verarbeitet. Der „usersubname$“ kann eine beliebige Zeichenfolge oder Variable oder Funktion sein, die sich in den Namen einer normalen Benutzersubroutine auflöst (kein eingebauter Befehl). Die „usersubparameters“ sind die gleichen Parameter, die verwendet würden, um die Subroutine direkt aufzurufen. Eine typische Verwendung könnte das Schreiben einer beliebigen Art von Emulator sein, bei dem eine von einer großen Anzahl von Unterroutinen abhängig von einer Variablen aufgerufen werden sollte. Es ermöglicht auch die Übergabe eines Subroutinennamens an eine andere Subroutine oder Funktion als Variable.
CHDIR dir$	Ändert das aktuelle Arbeitsverzeichnis auf der SD-Karte in „dir$“. Der spezielle Eintrag „..“ repräsentiert das Elternverzeichnis des aktuellen Verzeichnisses und „.“ stellt das aktuelle Verzeichnis dar. "/" ist das Root-Verzeichnis.
CIRCLE x, y, r [,lw] [, a] [, c] [, fill]	Zeichnen Sie einen Kreis auf dem Videoausgang, zentriert bei 'x' und 'y' mit einem Radius von 'r' auf dem angeschlossenen LCD-Panel . „lw“ ist optional und ist die Linienbreite (standardmäßig 1). 'c' ist die optionale Farbe und ist standardmäßig die aktuelle Vordergrundfarbe, wenn sie nicht angegeben ist. Das optionale 'a' ist eine Fließkommazahl, die das Seitenverhältnis definiert. Wenn das Seitenverhältnis nicht angegeben ist, ist der Standardwert 1,0, was einen Standardkreis ergibt. 'fill' ist die Füllfarbe. Es kann weggelassen oder auf -1 gesetzt werden, in diesem Fall wird das Feld nicht ausgefüllt. Alle Parameter können als Arrays ausgedrückt werden, und die Software zeichnet die Anzahl der Kreise auf, die durch die Abmessungen des kleinsten Arrays bestimmt werden. 'x', 'y' und 'r' müssen alle Arrays oder alle einzelne Variablen/Konstanten sein, sonst wird ein Fehler generiert. 'lw', 'a', 'c' und fill können entweder Arrays oder einzelne Variablen/Konstanten sein. Eine Definition der Farben und Grafikkoordinaten finden Sie im Kapitel "Grundlegende Zeichenbefehle".
CLEAR	Löscht alle Variablen und stellt den von ihnen belegten Speicher wieder her. Siehe ERASE zum Löschen bestimmter Array-Variablen.
CLOSE [#]fnbr [,[#]fnbr] …	Schließt die zuvor geöffnete(n) Datei(en) mit der Dateinummer „#fnbr“. Das # ist optional. Siehe auch den OPEN-Befehl.
CLS [colour]	Löscht den Bildschirm des LCD-Panels. Optional kann 'Farbe' angegeben werden, die beim Löschen des Bildschirms für den Hintergrund verwendet wird.
COLOUR fore [, back] or COLOR fore [, back]	Legt die Standardfarbe für Befehle (PRINT usw.) fest, die auf dem angeschlossenen LCD-Feld angezeigt werden. 'fore' ist die Vordergrundfarbe, 'back' ist die Hintergrundfarbe. Der Hintergrund ist optional und wird, wenn nicht angegeben, standardmäßig schwarz.
CONST id = expression [, id = expression] … etc	Erstellen Sie eine konstante Kennung, die nach der Erstellung nicht mehr geändert werden kann. 'id' ist der Bezeichner, der den gleichen Regeln wie für Variablen folgt. Der Bezeichner kann ein Typensuffix (!, % oder $) haben, dies ist jedoch nicht erforderlich. Wenn es angegeben ist, muss es mit dem Typ von 'Ausdruck' übereinstimmen. 'Ausdruck' ist der Wert des Bezeichners und kann ein normaler Ausdruck (einschließlich benutzerdefinierter Funktionen) sein, der ausgewertet wird, wenn die Konstante erstellt wird.

	Eine außerhalb eines Subs oder einer Funktion definierte Konstante ist global und kann im gesamten Programm angezeigt werden. Eine in einem Sub oder einer Funktion definierte Konstante ist lokal für diese Routine und verdeckt eine globale Konstante mit demselben Namen.
CONTINUE	Setzt die Ausführung eines Programms fort, das durch eine END-Anweisung, einen Fehler oder STRG-C gestoppt wurde. Das Programm wird mit der nächsten Anweisung nach dem vorherigen Haltepunkt neu gestartet. Beachten Sie, dass es nicht immer möglich ist, das Programm korrekt fortzusetzen – dies gilt insbesondere für komplexe Programme mit Grafiken, verschachtelten Schleifen und/oder verschachtelten Unterprogrammen und Funktionen.
CONTINUE DO or CONTINUE FOR	Zum Ende einer DO/LOOP- oder FOR/NEXT-Schleife springen. Die Schleifenbedingung wird dann getestet, und wenn sie noch gültig ist, wird die Schleife mit der nächsten Iteration fortgesetzt.
COPY fname1$ TO fname2$	Kopieren Sie eine Datei von „fname1$“ nach „fname2$“. Beides sind Saiten. Ein Verzeichnispfad kann sowohl in „fname$“ als auch in „fname$“ verwendet werden. Wenn sich die Pfade unterscheiden, wird die in „fname$“ angegebene Datei mit dem angegebenen Dateinamen in den in „fname2$“ angegebenen Pfad kopiert.
CPU RESTART	Erzwingt einen Neustart des Prozessors. Dadurch werden alle Variablen gelöscht und alles zurückgesetzt (z. B. Timer, COM-Ports, I2C usw.), ähnlich wie bei einer Einschaltsituation, jedoch ohne das Einschaltbanner. Wenn OPTION AUTORUN gesetzt wurde, wird das Programm im angegebenen Flash-Speicherort neu gestartet.
CPU SLEEP n	Veranlasst den Prozessor, für ‘n’ Sekunden zu schlafen. Beachten Sie, dass die CPU keinen echten Energiesparmodus hat, sodass die Energieeinsparung begrenzt ist.
CSUB name [type [, type] …] hex [[hex[…] hex [[hex[…] END CSUB	Definiert den Binärcode für ein eingebettetes Maschinencode-Programmmodul, das in C oder ARM-Assembler geschrieben ist. Das Modul erscheint in MMBasic als Befehl „Name“ und kann auf die gleiche Weise wie ein eingebauter Befehl verwendet werden. In einem Programm können mehrere eingebettete Routinen verwendet werden, wobei jede ein anderes Modul mit einem anderen "Namen" definiert. Das erste „Hex“-Wort ist ein 32-Bit-Wort, das der Offset in Bytes vom Start des CSUB bis zum Einstiegspunkt der eingebetteten Routine (normalerweise die Funktion main()) ist. Die folgenden Hex-Wörter sind der kompilierte Binärcode für das Modul. Diese werden beim Speichern des Programms automatisch in MMBasic programmiert. Jedes „Hex“ muss aus genau acht Hex-Ziffern bestehen, die die Bits in einem 32-Bit-Wort darstellen, und durch ein oder mehrere Leerzeichen oder neue Zeilen getrennt sein. Der Befehl muss mit einem passenden END CSUB abgeschlossen werden. Alle Fehler im Datenformat werden gemeldet, wenn das Programm ausgeführt wird. Während der Ausführung überspringt MMBasic alle CSUB-Befehle, sodass sie an beliebiger Stelle im Programm platziert werden können. Der Typ jedes Parameters kann in der Definition angegeben werden. Beispielsweise: CSUB MySub Ganzzahl, Ganzzahl, Zeichenfolge Dies gibt an, dass es drei Parameter geben wird, wobei die ersten beiden Ganzzahlen und der dritte eine Zeichenfolge sind. Notiz: • Es können bis zu zehn Argumente angegeben werden ('arg1', 'arg2' usw.).

	• Wenn eine Variable oder ein Array als Argument angegeben wird, erhält die C-Routine einen Zeiger auf den Speicher, der der Variablen oder dem Array zugewiesen ist, und die C-Routine kann diesen Speicher ändern, um einen Wert an den Aufrufer zurückzugeben. Bei Arrays sollten diese mit leeren Klammern übergeben werden, z. arg(). In der CSUB wird das Argument als Zeiger auf das erste Element des Arrays geliefert. • Konstanten und Ausdrücke werden an die eingebettete C-Routine als Zeiger auf einen temporären Speicherplatz übergeben, der den Wert enthält.
CTRLVAL(#ref) =	Dieser Befehl setzt den Wert eines erweiterten Steuerelements. '#ref' ist die Referenznummer des Steuerelements. Für Ein-/Aus-Steuerelemente wie Kontrollkästchen überschreibt es jede Berührungseingabe und kann zum Drücken/Loslassen von Schaltern, Aktivieren/Deaktivieren von Kontrollkästchen usw. verwendet werden. Ein Wert von Null ist deaktiviert oder deaktiviert, und ein Wert ungleich Null schaltet das Steuerelement ein. Bei einer LED bewirkt dies, dass die LED leuchtet oder ausgeschaltet wird. Es kann auch verwendet werden, um den Anfangswert von Drehfeldern, Textfeldern usw. festzulegen. Beispielsweise: CTRLVAL(#10) = 12.4 Allen Steuerelementen wird erwartet, dass ihnen eine Zahl (float oder integer) zugewiesen wird, mit Ausnahme von Frame, Caption, Display Box, Text Box und Format Box, die eine Zeichenfolge erwarten.
DATA constant[,constant]...	Speichert numerische und String-Konstanten, auf die durch READ zugegriffen werden soll. Im Allgemeinen sollten Zeichenfolgenkonstanten in doppelte Anführungszeichen (") eingeschlossen werden. Eine Ausnahme besteht, wenn die Zeichenfolge nur aus alphanumerischen Zeichen besteht, die keine MMBasic-Schlüsselwörter darstellen (wie THEN, WHILE usw.). In diesem Fall sind keine Anführungszeichen erforderlich. Numerische Konstanten können auch Ausdrücke wie 5 * 60 sein.
DATE$ = "DD-MM-YY[YY]" or DATE$ = "DD/MM/YY[YY]" or DATE$ =”YYYY-MM-DD” or DATE$=”YYYY/MM/DD”	Stellen Sie das Datum der internen Uhr/des internen Kalenders ein. TT, MM und JJ sind Zahlen, zum Beispiel: DATE$ = "28-7-2014" Beim Einschalten wird das Datum auf „01.01.2000“ gesetzt.

DEFINEFONT #Nbr hex [[hex[…] hex [[hex[…] END DEFINEFONT	Dies definiert eine eingebettete Schriftart, die neben oder als Ersatz für die auf einem angeschlossenen LCD-Panel verwendete(n) integrierte(n) Schriftart(en) verwendet werden kann. Diese funktionieren genauso wie die eingebauten Schriftarten (d. h. mit dem FONT-Befehl ausgewählt oder im TEXT-Befehl angegeben). Im Ordner „Embedded Fonts“ in der ZIP-Datei der PicoMite-Distribution finden Sie eine Auswahl eingebetteter Schriftarten und eine vollständige Beschreibung, wie sie erstellt werden. '#Nbr' ist die Referenznummer der Schriftart (von 1 bis 16). Es kann die gleiche Nummer wie eine eingebaute Schriftart sein und ersetzt in diesem Fall die eingebaute Schriftart. Jedes „Hex“ muss aus genau acht Hex-Ziffern bestehen und durch Leerzeichen oder neue Zeilen vom nächsten getrennt sein. • Mehrere Zeilen von 'Hex'-Wörtern können verwendet werden, wobei der Befehl durch ein passendes END DEFINEFONT abgeschlossen wird. • In einem Programm können mehrere eingebettete Schriftarten verwendet werden, wobei jede eine andere Schriftart mit einer anderen Schriftartnummer definiert. • Während der Ausführung überspringt MMBasic alle DEFINEFONT-Befehle, damit sie an beliebiger Stelle im Programm platziert werden können. • Eventuelle Fehler im Datenformat werden beim Speichern des Programms gemeldet.

DIM [type] decl [,decl]...
wo 'decl' ist:
var [länge] [typ] [init]
„var“ ist ein Variablenname mit optionalen Dimensionen
'length' wird verwendet, um die maximale Größe des Strings auf 'n' zu setzen, wie in LENGTH n
'Typ' ist entweder FLOAT oder
INTEGER oder STRING (dem Typ kann das Schlüsselwort AS vorangestellt werden - wie bei AS FLOAT)
'init' ist der Wert zum Initialisieren der Variablen und besteht aus:
= <Ausdruck>
Für eine einfache Variable wird ein Ausdruck verwendet, für ein Array wird eine Liste von kommagetrennten Ausdrücken verwendet, die von Klammern umgeben sind.
Beispiele:
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
	Deklariert eine oder mehrere Variablen (d. h. macht den Variablennamen und seine Eigenschaften dem Interpreter bekannt).
	Wenn OPTION EXPLICIT verwendet wird (wie empfohlen), sind die Befehle DIM, LOCAL oder STATIC die einzige Möglichkeit, eine Variable zu erstellen. Wenn diese Option nicht verwendet wird, ist die Verwendung des DIM-Befehls optional, und wenn sie nicht verwendet wird, wird die Variable automatisch erstellt, wenn zum ersten Mal darauf verwiesen wird.
	Der Typ der Variablen (d. h. String, Float oder Integer) kann auf drei Arten angegeben werden:
	Durch Verwendung eines Typsuffixes (z. B. !, % oder $ für Float, Integer oder String). Beispielsweise:
	`DIM nbr%, amount!, name$`
	Durch Verwendung eines der Schlüsselwörter FLOAT, INTEGER oder STRING unmittelbar nach dem Befehl DIM und bevor die Variable(n) aufgelistet werden. Der angegebene Typ gilt dann für alle aufgeführten Variablen (muss also nicht wiederholt werden). Beispielsweise:
	`DIM STRING first_name, last_name, city`
	Durch Verwendung der Microsoft-Konvention, das Schlüsselwort „AS“ und das Typschlüsselwort (d. h. FLOAT, INTEGER oder STRING) nach jeder Variablen zu verwenden. Bei dieser Methode muss der Typ für jede Variable angegeben werden und kann von Variable zu Variable geändert werden.
	Beispielsweise:
	`DIM amount AS FLOAT, name AS STRING`
	Fließkomma- oder Integer-Variablen werden beim Erstellen auf Null gesetzt und Zeichenfolgen werden auf eine leere Zeichenfolge (z. B. "") gesetzt. Sie können den Wert der Variablen mit etwas anderem initialisieren, indem Sie ein Gleichheitszeichen (=) und einen Ausdruck nach der Variablendefinition verwenden. Beispielsweise:
	`DIM STRING city = "Perth", house = "Brick"`
	Der Initialisierungswert kann ein Ausdruck (einschließlich anderer Variablen) sein und wird ausgewertet, wenn der DIM-Befehl ausgeführt

DIM STRING strn(200) LENGTH 20
DIM a = 1234, b = 345
DIM STRING strn = "text"
DIM x%(3) = (11, 22, 33, 44)
	wird. Weitere Beispiele zur Syntax finden Sie im Kapitel „Variablen definieren und verwenden“.
	Der DIM-Befehl deklariert nicht nur einfache Variablen, sondern auch Array-Variablen (d. h. eine indizierte Variable mit einer Reihe von Dimensionen). Nach dem Namen der Variablen werden die Dimensionen durch eine Liste von Zahlen angegeben, die durch Kommas getrennt und in Klammern eingeschlossen sind. Beispielsweise:
	`DIM array(10, 20)`
	Jede Zahl gibt die Anzahl der Elemente in jeder Dimension an. Normalerweise beginnt die Nummerierung jeder Dimension bei 0, aber der Befehl OPTION BASE kann verwendet werden, um dies auf 1 zu ändern.
	Das obige Beispiel spezifiziert ein zweidimensionales Array mit 11 Elementen (0 bis 10) in der ersten Dimension und 21 (0 bis 20) in der zweiten Dimension. Die Gesamtzahl der Elemente beträgt 231, und da jede Gleitkommazahl auf dem PicoMite 8 Bytes benötigt, werden insgesamt 1848 Bytes Speicher zugewiesen.
	Strings werden standardmäßig 255 Bytes (d. h. Zeichen) Speicher für jedes Element zuweisen, und dies kann schnell Speicher verbrauchen, wenn Arrays von Strings definiert werden. In diesem Fall kann das Schlüsselwort LENGTH verwendet werden, um die Speichermenge anzugeben, die jedem Element zugewiesen werden soll, und damit die maximale Länge der Zeichenfolge, die gespeichert werden kann. Diese Zuordnung ('n') kann zwischen 1 und 255 Zeichen lang sein.
	Beispiel:: `DIM STRING s(5, 10)` deklariert ein String-Array mit 66 Elementen, das 16.896 Byte Speicher verbraucht, während:
	`DIM STRING s(5, 10) LENGTH 20`
	Verbraucht nur 1.386 Byte Speicher. Beachten Sie, dass die jedem Element zugewiesene Speichermenge n + 1 beträgt, da das zusätzliche Byte verwendet wird, um die tatsächliche Länge der in jedem Element gespeicherten Zeichenfolge zu verfolgen.
	Wenn einem Element des Arrays ein String zugewiesen wird, der länger als 'n' ist, wird ein Fehler erzeugt. Abgesehen davon verhalten sich String-Arrays, die mit dem LENGTH-Schlüsselwort erstellt wurden, genauso wie andere String-Arrays. Dieses Schlüsselwort kann auch mit Nicht-Array-String-Variablen verwendet werden, spart jedoch keinen Speicherplatz.
	Im obigen Beispiel können Sie auch die Microsoft-Syntax verwenden, um den Typ nach dem Längenqualifizierer anzugeben. Beispielsweise:
	`DIM s(5, 10) LENGTH 20 AS STRING`
	Arrays können auch initialisiert werden, wenn sie deklariert werden, indem am Ende der Deklaration ein Gleichheitszeichen (=) gefolgt von einer Werteliste in Klammern hinzugefügt wird. Beispielsweise:
	`DIM INTEGER nbr(4) = (22, 44, 55, 66, 88)`
	oder `DIM s$(3) = ("foo", "boo", "doo", "zoo")`
	Beachten Sie, dass die Anzahl der Initialisierungswerte mit der Anzahl der Elemente im Array übereinstimmen muss, einschließlich des von OPTION BASE festgelegten Basiswerts. Wenn ein mehrdimensionales Array initialisiert wird, wird die erste Dimension zuerst initialisiert, gefolgt von der zweiten usw.
	Beachten Sie außerdem dass die Initialisierungswerte nach LENGTH und nach der Typdeklaration stehen müssen.

DO <statements> LOOP	Diese Struktur wird für immer wiederholt; der Befehl EXIT DO kann verwendet werden, um die Schleife zu beenden, oder die Steuerung muss explizit durch Befehle wie GOTO oder EXIT SUB (falls in einer Unterroutine) außerhalb der Schleife übertragen werden.

DO WHILE expression <statements> LOOP	Schleifen, während „Ausdruck“ wahr ist (dies entspricht der älteren WHILE-WEND-Schleife). Wenn der Ausdruck zu Beginn falsch ist, werden die Anweisungen in der Schleife kein einziges Mal ausgeführt.
DO <statements> LOOP UNTIL expression	Ausdruck Führt eine Schleife durch, bis der UNTIL folgende Ausdruck wahr ist. Da der Test am Ende der Schleife durchgeführt wird, werden die Anweisungen innerhalb der Schleife mindestens einmal ausgeführt, selbst wenn der Ausdruck wahr ist.
EDIT	Ruft den Vollbild-Editor auf. Einzelheiten zur Verwendung des Editors finden Sie im Abschnitt Vollbild-Editor.
ELSE	Fügt eine optionale Standardbedingung in eine mehrzeilige IF-Anweisung ein. Weitere Einzelheiten finden Sie in der mehrzeiligen IF-Anweisung.
ELSEIF expression THEN oder ELSE IF expression THEN	Fügt eine optionale Nebenbedingung in eine mehrzeilige IF-Anweisung ein. Weitere Einzelheiten finden Sie in der mehrzeiligen IF-Anweisung.
END	Beendet das laufende Programm und kehrt zur Eingabeaufforderung zurück.
END CSUB	Markiert das Ende einer C-Subroutine. Siehe CSUB-Befehl. Jede CSUB muss eine und nur eine übereinstimmende END CSUB-Anweisung haben.
END FUNCTION	Markiert das Ende einer benutzerdefinierten Funktion. Siehe FUNCTION-Befehl. Jede Funktion darf genau eine passende END FUNCTION-Anweisung haben. Verwenden Sie EXIT FUNCTION, wenn Sie von einer Funktion innerhalb ihres Körpers zurückkehren müssen.

ENDIF
oder
END IF
	Beendet eine mehrzeilige IF-Anweisung.
	Weitere Einzelheiten finden Sie in der mehrzeiligen IF-Anweisung.

END SUB	Markiert das Ende einer benutzerdefinierten Subroutine. Siehe den SUB-Befehl. Jeder Sub muss genau eine übereinstimmende END SUB-Anweisung haben. Verwenden Sie EXIT SUB, wenn Sie von einer Subroutine innerhalb ihres Hauptteils zurückkehren müssen.
ERASE variable [,variable]...	Löscht Variablen und gibt den ihnen zugewiesenen Speicher frei. Dies funktioniert mit Array-Variablen und normalen (Nicht-Array-) Variablen. Arrays können mit leeren Klammern (z. B. dat()) oder einfach durch Angabe des Variablennamens (z. B. dat) angegeben werden. Verwenden Sie CLEAR, um alle Variablen auf einmal zu löschen (einschließlich Arrays).
ERROR [error_msg$]	**Erzwingt einen Fehler und beendet das Programm. Dies wird normalerweise beim Debuggen oder zum Abfangen von Ereignissen verwendet, die nicht auftreten sollten**.
EXIT DO EXIT FOR EXIT FUNCTION EXIT SUB	EXIT DO bietet einen vorzeitigen Ausstieg aus einem DO...LOOP EXIT FOR bietet einen frühen Austritt aus einer FOR...NEXT-Schleife. EXIT FUNCTION provides an early exit from a defined function. EXIT SUB bietet ein vorzeitiges Verlassen einer definierten Funktion. Der alte Standard von EXIT allein (Beenden einer Do-Schleife) wird ebenfalls unterstützt.

FILES [fspec$]	Listet Dateien im aktuellen Verzeichnis auf der SD-Karte auf. 'fspec$' (falls angegeben) kann Suchplatzhalter enthalten. Fragezeichen (?) entsprechen einem beliebigen Zeichen und ein Sternchen (*) entspricht einer beliebigen Anzahl von Zeichen. Wenn weggelassen, werden alle Dateien aufgelistet. Beispielsweise: *.* Alle Einträge finden *.TXT Findet alle Einträge mit der Erweiterung TXT E*.* Finde alle Einträge, die mit E beginnen X?X.* Finde alle Dateinamen mit drei Buchstaben, die mit X beginnen und enden
FLASH FLASH LIST n FLASH ERASE n FLASH ERASE ALL FLASH SAVE n FLASH LOAD n FLASH RUN n FLASH CHAIN n FLASH OVERWRITE n	Verwaltet die Speicherung von Programmen im Flash-Speicher. Bis zu sieben Programme können im Flash-Speicher gespeichert und bei Bedarf abgerufen werden. Beachten Sie, dass diese gespeicherten Programme bei einem Firmware-Upgrade gelöscht werden. Einer dieser Flash-Speicherorte kann automatisch geladen und ausgeführt werden, wenn Strom angelegt wird, indem der Befehl OPTION AUTORUN n verwendet wird. Im Folgenden ist „n“ eine Zahl von 1 bis 10. Listet das gesamte Programm auf, das in Steckplatz n gespeichert ist. Löschen Sie einen Flash-Programmplatz. Löschen Sie alle Flash-Programmplätze. Speichern Sie das aktuelle Programm am angegebenen Flash-Speicherort. Laden Sie ein Programm vom angegebenen Flash-Speicherort in den Programmspeicher. Führt das Programm in Flash-Speicherort n aus, löscht alle Variablen. Verändert den Programmspeicher nicht. Führt das Programm in Flash-Speicherort n aus, wobei alle Variablen intakt bleiben (ermöglicht ein Programm, das viel größer als das RAM ist). Verändert den Programmspeicher nicht. Löschen Sie einen Flash-Programmspeicherort und speichern Sie dann das aktuelle Programm an dem angegebenen Flash-Speicherort..
FONT [#]font-number, scaling	Skalierung Dies stellt die Standardschriftart für die Anzeige von Text auf einem LCD-Bildschirm ein. Schriftarten werden als Zahl angegeben. Zum Beispiel #2 (das # ist optional). Einzelheiten zu den verfügbaren Schriftarten finden Sie im Kapitel „Grundlegende Zeichenbefehle“. Die „Skalierung“ kann von 1 bis 15 reichen und multipliziert die Größe der Pixel, wodurch das angezeigte Zeichen entsprechend breiter und höher wird. Z.B. eine Skala von 2 verdoppelt die Höhe und Breite.
FOR counter = start TO finish [STEP increment]	Leitet eine FOR-NEXT-Schleife ein, wobei der 'counter' anfänglich auf 'start' gesetzt ist und in 'increment'-Schritten erhöht wird (Standard ist 1), bis 'counter' größer als 'finish' ist. Das „Inkrement“ kann eine Ganzzahl oder eine Fließkommazahl sein. Beachten Sie, dass die Verwendung einer Fließkomma-Bruchzahl für „Inkrement“ Rundungsfehler in „Zähler“ akkumulieren kann, was dazu führen kann, dass die Schleife vorzeitig oder spät beendet wird. 'increment' kann negativ sein, in diesem Fall sollte 'finish' kleiner als 'start' sein und die Schleife wird abwärts zählen. Siehe auch den NEXT-Befehl.
FUNCTION xxx (arg1 [,arg2, …]) [AS <type>} <statements> <statements>	FUNCTION Definiert eine aufrufbare Funktion. Dies ist dasselbe wie das Hinzufügen einer neuen Funktion zu MMBasic, während es Ihr Programm ausführt.

xxx = <return value>
END FUNCTION
	'xxx' ist der Funktionsname und muss den Vorgaben für die Benennung einer Variablen entsprechen. Der Typ der Funktion kann durch Verwendung eines Typ-Suffixes (z. B. xxx$) oder durch Angabe des Typs mit AS <Typ> am Ende der Funktionsdefinition angegeben werden. Beispielsweise:
	`FUNCTION xxx (arg1, arg2) AS STRING`
	'arg1', 'arg2' usw. sind die Argumente oder Parameter der Funktion (die Klammern sind immer erforderlich, auch wenn keine Argumente vorhanden sind). Ein Array wird mit leeren Klammern angegeben. dh arg3(). Der Typ des Arguments kann durch Verwendung eines Typsuffixes (z. B. arg1$) oder durch Angabe des Typs mit AS <Typ> (z. B. arg1 AS STRING) angegeben werden.
	Das Argument kann auch eine andere definierte Funktion oder dieselbe Funktion sein, wenn Rekursion verwendet werden soll (der Rekursionsstack ist auf 50 verschachtelte Aufrufe begrenzt).
	Um den Rückgabewert der Funktion festzulegen, weisen Sie den Wert dem Namen der Funktion zu. Beispielsweise:
	`FUNCTION SQUARE(a)`
	`  SQUARE = a * a`
	`END FUNCTION`
	Jede Definition muss eine END FUNCTION-Anweisung haben. Wenn dies erreicht ist, gibt die Funktion ihren Wert an den Ausdruck zurück, von dem sie aufgerufen wurde. Für ein vorzeitiges Verlassen kann der Befehl EXIT FUNCTION verwendet werden.
	Sie verwenden die Funktion, indem Sie ihren Namen und ihre Argumente in einem Programm genauso verwenden, wie Sie es mit einer normalen MMBasic-Funktion tun würden. Beispielsweise:
	`PRINT SQUARE(56.8)`
	Wenn die Funktion aufgerufen wird, wird jedes Argument im Aufrufer mit dem Argument in der Funktionsdefinition abgeglichen. Diese Argumente sind nur innerhalb der Funktion verfügbar.
	Funktionen können mit einer variablen Anzahl von Argumenten aufgerufen werden. Alle weggelassenen Argumente in der Liste der Funktion werden auf Null oder eine Nullzeichenfolge gesetzt.
	Argumente in der Aufruferliste, die eine Variable sind und den richtigen Typ haben werden per Referenz an die Funktion übergeben. Das bedeutet dass alle Änderungen am entsprechenden Argument in der Funktion auch in die Variable des Aufrufers kopiert werden und daher nach dem Beenden der Funktion auf sie zugegriffen werden kann. Arrays werden durch Angabe des Arraynamens mit leeren Klammern (z. B. arg()) übergeben und werden immer per Referenz übergeben und müssen vom richtigen Typ sein.
	Sie dürfen nicht mit Befehlen wie GOTO, GOSUB usw. in oder aus einer Funktion springen. Dies hat unberechenbare Effekte zur Folge einschließlich dem, Ihren Tag zu ruinieren.

GOTO target	Verzweigt die Programmausführung zum Ziel, das eine Zeilennummer oder ein Label sein kann. **Vorsicht, Programm kann unübersichtlich werden !**
GUI AREA #ref, startX, startY, width, height	Dies definiert einen unsichtbaren Bereich des Bildschirms, der berührungsempfindlich ist und Touchdown- und Touchup-Interrupts erzeugt. Es kann als Grundlage für die Erstellung benutzerdefinierter Steuerelemente verwendet werden, die vom Programm definiert und verwaltet werden. '#ref' ist die Referenznummer des Steuerelements. 'startX' und 'startY' sind die Koordinaten oben links während 'width' und 'height' die Abmessungen festlegen.
GUI BCOLOUR colour, #ref1 [, #ref2, #ref3, etc]	Dies ändert die Hintergrundfarbe der angegebenen Steuerelemente in „Farbe“, was ein RGB-Wert für die Zeichenfarbe ist. '#ref' ist die Referenznummer des Steuerelements.

GUI BARGAUGE #ref, StartX, StartY, width, height, FColour, BColour, min, max, c1, ta, c2, tb, c3, tc, c4	Definiert entweder eine horizontale oder vertikale analoge Balkenanzeige. '#ref' ist die Referenznummer des Steuerelements. „StartX“ und „StartY“ sind die oberen linken Koordinaten des Balkens, während „Breite“ die horizontale Breite und „Höhe“ die vertikale Höhe ist. Wenn die Breite kleiner als die Höhe ist, wird die Balkenanzeige vertikal gezeichnet, wobei das Diagramm von unten nach oben wächst. Andernfalls wird es horizontal gezeichnet, wobei der Graph von links nach rechts wächst. „Fcolour“ ist die Farbe, die für das Messgerät verwendet wird, während „Bcolour“ die Hintergrundfarbe ist. 'min' ist der Minimalwert des Messgeräts und 'max' ist der Maximalwert (beides Fließkommazahlen). Ein mehrfarbiges Messgerät kann mit „c1“ bis „c4“ für die Farben und „ta“ bis „tc“ für die Schwellenwerte erstellt werden, die verwendet werden, um zu bestimmen, wann sich die Farbe ändert. 'Breite', 'Höhe', 'FFarbe', 'BFarbe', 'Min' und 'Max' sind optional und werden standardmäßig auf die Werte gesetzt, die in der vorherigen Definition eines GUI-BARGAUGE verwendet wurden. 'c1', 'ta', 'c2', 'tb', 'c3', 'tc' und 'c4' sind optional und wenn nicht angegeben, verwendet das Messgerät weniger Farben. Wenn alle weggelassen werden, wird die Anzeige mit 'Fcolour' gezeichnet. Der Abschnitt Advanced Graphics enthält eine detailliertere Beschreibung.
GUI BUTTON #ref, caption$, startX, startY, width, height [, FColour] [,BColour]	Dies zeichnet eine momentane Schaltfläche, die ein quadratischer Schalter mit der Beschriftung auf der Vorderseite ist. Bei Berührung erscheint das sichtbare Bild der Schaltfläche gedrückt und der Wert des Steuerelements ist 1. Wenn die Berührung entfernt wird, wird der Wert auf Null zurückgesetzt. #ref' ist die Referenznummer des Steuerelements. 'caption$' ist die Zeichenfolge, die auf der Vorderseite der Schaltfläche angezeigt werden soll. Es kann eine einzelne Zeichenfolge mit zwei Beschriftungen sein, die durch ein | getrennt sind Zeichen (z. B. "UP|DOWN"). Wenn die Taste oben ist, wird die erste Saite verwendet und wenn sie gedrückt wird, wird die zweite verwendet. 'startX' und 'startY' sind die Koordinaten oben links, während 'width' und 'height' die Abmessungen festlegen. ' FColor und 'BColor' sind RGB-Werte für die Vorder- und Hintergrundfarbe. 'with', 'height', FColour und 'BColour' sind optional und entsprechen standardmäßig denen die in vorherigen Steuerelementen verwendet oder mit dem COLOR-Befehl festgelegt wurden.

GUI CAPTION #ref, text$, startX, startY [,align$] [, FColour] [, BColour]	Dies zeichnet eine Textzeichenfolge auf dem Bildschirm. '#ref' ist die Referenznummer des Steuerelements. 'text$' ist die anzuzeigende Zeichenkette. 'startX' und 'startY' sind die Koordinaten oben links. 'align$' besteht aus null bis drei Zeichen (ein Zeichenfolgenausdruck oder eine Variable ist ebenfalls zulässig), wobei der erste Buchstabe die horizontale Ausrichtung um X herum darstellt und L, C oder R für LEFT, CENTER, RIGHT sein kann und der zweite Buchstabe die vertikale ist Ausrichtung um Y und kann T, M oder B für TOP, MIDDLE, BOTTOM sein. Ein drittes Zeichen kann in der Zeichenfolge verwendet werden, um die Drehung des Textes anzugeben. Dies kann 'N' für normale Ausrichtung sein, 'V' für vertikalen Text, wobei jedes Zeichen unter dem vorherigen von oben nach unten verläuft, 'I' der Text wird invertiert (dh auf dem Kopf stehend), 'U' der Text wird gedreht gegen den Uhrzeigersinn um 90º und 'D' wird der Text um 90º im Uhrzeigersinn gedreht. Die Standardausrichtung ist links/oben ohne Drehung. 'FColor und 'BColor' sind RGB-Werte für die Vorder- und Hintergrundfarbe. Auf einem Display, das transparenten Text unterstützt, kann BColor -1 sein, was bedeutet, dass der Hintergrund durch die Lücken in den Zeichen sichtbar ist. FColor und 'BColor' sind optional und standardmäßig auf die mit dem COLOR-Befehl gesetzten Farben eingestellt. GUI CHECKBOX #ref, caption$, startX, startY [, size] [, color] Dies zeichnet ein Kontrollkästchen, das ein kleines Kästchen mit einer Beschriftung ist. Bei Berührung wird ein X in das Kästchen gezeichnet, um anzuzeigen, dass diese Option ausgewählt wurde, und der Wert des Steuerelements wird auf 1 gesetzt. Bei einer zweiten Berührung wird das Häkchen entfernt und der Wert des Steuerelements ist Null. '#ref' ist die Referenznummer des Steuerelements. Der String 'caption$' wird rechts vom Steuerelement gezeichnet, wobei die Farben verwendet werden, die durch den COLOR-Befehl festgelegt wurden.
GUI CHECKBOX #ref, caption$, startX, startY [, size] [, colour]	Dies zeichnet ein Kontrollkästchen, das ein kleines Kästchen mit einer Beschriftung ist. Bei Berührung wird ein X in das Kästchen gezeichnet, um anzuzeigen, dass diese Option ausgewählt wurde, und der Wert des Steuerelements wird auf 1 gesetzt. Bei einer zweiten Berührung wird das Häkchen entfernt und der Wert des Steuerelements ist Null. '#ref' ist die Referenznummer des Steuerelements. Der String 'caption$' wird rechts vom Steuerelement gezeichnet, wobei die Farben verwendet werden, die durch den COLOR-Befehl festgelegt wurden. 'startX' und 'startY' sind die Koordinaten oben links, während 'size' die Höhe und Breite festlegt (das Bix ist quadratisch). 'Farbe' ist ein RGB-Wert für die Zeichenfarbe. „Größe“ und „Farbe“ sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden.
GUI DELETE #ref1 [,#ref2, #ref3, etc] oder GUI DELETE ALL	Dies löscht die Steuerelemente in der Liste. Dazu gehört das Entfernen des Bildes des Steuerelements vom Bildschirm unter Verwendung der aktuellen Hintergrundfarbe und das Freigeben des vom Steuerelement verwendeten Speichers. '#ref' ist die Referenznummer des Steuerelements. Als Argument kann das Schlüsselwort ALL verwendet werden, das alle Steuerelemente löscht.
GUI DISABLE #ref1 [,#ref2, #ref3, etc] oder GUI DISABLE ALL	Dies deaktiviert die Steuerelemente in der Liste. Deaktivierte Steuerelemente reagieren nicht auf Berührungen und werden abgeblendet angezeigt. '#ref' ist die Referenznummer des Steuerelements. Das Schlüsselwort ALL kann als Argument verwendet werden, wodurch alle Steuerelemente deaktiviert werden. GUI ENABLE kann verwendet werden, um die Steuerung wiederherzustellen.

GUI DISPLAYBOX #ref, startX, startY, width, height, FColour, BColour	Dadurch wird eine Box mit abgerundeten Ecken gezeichnet, die zur Anzeige einer Zeichenfolge verwendet werden kann '#ref' ist die Referenznummer des Steuerelements. 'startX' und 'startY' sind die Koordinaten oben links, während 'width' und 'height' die Abmessungen festlegen. ' FColor und 'BColor' sind RGB-Werte für die Vorder- und Hintergrundfarbe. 'Breite', 'Höhe', FFarbe und 'BFarbe' sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. Mit dem Befehl CtrlVal(r) = kann ein beliebiger Text im Feld angezeigt werden. Dies ist nützlich, um Text, Zahlen und Nachrichten anzuzeigen. Diese Steuerung reagiert nicht auf Berührungen.
GUI ENABLE #ref1 [,#ref2, #ref3, etc] or GUI ENABLE ALL	Dies macht die Effekte von GUI DISABLE rückgängig und stellt die Steuerung(en) wieder in den normalen Betrieb. '#ref' ist die Referenznummer des Steuerelements. Das Schlüsselwort ALL kann als Argument verwendet werden, wodurch alle Steuerelemente deaktiviert werden.
GUI FCOLOUR colour, #ref1 [, #ref2, #ref3, etc]	Dies ändert die Vordergrundfarbe der angegebenen Steuerelemente in „Farbe“, was ein RGB-Wert für die Zeichnungsfarbe ist.
GUI FRAME #ref, caption$, startX, startY, width, height, colour	Dies zeichnet einen Rahmen, der ein Kästchen mit runden Ecken und einer Beschriftung ist. '#ref' ist die Referenznummer des Steuerelements. 'caption$' ist eine Zeichenfolge, die als Beschriftung angezeigt wird. 'startX' und 'startY' sind die Koordinaten oben links, während 'width' und 'height' die Abmessungen festlegen. 'Farbe' ist ein RGB-Wert für die Zeichenfarbe. „Breite“, „Höhe“ und „Farbe“ sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. Ein Rahmen ist nützlich, wenn eine Gruppe von Steuerelementen visuell zusammengeführt werden muss. Es wird auch verwendet, um eine Gruppe von Optionsfeldern einzuschließen, und MMBasic sorgt dafür, dass die vom Rahmen umgebenen Optionsfelder exklusiv sind. d.h. wenn eine Optionsschaltfläche ausgewählt wird, wird jede andere Schaltfläche, die ausgewählt war und sich innerhalb des Rahmens befindet, automatisch deselektiert. Ein Rahmen reagiert nicht auf Berührung.
GUI FORMATBOX #ref, Format, startX, startY, width, height, FColour, BColour	BColor Dadurch wird ein Feld mit abgerundeten Ecken gezeichnet, das verwendet werden kann, um eine virtuelle Tastatur zur Eingabe von Daten in einem bestimmten Format zu erstellen. '#ref' ist die Referenznummer des Steuerelements. 'startX' und 'startY' sind die Koordinaten oben links, während 'width' und 'height' die Abmessungen festlegen. ' FColor und 'BColor' sind RGB-Werte für die Vorder- und Hintergrundfarbe. 'Breite', 'Höhe', FFarbe und 'BFarbe' sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. Das Argument 'Format' gibt das Format des Eintrags wie folgt an: DATE1 Datum im UK/Aust/NZ-Format (TT/MM/JJ) DATE2 Datum im US-Format (MM/TT/JJ) DATE3 Datum im internationalen Format (JJJJ/MM/TT) TIME1 Uhrzeit im 24-Stunden-Format (hh:mm) TIME2 Zeit im 24-Stunden-Format mit Sekunden (hh:mm:ss) TIME3 Zeit im 12-Stunden-Format (hh:mm AM/PM) TIME4 Zeit im 12-Stunden-Format mit Sekunden (hh:mm:ss AM/PM) DATETIME1 Datum (UK fmt) und Zeit (12 Stunden) (dd/mm/yy hh:mm AM/PM)

	DATETIME2 Datum (UK fmt) und Uhrzeit (24 Stunden) (tt/mm/jj hh:mm) DATETIME3 Datum (USA fmt) und Uhrzeit (12 Stunden) (mm/tt/jj hh:mm AM/PM)DATETIME4 Date (USA fmt) and time (24 hour) (mm/dd/yy hh:mm) LAT1 Breitengrad in Grad, Minuten und Sekunden (dd° mm' ss" N/S) LAT2 Breitengrad mit Sekunden auf eine Dezimalstelle (dd° mm' ss.s" N/S) LONG1 Längengrad in Grad, Minuten und Sekunden (ddd° mm' ss" O/W) LONG2 Längengrad Sekunden mit einer Dezimalstelle (ddd° mm' ss.s" O/W) ANGLE1 Winkel in Grad und Minuten (ddd° mm') Zum Beispiel dieser Befehl: GUI FORMATBOX #1, LAT1, 50, 50, 300, 50 würde ein Formatfeld erstellen, das die Eingabe von Breitengraden im Format dd° mm' ss" N/S akzeptieren würde. Der Wert von CtrlVal(#1) wäre eine Zeichenfolge, die die Zahlen und Trennzeichen enthält. Zum Beispiel eine Eingabe von 17 Grad, 32 Minuten und 1 Sekunde Süd würde die Saite 17° 32' 01" S ergeben. MMBasic versucht, das virtuelle Tastenfeld auf dem Bildschirm so zu positionieren, dass das Formatfeld, das es angezeigt hat, nicht verdeckt wird. Eine Stift-unten-Unterbrechung wird erzeugt, kurz bevor das Tastenfeld eingesetzt wird, und eine Taste-oben-Unterbrechung wird erzeugt, wenn die Eingabe abgeschlossen ist und das Tastenfeld ausgeblendet ist.
GUI FORMATBOX ACTIVATE #ref	Dies bewirkt, dass die virtuelle Tastatur für das Steuerelement „#ref“ unter Programmsteuerung angezeigt wird, ohne dass das Steuerelement berührt wird. Es ist dasselbe, als ob der Benutzer das Steuerelement berührt, außer dass die Berührungsunterbrechung nicht erzeugt wird.
GUI FORMATBOX CANCEL	Dies schließt eine virtuelle Tastatur, wenn sie auf dem Bildschirm angezeigt wird. Es ist dasselbe, als ob der Benutzer die Löschtaste berührt, außer dass die Berührungsunterbrechung nicht erzeugt wird. Wenn keine Tastatur angezeigt wird, hat dieser Befehl keine Auswirkung.
GUI GAUGE #ref, StartX, StartY, Radius, FColour, BColour, min, max, nbrdec, units$, c1, ta, c2, tb, c3, tc, c4	Definieren Sie ein grafisches kreisförmiges analoges Messgerät mit einer digitalen Anzeige in der Mitte. '#ref' ist die Referenznummer des Steuerelements. 'StartX' und 'StartY' sind die Koordinaten des Mittelpunkts der Lehre, 'Radius' ist der Abstand vom Mittelpunkt zum äußeren Rand. 'min' ist der Minimalwert des Messgeräts und 'max' ist der Maximalwert (beides Fließkommazahlen). 'nbrdec' gibt die Anzahl der Dezimalstellen an, die beim Zeichnen des digitalen Werts in der Mitte des Messgeräts verwendet werden sollen. Darunter wird 'units$' angezeigt. „Fcolour“ ist die Farbe, die für das Messgerät verwendet wird, während „Bcolour“ die Hintergrundfarbe ist. Ein mehrfarbiges Messgerät kann mit „c1“ bis „c4“ für die Farben und „ta“ bis „tc“ für die Schwellenwerte erstellt werden, die verwendet werden, um zu bestimmen, wann sich die Farbe ändert. Wenn Farben und Schwellenwerte angegeben sind, wird der Hintergrund des Messgeräts mit einer matten Version der Farbe auf dieser Ebene gezeichnet. Auch der digitale Wert ändert sich in die Farbe, die durch den aktuellen Wert angegeben ist. 'Radius', 'FColour', 'BColor', 'min', 'max', 'nbrdec' und 'units$' sind optional und werden standardmäßig auf die Werte gesetzt, die in der vorherigen Definition eines GUI GAUGE verwendet wurden. 'c1', 'ta', 'c2', 'tb', 'c3', 'tc' und 'c4' sind optional und wenn nicht angegeben, verwendet das Messgerät weniger Farben. Wenn alle weggelassen werden, wird die Anzeige mit 'Fcolour' gezeichnet. Der Abschnitt Advanced Graphics enthält eine detailliertere Beschreibung.

GUI HIDE #ref1 [,#ref2, #ref3, etc] oder GUI HIDE ALL	Dies blendet die Steuerelemente in der Liste aus. Ausgeblendete Bedienelemente reagieren nicht auf Berührungen und sind nicht sichtbar. '#ref' ist die Referenznummer des Steuerelements. Das Schlüsselwort ALL kann als Argument verwendet werden, wodurch alle Steuerelemente ausgeblendet werden. GUI SHOW kann verwendet werden, um die Steuerung wiederherzustellen.
GUI INTERRUPT down [, up]	Dieser Befehl richtet einen Interrupt ein der bei einer Berührung auf dem LCD-Panel und optional beim Loslassen der Berührung ausgelöst wird. 'down' ist die Unterroutine, die aufgerufen werden soll, wenn ein Aufsetzen festgestellt wurde. 'up' ist die Subroutine, die aufgerufen wird, wenn die Berührung vom Bildschirm aufgehoben wurde ('up' und 'down' können bei Bedarf auf dieselbe Subroutine zeigen). Die Angabe der Zahl Null (einstellige Zahl) als Argument löscht diese beiden Interrupts: `GUI INTERRUPT 0`
GUI LED #ref, caption$, centerX, centerY, radius, colour	Dadurch wird eine Anzeigeleuchte gezeichnet, die wie eine auf einer Platte montierte LED aussieht. Eine LED reagiert nicht auf Berührung. '#ref' ist die Referenznummer des Steuerelements. Der String 'caption$' wird rechts vom Steuerelement gezeichnet, wobei die Farben verwendet werden, die durch den COLOR-Befehl festgelegt wurden. 'centerX' und 'centerY' sind die Koordinaten des Zentrums der LED und 'radius' ist der Radius der LED. 'Farbe' ist ein RGB-Wert für die Zeichenfarbe. „Radius“ und „Farbe“ sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. Wenn der Wert einer LED auf den Wert eins gesetzt wird, leuchtet sie und wenn sie auf Null gesetzt wird, ist sie aus (eine stumpfe Version ihres Farbattributs). Die LED kann dazu gebracht werden, ein- und dann auszuschalten, indem der Wert der LED auf eine Zahl größer als eins eingestellt wird, was die Zeit in Millisekunden ist, die sie eingeschaltet bleiben soll. Die Farbe kann mit dem GUI-Befehl FCOLOUR geändert werden.
GUI NUMBERBOX #ref, startX, startY, width, height, FColour, BColour	Dadurch wird ein Kästchen mit abgerundeten Ecken gezeichnet, das verwendet werden kann, um einen virtuellen Ziffernblock für die Dateneingabe zu erstellen. '#ref' ist die Referenznummer des Steuerelements. 'startX' und 'startY' sind die Koordinaten oben links, während 'width' und 'height' die Abmessungen festlegen. ' FColor und 'BColor' sind RGB-Werte für die Vorder- und Hintergrundfarbe. 'Breite', 'Höhe', FFarbe und 'BFarbe' sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. Wenn das Feld berührt wird, erscheint eine numerische Tastatur auf dem Bildschirm. Mit dieser virtuellen Tastatur kann eine beliebige Zahl in das Feld eingegeben werden, einschließlich einer Fließkommazahl im Exponentialformat. Die neue Nummer ersetzt die Nummer, die zuvor im Feld stand. Der Wert des Steuerelements kann auf eine Literalzeichenfolge (kein Ausdruck) gesetzt werden, die mit zwei Hash-Zeichen beginnt. Beispielsweise: CtrlVal(nnn) = "##Nummer eingeben" und in diesem Fall wird die Zeichenfolge (ohne die führenden zwei Hash-Zeichen) in der Box mit reduzierter Helligkeit angezeigt. Dies kann verwendet werden, um dem Benutzer einen Hinweis darauf zu geben, was eingegeben werden sollte (sogenannter "Ghost-Text"). Das Lesen des Werts

	des Steuerelements, das Geistertext anzeigt, gibt Null zurück. Wenn das Steuerelement normal verwendet wird, verschwindet der Geistertext. MMBasic versucht, die virtuelle Tastatur so auf dem Bildschirm zu positionieren, dass das Zahlenfeld, das sie angezeigt hat, nicht verdeckt wird. Eine Stift-unten-Unterbrechung wird generiert, kurz bevor das Tastenfeld aktiviert wird, und eine Taste-oben-Unterbrechung wird generiert, wenn die Eingabetaste berührt wird und das Tastenfeld ausgeblendet wird. Auch wenn die Eingabetaste berührt wird, wird die eingegebene Zahl als Zahl ausgewertet und das NUMBERBOX-Steuerelement neu gezeichnet, um diese Zahl anzuzeigen.
GUI NUMBERBOX ACTIVATE #ref	Dies bewirkt, dass die virtuelle Tastatur für das Bedienelement „#ref“ unter Programmsteuerung angezeigt wird, ohne dass das Bedienelement berührt wird. Es ist dasselbe, als ob der Benutzer das Steuerelement berührt, außer dass die Berührungsunterbrechung nicht erzeugt wird.
GUI NUMBERBOX CANCEL	Dies schließt eine virtuelle Tastatur, wenn sie auf dem Bildschirm angezeigt wird. Es ist dasselbe, als ob der Benutzer die Löschtaste berührt, außer dass die Berührungsunterbrechung nicht erzeugt wird. Wenn keine Tastatur angezeigt wird, hat dieser Befehl keine Auswirkung.
GUI RADIO #ref, caption$, centerX, centerY, radius, colour	Dies zeichnet ein Optionsfeld mit einer Beschriftung. '#ref' ist die Referenznummer des Steuerelements. Der String 'caption$' wird rechts vom Steuerelement gezeichnet, wobei die Farben verwendet werden, die durch den COLOR-Befehl festgelegt wurden. 'centerX' und 'centerY' sind die Koordinaten der Schaltflächenmitte und 'radius' ist der Radius der Schaltfläche. 'Farbe' ist ein RGB-Wert für die Zeichenfarbe. „Radius“ und „Farbe“ sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. Bei Berührung leuchtet die Mitte der Schaltfläche auf, um anzuzeigen, dass diese Option ausgewählt wurde, und der Wert des Steuerelements ist 1. Wenn eine andere Optionsschaltfläche ausgewählt wird, wird die Markierung auf dieser Schaltfläche entfernt und ihr Wert ist Null. Optionsfelder werden gruppiert, wenn sie von einem Rahmen umgeben sind, und wenn ein Feld in der Gruppe ausgewählt wird, werden alle anderen in der Gruppe deselektiert. Wenn kein Rahmen verwendet wird, werden alle Schaltflächen auf dem Bildschirm gruppiert.
GUI REDRAW #ref1 [,#ref2, #ref3, etc] oder GUI REDRAW ALL	Dies zeichnet die Steuerelemente auf dem Bildschirm neu. Es ist nützlich, wenn das Bildschirmbild irgendwie beschädigt wurde. '#ref' ist die Referenznummer des Steuerelements. Das Schlüsselwort ALL kann als Argument verwendet werden, wodurch zuerst der Bildschirm gelöscht und dann alle Steuerelemente neu gezeichnet werden. Dies ist nützlich, wenn der gesamte Bildschirm aktualisiert werden muss.
GUI SETUP #n	Dadurch werden alle neu erstellten Steuerelemente der Seite '#n' zugewiesen. Dieser Befehl kann so oft wie nötig verwendet werden, während GUI-Steuerelemente definiert werden. Der Standardwert, wenn ein Programm gestartet wird, ist GUI SETUP 1. Siehe auch den GUI PAGE-Befehl.
GUI SHOW #ref1 [,#ref2, #ref3, etc] oder GUI SHOW ALL	ALL Dies macht die Effekte von GUI HIDE rückgängig und stellt die Steuerung(en) wieder her, damit sie sichtbar und normal bedienbar sind. '#ref' ist die Referenznummer des Steuerelements. Das Schlüsselwort ALL kann als Argument verwendet werden, wodurch alle Steuerelemente deaktiviert werden.
GUI SPINBOX #ref, startX, startY, width, height, FColour, BColour, Step, Minimum, Maximum	Dadurch wird ein Kästchen mit Oben/Unten-Symbolen an beiden Enden gezeichnet. Wenn diese Symbole berührt werden, wird die Zahl im Feld erhöht oder verringert. Wenn Sie die Hoch-/Runter-Symbole gedrückt halten, wird der Schritt schnell wiederholt.

	'#ref' ist die Referenznummer des Steuerelements. 'startX' und 'startY' sind die Koordinaten oben links, während 'width' und 'height' die Abmessungen festlegen. ' FColor und 'BColor' sind RGB-Werte für die Vorder- und Hintergrundfarbe. 'Breite', 'Höhe', FFarbe und 'BFarbe' sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. 'Step' legt den Betrag fest, um den die Zahl bei jeder Berührung erhöht/verringert wird. „Minimum“ und „Maximum“ begrenzen die Anzahl, die eingegeben werden kann. Alle drei Parameter können Fließkommazahlen sein und sind optional. Der Standardwert für „Schritt“ ist 1 und „Minimum“ und „Maximum“, wenn sie weggelassen werden, sind standardmäßig unbegrenzt..
GUI SWITCH #ref, caption$, startX, startY, width, height, FColour, BColour	Dies zeichnet einen Verriegelungsschalter, der ein quadratischer Schalter ist, der bei Berührung einrastet. '#ref' ist die Referenznummer des Steuerelements. 'caption$' ist eine Zeichenfolge, die als Beschriftung auf der Vorderseite des Schalters angezeigt wird. 'startX' und 'startY' sind die Koordinaten oben links, während 'width' und 'height' die Abmessungen festlegen. ' FColor und 'BColor' sind RGB-Werte für die Vorder- und Hintergrundfarbe. 'Breite', 'Höhe', FFarbe und 'BFarbe' sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. Bei Berührung erscheint das sichtbare Bild der Schaltfläche gedrückt und der Wert des Steuerelements ist 1. Bei einer zweiten Berührung wird der Schalter losgelassen und der Wert wird auf Null zurückgesetzt. Die Beschriftung kann aus zwei Beschriftungen bestehen, die durch ein | getrennt sind Zeichen (z. B. "ON|OFF"). Wenn dies verwendet wird, erscheint der Schalter als Kippschalter, wobei jede Hälfte der Beschriftung verwendet wird, um jede Hälfte des Kippschalters zu beschriften.

GUI TEXTBOX #ref, startX, startY, width, height, FColour, BColour	Dadurch wird ein Feld mit abgerundeten Ecken gezeichnet, das verwendet werden kann, um eine virtuelle Tastatur für die Dateneingabe zu erstellen '#ref' ist die Referenznummer des Steuerelements. 'startX' und 'startY' sind die Koordinaten oben links, während 'width' und 'height' die Abmessungen festlegen. ' FColor und 'BColor' sind RGB-Werte für die Vorder- und Hintergrundfarbe. 'Breite', 'Höhe', FFarbe und 'BFarbe' sind optional und entsprechen standardmäßig denen, die in vorherigen Steuerelementen verwendet wurden. Auf einem Display, das transparenten Text unterstützt, kann BColor -1 sein, was bedeutet, dass der Hintergrund durch die Lücken in den Zeichen sichtbar ist. Wenn das Feld berührt wird, erscheint eine QWERTZ-Tastatur auf dem Bildschirm. Mit dieser virtuellen Tastatur kann ein beliebiger Text in das Feld eingegeben werden, einschließlich Groß-/Kleinbuchstaben, Zahlen und alle anderen Zeichen des ASCII-Zeichensatzes. Der neue Text ersetzt jeden Text, der sich zuvor im Feld befand. Der Wert des Steuerelements kann auf eine Zeichenfolge festgelegt werden, die mit zwei Hash-Zeichen beginnt. Beispielsweise: CtrlVal(nnn) = "##Dateinamen eingeben" und in diesem Fall wird die Zeichenfolge (ohne die führenden zwei Hash-Zeichen) in der Box mit reduzierter Helligkeit angezeigt. Dies kann verwendet werden, um dem Benutzer einen Hinweis darauf zu geben, was eingegeben werden sollte (sogenannter "Ghost-Text"). Das Lesen des Werts des Steuerelements, das Geistertext anzeigt, gibt eine leere Zeichenfolge zurück. Wenn das Steuerelement normal verwendet wird, verschwindet der Geistertext. MMBasic versucht, die virtuelle Tastatur auf dem Bildschirm so zu positionieren, dass das Textfeld, das sie angezeigt hat, nicht verdeckt wird. Eine Stift-unten-Unterbrechung wird generiert, kurz bevor die Tastatur eingesetzt wird, und eine Taste-oben-Unterbrechung wird generiert, wenn die Eingabetaste berührt wird und die Tastatur ausgeblendet wird.
GUI TEXTBOX ACTIVATE #ref	Dies bewirkt, dass die virtuelle Tastatur für das Steuerelement „#ref“ unter Programmsteuerung angezeigt wird, ohne dass das Steuerelement berührt wird. Es ist dasselbe, als ob der Benutzer das Steuerelement berührt, außer dass die Berührungsunterbrechung nicht erzeugt wird.
GUI TEXTBOX CANCEL	Dies schließt eine virtuelle Tastatur, wenn sie auf dem Bildschirm angezeigt wird. Es ist dasselbe, als ob der Benutzer die Löschtaste berührt, außer dass die Berührungsunterbrechung nicht erzeugt wird. Wenn keine Tastatur angezeigt wird, wird dieser Befehl nichts tun.
GUI BITMAP x, y, bits [, width] [, height] [, scale] [, c] [, bc]	Zeigt die Bits in einer Bitmap auf einem LCD-Panel an, beginnend bei 'x' und 'y' auf einem angeschlossenen LCD Display. 'Höhe' und 'Breite' sind die Abmessungen der Bitmap, wie sie auf dem LCD-Panel angezeigt werden, und sind standardmäßig 8x8. 'scale' ist optional und wird standardmäßig durch den FONT-Befehl festgelegt. 'c' ist die Zeichenfarbe und 'bc' ist die Hintergrundfarbe. Sie sind optional und standardmäßig auf die aktuellen Vorder- und Hintergrundfarben eingestellt. Die Bitmap kann eine Ganzzahl oder eine String-Variable oder -Konstante sein und wird mit dem ersten Byte als den ersten Bits der obersten Zeile gezeichnet (Bit 7 zuerst, dann Bit 6 usw.), gefolgt vom nächsten Byte usw. Wenn die oberste Zeile gefüllt wurde, beginnt die nächste Zeile der angezeigten Bitmap mit dem nächsten Bit in der Ganzzahl oder Zeichenfolge. Siehe den Abschnitt Grundlegende Zeichenbefehle für eine Definition der Farben und Grafikkoordinaten.

GUI CALIBRATE	Dieser Befehl wird verwendet, um die Berührungsfunktion auf einem LCD-Bildschirm zu kalibrieren. Es zeigt eine Reihe von Zielen auf dem Bildschirm an und wartet darauf, dass jedes genau berührt wird.
GUI RESET LCDPANEL	Initialisiert das konfigurierte LCD-Panel neu. Die Initialisierung erfolgt automatisch beim Start des PicoMite, aber unter bestimmten Umständen kann es erforderlich sein, die Stromversorgung des LCD-Panels zu unterbrechen (z. B. um Batteriestrom zu sparen), und dieser Befehl kann dann verwendet werden, um das Display neu zu initialisieren.
GUI TEST LCDPANEL or GUI TEST TOUCH	Testet die Anzeige- oder Berührungsfunktion auf einem LCD-Panel. Mit GUI TEST LCDPANEL wird eine animierte Darstellung von Farbkreisen schnell übereinander gezeichnet. Mit GUI TEST TOUCH wird der Bildschirm leer und wartet auf eine Berührung, die dazu führt, dass ein weißer Punkt auf dem Display platziert wird, der die Berührungsposition auf dem Bildschirm markiert. Jedes an der Konsole eingegebene Zeichen beendet den Test.
I2C OPEN speed, timeout	Aktiviert das erste I2C-Modul im Master-Modus. „Geschwindigkeit“ ist die zu verwendende Taktgeschwindigkeit (in KHz) und muss 100, 400 oder 1000 sein. „timeout“ ist ein Wert in Millisekunden, nach dem die Sende- und Empfangsbefehle des Masters unterbrochen werden, wenn sie nicht abgeschlossen sind. Der Mindestwert ist 100. Ein Wert von Null deaktiviert das Timeout (obwohl dies nicht empfohlen wird).
I2C WRITE addr, option, sendlen, senddata [,sendata ....]	Daten an das I2C-Slave-Gerät senden. „addr“ ist die I2C-Adresse des Slaves. „Option“ kann 0 für den normalen Betrieb oder 1 sein, um die Kontrolle über den Bus nach dem Befehl zu behalten (nach Abschluss des Befehls wird keine Stoppbedingung gesendet). „sendlen“ ist die Anzahl der zu sendenden Bytes. „senddata“ sind die zu sendenden Daten – diese können auf verschiedene Arten angegeben werden (alle gesendeten Werte liegen zwischen 0 und 255). Anmerkungen: • Die Daten können als einzelne Bytes auf der Kommandozeile übergeben werden. Beispiel: I2C WRITE &H6F, 0, 3, &H23, &H43, &H25 • Die Daten können sich in einem eindimensionalen Array befinden, das mit leeren Klammern (d. h. ohne Dimensionen) angegeben wird. 'sendlen' Bytes des Arrays werden beginnend mit dem ersten Element gesendet. Beispiel: I2C WRITE &H6F, 0, 3, ARRAY() Die Daten können eine String-Variable sein (keine Konstante). Beispiel: I2C WRITE &H6F, 0, 3, STRING$
I2C READ addr, option, rcvlen, rcvbuf	Ruft Daten vom I2C-Slave-Gerät ab. „addr“ ist die I2C-Adresse des Slaves. „Option“ kann 0 für den normalen Betrieb oder 1 sein, um die Kontrolle über den Bus nach dem Befehl zu behalten (nach Abschluss des Befehls wird keine Stoppbedingung gesendet). „rcvlen“ ist die Anzahl der zu empfangenden Bytes. ‘rcvbuf’ ist die Variable oder das Array, das zum Speichern der empfangenen Daten verwendet wird – dies kann sein: • Eine String-Variable. Bytes werden als aufeinanderfolgende Zeichen gespeichert. • Ein eindimensionales Array von Zahlen, die mit leeren Klammern angegeben werden. Empfangene Bytes werden in aufeinanderfolgenden Elementen des Arrays gespeichert, beginnend mit dem ersten.

	Beispiel: I2C READ &H6F, 0, 3, ARRAY() Eine normale numerische Variable (in diesem Fall muss rcvlen 1 sein)
I2C CLOSE	Deaktiviert das Master-I2C-Modul. Dieser Befehl sendet auch ein Stoppen, wenn der Bus noch gehalten wird.
I2C2	Derselbe Befehlssatz wie für I2C (oben), aber für den zweiten I2C-Kanal.
IF expr THEN stmt [: stmt] oder IF expr THEN stmt ELSE stmt	Wertet den Ausdruck „expr“ aus und führt die Anweisung nach dem Schlüsselwort THEN aus, wenn es wahr ist, oder springt zur nächsten Zeile, wenn es falsch ist. Wenn es mehr Anweisungen in der Zeile gibt (getrennt durch Doppelpunkte (:), werden sie ebenfalls ausgeführt, wenn wahr, oder übersprungen, wenn falsch. Das Schlüsselwort ELSE ist optional, und falls vorhanden, wird/werden die folgende(n) Anweisung(en) ausgeführt, wenn 'expr' aufgelöst wird falsch sein. Das Konstrukt „DANN-Anweisung“ kann auch ersetzt werden durch: GOTO Leinennummer | Etikett'. Diese Art von IF-Anweisung befindet sich alles in einer Zeile.

IF expression THEN
<statements>
[ELSEIF expression THEN
<statements>]
[ELSE
<statements>]
ENDIF
	Mehrzeilige IF-Anweisung mit optionalen ELSE- und ELSEIF-Fällen und Endung mit ENDIF. Jede Komponente befindet sich in einer separaten Zeile.
	Wertet 'Ausdruck' aus und führt die Anweisung(en) nach THEN aus, wenn der Ausdruck wahr ist, oder optional die Anweisung(en) nach der ELSE-Anweisung, wenn er falsch ist. Die ELSEIF-Anweisung (falls vorhanden) wird ausgeführt, wenn die vorherige Bedingung falsch ist, und sie beginnt eine neue IF-Kette mit weiteren ELSE- und/oder ELSEIF-Anweisungen nach Bedarf.
	Ein ENDIF wird verwendet, um das mehrzeilige IF abzuschließen.

INPUT ["prompt$";] var1 [,var2 [, var3 [, etc]]]	Nimmt eine durch Kommas (,) getrennte Liste von Werten, die an der Konsole eingegeben werden, und weist sie einer sequentiellen Liste von Variablen zu. Beispiel: Der Befehl lautet: INPUT a, b, c Und auf der Tastatur wird folgendes eingegeben: 23, 87, 66 Dann ist a = 23 und b = 87 und c = 66 Die Liste der Variablen kann eine Mischung aus Float-, Integer- oder String-Variablen sein. Die an der Konsole eingegebenen Werte müssen dem Variablentyp entsprechen. Wenn ein einzelner Wert eingegeben wird, ist kein Komma erforderlich (dieser Wert darf jedoch kein Komma enthalten). ‘prompt$’ ist eine String-Konstante (keine Variable oder ein Ausdruck) und wenn angegeben, wird sie zuerst gedruckt. Normalerweise wird die Eingabeaufforderung mit einem Semikolon (;) abgeschlossen und in diesem Fall wird nach der Eingabeaufforderung ein Fragezeichen ausgegeben. Wenn die Eingabeaufforderung mit einem Komma (,) statt mit einem Semikolon (;) abgeschlossen wird, wird das Fragezeichen unterdrückt.
INPUT #nbr, list of variables	Liste der Variablen Wie oben, außer dass die Eingabe von einer seriellen Schnittstelle oder Datei gelesen wird, die zuvor für INPUT als „nbr“ geöffnet wurde. Siehe den OPEN-Befehl.
IR dev, key , int or IR CLOSE	Dekodiert Infrarot-Fernbedienungssignale von NEC oder Sony. Ein IR-Empfängermodul wird verwendet, um das IR-Licht zu erfassen und das Signal zu demodulieren. Es kann an jeden Pin angeschlossen werden, dieser Pin muss jedoch im Voraus mit dem Befehl konfiguriert werden: SETPIN n, IR Die Dekodierung des IR-Signals erfolgt im Hintergrund und das Programm wird nach diesem Befehl ohne Unterbrechung fortgesetzt. „dev“ und „key“ sollten numerische Variablen sein und ihre Werte werden aktualisiert, wenn

	ein neues Signal empfangen wird („dev“ ist der von der Fernbedienung übertragene Gerätecode und „key“ ist die gedrückte Taste). 'int' ist eine benutzerdefinierte Unterroutine, die aufgerufen wird, wenn ein neuer Tastendruck empfangen wird oder wenn die vorhandene Taste für die automatische Wiederholung gedrückt gehalten wird. In der Interrupt-Subroutine kann das Programm die Variablen 'dev' und 'key' untersuchen und entsprechende Maßnahmen ergreifen. Der IR CLOSE-Befehl beendet den IR-Decoder. Beachten Sie, dass für das NEC-Protokoll die Bits in „dev“ und „key“ vertauscht sind. Zum Beispiel sollte in „Schlüssel“ Bit 0 Bit 7 sein, Bit 1 Bit 6 usw. Dies hat keinen Einfluss auf die normale Verwendung, aber wenn Sie nach einem bestimmten Zahlencode suchen, der von einem Hersteller bereitgestellt wird, sollten Sie die Bits umkehren. Dies beschreibt, wie es geht: http://www.thebackshed.com/forum/forum_posts.asp?TID=8367 Weitere Informationen finden Sie im Abschnitt Spezielle Hardwaregeräte.
IR SEND pin, dev, key	Generiert ein 12-Bit-Infrarotsignal für das Sony Remote Control-Protokoll. 'pin' ist der zu verwendende I/O-Pin. Dies kann ein beliebiger I/O-Pin sein, der automatisch als Ausgang konfiguriert wird und mit einer Infrarot-LED verbunden werden sollte. Leerlauf ist niedrig, wobei hohe Pegel anzeigen, wann die LED eingeschaltet werden sollte. „dev“ ist das gesteuerte Gerät und eine Zahl von 0 bis 31, „key“ ist der simulierte Tastendruck und eine Zahl von 0 bis 127. Das IR-Signal wird mit etwa 38 kHz moduliert und das Senden des Signals dauert etwa 25 ms.
KEYPAD var, int, r1, r2, r3, r4, c1, c2, c3 [, c4] or KEYPAD CLOSE	Überwachung und Dekodierung von Tastendrücken auf einem 4x3- oder 4x4-Keypad. Die Überwachung der Tastatur erfolgt im Hintergrund und das Programm läuft nach diesem Befehl ohne Unterbrechung weiter. 'var' sollte eine numerische Variable sein und ihr Wert wird aktualisiert, wenn ein Tastendruck erkannt wird. 'int' ist eine benutzerdefinierte Unterroutine, die aufgerufen wird, wenn ein neuer Tastendruck empfangen wird. In der Interrupt-Subroutine kann das Programm die Variable 'var' untersuchen und entsprechende Maßnahmen ergreifen. r1, r2, r3 und r4 sind Stiftnummern, die für die vier Reihenverbindungen zum Tastenfeld verwendet werden, und c1, c2, c3 und c4 sind die Spaltenverbindungen. c4 ist optional und wird nur mit 4x4-Tastaturen verwendet. Dieser Befehl konfiguriert diese Pins automatisch nach Bedarf. Bei einem Tastendruck ist der 'var' zugewiesene Wert die Nummer einer numerischen Taste (z. B. '6' gibt 6 zurück) oder 10 für die *-Taste und 11 für die #-Taste. Auf 4x4-Tastaturen wird die Zahl 20 für A, 21 für B, 22 für C und 23 für D zurückgegeben. Der Befehl KEYPAD CLOSE beendet die Tastaturfunktion und versetzt den I/O-Pin in einen nicht konfigurierten Zustand. Weitere Informationen finden Sie im Abschnitt Spezielle Hardwaregeräte.
KILL file$	Löscht die durch „file$“ angegebene Datei. Wenn eine Erweiterung vorhanden ist, muss diese angegeben werden.
LET variable = expression	Weist der Variablen den Wert von 'Ausdruck' zu. LET wird automatisch angenommen, wenn eine Anweisung nicht mit einem Befehl beginnt. Beispielsweise: Var = 56

LINE x1, y1, x2, y2 [, LW [, C]]	Auf einem angeschlossenen LCD-Display zeichnet dieser Befehl eine Linie, die bei den Koordinaten „x1“ und „y1“ beginnt und bei „x2“ und „y2“ endet. „LW“ ist die Breite der Linie und gilt nur für horizontale oder vertikale Linien. Der Standardwert ist 1, wenn nichts angegeben ist oder wenn die Linie eine Diagonale ist. „C“ ist eine Ganzzahl, die die Farbe darstellt und standardmäßig die aktuelle Vordergrundfarbe ist. Alle Parameter können als Arrays ausgedrückt werden, und die Software zeichnet die Anzahl der Linien auf, die durch die Abmessungen des kleinsten Arrays bestimmt werden. „x1“, „y1“, „x2“ und „y2“ müssen alle Arrays oder alle einzelne Variablen/Konstanten sein, andernfalls wird ein Fehler generiert. 'lw' und 'c' können entweder Arrays oder einzelne Variablen/Konstanten sein.
LINE INPUT [prompt$,] string-variable$	Liest eine ganze Zeile aus der Konsoleneingabe in ‚string variable$‘ ein. ‘prompt$’ ist eine String-Konstante (keine Variable oder ein Ausdruck) und wenn angegeben, wird sie zuerst gedruckt. Im Gegensatz zu INPUT liest dieser Befehl eine ganze Zeile und stoppt nicht bei durch Kommas getrennten Datenelementen. Ein Fragezeichen wird nicht gedruckt, es sei denn, es ist Teil von ‚prompt$‘.
LINE INPUT #nbr, string-variable$	Wie oben, außer dass die Eingabe von einem seriellen Kommunikationsanschluss oder einer zuvor für INPUT als „nbr“ geöffneten Datei gelesen wird. Siehe den OPEN-Befehl.
LIST [fname$] oder LIST ALL [fname$]	Listet ein Programm auf der Konsole auf. LIST allein listet das Programm mit einer Pause bei jedem vollen Bildschirm auf. LIST ALL listet das Programm ohne Pausen auf. Dies ist nützlich, wenn Sie das Programm auf einen Terminalemulator auf einem PC übertragen möchten, der seinen Eingabestrom in einer Datei erfassen kann. Wenn der optionale „fname$“ angegeben wird, wird diese Datei auf der SD-Karte aufgelistet
LIST COMMANDS oder LIST FUNCTIONS	Listet alle gültigen Befehle oder Funktionen auf
LOAD file$ [,R]	Lädt ein Programm namens „file$“ von einer SD-Karte in den Programmspeicher. Wenn das optionale Suffix ‚R hinzugefügt wird, wird das Programm sofort ohne Nachfrage ausgeführt (in diesem Fall muss „file$“ eine String-Konstante sein). Wenn keine Erweiterung angegeben wird, wird dem Dateinamen „.BAS“ hinzugefügt.
LOAD IMAGE file$ [, x, y]	Lädt ein Bitmap-Bild von der SD-Karte und zeigt es auf dem LCD-Feld an. „file$“ ist der Name der Datei und „x“ und „y“ sind die Bildschirmkoordinaten für die obere linke Ecke des Bildes. Wenn die Koordinaten nicht angegeben sind, wird das Bild an der oberen linken Position auf dem Bildschirm gezeichnet. Wenn keine Erweiterung angegeben wird, wird „.BMP“ zum Dateinamen hinzugefügt. Alle Typen des BMP-Formats werden unterstützt, einschließlich Schwarzweiß- und Echtfarben-24-Bit-Bildern.
LOCAL variable [, variables] Siehe DIM für die vollständige Syntax.	Definiert eine Liste von Variablennamen als lokal für die Subroutine oder Funktion. Dieser Befehl verwendet genau die gleiche Syntax wie DIM und erstellt Variablen, die nur innerhalb der Subroutine oder Funktion sichtbar sind. Sie werden automatisch verworfen, wenn das Unterprogramm oder die Funktion beendet wird.

LONGSTRING	Die LONGSTRING-Befehle ermöglichen die Bearbeitung von Zeichenfolgen, die länger als die normale MMBasic-Grenze von 255 Zeichen sind. Variablen zum Halten langer Zeichenfolgen müssen als eindimensionale Integer-Arrays definiert werden, wobei die Anzahl der Elemente auf die Anzahl der Zeichen festgelegt ist, die für die maximale Zeichenfolgenlänge geteilt durch acht erforderlich ist. Der Grund für die Division durch acht liegt darin, dass jede ganze Zahl in einem MMBasic-Array acht Bytes belegt. Beachten Sie, dass die Routinen für lange Zeichenfolgen nicht auf Überlauf in der Länge der Zeichenfolgen prüfen. Wenn versucht wird, eine Zeichenfolge zu erstellen, die länger als die Größe einer langen Zeichenfolgenvariablen ist, ist das Ergebnis undefiniert.
LONGSTRING APPEND array%(), string$	Fügt einen normalen MMBasic-String an eine lange String-Variable an. array%() ist eine lange String-Variable, während string$ ein normaler MMBasic-String-Ausdruck ist.
LONGSTRING CLEAR array%()	Löscht die lange String-Variable array%(). d.h. der String wird geleert.
LONGSTRING COPY dest%(), src%()	Kopiert einen langen String in einen anderen. dest%() ist die Zielvariable und src%() ist die Quellvariable. Was auch immer in dest%() war, wird überschrieben
LONGSTRING CONCAT dest%(), src%()	Verkettet einen langen String mit einem anderen. dest%() ist die Zielvariable und src%() ist die Quellvariable. src%() wird am Ende von dest%() hinzugefügt (das Ziel wird nicht überschrieben).
LONGSTRING LCASE array%()	Wandelt alle Großbuchstaben in array%() in Kleinbuchstaben um. array%() muss eine lange String-Variable sein.
LONGSTRING LEFT dest%(), src%(), nbr	Kopiert die 'nbr'-Zeichen der linken Hand von src%() nach dest%() und überschreibt alles, was in dest%() war. d.h. vom Anfang von src%() kopieren. src%() und dest%() müssen lange String-Variablen sein. 'nbr' muss eine ganzzahlige Konstante oder ein ganzzahliger Ausdruck sein.
LONGSTRING LOAD array%(), nbr, string$	Kopiert 'nbr' Zeichen aus string$ in die lange String-Variable array%() und überschreibt alles, was in array%() war.
LONGSTRING MID dest%(), src%(), start, nbr	Kopiert 'nbr' Zeichen von src%() nach dest%() beginnend bei Zeichen position 'start' überschreiben was auch immer in dest%() war. d.h. kopieren von der Mitte von src%(). 'nbr' ist optional und wenn weggelassen, die Zeichen von 'start' bis das Ende des Strings wird kopiert src%() und dest%() muss ein langer String sein Variablen. „start“ und „nbr“ müssen ganzzahlige Konstanten oder Ausdrücke sein.
LONGSTRING PRINT [#n,] src%()	Druckt den in ‘src%()’ gespeicherten Longstring in die Datei oder den COM-Port, der geöffnet wurde als '#n'. Wenn „#n“ nicht angegeben ist, wird die Ausgabe an die Konsole gesendet.
LONGSTRING REPLACE array%() , string$, start	Ersetzt Zeichen in der normalen MMBasic-Zeichenfolge string$ durch an vorhandenes langes String-Array%(), beginnend an Position „start“ in der langen Zeichenfolge.
LONGSTRING RESIZE addr%(), nbr	Setzt die Größe des Longstrings auf nbr. Dies überschreibt die von anderen festgelegte Größe longstring-Befehle sollten daher mit Vorsicht verwendet werden. Typische Verwendung wäre bei der Verwendung eines Longstrings als Byte-Array.
LONGSTRING RIGHT dest%(), src%(), nbr	Kopiert die rechten 'nbr'-Zeichen von src%() nach dest%() und überschreibt sie was auch immer in dest%() war. d.h. vom Ende von src%() kopieren. src%() und dest%() müssen lange String-Variablen sein. 'nbr' muss eine ganzzahlige Konstante oder Ausdruck sein.
LONGSTRING SETBYTE addr%(), nbr, data	Setzt Byte nbr auf den Wert „data“, nbr berücksichtigt OPTION BASE

LONGSTRING TRIM array%(), nbr	Schneidet „nbr“-Zeichen von der linken Seite einer langen Zeichenfolge ab. array%() muss eine lange String-Variable sein. 'nbr' muss eine ganzzahlige Konstante oder Ausdruck sein.
LONGSTRING UCASE array%()	Konvertiert alle Kleinbuchstaben in array%() in Großbuchstaben. Array%() muss eine lange Zeichenfolgenvariable sein.
LOOP [UNTIL expression]	Beendet eine Programmschleife: siehe DO.

MATH
 	
Simple array arithmetic
 	
MATH SET nbr, array()
 	
MATH SCALE in(), scale ,out()
 	
MATH ADD in(), num ,out()
 	
MATH INTERPOLATE in1(), in2(), ratio, out()
 	
MATH SLICE sourcearray(), [d1] [,d2] [,d3] [,d4] [,d5] , destinationarray()
	Der Math-Befehl führt viele einfache mathematische Berechnungen durch kann in BASIC programmiert werden, aber es gibt Geschwindigkeitsvorteile durch die in der Firmware codierten Funktionen, ohne hier das Rad neu zu erfinden.
	Hinweis: 2-dimensionale mathematische Matrizen werden immer angegeben DIM matrix(n_columns, n_rows) und natürlich respektieren die Dimensionen OPTION BASE. Quaternionen werden als 5-Elemente-Array w, x, y, z, Magnitude gespeichert.
	
	Setzt alle Elemente in array() auf den Wert nbr. Beachten Sie, dass dies der schnellste Weg ist Löschen eines Arrays durch Setzen auf Null.
	
	Dies skaliert die Matrix in() um die Skalar-Skala und setzt die Antwort in out(). Funktioniert für Arrays jeder Dimension von Integer und Float und kann dazwischen konvertieren. Das Setzen von b auf 1 ist optimiert und ist der schnellste Weg zum Kopieren eines ganzen Arrays.
	
	Dies fügt jedem Element der Matrix in() den Wert 'num' hinzu und setzt die Antwort in out(). Funktioniert für Arrays jeder Dimension von Integer und float und kann dazwischen konvertieren. Das Setzen von num auf 0 ist optimiert und ermöglicht es, schnell ein ganzes Array zu kopieren. in() und out() können dasselbe Array sein.
	
	Dieser Befehl implementiert die folgende Gleichung für jedes Array-Element: out = (in2 - in1) * Verhältnis + in1
	Arrays können eine beliebige Anzahl von Dimensionen haben und müssen eindeutig sein und haben die gleiche Anzahl von Gesamtelementen. Der Befehl arbeitet mit bot integer und Fließkomma-Arrays in beliebiger Mischung
	
	Dieser Befehl kopiert einen bestimmten Satz von Werten aus einem mehrdimensionalen
	Array in ein eindimensionales Array. Es ist viel schneller als die Verwendung eines FOR
	Schleife. Der Slice wird angegeben, indem für alle bis auf einen der Quelle ein Wert angegeben wird
	Array-Indizes und der Befehl sollte so viele Indizes enthalten, einschließlich das leere, da es Dimensionen im Quellarray gibt
	z.B.
	`OPTION BASE 1`
	`DIM a(3,4,5)`
	`DIM b(4)`
	`MATH SLICE a(), 2, , 3, b()`
	Kopiert die Elemente 2,1,3 und 2,2,3 und 2,3,3 und 2,4,3 in das Array b()

MATH INSERT targetarray(), [d1] [,d2] [,d3] [,d4] [,d5] , sourcearray()	Dies ist das Gegenteil von MATH SLICE, hat eine sehr ähnliche Syntax und ermöglicht es mit einem Einzelbefehl einen einzelnen Vektor durch ein Vektorarray zu ersetzen, Beispiel:
	`OPTION BASE 1`
	`DIM targetarray(3,4,5)`
	`DIM sourcearray(4)=(1,2,3,4)`
	`MATH INSERT targetarray(), 2, , 3, sourcearray()`
	Setzt die Elemente 2,1,3 = 1 und 2,2,3 = 2 und 2,3,3 = 3 und 2,4,3 = 4

Matrix arithmetic	
MATH M_INVERSE array!(), inversearray!()	Dies gibt die Umkehrung von array!() in inversearray!() zurück. Das Array muss square sein und Sie erhalten einen Fehler, wenn das Array nicht invertiert werden kann. (Determinante=0). array!() und inversearray!() dürfen nicht identisch sein.
MATH M_PRINT array()	Schnelldruck einer 2D-Matrix mit einer Zeile pro Zeile.
MATH M_TRANSPOSE in(), out()	Matrix in() transponieren und die Antwort in matrix out() schreiben, beide Arrays müssen 2D aber nicht quadratisch sein. Wenn nicht quadratisch, dann müssen die Arrays im Format in(m,n) out(n,m) sein.
MATH M_MULT in1(), in2(), out()	Multiplizieren Sie die Arrays in1() und in2() und geben Sie die Antwort in out()c ein. Alle Arrays müssen 2D aber nicht quadratisch sein. Wenn nicht quadratisch dann müssen die Arrays dimensioniert sein in1(m,n) in2(p,m) ,out(p,n)
Vektorarithmetik	
MATH V_PRINT array()	Schneller Mechanismus zum Drucken eines kleinen Arrays in einer einzelnen Zeile
MATH V_NORMALISE inV(), outV()	Konvertiert einen Vektor inV() in Einheitsskalierung und legt die Antwort in outV() (sqr(x*x + y*y +.....)=1 Die Anzahl der Elemente im Vektor ist unbegrenzt
MATH V_MULT matrix(), inV(), outV()	Multipliziert matrix() und Vektor inV() und gibt Vektor outV() zurück. Die Vektoren und die 2D-Matrix kann beliebig groß sein, muss aber die gleiche Kardinalität haben.
MATH V_CROSS inV1(), inV2(), outV()	Berechnet das Kreuzprodukt zweier dreielementiger Vektoren inV1() und inV2() und schreibt die Antwort in outV()
Quaternion-Arithmetik	
MATH Q_INVERT inQ(), outQ()	Kehrt die Quaternion in inQ() um und gibt die Antwort in outQ() ein
MATH Q_VECTOR x, y, z, outVQ()	Konvertiert einen durch x , y und z angegebenen Vektor in einen normalisierten Quaternion-Vektor outVQ() mit der ursprünglichen Magnitude gespeichert
MATH Q_CREATE theta, x, y, z, outRQ()	Erzeugt eine normalisierte Rotationsquaternion outRQ(), um die Quaternion zu rotieren Vektoren um die Achse x,y,z um einen Winkel von Theta. Theta wird im Bogenmaß angegeben aber verwendet die Einstellung von OPTION ANGLE
MATH Q_EULER yaw, pitch, roll, outRQ()	Erzeugt eine normalisierte Rotationsquaternion outRQ(), um die Quaternion Vektoren zu rotieren wie sie durch die Gier-, Nick- und Rollwinkel definiert sind

MATH Q_MULT inQ1(), inQ2(), outQ() MATH Q_ROTATE , RQ(), inVQ(), outVQ()	Mit dem Vektor vor dem „Betrachter“ schaut man von oben auf das Gieren auf der Spitze des Vektors und dreht sich im Uhrzeigersinn, Pitch dreht die Oberseite von der Kamera weg und Rollen dreht sich im Uhrzeigersinn um die z-Achse. Die Gier-, Nick- und Rollwinkel sind standardmäßig auf Radianten eingestellt, respektieren jedoch die Einstellung von OPTION ANGLE Multipliziert zwei Quaternionen inQ1() und inQ2() und schreibt das Ergebnis in outQ() Rotiert den Quell-Quaternion-Vektor inVQ() um die Rotate-Quaternion RQ() und schreibt die Antwort in outVQ()
MATH FFT signalarray!(), FFTarray!()	Führt eine schnelle Fourier-Transformation der Daten in „Signalarray!“ durch. "Signalarray" muss Fließkomma sein und die Größe muss eine Potenz von 2 sein (z. B. s(1023) unter der Annahme, dass OPTION BASE null ist) "FFTarray" muss Gleitkomma sein und die Dimension 2*N haben, wobei N die ist wie das Signalarray (z. B. f (1,1023) unter der Annahme, dass OPTION BASE Null ist) Der Befehl gibt die FFT als komplexe Zahlen mit dem Realteil f(0,n) und dem Imaginärteil in f(1,n) zurück.
MATH FFT INVERSE FFTarray!(), signalarray!()	Führt eine inverse schnelle Fourier-Transformation der Daten in „FFTarray!“ durch. "FFTarray" muss Gleitkomma sein und die Dimension 2*N haben, wobei N sein muss eine Potenz von 2 sein (z. B. f(1,1023) unter der Annahme, dass OPTION BASE gleich Null ist) mit dem Realteil in f(0,n) und Imaginärteil in f(1,n). "signalarray" muss ein Gleitkommawert sein und die einzelne Dimension muss der sein wie das FFT-Array. Der Befehl gibt den Realteil der inversen Transformation zurück
MATH FFT MAGNITUDE signalarray!(),magnitudearray!()	Erzeugt Beträge für Frequenzen für die Daten in „signalarray!“ "signalarray" muss Gleitkomma sein und die Größe muss eine Potenz von 2 sein (z.B. s(1023) unter der Annahme, dass OPTION BASE null ist) "magnitudearray" muss ein Gleitkommawert sein und die Größe muss die gleiche sein wie signal array. Der Befehl gibt die Größe des Signals bei verschiedenen Frequenzen zurück nach der formel: Frequenz bei Array-Position N = N * sample_frequency / number_of_samples
MATH FFT PHASE signalarray!(), phasearray!()	Erzeugt Phasen für Frequenzen für die Daten in „signalarray!“. "signalarray" muss Gleitkomma sein und die Größe muss eine Potenz von 2 sein (z.B. s(1023) unter der Annahme, dass OPTION BASE null ist) "phasearray" muss Fließkomma sein und die Größe muss die gleiche sein wie signal array Der Befehl gibt den Phasenwinkel des Signals bei verschiedenen Frequenzen nach obiger Formel zurück.
MATH SENSORFUSION type ax, ay, az, gx, gy, gz, mx, my, mz, pitch, roll, yaw [,p1] [,p2]	Typ kann MAHONY oder MADGWICK sein Ax, ay und az sind die Beschleunigungen in den drei Richtungen und sollten es sein angegeben in Einheiten der Standard-Erdbeschleunigung. Gx, gy und gz sind die Momentanwerte der Rotationsgeschwindigkeit, die sollten in Radianten pro Sekunde angegeben werden. Mx, my und mz sind die Magnetfelder in den drei Richtungen und sollten in Nano-Tesla (nT) angegeben werden Es muss darauf geachtet werden, dass die x-, y- und z-Komponenten konsistent sind zwischen den drei Eingängen. Also zum Beispiel bei Verwendung des MPU-9250 ist die korrekte Eingabe ax, ay,az, gx, gy, gz, my, mx, -mz basierend auf dem Sensor-Messwert. Nicken, Rollen und Gieren sollten Gleitkommavariablen sein und enthalten die Sensorausgänge . Die SENSORFUSION-Routine misst automatisch die Zeit zwischen

	aufeinanderfolgenden Aufrufen und verwendet dies in seinen internen Berechnungen. Der Madwick-Algorithmus nimmt einen optionalen Parameter p1. Dies wird in der Berechnung als Beta verwendet. Der Standardwert ist 0,5 wenn nicht anders angegeben Der Mahony-Algorithmus verwendet zwei optionale Parameter p1 und p2. Diese werden als Kp und Ki in der Berechnung verwendet. Wenn nicht anders angegeben sind diese standardmäßig 10.0 bzw. 0,0. Ein vollständig funktionierendes Beispiel für die Verwendung des Codes finden Sie im BackShed-Forum unter: https://www.thebackshed.com/forum/ViewTopic.php?TID=13459&PID=166962#166962

MEMORY	Zeigt die aktuelle Speicherbelegung an, Beispiel:
	`Program:`
	`  0K ( 0%) Program (0 lines)`
	`  127K (100%) Free`
	`RAM:`
	`  0K ( 0%) 0 Variables`
	`  0K ( 0%) General`
	`  112K (100%) Free`
	Hinweise:
	• Angezeigter Speicher wird auf 1 KB gerundet.
	• Allgemeiner Speicher wird von seriellen E/A-Puffer usw. verwendet.

MEMORY SET address, byte, numberofbytes MEMORY SET BYTE address, byte, numberofbytes MEMORY SET SHORT address, short, numberofshorts MEMORY SET WORD address, word, numberofwords MEMORY SET INTEGER address, integervalue ,numberofintegers [,increment] MEMORY SET FLOAT address, floatingvalue ,numberofloats [,increment]	Dieser Befehl setzt einen Speicherbereich auf einen Wert. BYTE = Ein Byte pro Speicheradresse. SHORT = Zwei Bytes pro Speicheradresse. WORD = Vier Bytes pro Speicheradresse. FLOAT = Acht Bytes pro Speicheradresse. increment‘ ist optional und steuert die Erhöhung des ‚address‘-Zeigers wenn die Operation ausgeführt wird. Beispiel: Inkrement = 3 dann wird nur jedes dritte Element des Ziels gesetzt. Standardwert ist 1.

MEMORY COPY sourceaddress, destinationaddres, numberofbytes MEMORY COPY INTEGER sourceaddress, destinationaddress, numberofintegers [,sourceincrement][,destinationincrement] MEMORY COPY FLOAT sourceaddress, destinationaddress, numberoffloats [,sourceincrement][,destinationincrement]	Dieser Befehl kopiert einen Speicherbereich in einen anderen. COPY INTEGER und FLOAT kopieren acht Bytes pro Operation. „sourceincrement“ ist optional und steuert die Erhöhung der 'Quelladresse'-Zeiger, wenn die Operation ausgeführt wird. Zum Beispiel, wenn sourceincrement=3 dann wird nur jedes dritte Element der Quelle kopiert. Standardwert ist 1. „destinationincrement“ ist ähnlich und arbeitet mit dem „destinationaddress“- Zeiger.
MKDIR dir$	Erstellt das Verzeichnis ‘dir$’ auf der SD Karte.
MID$(str$, start [, num]) = str2$	Die Zeichen in 'str$' beginnend bei Position 'start' werden durch die Zeichen in 'str2$' ersetzt. Das optionale 'num' bezieht sich auf die Anzahl der Zeichen aus 'str2', die beim Ersetzen verwendet werden. Wenn 'num' weggelassen wird, ist 'str2' vollständig gebraucht. Ob 'num' weggelassen oder eingeschlossen wird, die Ersetzung von Zeichen geht nie über die ursprüngliche Länge von 'str$' hinaus.
NEW	Löscht das Programm im RAM und alle Variablen einschließlich gespeicherter Variablen. Dieser Befehl löscht auch die darin gespeicherte Sicherungskopie des Programms im Flash-Speicher.
NEXT [counter-variable] [, counter-variable], etc	NEXT steht am Ende einer FOR-NEXT-Schleife siehe bei FOR „counter-variable“ gibt genau an in welcher Schleife gearbeitet wird. Wenn „couter-variable“ fehlt wird NEXT standardmäßig auf die innerste Schleife gesetzt. Es ist auch möglich, mehrere Gegenvariablen anzugeben, wie in: NEXT x,y, z
ON ERROR ABORT oder ON ERROR IGNORE oder ON ERROR SKIP [nn] oder ON ERROR CLEAR	Dies steuert die Aktion die durchgeführt wird, wenn während der Ausführung eines Programms ein Fehler auftritt und gilt für alle von MMBasic entdeckten Fehler einschließlich Syntaxfehler, falsche Daten, fehlende Hardware etc. ON ERROR ABORT veranlasst MMBasic eine Fehlermeldung anzuzeigen, bricht das Programm ab und kehrt zur Eingabeaufforderung zurück. Dies ist das normale Verhalten und ist die Standardeinstellung, wenn ein Programm gestartet wird. ON ERROR IGNORE bewirkt dass alle Fehler ignoriert werden. ON ERROR SKIP ignoriert einen Fehler in einer Reihe von Befehlen (angegeben durch die Zahl 'nn'), die nach diesem Befehl ausgeführt wird. 'nn' ist optional der Defaultwert ist eins. Nachdem die Befehle abgeschlossen ist (mit oder ohne Fehler) kehrt das Verhalten von MMBasic zurückgreifen auf ON ERROR ABORT. Wenn ein Fehler auftritt und ignoriert/übersprungen wird wird die Nur-Lese-Variable MM.ERRNO wird auf einen Wert ungleich Null gesetzt und MM.ERRMSG$ wird auf die Fehlermeldung gesetzt die normalerweise erzeugt wird. Diese werden durch ON ERROR CLEAR auf Null und einen leeren String zurückgesetzt. Sie werden auch gelöscht wenn das Programm ausgeführt wird und wenn ON ERROR IGNORE und ON ERROR SKIP verwendet werden. ON ERROR IGNORE kann das Debugging eines Programms sehr erschweren, es wird daher dringend empfohlen nur ON ERROR SKIP zu verwenden.

ON KEY target oder ON KEY ASCIIcode, target	Die erste Version des Befehls setzt einen Interrupt, der die vom „target“-Benutzer definierte Unterroutine aufruft und zwar immer dann wenn ein oder mehrere Zeichen im Eingabepuffer der seriellen Konsole warten. Beachten Sie dass alle diese Zeichen vom Unterprogramm gelesen werden andernfalls wird automatisch ein weiterer Interrupt generiert sobald das Programm vom Interrupt zurückkehrt. Die zweite Version ermöglicht es Ihnen eine Interrupt-Routine mit einer bestimmten Tastendruck zu verknüpfen. Dies arbeitet auf für die serielle Konsole auf “Low Level” und wenn aktiviert wird die Eingabe nicht in den Eingabepuffer gestellt, sondern löst lediglich den Interrupt aus. Es verwendet einen eigenen Interrupt vom einfachen ON KEY-Befehl, sodass es verwendet werden kann bei Bedarf. Verwenden Sie in beiden Varianten zum Deaktivieren des Interrupts die numerische Null für das Ziel, dh: ON KEY 0. oder ON KEY ASCII-Code, 0
ONEWIRE RESET pin oder ONEWIRE WRITE pin, flag, length, data [, data…] oder ONEWIRE READ pin, flag, length, data [, data…]	Befehle zur Kommunikation mit 1-Wire-Geräten. ONEWIRE RESET setzt den 1-Wire-Bus zurück ONEWIRE WRITE sendet eine Anzahl von Bytes ONEWIRE READ liest eine Anzahl von Bytes 'Pin' ist der I/O-Pin (befindet sich im hinteren Anschluss) der verwendet werden soll. Das kann ein beliebiger Pin sein geeignet für digitale I/O. 'flag' ist eine Kombination der folgenden Optionen: 1 - Reset vor dem Befehl senden 2 - Reset nach Befehl senden 4 - Senden/empfangen Sie nur ein Bit anstelle eines Datenbytes 8 – Ruft nach dem Befehl einen starken Pullup auf (der Stift wird H gesetzt und Open-Drain deaktiviert) „lenght“ ist die Länge der zu sendenden oder zu empfangenden Daten 'Data' sind die zu sendenden Daten oder die zu empfangende Variable. Die Anzahl der Datenelemente muss mit dem Längenparameter übereinstimmen. Siehe auch Anhang C.
OPEN fname$ FOR mode AS [#]fnbr	Öffnet eine Datei zum Lesen oder Schreiben. „fname“ ist der Dateiname mit optionaler Erweiterung, getrennt durch einen Punkt (.). Lange Dateinamen mit Groß- und Kleinschreibung werden unterstützt. Als Verzeichnistrennzeichen kann ein Verzeichnispfad mit dem Backslash angegeben werden. Das übergeordnete Verzeichnis des aktuellen Verzeichnisses kann mithilfe eines Verzeichnisnamen angegeben werden Name von .. (zwei Punkte) und das aktuelle Verzeichnis mit . (ein einzelner Punkt). Zum Beispiel OPEN "..\dir1\dir2\filename.txt" FOR INPUT AS #1 „mode“ ist INPUT, OUTPUT, APPEND oder RANDOM. INPUT öffnet die Datei zum Lesen und gibt einen Fehler aus wenn die Datei nicht existiert. OUTPUT öffnet die Datei zum Schreiben und wird automatisch eine vorhandene Datei mit demselben Namen überschreiben. APPEND öffnet die Datei auch zum Schreiben überschreibt aber keine vorhandene Datei; stattdessen werden alle Schreibvorgänge an das Ende der Datei angehängt. Wenn keine Datei vorhandene ist verhält sich der APPEND-Modus genauso wie der OUTPUT-Modus (d. h. die Datei wird erstellt und dann zum Schreiben geöffnet). RANDOM öffnet die Datei sowohl zum Lesen als auch zum Schreiben und wahlfreien Zugriff mit dem SEEK-Befehl. Beim Öffnen ist der Lese-/Schreibzeiger am Ende der Datei. „fnbr“ ist die Dateinummer (1 bis 10). Das # ist optional. Es können bis zu 10 Dateien gleichzeitig geöffnet sein. Die Funktionen INPUT, LINE INPUT, PRINT, WRITE und CLOSE-Befehle sowie die Funktionen EOF() und INPUT$() verwenden alle „fnbr“ um die bearbeitete Datei zu identifizieren.

	Siehe auch OPTION ERROR und MM.ERRNO zur Fehlerbehandlung.
OPEN comspec$ AS [#]fnbr	Öffnet einen seriellen Kommunikationsport zum Lesen und Schreiben. Zwei Ports COM1: und COM2: sind verfügbar und können beide gleichzeitig geöffnet sein. Eine vollständige Beschreibung mit Beispielen finden Sie in Anhang A. Mit 'fnbr' kann der Port mit jedem Befehl geschrieben und gelesen werden oder Funktion die eine Dateinummer verwendet.
OPEN comspec$ AS GPS [,timezone_offset] [,monitor]	Öffnet einen seriellen Kommunikationsanschluss zum Lesen eines GPS-Empfängers. Einzelheiten finden Sie in der GPS-Funktion. Die interpretierten Datensätze sind GPRMC, GNRMC, GPCGA und GNCGA. Der Parameter timezone_offset wird verwendet um die vom GPS gelieferte UTC in die lokale Zeitzone umzuwandeln. Wenn es weggelassen wird wird die Zeitzone standardmäßig auf UTC eingestellt. timezone_offset kann eine beliebige Zahl zwischen -12 und 14 sein, die die Zeit erlaubt auch für die Chatham-Inseln in Neuseeland (UTC +12:45). Wenn der Monitorparameter auf 1 gesetzt ist werden alle GPS-Eingaben an die Konsole weitergeleitet. Dies kann durch Schließen des GPS-Kanals beendet werden.
OPTION	Siehe Abschnitt Optionen weiter oben in diesem Handbuch.
PAGE #n [,#n2, #n3, etc]	Dies schaltet das Display um um zugewiesene Bedienelemente anzuzeigen (via GUI SETUP-Befehl) zu den Seitenzahlen die im Befehl angegeben sind Zeile (#n, #n2 usw.). Alle Steuerelemente die angezeigt wurden, sich aber nicht auf der aktuellen Seitenliste befinden werden automatisch ausgeblendet. Alle Steuerelemente einer Seite die auf dem alten Bildschirm angezeigt wurde und auch in der neuen PAGE Befehl spezifiziert sind bleiben davon unberührt. Die Standardeinstellung beim Start eines Programms ist PAGE 1 und GUI SETUP 1. Das bedeutet dass das Programm normal ausgeführt wird wenn diese Befehle nicht verwendet werden, wobei alle GUI-Steuerelemente angezeigt werden die definiert wurden. Siehe auch den GUI SETUP-Befehl.
PAUSE delay	Hält die Ausführung des laufenden Programms für „delay“ ms an. Dies kann ein Fraktion. Beispielsweise entspricht 0,2 200 µs. Die maximale Verzögerung beträgt 2147483647 ms (ca. 24 Tage). Beachten Sie dass Interrupts während einer Pause erkannt und verarbeitet werden.
PIN(pin) = value	Bei einem als Digitalausgang konfigurierten „Pin“ wird hierdurch der Ausgang auf Low gesetzt („Wert“ ist Null) oder hoch („Wert“ ungleich Null). Sie können einen Ausgang High oder Low einstellen bevor er als Ausgang konfiguriert wird und diese Einstellung ist der Standardausgang wenn der SETPIN-Befehl wirksam wird. Siehe die Funktion PIN() zum Lesen eines Pins und Befehl SETPIN zum konfigurieren. Allgemeine Informationen finden Sie im Kapitel „Verwendung der I/O-Pins“. Beschreibung der Ein-/Ausgabefähigkeiten von PicoMite.
PIO PIO INIT MACHINE pio%, statemachine%, clockspeed [,pinctrl] [,execctrl] [,shiftctrl] [,startinstruction]	Der im PicoMite verwendete RP2040-Chip enthält ein programmierbares I/O-System mit zwei identischen PIO-Geräten, die wie spezialisierte CPU-Kerne fungieren. Siehe die Anhang für eine ausführlichere Beschreibung. Initialisiert PIO 'pio%' mit Zustandsmaschine 'statemachine%'. 'clockspeed' ist die Taktrate der state machine in kHz. Die vier optionalen Argumente sind die Initialisierungs-Variablen der state machine-Register und der Adresse der ersten auszuführenden Anweisung (standardmäßig null). Diese entscheiden wie die PIO funktionieren wird. Es wird erwartet dass der PIO-Assembler in der Lage sein wird die Registerwerte für den Benutzer zu generieren zusammen mit dem Programmarray basierend auf den definierten Assembler-Direktiven.

PIO EXECUTE pio, state_machine, instruction%	Führt die Anweisung sofort auf dem pio und der Zustandsmaschine aus.
PIO WRITE pio, state_machine, count, data0 [,data1....]	Schreibt die Datenelemente in den angegebene pio und state machine. Das Schreiben blockiert,also muß die state machine in der Lage sein die gelieferten Daten aufzunehmen. NB: Dieser Befehl wird wahrscheinlich in zukünftigen Versionen zusätzliche Fähigkeiten benötigen
PIO READ pio, state_machine, count, data%()	Eeads 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. NB: this command will probably need additional capability in future releases
PIO START pio, statemachine	Starten Sie eine gegebene Zustandsmaschine auf pio
PIO STOP pio, statemachine	Stoppen Sie eine gegebene Zustandsmaschine auf pio
PIO CLEAR pio	Dies stoppt das auf allen Zustandsmaschinen angegebene pio und löscht die Steuerregister für die State machines PINCTRL, EXECTRL und SHIFTCTRL auf die Voreinstellungen
PIO PROGRAM LINE pio, line, instruction	Programmiert nur die angegebene Zeile in einem PIO-Programm
PIXEL x, y [,c]	Setzt ein Pixel auf einem angeschlossenen LCD-Panel auf eine Farbe. 'x' ist die Horizontale Koordinate und 'y' ist die vertikale Koordinate des Pixels. 'c' ist ein 24-Bit Nummer, die die Farbe angibt. 'c' ist optional, wenn weggelassen, wird die Vordergrundfarbe verwendet. Alle Parameter können als Arrays ausgedrückt werden und die Software zeichnet die Anzahl der Pixel auf wie durch die Abmessungen des kleinsten Arrays bestimmt. 'x' und 'y' müssen beide Arrays sein oder beides sein einzelne Variablen/Konstanten sonst wird ein Fehler generiert. 'c' kann sein entweder ein Array oder eine einzelne Variable oder Konstante. Siehe den Abschnitt Grundlegende Zeichenbefehle für eine Definition der Farben und Grafikkoordinaten.
PLAY	Dieser Befehl erzeugt eine Vielzahl von Audioausgaben. Siehe den Befehl OPTION AUDIO zum Einstellen der zu verwendenden I/O-Pins die Ausgabe. Das Audio ist ein PWM-Signal, also ist ein Tiefpassfilter erforderlich um die Trägerfrequenz zu entfernen.
PLAY TONE left [, right [, dur]]	Erzeugt zwei separate Sinuswellen am linken und rechten Tonausgang. 'links' und 'rechts' sind die Frequenzen in Hz die für den linken und rechten Kanäle verwendet werden. Der Ton spielt im Hintergrund (das Programm wird nach diesem Befehl weiterlaufen) und 'dur' gibt die Anzahl der Millisekunden an die der Ton ertönen wird. Wenn die Dauer nicht angegeben ist wird der Ton fortgesetzt bis er explizit angehalten oder das Programm beendet wird. Die Frequenz kann zwischen 1 Hz und 20 kHz liegen und ist sehr genau (sie basiert auf auf einem Quarzoszillator). Die Frequenz kann jederzeit durch geändert werden Ausgeben eines neuen PLAY TONE-Befehls.
PLAY WAV file$ [, interrupt]	Spielt eine WAV-Datei auf der Tonausgabe ab. 'file$' ist die abzuspielende WAV-Datei (die Erweiterung .wav wird angehängt, wenn sie fehlt). Die WAV-Datei muss PCM-kodiert in Stereo im 8-Bit -Format. Die Abtastrate kann bis zu 48 kHz in Stereo betragen. Die WAV-Datei wird im Hintergrund abgespielt. 'interrupt' ist optional und ist die Name eines Unterprogramms, das aufgerufen wird wenn die Datei das Abspielen beendet hat.

PLAY SOUND soundno, channelno, type [,frequency] [,volume]	Spielen Sie eine Reihe von Tönen gleichzeitig am Audioausgang ab. 'soundno' ist die Soundnummer und kann zwischen 1 und 4 liegen wobei vier Töne auf jedem Kanal gleichzeitig zulässig sind. 'channelno' spezifiziert die Ausgabe Kanal. Es kann L (linker Lautsprecher), R (rechter Lautsprecher) oder B (beide Lautsprecher) sein. 'Typ' ist der Typ der Wellenform. Möglich sind S (Sinus), Q (Rechteck), T (Dreieck), W (ansteigender Sägezahn), N (Rauschen), P (periodisches Rauschen) oder O (Ton aus). Typ N ist echtes weißes Rauschen. In diesem Fall spezifiziert der Frequenzparameter die Anzahl der Perioden von 1/70000 Sekunden die der Ausgang auf einem bestimmten Zufallswert bleibt. Typ P ist periodisches weißes Rauschen. In diesem Fall ist die Frequenz eine Art Beziehung zur periodischen Frequenz des Rauschens „frequency“ ist die Frequenz von 1 bis 20000 (Hz) und muss angegeben werden außer beim Typ O. „volume“ ist optional und muss zwischen 1 und 25 liegen default ist 25. Beim ersten Aufruf von PLAY SOUND wird jede andere Audionutzung gesperrt und bleibt gesperrt bis PLAY STOP aufgerufen wird. Die Ausgabe pausiert mit PLAY PAUSE / PLAY RESUME. Das Aufrufen von SOUND auf einem bereits laufenden 'soundno' ersetzt die vorherige Ausgabe. Individuelle Sounds werden mit Typ „O“ abgeschaltet. 4 Sounds gleichzeitig auf beiden Kanälen des Audioausgangs zu erzeugen kostet ca. 23% CPU-Last.
PLAY PAUSE	PLAY PAUSE hält die aktuell wiedergegebene Datei oder den Ton vorübergehend an.
PLAY RESUME	PLAY RESUME setzt die Wiedergabe eines angehaltenen Sounds fort.
PLAY STOP	PLAY STOP beendet die Wiedergabe der Datei oder des Tons. Wenn das Programm abbricht wird die Tonausgabe automatisch gestoppt.
PLAY VOLUME left, right	Stellt die Lautstärke der Audioausgabe ein. „left“ und „right“ sind die Pegel die für den linken und rechten Kanal verwendet werden können im Bereich 0 -100. Es gilt eine lineare Beziehung zwischen dem Wert und Ausgangssignal. Die Lautstärke ist standardmäßig auf Maximum.
POKE BYTE addr%, byte oder POKE SHORT addr%, short% Oder POKE WORD addr%, word% oder POKE INTEGER addr%, int% oder POKE FLOAT addr%, float! oder POKE VAR var, offset, byte oder POKE VARTBL, offset, byte	Setzt ein Byte oder ein Wort innerhalb des virtuellen PIC32-Speicherraums. POKE BYTE setzt das Byte (d. h. 8 Bits) an der Speicherstelle 'addr%' auf 'Byte'. 'addr%' sollte eine ganze Zahl sein. POKE SHORT setzt die kurze Ganzzahl (d. h. 16 Bit) an der Speicherstelle 'adr%' zu 'word%'. 'addr%' und short%' sollten ganze Zahlen sein. POKE WORD setzt das Wort (d.h. 32 Bits) auf den Speicherplatz 'addr%' zu 'Wort%'. „addr%“ und „word%“ sollten ganze Zahlen sein. POKE INTEGER setzt die MMBasic-Ganzzahl (d. h. 64 Bit) im Speicher Speicherort 'addr%' bis int%'. 'addr%' und int%' sollten ganze Zahlen sein. POKE FLOAT setzt das Wort (d.h. 32 Bits) an den Speicherplatz 'addr%' auf Gleitkomma. 'addr%' sollte eine ganze Zahl sein und 'float!' eine Gleitkommazahl. POKE VAR setzt ein Byte in der Speicheradresse von 'var'. "Offset" ist der ±Offset von der Adresse der Variablen. Ein Array wird als var() angegeben. POKE VARTBL setzt ein Byte in der Variablentabelle von MMBasic. “Offset" ist ±Offset vom Anfang der Variablentabelle. Beachten Sie, dass nach dem Schlüsselwort VARTBL ein Komma erforderlich ist.
POLYGON n, xarray%(), yarray%() [, bordercolour] [, fillcolour] POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()]	Zeichnet ein gefülltes oder umrissenes Polygon mit n xy-Koordinatenpaaren in xarray%() und yarray%(). Wird „fillcolour“ weggelassen ist nur der Polygonumriss gezeichnet. Wird „bordercolour“ weggelassen wird, wird standardmäßig die aktuelle Vordergrundfarbe verwendet. Wenn das letzte xy-Koordinatenpaar nicht dasselbe wie das erste ist wird die Firmware automatisch ein zusätzliches xy-Koordinatenpaar erstellen um das Polygon zu vervollständigen. Die Größe der Arrays sollte mindestens so groß sein wie die Anzahl von x,y Koordinatenpaare.

POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour]	'n' kann ein Array sein und die Farben können optional auch wie folgt Arrays sein:
	POLYGON n(), xarray%(), yarray%() [, bordercolur()] [, fillcolour()]
	POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour]
	Die Elemente des Arrays n() definieren jeweils die Anzahl der xy-Koordinatenpaare der Polygone. z.B. DIM n(1)=(3,3) würde definieren, dass 2 Polygone jeweils drei Eckpunkten gezeichnet werden sollen. Die Größe des n-Arrays bestimmt die Anzahl der Polygone die gezeichnet werden es sei denn, ein Element wird mit dem Wert Null gefunden in diesem Fall verarbeitet die Firmware nur Polygone bis zu diesem Punkt. Die x,y-Koordinatenpaare für alle Polygone werden in xarray%() und yarray%() gespeichert. Die Parameter xarray%() und yarray%() müssen mindestens so viele Elemente haben wie die Summe der Werte im Array n.
	Jedes Polygon kann geschlossen werden, wobei das erste und das letzte Element gleich sind. Sind das letzte und das erste Element ungleich wird die Firmware automatisch ein zusätzliches XY-Koordinatenpaar erstellen um das Polygon zu vervollständigen. Wenn fillcolour weggelassen wird, werden nur die Umrisse der Polygone gezeichnet.
	Die Farbparameter können ein einzelner Wert sein, in diesem Fall sind es alle Polygone in der gleichen Farbe gezeichnet oder sie können Arrays mit der gleichen Kardinalität wie sein n. In diesem Fall kann jedes gezeichnete Polygon eine andere Farbe beider Umrandungen haben und/oder füllen. Dadurch werden beispielsweise 3 Dreiecke in Gelb, Grün und Rot gezeichnet:
	`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%()`

PORT(start, nbr [,start, nbr]…) = value	Setzen Sie mehrere I/O-Pins gleichzeitig d. h. mit einem Befehl. 'start' ist eine E/A-Pin-Nummer und das niedrigste Bit in 'value' (Bit 0) wird verwendet um diesen Stift zu setzen. Bit 1 wird verwendet um den Pin 'Start' plus 1 zu setzen, Bit 2 setzt Pin 'start'+2 und so weiter für 'nbr' Anzahl von Bits. Verwendete I/O-Pins müssen aufsteigend nummeriert sein und alle E/A-Pins die ungültig oder nicht als Ausgang konfiguriert sind verursachen eine Fehlermeldung. Das Paar Start/nbr. kann wiederholt werden wenn eine weitere Gruppe benötigter Ausgangspins hinzugefügt werden müssen. Beispielsweise; PORT(15, 4, 23, 4) = &B10000011 Setzt acht E/A-Pins. Die Pins 15 und 16 werden auf High gesetzt, während 17, 18, 23, 24 und 25 wird auf Low gesetzt und schließlich wird 26 auf High gesetzt. Mit diesem Befehl kann bequem mit Geräten mit par. Interface kommuniziert werden etwa LCD-Displays. Es kann eine beliebige Anzahl von I/O-Pins (und damit Bits) verwendet werden von 1 bis zur Anzahl der I/O-Pins auf dem Chip. Siehe PORT-Funktion, um gleichzeitig von mehreren Pins zu lesen.
PRINT expression [[,;]expression] … etc	Gibt Text an die serielle Konsole aus, gefolgt von einem Wagenrücklauf/Neuzeilen-Paar. Es können mehrere Ausdrücke verwendet werden, die entweder durch a getrennt werden müssen: • Komma (,), das das TAB-Zeichen ausgibt • Semikolon (;), das nichts ausgibt (es wird nur zum Trennen von ausdrücken verwendet). • Nichts oder ein Leerzeichen das sich wie ein Semikolon verhält. Ein Semikolon (;) oder ein Komma (,) am Ende der Liste der Ausdrücke unterdrückt die Ausgabe des CR/LF-Paares am Ende einer Druckanweisung. Beim Drucken wird einer Zahl ein Leerzeichen vorangestellt wenn sie positiv ist oder ein Minus (-) wenn sie negativ ist aber es folgt kein

	Leerzeichen. Ganze Zahlen werden ohne Dezimalpunkt gedruckt während Brüche mit Dezimalzeichen und signifikanten Dezimalstellen gedruckt werden. Große oder kleine Fließkommazahlen werden automatisch im wissenschaftlichen Zahlenformat gedruckt. Die Funktion TAB() kann verwendet werden, um zu einer bestimmten Spalte Abstand zu halten und die STR$() -Funktion kann verwendet werden, um Zeichenfolgen auszurichten oder anderweitig zu formatierens.
PRINT #nbr, expression [[,;]expression] … etc	Dasselbe wie oben, außer dass die Ausgabe auf eine serielle Schnittstelle gerichtet ist oder eine für OUTPUT oder APPEND geöffnete Datei mit der Dateinummer 'nbr'. Siehe OPEN-Befehl.
PRINT #GPS, expression [[,;]expression] … etc	Gibt einen NMEA-String an ein geöffnetes GPS-Gerät aus. Die Zeichenfolge muss mit beginnen ein $-Zeichen und endet mit einem *-Zeichen. Die Prüfsumme wird automatisch berechnet und zusammen mit den CR/LF-Zeichen an den String angehängt.
PULSE pin, width	Erzeugt einen Impuls auf 'pin' mit einer Dauer von 'width' ms. 'width' kann eine Bruchzahl sein. Beispielsweise entspricht 0,01 10 µs und dies ermöglicht die Generierung sehr schmaler Impulse. Das Minimum ist 5 µs bei 40 MHz bis 40 µs bei 5 MHz. Der erzeugte Impuls hat die entgegengesetzte Polarität zum Zustand des I/O-Pins wenn der Befehl ausgeführt wird. Wenn zum Beispiel der Ausgang High eingestellt ist erzeugt der PULSE-Befehl einen negativen Impuls. Anmerkungen: • 'pin' muss als Ausgang konfiguriert werden. • Bei einem Impuls von weniger als 3 ms beträgt die Genauigkeit ± 1 µs. • Bei einem Impuls von 3 ms oder mehr beträgt die Genauigkeit ± 0,5 ms. • Im Hintergrund läuft ein Impuls von 3 ms oder mehr. Bis zu fünf verschiedene und gleichzeitige Impulse können im Hintergrund laufen und jeder kann zeitlich geändert werden indem ein neuer PULSE-Befehl ausgegeben wird, oder es kann beendet werden durch Ausgabe eines PULSE-Befehls mit Null für 'width'.
PWM channel, frequency, [dutyA] [,dutyB] PWM channel, OFF	Es stehen 8 separate PWM-Frequenzen zur Verfügung (Kanäle 0 bis 7) und bis zu 16 Ausgänge mit einzeln gesteuertem Tastverhältnis. Sie können auf beiden ausgeben PWMnA oder PWMnB oder beides für jeden Kanal - ohne Einschränkung. Die Mindestfrequenz beträgt 15 Hz. Die maximale Geschwindigkeit ist OPTION CPUSPEED/4. Bei sehr hohen Geschwindigkeiten werden die Arbeitszyklen zunehmend begrenzt
RANDOMIZE nbr	Initialisiert den Zufallszahlengenerator mit „nbr“. Beim Einschalten wird der Zufallszahlengenerator mit Null initialisiert und wird jedes Mal die gleiche Folge von Zufallszahlen erzeugen. Um jedes Mal eine andere Folge zu generieren müssen Sie einen anderen Wert für „nbr“ verwenden. Die TIMER-Funktion ist dafür praktisch.
REFRESH	Initiiert eine Aktualisierung des Bildschirms für bestimmte Schwarz-Weiß-Anzeigen. Diese können jeweils nur im Vollbildmodus aktualisiert werden und bei OPTION AUTOREFRESH OFF kann dieser Befehl verwendet werden um den Schreibvorgang auszulösen. Dies betrifft folgende Displays: N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI, ST7920, GDEH029A1
RBOX x, y, w, h [, r] [,c] [,fill]	Zeichnet ein Kästchen mit abgerundeten Ecken auf einem angeschlossenen LCD-Panel beginnend bei 'x' und 'y', das 'w' Pixel breit und 'h' Pixel hoch ist. 'r' ist der Radius der Ecken der Box. Standard = 10. 'c' gibt die Farbe an und verwendet standardmäßig die Standard-Vordergrundfarbe, wenn nicht anders angegeben. 'fill' ist die Füllfarbe. In diesem Fall kann sie weggelassen oder auf -1 gesetzt werden Die Box wird nicht gefüllt. Alle Parameter können als Arrays ausgedrückt werden und die Software zeichnet sie auf Anzahl der Boxen, die durch die Abmessungen des kleinsten Arrays bestimmt wird. 'x', 'y', 'w' und 'h' müssen alle Arrays oder alle

	einzelne Variablen/Konstanten sein andernfalls wird ein Fehler generiert. 'r', 'c' und 'fill' können entweder Arrays oder sein einzelne ariablen/Konstanten. Eine Definition der Farben finden Sie im Kapitel "Grundlegende Zeichenbefehle" und Grafikkoordinaten.
READ variable[, variable]...	Liest Werte aus DATA-Anweisungen und weist diese Werte dem benannten Variablen zu. Variablentypen in einer READ-Anweisung müssen mit den Datentypen darin übereinstimmen DATA-Anweisungen, wie sie gelesen werden. Arrays können als Variablen verwendet werden (mit leeren Klammern angegeben, z. B. a()) und In diesem Fall wird die Größe des Arrays verwendet um die Anzahl der zu lesenden Elemente zu bestimmen. Wenn das Array mehrdimensional ist wird die Dimension ganz links sich am schnellsten bewegen. Siehe auch DATA und RESTORE.
REM string	REM ermöglicht das Einfügen von Bemerkungen in ein Programm. Beachten Sie die Verwendung des einfachen Anführungszeichens (‘) im Microsoft-Stil zur Kennzeichnung. Bemerkung wird ebenfalls unterstützt und bevorzugt.
RENAME old$ AS new$	Benennen Sie eine Datei oder ein Verzeichnis von „old$“ in „new$“ um beides Strings. Ein Verzeichnispfad kann sowohl in 'old$' als auch in 'new$' verwendet werden. Wenn sich die Pfade unterscheiden wird die in 'old$' angegebene Datei in den in 'new$' angegebenen Pfad verschoben mit dem Dateinamen wie angegeben.
RESTORE [line]	Setzt die Zeilen- und Positionszähler für die READ-Anweisung zurück. Wenn „line“ angegeben wird werden die Zähler auf den Beginn der angegebenen Zeile zurückgesetzt . ‚line‘ kann dabei eine Zeilennummer, ein Label oder eine Variable mit diesen Werten sein. Wenn „line“ nicht angegeben wird, werden die Zähler auf den Start des Programms zurückgesetzt.
RMDIR dir$	Entfernen oder löschen des Verzeichnisses „dir$“ auf der SD-Karte.
RTC GETTIME	RTC GETTIME erhält das aktuelle Datum/die aktuelle Uhrzeit von einem PCF8563, DS1307 oder DS3231-Echtzeituhr und stellen die interne MMBasic-Uhr passend ein. Datum und Uhrzeit können mit den Funktionen DATE$ und TIME$ abgerufen werden.
RTC SETTIME year, month, day, hour, minute, second	stellt die Zeit im Uhrenchip ein. 'hour' muss die 24 Stunden-Notation verwenden. Der Befehl RTC SETTIME akzeptiert auch eine einzelne Zeichenfolge Argument im Format tt/mm/jj hh:mm. Das heißt, das Datum/die Uhrzeit könnte vom Benutzer unter Verwendung einer GUI FORMATBOX im DATETIME2-Format eingegeben werden
RTC SETREG reg, value RTC GETREG reg, var	Die RTC-Befehle SETREG und GETREG können zum Setzen oder Lesen der Registerinhalte innerhalb des Chips verwendet werden. 'reg' ist die Nummer des Registers, 'value' ist die Nummer, die im Register gespeichert werden soll, und 'var' ist eine Variable der die gelesenen Werte zugewiesen werden. Diese Befehle sind nicht erforderlich für normalen Betrieb, aber sie können verwendet werden, um spezielle Funktionen des Chips zu manipulieren etwa Alarme, Ausgangssignale etc. Sie eignen sich auch zur temporären Aufbewahrung der Informationen im gepufferten RAM des Chips. Diese Chips nutzen I2C und müssen daher mit den beiden I2C-Pins mit Pullup-Widerständen verbunden werden. Wenn der I2C-Bus bereits geöffnet ist wird der RTC-Befehl die aktuellen Einstellungen verwenden andernfalls öffnet er vorübergehend die Verbindung mit einer Geschwindigkeit von 100 kHz. Siehe auch OPTION RTC AUTO ENABLE.
RUN oder RUN[file$]	Führt das im RAM gespeicherte oder auf einer SD-Karte gespeicherte Programm ‚file$‘ aus. Verwenden Sie FLASH RUN n, um ein im Flash gespeichertes Programm auszuführen.

SAVE file$	Speichert das Programm im aktuellen Arbeitsverzeichnis der SD-Karte als ‚file$'. Beispiel: SAVE „TEST.BAS" Wenn keine Erweiterung angegeben wird, wird dem Dateinamen „.BAS" hinzugefügt. Siehe auch FLASH SAVE *n*
SAVE IMAGE file$ [,x, y, w, h]	Speichert das aktuelle Bild auf dem ILI9341-basierten LCD-Panel als BMP-Datei unter dem Namen 'file$'. Wenn keine Erweiterung angegeben ist, wird „.BMP" dem Dateinamen hinzugefügt. Das Bild wird als Echtfarben-24-Bit-Bild gespeichert. 'x', 'y', 'w' und 'h' sind optional und sind die Koordinaten (x und y sind die oberen linke Koordinate) und Abmessungen (Breite und Höhe) des zu speichernden Bereichs. Wenn nicht angegeben, wird der gesamte Bildschirm gespeichert. Dieser Befehl funktioniert nur auf LCD-Panels mit dem ILI9341-Controller.
SEEK [#]fnbr, pos	Positioniert den Lese-/Schreibzeiger in einer Datei, die auf der SD geöffnet wurde Karte für RANDOM-Zugriff auf das 'pos'-Byte. Das erste Byte in einer Datei ist mit eins nummeriert, also positioniert SEEK #5,1 den Lese-/Schreibzeiger auf Dateianfang.

SELECT CASE value
CASE testexp [[, testexp] …]
<statements>
<statements>
CASE ELSE
<statements>
<statements>
END SELECT
	Führt abhängig vom Wert eines Ausdrucks eine von mehreren Gruppen von Anweisungen aus. 'value' ist der zu testende Ausdruck. Es kann eine Zahl sein bzw String-Variable oder ein komplexer Ausdruck.
	'testexp' ist der Wert, mit dem verglichen werden soll. Es kann sein:
	• Ein einzelner Ausdruck (z. B. 34, „Zeichenfolge" oder PIN(4)*5), dem er entsprecht
	• Ein Bereich von Werten in Form von zwei einzelnen Ausdrücken, die durch das getrennt sind Schlüsselwort "TO" (d.h. 5 TO 9 oder "aa" TO "cc")
	• Ein Vergleich, der mit dem Schlüsselwort „IS" beginnt (was optional ist). Für Beispiel: IS > 5, IS <= 10.
	Wenn mehrere Testausdrücke (durch Kommas getrennt) verwendet werden, wird die Die CASE-Anweisung ist wahr wenn einer dieser Tests als wahr gewertet wird.
	Wenn 'value' nicht mit einem 'testexp' abgeglichen werden kann, wird es automatisch abgeglichen mit CASE ELSE. Wenn CASE ELSE nicht vorhanden ist wird <statements> nicht ausgeführt und das Programm nach END SELECT fortgesetzt.
	Wird eine Übereinstimmung gefunden werden die <statements> nach der CASE-Anweisung ausgeführt bis END SELECT oder ein anderer CASE erreicht wird . Das Programm fährt dann mit dem Programm nach END SELECT fort. Es kann eine unbegrenzte Anzahl von CASE-Anweisungen verwendet werden, aber es darf nur ein CASE ELSE geben dies muß direkt vor END SELECT stehen.
	Beispiel:
	`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`
	Jeder SELECT CASE muss eines und nur ein passendes END SELECT haben. Es können beliebig viele SELECT…CASE-Anweisungen verschachtelt werden innerhalb der CASE-Anweisungen anderer SELECT…CASE-Anweisungen

SETPIN pin, cfg [, option]	Konfiguriert einen externen I/O-Pin. Siehe Kapitel „Verwendung der I/O-Pins" für eine allgemeine Beschreibung der Ein-/Ausgabefähigkeiten von

	PicoMite. „pin“ ist der zu konfigurierende I/O-Pin, „cfg“ ist der Modus, auf den der Pin eingestellt werden soll und 'option' ist ein optionaler Parameter. 'cfg' kann eines der folgenden Schlüsselwörter sein: OFF Nicht konfiguriert oder inaktiv AIN Analoger Eingang (d.h. Spannung am Eingang messen). DIN-Digitaleingang Wenn 'Option' weggelassen wird, ist der Eingang hochohmig Wenn 'Option' das Keyword "PULLUP" ist wird ein simulierter Widerstand verwendet um den Eingangsstift auf 3,3 V hochzuziehen. Wenn das Keyword "PULLDOWN" verwendet wird wird der Stift auf Null V gezogen. Bei Pullup/ Pulldown fließt ein konstanter Strom von etwa 50 µA. FIN Frequenzeingang 'option' kann verwendet werden, um die Torzeit festzulegen (die Länge der Zeit zum Zählen der Eingangszyklen). Möglich ist eine beliebige Zahl zwischen 10 ms und 100000 ms. Beachten Sie dass die Funktion PIN() immer verwendet wird geben unabhängig vom Gate die Frequenz korrekt skaliert in Hz zurück. Wenn 'Option' weggelassen wird, beträgt die Torzeit 1 Sekunde. PIN Period input Mit 'option' kann die Anzahl der Eingangszyklen angegeben werden um den Durchschnittswert der Periodenmessungungen zu ermitteln. Es kann eine beliebige Zahl sein zwischen 1 und 10000. Beachten Sie : Die Funktion PIN() immer die durchschnittliche Periode eines Zyklus korrekt skaliert in ms zurückgibt unabhängig von der Anzahl der für den Mittelwert verwendeten Zyklen. Wird 'Option' weggelassen wird als Zeitraum nur ein Zyklus verwendet. CIN Zähleingang DOUT Digitalausgang 'option' kann "OC" sein, in diesem Fall ist der Ausgang Open Kollektor (oder richtiger “Open Drain”). Die Funktionen PIN() und PORT() kann auch verwendet werden, um den Wert auf einem oder mehr Ausgangspins zurück zu geben. Frühere Versionen von MMBasic verwendeten Zahlen für 'cfg' und den Modus OOUT. Aus Gründen der Abwärtskompatibilität werden sie weiterhin erkannt. Siehe die Funktion PIN() zum Lesen von Eingaben und die Anweisung PIN()= für einen Ausgang setzen. Siehe den folgenden Befehl, wenn ein Interrupt konfiguriert ist.
SETPIN pin, cfg, target [, option]	Konfiguriert „pin“, um einen Interrupt gemäß „cfg“ zu erzeugen. Beliebiger I/O-Pin Digitaleingangsfähig kann konfiguriert werden, um einen Interrupt mit a zu erzeugen maximal zehn Interrupts gleichzeitig konfiguriert. „cfg“ ist ein Schlüsselwort und kann Folgendes sein: OFF Nicht konfiguriert oder inaktiv INTH Interrupt bei ansteigender Eingangflanke INTL Interrupt bei fallender Eingangflanke INTB Interrupt auf beiden Flanken (d.h. jede Änderung am Eingang) „target“ ist eine benutzerdefinierte Unterroutine, die aufgerufen wird wenn das Ereignis eintritt. Die Rückkehr vom Interrupt erfolgt über das END SUB oder EXIT SUB Befehle. Als 'Option' können die Schlüsselwörter "PULLUP" oder "PULLDOWN" verwendet werden für einen normalen Eingangspin (SETPIN-Pin DIN) spezifiziert. Wenn 'Option' weggelassen wird, wird der Eingang hochohmig. Dieser Modus konfiguriert den Pin auch als digitalen Eingang, also den Wert des Pins können immer mit der Funktion PIN() abgerufen werden. Eine allgemeine Beschreibung finden Sie

	im Kapitel „Verwendung der I/O-Pins“ für die Ein-/Ausgabefähigkeiten von PicoMite.
SETPIN GP25, DOUT | HEARTBEAT	Diese Version von SETPIN steuert die integrierte LED. Wenn es als DOUT konfiguriert ist kann es per Software ein- und ausgeschaltet werden. Wenn es als HEARTBEAT konfiguriert ist blinkt es kontinuierlich 1 s ein, 1 s aus während es eingeschaltet ist. Dies ist der Standardzustand und wird beim Programmabbruch auf diesen Wert gesetzt .
SETPIN p1[, p2 [, p3]], device	Diese Befehle dienen der Pinzuordnung für spezielle Geräte. Pins müssen aus dem Pinbezeichnungsplan ausgewählt und zugeordnet werden bevor die Geräte verwendet werden können. Die Pins (z. B. rx, tx usw.) können in beliebiger Reihenfolge deklariert werden, auf die Pins kann über deren Pin-Nr. verwiesen werden (z. B. 1, 2) oder GP-Nummer (z. B. GP0, GP1).
SETPIN rx, tx, COM1	Ordnet die PINs COM1 zu. Gültige PINs sind RX: GP1, GP13 oder GP17 TX: GP0, GP12, GP16 oder GP28
SETPIN rx, tx, COM2	COM2. RX: GP5, GP9 oder GP21 TX: GP4, GP8 oder GP20
SETPIN rx, tx, clk, SPI	SPI port SPI 1. RX: GP0, GP4, GP16 oder GP20 TX: GP3, GP7 oder GP19 CLK: GP2, GP6 oder GP18
SETPIN rx, tx, clk, SPI2	SPI port SPI 2. RX: GP8, GP12 oder GP28 TX: GP11, GP15 oder GP27 CLK: GP10, GP14 oder GP26
SETPIN sda, scl, I2C	I^2C port I2C. SDA:GP0, GP4, GP8, GP12, GP16, GP20 oder GP28 SCL:GP1, GP5, GP9, GP13, GP17, GP21 oder GP27
SETPIN sda, scl, I2C2	I^2C port I2C2. SDA:GP2, GP6, GP10, GP14, GP18, GP22 oder GP26 SCL: GP3, GP7, GP11, GP15, GP19 oder GP27
SETPIN pin, PWMnx	Weist PWMnx Pin zu 'n' ist die PWM-Nummer (0 bis 7) und 'x' ist der Kanal (A oder B) Der Setpin kann geändert werden bis der PWM-Befehl ausgegeben wird. An diesem Punkt der Stift wird auf PWM gesperrt bis PWMn,OFF ausgegeben wird.
SETPIN pin, IR	Weist Pins für Infrarot (IR)-Kommunikation zu (kann ein beliebiger Pin sein).
SETPIN pin, PIOn	Reserve-Pin zur Verwendung durch PIO0 oder PIO1 (siehe Anhang E für PIO-Details).
SETTICK period, target [, nbr]	Dies richtet einen periodischen Interrupt (oder "Tick") ein. Es stehen vier Tick-Timer zur Verfügung ('nbr' = 1, 2, 3 oder 4). 'nbr' ist optional, wenn nicht anders angegeben wird Timer Nummer 1 verwendet. Die Zeit zwischen Interrupts ist „period“ Millisekunden und „target“ ist die Interrupt-Subroutine die aufgerufen wird, wenn das zeitgesteuerte Ereignis eintritt. Der Zeitraum kann zwischen 1 und 2147483647 ms (ca. 24 Tage) liegen. Diese Interrupts können deaktiviert werden, indem „periode“ auf Null gesetzt wird (d. h. SETTICK 0, 0, 3 deaktiviert den Tick-Timer Nummer 3.

SETTICK PAUSE, target [, nbr] oder SETTICK RESUME, target [, nbr]	Pausiert oder reaktiviert den Timer. Während des Pausierens ist der Interrupt verzögert aber die aktuelle Zählung wird beibehalten.
SORT array() [,indexarray()] [,flags] [,startposition] [,elementstosort]	Dieser Befehl nimmt ein Array beliebigen Typs (Integer, Float oder String) und sortiert es in aufsteigender Reihenfolge. Es hat einen optionalen Parameter ‚indexarray%()'. Falls verwendet, muss dies eine Ganzzahl sein Array mit der gleichen Größe wie das zu sortierende Array. Nach der Sortierung enthält dieses Array die ursprüngliche Indexposition jedes Elements im zu sortierenden Array bevor es sortiert wurde. Alle Daten im Array werden überschrieben. Dies erlaubt verbundene Arrays, die sortiert werden sollen. Siehe den Abschnitt Sortieren von Daten im Tutorial Programmieren mit dem Color Maximite 2 als Beispiel. Der Parameter „Flag" ist optional und gültige Flag-Werte sind: bit0: 0 (Standard, wenn weggelassen) normale Sortierung - 1 umgekehrte Sortierung bit1: 0 (Standard) abhängig von der Groß-/Kleinschreibung - 1 sort ist unabhängig von der Groß-/Kleinschreibung (nur Zeichenfolgen). Die optionale „Startposition" definiert, mit welchem Element im Array begonnen werden soll Sortieren. Standard ist 0 (OPTION BASIS 0) oder 1 (OPTION BASIS 1) Das optionale „elementstosort" definiert, wie viele Elemente das Array enthält sortiert werden soll. Voreingestellt sind alle Elemente nach der Startposition. Jeder der optionalen Parameter kann weggelassen werden, um beispielsweise nur zu sortieren die ersten 50 Elemente eines Arrays, das Sie verwenden könnten: `SORT array(), , , ,50`
SPI OPEN speed, mode, bits oder SPI READ nbr, array() oder SPI WRITE nbr, data1, data2, data3, … etc oder SPI WRITE nbr, string$ oder SPI WRITE nbr, array() oder SPI CLOSE	Kommunikation über einen SPI-Kanal. Einzelheiten finden Sie in Anhang D. 'nbr' ist die Anzahl der zu sendenden oder zu empfangenden Datenelemente, 'data1', 'data2' usw. können Float oder Integer sein und im Fall von WRITE kann es eine Konstante oder Ausdruck sein. Wenn 'string$' verwendet wird werden 'nbr' Zeichen gesendet. 'array' muss ein eindimensionales Array sein im Float- oder Integer-Format und 'nbr'-Elemente werden gesendet oder empfangen werden.
SPI2	Derselbe Befehlssatz wie für SPI (oben), aber für den zweiten SPI Kanal.
STATIC variable [, variables] Siehe "DIM".	Definiert eine Liste von Variablennamen die für das Unterprogramm oder die Funktion lokal sind. Diese Variablen behalten ihren Wert zwischen Aufrufen der Subroutine oder Funktion (im Gegensatz zu Variablen, die mit dem Befehl LOCAL erstellt wurden). Dieser Befehl verwendet genau die gleiche Syntax wie DIM. Der einzige Unterschied ist dass die Länge des von STATIC erstellten Variablennamens und die Länge der Unterroutinen- oder Funktionsnamen zusammen dürfen nicht länger als 32 Zeichen sein. Statische Variablen können auf einen Wert initialisiert werden. Diese Initialisierung dauert wirken nur beim ersten Aufruf des Unterprogramms (nicht bei nachfolgenden Aufrufen).

SUB xxx (arg1 [,arg2, …]) <statements> <statements> END SUB	Definiert eine aufrufbare Subroutine. Dies ist dasselbe wie das Hinzufügen eines neuen Befehls zu MMBasic während es ein Programm ausführt. 'xxx' ist der Name des Unterprogramms und muss den Spezifikationen für die Benennung von Variablen entsprechen. 'arg1', 'arg2' usw. sind die Argumente oder Parameter für das Unterprogramm. Ein Array wird mit leeren Klammern angegeben. dh arg3(). Der Typ des Arguments kann durch Verwendung eines Typsuffixes (z. B. arg1$) oder durch Angabe des Typs angegeben werden mit AS <Typ> (d. h. arg1 AS STRING). Jede Definition muss eine END SUB-Anweisung haben. Wenn diese erreicht wird kehrt das Programm zum nächsten Statement nach der Subroutine zurück. Der Befehl EXIT SUB kann für ein vorzeitiges Verlassen verwendet werden. Man verwendet die Subroutine indem Sie ihren Namen und ihre Argumente in einem Programm genauso verwenden wie einen normalen Befehl. Zum Beispiel: MySub a1, a2 Wenn die Subroutine aufgerufen wird wird jedes Argument im Aufrufer mit dem Argument in der Subroutinendefinition abgeglichen. Diese Argumente sind nur innerhalb des Unterprogramms verfügbar. Unterprogramme können mit einer variablen Anzahl von Argumenten aufgerufen werden. Alle weggelassenen Argumente in der Liste der Subroutine werden auf Null bzw. Nullstring gesetzt. Argumente in der Anruferliste, die eine Variable sind und den richtigen Typ haben wird per Referenz an das Unterprogramm übergeben. Dies bedeutet dass keine Änderungen zu dem entsprechenden Argument in der Subroutine wird auch in die Aufrufer-Variable kopiert und kann daher zugegriffen werden, nachdem die Unterroutine dies getan hat beendet. Arrays werden übergeben, indem der Array-Name mit leeren Klammern angegeben wird (zB arg()) und werden immer per Referenz übergeben. Klammern um die Die Argumentliste sowohl im Aufrufer als auch in der Definition ist optional.
TEMPR START pin [, precision]	Dieser Befehl kann verwendet werden, um eine Konvertierung zu starten, die auf einem DS18B20-Temperatursensor an 'Pin' läuft. Normalerweise reicht die Funktion TEMPR() allein aus um eine Temperatur zu erzeugen Messung, daher ist die Verwendung dieses Befehls optional. Dieser Befehl startet die Messung am Temperatursensor. Der Programm kann sich dann während der Messung anderen Aufgaben widmen Der Wert kann mit TEMPR() gelesen werden. Wenn das TEMPR() Funktion verwendet wird bevor die Konvertierungszeit die Funktion abgeschlossen ist wartet die Funktion die verbleibende Zeit ab bevor der Wert zurückgegeben wird. Es können beliebig viele dieser Konvertierungen auf verschiedenen Pins gestartet und ausgeführt werden und gleichzeitig laufen. 'precision' ist die Auflösung der Messung und ist optional. Es ist ein Zahl zwischen 0 und 3: 0 = 0,5 °C Auflösung, 100 ms Wandlungszeit 1 = 0,25 °C Auflösung, 200 ms Konvertierungszeit (Standardeinstellung) 2 = 0,125 °C Auflösung, 400 ms Wandlungszeit 3 = 0,0625 ºC Auflösung, 800 ms Wandlungszeit

TEXT x, y, string$ [,alignment$] [, font] [, scale] [, c] [, bc]	Zeigt eine Zeichenfolge auf einem angeschlossenen LCD-Panel an, beginnend bei „x“ und „y“. „string$“ ist der String, der angezeigt werden soll. Numerische Daten sollten in einen String konvertiert werden und mit der Funktion Str$() formatiert werden. „alignment$“ ist ein Stringausdruck oder eine Stringvariable aus 0, 1 oder 2 Buchstaben wobei der erste Buchstabe die horizontale Ausrichtung um 'x' ist und sein kann L, C oder R für LEFT, CENTER, RIGHT und der zweite Buchstabe ist die Vertikale Ausrichtung um 'y' und kann T, M oder B für TOP, MIDDLE, BOTTOM sein. Der Standard ist links oben. Beispielsweise. „CM“ zentriert den Text vertikal und horizontal. Die Zeichenfolge „alignment$“ kann eine Konstante (z. B. „CM“) oder eine Stringvariable sein. Aus Gründen der Abwärtskompatibilität mit früheren Versionen von MMBasic kann die Zeichenfolge auch ohne Anführungszeichen sein (z. B. CM). Bei PicoMite kann ein dritter Buchstabe in der Ausrichtungszeichenfolge verwendet werden, um dies anzuzeigen die Drehung des Textes. Dies kann 'N' für normale Ausrichtung sein, 'V' für vertikaler Text, wobei jedes Zeichen unter dem vorherigen von oben nach oben läuft unten, 'I' wird der Text invertiert (d.h. auf dem Kopf stehend), 'U' wird der Text sein um 90º gegen den Uhrzeigersinn gedreht und 'D' wird der Text im Uhrzeigersinn gedreht 90º 'font' und 'scale' sind optional und werden standardmäßig durch den FONT-Befehl festgelegt. 'c' ist die Zeichenfarbe und 'bc' ist die Hintergrundfarbe. Sie sind optional und standardmäßig auf die aktuellen Vorder- und Hintergrundfarben eingestellt. Eine Definition der Farben finden Sie im Kapitel "Grundlegende Zeichenbefehle". und Grafikkoordinaten.
TIME$ = "HH:MM:SS" oder TIME$ = "HH:MM" oder TIME$ = "HH"	Stellt die Uhrzeit der internen Uhr ein. MM und SS sind optional und werden standardmäßig verwendet auf Null, falls nicht angegeben. Zum Beispiel TIME$ = "14:30" stellt die Uhr auf 14:30 null Sekunden. Beim Einschalten wird die Uhrzeit auf „00:00:00“ eingestellt.
TIMER = msec	Setzt den Timer auf eine Anzahl von Millisekunden zurück. Normalerweise setzt das den Timer auf Null zurück aber es ist jede positive ganze Zahl möglich. Weitere Einzelheiten finden Sie unter der TIMER-Funktion.
TRACE ON oder TRACE OFF oder TRACE LIST nn	TRACE ON/OFF schaltet die Trace-Funktion ein/aus. Diese Einrichtung wird die Nummer jeder Zeile drucken vom Beginn des Programms an in eckige Klammern während das Programm ausgeführt wird. Dies ist beim Debuggen der Programme nützlich. TRACE LIST listet die letzten 'nn' Zeilen auf, die im beschriebenen Format ausgeführt wurden Oben. MMBasic protokolliert immer die ausgeführten Zeilen, also ist dies immer verfügbar d.h. es muss nicht eingeschaltet sein.
TRIANGLE X1, Y1, X2, Y2, X3, Y3 [, C [, FILL]]	Zeichnet ein Dreieck auf dem LCD-Panel mit den Ecken bei X1, Y1 und X2, Y2 und X3, Y3. 'C' ist die Farbe des Dreiecks und standardmäßig die aktuelle Vordergrundfarbe. 'FILL' ist die Füllfarbe Standard= keine Füllung (es kann auch auf -1 für keine Füllung gesetzt werden). Alle Parameter können als Arrays ausgedrückt werden und die Software zeichnet sie auf Anzahl der Dreiecke, die durch die Abmessungen des kleinsten Arrays bestimmt wird außer X1 = Y1 = X2 = Y2 = X3 = Y3 = -1, in diesem Fall stoppt die Verarbeitung an diesem Punkt 'x1', 'y1', 'x2', 'y2', 'x3' und 'y3' müssen alle Arrays sein oder einzelne Variablen / Konstanten sonst wird ein Fehler generiert 'c' und 'fill' können entweder Arrays oder einzelne Variablen/Konstanten sein.
UPDATE FIRMWARE	Veranlasst den PicoMite, in den Firmware-Aktualisierungsmodus zu wechseln (dasselbe wie Einschalten, während Sie die BOOTSEL-Taste gedrückt halten). Das im RAM gespeicherte Programm geht verloren und im

	Flash gespeicherte Programme können beim Upgrade beschädigt werden. Diese sollten also vor dem Upgrade irgendwie gespeichert werden.
VAR SAVE var [, var]… oder VAR RESTORE oder VAR CLEAR	VAR SAVE speichert eine oder mehrere Variablen im nichtflüchtigen Flash-Speicher wo sie später (normalerweise nach einer Stromunterbrechung) wiederhergestellt werden können. 'var' kann eine beliebige Anzahl von numerischen oder String-Variablen und/oder Arrays sein. Arrays werden mit leeren Klammern angegeben. Zum Beispiel: var() VAR RESTORE ruft die zuvor gespeicherten Variablen ab und fügt sie ein (und deren Werte) in die Variablentabelle. Der Befehl VAR SAVE kann wiederholt verwendet werden. Variablen die zuvor gespeichert wurden werden mit ihrem neuen Wert und allen neuen Variablen aktualisiert (zuvor nicht gespeichert) werden zur späteren Wiederherstellung zur gespeicherten Liste hinzugefügt. VAR CLEAR löscht alle gespeicherten Variablen. Auch die gespeicherten Variablen werden automatisch durch den NEW-Befehl gelöscht oder wenn ein neues Programm über AUTOSAVE, XMODEM, etc. geladen wird. Dieser Befehl wird normalerweise zum Speichern von Kalibrierungsdaten, Optionen und anderen Daten verwendet die sich nicht oft ändern, aber über eine Spannungsunterbrechung hinweg beibehalten werden müssen. Normalerweise wird der Befehl VAR RESTORE an den Anfang gestellt des Programms, damit zuvor gespeicherte Variablen wiederhergestellt werden und stehen dem Programm beim Start sofort zur Verfügung. Anmerkungen: • Der für diesen Befehl verfügbare Speicherplatz beträgt 16 KB. • Die Verwendung von VAR RESTORE ohne vorheriges Speichern hat keine Auswirkung und erzeugt keinen Fehler. • Wenn bei Verwendung von RESTORE bereits eine gleichnamige Variable existiert wird der Wert überschrieben. • Gespeicherte Arrays müssen (unter Verwendung von DIM) deklariert werden bevor sie wiederhergestellt werden können. • Beachten Sie, dass String-Arrays bei diesem Befehl schnell den gesamten zugewiesenen Speicher verbrauchen können. Der LENGTH-Qualifizierer kann verwendet werden, wenn es sich um eine Zeichenfolge handelt array wird deklariert, um die Größe des Arrays zu reduzieren (siehe DIM-Befehl). Für gewöhnliche String-Variablen ist dies nicht erforderlich
WATCHDOG timeout oder WATCHDOG OFF	Startet den Watchdog-Timer, der nach Ablauf den Prozessor automatisch neu startet. Dies kann verwendet werden um sich von einem Ereignis zu erholen dass das laufende Programm “lahmlegt” z. B. eine Endlosschleife oder anderer Fehler der ein laufendes Programm blockiert. Dies kann im unbeaufsichtigten Fall wichtig sein. „timeout“ ist die Zeit in Millisekunden (ms) bevor der Neustart erzwungen wird. Der Befehl sollte an strategischen Stellen im laufenden BASIC platziert werden um den Watchdog-Timer ständig zurückzusetzen (auf „Timeout“) und so einen Countdown auf Null zu verhindern. Erreicht der Zähler Null wird PicoMite automatisch neu gestartet und die die automatische Variable MM.WATCHDOG wird auf wahr d.h. 1 gesetzt = Fehler. Bei einem normalen Start wird MM.WATCHDOG auf falsch also 0 gesetzt. Beachten Sie, dass für die OPTION AUTORUN festgelegt werden muss um das Programm neu zu starten. Mit WATCHDOG OFF kann der Watchdog-Timer jederzeit deaktiviert werden. Dies ist die Standardeinstellung bei Reset oder Einschalten. Der Timer wird auch ausgeschaltet wenn das laufende Programm z.B. mit STRG-C in der Konsole unterbrochen wird.

XMODEM SEND oder XMODEM SEND file$ oder XMODEM RECEIVE oder XMODEM RECEIVE file$ oder XMODEM CRUNCH	Überträgt ein BASIC-Programm zu oder von einem entfernten Computer mithilfe des XModem-Protokolls. Die Übertragung erfolgt über die USB-Konsolenverbindung. XMODEM SEND sendet das aktuelle Programm aus PicoMites Programmspeicher zum entfernten Gerät. XMODEM RECEIVE akzeptiert ein Programm das vom entfernten Gerät gesendet wurde und speichert es im RAM des PicoMite wobei das dort gespeicherte Programms überschrieben wird. Sie können auch 'file$' angeben wodurch die Daten an/von einer Datei auf der SD-Karte übertragen werden. Ist die Datei bereits vorhanden wird sie beim Empfang einer Datei überschrieben. Beachten Sie dass die Daten im RAM gepuffert werden was die max. Programmgröße begrenzt. Dieser Befehl erstellt auch eine Sicherungskopie des Programms im Flash-Speicher die automatisch abgerufen wird bei einem Reset der CPU oder bei Spannungsverlust. Die CRUNCH-Option funktioniert wie RECEIVE, aber sie weist MMBasic an vor dem speichern alle Kommentare, Leerzeilen und unnötigen Leerzeichen aus dem Programm zu entfernen. Dies kann bei großen Programmen verwendet werden damit sie besser in den Speicher passen. SEND, RECEIVE und CRUNCH können mit S, R und C abgekürzt werden. Das XModem-Protokoll erfordert ein kooperierendes Softwareprogramm auf dem entfernten Computer und eine serielle Verbindung zwischen beiden. Dafür wurde Tera Term für Windows getestet, die SW wird hier empfohlen. Wählen Sie nach dem Befehl “XMODEM” in MMBasic Folgendes aus: Datei -> Übertragen -> XMODEM -> Empfangen/Senden aus dem Tera Term-Menü um die Übertragung zu starten. Die Übertragung kann bis zu 15 Sekunden dauern. Wenn der XMODEM-Befehl keine Verbindung aufbaut wird der Prozeß nach 60 s abbrechen und der Programmspeicher bleibt unberührt. Tera Term kann z.B. von http://ttssh2.sourceforge.jp/ heruntergeladen werden.

Funktionen

Beachten Sie dass die Funktionen im Zusammenhang mit Kommunikationsfunktionen (I2C, 1-Wire und SPI) hier nicht aufgeführt sind, aber es sind in den Anhängen am Ende dieses Dokuments beschrieben.

Eckige Klammern zeigen an, dass der Parameter oder die Zeichen optional sind.

ABS(number)	Gibt den absoluten Wert des Arguments „Zahl“ zurück (d. h. ein negatives Vorzeichen wird entfernt und eine positive Zahl zurückgegeben).
ACOS(number)	Gibt den inversen Kosinus des Arguments „Zahl“ im Bogenmaß zurück.
ASC(string$)	Gibt den ASCII-Code (d. h. Byte-Wert) für den ersten Buchstaben in „string$“ zurück’.
ASIN(number)	Gibt den inversen Sinuswert des Arguments „Zahl“ im Bogenmaß zurück.
ATN(number)	Gibt den Arkustangens des Arguments „Zahl“ im Bogenmaß zurück.
ATAN2(y, x)	Gibt den Arkustangens der beiden Zahlen x und y als Winkel ausgedrückt im Bogenmaß zurück. Es ähnelt der Berechnung des Arkustangens von y / x außer dass die Vorzeichen beider Argumente verwendet werden um den Quadranten des Ergebnisses zu bestimmen.
BIN$(number [, chars])	Gibt einen String zurück der den Binärwert (Basis 2) für die 'number' angibt. 'chars' ist optional und gibt die Anzahl der Zeichen im String an mit Null als führendes Füllzeichen.
BIN2STR$(type, value [,BIG])	Gibt einen String zurück, der die binäre Darstellung von 'value' enthält. 'Typ' kann sein: INT64 64-Bit-Ganzzahl mit Vorzeichen, konv. in 8-Byte-Zeichenfolge UINT64 64-Bit-Ganzzahl ohne Vorzeichen, konv. in 8-Byte-Zeichenfolge INT32 32-Bit-Ganzzahl m. Vorz., konv. in 4-Byte-Zeichenfolge UINT32 32-Bit-Ganzzahl ohne Vorzeichen, konv. in 4-Byte-Zeichenfolge INT16 16-Bit-Ganzzahl m. Vorz., konv. in 2-Byte-Zeichenfolge UINT16 16-Bit-Ganzzahl ohne Vorz., konv. in 2-Byte-Zeichenfolge INT8 8-Bit-Ganzzahl m. Vorz., konv. in eine 1-Byte-Zeichenfolge UINT8 8-Bit-Ganzzahl ohne Vorz., konv. in eine 1-Byte-Zeichenfolge SINGLE Fließkommaz. m. einf. Genauigkeit, konv. in ein 4-Byte String DOUBLE Fließkommaz. m. dopp. Genauigkeit, konv. in 8 Byte-String Standardmäßig enthält der String die Zahl im Little-Endian-Format (d. h. das niederwertigste Byte ist das erste in der Zeichenfolge). Setzen des dritten Parameters auf „BIG“ gibt die Zeichenfolge im Big-Endian-Format zurück (d. h. das most significant Byte ist das erste in der Zeichenfolge) Im Fall von Ganzzahl-Konvertierungen wird ein Fehler generiert wenn „value“ nicht in ‘type’ passt z. B. ein Versuch den Wert 400 in einem INT8 zu speichern. Diese Funktion erleichtert das Vorbereiten von Daten für eine effiziente Binärdatei-I/O oder zum Aufbereiten von Zahlen für die Ausgabe an Sensoren und speichern im Flash-Speicher. Siehe auch die Funktion STR2BIN

BOUND(array() [,dimension]	Dies gibt die Obergrenze des Arrays für die angeforderte Dimension zurück. Die Dimension ist standardmäßig eins, wenn nicht angegeben. Der Wert 0 gibt den aktuellen Wert von OPTION BASE zurück. Nicht verwendete Dimensionen geben den Wert Null zurück. Beispielsweise: DIM myarray(44,45) BOUND(myarray(),2) gibt 45 zurück
CALL(userfunname$, [,userfunparameters,....])	Dies ist eine effiziente Möglichkeit benutzerdefinierte Funktionen programmgesteuert aufzurufen. (Siehe auch den CALL-Befehl). In vielen Fällen kann es zur Beseitigung verwendet werden um komplexe SELECT- und IF THEN ELSEIF ENDIF-Klauseln zu eliminieren und wird auf eine viel effizientere Weise verarbeitet. „userfunname$“ kann eine beliebige Zeichenfolge, Variable oder Funktion sein, die in den Namen einer normalen Benutzerfunktion (kein eingebauter Befehl) aufgelöst wird. „userfunparameters“ sind die gleichen Parameter, die direkt zum Aufrufen der Funktion verwendet würden. Eine typische Verwendung für diesen Befehl könnte darin bestehen eine Art von Emulator zu schreiben, wo eine von vielen Funktionen in Abhängigkeit von einer Variablen aufgerufen werden. Es bietet auch eine Methode zum Übergeben eines Funktionsnamens an eine andere Unterprogramm oder Funktion als einer Variablen.
CHOICE(condition, ExpressionIfTrue, ExpressionIfFalse)	Mit dieser Funktion können Sie einfache Entweder/Oder-Auswahlen effizienter durchführen und schneller als die Verwendung von IF THEN ELSE ENDIF-Klauseln. Die Bedingung ist alles, was zu Nicht-Null (wahr) oder Null (falsch) aufgelöst wird. Die Ausdrücke sind alles, was Sie normalerweise einer Variablen oder zuweisen könnten in einem Befehl verwenden und können Ganzzahlen, Gleitkommazahlen oder Zeichenfolgen sein. Beispiele: PRINT CHOICE(1, "hallo","bye") gibt "Hallo" aus AUSWAHL DRUCKEN (0, "Hallo", "Tschüss") druckt "Tschüss" a=1 : b=1 : PRINT CHOICE (a=b, 4, 5) druckt 4
CHR$(number)	Gibt einen “1 Zeichen”-String zurück der aus dem Zeichen besteht dass dem binären ASCII-Code entspr. der durch das Argument 'Zahl' angegeben wird.
CINT(number)	Runden Sie Zahlen mit Bruchteilen auf oder ab auf die nächste ganze Zahl Zahl oder Ganzzahl. Beispielsweise wird 45,47 auf 45 gerundet 45,57 wird auf 46 gerundet -34,45 wird auf -34 gerundet -34,55 wird auf -35 gerundet Siehe auch INT() und FIX()..
COS(number)	Gibt den Kosinus des Arguments „number” im Bogenmaß zurück.

CTRLVAL(#ref)	Gibt den aktuellen Wert eines erweiterten Steuerelements zurück. '#ref' ist die Referenz des Steuerelements. Für Steuerelemente wie Kontrollkästchen oder Schalter wird die Zahl eins (true) sein, die angibt, dass das Steuerelement ausgewählt wurde vom Benutzer oder Null (false), wenn nicht. Für Steuerelemente, die eine Zahl enthalten (z. B. a SPINBOX) ist der Wert die Zahl normalerweise eine Gleitkommazahl. Für Steuerelemente die eine Zeichenfolge enthalten (z. B. TEXTBOX) ist der Wert ein String.
CWD$	Das aktuelle Arbeitsverzeichnis auf der SD-Karte. Ungültig für das exFAT-Format. Das Format ist: A:/dir1/dir2.
DATE$	Gibt das aktuelle Datum basierend auf der internen Uhr von MMBasic als Zeichenfolge zurück das Formular "TT-MM-JJJJ". Beispiel: „28.07.2012“. Die interne Uhr/Kalender verfolgt die Zeit und das Datum einschließlich Schaltjahre. Um das Datum einzustellen, verwenden Sie den Befehl DATE$ =
DATETIME$(n)	Gibt Datum und Uhrzeit zurück, die der Epochennummer n (Anzahl von Sekunden, die seit Mitternacht GMT am 1. Januar 1970 vergangen sind). Das Format der zurückgegebenen Zeichenfolge ist „tt-mm-jjjj hh:mm:ss“. Verwenden Sie den Text NOW um die aktuelle Datetime-Zeichenfolge zu erhalten, d.h. ? DATETIME$(NOW) DAY$(date$)
DAY$(date$)	Gibt den Wochentag für ein gegebenes Datum als String „Montag“, „Dienstag“ usw. Das Format für date$ ist „TT-MM-JJ“, „TT-MM-JJJJ“, oder "JJJJ-MM-TT". Verwenden Sie NOW um den Tag für das aktuelle Datum zu erhalten? DATETIME$(NOW)
DEG(radians)	Konvertiert Bogenmaß in Grad.

DIR$(fspec, type)
oder
DIR$(fspec)
oder
DIR$( )
	Durchsucht eine SD-Karte nach Dateien und gibt die Namen der gefundenen Einträge zurück.
	'fspec' ist eine Dateispezifikation, die die gleichen Platzhalter verwendet, wie sie von FILES verwendet werden. Z.B. "*.*" gibt alle Einträge zurück, "*.TXT" gibt Textdateien zurück. Beachten Sie, dass der Platzhalter *.* keine Dateien oder Ordner ohne Dateinamenerweiterung findet. 'type' ist der Typ des zurückzugebenden Eintrags und kann einer der folgenden sein:
	VOL Nur nach der Datenträgerbezeichnung suchen
	DIR Suche nur nach Verzeichnissen
	FILE Nur nach Dateien suchen (Standard, wenn „Typ“ nicht angegeben)
	Die Funktion gibt den ersten gefundenen Eintrag zurück. Die Abfrage weiterer Einträge verwendet die Funktion ohne Argumente. d.h. DIR$( ). Die Rückkehr eines Eine leeren String gibt an, dass keine weiteren Einträge zum Abrufen vorhanden sind. Dieses Beispiel druckt alle Dateien in einem Verzeichnis:
	`f$ = DIR$("*.*", FILE)`
	`DO WHILE f$ <> ""`
	`  PRINT f$`
	`  f$ = DIR$()`
	`LOOP`
	Sie müssen in das erforderliche Verzeichnis wechseln, bevor Sie diesen Befehl aufrufen.

DISTANCE(trigger, echo) oder DISTANCE(trig-echo)	Misst die Entfernung mit dem Ultraschall-Entfernungssensor HC-SR04 Sensor. Vierpolige Sensoren haben separate Trigger- und Echoanschlüsse. Trigger ist der I / O-Pin, der mit dem "Trig" -Eingang des Sensors verbunden ist, und "Echo" ist der Pin an den "Echo"-Ausgang des Sensors angeschlossen. Dreipolige Sensoren haben einen kombinierten Trigger- und Echoanschluss in diesem Fall muss nur einen I/O-Pin als Schnittstelle zum Sensor angegeben werden. Beachten Sie, dass alle mit dem HC-SR04 verwendeten E/A-Pins 5-V-fähig sein sollten da HC-SR04 mit 5 V betrieben wird. Die I/O-Pins werden automatisch konfiguriert und mehrere Sensoren können an verschiedenen I/O-Pins verwendet werden. Der zurückgegebene Wert ist die Entfernung in Zentimetern zum Ziel oder -1 falls kein Ziel erkannt wurde oder -2 bei einem Fehler z. B. Sensor nicht angeschlossen.
EOF([#]fnbr)	Gibt true zurück, wenn die Datei zuvor auf der SD-Karte für INPUT geöffnet wurde mit der Dateinummer '#fnbr' wird am Ende der Datei positioniert. Das # ist optional. Siehe auch OPEN, INPUT und LINE INPUT Befehle und die Funktion INPUT$.
EPOCH(DATETIME$)	Gibt die Epochennummer zurück (Anzahl der Sekunden, die seit dem GMT am 1. Januar 1970 für die angegebenen DATETIME$-String vergangen sind. Das Format für DATETIME$ ist „dd-mm-yyyy hh:mm:ss“, „dd-mm-yy hh:mm:ss“ oder „jjjj-mm-tt hh:mm:ss“. Verwenden Sie NOW um die Epoche zu erhalten Nummer für das aktuelle Datum und die Uhrzeit, z. B. PRINT EPOCH(NOW)
EVAL(string$)	Wertet 'string$' aus, als wäre es ein BASIC-Ausdruck und gibt das Ergebnis zurück. 'string$' kann eine Konstante, eine Variable oder ein String-Ausdruck sein. Der Ausdruck kann alle bekannten Operatoren, Funktionen, Variablen, Unterprogramme usw. verwenden die zum Zeitpunkt der Ausführung bekannt sind. Der zurückgegebene Wert ist eine Ganzzahl, Gleitkomma oder String abhängig vom Ergebnis der Auswertung. Zum Beispiel: S$ = "COS(RAD(30)) * 100" : PRINT EVAL(S$) Wird angezeigt: 86.6025
EXP(number)	Gibt den Exponentialwert von „Zahl“ zurück, d. h. ex, wobei x „Zahl“ ist.
FIX(number)	Kürzt eine Zahl auf eine ganze Zahl, indem der Dezimalpunkt und alle Zeichen rechts vom Dezimalpunkt weggelassen wird. Beispielsweise gibt 9,89 9 zurück und -2,11 gibt -2 zurück. Der Hauptunterschied zwischen FIX() und INT() besteht darin, dass FIX() ein echte Integer-Funktion liefrt d.h. gibt nicht die nächstniedrigere Zahl für negative Zahlen zurück wie INT(). Dieses Verhalten dient der Microsoft-Kompatibilität. Siehe auch CINT().
GPS() GPS(ALTITUDE)	Die GPS-Funktionen werden verwendet, um Daten von einem seriellen Kommunikationskanal zurückzugeben der als GPS geöffnet ist. Die Funktion GPS(VALID) sollte vor jeder dieser Funktionen überprüft werden werden verwendet, um sicherzustellen, dass der zurückgegebene Wert gültig ist. Gibt die aktuelle Höhe zurück (wenn der Satz GGA aktiviert ist).

GPS(DATE)	Gibt die normale Datumszeichenfolge korrigiert für die Ortszeit zurück, z. „01.12.2020“..
GPS(DOP)	Gibt den DOP-Wert (Dilution of Precision) zurück (wenn der Satz GGA aktiviert ist).
GPS(FIX)	Gibt Nicht-Null (true) zurück wenn das GPS einen Fix auf genügend Satelliten hat und hat gültige Daten produziert.
GPS(GEOID)	Gibt die Geoid-Ellipsoid-Trennung zurück (wenn der Satz GGA aktiviert ist).
GPS(LATITUDE)	Gibt den Breitengrad in Grad als Fließkommazahl zurück, Werte sind negativ für südlich des Äquators
GPS(LONGITUDE)	Gibt den Längengrad in Grad als Fließkommazahl zurück, Werte sind negativ für westlich des Meridians.
GPS(SATELLITES)	Gibt die Anzahl der sichtbaren Satelliten zurück (wenn der Satz GGA aktiviert ist).
GPS(SPEED)	Gibt die Geschwindigkeit über Grund in Knoten als Fließkommazahl zurück.
GPS(TIME)	Gibt die normale Zeitzeichenfolge korrigiert für die Ortszeit zurück, z. „12:09:33“.
GPS(TRACK)	Gibt den Track über dem Boden (in Grad wahr) als Fließkommazahl zurück.
GPS(VALID)	Gibt zurück: 0=ungültige Daten, 1=gültige Daten
HEX$(number [, chars])	Gibt eine Zeichenfolge zurück, die den Hexadezimalwert (Basis 16) für die „number“ angibt. 'chars' ist optional und gibt die Anzahl der Zeichen im String mit Null als führendes Füllzeichen
INKEY$	Prüft den Eingabepuffer der Konsole und ob ein oder mehrere Zeichen in der Warteschlange sind entfernt das erste Zeichen und gibt es als einzelnes Stringzeichen zurück. Wenn der Eingabepuffer leer ist kehrt diese Funktion sofort mit einem Leerstring zurück (d. h. "").
INPUT$(nbr, [#]fnbr)	Gibt eine Zeichenfolge zurück die aus „nbr“-Zeichen besteht, die aus einer Seriellem Kommunikationsport namens „fnbr“ gelesen wurde. Diese Funktion gibt so viele zurück Zeichen, wie im Empfangspuffer bis ‚nbr‘ warten. Wenn keine Zeichen warten wird ein Leerstring zurückgegeben. #0 kann verwendet werden, was sich auf den Eingabepuffer der Konsole bezieht. # ist optional. Siehe auch den OPEN-Befehl.

INSTR([start-position,] string-searched$, string-pattern$)	Gibt die Position zurück an der 'string-pattern$' in 'string-searched$' vorkommt, beginnend bei 'start-position'. Wenn 'start-position' nicht angegeben ist, wird es standardmäßig 1 verwendet. Sowohl die zurückgegebene Position als auch 'start-position' verwenden 1 für das erste Zeichen, 2 für die zweite usw. Die Funktion gibt Null zurück, wenn 'string-pattern$' nicht gefunden wird.
INT(number)	Kürzt einen Ausdruck auf die nächste ganze Zahl, die kleiner oder gleich Argument ist. Beispielsweise gibt 9,89 9 zurück und -2,11 gibt -3 zurück. Dieses Verhalten dient der Microsoft-Kompatibilität, die FIX()-Funktion bietet a echte Integer-Funktion. Siehe auch CINT().
LCASE$(string$)	Gibt „string$“ konvertiert in Kleinbuchstaben zurück.
LCOMPARE(array1%(), array2%())	Vergleichen Sie den Inhalt zweier langer String-Variablen array1%() und array2%(). Der zurückgegebene Wert ist eine Ganzzahl und ist -1, wenn array1%() kleiner als array2%() ist. Es ist null, wenn sie in Länge und Inhalt gleich sind, und +1, wenn array1%() ist größer als array2%(). Der Vergleich verwendet den ASCII-Zeichensatz und ist Case-sensitive.
LEFT$(string$, nbr)	Gibt eine Teilzeichenfolge von „string$“ mit „nbr“ von Zeichen beginnend links vom Anfang der Zeichenfolge zurück.
LEN(string$)	Gibt die Anzahl der Zeichen in 'string$' zurück
LGETBYTE(array%(), n)	Gibt den numerischen Wert des 'n'ten Bytes im enthaltenen LONGSTRING zurück 'array%()'. Diese Funktion berücksichtigt die Einstellung von OPTION BASE bei der Bestimmung welches Byte zurückgegeben werden soll.
LGETSTR$(array%(), start, length)	Gibt einen Teil eines langen Strings zurück, die in array%() als normaler MMBasic String gespeichert ist. Die Parameter start und length definieren den Teil des Strings, der zurückgegeben wird.
LINSTR(array%(), search$ [,start])	Gibt die Position eines Suchstrings in einem langen String zurück. Der zurückgegebene Wert ist eine Ganzzahl und gleich Null wenn der Teilstring nicht gefunden werden kann. array%() ist der String der durchsucht werden soll und muss eine lange Stringvariable sein. Search$ ist der Unterstring nach dem gesucht werden soll und es muss sich um einen normalen MMBasic-String oder -Ausdruck handeln (keine lange Zeichenfolge). Bei der Suche wird zwischen Groß- und Kleinschreibung unterschieden. Normalerweise beginnt die Suche beim ersten Zeichen in 'str', aber der optionale dritte Parameter ermöglicht die Angabe der Startposition der Suche.
LLEN(array%())	Gibt die Länge einer langen Zeichenfolge zurück, die in array%() gespeichert ist
LOC([#]fnbr)	Für einen seriellen Kommunikationsanschluss der als 'fnbr' geöffnet ist gibt diese Funktion die Anzahl der empfangenen Bytes im Empfangspuffer wieder darauf warten, gelesen zu werden. #0 kann verwendet werden, was sich auf den Eingabepuffer der Konsole bezieht. Das # ist optional

LOF([#]fnbr)	Für einen seriellen Kommunikationsanschluss, der als 'fnbr' geöffnet ist, gibt diese Funktion den im Sendepuffer verbleibenden Platz in Zeichen zurück. Beachten Sie dass MMBasic pausiert wenn der Puffer voll ist, wenn ein neuses Zeichen hinzugefügt wird und wartet bis Platz frei wird. Das # ist optional.
LOG(number)	Gibt den natürlichen Logarithmus des Arguments „Zahl“ zurück
MATH **Einfache Funktionen**	Die mathematische Funktion führt viele einfache mathematische Berechnungen durch die in Basic programmiert werden können aber es gibt Geschwindigkeitsvorteile gegenüber C wegen der Schleifen und es gibt den Vorteil dass sie einmal debuggt für alle da sind ohne hier das Rad neu zu erfinden.
MATH(ATAN3 x,y)	Gibt ATAN3 von x und y zurück
MATH(COSH a)	Gibt den hyperbolischen Kosinus von a zurück
MATH(LOG10 a)	Gibt den Logarithmus zur Basis 10 von a zurück
MATH(SINH a)	Gibt den hyperbolischen Sinus von a zurück
MATH(TANH a)	Gibt den hyperbolischen tan von a zurück
Einfache Statisik	
MATH(CHI a())	Gibt den Pearson-Chi-Quadrat-Wert des zweidimensionalen Arrays a()) zurück
MATH(CHI_p a())	Gibt die zugehörige Wahrscheinlichkeit in % des Chi-Quadrat-Werts von Pearson zurück des zweidimensionalen Arrays a())
MATH(CORREL a(), a())	Gibt den Pearson-Korrelationskoeffizienten zwischen den Arrays a() und b() zurück
MATH(MAX a() [,index%])	Gibt das Maximum aller Werte im Array a() zurück a() kann beliebige Anzahl Dimensionen haben. Wenn die Integer-Variable angegeben ist, wird sie mit dem Index des Maximalwerts im Array aktualisiert. Das ist nur auf eindimensionalen Arrays verfügbar
MATH(MEAN a())	Gibt den Durchschnitt aller Werte im Array a() zurück, a() kann eine beliebige Zahl Dimensionen haben
MATH(MEDIAN a())	Gibt den Median aller Werte im a()-Array zurück, a() kann eine beliebige Anzahl Dimensionen haben
MATH(MIN a(), [index%])	Gibt das Minimum aller Werte im a()-Array zurück, a() kann eine beliebige Zahl von Dimensionen haben. Wenn die Integer-Variable angegeben ist, wird sie mit dem Index des Maximalwerts im Array aktualisiert. Dies ist nur auf eindimensionalen Arrays verfügbar.

MATH(SD a())	Gibt die Standardabweichung aller Werte im a()-Array zurück, a() kann beliebig viele Dimensionen haben.
MATH(SUM a())	Gibt die Summe aller Werte im a()-Array zurück, a() kann eine beliebige Anzahl Dimensionen haben
Vektorarithmetik	
MATH(MAGNITUDE v())	Gibt die Größe des Vektors v() zurück. Der Vektor kann eine beliebige Anzahl von Elementen haben.
MATH(DOTPRODUCT v1(), v2())	Gibt das Skalarprodukt zweier Vektoren v1() und v2() zurück. Die Vektoren können beliebig viele Elemente hanen aber dieselbe Kardinalität !
Matrix Arithmetic	
MATH(M_DETERMINANT array!())	Gibt die Determinante des Arrays zurück. Das Array muss quadratisch sein
MAX(arg1 [, arg2 [, …]]) oder MIN(arg1 [, arg2 [, …]])	Gibt die maximale oder minimale Zahl in der Argumentliste zurück. Beachten Sie dass der Vergleich ein Gleitkomma-Vergleich ist (ganzzahlige Argumente werden in Gleitkomma umgewandelt) und Gleitkomma wird zurückgegeben.
MID$(string$, start) oder MID$(string$, start, nbr)	Gibt einen Teilstring von „string$“ zurück, beginnend bei „start“ für „nbr“ Zeichen. Das erste Zeichen in der Zeichenfolge ist die Nummer 1. Wenn „nbr“ weggelassen wird der zurückgegebene String bis zum Ende von „string$“ erstrecken.
MSGBOX (msg$, b1$ [,b2$ … b4$])	Diese Funktion zeigt ein Nachrichtenfeld mit eins bis vier berührungsempfindlichen Tasten auf dem Schirm an. Alle anderen Steuerelemente werden deaktiviert bis der Benutzer eine der Tasten berührt. Das Nachrichtenfeld wird dann gelöscht die vorherige Steuerelemente werden wiederhergestellt und die Funktion gibt die Nummer der berührten Schaltfläche zurück, die erste Schaltfläche ist die Nummer eins. 'msg$' ist die anzuzeigende Nachricht. Diese kann eine oder mehrere Tilden (~) enthalten die einen Zeilenumbruch anzeigen. Es können bis zu 10 Zeilen im Feld angezeigt werden 'b1$' ist die Beschriftung für die erste Schaltfläche, 'b2$' ist die Beschriftung für die zweite Schaltfläche usw. Mindestens eine Schaltfläche muss angegeben werden das Maximum sind vier. Alle Schaltflächen die nicht in der Argumentliste enthalten sind, werden nicht angezeigt.
OCT$(number [, chars])	Gibt eine Zeichenfolge zurück, die die oktale (Basis 8) Darstellung von „number“ angibt. 'chars' ist optional und gibt die Anzahl der Zeichen im String an mit der Null als führendes Füllzeichen
PEEK(BYTE addr%) oder	Gibt ein Byte oder ein Wort innerhalb des virtuellen PIC32-Speicherbereichs zurück. BYTE gibt das Byte (8 Bit) zurück, das sich bei 'addr%' befindet '

PEEK(SHORT addr%) oder PEEK(WORD addr%) oder PEEK(INTEGER addr%) oder PEEK(FLOAT addr%) oder PEEK(VARADDR var) oder PEEK(CFUNADDR cfun) oder PEEK(VAR var, ±offset) oder PEEK(VARTBL, ±offset) oder PEEK(PROGMEM, ±offset)	SHORT gibt die kurze Ganzzahl (16 Bit) zurück, die sich bei „addr%“ befindet WORD gibt das Wort (32 Bit) zurück, das sich bei „addr%“ befindet INTEGER gibt die Ganzzahl (64 Bit) zurück, die sich bei „addr%“ befindet FLOAT gibt die Gleitkommazahl (32-Bit) zurück, die sich bei „addr%“ befindet VARADDR gibt die Adresse (32-bits) der Variablen 'var' im Speicher zurück. Ein Array wird als var() definiert. CFUNADDR gibt die Adresse (32-bits) der CFunction 'cfun' im Speicher zurück. Diese Adresse kann weiter gegeben werden an eine andere CFunction die sie dann für einen gemeinsamen Prozess aufrufen kann. VAR, gibt ein Byte im Speicher zurück, das 'var' zugewiesen ist. Ein Array ist angegeben als var(). VARTBL, gibt ein Byte im Speicher zurück, das der Variablentabelle zugeordnet ist verwaltet von MMBasic. Beachten Sie dass nach dem Schlüsselwort ein Komma steht VARTBL. gibt ein Byte in dem dem Programm zugewiesenen Speicher zurück. Beachten Sie, dass hinter dem Schlüsselwort PROGMEM ein Komma steht. Beachten Sie, dass „addr%“ eine ganze Zahl sein sollte.
PI	Gibt den Wert von pi zurück.
PIN(pin)	Gibt den Wert auf dem externen I/O „Pin“ zurück. Null bedeutet digitales Low, 1 bedeutet digital High und analoge Eingänge geben die gemessene Spannung als Fließkommazahl zurück. Frequenzeingänge geben die Frequenz in Hz zurück. Eine Periodeneingabe gibt die Zeitspanne in Millisekunden seit dem Reset zurück (es wird bei der steigenden Flanke gezählt). Der Zähleingang kann auf Null zurückgesetzt werden indem Sie den Pin auf den Zähleingang zurücksetzen auch wenn er bereits so konfiguriert wurde. Diese Funktion gibt auch den Status eines als Ausgang konfigurierten Pins zurück. Siehe auch die Befehle SETPIN und PIN() . Siehe Kapitel „Verwenden die I/O-Pins" für eine allgemeine Beschreibung der Ein-/Ausgänge.
PIN(TEMP)	Gibt die Temperatur des RP2040-Chips zurück (siehe Datenblatt des RP2040 für die Details)
PIO (SHIFTCTRL push_threshold [,pull_threshold] [,autopush] [,autopull]) PIO (PINCTRL no_side_set_pins [,no_set_pins] [,no_out_pins] [,IN base] [,side_set_base] [,set_base])	Hilfsfunktion, um den Wert von shiftctrl für den INIT MACHINE Befehl zu berechnen Hilfsfunktion um den Wert von pinctrl für den INIT MACHINE Befehl zu berechnen

PIO (EXECCTRL jmp_pin ,wrap_target, wrap	Hilfsfunktion, um den Wert von execctrl für den INIT MACHINE Befehl zu berechnen
PIO (FDEBUG pio)	gibt den Wert des FSDEBUG-Registers für das angegebene pio zurück
PIO (FSTAT pio)	gibt den Wert des FSTAT-Registers für das angegebene pio zurück
PIO (FLEVEL pio	gibt den Wert des FLEVEL-Registers für das angegebene pio zurück
PORT(start, nbr [,start, nbr]…)	Gibt den Wert einer Reihe von I/O-Pins in einer Operation zurück. 'start' ist eine I/O-Pin-Nummer und der Wert wird als Bit 0 zurückgegeben. 'start'+1 wird als Bit 1 zurückgegeben werden, 'start'+2 wird als Bit 2 zurückgegeben und so weiter für 'nbr' Anzahl von Bits. Verwendete I/O-Pins müssen fortlaufend nummeriert sein und jeder I/O-Pin der ungültig ist oder nicht als Eingang konfiguriert führt zu einer Fehlermeldung. Das start/nbr-Paar kann bis zu 25 Mal wiederholt werden wenn zusätzliche Gruppen von Eingangspins hinzugefügt werden müssen. Diese Funktion gibt auch den Status eines als Ausgang konfigurierten Pins zurück. Es kann verwendet werden, um bequem mit parallelen Geräten zu kommunizieren wie z.B. Speicher Chips. Es kann eine beliebige Anzahl von I/O-Pins und daher Bits verwendet werden von 1 bis zur Anzahl der I/O-Pins auf dem Chip. Siehe den PORT-Befehl zur gleichzeitigen Ausgabe an eine Reihe von Pins.
PULSIN(pin, polarity) oder PULSIN(pin, polarity, t1) oder PULSIN(pin, polarity, t1, t2)	Misst die Breite eines Eingangsimpulses von 1 µs bis 1 Sekunde mit 0,1 µs Auflösung. 'pin' ist der I/O-Pin, der für die Messung verwendet wird, er muss vorher als digitaler Eingang konfiguriert werden. 'Polarität' ist die Art des zu messenden Impulses, wenn Null gibt die Funktion die Breite des nächsten negativen Impulses zurück, wenn ungleich Null wird der nächste positive Impuls gemessen. 't1' ist das Timeout der angewendet wird während auf das Eintreffen des Impulses gewartet wird, 't2' ist das Timeout das während der Pulsmessung verwendet wird. Beide sind in Mikrosekunden (µs) und sind optional. Wird 't2' weggelassen wird der Wert von 't1' für beide Timeouts verwendet. Wenn sowohl 't1' als auch 't2' weggelassen werden, werden die Timeouts auf 100000 d. h. 100 ms gesetzt. Diese Funktion gibt die Breite des Impulses in Mikrosekunden (µs) an oder -1 wenn ein Timeout aufgetreten ist. Die Messung ist auf ±0,5 % genau und ±0,5 µs Die Funktion führt dazu dass das laufende Programm angehalten wird während die Messung durchgeführt wird, auch Interrupts werden ignoriert.
RAD(degrees)	Konvertiert degrees ins Bogenmaß.
RGB(red, green, blue) oder RGB(shortcut)	Erzeugt einen RGB-True-Color-Wert. „Rot“, „Blau“ und „Grün“ repräsentieren die Intensität jeder Farbe. Ein Wert von Null steht für Schwarz und 255 für volle Intensität. 'shortcut' ermöglicht es gemeinsame Farben durch Benennung zu spezifizieren. Diese sind white, black, blue, green, cyan, red, magenta, yellow, brown, white, orange, pink, gold, salmon, beige, lightgrey and grey (oder USA: gray/lightgray). For example, RGB(red) or RGB(cyan). Zum Beispiel RGB(Rot) oder RGB(Cyan). Beachten Sie dass der zurückgegebene Wert eine Ganzzahl ist. Beim speichern sollte die Variable als Ganzzahl deklariert werden um die volle Genauigkeit zu erhalten

RIGHT$(string$, number-of-chars)	Gibt einen Teilstring von 'string$' mit 'number-of-chars' von rechts (Ende) des String zurück.
RND(number) oder RND	Gibt eine Pseudo-Zufallszahl im Bereich von 0 bis 0,999999 zurück. Der Der Wert „number“ wird ignoriert falls angegeben. Der RANDOMIZE-Befehl setzt den Zufallszahlengenerator neu.
SGN(number)	Gibt das Vorzeichen des Arguments „number“ zurück, +1 für positive Zahlen, 0 für 0 und -1 für negative Zahlen.
SIN(number)	Gibt den Sinus des Arguments „Zahl“ im Bogenmaß zurück.
SPACE$(number)	Gibt eine Zeichenfolge aus Leerzeichen mit einer Länge von 'number' zurück.
SPI (data) oder SPI2 (data)	Senden und empfangen Sie Daten über einen SPI-Kanal. Eine einzelne SPI-Transaktion sendet Daten und empfängt gleichzeitig Daten vom Slave. ‘data’ sind die zu sendenden Daten und die Funktion gibt die während der Transaktion erhaltenen Daten zurück. „data“ kann eine Ganzzahl oder ein Gleitkommawert oder eine Konstante sein.
SQR(number)	Gibt die Quadratwurzel des Arguments „number“ zurück.
STR$(number) oder STR$(number, m) oder STR$(number, m, n) oder STR$(number, m, n, c$)	Gibt eine Zeichenfolge in der Dezimaldarstellung (Basis 10) von „Zahl“ zurück. Wenn 'm' angegeben ist werden am Anfang der Nummer genügend Leerzeichen hinzugefügt sicherzustellen, dass die Anzahl der Zeichen vor dem Komma (inkl negatives oder positives Vorzeichen) sind mindestens 'm'-Zeichen. Wenn 'm' Null ist oder die Zahl mehr als 'm' signifikante Ziffern hat werden keine Füllzeichen hinzugefügt. Wenn 'm' negativ ist, wird positiven Zahlen das Pluszeichen vorangestellt und negative Zahlen das Minuszeichen. Wenn 'm' positiv ist wird nur das Minuszeichen verwendet. 'n' ist die Anzahl der Ziffern, die nach der Dezimalstelle benötigt werden. Wenn es Null ist wird der String ohne Dezimalpunkt zurückgegeben. Wenn es negativ ist wird die die Ausgabe im Exponentialformat mit einer Auflösung von 'n' Stellen verwendet. Wird 'n' nicht verwendet variieren die Anzahl der Dezimalstellen und das Ausgabeformat automatisch entspr. der Nummer. 'c$' ist ein String und wenn angegeben wird das erste Zeichen dieses Strings verwendet als Füllzeichen anstelle eines Leerzeichens (siehe das 'm'-Argument). Beispiele für die Formatierung: 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])	Gibt eine Zahl zurück, die der binären Darstellung in „string$“ entspricht. ‘Typ’ kann sein: INT64 konvertiert eine 8-Byte-Zeichenfolge die eine vorzeichenbehaftete 64-Bit-Ganzzahl darstellt in eine ganze Zahl UINT64 konvertiert eine 8-Byte-Zeichenfolge die eine vorzeichenlose 64-Bit-Ganzzahl darstellt in eine ganze Zahl INT32 konvertiert einen 4-Byte-String der eine vorzeichenbehaftete 32-Bit-Ganzzahl darstellt in eine ganze Zahl UINT32 konvertiert eine 4-Byte-Zeichenfolge die eine vorzeichenlose 32-Bit-Ganzzahl darstellt in eine ganze Zahl INT16 konvertiert einen 2-Byte-String der eine vorzeichenbehaftete 16-Bit-Ganzzahl darstellt in eine ganze Zahl UINT16 konvertiert eine 2-Byte-Zeichenfolge die eine vorzeichenlose 16-Bit-Ganzzahl darstellt in eine ganze Zahl INT8 konvertiert einen 1-Byte-String der eine vorzeichenbehaftete 8-Bit-Ganzzahl darstellt in eine Ganzzahl UINT8 konvertiert einen 1-Byte-String der eine vorzeichenlose 8-Bit-Ganzzahl darstellt in eine ganze Zahl SINGLE konvertiert eine 4-Byte-Zeichenfolge die Gleitkommazahlen mit einfacher Genauigkeit darstellt in eine Gleitkommazahl DOUBLE konvertiert eine 8-Byte-Zeichenfolge die Gleitkommazahlen mit einfacher Genauigkeit darstellt in eine Gleitkommazahl Standardmäßig muss der String die Zahl im Little-Endian-Format enthalten (d. h. das LSB ist das erste in der Zeichenfolge). Setzen des dritten Parameters zu „BIG“ interpretiert die Zeichenfolge im Big-Endian-Format (d. h. das MSB ist das erste in der Zeichenfolge). Diese Funktion erleichtert das Lesen von Daten aus Binärdateien, Sensorwerte zu interpretieren oder Binärdaten effizient aus Flash-Speichern zu lesen. Ein Fehler wird generiert wenn der String die falsche Länge für die angeforderte Konversion hat Siehe auch die Funktion BIN2STR$
STRING$(nbr, ascii) oder STRING$(nbr, string$)	Gibt eine 'nbr' Bytes lange Zeichenfolge zurück die entweder aus dem ersten Zeichen von string$ besteht oder dem Zeichen das den ASCII-Wert 'ascii' darstellt der eine Ganzzahl oder Gleitkommazahl ist im Bereich von 0 bis 255.
TAB(number)	Gibt Leerzeichen aus bis die durch 'number' angegebene Spalte auf der Konole erreicht ist.
TAN(number)	Gibt den Tangens des Arguments “number“ im Bogenmaß zurück.

TEMPR(pin)	Gibt die von einem DS18B20-Temperatursensor gemessene Temperatur zurück verbunden mit 'pin' (muss nicht konfiguriert werden). Der zurückgegebene Wert ist Grad C mit einer Standardauflösung von 0,25 °C. Wenn bei der Messung ein Fehler auftritt wird der Wert 1000 zurückgegeben. Die für die Gesamtmessung benötigte Zeit beträgt 200 ms und Interrupts während dieser Zeit werden ignoriert. Alternativ kann der Befehl TEMPR START verwendet werden um die Messung zu starten, das Programm kann während der Konvertierung andere Dinge tun. Wird diese Funktion aufgerufen wird der Messwert zurückgegeben wenn angenommen wird dass die Konversion beendet ist. Wenn dies nicht der Fall ist, wartet die Funktion den Rest der Konvertierungszeit ab bevor der Wert zurückgegeben wird. Der DS18B20 kann separat mit 3- 5V versorgt werden oder es kann parasitär gespeist werden vom PicoMite. Weitere Informationen finden Sie im Abschnitt Spezielle Hardwaregeräte.
TIME$	Gibt die aktuelle Uhrzeit als String zurück, basierend auf der internen Uhr von MMBasic im Format: "HH:MM:SS" in 24-Stunden-Notation. Beispiel: „14:30:00“. Um die aktuelle Zeit einzustellen, verwenden Sie den Befehl TIME$ = .
TIMER	Gibt die verstrichene Zeit in Millisekunden (z. B. 1/1000 Sekunde) seit einem Reset zurück. Der Timer wird beim Einschalten oder einem CPU-Neustart auf Null zurückgesetzt, man kann es auch zurücksetzen wenn man TIMER als Befehl verwendet. Der Timer zählt endlos weiter. Er verwendet eine 64-Bit-Zahl, ein Rollover des Zählers wird erst nach 200 Millionen Jahren erfolgen.
TOUCH(X) oder TOUCH(Y)	Gibt die X- bzw. Y-Koordinate des aktuell berührten Punkts auf dem LCD-Panel zurück. Wird das Panel nicht berührt gibt die Funktion -1 zurück.
UCASE$(string$)	Gibt „string$“ in Großbuchstaben umgewandelt zurück.
VAL(string$)	Gibt den numerischen Wert von „string$“ zurück. Wenn 'string$' eine ungültige Zahl ist gibt die Funktion Null zurück. Diese Funktion erkennt das &H-Präfix als Hexadezimalzahl, &O für oktal und &B für binär.

Veraltete Befehle und Funktionen

Diese Befehle und Funktionen sind hauptsächlich enthalten um beim Konvertieren von Programmen zu helfen, die für Microsoft BASIC geschrieben wurden. Für neue Programme sollten die entsprechenden modernen Befehle in MMBasic verwendet werden. Beachten Sie dass diese Befehle in Zukunft möglicherweise entfernt werden, um Speicher für andere Funktionen freizugeben.

GOSUB target	Initiiert einen Subroutinenaufruf zum Ziel, das eine Zeilennummer oder ein Label sein kann. Das Unterprogramm muss mit RETURN enden. Neue Programme sollten definierte Unterroutinen verwenden (z. B. SUB…END SUB).
IF condition THEN linenbr	Für die Microsoft-Kompatibilität wird ein GOTO angenommen wenn der THEN-Anweisung eine Zahl folgt. Ein Label ist in diesem Konstrukt ungültig. Neue Programme sollten verwenden: IF condition THEN GOTO linenbr | label
IRETURN	Kehrt von einem Interrupt zurück wenn das Interrupt-Ziel eine Zeilennummer war oder ein Label. Neue Programme sollten ein benutzerdefiniertes Unterprogramm als Interruptziel verwenden. In diesem Fall bewirkt END SUB oder EXIT SUB einen Rücksprung vom Interrupt.
ON nbr GOTO | GOSUB target[,target, target,...]	ON verzweigt entweder (GOTO) oder ruft ein Unterprogramm (GOSUB) auf basierend auf dem gerundeten Wert von 'nbr'; bei 1 wird das erste Ziel aufgerufen, bei 2 das zweite Ziel usw. Ziel kann eine Zeilennummer oder ein Label sein. Neue Programme sollten SELECT CASE verwenden.
POS	Gibt für die Konsole die aktuelle Cursorposition in der Zeile in Zeichen zurück.
RETURN	RETURN schließt eine von GOSUB aufgerufene Subroutine ab und kehrt zu zurück Anweisung nach dem GOSUB.

Anhang A
Serielle Kommunikation

Auf dem PicoMite stehen zwei serielle Schnittstellen für asynchrone serielle Kommunikation zur Verfügung. Sie sind beschriftet COM1: und COM2:.

I/O Pins

Bevor eine serielle Schnittstelle verwendet werden kann, müssen zunächst die I/O-Pins mit folgendem Befehl definiert werden

Kanal (als COM1 bezeichnet):

SETPIN rx, tx, COM1 gültige Pins RX: GP1, GP13 oder GP17
TX: GP0, GP12, GP16 oder GP28

Und der folgende Befehl für den zweiten Kanal (als COM2 bezeichnet):

SETPIN rx, tx, COM2
Gültige Pins RX: GP5, GP9 oder GP21
TX: GP4, GP8 oder GP20

TX = Sendedaten PicoMite / RX = Empfangsdaten.

Die Signalpolarität ist Standard für Geräte, die mit TTL-Spannungen betrieben werden. Leerlauf ist High, das Startbit ist Spannung Low, Daten verwenden eine hohe Spannung für logisch 1 und das Stoppbit ist Spannung hoch. Diese Signalpegel ermöglichen es Ihnen direkt an Geräte wie GPS-Module anschließen (die im Allgemeinen TTL-Spannungspegel verwenden).).

Befehle

Nach dem Öffnen hat die serielle Schnittstelle eine zugehörige Dateinummer und Sie können alle funktionierenden Befehle verwenden mit einer Dateinummer zum Lesen und Schreiben . Die serielle Schnittstelle kann mit CLOSE geschlossen werden.

Das Folgende ist ein Beispiel:

```
SETPIN GP13, GP16, COM1     ' weist die E/A-Pins für die erste serielle Schnittstelle zu
OPEN "COM1:4800" AS #5      ' öffnet die erste serielle Schnittstelle mit 4800  Baud
PRINT #5, "Hello"           ' sende "Hello"
dat$ = INPUT$(20, #5)       ' bis zu 20 Zeichen einlesen
CLOSE #5                    ' schliessen
```

Der Befehl OPEN

Eine serielle Schnittstelle wird mit dem Befehl geöffnet:

```
OPEN comspec$ AS #fnbr
```

„fnbr“ ist die zu verwendende Dateinummer. Es muss im Bereich von 1 bis 10 liegen. Das # ist optional.

„comspec$“ ist die Kommunikationsspezifikation und ist eine Zeichenfolge (es kann eine Zeichenfolgenvariable sein), die die Seriennummer angibt

zu öffnender Port und optionale Parameter. Die Voreinstellung ist 9600 Baud, 8 Datenbits, keine Parität und ein Stoppbit.

Es hat die Form "COMn: baud, buf, int, int-trigger, EVEN, ODD, S2, 7BIT", wobei:

- „n“ ist die serielle Anschlussnummer für entweder COM1: oder COM2:.
- „Baud“ ist die Baudrate. Dies kann eine beliebige Zahl zwischen 1200 und weit über 1000000 sein. Der Standardwert ist 9600.
- „buf“ ist die Größe des Empfangspuffers in Byte (Standardgröße ist 256). Der Sendepuffer ist auf 256 Byte festgelegt.
- „int“ ist eine Interrupt-Subroutine, die aufgerufen wird, wenn die serielle Schnittstelle einige Daten empfangen hat.
- „int-trigger“ ist die Anzahl der empfangenen Zeichen, die einen Interrupt auslösen.

Alle Parameter außer dem Namen der seriellen Schnittstelle (COMn:) sind optional. Wenn irgendein Parameter weggelassen wird, dann müssen die folgenden Parameter ebenfalls weggelassen werden und es werden die Standardwerte verwendet. Am Ende von 'comspec$' können vier Optionen hinzugefügt werden. Diese sind:

- „S2“ gibt an, dass nach jedem übertragenen Zeichen zwei Stoppbits gesendet werden.
- EVEN gibt an, dass ein gerades Paritätsbit angewendet wird, dies führt zu einer 9-Bit-Übertragung, es sei denn, 7BIT ist es einstellen.
- ODD gibt an, dass ein ungerades Paritätsbit angewendet wird, dies führt zu einer 9-Bit-Übertragung, es sei denn, 7BIT ist gesetzt
- 7BIT gibt an, dass 7 Datenbits vorhanden sind. Dies wird normalerweise mit GERADE oder UNGERADE verwendet

Beispiele

```
Öffnen einer seriellen Schnittstelle mit allen Standardeinstellungen:
ÖFFNEN SIE „COM1:“ ALS #2
Öffnen einer seriellen Schnittstelle, die nur die Baudrate (4800 Bit pro Sekunde)
angibt:
ÖFFNE "COM1:4800" ALS #1
Öffnen einer seriellen Schnittstelle mit Angabe der Baudrate (9600 Bit pro
Sekunde) und Empfangspuffergröße (1 KB):
ÖFFNE "COM2:9600, 1024" AS #8
Dasselbe wie oben, aber mit zwei aktivierten Stoppbits:
ÖFFNE "COM2:9600, 1024, S2" AS #8
Ein Beispiel, das alles spezifiziert, einschließlich eines Interrupts, eines
Interrupt-Levels und zwei Stoppbits:
OPEN "COM2:19200, 1024, ComIntLabel, 256, S2" AS #5
```

Lesen und Schreiben

Sobald eine serielle Schnittstelle geöffnet wurde, können Sie jeden Befehl oder jede Funktion verwenden, die eine Dateinummer für Lese-/Schreibzugriffe an den Port verwendet . Von der seriellen Schnittstelle empfangene Daten werden von MMBasic automatisch im Speicher gepuffert bis es vom Programm gelesen wird, und die INPUT$()-Funktion ist der bequemste Weg, dies zu tun. Bei Verwendung der Funktion INPUT$() ist die angegebene Anzahl von Zeichen die maximale Anzahl von Zeichen

zurückgegeben, aber es könnte weniger sein, wenn weniger Zeichen im Empfangspuffer sind. Tatsächlich wird die INPUT$()-Funktion sofort einen leeren String zurückgeben wenn im Empfangspuffer keine Zeichen verfügbar sind. Die Funktion LOC() ist ebenfalls praktisch; Es gibt die Anzahl der Zeichen zurück, die im Empfangspuffer warten (d.h. die maximale Anzahl an Zeichen, die von der Funktion INPUT$() abgerufen werden können). Beachten Sie, dass wenn der Empfangspuffer “überläuft” verwirft die serielle Schnittstelle automatisch die ältesten Daten, um Platz für die neuen zu schaffen

Der PRINT-Befehl wird zum Ausgeben an eine serielle Schnittstelle verwendet, und alle zu sendenden Daten werden in einem Zwischenspeicher gehalten während die serielle Schnittstelle sendet. Das bedeutet dass MMBasic mit der Ausführung der Befehle nach dem PRINT-Befehl, fortfährt während die Daten übertragen werden. Die einzige Ausnahme ist, wenn der Ausgangspuffer voll ist, in diesem Fall wird MMBasic anhalten und warten, bis genügend Speicherplatz vorhanden ist, bevor es fortfährt. Die LOF() Funktion gibt den im Übertragungspuffer verbleibenden Speicherplatz zurück, und Sie können dies verwenden, um eine verzögerte Programmverarbeitung zu vermeiden während gewartet wird, dass Speicherplatz im Puffer frei wird.

Wenn Sie sicher sein wollen, dass alle Daten gesendet wurden (vielleicht weil Sie die Antwort vom

entfernten Gerät lesen wollen) sollten Sie warten, bis die LOF()-Funktion 256 anzeigt (die Übertragungspuffergröße) d.h. es gibt nichts mehr zu senden.

Serielle Ports können mit dem CLOSE-Befehl geschlossen werden. Dieser wartet darauf, dass der Sendepuffer dann geleert wird

Geben Sie den von den Puffern verwendeten Speicher frei und brechen Sie den Interrupt ab (falls gesetzt). Eine serielle Schnittstelle ist ebenfalls automatisch vorhanden

geschlossen, wenn Befehle wie RUN und NEW ausgegeben werden.

Interrupts

Die Interrupt-Subroutine (falls angegeben) funktioniert genauso wie ein allgemeiner Interrupt auf einem externen I/O-Pin (siehe Kapitel "Verwendung der I/O-Pins").

Wenn Sie Interrupts verwenden, müssen Sie sich darüber im Klaren sein, dass es einige Zeit dauern wird, bis MMBasic auf den Interrupt reagiert und es könnten zwischenzeitlich mehr Zeichen angekommen sein, besonders bei hohen Baudraten. Wenn Sie den Interruptlevel mit 200 Zeichen angeben und einen Buffer von 256 Zeichen dann ganz einfach den Buffer zu dem Zeitpunkt übergelaufen sein, zu dem die Interrupt-Subroutine die Daten lesen kann. In diesem Fall sollte der Puffer auf 512 Zeichen oder mehr erhöht werden.

Anhang B
I²C Kommunikation

Der PicoMite implementiert zwei I2C-Kanäle. Sie können im Master- oder Slave-Modus arbeiten..

I/O Pins

Bevor die I2C-Schnittstelle verwendet werden kann, müssen zunächst die I/O-Pins mit dem folgenden Befehl definiert werden

Kanal (als I2C bezeichnet):

SETPIN sda, scl, I2C gültige Pins sind SDA: GP0, GP4, GP8, GP12, GP16, GP20 or GP28
SCL: GP1, GP5, GP9, GP13, GP17, GP21 or GP27

Und der folgende Befehl für den zweiten Kanal (als I2C2 bezeichnet):

SETPIN sda, scl, I2C2
Gültige Pins sind SDA: GP2, GP6, GP10, GP14, GP18, GP22 or GP26
SCL: GP3, GP7, GP11, GP15, GP19 or GP27

Wenn der I2C-Bus mit über 100 kHz betrieben wird, wird die Verkabelung zwischen den Geräten wichtig. Die Kabel sollten so kurz wie möglich sein um die Kapazität zu reduzieren und die Daten- und Taktleitungen sollten nicht nebeneinander verlaufen zusätzlich mit einem Erdungskabel dazwischen um Übersprechen zu reduzieren.

Wenn die Datenleitung bei hohem Takt instabil ist oder oder jittert kann die I2C Peripherie durcheinandergeraten und den Bus blockieren dadurch das der Takt auf Low gezogen wird. Wenn Sie die höheren Geschwindigkeiten nicht benötigen dann sind 100 kHz die erste Wahl.

I²C Master Befehle

Es gibt vier Befehle, die für den ersten Kanal (I2C) im Master-Modus wie folgt verwendet werden können.

Die Befehle für den zweiten Kanal (I2C2) sind identisch, außer dass der Befehl I2C2 ist.

I2C OPEN speed, timeout

Aktiviert das I2C-Modul im Master-Modus. Der I2C-Befehl bezieht sich auf Kanal 1 während sich der Befehl I2C2 auf Kanal 2 bezieht und dieselbe Syntax verwendet. „Geschwindigkeit“ ist die zu verwendende Taktgeschwindigkeit (in KHz) und muss entweder 100 oder 400 sein. „timeout“ ist ein Wert in Millisekunden, nach dem der Master sendet und empfängt Befehle werden unterbrochen, wenn sie nicht abgeschlossen sind. Der Mindestwert ist

100. Ein Wert von Null deaktiviert das Timeout (obwohl dies nicht empfohlen wird).

I2C WRITE addr, option, sendlen, senddata [,sendata ....]

Senden Sie Daten an das I2C-Slave-Gerät. Der I2C-Befehl bezieht sich auf Kanal 1, während der Befehl I2C2 bezieht sich auf Kanal 2 mit der gleichen Syntax.

„addr“ ist die I2C-Adresse des Slaves. „Option“ kann 0 für den normalen Betrieb oder 1 sein, um die Kontrolle über den Bus nach dem zu behalten

Befehl (bei Abschluss des Befehls wird keine Stoppbedingung gesendet)

„sendlen“ ist die Anzahl der zu sendenden Bytes. „senddata“ sind die zu sendenden Daten – diese können auf verschiedene Weise angegeben werden (alle data gesendet werden als Bytes mit einem Wert zwischen 0 und 255 gesendet):

• Die Daten können als einzelne Bytes auf der Kommandozeile übergeben werden.

Beispiel: I2C WRITE &H6F, 0, 3, &H23, &H43, &H25

• Die Daten können sich in einem eindimensionalen Array befinden, das mit leeren Klammern angegeben wird (d. h.

keine Maße). 'sendlen' Bytes des Arrays werden beginnend mit dem ersten gesendet

Element. Beispiel: I2C WRITE &H6F, 0, 3, ARRAY()

• Die Daten können eine String-Variable sein (keine Konstante).

Beispiel: I2C WRITE &H6F, 0, 3, STRING$

I2C READ addr, option, rcvlen, rcvbuf	Holen Sie sich Daten vom I2C-Slave-Gerät. Der I2C-Befehl bezieht sich dabei auf Kanal 1 der Befehl I2C2 bezieht sich auf Kanal 2 mit der gleichen Syntax. „addr“ ist die I2C-Adresse des Slaves. „Option“ kann 0 für den normalen Betrieb oder 1 sein, um die Kontrolle über den Bus nach dem zu behalten Befehl (bei Abschluss des Befehls wird keine Stoppbedingung gesendet) „rcvlen“ ist die Anzahl der zu empfangenden Bytes. ‘rcvbuf’ ist die Variable oder das Array, das zum Speichern der empfangenen Daten verwendet wird – dies kann sein: • Eine String-Variable. Bytes werden als aufeinanderfolgende Zeichen in der Zeichenfolge gespeichert. • Ein eindimensionales Array von Zahlen, die mit leeren Klammern angegeben werden. Erhaltene Bytes werden in aufeinanderfolgenden Elementen des Arrays gespeichert, beginnend mit dem ersten. Beispiel: I2C READ &H6F, 0, 3, ARRAY() • Eine normale numerische Variable (in diesem Fall muss rcvlen 1 sein).).
I2C CLOSE	Deaktiviert das Master-I2C-Modul und versetzt die I/O-Pins in einen "nicht konfigurierten" Zustand.

I²C Slave Befehle

I2C SLAVE OPEN addr, send_int, rcv_int	Aktiviert das I2C-Modul im Slave-Modus. Der I2C-Befehl bezieht sich auf Kanal 1 während sich der Befehl I2C2 auf Kanal 2 bezieht und dieselbe Syntax verwendet. ‚addr‘ ist die Slave-I2C-Adresse. „send_int“ ist die Subroutine, die aufgerufen werden soll, wenn das Modul erkannt hat, dass der Master Daten erwartet. ‘rcv_int ist die Subroutine, die aufgerufen werden soll, wenn das Modul Daten vom Master empfangen hat. Beachten Sie, dass dies beim ersten empfangenen Byte ausgelöst wird, sodass Ihr Programm möglicherweise wartet bis alle Daten empfangen wurden.
I2C SLAVE WRITE sendlen, senddata [,sendata ....]	Sendet Daten an den I2C-Master. Der I2C-Befehl bezieht sich auf Kanal 1, während der Befehl I2C2 sich auf Kanal 2 bezieht. Dieser Befehl sollte im Sende-Interrupt verwendet werden (d. h. in der Subroutine 'send_int' wenn der Master Daten angefordert hat). Alternativ kann in der ein Flag gesetzt werden in der Interrupt-Subroutine und der Befehl, der von der Hauptprogrammschleife aufgerufen wird, wenn das Flag gesetzt ist. ‘sendlen ist die Anzahl der zu sendenden Bytes. „senddata“ sind die zu sendenden Daten. Dies kann auf verschiedene Arten angegeben werden, siehe I2C WRITE-Befehle für Details.
I2C SLAVE READ rcvlen, rcvbuf, rcvd	Empfängt Daten vom I2C-Master. Der I2C-Befehl bezieht sich auf Kanal 1 während sich der Befehl I2C2 auf Kanal 2 bezieht und dieselbe Syntax verwendet. Dieser Befehl sollte im Empfangsinterrupt verwendet werden (dh in der Subroutine 'rcv_int' wenn der Master einige Daten gesendet hat). Alternativ kann im Empfang ein Flag gesetzt werden. Interrupt-Subroutine und der Befehl, der von der Hauptprogrammschleife aufgerufen wird, wenn das Flag gesetzt ist. „rcvlen“ ist die maximale Anzahl an zu empfangenden Bytes. ‚rcvbuf‘ ist die Variable zum Empfangen der Daten. Dies kann auf verschiedene Arten angegeben werden, Einzelheiten finden Sie in den I2C READ-Befehlen. ‘rcvd’ ist eine Variable, die nach Abschluss des Befehls die tatsächliche enthält Anzahl der empfangenen Bytes (die von „rcvlen“ abweichen können).

I2C SLAVE CLOSE Deaktiviert das Slave-I2C-Modul und setzt die externen E/A-Pins auf „nicht konfiguriert"-Zustand. Sie können dann mit SETPIN konfiguriert werden.

Fehlermeldungen

Nach einem I2C-Schreib- oder Lesevorgang wird die automatische Variable MM.I2C gesetzt, um das Ergebnis wie folgt anzuzeigen:

0 = Der Befehl wurde ohne Fehler ausgeführt.
1 = Eine NACK-Antwort empfangen
2 = Zeitüberschreitung des Befehls

7-Bit Addressierung

Die in diesen Befehlen verwendeten Standardadressen sind 7-Bit-Adressen (ohne das Lese-/Schreibbit). MMBasic fügt das Lese-/Schreibbit hinzu und manipuliert es während der Übertragung entsprechend. Einige Anbieterverwenden 8-Bit-Adressen die das Lese-/Schreibbit enthalten. Sie können feststellen, ob dies der Fall ist weil sie eine Adresse zum Schreiben an den Slave und eine andere zum Lesen bereitstellen. In diesen Situationen sollten Sie nur die obersten sieben Bits der Adresse verwenden. Zum Beispiel: Wenn die Leseadresse 9B ist (Hex) und die Schreibadresse ist 9A (Hex), dann erhalten Sie eine Adresse von 4D (Hex), wenn Sie nur die oberen sieben Bits verwenden.

Ein weiterer Hinweis, dass ein Anbieter 8-Bit-Adressen anstelle von 7-Bit-Adressen verwendet ist die Überprüfung des Adressbereichs. Alle 7-Bit-Adressen sollten im Bereich von 08 bis 77 (Hex) liegen. Wenn Ihre Slave-Adresse größer als dieser Bereich ist, dann Wahrscheinlich hat Ihr Anbieter eine 8-Bit-Adresse verwendet.

Beispiele

Ein Beispiel für eine einfache Kommunikation, bei der PicoMite der Master ist. Das folgende Programm liest und zeigt die aktuelle Uhrzeit (Stunden und Minuten) an, die von einer PCF8563-Echtzeituhr verwaltet wird die mit dem zweiten I2C-Kanal verbunden ist:

```
DIM AS INTEGER RData(2)              ' speichert die Empfangsdaten
SETPIN GP6, GP5, I2C2                ' weist I2C2 die I/O Pins zu
I2C2 OPEN 100, 1000                  ' öffnet den I2C Kanal
I2C2 WRITE &H51, 0, 1, 3             ' setzt das erste Register auf 3
I2C2 READ &H51, 0, 2, RData()        ' liest zwei Register
I2C2 CLOSE                           ' schliesst den den I2C Kanal
PRINT "Es ist " RData(1) ":" RData(0)
```

Dies ist ein Beispiel für die Kommunikation zwischen zwei PicoMites wobei einer Master und der andere Slave ist.

Master:

```
SETPIN GP2, GP3, i
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
```

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
```

Anhang C
1-Wire Kommunikation

Das 1-Wire-Protokoll wurde von Dallas Semiconductor entwickelt, um mit Chips über einen einzigen zu kommunizieren. Signalleitung. Diese Implementierung wurde von Gerard Sexton für MMBasic geschrieben. Es gibt drei Befehle, die Sie verwenden können:

ONEWIRE RESET pin	1-Wire-Bus zurücksetzen
ONEWIRE WRITE pin, flag, length, data [, data…]	Sende Bytes
ONEWIRE READ pin, flag, length, data [, data…]	Lese Bytes

Erläuterung:

pin - Der zu verwendende PicoMite I/O-Pin. Es kann jeder Pin sein, der für digitale E/A geeignet ist.

flag - Eine Kombination der folgenden Optionen:

1 – Sende Reset vor dem Befehl

2 - Sende Reset nach dem Befehl

4 – Sende / empfange nur ein Bit anstelle eines Datenbytes

8 - Rufen Sie nach dem Befehl einen starken Pullup auf (der Pin wird hoch gesetzt und Open Drain deaktiviert)

length – Länger der Sende- oder Empfangsdaten

data - Zu sendende Daten oder zu empfangende Variable.
Die Anzahl der Datenelemente muss mit dem Längenparameter übereinstimmen.

Die automatische Variable MM.ONEWIRE gibt True zurück, wenn ein Gerät gefunden wurde

Nachdem der Befehl ausgeführt wurde, wird der I/O-Pin in den nicht konfigurierten Zustand gesetzt asser Flag-Option 8 wird verwendet. Wenn ein Reset angefordert wird, gibt die automatische Variable MM.ONEWIRE True zurück sobald ein Gerät gefunden wurde. Dies wird mit dem Befehl OONEWIRE RESET und OONEWIRE READ und OONEWIRE WRITE Befehl auftreten wenn ein Reset angefordert wurde (Flag = 1 oder 2). Das 1-Wire-Protokoll wird häufig bei der Kommunikation mit dem Temperaturmesssensor DS18B20 und bitte beachten : MMBasic enthält die TEMPR()-Funktion, die eine bequeme direkte Methode zur Verfügung stellt Lesen der Temperatur eines DS18B20 ohne Verwendung dieser Funktionen.

Anhang D
SPI Schnittstelle

Das Kommunikationsprotokoll Serial Peripheral Interface (SPI) wird zum Senden und Empfangen von Daten verwendet integrierte Schaltkreise. Der PicoMite fungiert als Master d.h. er erzeugt den Takt.

I/O Pins

Bevor eine SPI-Schnittstelle verwendet werden kann, müssen die I/O-Pins für den Kanal wie folgt zugewiesen werden

Befehle. Für den ersten Kanal (als SPI bezeichnet) gilt:

SETPIN rx, tx, clk, SPI

Gültig sind	RX:	GP0, GP4, GP16 oder GP20
	TX:	GP3, GP7 oder GP19
	CLK:	GP2, GP6 oder GP18

Und der folgende Befehl für den zweiten Kanal (als SPI2 bezeichnet) lautet:

SETPIN rx, tx, clk, SPI2

Gültig sind	RX:	GP8, GP12 oder GP28
	TX:	GP11, GP15 oder GP27
	CLK:	GP10, GP14 oder GP26

TX sind die Sendedaten des PicoMite and RX die Empfangsdaten.

SPI öffnen

Um die SPI-Funktion nutzen zu können, muss zuerst der SPI-Kanal geöffnet werden.

Die Syntax zum Öffnen des ersten SPI-Kanals lautet (SPI2 = zweiter Kanal):

```
SPI OPEN speed, mode, bits
```

Erläuterung:

- ‘speed’ ist die Geschwindigkeit der Uhr. Es ist eine Zahl, die die Taktfrequenz in Hz darstellt.
- 'mode' ist eine einzelne Ziffer, die den Übertragungsmodus darstellt – siehe Übertragungsformat unten.
- 'bits' ist die Anzahl der zu sendenden/empfangenden Bits. Dies kann eine beliebige Zahl im Bereich von 4 bis 16 Bit sein.
- Es liegt in der Verantwortung des Programms, den CS (Chip Select)-Pin bei Bedarf separat zu manipulieren.

Format

Das höchstwertige Bit wird zuerst gesendet und empfangen. Das Format der Übertragung kann durch den 'Modus' festgelegt werden Modus 0 ist das gebräuchlichste Format.

Mode	Description	CPOL	CPHA
0	Takt ist aktiv High, Daten werden bei steigender Flanke erfasst und bei fallender Flanke ausgegeben.	0	0
1	Takt ist aktiv High, Daten werden bei der fallenden Flanke erfasst und bei der steigenden Flanke ausgegeben	0	1
2	Takt ist aktiv Low, Daten werden bei der fallenden Flanke erfasst und bei der steigenden Flanke ausgegeben	1	0
3	Takt ist aktiv Low, Daten werden bei der steigenden Flanke erfasst und bei der fallenden Flanke ausgegeben	1	1

Eine vollständige Erklärung finden Sie unter: http://www.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Standard Senden- Empfangen

Wenn der erste SPI-Kanal offen ist, können Daten mit der SPI-Funktion gesendet und empfangen werden (verwenden Sie SPI2 für den zweiten Kanal). Syntax:

```
received_data = SPI(data_to_send)
```

Beachten Sie, dass eine einzelne SPI-Transaktion Daten sendet und gleichzeitig Daten vom Slave empfängt.

„data_to_send“ sind die zu sendenden Daten und die Funktion gibt die während der Transaktion empfangenen Daten zurück. „data_to_send“ kann eine Ganzzahl oder eine Fließkommavariable oder eine Konstante sein.

Wenn Sie keine Daten senden möchten (d. h. nur empfangen möchten), kann für die eine beliebige Zahl (z. B. Null) verwendet werden

Daten zu senden. Wenn Sie die empfangenen Daten nicht verwenden möchten, können Sie sie ebenfalls einer Variablen zuweisen und ignorieren.

Größere Datenblöcke Senden/ Empfangen

Daten können auch in großen Mengen gesendet werden (verwenden Sie SPI2 für den zweiten Kanal):

```
SPI WRITE nbr, data1, data2, data3, … etc
```

oder

```
SPI WRITE nbr, string$
```

oder

```
SPI WRITE nbr, array()
```

Bei der ersten Methode ist 'nbr' die Anzahl der zu sendenden Datenelemente und die Daten sind die Ausdrücke in der Argumentliste (d. h. 'Daten1', Daten2' usw.). Die Daten können eine Ganzzahl oder eine Fließkommavariable oder eine Konstante sein. Bei der zweiten oder dritten oben aufgeführten Methode sind die zu sendenden Daten im 'String$' oder dem Inhalt von enthalten 'array()' (muss ein eindimensionales Array aus Ganzzahlen oder Gleitkommazahlen sein). Die Saitenlänge bzw die Größe des Arrays muss gleich oder größer als nbr sein. Alle vom Slave zurückgesendeten Daten werden verworfen.

Daten können auch in großen Mengen empfangen werden (benutzen Sie SPI2 für den zweiten Kanal):

```
SPI READ nbr, array()
```

Wobei 'nbr' die Anzahl der zu empfangenden Datenelemente ist und array() ein eindimensionales ganzzahliges Array ist, wobei die empfangene Datenelemente gespeichert werden. Dieser Befehl sendet beim Lesen der Daten vom Slave Nullen.

SPI Close

Bei Bedarf kann der erste SPI-Kanal wie folgt geschlossen werden (die I/O-Pins werden auf inaktiv gesetzt):

```
SPI CLOSE
```

Verwenden Sie SPI2 für den zweiten Kanal.

Beispiele

Das folgende Beispiel zeigt, wie der SPI-Port für allgemeine E/A verwendet wird. Es sendet einen Befehl 80 (hex) und empfängt zwei Bytes vom Slave-SPI-Gerät mit der Standard-Sende-/Empfangsfunktion:

```
PIN(10) = 1 : SETPIN 10, DOUT     ' Pin 10 wird als Freigabe verwendet
SETPIN GP20, GP4, GP1, SPI        ' I/O Pins zuweisen
SPI OPEN 5000000, 3, 8            ' Geschw.= 5MHz, Datenlänge 8 Bits
PIN(10) = 0                       ' Freigabe aktivieren (Activ Low)
junk = SPI(&H80)                  ' sende Befehl, ignoriere Rückgabe
byte1 = SPI(0)                    ' Hole erstes Byte vom Slave
byte2 = SPI(0)                    ' Hole 2.Byte vom Slave
PIN(10) = 1                       ' Slave abwählen
SPI CLOSE                         ' Kanal schließen
```

Das folgende ähnelt dem oben angegebenen Beispiel aber dieses Mal wird die Übertragung unter Verwendung des Bulk-Transfers durchgeführt:

```
OPTION BASE 1                     ' unser Array beginnt mit dem Index 1
DIM data%(2)                      ' definiert das Array zum Empfangen der Daten
SETPIN GP20, GP4, GP1, SPI        ' I/O pins zuweisen
PIN(10) = 1 : SETPIN 10, DOUT     ' Pin 10 wird als Freigabesignal verwendet
SPI OPEN 5000000, 3, 8            ' speed= 5MHz, 8 bits data
```

```
PIN(10) = 0                         ' 'Freigabeleitung aktivieren(active low)
SPI WRITE 1, &H80                   ' sende den Befehl
SPI READ 2, data%()                 ' zwei Bytes vom Slave erhalten
PIN(10) = 1                         ' Slave abwählen
SPI CLOSE                           ' und schließe den Kanal
```

Anhang E
Das PIO Programmierpaket

Die PIO-Programmierung wird Anfängern nicht empfohlen. Es erfordert einige Kenntnisse im Bereich Systemprogrammierung und dieser Abschnitt des Handbuchs kann nur eine vereinfachte Beschreibung des PIO geben Hardware und eine kurze Einführung in deren Programmierung. Der Leser sei auf das ausgezeichnete Raspberry Pi RP2040 Datenblatt verwiesen insbesondere das Kapitel 3 (ca. 70 Seiten!), für weitere Informationen und Programmierbeispiele. Es ist nicht notwendig über PIO-Programmierkenntnisse zu verfügen um MMBasic zu nutzen (tatsächlich hat zum Zeitpunkt des Erstellung dieses Dokuments nur PicoMite dieses PIO-System).

Einführung in PIO

Der im PicoMite verwendete RP2040-Chip enthält zwei PIO-Blöcke, ähnlich wie Cut-Down, hoch spezialisierte CPU-Kerne. Sie sind in der Lage, völlig unabhängig vom Hauptsystem und voneinander zu laufen. Sie können verwendet werden um beispielsweise serielle Daten-Interfaces mit sehr hoher Genauigkeit zu erstellen, obwohl sie keineswegs darauf beschränkt sind. Sie können schnell laufen, mit einem Durchsatz von bis zu 32 Bit pro Taktzyklus.
Im Folgenden wird ein einzelner PIO-Block beschrieben sofern nicht anders angegeben. Der andere ist natürlich identisch. MMBasic bezeichnet sie in Übereinstimmung mit der Raspberry Pi-Dokumentation als PIO0 und PIO1. Entschuldigung wenn das Folgende ziemlich komplex erscheint, aber es ist nicht möglich, PIO-Programme ohne zu schreiben zu wissen, wie es funktioniert. Ein einzelner PIO-Block hat vier unabhängige “state machines”. Alle vier teilen sich einen einzigen 32 Befehle-Programmbereich des Flash-Speichers. Dieser Speicher hat nur Schreibzugriff aus dem Hauptsystem, hat aber vier Leseports einen für jede Zustandsmaschine, so dass jeder unabhängig darauf mit eigener Geschwindigkeit zugreifen kann. Jede Zustandsmaschine hat ihren eigenen Programmzähler. Jede Zustandsmaschine hat auch zwei 32-Bit-"Notizblock"-Register X und Y die als temporäre Datenspeicher verwendet werden. Der Zugriff auf I/O-Pins erfolgt über ein Input/Output-Mapping-Modul das auf 32 Pins zugreifen kann was jedoch beim Pico auf 30 begrenzt ist.
Alle Zustandsmaschinen können unabhängig und gleichzeitig auf alle Pins zugreifen. Das System kann Daten in das Eingangsende eines 32 Bit breiten TX-FIFO-Puffers mit 4 Wörtern schreiben. Die state machine kann dann Pull verwenden um das Ausgangswort des FIFO in das OSR (Output Shift Register) zu verschieben. Es kann auch verwendet werden, um jeweils 1-32 Bits vom OSR in das Ausgangs-Zuordnungsmodul oder andere Richtungen zu verschieben. AUTOPULL kann verwendet werden, um Daten automatisch zu holen bis das TX-FIFO leer ist oder einen voreingestellten Pegel erreicht. Das System kann Daten vom Ausgangsende eines 32 Bit breiten RX-FIFO-Puffers mit 4 Wörtern lesen. Die ste machine kann verwendet werden um 1-32 Datenbits gleichzeitig vom Eingangs-Zuordnungsmodul in das ISR (Eingangsschieberegister) zu verschieben. Es kann auch Push verwenden, um den Inhalt des ISR in den FIFO zu verschieben. AUTOPUSH kann verwendet werden, um Daten automatisch zu verschieben, bis das RX-FIFO voll ist oder eine Voreinstellung erreicht eben.
Die FIFO-Puffer können rekonfiguriert werden, um einen 8-Wort-32-Bit-FIFO in einer einzigen Richtung zu bilden. Die Puffer ermöglichen die Weiterleitung von Daten zu und von den Zustandsmaschinen ohne dass System oder die Zustandsmaschine aufeinander warten müssen.
Jeder der vier Zustandsmaschinen im PIO sind vier Register zugeordnet:

• CLKDIV ist der Taktteiler, der einen 16-Bit-Ganzzahlteiler und einen 8-Bit-Fractional-Teiler hat. Dies legt fest wie schnell die Zustandsmaschine läuft. Es dividiert den Systemtakt herunter.
• EXECCTRL enthält Informationen die die Übersetzung und Ausführung des Programmspeicher steuern
• SHIFTCTRL steuert die Anordnung und Verwendung der Schieberegister
• PINCTRL steuert, welche GPIO-Pins wie verwendet werden.

Die vier Zustandsmaschinen eines PIO haben gemeinsamen Zugriff auf seinen Block von 8 Interrupt-Flags. Jede state machine kann jedes Flag verwenden. Sie können eingestellt, zurückgesetzt oder auf ihre Änderung warten. Auf diese Weise können sie bei Bedarf synchron laufen. Die unteren vier Flaggen sind auch von und zur Hauptsystem zugänglich damit der PIO gesteuert werden kann oder Interrupts zurückgeben kann.

PIO Operation

Bei jedem Taktzyklus (Geschwindigkeit bestimmt durch den Inhalt von CLKDIV) liest jede Zustandsmaschine und führt eine Anweisung aus. Alle Anweisungen benötigen einen Taktzyklus (sofern sie nicht absichtlich durch zB. ein Wait-Befehl) zur Ausführung so dass dies ein sehr genaues Timing ergibt. Es gibt keine Teile des Programms, die für irgendeine Zustandsmaschine spezifisch sind. Die Zustandsmaschinen lesen eine Anweisung und vervollständigen sie dann mit Informationen aus ihren einzelnen Registern bevor sie ausgeführt werden. z.B. Der Programmbereich kann eine Anweisung wie "jmp pin 5" enthalten, die zur Adresse 5 springen würde
wenn Pin "Pin" auf High geht aber welcher "Pin" wird durch den Inhalt des EXECCTRL-Registers bestimmt.
EXECCTRL enthält einige wichtige Daten. Nur zum Beispiel:
JMP_PIN ist die PIN-Nummer, die verwendet wird, um einen JMP-PIN-Befehl auszulösen.
• WRAP_BOTTOM ist die Low-End-Programmadresse, die von dieser Zustandsmaschine verwendet wird. Der Standardwert ist 0, der Beginn des Programmbereichs.
• WRAP_TOP ist das obere Ende – wenn die Verarbeitung diese Adresse erreicht wird zu WRAP_BOTTOM gesprungen. Der Standardwert ist 31, die letzte Adresse des Programmbereichs.
WRAP_BOTTOM und WRAP_TOP werden im Assembler mit .wrap target und .wrap geändert.
Beachten Sie, dass Wrap so wirkt, als ob das Ende des Programmbereichs erreicht wurde und der Programmzähler eine Schleife zum Ziel macht. Es ist keine JMP-Anweisung, es ist eine echte definierte Schleife die in EXECCTRL definiert ist sie erfordert keinen Taktzyklus oder eine Anweisung im Programmspeicher.
Da die Zustandsmaschinen unabhängig sind können Sie WRAP_BOTTOM und WRAP_TOP so einrichten, dass sie eigene oder überlappende Beriche des Programms lesen.
Es machr absolut keinen Unterschied zum Inhalt des Programmbereichs. Nicht nur das, wenn die WRAP_TOP-Adresse eine JMP-Anweisung enthält, dann das hat Vorrang, sodass einige sehr komplexe Schleifen möglich sind.
Das PINCTRL-Register enthält Daten, die festlegen, auf welche Pins durch die state machine

zugegriffen wird z.B.

OUT_BASE setzt den ersten Ausgangspin eines Blocks

OUT_COUNT gibt an wie viele Pins ab OUT_BASE aufwärts verwendet werden (0 bis 32).).

PIO Programming

Ein PIO hat neun mögliche Programmierbefehle, aber es gibt viele Variationen.
Beispielsweise kann Mov bis zu 8 Quellen, 8 Ziele, 3 Verarbeitungsvorgänge während des Kopierens haben mit optionalen Delay- und/oder Side-Set-Operationen!

• Jmp - Springt zu einer absoluten Adresse im Programmspeicher, wenn eine Bedingung wahr ist (oder sofort).
• Wait - Hält den Betrieb der state machine an bis eine Bedingung wahr ist.
• In - Verschiebe eine Anzahl von Bits von einer Quelle in ISR.
• Out - Verschiebt eine Anzahl von Bits aus dem OSR zu einem Ziel.
• Push - Schiebt den Inhalt des ISR als einzelnes 32-Bit-Wort in das RX-FIFO.
• Pull – Lädt ein 32-Bit-Wort aus dem TX-FIFO in das OSR.
• Mov - Datum von einer Quelle zu einem Ziel kopieren.
• Irq - Setzt oder löscht ein Interrupt-Flag.
• Set - Daten sofort an ein festgelegte Ziel schreiben.

Die Instruktionen sind alle 16-Bit und enthalten sowohl die Anweisung als auch alle damit verbundenen Daten. Alle Befehle arbeiten im 1-Taktzyklus aber es ist möglich eine Verzögerung von mehreren Leerlaufzyklen zwischen zwei Anweisungen zu realisieren.
Zusätzlich gibt es eine Einrichtung namens “side-set” die es ermöglicht einen Wert in vordefinierte Ausgangs-PINs schreiben während eine Anweisung aus dem Speicher gelesen wird. Das ist für das Programm transparent.
MMBasic bietet eine Reihe von Befehlen zur Steuerung des PIO-Betriebs:

- PIO INIT MACHINE pio%, statemachine%, clockspeed [,pinctrl] [,execctrl] [,shiftctrl]

[,startinstruction]
Initialisiert eine state machine auf einem PIO.

- PIO EXECUTE pio, state_machine, instruction%

Sofortige Ausführung einer Anweisung auf einer angegebenen PIO und state machine. Das wird sofort das Programm stoppen das aktuell dort läuft und das bei “instruction%” neu starten

- PIO WRITE pio, state_machine, count, data0 [,data1....]

Schreiben Sie eine Reihe von Datenelementen aus einem Integer-Array in einen angegebenen PIO und eine Zustandsmaschine.

- PIO READ pio, state_machine, count, data%()

Liest eine Reihe von Datenelementen aus einem angegebenen PIO und einer Zustandsmaschine in ein Integer-Array.

PIO START pio, statemachine
Starten Sie eine bestimmte Zustandsmaschine, die auf einem PIO ausgeführt wird. Das Programm startet

mit der in PIO angegebenen Anweisung INIT MACHINE 'Startinstruction'.

- PIO STOP pio, statemachine

Stoppen Sie eine angegebene Zustandsmaschine auf einem PIO.

- PIO CLEAR pio

Stoppt alle Zustandsmaschinen auf einem PIO und setzt PINCTRL, EXECCTR und SHIFTCTRL auf ihre Standardwerte für alle Zustandsmaschinen auf diesem PIO.

- PIO PROGRAM LINE pio, line, instruction

Programmiere nur die spezifizierte Zeile im PIO

- Zusätzlich gibt es einige PIO-Hilfsfunktionen, die verwendet werden, um die Werte von PIO-Registern zu berechnen oder zu lesen:

- PIO (SHIFTCTRL push_threshold [,pull_threshold] [,autopush] [,autopull])

Gibt den berechneten Wert für SHIFTCTRL zurück, wenn es mit den angegebenen Optionen verwendet wird.

- PIO (PINCTRL no_side_set_pins [,no_set_pins] [,no_out_pins] [,IN base] [,side_set_base] [,set_base])

Gibt den berechneten Wert für PINCTRL zurück, wenn er mit den angegebenen Optionen verwendet wird.

- PIO (EXECCTRL jmp_pin ,wrap_target, wrap)

Gibt den berechneten Wert für EXECCTRL zurück, wenn es mit den angegebenen Optionen verwendet wird.

- PIO (FDEBUG pio)

Gibt den Inhalt des FIFO-Debug-Registers zurück.

- PIO (FSTAT pio)

Gibt den Inhalt des FIFO-Statusregisters zurück.

- PIO (FLEVEL pio)

Gibt die Füllstände der FIFO-Puffer zurück.

- In MMBasic wird die PIO-Programmierung mit dem Befehl initiiert:

PIO INIT MACHINE pio%, statemachine%, clockspeed [,pinctrl] [,execctrl] [,shiftctrl]

Beispiel:

Dadurch wird an Pin-1 eine Rechteckwelle mit 1/4 der angeforderten Taktfrequenz erzeugt. Das Programm selbst ist gespeichert in %(). Das PIO-Programm 0,a%() richtet die Zustandsmaschine 0 ein, um das Programm auszuführen.

```
Dim a%(7)=(&H0001E000E101E081,0,0,0,0,0,0,0)  'Das eigentliche Programm
SetPin 1,pio0                  'Pin 1 wird pio0 zugewiesen
PIO program 0,a%()             'pio0  führt das in a%() gespeicherte Programm aus
PIO init machine 0,0,100000    'Initialialisiere pio0, state machine 0 , f= 100000Hz
                               ' Achtung: PINCTRL, EXECCTRL & SHIFTCTRL werden nicht
                               'geändert.
PIO start 0,0                  'Start pio0, state machine 0
```

Das in a%() gezeigte Programm kann von Hand zusammengestellt oder der PASM-Assembler verwendet werden. Das ist ein MMBasic-Programm mit dem der Benutzer Assembler-Anweisungen als DATA-Anweisungen eingeben kann. PASM erzeugt dann das erforderliche Programm-Array und setzt das Wrap-Ziel und Wrap-Adressen EXECCTRL falls diese im Programm verwendet wurden.

Das PIO Programmierpaket [z. Zt. in Erprobung]

Dies ist ein Paket von MMBasic-Programmen und Unterroutinen zur Unterstützung bei der Programmierung der PIO-Geräte. Die Subroutinen sollen in das Programm des Benutzers eingebaut werden.
PASM (PIO Assembler) ist eine Assembler-Unterroutine zur Erzeugung von PIO-Code aus einer Liste von DATEN Aussagen.
PREVAS (PIO Reverse-Assembler) ist ein eigenständiger einfacher Disassembler, der den Code lesen kann der mit PASM erstellt wurde.
PREDIT (PIO Register Editor) ermöglicht dem Benutzer die state machine-Register mit einer einfachen Konsolenschnittstelle zu bearbeiten. Es ist ein eigenständiges Programm und gibt dem Benutzer die Hex-Werte für die Register wenn die ausgewählten Felder eingegeben werden. Alle vier Zustandsmaschinen sind gleichzeitig auf dem Bildschirm zu sehen.
Die oben genannten Programme können alle auf ein gemeinsames *.PIO-Datendateiformat auf einer SD-Karte zugreifen. PASM tut dies über eine Subroutine namens "fileman", die das Auflisten, Speichern und Laden von Verzeichnissen ermöglicht..

Einführung in PASM

Der PASM-Assembler ist ein gut kommentiertes MMBasic-Programm, das als "Hülle" für ein Bentzerprogramm verwendet werden kann. Es läuft auch eigenständig um das PIO-Programm zu erstellen. Dieses kann als *.PIO-Datei auf der angeschlossenen SD-Karte (falls vorhanden) gespeichert und wiederhergestellt werden.
Um PASM zu vereinfachen erzeugt es den Code für EINE Zustandsmaschine. Wenn also alle vier Zustandsmaschinen eingeschaltet sind müssen Sie ggf. vier Programme mit unterschiedlichen Offsets zusammenstellen und sie dann kombinieren, um den gesamten Code zu produzieren WRAP_BOTTOM und WRAP_TOP würden für jede Zustandsmaschine so gesetzt werden das sie ihre eigenen Schleifen gleichzeitig ausführen können.
Um PASM zu verwenden muss der Benutzer ein Array dimensionieren, das für den Ausgabecode verwendet werden soll z.B. DIM out %(7). Derzeit wird PASM unabhängig davon immer einen Block mit 32 Anweisungen erzeugen unabhängig von der Länge des Programms.
Die Eingabe in den Assembler erfolgt über DATA-Anweisungen. Es können bis zu 32 Anweisungen programmiert werden aufgrund des Vorhandenseins von Direktiven und Pseudooperationen kann es mehr DATA-Anweisungen als diese geben. Die abschließende DATA-Anweisung muss eine Nullzeichenfolge sein. Trennzeichen in den Anweisungen müssen ein einzelnes Leerzeichen oder ein einzelnes Komma sein.
Zusätzliche Trennzeichen führen dazu, dass die Anweisung falsch zusammengesetzt wird.
Unterstützte Richtlinien:

- *wrap target* Die Anweisung nach dieser Direktive wird angesprungen wenn eine *.wrap A*nweisung erreicht wird.
- *wrap* Springe zum *.wrap Ziel*.

Hinweis: Die obigen Anweisungen können nur einmal für einen Zustandsautomaten verwendet werden.

- *.origin n* Wird am Anfang des Programms verwendet, dieses startet bei Addresse n. Der Wert wird bei JMP-Befehlen verwendet. Mit Vorsicht verwenden !
- *.word n* Fügt den 16-Bit-Wert n ein. Das können eingebettete Daten sein.

Pseudoooperationen:

- *nop* Keine Operation. Tatsächlich wird dies als &HA042 (mov_y,y) zusammengesetzt

Verwendung von PASM

Die PASM-Unterprogramme können in das Anwenderprogramm eingebunden werden. Typische Verwendung zum Schreiben eines Programms für Zustandsmaschine 0 wäre:

```
DIM INTEGER out%(7)                                          'dimensioniert ein Ausgabearray
DIM INTEGER EXECCTRL%(3), PINCTRL%(3), SHIFTCTRL%(3)  'dimensioniert die Regarrays.

datalabel:                                   'Hier beginnt das PIO-Programm des Benutzers
DATA "instruction"
DATA "instruction"
....
....
DATA "instruction"
DATA ""                                      'benutze einen Null-String, um die Daten zu beenden
initrp                                       'Zustandsmaschinenregister auf Standardwerte
                                              initialisieren
                                            'In diesem Stadium können Sie berechnete Werte von
EXECCTRL%(), PINCTRL%() und SHIFTCTRL%() setzen
'Falls erforderlich. Verwenden Sie nicht initrp, wenn Sie Registerwerte aus einer *.PIO-Datei
importieren, da sie dadurch zurückgesetzt werden them..

PASM "datalabel", out%(), 0                  'assembliert den Code. Ungenutzte Adr. im Block
                                              ' werden mit NOP Befehlen gefüllt.

fileman("s","myPIOprog")                     'speichere Prog. Audf der SD-Karte
END                                          'Programmende
```

Beim Verlassen:
out%() ist ein Array of 8 64-bit Ganzzahlen, jede enthält 4 16-bit Anweisungen.
EXECCTRL%(state_machine_number)- WRAP_BOTTOM
und
EXECCTRL%(state_machine_number)- WRAP_TOP
werden gemäß den Befehlen .wrap top und .wrap aktualisiert.

fileman ist eine einfache Befehlszeilen-Dateimanager-Subroutine für *.PIO-Dateien. *.PIO-Dateien werden auf einer angeschlossene SD-Karte im aktuellen Verzeichnis gespeichert. Die Dateien können auch von den anderen Programmen des Pakets verwendet werden sodass die Datenübertragung zwischen ihnen recht einfach sein kann.

fileman("d") gibt eine Verzeichnisliste der *.PIO-Dateien im aktuellen Verzeichnis aus.

fileman("s","filename") speichert "filename.PIO" im aktuellen Verzeichnis.

fileman("l","filename") lädt "filename.PIO" aus dem aktuellen Verzeichnis

Es findet keine Fehlerprüfung statt und fileman schlägt mit einer Fehlermeldung fehl, wenn die Datei nicht existiert. Diese Unterroutine kann verwendet werden, um die Register der Zustandsmaschine von PREDIT vor der Ausführung zu importieren PASM, um das Programm zu erstellen. Es kann dann verwendet werden, um das fertige Programm und die Register zu speichern. Wenn die Dateierweiterung nicht angegeben ist, wird ".PIO" automatisch angehängt

PIO Dateien

This is a simple ASCII text file format. A PIO file contains the PIO program array followed by the EXECCTRL, SHIFTCTRL, PINCTRL and CLKDEV registers for each of the four state machines.

PREVAS

Dies ist ein einfaches ASCII-Textdateiformat. Eine PIO-Datei enthält das PIO-Programm-Array gefolgt von der

EXECCTRL-, SHIFTCTRL-, PINCTRL- und CLKDEV-Register für jede der vier Zustandsmaschinen.

PREVAS

PREVAS ist ein eigenständiges Programm und recht einfach aber dennoch nützlich. Es gibt ein paar einstellbare Variablen:

• Die Anzahl der Seitenauswahl-Steuerbits (Standard ist 2)

• Ob das MSB des Seitenauswahlzählers ein Aktivierungsbit ist (die Voreinstellung ist ja)

• Wenn die Binärwerte der Anweisungen in die Anzeige aufgenommen werden

• Wenn die Dezimalwerte der Anweisungen in der Anzeige enthalten sind

• Wenn die Hex-Werte der Anweisungen in der Anzeige enthalten sind

PREVAS liest die Programmdaten und das EXECCTRL-Register aus einer *.PIO-Datei.

Die ersten Veröffentlichungen von PREVAS sind sehr einfach und können herausgefunden werden. Es kann nicht sagen, ob das .word n und .origin n Direktiven wurden verwendet, da diese nicht im Programm erscheinen. Es nimmt seine Eingabedaten ausschließlich aus dem zusammengesetzten Programmarray und den Programmargumenten. Ein .word-Wert ist umgekehrt als Anweisung zusammengebaut (und Sie wirklich verwirren!). Der .origin-Wert ist nicht bekannt, also wird er es immer sein zeigt die Speicheradressen von 0 bis 31 an. PREVAS liest das EXECCTRL-Register, um die Positionen von .wrap und .wrap target der Reihe nach zu bestimmen um sie in der Liste anzuzeigen. Beachten Sie, dass PREVAS zwar den Programmspeicher entschlüsselt, Ihnen aber nicht unbedingt mitteilt, was eine bestimmte Zustandsmaschine tut da sie außer Wrap und Wrap Target nichts davon liest

PREDIT
Dieses eigenständige Programm kann verwendet werden, um die Werte für die Zustandsmaschinenregister zu berechnen. Es dupliziert mehr oder weniger die Hilfsfunktionen, hat aber eine Konsolenschnittstelle. Es enthält alle möglichen Einstellungen für diese Register (einschließlich einiger illegaler!) und kann natürlich *.PIO-Dateien speichern und laden. es ist wahrscheinlich von größtem Nutzen, wenn Sie an einem System arbeiten, auf dem mehrere Zustandsmaschinen in Betrieb sind. Es ermöglicht die Anzeige der berechneten Werte in Echtzeit, als Optionen und Einstellungen innerhalb der Register geändert werden. Es berechnet noch keine Einstellungen für CLKDIV.

Anhang F
BASIC-Programmierung - Eine Einführung

Der PicoMite wird mit der Programmiersprache BASIC programmiert. Die PicoMite-Version von BASIC heißt MMBasic das den beliebten Microsoft BASIC-Interpreter lose emuliert. Die BASIC-Sprache wurde 1964 vom Dartmouth College in den USA als Computersprache für den Programmierunterricht eingeführt und ist dementsprechend einfach anzuwenden und zu erlernen. Gleichzeitig ist es hat sich als kompetente und mächtige Programmiersprache erwiesen und ist dadurch Ende der 70er bis Anfang der 80er Jahre sehr populär geworden. Auch heute noch werden einige große kommerzielle Datensysteme geschrieben in der BASIC-Sprache (hauptsächlich Pick Basic). Für den PicoMite ist der größte Vorteil von BASIC die Benutzerfreundlichkeit. Einige modernere Sprachen wie C und C++ können wirklich umwerfend sein aber mit BASIC können Sie mit einem einzeiligen Programm beginnen und etwas Vernünftiges daraus machen. MMBasic ist auch dahingehend leistungsfähig, dass Sie anspruchsvolle Grafiken erstellen können oder externen E/A-Pins manipuliert werden können um andere Geräte zu steuern und mit anderen zu kommunizieren wobei Sie auf eine Reihe integrierter Kommunikationsprotokolle zurückgreifen können.

Struktur eines BASIC Programms

Ein BASIC-Programm beginnt in der ersten Zeile und wird bis zur letzten Zeile fortgesetzt oder trifft auf einen END-Befehl. Dann erscheint bei MMBasic die Eingabeaufforderung (>) auf der Konsole und wartet auf eine Eingabe. Ein Programm besteht aus einer Reihe von Anweisungen oder Befehlen, von denen jeder den BASIC-Interpreter veranlasst etwas zu tun (die Wörter Anweisung und Befehl bedeuten im Allgemeinen dasselbe und werden in diesem Tutorial gleichwertig verwendet). Normalerweise befindet sich jede Anweisung in einer extra Zeile. Sie können aber auch mehrere Anweisungen in einer Zeile aufführen. Diese müssen einen Doppelpunkt (:) getrennt sein.

Beispiel:

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

Jede Zeile kann mit einer Zeilennummer beginnen. Zeilennummern waren in den alten BASIC-Versionen obligatorisch, die moderne Implementierungen wie MMBasic benötigen sie jedoch nicht. Sie können sie immer noch verwenden aber sie haben heutzutage keinen Nutzen mehr und überladen nur Ihre Programme.

Beispiel für ein Programm, das Zeilennummern verwendet:

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

Eine Zeile kann auch mit einem Label beginnen das mit GOTO als Ziel für einen Sprungbefehl verwendet werden kann. Dies wird ausführlicher erklärt, wenn wir den GOTO-Befehl behandeln, aber dies ist ein Beispiel (der Labelname ist JmpBack):

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

Kommentare

Ein Kommentar ist ein beliebiger Text, der auf das einfache Anführungszeichen (') folgt. Ein Kommentar kann überall platziert werden und reicht bis zum Zeilenende. Wenn MMBasic auf einen Kommentar trifft springt es einfach an dessen Ende (dh der Kommentar wird igniert). Kommentare sollten verwendet werden um nicht offensichtliche Teile des Programms jemanden zu erklären der nicht mit dem Programm vertraut ist. Denken Sie daran dass Sie sich binnen weniger Monate an Details eines von Ihnen geschriebenes Programms kaum mehr erinnern werden. Sie werdensich selbst dankbar sein wenn Sie genügend Kommentare verwendet haben.

Beispiel:

```
' Die Hypotenuse berechnen
PRINT SQR(a * a + b * b)
```

oder

```
INPUT var    ' Temperatur eingeben
```

Ältere BASIC Programme vervendeten den Befehl REM am anfang eines Kommentars. Das ist zulässig doch doch das Anführunszeichen ist hier einfacher zu verwenden.

Der PRINT Befehl

Es gibt eine Reihe allgemeiner Befehle, die ständig genutzt werden sind und wir werden sie in diesem Abschnitt behandeln. Der nützlichste davon ist der PRINT-Befehl. Seine Aufgabe ist einfach etwas auf der Konsole anzuzzeigen. Dies wird hauptsächlich verwendet um Daten auszugeben die Sie sehen können wie das Ergebnis von Berechnungen oder Meldungen liefern. PRINT ist auch nützlich wenn Sie einen Fehler in Ihrem Programm verfolgen; Sie können es zum verwenden um Werte von

Variablen und Anzeigemeldungen in Schlüsselphasen der Programmausführung anzuzeigen. In seiner einfachsten Form gibt der Befehl einfach aus was in der Befehlszeile steht, etwa:

PRINT 54 zeigt auf der Konsole die Nummer 54 an gefolgt von einer neuen Zeile. Die zu druckenden Daten können etwas Einfaches sein oder ein Ausdruck der etwas berechnet. Wir werden Ausdrücke später ausführlicher behandeln, hier nur als Beispiel:

> PRINT 3/21

0,1428571429

>

würde das Ergebnis von drei geteilt durch einundzwanzig berechnen und anzeigen. Beachten Sie, dass das größer als Symbol (>) ist die MMBasic-Eingabeaufforderung.

Weitere Beispiele:

> PRINT "Wunderbare Welt"

Wundervolle Welt

> PRINT (999 + 1) / 5

200

>

Sie können diese an der Eingabeaufforderung ausprobieren.

Der PRINT-Befehl funktioniert auch mit mehreren Werten gleichzeitig, zum Beispiel:

> DRUCKEN „Der Betrag ist „345“ und der zweite Betrag ist „456“.

Der Betrag ist 345 und der zweite Betrag ist 456

>

Normalerweise wird jeder Wert durch ein Leerzeichen getrennt wie im vorherigen Beispiel gezeigt, aber Sie können Werte auch mit einem Komma (,) trennen. Das Komma bewirkt, dass zwischen den beiden Werten ein Tab eingefügt wird. In MMBasic sind TABS im PRINT-Befehl acht Zeichen voneinander entfernt. Zur Veranschaulichung des Tabulators

Der folgende Befehl druckt eine tabellarische Liste mit Zahlen:

> PRINT 12, 34, 9.4, 1000

12 34 9,4 1000

Variablen

Bevor wir weiter gehen, müssen wir definieren was eine "Variable" ist da sie für die von grundlegender Bedeutung ist für die Nutzung der BASIC-Sprache (tatsächlich für die meisten Programmiersprachen). Eine Variable ist einfach ein Speicherplatz an dem ein Datenelement (dh sein "Wert") gespeichert wird. Dieser Wert kann während der Ausführung des Programms geändert werden warum es eine "Variable" genannt wird. Variablen in MMBasic können einem von drei Typen angehören. Am gebräuchlichsten ist Gleitkomma und das ist automatisch angenommen, wenn der Typ der Variablen nicht angegeben ist. Die anderen beiden Typen sind Integer und Zeichenfolge. Wir werden diese später behandeln. Eine Gleitkommazahl ist eine gewöhnliche Zahl die einen Dezimalpunkt enthalten darf. Zum Beispiel sind 3,45 oder -0,023 oder 100,00 alles Gleitkommazahlen. Eine Variable kann verwendet werden um eine solche Zahl zu speichern und sie

kann dann auf die gleiche Weise wie die Zahl verwendet werden, in diesem Fall stellt es den Wert der letzten ihm zugewiesenen Zahl dar.

Einfaches Beispiel:

```
A = 3
B = 4
PRINT A + B
```

zeigt die Zahl 7 an. In diesem Fall sind sowohl A als auch B Variablen und MMBasic verwendet ihre aktuellen Werte in der PRINT-Anweisung. MMBasic erstellt automatisch eine Variable wenn es das erste Mal darauf trifft Daher erstellte die Anweisung A = 3 beide eine Gleitkommavariable (den Standardtyp) mit dem Namen A und ihm dann den Wert 3 zugewiesen. Der Name einer Variablen muss mit einem Buchstaben beginnen, während der Rest des Namens Buchstaben enthalten kann. Ziffern, Unterstriche oder Punkte (oder Punkte). Der Name kann bis zu 32 Zeichen lang sein lang und die Groß-/Kleinschreibung (dh Großbuchstaben oder nicht) ist nicht wichtig. Hier sind einige Beispiele:

```
Total_Count
ForeColour
temp3
count
x
ThisIsALongVariableName
increment.value
```

Sie können den Wert einer Variablen überall in Ihrem Programm ändern, indem Sie den Zuweisungsbefehl verwenden, dh:

```
variable = expression
```

Beispiel:

```
temp3 = 24.6
count = 5
CTemp = (FTemp – 32) * 0.5556
```

Im letzten Beispiel sind sowohl CTemp als auch FTemp Variablen und diese Zeile konvertiert den Wert von FTemp (in Grad Fahrenheit) in Grad Celsius und speichert das Ergebnis in der Variablen CTemp.

Ausdrücke

Wir sind dem Begriff „Ausdruck“ bereits in diesem Tutorial begegnet und beim Programmieren hat er eine spezifische Bedeutung. Es ist eine Formel die vom BASIC-Interpreter in eine einzelne Zahl oder einen Wert aufgelöst werden kann. MMBasic wertet numerische Ausdrücke nach denselben Regeln aus die wir in der Schule gelernt haben. Beispielsweise werden zuerst Multiplikation und Division durchgeführt, gefolgt von Addition und Subtraktion. Diese werden als Vorrangregeln bezeichnet und in diesem Handbuch ausführlich beschrieben. Das bedeutet, dass MMBasic 2 + 3 * 6 auflöst, indem es zuerst 3 mit 6 multipliziert, was 18 ergibt, und dann 2 addiert was zu einem Endwert von 20 führt. Ebenso werden sowohl 5 * 4 als auch 10 + 4 * 3 – 2 ebenfalls zu 20 aufgelöst. Wenn Sie den Interpreter zwingen möchten, zuerst Teile des Ausdrucks auszuwerten, können Sie diesen Teil des Ausdrucks in Klammern setzen. Zum Beispiel wird (10 + 4) * (3 – 2) zu 14 aufgelöst und nicht zu 20, wie dies der Fall wäre der Fall gewesen wäre wenn die Klammern nicht verwendet worden wären. Die Verwendung von Klammern verlangsamt das Programm nicht merklich also sollten Sie sie großzügig verwenden, wenn die Möglichkeit besteht, dass MMBasic Ihre Absicht falsch interpretiert. Wie bereits erwähnt können Sie Variablen in einem Ausdruck genauso verwenden wie gerade Zahlen.

Dies erhöht beispielsweise den Wert der Variablen temp um eins:

```
temp = temp + 1
```

Sie können auch Funktionen in Ausdrücken verwenden. Dies sind spezielle Operationen, die von MMBasic bereitgestellt werden, z.B. zur Berechnung trigonometrischer Werte. Als Beispiel wird im Folgenden die Länge der Hypotenuse eines rechtwinkligen Dreiecks mit der Funktion SQR() die die Quadratwurzel von a zurückgibt Zahl (a und b sind Variablen die die Längen der anderen Seiten enthalten):

```
PRINT SQR(a * a + b * b)
```

MMBasic wertet diesen Ausdruck zuerst aus indem es a mit a multipliziert, dann b mit b und dann die Ergebnisse zusammen addiert. Die resultierende Zahl wird dann an die Funktion SQR() übergeben, die die Quadratwurzel dieser Zahl (dh die Hypotenuse) berechnet und gibt sie an den PRINT-Befehl zur anzeige weiter.

Einige andere mathematische Funktionen die von MMBasic bereitgestellt werden umfassen:

SIN(r) – der Sinus von r

COS(r) – der Kosinus von r

TAN(r) – der Tangens von r

Es stehen Ihnen noch viele weitere Funktionen zur Verfügung die alle weiter oben in diesem Handbuch aufgeführt sind. Beachten Sie, dass in den obigen trigonometrischen Funktionen der an die Funktion übergebene Wert (dh 'r') der Winkel im Bogenmaß ist. In MMBasic können Sie die Funktion RAD(d) verwenden, um einen Winkel von Grad in Bogenmaß umzuwandeln ('d' ist der Winkel in Grad). Ein weiteres Merkmal von BASIC ist, dass Sie Funktionsaufrufe ineinander verschachteln können. Zum Beispiel gegeben der Winkel in Grad (d. h. 'd') der Sinus dieses Winkels kann mit diesem Ausdruck gefunden werden::

```
PRINT SIN(RAD(d))
```

In diesem Fall nimmt MMBasic zuerst den Wert von d und konvertiert ihn mit der Funktion RAD() in Radiant. Die Ausgabe dieser Funktion wird dann zur Eingabe der SIN()-Funktion.

Die IF Anweisung

Entscheidungen zu treffen ist der Kern der meisten Computerprogramme und in BASIC wird dies normalerweise erledigt die IF-Anweisung. Dies ist fast wie ein englischer Satz geschrieben:

IF Bedingung THEN Aktion. Die Bedingung ist normalerweise ein Vergleich wie gleich, kleiner als, mehr als usw.

Beispielsweise:

```
IF Temp < 25 THEN PRINT "Kalt"
```

`Temp` wäre eine Variable die die aktuelle Temperatur (in ºC) enthält und PRINT "Kalt" die Aktion die ausgeführt wird. Es gibt eine Reihe von Tests die Sie durchführen können:

=	gleich	<>	ungleich
<	kleine als	<=	kleiner oder gleich
>	größer als	>=	größer oder gleich

Sie können auch eine ELSE-Klausel hinzufügen, die ausgeführt wird, wenn die Anfangsbedingung als falsch getestet wird: IF Bedingung THEN wahre Aktion ELSE falsche Aktion

Dies führt beispielsweise verschiedene Aktionen aus, wenn die Temperatur unter 25 oder 25 oder mehr liegt:

```
IF Temp < 25 THEN PRINT "Kalt" ELSE PRINT "Heiß"
```

Die vorherigen Beispiele verwendeten alle einzeilige IF-Anweisungen, aber Sie können auch mehrzeilige IF verwenden Aussagen. Sie sehen so aus:

IF *condition* THEN
TrueActions

TrueActions
ENDIF

oder

IF *condition* THEN
TrueActions
TrueActions
ELSE
FalseActions
FalseActions
ENDIF

Anders als bei der einzeiligen IF-Anweisung können Sie viele wahre Aktionen mit jeder in ihrer eigenen Zeile und haben ebenso viele Fehlhandlungen. Im Allgemeinen ist die einzeilige IF-Anweisung praktisch, wenn Sie eine einfache Aktion haben die ausgeführt werden muss, während die mehrzeilige Version viel einfacher zu verstehen ist wenn die durchzuführenden Aktionen zahlreich und komplizierter sind.

Ein Beispiel für eine mehrzeilige IF-Anweisung mit mehr als einer Aktion ist:

```
IF Amount < 25 THEN
    PRINT "Zu niedrig"
    PRINT "Minimalwert=25"
ELSE
    PRINT "Eingabe akzeptiert"
    SaveToSDCard
    PRINT "Zweiten Betrag eingeben"
ENDIF
```

Beachten Sie, dass im obigen Beispiel jede Aktion eingerückt ist, um anzuzeigen, zu welchem Teil der IF-Struktur sie gehört zu. Das Einrücken ist nicht obligatorisch, aber es macht ein Programm für jemanden, der es damit nicht vertraut ist, viel einfacher zu verstehen und daher sehr zu empfehlen. In einer mehrzeiligen IF-Anweisung können Sie zusätzliche Tests mit dem ELSE IF-Befehl durchführen. Das ist das Beste an einem Beispiel erklärt (die Temperaturen sind alle in ºC):

```
IF Temp < 0 THEN
    PRINT "Einfrieren"
ELSE IF Temp < 20 THEN
    PRINT "Kalt"
ELSE IF Temp < 35 THEN
    PRINT "Warm"
ELSE
    PRINT "Heiß"
ENDIF
```

Das ELSE IF kann die gleichen Tests wie ein gewöhnliches IF verwenden (dh <, <=, etc.), aber dieser Test wird nur dann durchgeführt, wenn der vorhergehende Test falsch war. So erhalten Sie zum Beispiel nur die Meldung “Warm” wenn Temp < 0 fehlgeschlagen, und Temp < 20 fehlgeschlagen, aber Temp < 35 war wahr. Das abschließende ELSE fängt den Fall ab, wenn alle Tests falsch wären. Ein Ausdruck wie Temp < 20 wird von MMBasic entweder als wahr oder falsch ausgewertet, wobei wahr den Wert eins und falsch null hat. Sie können dies sehen, wenn Sie Folgendes an der Konsole eingegeben haben:

```
PRINT 30 > 20
```

MMBasic gibt 1 aus, was bedeutet, dass der Wert wahr ist, und ähnlich wird im Folgenden 0 ausgegeben, was bedeutet dass der Ausdruck als falsch ausgewertet wurde.

```
PRINT 30 < 20
```

Die IF-Anweisung kümmert sich nicht wirklich darum, was die Bedingung tatsächlich ist, sie wertet nur die aus Bedingung und wenn das Ergebnis Null ist, wird es als falsch angenommen und wenn es nicht Null ist, wird es als wahr angenommen. Dies ermöglicht einige praktische Verknüpfungen. Wenn beispielsweise BalanceCorrect eine Variable ist, die wahr ist (Nicht-Null), wenn eine Funktion des Programms korrekt ist, dann kann das Folgende verwendet werden, um eine Entscheidung zu treffen basierend auf diesem Wert:

```
IF BalanceCorrect THEN …do something…
```

FOR Schleifen

Eine weitere häufige Anforderung beim Programmieren ist das Wiederholen einer Reihe von Aktionen. Zum Beispiel könnten Sie alle sieben Tage der Woche durchlaufen und für jeden Tag die gleiche Funktion ausführen möchten. BASIC stellt das FOR-Schleifenkonstrukt für diese Art von Job bereit und funktioniert so:

```
FOR day = 1 TO 7
  Do something based on the value of 'day'
NEXT day
```

Dies beginnt damit, die Variable day zu erstellen und ihr den Wert 1 zuzuweisen. Das Programm wird dann die folgenden Anweisungen ausführen bis es zur NEXT-Anweisung kommt. Dies meldet dem BASIC-Interpreter, den Wert von day zu erhöhen, geht zurück zur vorherigen FOR-Anweisung und führt die folgenden Aussagen ein zweites Mal aus. Dies wird fortgesetzt, bis der Wert des Tages überschritten wird 7 und das Programm wird dann die Schleife verlassen und mit den Anweisungen nach NEXT fortfahren Erklärung. Als einfaches Beispiel können Sie die Zahlen von eins bis zehn so drucken:

```
FOR nbr = 1 TO 10
  PRINT nbr,;
NEXT nbr
```

Das Komma am Ende der PRINT-Anweisung weist den Interpreter an, mit der Tabulatortaste zur nächsten Tabulatorspalte danach zu wechseln Wenn Sie die Nummer und das Semikolon drucken, bleibt der Cursor nicht automatisch in dieser Zeile zur nächsten Zeile wechseln. Als Ergebnis werden die Zahlen in ordentlichen Spalten über die Seite gedruckt.

Das würden Sie sehen:

```
1       2       3       4       5       6       7       8       9       10
```

Die FOR-Schleife hat auch ein paar zusätzliche Tricks im Ärmel. Sie können den Betrag ändern, den die Die Variable wird mit dem Schlüsselwort STEP inkrementiert. So wird beispielsweise im Folgenden nur die ungeraden Zahlen gedruckt:

```
FOR nbr = 1 TO 10 STEP 2
  PRINT nbr,;
NEXT nbr
```

Der Wert des Schritts (oder Inkrementwerts) ist standardmäßig eins wenn das Schlüsselwort STEP nicht verwendet wird, aber Sie können es auf jede gewünschte Nummer einstellen.

Wenn MMBasic die Variable erhöht, prüft es, ob die Variable das TO überschritten hat Wert . Falls ja wird die Schleife verlassen. Im obigen Beispiel wird also der Wert neun erreicht und es wird gedruckt, aber in der nächsten Schleife wird nbr elf sein und an diesem Punkt wird die Schleife verlassen. Dieser Test wird auch am Anfang der Schleife angewendet (d. h. wenn am Anfang der Wert der Variable den TO-Wert überschreitet, wird die Schleife niemals ausgeführt, nicht einmal). Indem Sie den STEP-Wert auf eine negative Zahl setzen können Sie die FOR-Schleife zum abwärtszählen verwenden. Im Folgenden werden beispielsweise die Zahlen von 1 bis 10 in umgekehrter Reihenfolge gedruckt:

```
FOR nbr = 10 TO 1 STEP -1
  PRINT nbr,;
```

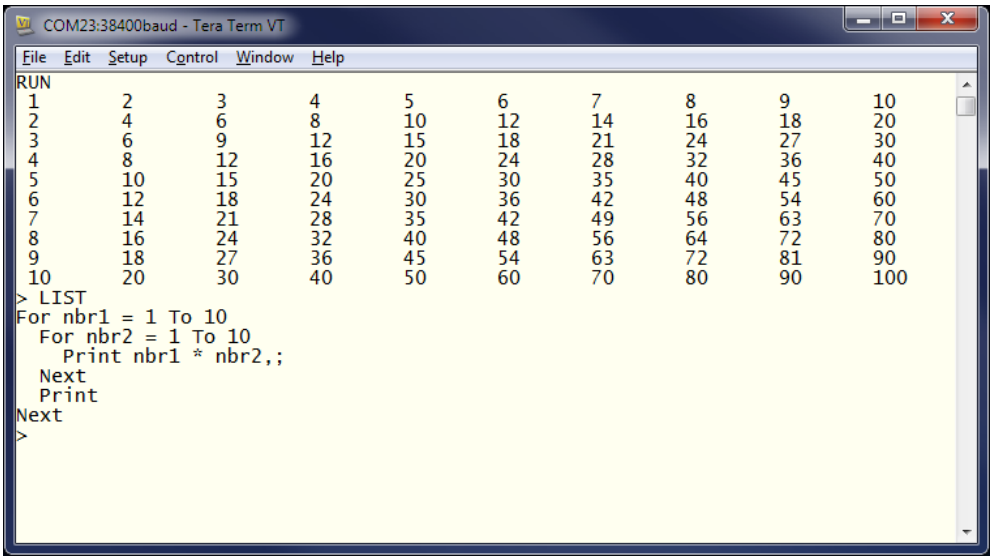
```
NEXT nbr
```

Multiplikationstabelle

Um weiter zu veranschaulichen wie Schleifen funktionieren und wie nützlich sie sein können wird das folgende kurze Programm zwei FOR-Schleifen verwenden das Einmaleins auszudrucken das wir alle in der Schule gelernt haben. Das Programm dazu ist nicht kompliziert:

```
FOR nbr1 = 1 to 10
  FOR nbr2 = 1 to 10
    PRINT nbr1 * nbr2,;
  NEXT nbr2
  PRINT
NEXT nbr1
```

Auf dem Screenshot rechts ist die Ausgabe des Programms und das Programmlisting zu sehen.



Sie müssen die Logik dieses Beispiels Zeile für Zeile durcharbeiten, um zu verstehen, was es tut. Im Wesentlichen besteht es aus einer Schleife in einer anderen. Die innere Schleife, die die Variable nbr2 erhöht druckt eine horizontale Zeile der Tabelle. Wenn diese Schleife beendet ist, wird der folgende PRINT-Befehl ausgeführt, der nichts zu drucken hat - also gibt er einfach eine neue Zeile aus (d. h. beendet die Linie, die von der inneren Schleife gedruckt wird). Das Programm führt dann eine weitere Iteration der äußeren Schleife durch Inkrementieren von nbr1 und aus erneutes Ausführen der inneren Schleife erneut. Schließlich, wenn die äußere Schleife erschöpft ist (wenn nbr1 10 überschreitet) das Programm erreicht das Ende und wird beendet. Ein letzter Punkt, Sie können den Variablennamen aus der NEXT-Anweisung weglassen und MMBasic wird raten welche Variable du meinst. Es empfiehlt sich jedoch, den Namen zur Vereinfachung einzufügen damit jemand anderes, der das Programm liest, es versteht. Sie können auch mehrere Schleifen beenden Verwenden einer durch Kommas getrennten Liste von Variablen in der NEXT-Anweisung. Beispielsweise:

```
FOR var1 = 1 TO 5
  FOR var2 = 10 to 13
    PRINT var1 * var2
NEXT var1, var2
```

DO Schleifen

Ein anderes Schleifenkonstrukt ist die DO…LOOP-Struktur, die so aussieht:

```
DO WHILE condition
   statement
   statement
LOOP
```

Dies beginnt mit dem Testen der Bedingung und wenn sie wahr ist werden die Anweisungen ausgeführt bis zum LOOP-Befehl , dort wird die Bedingung erneut getestet und wenn sie immer noch wahr ist, wird die Schleife erneut ausgeführt. Die „Bedingung“ ist die gleiche wie im IF-Befehl (dh X < Y).

Im Folgenden wird beispielsweise das Wort „Hello“ 4 Sekunden lang auf der Konsole ausgegeben und dann gestoppt:

```
Timer = 0
DO WHILE Timer < 4000
  PRINT "Hallo"
LOOP
```

Beachten Sie dass Timer eine Funktion in MMBasic ist die die Zeit seit dem Timer-Reset in Millisekunden zurückgibt. Ein Reset erfolgt durch Zuweisung von Null an Timer (siehe oben) oder beim Einschalten die PicoMite.

Eine Variation der DO-LOOP-Struktur ist die folgende:

```
DO
   statement
   statement
LOOP UNTIL condition
```

Bei dieser Anordnung wird die Schleife zunächst einmal ausgeführt dann wird die Bedingung geprüft und wenn die Bedingung falsch ist die Schleife wird wiederholt ausgeführt, bis die Bedingung wahr wird. Beachten Sie, dass der Test in LOOP UNTIL ist das Gegenteil von DO WHILE. Ähnlich wie im vorherigen Beispiel wird im Folgenden beispielsweise vier Sekunden lang „Hallo“ ausgegeben:

```
Timer = 0
DO
  PRINT "Hallo"
LOOP UNTIL Timer >= 4000
```

Beide Formen des DO-LOOP tun im Wesentlichen dasselbe, sodass Sie jede passende Struktur verwenden können mit der Logik, die Sie implementieren möchten.

Schließlich ist es möglich, eine DO-Schleife zu haben, die überhaupt keine Bedingungen hat - dh,

```
DO
  statement
  statement
LOOP
```

Dieses Konstrukt wird sich endlos wiederholen und Sie als Programmierer müssen einen Weg finden um die Schleife explizit zu verlassen (der Befehl EXIT DO wird dies tun). Beispielsweise:

```
Timer = 0
DO
  PRINT "Hallo"
  IF Timer >= 4000 THEN EXIT DO
LOOP
```

Konsoleneingabe

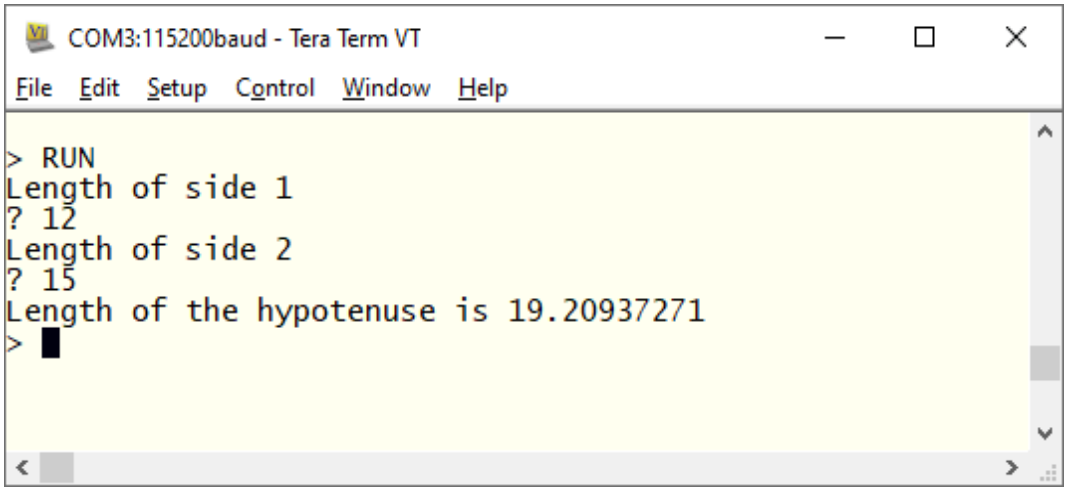
Neben dem Drucken von Daten, die der Benutzer sehen kann, möchten Ihre Programme auch Eingaben vom Benutzer erhalten. Damit dies funktioniert, müssen Sie Tastenanschläge von der Konsole erfassen, und dies kann mit dem INPUT-Befehl erfolgen. In seiner einfachsten Form lautet der Befehl:

EINGANG Var

Dieser Befehl druckt ein Fragezeichen auf dem Bildschirm der Konsole und wartet auf die Eingabe einer Zahl gefolgt von der Eingabetaste. Diese Nummer wird dann der Variablen var zugewiesen. Beispielsweise erweitert das folgende Programm den Ausdruck zum Finden der Hypotenuse eines Dreiecks um ermöglicht es dem Benutzer die Längen der anderen Seiten von der Konsole aus einzugeben.

```
PRINT "Länge der Seite 1"
INPUT a
PRINT "Länge der Seite 2"
INPUT b
PRINT "Die Länge der Hypotenuse beträgt" SQR(a * a + b * b)
```

Dies ist ein Screenshot einer typischen Sitzung:



Der INPUT-Befehl kann auch Ihre Eingabeaufforderung für Sie drucken, sodass Sie keinen separaten PRINT-Befehl benötigen. Dies funktioniert beispielsweise genauso wie das obige Programm:

```
INPUT "Länge der Seite 1"; a
INPUT "Länge der Seite 2”; b
PRINT "Die Länge der Hypotenuse beträgt" SQR(a * a + b * b)
```

Schließlich ermöglicht Ihnen der INPUT-Befehl die Eingabe einer Reihe von Zahlen, die durch Kommas getrennt sind jede Zahl wird in verschiedenen Variablen gespeichert.

Beispielsweise::

```
INPUT "Geben Sie die Länge der zwei Seiten ein: ", a, b
PRINT "Die Länge der Hypotenuse beträgt" SQR(a * a + b * b)
```

Wenn der Benutzer 12,15 eingibt, wird die Zahl 12 in der Variablen a und 15 in b gespeichert. Eine andere Methode, Eingaben von der Konsole zu erhalten, ist der Befehl LINE INPUT. Dadurch wird die ganze Zeile wie vom Benutzer eingegeben einer String-Variablen zugewiesen. Wie beim INPUT-Befehl können Sie eine Eingabeaufforderung angeben. Ein einfaches Beispiel:

```
LINE INPUT "Wie heißen Sie? ", s$
PRINT "Hallo " s$
```

Wir werden später in diesem Tutorial String-Variablen behandeln, aber im Moment können Sie sie sich als Variable vorstellen die eine Folge von einem oder mehreren Zeichen enthält. Wenn Sie das obige Programm ausgeführt haben und “John” eingeben antwortet das Programm mit “Hallo John”. Manchmal möchten Sie nicht darauf warten dass der Benutzer die Eingabetaste drückt sondern jedes Zeichen beim Erhalt angezeigt wird. Dies kann mit der INKEY$-Funktion erfolgen, die den Wert des Zeichens zurückgibt als Zeichenfolge, die nur aus einem Zeichen besteht, oder als leere Zeichenfolge, wenn nichts eingegeben wurde.

GOTO und Labels

Ein Verfahren zur Steuerung des Programmflusses ist der GOTO-Befehl. Das sagt im Wesentlichen MMBasic an einen anderen Teil des Programms zu springen und die Ausführung von dort aus fortzusetzen. Das Ziel der GOTO ist ein Label und das muss zuerst erklärt werden. Ein Label ist eine “Aufkleber” der einen Teil des Programms kennzeichnet. Es muss das erste sei was auf der Zeile steht und muss mit dem Doppelpunkt (:) abgeschlossen werden. Der verwendete Name kann bis zu 32 Zeichen lang sein und müssen den gleichen Regeln folgen wie für den Namen einer Variablen. Zum Beispiel in der folgenden Programmzeile ist LoopBack ein Label:

```
LoopBack:  a = a + 1
```

Wenn Sie den GOTO-Befehl verwenden um zu diesem bestimmten Teil des Programms zu springen sollte der Befehl so verwenden:

```
GOTO LoopBack
```

Um dies alles in einen Zusammenhang zu bringen, druckt das folgende Programm alle Zahlen von 1 bis 10 aus:

```
z = 0
LoopBack:  z = z + 1
PRINT z
IF z < 10 THEN GOTO LoopBack
```

Das Programm beginnt damit die Variable z auf Null zu setzen und sie dann in der nächsten Zeile auf 1 zu erhöhen. Der Wert von z wird gedruckt und dann geprüft ob er kleiner als 10 ist. Wenn er kleiner als 10 ist, wird das Programm zurückspringen zum Label LoopBack, wo sich dieser Prozess wiederholt. Schließlich erreicht der Wert von z mehr als 10 und das Programm wird beendet. Beachten Sie dass eine FOR-Schleife dasselbe tun kann und auch einfacher ist daher ist dieses

Beispiel nur darauf ausgelegt zu veranschaulichen was der GOTO-Befehl bewirken kann. In der Vergangenheit hatte der GOTO-Befehl einen schlechten Ruf. Dies liegt daran dass die Verwendung von GOTOs möglich ist um ein Programm zu erstellen das kontinuierlich von einem Punkt zum anderen springt. Diese Art Programm ist für einen anderen Programmierer fast unmöglich zu verstehen. Dadurch entstand der Name “Spaghetti-Code” was einem Schimpfwort nahe kommt.

Mit Konstrukten wie die mehrzeiligen IF-Anweisungen wurde die Notwendigkeit der GOTO-Anweisung reduziert . Diese sollte **NUR** verwendet werden wenn es keine andere Möglichkeit gibt den Programmablauf zu ändern. Bitte das unbedingt beachten !

Primzahlentest

Das Folgende ist ein einfaches Programm das viele der zuvor besprochenen Programmierfunktionen vereint.

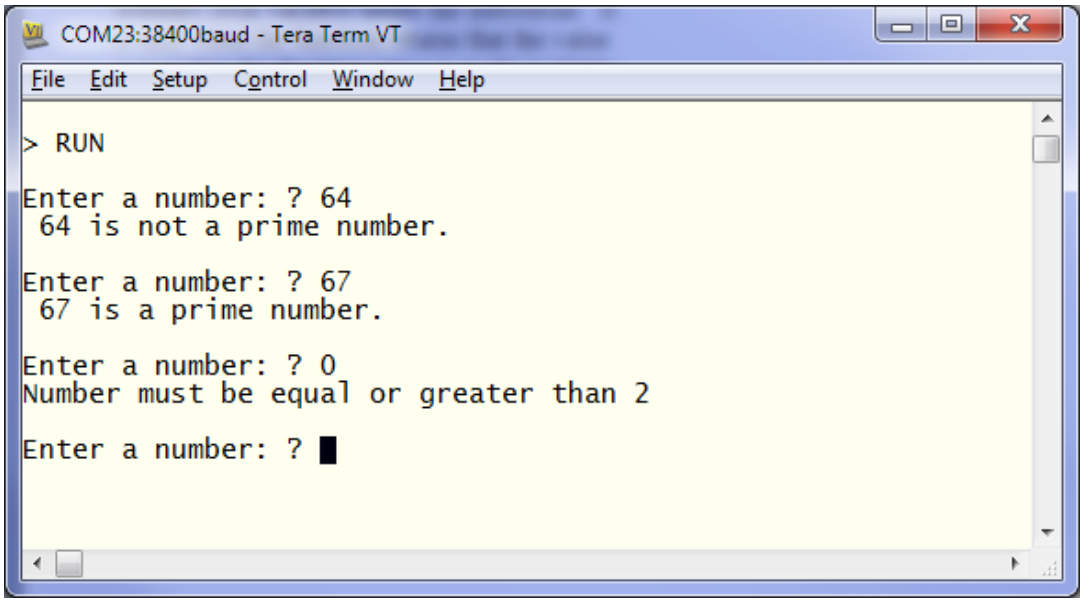
```
DO
  InpErr:
  PRINT
  INPUT "Bitte eine Zahl eingeben: "; a
  IF a < 2 THEN
    PRINT "Die Zahl muß größer oder gleich 2 sein"
    GOTO InpErr
  ENDIF

  Divs = 0
  FOR x = 2 TO SQR(a)
    r = a/x
    IF r = FIX(r) THEN Divs = Divs + 1
  NEXT x

  PRINT a " ist ";
  IF Divs > 0 THEN PRINT "keine ";
  PRINT "Primzahl."
LOOP
```

Dies wird auf der Konsole zur Eingabe einer Nummer auffordern und wenn diese eingegeben wurde, testen, ob dies der Fall ist Zahl ist eine Primzahl oder nicht und zeigt dann eine passende Meldung an. Es beginnt mit einer DO-Schleife die keine Bedingung hat. Es wird endlos fortgesetzt, genau was wir wollen. Das bedeutet, dass wenn der Benutzer eine Zahl eingegeben hat gemeldet wird ob es sich um eine Primzahl handelt oder nicht zum Anfang springen und nach einer anderen Nummer fragen. Die Art und Weise wie der Benutzer das Programm ggf. beenden kann ist durch die Eingabe des Unterbrechungszeichens (normalerweise STRG-C). Das Programm gibt dann eine Eingabeaufforderung für den Benutzer aus die mit einem Semikolon abgeschlossen wird. Dies bedeutet dass der Cursor am Ende der Eingabeaufforderung für den INPUT-Befehl bleibt der die Zahl in der Variablen a speichert. Anschließend wird die Nummer getestet. Ist es weniger als 2 wird eine Fehlermeldung gedruckt und die Das Programm springt zurück und fragt erneut nach der Nummer. Wir können jetzt testen ob die Zahl eine Primzahl ist. Das Programm verwendet eine FOR-Schleife um mögliche Teiler darauf zu testen ob jeder die eingegebene Zahl gleichmäßig teilen kann. Jedes Mal wenn dies der Fall ist erhöht das Programm die Variable Divs. Beachten Sie, dass der Test mit der Funktion FIX(r) durchgeführt wird die einfach alle Ziffern nach dem Komma entfernt. Die Bedingung r = FIX(r) ist also wahr wenn r eine Ganzzahl ist (d. h. keine Ziffern nach dem Komma hat). Schließlich erstellt das Programm die Nachricht für den Benutzer. Der Schlüsselteil ist dass wenn die Variable Divs größer als Null ist bedeutet dies, dass eine oder mehrere Zahlen gefunden wurden, die sich gleichmäßig in die Testnummer teilen lassen. In diesem Fall fügt die IF-Anweisung

das Wort "not" in die Ausgabenachricht ein. Wenn die eingegebene Zahl beispielsweise 21 war, sieht der Benutzer diese Antw:

```
21 ist keine Primzahl.
```

Dies ist das Ergebnis der Ausführung des Programms und einiger Ausgaben:



Sie können dieses Programm testen indem Sie den Editor (den EDIT-Befehl) verwenden, um es einzugeben. Mit Ihren neu erlernten Fähigkeiten könnten Sie dann versuchen es effizienter zu machen. Beispielsweise weil das Programm zählt wie oft eine Zahl in die Testzahl geteilt werden kann dauert es viel länger als es sollte um eine Nicht-Primzahl zu erkennen. Das Programm würde viel effizienter laufen wenn es bei der ersten Zahl, die sich gleichmäßig teilt die FOR-Schleife verlässt. Sie könnten z.B. GOTO verwenden oder Sie könnten den Befehl EXIT FOR verwenden – das würde die FOR-Schleife sofort beenden. Andere Effizienzvorteile bestehen darin die Division nur mit ungeraden Zahlen zu testen (ein anfänglicher Test für eine gerade Zahl dann die FOR-Schleife bei 3 beginnen und Verwendung von STEP 2) oder indem nur Primzahlen für den Test verwendet werden, das wäre aber viel komplizierter!.

Arrays

Arrays sind etwas, was man auf den ersten Blick nicht als nützlich erachtet, aber wenn Sie sie verwenden müssen werden Sie sie wirklich sehr praktisch finden. Ein Array stellt man sich am besten als eine Reihe von Briefkästen für einen Wohnblock vor wie rechts gezeigt. Die Briefkästen sind alle an derselben Adresse und jedes Kästchen repräsentiert eine Einheit oder Eigentumswohnung an dieser Adresse. Sie können einen Buchstaben in das Feld für die Einheit einfügen eins oder Einheit zwei usw. Ähnlich ist ein Array in BASIC eine einzelne Variable mit mehreren Untereinheiten (in BASIC Elemente genannt), die nummeriert sind. Sie kann Daten in Element eins oder Element zwei usw. platzieren. In BASIC wird ein Array durch den DIM-Befehl erstellt, zum Beispiel:

```
DIM numarr(300)
```

Dadurch wird ein Array mit dem Namen **numarr** erstellt das 301 Elemente enthält (stellen Sie sich diese als Briefkästen im Bereich von 0 bis 300 vor. Standardmäßig beginnt ein Array bei Null deshalb gibt es ein zusätzliches Element das die Gesamtzahl 301 ergibt. Um ein bestimmtes Element im Array anzugeben etwa ein bestimmter Briefkasten, verwenden sie einen Index der einfach die Nummer des Array-Elements ist auf das Sie zugreifen möchten. Für Wenn Sie beispielsweise die

Elementnummer 100 in diesem Array auf (sagen wir) die Nummer 876 setzen möchten machen Sie dies so:

```
numarr(100) = 876
```

Normalerweise ist der Index eines Arrays keine konstante Zahl wie in diesem Beispiel die 100, sondern eine Variable die geändert werden kann um auf verschiedene Array-Elemente zuzugreifen. Beispiel: Man möchte die Maximaltemperatur für jeden Tag des Jahres aufzeichnen und am Ende des Jahres daraus den Jahresduchschnitt berechnen. Sie könnten gewöhnliche Variablen verwenden um die Temperatur für jeden Tag aufzuzeichnen aber Sie würden es brauchen 365 davon und das würde Ihr Programm in der Tat unhandlich machen. Stattdessen könnten Sie ein Array definieren um die Werte so zu halten:

```
DIM days(365)
```

Jeden Tag müssten Sie die Temperatur an der richtigen Stelle im Array speichern. Wenn die Anzahl der Der Tag im Jahr wurde in der Variable doy gehalten und die maximale Temperatur wurde in der gehalten Variable maxtemp würden Sie den Messwert wie folgt speichern:

Tage (doy) = maxtemp

Am Ende des Jahres wäre es einfach, den Jahresdurchschnitt zu berechnen:

```
total = 0
FOR i = 1 to 365
  total = total + days(i)
NEXT i
PRINT "Die Durchschnittstemperatur beträgt:"  total / 365
```

Das ist viel einfacher, als 365 einzelne Variablen zu addieren und zu mitteln. Das obige Array war eindimensional, aber Sie können auch mehrdimensional sein. Um auf unsere Analogie von den Briefkästen zurückzukommen kann man sich ein Array mit zwei Dimensionen als Wohnblock mit mehreren Etagen vorstellen. Ein Block könnte eine Reihe von vier Briefkästen für Ebene eins haben, eine weitere Reihe von vier Feldern für Ebene zwei und so weiter. Um einen Brief zu platzieren in einem Briefkasten müssen Sie die Etagennummer und die Nummer der Einheit dieser Etage angeben. In BASIC wird das Array-Element durch zwei Komma-getrennte Indizes angegeben

Beispiel:

```
LetterBox(floor, unit)
```

Nehmen Sie als praktisches Beispiel an, dass Sie die maximale Temperatur für jeden Tag über 5 Jahre aufzeichnen müssten. Dazu könnten Sie das Array wie folgt dimensionieren:

```
DIM days(365, 5)
```

Der erste Index ist der Tag im Jahr und der zweite eine Zahl die das Jahr darstellt. Um Tag 100 im Jahr 3 auf 24 Grad einzustellen kann man es so machen:

```
days(100, 3) = 24
```

Arrays in MMBasic für PicoMite kann bis zu fünf Dimensionen haben (anders als andere Versionen von MMBasic die acht Dimensionen unterstützen). Die max. Größe eines Arrays ist lediglich durch das freie RAM des RP Pico begrenzt.

Ganzzahlen

Bisher waren alle Zahlen und Variablen, die wir verwendet haben, Gleitkommazahlen. Wie bereits erklärt ist die Nutzung von Gleitkomma praktisch da es Ziffern nach dem Dezimalkomma sich merkt und bei der Verwendung wird eine Division ein vernünftiges Ergebnis zurückgegeben. Im Zweifelsfall halten Sie sich immer an Gleitkommazahlen. Achtung: Der PicoMite speichert Zahlen immer als Näherungswerte und die Genauigkeit liegt bei 14 Stellen ! Meistens ist das bei Gleitkommazahlen kein Problem. Es gibt aber Fälle in denen Sie mit größeren Zahlen hantieren und diese genau speichern müssen.

Nehmen wir als Beispiel an dass Sie die Zeit auf die Mikrosekunde genau so manipulieren möchten dass Sie die Genauigkeit zweier verschiedene Datums-/Zeitangaben vergleichen. Der einfachste Weg dazu ist einen Stichtag als Vergleichspunkt zu wählen z. B. 1. Januar im Jahr Null und dann die Anzahl Mikrosekunden über ein IF Statement zu vergleichen.

Problem: Die Anzahl der Mikrosekunden seit diesem Stichtag würde die Genauigkeit von Fließkommavariablen sprengen daher kommen hier Ganzzahlvariablen zum Einsatz. Eine Integer-Variable in MMBasic Auf dem PicoMite ausgeführt kann Zahlen bis zu neun Millionen Millionen Millionen speichern, exakt ±9223372036854775807 um genau zu sein. Nachteil einer Ganzzahl ist dass sie keine Brüche also die Zahlen nach dem Dezimalkomma speichern kann. Jede Berechnung die zu einem Bruchergebnis führt wird auf die nächste ganze Zahl auf- oder abgerundet. Es ist einfach, eine Integer-Variable zu erstellen fügen Sie einfach das Prozentzeichen (%) als Suffix zu einem Variablennamen hinzu. Beispielsweise ist **sec%** eine ganzzahlige Variable. Innerhalb eines Programms können Sie Ganzzahlen und Fließkommazahlen mischen und MMBasic wird die notwendigen Konvertierungen vornehmen, aber wenn Sie die volle Genauigkeit der Ganzzahlen beibehalten möchten sollten Sie vermeiden beide Arten zu mischen. Genau wie Gleitkommazahlen können Sie Arrays von Ganzzahlen mit bis zu fünf Dimensionen haben. Alles was Sie tun müssen: Fügen Sie dem Array-Namen das Prozentzeichen als Suffix hinzu. Beispiel: Tage %(365, 5). Anfänger sind oft verwirrt wann sie Fließkommazahlen oder ganze Zahlen verwenden sollten, und die Antwort lautet einfach “immer Gleitkommazahlen es sei denn Sie benötigen extrem hohe Genauigkeit in der resultierende Zahl. Dies passiert nicht oft aber inn diesem Fall werden Sie feststellen dass ganze Zahlen ein Lebensretter sind .

Stringvariablen

Strings sind ein weiterer Variablentyp (wie Gleitkommazahlen und Ganzzahlen). Strings werden verwendet, um eine Zeichenkette zu speichern, Beispiel:

```
PRINT "Hello"
```

Der String „Hello“ ist eine Stringkonstante. Beachten Sie dass eine Konstante etwas ist das sich nicht ändert im Gegensatz zu einer Variablen, Stringkonstanten werden immer in doppelte Anführungszeichen gesetzt. Namen von Stringvariablen verwenden das Dollarzeichen ($) als Suffix, um sie als String anstelle von Fließkommavariablen zu identifizieren, Sie können deren Werte mit einer gewöhnlichen Zuweisung festlegen. Hier einige Beispiele (beachten Sie, dass das zweite Beispiel ein Array von Zeichenfolgen verwendet):

```
Car$ = "Holden"
Country$(12) = "India"
Name$ = "Fred"
```

Zeichenfolgen können einfach mit einem “+” verbunden werden:

```
Word1$ = "Hello"
Word2$ = "World"
Greeting$ = Word1$ + " " + Word2$
```

Hier ist der Inhalt von `Greeting$ "Hello World"`.

Strings können auch mit Operatoren wie = (gleich), <> (ungleich), < (kleiner als) usw. verglichen werden. Beispielsweise:
IF Car$ = "Holden" THEN PRINT "Was an Aussie made car"
Der Vergleich wird unter Verwendung des vollständigen ASCII-Zeichensatzes durchgeführt, sodass vor einem druckbaren Zeichen ein Leerzeichen steht . Außerdem wird beim Vergleich zwischen Groß- und Kleinschreibung unterschieden, sodass „holden“ nicht gleich „Holden“ ist. Bei Verwendung der Funktion UCASE() zur Umwandlung in Großbuchstaben können Sie einen Vergleich ohne Berücksichtigung der Groß-/Kleinschreibung durchführen.
Beispielsweise:
IF UCASE$(Car$) = "HOLDEN" THEN PRINT "War ein in Australien hergestelltes Auto"

Stringarrays sind möglich aber bitte setzen Sie sie sparsam ein da sonst evtl. das zur Verfügung stehende RAM nicht ausreicht. MMBasic weist jedem Element eines Arrays 255 Bytes RAM zu .

Ein String-Array mit 100 Elementen belegt standardmässig 25K RAM. Um dies abzumildern können Sie den LENGTH-Qualifizierer verwenden. Begrenzen Sie die maximale Größe jedes Elements. Zum Beispiel wenn Sie wissen, dass die maximale Länge von jedem Zeichenfolge die im Array gespeichert wird weniger als 20 Zeichen lang ist können Sie lediglich 20 Bytes für jedes Element zuzuweisen:

DIM MyArray$(100) LENGTH 20

Das resultierende Array verwendet nur 2 KB RAM.

Strings manipulieren

Die Handhabung von Zeichenfolgen ist eine der Stärken von MMBasic. Mit ein paar einfachen Funktionen können Sie sie trennen manipulieren. Diese Funktionen sind:

LEFT$(string$, nbr)	Gibt einen Teilstring von string$ mit nbr Zeichen links vom Anfang der Zeichenfolge zurück.
RIGHT$(string$, nbr)	wie oben aber mit *nbr* Zeichen rechts vom Ende des String.
MID$(string$, pos, nbr)	Gibt einen Teilstring von string$ mit nbr von Zeichen zurück, beginnend mit dem Zeichen pos in der Zeichenfolge (dh in der Mitte der Zeichenfolge).

Beispiel wenn `S$ = "Das ist ein String"`

dann:
```
R$ = LEFT$(S$, 7)    bedeutet R$ = "Das ist"
R$ = RIGHT$(S$, 10)  bedeutet R$= "ein String"
R$ = MID$(S$, 5, 3)  bedeutet R$= "ist"
```

Beachten Sie, dass in MID$() die erste Zeichenposition in einer Zeichenfolge die Nummer 1 ist, die zweite die Nummer 2 und so weiter. Wenn man also das erste Zeichen als eins zählt ist die sechste Position der Beginn des Wortes „ist“. Eine weitere nützliche Funktion ist:

`INSTR(string$, pattern$ )` Gibt eine Zahl zurück, die die Position darstellt, an der pattern$ vorkommt

Dies kann zur Suche einer Zeichenfolge verwendet werden. Die zurückgegebene Zahl ist die Position des Teilstrings innerhalb des Hauptstrings. Wie bei MID$() beginnt der String an Position 1.
Zum Beispiel, wenn S$ = "Dies ist eine Zeichenfolge"

```
pos = INSTR(S$, " ")
```

würde dazu führen dass pos auf die Position des ersten Leerzeichens in S$ gesetzt wird (dh 5).
INSTR() kann mit anderen Funktionen kombiniert werden, sodass dies das erste Wort in S$ zurückgeben würde:

```
R$ = LEFT$(S$, INSTR(S$, " ") - 1)
```

Es gibt auch eine erweiterte Version von INSTR():

INSTR(pos, string$, pattern$)	Gibt eine Zahl zurück, die die Position darstellt an der sich pattern$ befindet

So können wir das zweite Wort in S$ folgendermaßen finden:

```
pos = INSTR(S$, " ")
R$ = LEFT$(S$, INSTR(pos + 1, S$, " ") - 1)
```

Dieses letzte Beispiel ist ziemlich kompliziert daher könnte es sich lohnen es im Detail durchzuarbeiten, damit Sie nachvollziehen können wie es funktioniert. Beachten Sie, dass INSTR() die Zahl Null zurückgibt wenn die Teilzeichenfolge nicht gefunden wird und dass eine beliebige Zeichenfolge -Funktion wird einen Fehler ausgeben und das Programm anhalten, wenn dies als Zeichenposition verwendet wird. Also in einem In einem praktischen Programm würden Sie zuerst prüfen, ob Null von INSTR() zurückgegeben wird, bevor Sie diesen Wert verwenden.

Beispielsweise:

```
pos = INSTR(S$, " ")
if pos > 0 THEN R$ = LEFT$(S$, INSTR(pos + 1, S$, " ") - 1)
```

Wissenschaftliche Schreibweise

Bevor wir die Diskussion der Datentypen beenden, müssen wir das Thema Fließkommazahlen und wissenschaftliche Schreibweise abdecken. Die meisten Zahlen können normal geschrieben werden, zum Beispiel 11 oder 24,5 aber sehr große oder kleine Zahlen wird es schwieriger. Zum Beispiel wurde geschätzt dass die Anzahl der Sandkörner auf dem Planeten Erde bei 750000000000000000 liegt . Das Problem bei dieser Zahl ist dass man leicht den Überblick über die Anzahl der Nullen verliert folglich ist es schwierig diese mit einer ähnlich großen Zahl zu vergleichen. Wissenschaftler würden diese Zahl als 7,5 x 10^18 schreiben was als wissenschaftliche Notation bezeichnet wird und viel einfacher nachvollziehen ist. MMBasic schaltet automatisch auf die wissenschaftliche Notation um wenn es um sehr große oder kleine Gleitkommazahlen geht. Wenn beispielsweise die obige Zahl in einer Gleitkommavariablen gespeichert wurde, wird der PRINT-Befehl es als 7,5E+18 anzeigen, das ist BASIC-Art 7,5 x 10^18 darzustellen. Anderes Beispiel: Die Zahl 0,0000000456 als 4,56E-8 angezeigt, was 4,56 x 10-8 entspricht. Sie können auch die wissenschaftliche Notation auch für Konstanten verwenden. Beispielsweise: Sandkörner = 7,5E+18

MMBasic verwendet die wissenschaftliche Schreibweise nur zur Darstellung von Fließkommazahlen keine Ganzzahlen. Für Wenn Sie beispielsweise die Anzahl der Sandkörner einer Integer-Variablen zuweisen würde sie als normale Zahl mit vielen Nullen ausgegeben.

DIM-Befehl

Wir haben den DIM-Befehl zuvor zum Definieren von Arrays verwendet, aber er kann auch zum Erstellen von gewöhnlichen Variablen verwendet werden. Beispielsweise können Sie gleichzeitig vier String-Variablen wie folgt erstellen:

```
DIM STRING Car, Name, Street, City
```

Beachten Sie, dass diese Variablen mit dem DIM-Befehl als Zeichenfolgen definiert wurden deshalb wird das $-Suffix nicht benötigt. Die Definition allein reicht MMBasic aus um ihren Typ zu identifizieren. Ebenso wann Wenn Sie diese Variablen in einem Ausdruck verwenden, benötigen Sie das Typsuffix nicht: Zum Beispiel:

Stadt = "Sydney"

Sie können auch das Schlüsselwort INTEGER verwenden um eine Reihe von Integer-Variablen und FLOAT zu definieren das gleiche gilt für Fließkommavariablen. Diese Art der Notation kann in ähnlicher Weise verwendet werden, um Arrays zu definieren.

Beispielsweise:

```
DIM INTEGER seconds(200)
```

Eine andere Methode zur Definition des Variablentyps ist die Verwendung des Schlüsselworts AS. Beispielsweise:

```
DIM Car AS STRING, Name AS STRING, Street AS STRING
```

Dies ist die von Microsoft verwendete Methode (MMBasic versucht, die Microsoft-Kompatibilität aufrechtzuerhalten) und das ist sie auch nützlich wenn die Variablen unterschiedliche Typen haben. Beispielsweise:

```
DIM Car AS STRING, Age AS INTEGER, Value AS FLOAT
```

Sie können jede dieser Methoden zum Definieren des Typs einer Variablen verwenden sie verhalten sich alle gleich. Die Definition von Variablen mit dem DIM-Befehl hat den Vorteil dass sie eindeutig definiert sind (vorzugsweise beim Start des Programms) und deren Typ (Float, Integer oder String)

zur keiner Fehlinterpretation führt. Sie können das noch verstärken indem Sie die folgenden Befehle ganz oben in Ihrem Programm verwenden:

```
OPTION EXPLICIT
OPTION DEFAULT NONE
```

Die erste gibt MMBasic an, dass alle Variablen explizit mit DIM definiert werden müssen, bevor sie dies können verwendet werden. Die zweite gibt an, dass der Typ aller Variablen angegeben werden muss, wenn sie erstellt werden. Warum sind diese beiden Befehle wichtig? Die erste kann helfen, einen häufigen Programmierfehler zu vermeiden, bei dem Sie versehentlich einen Variablennamen falsch schreiben. Beispielsweise könnte Ihr Programm die aktuelle Temperatur in einer Variablen gespeichert haben namens Temp aber irgendwann haben Sie es versehentlich als Tmp falsch geschrieben. Dies veranlasst MMBasic dazu automatisch eine Variable namens Tmp erstellen und ihren Wert auf Null setzen. dies ist offensichtlich nicht das was Sie wollen. Es wird einen versteckten Fehler ins Programm einführen der schwer zu finden sein könnte – auch wenn dir bewusst war, dass etwas nicht stimmt. Auf der anderen Seite wenn Sie die OPTION EXPLICIT Befehl beim Start Ihres Programms MMBasic würde sich weigern die Variable automatisch zu erstellen und stattdessen einen Fehler anzeigen. Der Befehl OPTION DEFAULT NONE hilft weiter, weil er MMBasic mitteilt, dass der Programmierer den Typ jeder Variablen bei der Deklaration ausdrücklich angeben muss. Es ist einfach zu vergessen den Typ anzugeben und MMBasic zu erlauben den Typ automatisch anzunehmen was unerwartete Folgen haben kann. Für kleine “quick and dirty” Programme ist es in Ordnung MMBasic zu erlauben Variablen automatisch zu erstellen, aber in größeren Programmen sollten Sie dieses immer OPTION EXPLICIT und OPTION STANDARD NONE nutzen. Wenn eine Variable erstellt wird wird sie für Gleitkomma- und Ganzzahlen auf Null gesetzt und ein leerer String (d. h. onhe Zeichen) für eine String-Variable. Sie können seinen Anfangswert auf einen anderen Wert setzen, wenn er mit DIM erstellt wird . Beispielsweise:

```
DIM FLOAT nbr = 12.56
DIM STRING Car = "Ford", City = "Perth"
```

Sie können Arrays auch initialisieren, indem Sie die Initialisierungswerte wie folgt in Klammern setzen: `DIM s$(2) = ("zero", "one", "two")`

Achtung: Arrays beginnen bei Null zu zählen, Sie haben also drei Elemente mit dem Index 0, 1 und 2.

Konstanten

Eine übliche Anforderung bei der Programmierung besteht darin einen passenden Bezeichner zu definieren der einen Wert ohne die Gefahr einer versehentlichen Änderung darstellt was passieren kann, wenn Variablen für diesen Zweck verwendet wurden. Diese werden als Konstanten bezeichnet und können E/A-Pin-Nummern, Signalgrenzen und mathematische Werte darstellen, Konstanten und so weiter. Sie können eine Konstante mit dem Befehl CONST erstellen. Dies definiert einen Bezeichner, der auf einen Wert gesetzt der nicht geändert werden kann. Wenn Sie beispielsweise die Spannung einer an Pin 31 angeschlossenen Batterie überprüfen möchten können Sie die Werte so definieren:

```
CONST BatteryVoltagePin = 31
CONST BatteryMinimum = 1.5
```

Diese Konstanten können im Programm verwendet werden wo sie sinnvoller sind als einfache Nummern.

Beispiel:

```
SETPIN BatteryVoltagePin, AIN
IF PIN(BatteryVoltagePin) < BatteryMinimum THEN SoundAlarm
```

Es ist eine gute Programmierpraxis Konstanten für jede feste Zahl zu verwenden die eine wichtige Bedeutung darstellt Wert. Normalerweise werden sie zu Beginn eines Programms definiert wo sie leicht zu sehen und bequem sind befindet sich für einen anderen Programmierer zur Anpassung (falls erforderlich).

Unterprogramme

Ein Unterprogramm ist ein Block Programmiercode die in sich abgeschlossen ist wie ein Modul und und innerhalb des Programms von überall aufgerufen werden kann. Für das Programm sieht es aus wie ein neu definierter Befehl. Beispielsweise soll ein Fehler entdeckt und gemeldet werden. Als Unterprogramm könnte das so aussehen:

```
SUB ErrMsg
  PRINT "Fehler entdeckt"
END SUB
```

Mit dieser in Ihr Programm eingebetteten Subroutine brauchen Sie nur noch den Befehl **ErrMsg** zu verwenden wenn immer Sie die Nachricht anzeigen möchten. Beispielsweise:

```
IF A < B THEN ErrMsg
```

Die Definition einer Subroutine kann überall im Programm stehen, normalerweise steht sie jedoch am Ende. Wenn MMBasic in die Definition läuft während es Ihr Programm ausführt wird es sie einfach überspringen. Das obige Beispiel ist zwar gut aber es wäre besser, wenn mehr Informationen angezeigt werden könnten, die bei jedem Aufruf der Subroutine angepasst werden kann. Dies kann durch Übergeben eines Strings als Argument- manchmal auch als Parameter bezeichnet- an die Subroutine. In diesem Fall würde die Definition des Unterprogramms wie folgt aussehen:

```
SUB ErrMsg  Msg$
  PRINT "Fehler: " + Msg$
END SUB
```

Wenn Sie dann die Subroutine aufrufen, können Sie die auszugebende Zeichenfolge in der Befehlszeile der Unterprogramms angeben. Beispielsweise:

```
IF A < B THEN ErrMsg "Anzahl zu klein”
```

Wenn das Unterprogramm so aufgerufen wird, kommt die Meldung "Fehler: Anzahl zu klein" auf der Konsole ausgegeben. Innerhalb des Unterprogramms hat Msg$ den Wert "Anzahl zu klein", wenn sie so aufgerufen wird und mit PRINT verkettet wird um so die vollständige Fehlermeldung zu erstellen. Eine Unterroutine kann eine beliebige Anzahl von Argumenten haben die jeweils Gleitkomma, Ganzzahl oder String sein können wobei die Argumente durch Komma getrennt sind. Innerhalb der Subroutine verhalten sich die Argumente wie gewöhnliche Variablen aber sie existieren nur innerhalb der Subroutine und verschwinden, wenn das Unterprogramm endet. Es können gleichnamige Variablen im Hauptprogramm und Unterprogramm verwenden, diese werden vor dem UP versteckt und sich von den Argumenten dort unterscheiden. Der Typ des zu liefernden Arguments kann mit einem Typensuffix angegeben werden (dh $, % oder ! für String, Ganzzahl und Gleitkomma). Beispielsweise muss im Folgenden das erste Argument ein String und das zweite eine Ganzzahl sein:

```
SUB MySub Msg$, Nbr%
  …
END SUB
```

MMBasic konvertiert die gelieferten Werte wenn es kann, also wenn das Programm einen Gleitkommawert geliefert hat als zweites Argument wird MMBasic es in eine Ganzzahl konvertieren. Wenn MMBasic den Wert nicht konvertieren kann Es zeigt einen Fehler an und kehrt zur Eingabeaufforderung zurück. Wenn man eine Zeichenfolge als zweites Argument angegeben hat wird das Programm mit einem Fehler stoppen. Man muss die Typ-Suffixe nicht verwenden und kann stattdessen den Typ der Argumente mithilfe von AS definieren ähnlich wie im DIM-Befehl. Folgendes ist beispielsweise identisch mit dem obigen Beispiel:

```
SUB MySub Msg AS STRING, Nbr AS INTEGER
  …
END SUB
```

Natürlich, wenn Sie im gesamten Programm nur einen Variablentyp verwendet und OPTION DEFAULT verwendet haben Um diesen Typ festzulegen, könnten Sie die Frage nach Variablentypen

vollständig ignorieren. Wenn ein Unterprogramm mit einem Argument aufgerufen wird, das eine Variable ist (d. h. keine Konstante oder kein Ausdruck) erstellt MMBasic eine entsprechende Variable innerhalb de s Unterprogramms die auf diese Variable zurückzeigt. Alle Änderungen an der Variablen die das Argument innerhalb des Unterprogramms darstellt ändern auch die Variable die im Aufruf verwendet wird. Dies wird als Übergabe von Argumenten per Referenz bezeichnet. Das lässt sich am besten an einem Beispiel erklären:

```
DIM MyNumber = 5       ' setze die Variable auf 5
CalcSquare MyNumber    ' UP quadriert den Wert
PRINT MyNumber         ' gibt 25 aus
END

SUB CalcSquare nbr
  nbr = nbr * nbr      ' quadriere das Argument und übergebe es
                         zurück
END SUB
```

Die Subroutine CalcSquare nimmt ihr Argument quadriert es und schreibt es zurück in die Variable die das Argument (nbr) darstellt. Da das Unterprogramm mit einer Variablen (MyNumber) aufgerufen wurde zeigt die Die Variable nbr zurück auf MyNumber und jede Änderung an nbr ändert auch MyNumber entsprechend. Als Ergebnis gibt die PRINT-Anweisung 25 aus. Das Übergeben von Argumenten als Referenz ist praktisch da es einer Unterroutine ermöglicht Werte an den Code zurückzugeben der es aufgerufen hat. Es könnte jedoch zu Problemen führen wenn eine Subroutine die Variable verwendet die ein Argument als Allzweckvariable verwendet und ändert seinen Wert. Wenn es dann mit einer Variablen als Argument aufgerufen würde würde diese Variable versehentlich geändert werden. **Aus diesem Grund sollten Sie vermeiden Variablen zu manipulieren, die Argumente innerhalb einer Subroutine darstellen, weisen Sie den Wert stattdessen einer lokalen Variable zu siehe unten und manipulieren Sie diese**. Wenn Sie eine Unterroutine aufrufen können Sie einige (oder alle) Parameter weglassen und sie nehmen den Wert Null an (für Gleitkommazahlen oder ganze Zahlen) oder eine leere Zeichenfolge. Dies ist praktisch da das Unterprogramm feststellen kann ob ein Parameter fehlt und handelt entsprechend. Hier ist zum Beispiel unsere Subroutine zum Generieren einer Fehlermeldung aber diese Version kann auch ohne Angabe einer Fehlermeldung als Parameter verwendet werden:

```
SUB ErrMsg  Msg$
  IF Msg$ = "" THEN
    PRINT "Fehler erkann"
  ELSE
    PRINT "Fehler: " + Msg$
  ENDIF
END SUB
```

Innerhalb einer Subroutine kann man die meisten Funktionen von MMBasic verwenden einschließlich des Aufrufs anderer Subroutinen, IF…THEN-Befehle, FOR…NEXT-Schleifen und so weiter. **Eine Sache die nicht funktioniert ist mit GOTO aus einem Unterprogramm springen (das Ergebnis wäre undefiniert)**. Normalerweise wird die Die Subroutine beendet wenn der END SUB-Befehl erreicht wird, aber man kann das Unterprogramm vorzeitig mit dem EXIT SUB-Befehl beenden.

Funktionen

Funktionen ähneln Subroutinen, mit dem Hauptunterschied dass eine Funktion verwendet wird um einen Wert in einem Ausdruck zurückzugeben. Zum Beispiel wenn man eine Funktion zum Umwandeln einer Temperatur von Grad Celsius zu Grad Fahrenheit erstellen möchte kann man definieren:

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

Anschliessend kann man es in einem Ausdruck verwenden:

```
Input "Geben Sie eine Temperatur in Celsius ein: ", t
PRINT "Das ist dasselbe wie " Fahrenheit(t) "F"
```

Oder als weiteres Beispiel:

```
IF Fahrenheit(temp) <= 32 THEN PRINT "Einfrieren"
```

Man könnte auch umgekehrt definieren:

```
FUNCTION Celsius(F)
  Celsius = (F - 32) * 0.5556
END FUNCTION
```

Wie Sie sehen können wird der Funktionsname als gewöhnliche lokale Variable innerhalb des Unterprogramms verwendet. Nur wenn die Funktion zurückkehrt wird der Wert dem Ausdruck zur Verfügung gestellt der ihn aufgerufen hat. Die Regeln für die Argumentliste in einer Funktion sind ähnlich wie bei Unterprogrammen. Der einzige Unterschied ist dass immer Klammern um die Argumentliste herum erforderlich sind wenn Sie eine Funktion aufrufen, selbst wenn es keine Argumente gibt (Klammern sind beim Aufruf einer Subroutine optional). Um einen Wert von der Funktion zurückzuerhalten weisen Sie dem Namen der Funktion innerhalb der Funktion einen Wert zu. Wenn Der Name der Funktion wird mit einem Typensuffix abgeschlossen (dh $, ein % oder ein !) wird die Funktion diesen Typ zurückgeben (String, Ganzzahl oder Gleitkomma), andernfalls wird zurückgegeben, was auch immer für OPTION DEFAULT eingestellt ist. Für Beispielsweise gibt die folgende Funktion einen String zurück:

```
FUNCTION LVal$(nbr)
  IF nbr = 0 THEN LVal$ = "Falsch" ELSE LVal$ = "Wahr"
END FUNCTION
```

Sie können den Typ der Funktion explizit angeben indem Sie das Schlüsselwort AS verwenden und dann brauchen Sie es nicht um ein Typsuffix zu verwenden (ähnlich wie beim Definieren einer Variablen mit DIM). Dies ist das obige Beispiel, das umgeschrieben wurde, um diese Funktion zu nutzen:

```
FUNCTION LVal(nbr) AS STRING
  IF nbr = 0 THEN LVal = "Falsch" ELSE LVal = "Wahr"
END FUNCTION
```

In diesem Fall ist der von der Funktion LVal zurückgegebene Typ ein String. Was Unterroutinen betrifft, können Sie die meisten Features von MMBasic innerhalb von Funktionen verwenden. Das beinhaltet FOR…NEXT-Schleifen, Aufruf anderer Funktionen und Subroutinen usw. Außerdem kehrt die Funktion zu zurück Ausdruck, der ihn aufgerufen hat, als der Befehl END FUNCTION erreicht wurde, aber Sie können auch vorzeitig zurückkehren mit dem Befehl EXIT FUNCTION.

Lokale Variablen

Variablen die mit DIM erstellt oder nur automatisch erstellt werden, werden als globale Variablen bezeichnet. Das bedeut dass sie überall im Programm angezeigt und verwendet werden können auch innerhalb von Unterprogrammen und Funktionen. Innerhalb eines Unterprogramms oder Funktion müssen Sie jedoch häufig Variablen für verschiedene Aufgaben Zwecke verwenden die intern für das Unterprogramm/die Funktion sind. In portablem Code willst du keinen Namen wählen für eine solche Variable der mit einer globalen Variablen gleichen Namens kollidiert. Um das zu beenden kann man mit dem LOCAL-Befehl eine Variable innerhalb der Subroutine/ Funktion definieren. Die Syntax für LOCAL ist identisch mit dem DIM-Befehl, das heißt, die Variable kann ein Array sein, Sie können den Typ der Variablen festlegen und sie auf einen Wert initialisieren. Dies ist zum Beispiel unsere Subroutine ErrMsg, aber dieses Mal wurde sie erweitert um eine lokale Variable zu verwenden zum Verbinden der Fehler-Strings.

```
SUB ErrMsg  Msg$
  LOCAL STRING tstr
```

```
    tstr = "Fehler: " + Msg$
    PRINT tstr
  END SUB
```

Die Variable tstr wird innerhalb der Subroutine als LOCAL deklariert was bedeutet dass (wie das Argument list) sie nur innerhalb der Subroutine existiert und verschwindet wenn die Subroutine beendet wird. Sie können eine globale Variable namens tstr in Ihrem Hauptprogramm haben und sie unterscheidet sich von der Variablen tstr im Unterprogramm (in diesem Fall wird das globale tstr innerhalb des Unterprogramms versteckt). Sie sollten immer lokale Variablen für Operationen innerhalb Ihrer Subroutine oder Funktion verwenden, weil sie helfen das Modul unabhängiger und anderweitig nutzbar zu machen.

Statische Variablen

LOCAL-Variablen werden jedes Mal auf ihre Anfangswerte zurückgesetzt (normalerweise Null oder ein Leerstring) jedes Mal wenn das Unterprogramme oder die Funktion startet, es gibt jedoch Zeiten, in denen Sie möchten, dass die Variable ihren Wert zwischen Aufrufen behält. Dieser Variablentyp wird mit dem STATIC-Befehl definiert. Wir können demonstrieren, wie nützlich STATIC-Variablen sind indem wir das Unterprogramm ErrMsg erweitern um zu verhindern dass doppelte Aufrufe der Unterroutine wiederholt dieselbe Nachricht anzeigen. Zum Beispiel kann unser Programm diese Unterroutine von mehreren Stellen aus aufrufen aber wenn die Nachricht an mehreren Stellen gleich ist möchten wir die Nachricht nur einmal sehen. Dies ist unsere neue Subroutine:

```
SUB ErrMsg  Msg$
  STATIC STRING lastmsg
  LOCAL STRING tstr
  IF Msg$ <> lastmsg THEN
    tstr = "Fehler: " + Msg$
    PRINT tstr
    lastmsg = Msg$
  ENDIF
END SUB
```

Um die zuletzt angezeigte Nachricht zu verfolgen verwenden wir eine statische Variable namens lastmsg. Sie wird den Text der letzten Nachricht behalten und wir können ihn mit dem aktuellen Nachrichtentext vergleichen um festzustellen ob er unterschiedlich und deshalb gedruckt werden sollte. Jedes Mal wenn ein Aufruf getätigt wird nur eine Einzelnachricht angezeigt bei einem doppelten Nachrichtentext. Der Befehl STATIC verwendet genau die gleiche Syntax wie DIM. Das bedeutet dass man verschiedene Arten von statischen Variablen definieren kann einschließlich Arrays, und Sie können sie auch mit einem bestimmten Wert initialisieren. Die statische Variable wird erstellt, wenn das Programm zum ersten Mal auf der STATIC-Befehl trifft und wird automatisch auf Null (bei Gleitkomma oder Ganzzahl) oder eine leere Zeichenfolge gesetzt. Bei späteren Aufrufen an das Unterprogramm oder Funktion erkennt MMBasic dass die Variable bereits erstellt wurde und wird seinen Wert unverändert lassen (dh was auch immer es im vorherigen Anruf war). Wie bei DIM können Sie auch eine statische Variable auf einen Wert initialisieren. Beispielsweise:

```
STATIC INTEGER var = 123
```

Beim ersten Aufruf (wenn die Variable erstellt wird) wird sie auf 123 initialisiert bei den nachfolgenden Aufrufen behält sie jedoch den zuvor eingestellten Wert. Meistens werden statische Variablen verwendet um den Zustand von etwas zu verfolgen während man sich in einem Unterprogramm befindet oder einer Funktion. Ein Zustand ist eine Aufzeichnung von etwas das zuvor passiert ist. Beispiele beinhalten:

• Wurde der COM-Port bereits geöffnet?

• Welche Schritte in einer Sequenz haben wir abgeschlossen?

• Welcher Text wurde bereits angezeigt?

Normalerweise verwenden Sie globale Variablen (die mit DIM erstellt wurden) um einen Status zu verfolgen aber manchmal möchte man das in einem Modul behalten und hier sind statische Variablen wertvoll. Genau wie LOKAL hilft die Verwendung von STATIC dabei Ihre Subroutinen und Funktionen unabhängiger und frei nutzbar zu machen..

Tage berechnen

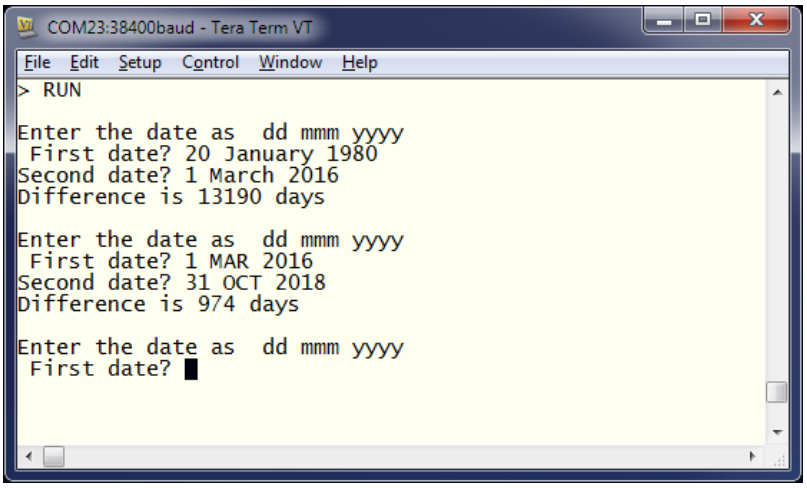
Wir haben bisher in diesem Tutorial und viele Programmierbefehle und -techniken behandelt Abschließend wäre es sinnvoll ein Beispiel dafür zu geben, wie sie zusammenarbeiten. Das Folgende ist ein Beispiel das viele Funktionen der BASIC-Sprache verwendet um die Anzahl der Tage zwischen zwei Daten zu berechnen:

```
' Beispielprogramm zur Berechnung der Anzahl der Tage zwischen zwei Daten

OPTION EXPLICIT
OPTION DEFAULT NONE

DIM STRING s
DIM FLOAT d1, d2

DO
  ' Hauptprogrammschleife
  PRINT : PRINT " Geben Sie das Datum als tt mmm jjjj ein "
  PRINT " Erstes Datum";
  INPUT s
  d1 = GetDays(s)
  IF d1 = 0 THEN PRINT "Ungültiges Datum!" : CONTINUE DO
  PRINT "Zweites Datum";
  INPUT s
  d2 = GetDays(s)
  IF d2 = 0 THEN PRINT "Ungültiges Datum!" : CONTINUE DO
  PRINT "Differenz =" ABS(d2 - d1) " Tage"
LOOP

' Berechnen Sie die Anzahl der Tage seit dem 1.1.1900
FUNCTION GetDays(d$) AS FLOAT
  LOCAL STRING Month(11) =
("jan","feb","mär","apr","mai","jun","jul","aug","sep","okt","nov","dez")
  LOCAL FLOAT Days(11) = (0,31,59,90,120,151,181,212,243,273,304,334)
  LOCAL FLOAT day, mth, yr, s1, s2

  ' Finden Sie das trennende Leerzeichen innerhalb eines Datums
  s1 = INSTR(d$, " ")
  IF s1 = 0 THEN EXIT FUNCTION
  s2 = INSTR(s1 + 1, d$, " ")
  IF s2 = 0 THEN EXIT FUNCTION

  ' Tag, Monat und Jahr als Zahlen erhalten
  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

  ' Berechnen Sie die Anzahl der Tage einschließlich der Anpassung für Schaltjahre
  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
```

Beachten Sie, dass die Zeile, die mit LOCAL STRING Month(11) beginnt, wegen begrenzter Seitenbreite umgebrochen wurde– es ist eine Zeile wie folgt:

LOCAL STRING Monat(11) = ("jan","feb","mar","apr","may","jun","jul","aug","sep","oct","nov","dez")

Dieses Programm funktioniert indem es zwei Daten vom Benutzer an der Konsole erhält und sie dann in ganze Zahlen umwandelt die die Anzahl der Tage seit 1900 darstellen. Mit diesen beiden Zahlen gibt eine einfache Subtraktion die Anzahl der Tage zwischen ihnen an. Wenn dieses Programm ausgeführt wird werden Sie aufgefordert die beiden Daten einzugeben und Sie müssen diese Form verwenden: **tt mmm jjjj**..



Diese BS-Hardcopy zeigt wie die Programmdarstellung aussieht. Das Hauptmerkmal des Programms ist die benutzerdefinierte Funktion GetDays() die einen String verarbeitet der in Tag, Monat und Jahr geteilt wird dann wird die Anzahl der Tage seit dem 1. Januar 1900 berechnet. Diese Funktion wird zweimal aufgerufen, einmal für das erste Datum und dann nochmal für das zweites. Es wird lediglich ein Datum (in Tagen) von die andere subtrahiert um die Differenz in Tagen zu erhalten. Wir werden nicht ins Detail gehen, wie die Berechnungen durchgeführt werden (dh Umgang mit Schaltjahren), das kann als Übung dem Leser überlassen werden. Es ist jedoch angebracht auf einige Merkmale von MMBasic hinzuweisen die vom Programm verwendet werden. Es zeigt, wie lokale Variablen verwendet und initialisiert werden können. In der Funktion GetDays() werden gleichzeitig zwei Arrays deklariert und initialisiert. Dies sind nur eine bequeme Methode zum Nachschlagen der Namen der Monate und der zunehmenden Anzahl von Tagen für jeden Monat. Später in der Funktion (der FOR-Schleife) können Sie sehen, wie sie mit zwölf verschiedenen Monaten recht effizient umgehen. Ein weiteres Merkmal dieses Programms sind die String-Handling-Features von MMBasic. Der Die INSTR()-Funktion wird verwendet um die beiden Leerzeichen in der Datumszeichenfolge zu finden, später verwendet MID$() diese um die Tages-, Monats- und Jahreskomponenten des Datums zu extrahieren. Die VAL()-Funktion wird verwendet um eine Ziffernfolge (wie das Jahr) in eine Zahl umzuwandeln die in einer numerischen Variablen gespeichert werden kann. Beachten Sie dass der Wert einer Funktion jedes Mal auf Null initialisiert wird wenn die Funktion ausgeführt wird und wenn sie nicht auf einem bestimmten Wert gesetzt wird wird sie einen Nullwert zurückgeben. Dies erleichtert die Fehlerbehandlung da wir die Funktion einfach beenden können wenn ein Fehler entdeckt wird. Es liegt dann in der Verantwortung des aufrufenden Programmcodes auf einen Rückgabewert von Null zu prüfen was einen Fehler bedeutet. Dieses Programm veranschaulicht einen der Vorteile der Verwendung von Unterprogrammen und Funktionen nämlich wenn sie geschrieben und vollständig getestet können sie als vertrauenswürdige "Black Box" behandelt werden die nicht mehr geöffnet werden muss. Aus diesem Grund sollten solche Funktionen gründlich getestet und dann möglichst unangetastet bleiben. Es gibt einige Funktionen dieses Programms die wir zuvor noch nicht behandelt haben. Die erste ist die MOD Operator der den Divisionsrest einer Zahl durch eine andere berechnet. Teilt man 15 durch 4 so erhält man einen Divisionsrest von 3 was genau dem Rückgabewert des Ausdrucks 15 MOD 4 entspricht. Die Funktion ABS() ist ebenfalls neu und gibt ihr Argument als positive Zahl zurück z. B. ABS(-15) gibt +15 zurück ebenso wie ABS(15). Der EXIT FOR-Befehl beendet eine FOR-Schleife obwohl das Ende seiner Schleife noch nicht erreicht ist. EXIT FUNCTION beendet eine Funktion sofort obwohl die Ausführung noch nicht das Ende der Funktion erreicht hat und CONTINUE DO bewirkt dass das Programm sofort zum Ende einer DO-Schleife springt und sie erneut ausführt.

Warum sollte dieses Programm nützlich sein? Nun, manche Leute zählen ihr Alter gerne in Tagen, sozusagen alle Tage ist ein Geburtstag! Sie können Ihr Alter in Tagen berechnen geben Sie einfach Ihr

Geburtsdatum ein und heutiges Datum. Das ist nicht besonders nützlich, aber das Programm selbst ist wertvoll, da es die Besonderheiten der Programmierung in MMBasic zeigt. Arbeiten Sie sich also durch das Programm und gehen Sie jeden Abschnitt durch, bis Sie ihn verstanden haben – es sollte sich lohnen.